D3.3 Mini-IGCMs

Nabeeh Jumah Julian Kunkel

Workpackage: WP3 Evaluation
Responsible institution: Yashiro
Contributing institutions: RIKEN, IPSL
Date of submission: February 2019

Contents

Abstract

This report provides a description of the mini-IGCM code that was developed under the work-package WP3 of the project AIMES. The task aims to develop mini applications that provide a basis to do experiments with codes that use icosahedral grids and to allow the user to run kernels that use icosahedral grids through ready-to-run simple applications. A package is developed for this purpose and published online through Github. Instead of facing the complexities relating to running the real models, the described package provides a set of kernels with different icosahedral grid structures. Along with the kernels, the necessary data structures and the code to test the kernels are provided within the same package. All the user needs to do is to use the provided makefiles or run scripts.

The package provides kernels under three groups based on three real icosahedral models; ICON, DYNAMICO, and NICAM. This allows to work with different grid structures including semi-structured and unstructured grids, triangular and hexagonal grids.

1 Introduction

In this report we describe the mini-IGCMs code that was developed as part of the work-package WP3 of the project. The code is built based on codes from the three icosahedral models ICON, DYNAMICO, and NICAM. Three mini-IGCMs are developed, one per icosahedral model. Each mini-IGCM consists mainly of a set of kernels, each of which was written based on the code of an original kernel from the corresponding model.
Besides to the kernels, the testbed code also includes the necessary grid structure and other configuration structures which the kernels need. Also the necessary code that allows to run and test those kernels is included. Besides to the computations that the kernels make, the testbed code includes codes that handle the I/O operations. Those I/O kernels allow the testbed code to initialize the different fields and to write the values of the needed fields at specific time steps.
Descriptions on how to configure and run the testbed code are delivered along with the package. The necessary makefiles or run scripts are provided along with the code. This simplifies running tests and allows non-expert people who want to run the testbed code to simply run the provided configurations with simple commands and a few steps.
The code of the mini-IGCMs was developed by the partners from University of Hamburg and RIKEN institute, supported also from other partners in the AIMES project. It is available online on the projects page on Github: `https://github.com/aimes-project/IcoAtmosBenchmark_v1`

Relation to the Project

This report describes the final version of the packaged testbed suite including the tools to run the DSL versions of the testbed codes.
The following text is the description of the project proposal for this task and deliverable:
In this task, we develop the mini-apps by stripping non-essential components from full codes. Compared to the full model their code base is reduced, as we restrict the models scope and, thus, the range of experiments that can be conducted with the mini-IGCM. We build one common package containing all different model flavors. The package of mini-app contains settings for one or two scientific experiments, required input data, and documentation. We setup and document the package in such a way that users not involved in these models, such as vendors, can compile and execute the provided scientific experiments easily.

2 ICON-based Mini IGCM

This mini application was developed based on the concepts from the ICON model. The grid used in the kernels is an unstructured grid. Triangular tessellation is used to discretize the surface. The dimensions of the surface are represented by a member variable within a grid data structure. Those dimensions can be controlled by providing values through the command line.
More details about the structure of the code are discussed in the coming section.

2.1 Structure

The code is run in time steps, which is controlled by the user through an argument to the command line. The main loop advances the simulation one time step per iteration, during which the components of the model are called to execute their parts of the simulation.
The modeling code is built within a set of components. A data structure is used to encapsulate the description and data of the component. This includes the functions that initialize the component, do the computations

that the component contributes to the simulation, handle the necessary I/O operations, and cleanup after the simulation is finished. Other functions allow to read the memory allocated for the variables that the components allocates, and the number of the floating point operations that the component executes in each call to its computation function. Also a validation function is included within the component structure for testing purposes.

Listing 1: Component Data Structure

```
typedef struct {
    int loaded;
    void (*init)(GRID*);
    void (*compute)(GRID*);
    void (*io)(GRID*);
    double (*flops)(GRID*);
    double (*memory)(GRID*);
    uint64_t (*checksum)(GRID*);
    void (*cleanup)(GRID*);
} MODEL_COMPONENT;
```

2.1.1 Component Initialization

When the application starts up, the set of the components that we intend to run during the simulation are initialized by calling the initialization function of each of those components. In this step, each loaded component allocates and loads the initial values of its variables.
The fields are declared within the components through the DSL specifiers.

Listing 2: Field Declaration

```
GVAL CELL 3D gv_temp;
GVAL EDGE 3D gv_grad;
GVAL CELL 3D gv_dvg;
```

The allocation of a field is done within the containing component initialization function. DSL constructs allow to express within the source code that a field is to be allocated, and the source-to-source translator generates the allocation code for the field.
The initialization of the variables is done through NetCDF within the component initialization phase. The DSL allows to use higher-level constructs for the I/O operations. It can also generate data structures for the variables to expose the attributes of the variables to give the programmers more flexibility within the source code.

Listing 3: Component Initialization

```
void com1_init(GRID* g)
{
    com1.loaded = 1;
    ALLOC gv_temp;
    ALLOC gv_grad;
    ALLOC gv_dvg;

    io_read_register(g, "gv_temp", (GVAL *)gv_temp, FLOAT32,
                     FLOAT32, GRID_POS_CELL, GRID_DIM_3D);
    io_write_define(g, "gv_temp", (GVAL *)gv_temp, FLOAT32,
                    GRID_POS_CELL, GRID_DIM_3D, &io_gv_temp);
    io_write_define(g, "gv_grad", (GVAL *)gv_grad, FLOAT32,
                    GRID_POS_EDGE, GRID_DIM_3D, &io_gv_grad);
    io_write_define(g, "gv_dvg",  (GVAL *)gv_dvg,  FLOAT32,
                    GRID_POS_CELL, GRID_DIM_3D, &io_gv_dvg);
}
```

2.1.2 Component Computations

Within the main time step loop, the 'compute' functions of the components that the user wants to run within the simulation are called. In a component's 'compute' function the component calls the set of kernels that it needs to execute.
The kernels are provided in their own files. They are written using the DSL language extensions besides to the general-purpose language C.

Listing 4: Component Compute Function

```
void com1_compute(GRID* g)
{
    grad(g);
    dvg(g);
    step(g);
}
```

2.1.3 I/O Operations

The mini application uses NetCDF to store and read field data. During the component initialization (within application initialization phase), the initialization functions allow reading field initial values and registering fields for output operations. At specific time steps, according to the user's needs, the main loop of the application calls the I/O handler of the components that the user wants to write out their output. When the I/O function of a component is called, the component calls the NetCDF API to write the data of the fields that are registered for output.

Listing 5: Component IO Operations

```
void com1_io(GRID* g){
    io_write_announce(g, &io_gv_grad);
    io_write_announce(g, &io_gv_dvg);
}
```

2.2 Operators

A set of kernels are developed to implement the component set that comprises the application. The kernels are developed with C and the GGDML language extensions.
The set of kernels covers a set of representing modeling cases. Both two- and three-dimensional grids are used. Different kernels that traverse the grid's cells and another set of kernels that traverse the grid's edges are provided. Accessing the neighboring grid components in different cases is also demonstrated. Vertical and horizontal neighborhoods are demonstrated, e.g. above cell, cell's neighboring cells, cells sharing an edge ...

2.2.1 A Vertical Integration Kernel

In the next code snippet in Listing 6 we show a kernel that performs vertical integration. The code is written using the DSL construct. The iterator traverses the cells of the three dimensional grid.
The *gv_vi* field is a two-dimensional field that covers the horizontal grid of the surface. The field *gv_temp* is located at the centers of the cells of the three-dimensional grid. The kernel sums the the values of the *gv_temp* field over the whole column to the value of the output horizontal field *gv_vi*.

Listing 6: Vertical Integration

```
    foreach cell in grid
    {
        gv_vi[cell] += gv_temp[cell];
    }
```

The translation tool knows from the declaration of the variables if they are located at the cells of the whole grid or on the surface. This allows the tool to generate the right code automatically.

2.2.2 A Divergence Kernel

The following code snippet in Listing 7 shows a kernel to compute the divergence of a field. The *foreach* iterator traverses the cells of the three dimensional grid. The *gv_grad* field is located on the edges of the grid. It represents the gradient of some field (*gv_temp*). Its values are computed in another kernel. The *gv_dvg* field is the divergence value and is located at the centers of the grid cells. The values of the divergence field are calculated based on the recorded gradient field values. The language extensions (*cell.edge*(n)) allowed to simplify the access to the edges of the traversed cells.

Listing 7: Divergence

```
    foreach cell in grid
```

```
{
    gv_dvg[cell] = 0;
    for(int n=0;n<NBRS;n++){
        gv_dvg[cell] += g->edge_weights[n][cell.cell %BLKSIZE] *
                        gv_grad[cell.edge(n)];
    }
}
```

2.2.3 A Gradient Kernel

In the following code in Listing 8 we compute the gradient of the *gv_temp* field. The iterator traverses the edges of the three dimensional grid. On each edge, the value of the gradient field *gv_grad* is computed based on the values of the *gv_temp* field at the centers of the cells that share the edge. The language extensions ($edge.cell(n)$) simplify the access and abstract the reference to the cells that share and edge.

Listing 8: Gradient

```
foreach edge in grid
{
        gv_grad[edge] = gv_temp[edge.cell(0)] - gv_temp[edge.cell(1)];
}
```

2.2.4 A Laplacian Kernel

The kernel in Listing 9 computes the Laplacian of the *gv_temp* field. The iterator traverses the cells of the three dimensional grid. At the center of each cell, a new value of the *gv_temp* field is computed based on the values of the *gv_temp* field at the centers of the neighboring cells. The language extensions ($edge.neighbor(n)$) simplify the access and abstract the reference to the neighboring cells.

Listing 9: Laplacian

```
foreach cell in grid
{
    gv_temp_alt[cell] = (gv_temp[cell.neighbor(0)] +
                         gv_temp[cell.neighbor(1)] +
                         gv_temp[cell.neighbor(2)])/3.0;
}
```

2.2.5 A 3D Variable Weighted by a Horizontal Parameter

In the code in Listing 10 the declarations show that the field *gv_temp*, which is used within the computation of the kernel, is defined at the cell centers of the three-dimensional grid. The *gv_outvar* field is defined over the edges of the three-dimensional grid. A parameter variable *gv_ind2Dparam* is defined such that each cell on the surface has one value. The kernel iterates the cells of the three-dimensional grid. It computes the output field *gv_outvar* on the grid edges based on the *gv_temp* values at the cell that share the edge. The kernel uses the parameter values, which are the same for all the cells in one column, as weights for the computation.

Listing 10: Horizontally weighted fields

```
extern GVAL CELL 3D gv_temp;
extern GVAL CELL 2D gv_ind2Dparam;
extern GVAL EDGE 3D gv_outvar;

...

    FOREACH edge IN grid
    {
            gv_outvar[edge] = gv_ind2Dparam[edge.cell(0)] * gv_temp[edge.cell(0)]
                - gv_ind2Dparam[edge.cell(1)] * gv_temp[edge.cell(1)];
    }
```

2.2.6 Computations with Vertical and Horizontal Neighbors and Different Kinds of Parameters

The following code in Listing 11 includes two kernels, one updates the values of the field *gv_o8var* over the whole grid except to vertical levels, and the other updates the values of the same field for the last vertical level only.
The kernels access two fields: *gv_grad* and *gv_o8var*. The field *gv_grad* is located at the edges of the 3D grid, and the field *gv_o8var* is located at the cell centers of the 3D grid.
Two parameter are used within the kernels: *gv_o8param* and *gv_o8par*2. Every cell on the surface has three values for the parameter *gv_o8param*. Those values are used as weights for the edges of the cell. The values are used allover the column cells in the 3D grid. The other parameter (*gv_o8par*2) has one value per cell in over all the 3D grid.
In both kernels we use the DSL 'REDUCE' expression to represent the sum over the edges of a cell. We also used the $cell.edge(n).below()$ and $cell.edge(n).below(m)$ to allow access from a cell to the edges in the vertical levels below the cell.

Listing 11: Computations using 2D and 3D neighbors

```
extern GVAL EDGE 3D gv_grad;
extern GVAL CELL 2D gv_o8param[3];
extern GVAL CELL 3D gv_o8par2;
extern GVAL CELL 3D gv_o8var;

...

    FOREACH cell IN grid|height{1..(g->height-1)}
    {
        GVAL v0 = REDUCE(+,N={0..2},gv_o8param[N][cell] * gv_grad[cell.edge(N)]);
        GVAL v1 = REDUCE(+,N={0..2},gv_o8param[N][cell] * gv_grad[cell.edge(N).below()]);
        gv_o8var[cell] = gv_o8par2[cell] * v0 + (1-gv_o8par2[cell]) * v1;
    }

    FOREACH cell IN grid|height{(g->height-1)..(g->height-1)}
    {
        GVAL v0 = REDUCE(+,N={0..2},gv_o8param[N][cell] * gv_grad[cell.edge(N)]);
        GVAL v1 = REDUCE(+,N={0..2},gv_o8param[N][cell] * gv_grad[cell.edge(N).below()]);
        GVAL v2 = REDUCE(+,N={0..2},gv_o8param[N][cell] * gv_grad[cell.edge(N).below(2)]);
        gv_o8var[cell] = 0.4 * v0 + 0.3 * v1 + 0.3 * v2;
    }
```

3 DYNAMICO-based Mini IGCM

The mini IGCM code is based on the code of the DYNAMICO dynamical core. It uses an icosahedral grid with hexagonal shapes. The diamonds of the icosahedron are divided into patches which allow distribution of the computation over multiple nodes. This division forms a semi-structured grid. The elements are accessed with one index to access the two-dimensional surface of the grid.
The benchmark provides a public documentation that can be accessed through the folder `docs/DYNAMICO` in the root folder of the benchmark. The documentation can be built using the make utility using the Makefile at the `docs` folder.

3.1 Structure

The code executes a time-step loop. The loop handles the halo exchange of the different fields. Other tasks are then executed like calling the time advancing subroutines, e.g. Runge-Kutta, and the computation of the dynamical parts. Physics are also computed if they should be within the current step.

3.1.1 Problem Domain and Size

The problem domain of the DYNAMICO mini-IGCM kernels is defined over the three dimensions with the indices *iim*, *jjm*, and *llm*. The two dimensions of the surface are converted into a new index based on *iim* and *jjm* to access data of fields located on the surface 2D grid (Listing 12).

Listing 12: DYNAMICO mini-IGCM problem size definition

```
  !! below are read from input data file.
  integer, public :: iim = -huge(1)
```

```
  integer, public :: jjm = -huge(1)
  integer, public :: llm = -huge(1)

  integer, public :: ij_begin     = -huge(1)
  integer, public :: ij_end       =  huge(1)
```

The ranges of the coordinates are read from a file to define the problem size when running the kernels.

3.1.2 I/O

The mini-IGCM provides a module for the kernels to execute I/O. A set of public procedures are exposed by this module (Listing 13).

Listing 13: DYNAMICO mini-IGCM I/O module

```
  public :: IO_setup
  public :: IO_LOG_setup
  public :: IO_get_available_fid
  public :: IO_make_idstr
  public :: IO_ARG_getfname
  public :: IO_CNF_open
```

The following I/O functions are provided to handle array storage:

- dumpio_fopen: open file IO stream
- dumpio_fclose: close file IO stream
- dumpio_write_data: write data array
- dumpio_read_data: read data array

3.2 Running the mini-IGCM

The mini-IGCM can be downloaded from the already mentioned benchmark link. The DYNAMICO-based mini-IGCM is located in the sub-folder `kernels/DYNAMICO`. Each of the kernels is provided in its own folder. One sub-folder within each kernel's folder contains the source code of the kernel and the necessary additional code to make the kernel ready to run. Another sub-folder contains files to access test data for testing purposes as input to run the kernel. Also, a sub-folder contains scripts to run the kernel on Linux based machines and on the machine Mistral (in the German Climate Computing Center–DKRZ). A fourth sub-folder is included in each kernel's folder, where log files are provided as reference for the purpose of validation.
To run a kernel, the input data can be first downloaded using the shell script `kernels/DYNAMICO/kernel-name/data/download.sh`, where kernel-name is the name of the kernel to run. The script downloads the necessary data files to run the kernel. To check the integrity of the data files, there exists an md5 file in the same folder. After the data files are downloaded, the kernel needs to be compiled. First, the environment variable IAB_SYS should be exported and assigned the name of one of the files in the `sysdep` folder at the root of the benchmark folder. One of this set of predefined files or an alternative modified version of them can be assigned to the environment variable. Next, the Makefile in the kernel source (`src`) folder can be used to build the code using the make tool.
After compiling a kernel, the shell scripts in `kernels/DYNAMICO/kernel-name/run` can be used to run the kernel (as mentioned there are two scripts to run the kernels on Linux machines in general or on Mistral).
When the kernels are run, the results can be validated using the files in `kernels/DYNAMICO/kernel-name/reference`.

3.3 Operators

In this section we present a set of procedures comprising the DYNAMICO based mini-IGCM code.

3.3.1 compute_pvort

This kernel is based on the subroutine *compute_pvort* from DYNAMICO.

Listing 14: compute_pvort

```
      SUBROUTINE compute_pvort(ps,u,theta_rhodz, rhodz,theta,qu,qv)
         use prec
         use mod_misc
         IMPLICIT NONE
         REAL(rstd),INTENT(IN)  :: u(iim*3*jjm,llm)
         REAL(rstd),INTENT(IN)  :: ps(iim*jjm)
         REAL(rstd),INTENT(IN)  :: theta_rhodz(iim*jjm,llm)
         REAL(rstd),INTENT(INOUT) :: rhodz(iim*jjm,llm)
         REAL(rstd),INTENT(INOUT) :: theta(iim*jjm,llm)
         REAL(rstd),INTENT(INOUT) :: qu(iim*3*jjm,llm)
         REAL(rstd),INTENT(INOUT) :: qv(iim*2*jjm,llm)

         INTEGER :: i,j,ij,l
         REAL(rstd) :: etav,hv, m

         CALL trace_start("compute_pvort")

         IF(caldyn_eta==eta_mass) THEN
            CALL wait_message(req_ps)
         ELSE
            CALL wait_message(req_mass)
         END IF
         CALL wait_message(req_theta_rhodz)

         IF(caldyn_eta==eta_mass) THEN ! Compute mass & theta
            DO l = ll_begin,ll_end
               CALL test_message(req_u)
               !DIR$ SIMD
               DO ij=ij_begin_ext,ij_end_ext
                  m = ( mass_dak(l)+ps(ij)*mass_dbk(l) )/g
                  rhodz(ij,l) = m
                  if ( rhodz(ij,l) > 0.0 ) then
                     theta(ij,l) = theta_rhodz(ij,l)/rhodz(ij,l)
                  else
                     theta(ij,l) = 0.0
                  endif
               ENDDO
            ENDDO
         ELSE ! Compute only theta
            DO l = ll_begin,ll_end
               CALL test_message(req_u)
               !DIR$ SIMD
               DO ij=ij_begin_ext,ij_end_ext
                  theta(ij,l) = theta_rhodz(ij,l)/rhodz(ij,l)
               ENDDO
            ENDDO
         END IF

         CALL wait_message(req_u)

   !!! Compute shallow-water potential vorticity
         DO l = ll_begin,ll_end
            !DIR$ SIMD
            DO ij=ij_begin_ext,ij_end_ext
               etav= 1./Av(ij+z_up)*(  ne_rup        * u(ij+u_rup,l)        * de(ij+u_rup)
                   ↪          &
                    + ne_left * u(ij+t_rup+u_left,l) * de(ij+t_rup+u_left)  &
                    - ne_lup        * u(ij+u_lup,l)        * de(ij+u_lup) )

               hv =  Riv2(ij,vup)          * rhodz(ij,l)            &
                    + Riv2(ij+t_rup,vldown) * rhodz(ij+t_rup,l)     &
                    + Riv2(ij+t_lup,vrdown) * rhodz(ij+t_lup,l)

               if ( hv > 0.0 ) then
                  qv(ij+z_up,l) = ( etav+fv(ij+z_up) )/hv
               else
                  qv(ij+z_up,l) = 0.0
               endif

               etav = 1./Av(ij+z_down)*(  ne_ldown         * u(ij+u_ldown,l)          * de(ij+
                   ↪ u_ldown)          &
                    + ne_right * u(ij+t_ldown+u_right,l)  * de(ij+t_ldown+u_right)  &
                    - ne_rdown         * u(ij+u_rdown,l)          * de(ij+u_rdown) )
```

```
            hv =  Riv2(ij,vdown)         * rhodz(ij,l)           &
                + Riv2(ij+t_ldown,vrup) * rhodz(ij+t_ldown,l)  &
                + Riv2(ij+t_rdown,vlup) * rhodz(ij+t_rdown,l)

            if ( hv > 0.0 ) then
               qv(ij+z_down,l) =( etav+fv(ij+z_down) )/hv
            else
               qv(ij+z_down,l) = 0.0
            endif

         ENDDO

         !DIR$ SIMD
         DO ij=ij_begin,ij_end
            qu(ij+u_right,l) = 0.5*(qv(ij+z_rdown,l)+qv(ij+z_rup,l))
            qu(ij+u_lup,l) = 0.5*(qv(ij+z_up,l)+qv(ij+z_lup,l))
            qu(ij+u_ldown,l) = 0.5*(qv(ij+z_ldown,l)+qv(ij+z_down,l))
         END DO

      ENDDO

      CALL trace_end("compute_pvort")

   END SUBROUTINE compute_pvort
```

3.3.2 compute_geopot

This kernel is based on the subroutine *compute_geopot* from DYNAMICO.

Listing 15: compute_geopot

```
   SUBROUTINE compute_geopot(ps,rhodz,theta, pk,geopot)
      use prec
      use mod_misc
      IMPLICIT NONE
      REAL(rstd),INTENT(INOUT) :: ps(iim*jjm)
      REAL(rstd),INTENT(IN)    :: rhodz(iim*jjm,llm)
      REAL(rstd),INTENT(IN)    :: theta(iim*jjm,llm)     ! potential temperature
      REAL(rstd),INTENT(INOUT) :: pk(iim*jjm,llm)        ! Exner function
      REAL(rstd),INTENT(INOUT) :: geopot(iim*jjm,llm+1) ! geopotential

      INTEGER :: i,j,ij,l
      REAL(rstd) :: p_ik, exner_ik

      CALL trace_start("compute_geopot")

      IF(caldyn_eta==eta_mass) THEN

!!! Compute exner function and geopotential
         DO l = 1,llm
            !DIR$ SIMD
            DO ij=ij_begin_ext,ij_end_ext
               p_ik = ptop + mass_ak(l) + mass_bk(l)*ps(ij) ! FIXME : leave ps for the
                   ↪ moment ; change ps to Ms later
               !         p_ik = ptop + g*(mass_ak(l)+ mass_bk(l)*ps(i,j))
               exner_ik = cpp * (p_ik/preff) ** kappa
               pk(ij,l) = exner_ik
               ! specific volume v = kappa*theta*pi/p = dphi/g/rhodz
               if ( p_ik > 1.D-30 ) then
                  geopot(ij,l+1) = geopot(ij,l) + (g*kappa)*rhodz(ij,l)*theta(ij,l)*
                      ↪ exner_ik/p_ik
               endif
            ENDDO
         ENDDO
         !       ENDIF
      ELSE
         ! We are using a Lagrangian vertical coordinate
         ! Pressure must be computed first top-down (temporarily stored in pk)
         ! Then Exner pressure and geopotential are computed bottom-up
         ! Notice that the computation below should work also when caldyn_eta=eta_mass

         IF(boussinesq) THEN ! compute only geopotential : pressure pk will be computed in
             ↪ compute_caldyn_horiz
```

```
            ! specific volume 1 = dphi/g/rhodz
            !         IF (is_omp_level_master) THEN ! no openMP on vertical due to
                ↪ dependency
            DO l = 1,llm
               !DIR$ SIMD
               DO ij=ij_begin_ext,ij_end_ext
                  geopot(ij,l+1) = geopot(ij,l) + g*rhodz(ij,l)
               ENDDO
            ENDDO
         ELSE ! non-Boussinesq, compute geopotential and Exner pressure
            ! uppermost layer

            !DIR$ SIMD
            DO ij=ij_begin_ext,ij_end_ext
               pk(ij,llm) = ptop + (.5*g)*rhodz(ij,llm)
            END DO
            ! other layers
            DO l = llm-1, 1, -1
               !DIR$ SIMD
               DO ij=ij_begin_ext,ij_end_ext
                  pk(ij,l) = pk(ij,l+1) + (.5*g)*(rhodz(ij,l)+rhodz(ij,l+1))
               END DO
            END DO
            ! surface pressure (for diagnostics)
            DO ij=ij_begin_ext,ij_end_ext
               ps(ij) = pk(ij,1) + (.5*g)*rhodz(ij,1)
            END DO

            ! specific volume v = kappa*theta*pi/p = dphi/g/rhodz
            DO l = 1,llm
               !DIR$ SIMD
               DO ij=ij_begin_ext,ij_end_ext
                  p_ik = pk(ij,l)
                  exner_ik = cpp * (p_ik/preff) ** kappa
                  geopot(ij,l+1) = geopot(ij,l) + (g*kappa)*rhodz(ij,l)*theta(ij,l)*
                      ↪ exner_ik/p_ik
                  pk(ij,l) = exner_ik
               ENDDO
            ENDDO
         END IF

      END IF

      !ym flush geopot
      !$OMP BARRIER

      CALL trace_end("compute_geopot")

   END SUBROUTINE compute_geopot
```

3.3.3 compute_caldyn_horiz

This kernel is based on the subroutine *compute_caldyn_horiz* from DYNAMICO.

Listing 16: compute_caldyn_horiz

```
   SUBROUTINE compute_caldyn_horiz(u,rhodz,qu,theta,pk,geopot, hflux,convm, dtheta_rhodz,
       ↪ du)
      use prec
      use mod_misc
      IMPLICIT NONE
      REAL(rstd),INTENT(IN)  :: u(iim*3*jjm,llm)     ! prognostic "velocity"
      REAL(rstd),INTENT(IN)  :: rhodz(iim*jjm,llm)
      REAL(rstd),INTENT(IN)  :: qu(iim*3*jjm,llm)
      REAL(rstd),INTENT(IN)  :: theta(iim*jjm,llm)  ! potential temperature
      REAL(rstd),INTENT(INOUT) :: pk(iim*jjm,llm) ! Exner function
      REAL(rstd),INTENT(IN)  :: geopot(iim*jjm,llm+1)     ! geopotential

      REAL(rstd),INTENT(INOUT) :: hflux(iim*3*jjm,llm) ! hflux in kg/s
      REAL(rstd),INTENT(INOUT) :: convm(iim*jjm,llm)  ! mass flux convergence
      REAL(rstd),INTENT(INOUT) :: dtheta_rhodz(iim*jjm,llm)
      REAL(rstd),INTENT(INOUT) :: du(iim*3*jjm,llm)
```

```
        REAL(rstd) :: cor_NT(iim*jjm,llm)  ! NT coriolis force u.(du/dPhi)
        REAL(rstd) :: urel(3*iim*jjm,llm)  ! relative velocity
        REAL(rstd) :: Ftheta(3*iim*jjm,llm) ! theta flux
        REAL(rstd) :: berni(iim*jjm,llm)  ! Bernoulli function

        INTEGER :: i,j,ij,l
        REAL(rstd) :: ww,uu

        CALL trace_start('compute_caldyn_horiz')

        !    CALL wait_message(req_theta_rhodz)

        DO l = ll_begin, ll_end
!!!  Compute mass and theta fluxes
           IF (caldyn_conserv==energy) CALL test_message(req_qu)
           !DIR$ SIMD
           DO ij=ij_begin_ext,ij_end_ext
              hflux(ij+u_right,l)=0.5*(rhodz(ij,l)+rhodz(ij+t_right,l))*u(ij+u_right,l)*le(ij
                  ↪ +u_right)
              hflux(ij+u_lup,l)=0.5*(rhodz(ij,l)+rhodz(ij+t_lup,l))*u(ij+u_lup,l)*le(ij+u_lup
                  ↪ )
              hflux(ij+u_ldown,l)=0.5*(rhodz(ij,l)+rhodz(ij+t_ldown,l))*u(ij+u_ldown,l)*le(ij
                  ↪ +u_ldown)

              Ftheta(ij+u_right,l)=0.5*(theta(ij,l)+theta(ij+t_right,l))*hflux(ij+u_right,l)
              Ftheta(ij+u_lup,l)=0.5*(theta(ij,l)+theta(ij+t_lup,l))*hflux(ij+u_lup,l)
              Ftheta(ij+u_ldown,l)=0.5*(theta(ij,l)+theta(ij+t_ldown,l))*hflux(ij+u_ldown,l)
           ENDDO

!!! compute horizontal divergence of fluxes
           !DIR$ SIMD
           DO ij=ij_begin,ij_end
              ! convm = -div(mass flux), sign convention as in Ringler et al. 2012, eq. 21
              convm(ij,l)= -1./Ai(ij)*(ne_right*hflux(ij+u_right,l)   +  &
                   ne_rup*hflux(ij+u_rup,l)       +  &
                   ne_lup*hflux(ij+u_lup,l)       +  &
                   ne_left*hflux(ij+u_left,l)     +  &
                   ne_ldown*hflux(ij+u_ldown,l)   +  &
                   ne_rdown*hflux(ij+u_rdown,l))

              ! signe ? attention d (rho theta dz)
              ! dtheta_rhodz =  -div(flux.theta)
              dtheta_rhodz(ij,l)=-1./Ai(ij)*(ne_right*Ftheta(ij+u_right,l)    +  &
                   ne_rup*Ftheta(ij+u_rup,l)       +  &
                   ne_lup*Ftheta(ij+u_lup,l)       +  &
                   ne_left*Ftheta(ij+u_left,l)     +  &
                   ne_ldown*Ftheta(ij+u_ldown,l)   +  &
                   ne_rdown*Ftheta(ij+u_rdown,l))
           ENDDO

        END DO

!!! Compute potential vorticity (Coriolis) contribution to du

        SELECT CASE(caldyn_conserv)
        CASE(energy) ! energy-conserving TRiSK

           CALL wait_message(req_qu)

           DO l=ll_begin,ll_end
              !DIR$ SIMD
              DO ij=ij_begin,ij_end

                 !             if ( de(ij+u_right) > 1.0 ) then
                 uu = wee(ij+u_right,1,1)*hflux(ij+u_rup,l)*(qu(ij+u_right,l)+qu(ij+u_rup,l))
                     ↪ +                                  &
                      wee(ij+u_right,2,1)*hflux(ij+u_lup,l)*(qu(ij+u_right,l)+qu(ij+u_lup,l))
                          ↪ +                             &
                      wee(ij+u_right,3,1)*hflux(ij+u_left,l)*(qu(ij+u_right,l)+qu(ij+u_left,l
                          ↪ ))+                           &
                      wee(ij+u_right,4,1)*hflux(ij+u_ldown,l)*(qu(ij+u_right,l)+qu(ij+u_ldown
                          ↪ ,l))+                         &
                      wee(ij+u_right,5,1)*hflux(ij+u_rdown,l)*(qu(ij+u_right,l)+qu(ij+u_rdown
                          ↪ ,l))+                         &
```

```
                    wee(ij+u_right,1,2)*hflux(ij+t_right+u_ldown,l)*(qu(ij+u_right,l)+qu(ij
                        ↪ +t_right+u_ldown,l))+          &
                    wee(ij+u_right,2,2)*hflux(ij+t_right+u_rdown,l)*(qu(ij+u_right,l)+qu(ij
                        ↪ +t_right+u_rdown,l))+          &
                    wee(ij+u_right,3,2)*hflux(ij+t_right+u_right,l)*(qu(ij+u_right,l)+qu(ij
                        ↪ +t_right+u_right,l))+          &
                    wee(ij+u_right,4,2)*hflux(ij+t_right+u_rup,l)*(qu(ij+u_right,l)+qu(ij+
                        ↪ t_right+u_rup,l))+             &
                    wee(ij+u_right,5,2)*hflux(ij+t_right+u_lup,l)*(qu(ij+u_right,l)+qu(ij+
                        ↪ t_right+u_lup,l))
                du(ij+u_right,l) = .5*uu/de(ij+u_right)
                !             endif

                if ( de(ij+u_lup) > 1.0 ) then
                   uu = wee(ij+u_lup,1,1)*hflux(ij+u_left,l)*(qu(ij+u_lup,l)+qu(ij+u_left,l)
                       ↪ ) +                             &
                        wee(ij+u_lup,2,1)*hflux(ij+u_ldown,l)*(qu(ij+u_lup,l)+qu(ij+u_ldown,
                            ↪ l)) +                      &
                        wee(ij+u_lup,3,1)*hflux(ij+u_rdown,l)*(qu(ij+u_lup,l)+qu(ij+u_rdown,
                            ↪ l)) +                      &
                        wee(ij+u_lup,4,1)*hflux(ij+u_right,l)*(qu(ij+u_lup,l)+qu(ij+u_right,
                            ↪ l)) +                      &
                        wee(ij+u_lup,5,1)*hflux(ij+u_rup,l)*(qu(ij+u_lup,l)+qu(ij+u_rup,l))
                            ↪ +                          &
                        wee(ij+u_lup,1,2)*hflux(ij+t_lup+u_right,l)*(qu(ij+u_lup,l)+qu(ij+
                            ↪ t_lup+u_right,l)) +        &
                        wee(ij+u_lup,2,2)*hflux(ij+t_lup+u_rup,l)*(qu(ij+u_lup,l)+qu(ij+
                            ↪ t_lup+u_rup,l)) +          &
                        wee(ij+u_lup,3,2)*hflux(ij+t_lup+u_lup,l)*(qu(ij+u_lup,l)+qu(ij+
                            ↪ t_lup+u_lup,l)) +          &
                        wee(ij+u_lup,4,2)*hflux(ij+t_lup+u_left,l)*(qu(ij+u_lup,l)+qu(ij+
                            ↪ t_lup+u_left,l)) +         &
                        wee(ij+u_lup,5,2)*hflux(ij+t_lup+u_ldown,l)*(qu(ij+u_lup,l)+qu(ij+
                            ↪ t_lup+u_ldown,l))
                   du(ij+u_lup,l) = .5*uu/de(ij+u_lup)
                endif

                if ( de(ij+u_ldown) > 1.0 ) then
                   uu = wee(ij+u_ldown,1,1)*hflux(ij+u_rdown,l)*(qu(ij+u_ldown,l)+qu(ij+
                       ↪ u_rdown,l)) +                   &
                        wee(ij+u_ldown,2,1)*hflux(ij+u_right,l)*(qu(ij+u_ldown,l)+qu(ij+
                            ↪ u_right,l)) +              &
                        wee(ij+u_ldown,3,1)*hflux(ij+u_rup,l)*(qu(ij+u_ldown,l)+qu(ij+u_rup,
                            ↪ l)) +                      &
                        wee(ij+u_ldown,4,1)*hflux(ij+u_lup,l)*(qu(ij+u_ldown,l)+qu(ij+u_lup,
                            ↪ l)) +                      &
                        wee(ij+u_ldown,5,1)*hflux(ij+u_left,l)*(qu(ij+u_ldown,l)+qu(ij+
                            ↪ u_left,l)) +               &
                        wee(ij+u_ldown,1,2)*hflux(ij+t_ldown+u_lup,l)*(qu(ij+u_ldown,l)+qu(
                            ↪ ij+t_ldown+u_lup,l)) +     &
                        wee(ij+u_ldown,2,2)*hflux(ij+t_ldown+u_left,l)*(qu(ij+u_ldown,l)+qu(
                            ↪ ij+t_ldown+u_left,l)) +    &
                        wee(ij+u_ldown,3,2)*hflux(ij+t_ldown+u_ldown,l)*(qu(ij+u_ldown,l)+qu
                            ↪ (ij+t_ldown+u_ldown,l)) +  &
                        wee(ij+u_ldown,4,2)*hflux(ij+t_ldown+u_rdown,l)*(qu(ij+u_ldown,l)+qu
                            ↪ (ij+t_ldown+u_rdown,l)) +  &
                        wee(ij+u_ldown,5,2)*hflux(ij+t_ldown+u_right,l)*(qu(ij+u_ldown,l)+qu
                            ↪ (ij+t_ldown+u_right,l))
                   du(ij+u_ldown,l) = .5*uu/de(ij+u_ldown)
                endif

             ENDDO
          ENDDO

       CASE(enstrophy) ! enstrophy-conserving TRiSK

          DO l=ll_begin,ll_end
             !DIR$ SIMD
             DO ij=ij_begin,ij_end

                !             if ( de(ij+u_right) > 1.0 ) then
                uu = wee(ij+u_right,1,1)*hflux(ij+u_rup,l)+                          &
                     wee(ij+u_right,2,1)*hflux(ij+u_lup,l)+                          &
                     wee(ij+u_right,3,1)*hflux(ij+u_left,l)+                         &
                     wee(ij+u_right,4,1)*hflux(ij+u_ldown,l)+                        &
```

```
                    wee(ij+u_right,5,1)*hflux(ij+u_rdown,l)+                    &
                    wee(ij+u_right,1,2)*hflux(ij+t_right+u_ldown,l)+            &
                    wee(ij+u_right,2,2)*hflux(ij+t_right+u_rdown,l)+            &
                    wee(ij+u_right,3,2)*hflux(ij+t_right+u_right,l)+            &
                    wee(ij+u_right,4,2)*hflux(ij+t_right+u_rup,l)+              &
                    wee(ij+u_right,5,2)*hflux(ij+t_right+u_lup,l)
                du(ij+u_right,l) = qu(ij+u_right,l)*uu/de(ij+u_right)
                !             endif

                if ( de(ij+u_lup) > 1.0 ) then
                   uu = wee(ij+u_lup,1,1)*hflux(ij+u_left,l)+                   &
                        wee(ij+u_lup,2,1)*hflux(ij+u_ldown,l)+                  &
                        wee(ij+u_lup,3,1)*hflux(ij+u_rdown,l)+                  &
                        wee(ij+u_lup,4,1)*hflux(ij+u_right,l)+                  &
                        wee(ij+u_lup,5,1)*hflux(ij+u_rup,l)+                    &
                        wee(ij+u_lup,1,2)*hflux(ij+t_lup+u_right,l)+            &
                        wee(ij+u_lup,2,2)*hflux(ij+t_lup+u_rup,l)+              &
                        wee(ij+u_lup,3,2)*hflux(ij+t_lup+u_lup,l)+              &
                        wee(ij+u_lup,4,2)*hflux(ij+t_lup+u_left,l)+             &
                        wee(ij+u_lup,5,2)*hflux(ij+t_lup+u_ldown,l)
                   du(ij+u_lup,l) = qu(ij+u_lup,l)*uu/de(ij+u_lup)
                endif

                if ( de(ij+u_ldown) > 1.0 ) then
                   uu = wee(ij+u_ldown,1,1)*hflux(ij+u_rdown,l)+                &
                        wee(ij+u_ldown,2,1)*hflux(ij+u_right,l)+                &
                        wee(ij+u_ldown,3,1)*hflux(ij+u_rup,l)+                  &
                        wee(ij+u_ldown,4,1)*hflux(ij+u_lup,l)+                  &
                        wee(ij+u_ldown,5,1)*hflux(ij+u_left,l)+                 &
                        wee(ij+u_ldown,1,2)*hflux(ij+t_ldown+u_lup,l)+          &
                        wee(ij+u_ldown,2,2)*hflux(ij+t_ldown+u_left,l)+         &
                        wee(ij+u_ldown,3,2)*hflux(ij+t_ldown+u_ldown,l)+        &
                        wee(ij+u_ldown,4,2)*hflux(ij+t_ldown+u_rdown,l)+        &
                        wee(ij+u_ldown,5,2)*hflux(ij+t_ldown+u_right,l)
                   du(ij+u_ldown,l) = qu(ij+u_ldown,l)*uu/de(ij+u_ldown)
                endif

             ENDDO
          ENDDO

       CASE DEFAULT
          STOP
       END SELECT

!!! Compute bernouilli term = Kinetic Energy + geopotential
       IF(boussinesq) THEN
          ! first use hydrostatic balance with theta*rhodz to find pk (Lagrange multiplier=
              ↪ pressure)
          ! uppermost layer
          !DIR$ SIMD
          DO ij=ij_begin_ext,ij_end_ext
             pk(ij,llm) = ptop + (.5*g)*theta(ij,llm)*rhodz(ij,llm)
          END DO
          ! other layers
          DO l = llm-1, 1, -1
             !          !$OMP DO SCHEDULE(STATIC)
             !DIR$ SIMD
             DO ij=ij_begin_ext,ij_end_ext
                pk(ij,l) = pk(ij,l+1) + (.5*g)*(theta(ij,l)*rhodz(ij,l)+theta(ij,l+1)*rhodz(
                    ↪ ij,l+1))
             END DO
          END DO
          ! surface pressure (for diagnostics) FIXME
          ! DO ij=ij_begin_ext,ij_end_ext
          !   ps(ij) = pk(ij,1) + (.5*g)*theta(ij,1)*rhodz(ij,1)
          ! END DO
          ! now pk contains the Lagrange multiplier (pressure)


          DO l=ll_begin,ll_end
             DO ij=ij_begin_ext,ij_end_ext
                berni(ij,l) = pk(ij,l)
             END DO
             !DIR$ SIMD
             DO ij=ij_begin,ij_end
```

```
                if ( Ai(ij) > 1.e-30 ) then
                   berni(ij,l) = berni(ij,l) + &
                        1/(4*Ai(ij))*(le(ij+u_right)*de(ij+u_right)*u(ij+u_right,l)**2 +
                            ↪ &
                        le(ij+u_rup)*de(ij+u_rup)*u(ij+u_rup,l)**2 +          &
                        le(ij+u_lup)*de(ij+u_lup)*u(ij+u_lup,l)**2 +          &
                        le(ij+u_left)*de(ij+u_left)*u(ij+u_left,l)**2 +       &
                        le(ij+u_ldown)*de(ij+u_ldown)*u(ij+u_ldown,l)**2 +    &
                        le(ij+u_rdown)*de(ij+u_rdown)*u(ij+u_rdown,l)**2 )
                endif
             ENDDO

             ! from now on pk contains the vertically-averaged geopotential
             DO ij=ij_begin_ext,ij_end_ext
                pk(ij,l) = .5*(geopot(ij,l)+geopot(ij,l+1))
             END DO
          ENDDO

       ELSE ! compressible

          berni(:,:) = 0.0

          DO l=ll_begin,ll_end
             DO ij=ij_begin_ext,ij_end_ext
                berni(ij,l) = .5*(geopot(ij,l)+geopot(ij,l+1))
             END DO
             !DIR$ SIMD
             DO ij=ij_begin,ij_end
                if ( Ai(ij) > 1.e-30 ) then
                   berni(ij,l) = berni(ij,l) &
                        + 1/(4*Ai(ij))*(le(ij+u_right)*de(ij+u_right)*u(ij+u_right,l)**2 +
                            ↪    &
                        le(ij+u_rup)*de(ij+u_rup)*u(ij+u_rup,l)**2 +          &
                        le(ij+u_lup)*de(ij+u_lup)*u(ij+u_lup,l)**2 +          &
                        le(ij+u_left)*de(ij+u_left)*u(ij+u_left,l)**2 +       &
                        le(ij+u_ldown)*de(ij+u_ldown)*u(ij+u_ldown,l)**2 +    &
                        le(ij+u_rdown)*de(ij+u_rdown)*u(ij+u_rdown,l)**2 )
                endif
             ENDDO
          ENDDO

       END IF ! Boussinesq/compressible

!!! Add gradients of Bernoulli and Exner functions to du
       DO l=ll_begin,ll_end
          !DIR$ SIMD
          DO ij=ij_begin,ij_end

             if ( de(ij+u_right) > 1.0 ) then
                du(ij+u_right,l) = du(ij+u_right,l) + 1/de(ij+u_right) * (             &
                     0.5*(theta(ij,l)+theta(ij+t_right,l))                  &
                     *( ne_right*pk(ij,l)+ne_left*pk(ij+t_right,l))    &
                     + ne_right*berni(ij,l)+ne_left*berni(ij+t_right,l) )
             endif

             if ( de(ij+u_lup) > 1.0 ) then
                du(ij+u_lup,l) = du(ij+u_lup,l) +  1/de(ij+u_lup) * (                  &
                     0.5*(theta(ij,l)+theta(ij+t_lup,l))                    &
                     *( ne_lup*pk(ij,l)+ne_rdown*pk(ij+t_lup,l))       &
                     + ne_lup*berni(ij,l)+ne_rdown*berni(ij+t_lup,l) )
             endif

             if ( de(ij+u_ldown) > 1.0 ) then
                du(ij+u_ldown,l) = du(ij+u_ldown,l) + 1/de(ij+u_ldown) * (             &
                     0.5*(theta(ij,l)+theta(ij+t_ldown,l))                  &
                     *( ne_ldown*pk(ij,l)+ne_rup*pk(ij+t_ldown,l))     &
                     + ne_ldown*berni(ij,l)+ne_rup*berni(ij+t_ldown,l) )
             endif

          ENDDO
       ENDDO

       CALL trace_end( 'compute_caldyn_horiz' )

    END SUBROUTINE compute_caldyn_horiz
```

3.3.4 compute_caldyn_vert

This kernel is based on the subroutine *compute_caldyn_vert* from DYNAMICO.

Listing 17: compute_caldyn_vert

```
      SUBROUTINE compute_caldyn_vert(u,theta,rhodz,convm, wflux,wwuu, dps,dtheta_rhodz,du)
         use prec
         use mod_misc
         IMPLICIT NONE
         REAL(rstd),INTENT(IN)  :: u(iim*3*jjm,llm)
         REAL(rstd),INTENT(IN)  :: theta(iim*jjm,llm)
         REAL(rstd),INTENT(IN)  :: rhodz(iim*jjm,llm)
         REAL(rstd),INTENT(INOUT)  :: convm(iim*jjm,llm)  ! mass flux convergence

         REAL(rstd),INTENT(INOUT) :: wflux(iim*jjm,llm+1) ! vertical mass flux (kg/m2/s)
         REAL(rstd),INTENT(INOUT) :: wwuu(iim*3*jjm,llm+1)
         REAL(rstd),INTENT(INOUT) :: du(iim*3*jjm,llm)
         REAL(rstd),INTENT(INOUT) :: dtheta_rhodz(iim*jjm,llm)
         REAL(rstd),INTENT(INOUT) :: dps(iim*jjm)

         ! temporary variable
         INTEGER :: i,j,ij,l
         REAL(rstd) :: p_ik, exner_ik
!!$!!$KERN     INTEGER,SAVE ::ij_omp_begin, ij_omp_end
!!$!!$KERN!$OMP THREADPRIVATE(ij_omp_begin, ij_omp_end)
!!$!!$KERN     LOGICAL,SAVE :: first=.TRUE.
         LOGICAL,SAVE :: first=.false.
         !$OMP THREADPRIVATE(first)

         CALL trace_start("compute_geopot")

!!$!!##KERNEL: this section is never called when kernelize.
!!$!!$KERN     IF (first) THEN
!!$!!$KERN       first=.FALSE.
!!$!!$KERN       CALL distrib_level(ij_end-ij_begin+1,ij_omp_begin,ij_omp_end)
!!$!!$KERN       ij_omp_begin=ij_omp_begin+ij_begin-1
!!$!!$KERN       ij_omp_end=ij_omp_end+ij_begin-1
!!$!!$KERN     ENDIF

!     REAL(rstd) :: wwuu(iim*3*jjm,llm+1) ! tmp var, don't know why but gain 30% on the
    ↪ whole code in opemp
         ! need to be understood

         !    wwuu=wwuu_out
         CALL trace_start("compute_caldyn_vert")

         !$OMP BARRIER
!!! cumulate mass flux convergence from top to bottom
         !  IF (is_omp_level_master) THEN
         DO  l = llm-1, 1, -1
            !    IF (caldyn_conserv==energy) CALL test_message(req_qu)

!!$OMP DO SCHEDULE(STATIC)
            !DIR$ SIMD
            DO ij=ij_omp_begin,ij_omp_end
               convm(ij,l) = convm(ij,l) + convm(ij,l+1)
            ENDDO
         ENDDO
         !  ENDIF

         !$OMP BARRIER
         ! FLUSH on convm
!!!!!!!!!!!!!!!!!!!!!!!!!!

         ! compute dps
         IF (is_omp_first_level) THEN
            !DIR$ SIMD
            DO ij=ij_begin,ij_end
               ! dps/dt = -int(div flux)dz
               dps(ij) = convm(ij,1) * g
            ENDDO
```

```
        ENDIF

!!! Compute vertical mass flux (l=1,llm+1 done by caldyn_BC)
        DO l=ll_beginp1,ll_end
           !    IF (caldyn_conserv==energy) CALL test_message(req_qu)
           !DIR$ SIMD
           DO ij=ij_begin,ij_end
              ! w = int(z,ztop,div(flux)dz) + B(eta)dps/dt
              ! => w>0 for upward transport
              wflux( ij, l ) = bp(l) * convm( ij, 1 ) - convm( ij, l )
           ENDDO
        ENDDO

        !--> flush wflux
        !$OMP BARRIER

        DO l=ll_begin,ll_endm1
           !DIR$ SIMD
           DO ij=ij_begin,ij_end
              dtheta_rhodz(ij, l ) = dtheta_rhodz(ij, l )  -   0.5 * (  wflux(ij,l+1) * (
                  ↪ theta(ij,l) + theta(ij,l+1)))
           ENDDO
        ENDDO

        DO l=ll_beginp1,ll_end
           !DIR$ SIMD
           DO ij=ij_begin,ij_end
              dtheta_rhodz(ij, l ) = dtheta_rhodz(ij, l )  +   0.5 * ( wflux(ij,l  ) * (theta
                  ↪ (ij,l-1) + theta(ij,l) ) )
           ENDDO
        ENDDO


        ! Compute vertical transport
        DO l=ll_beginp1,ll_end
           !DIR$ SIMD
           DO ij=ij_begin,ij_end
              wwuu(ij+u_right,l) = 0.5*( wflux(ij,l) + wflux(ij+t_right,l)) * (u(ij+u_right,l
                  ↪ ) - u(ij+u_right,l-1))
              wwuu(ij+u_lup,l) = 0.5* ( wflux(ij,l) + wflux(ij+t_lup,l)) * (u(ij+u_lup,l) - u
                  ↪ (ij+u_lup,l-1))
              wwuu(ij+u_ldown,l) = 0.5*( wflux(ij,l) + wflux(ij+t_ldown,l)) * (u(ij+u_ldown,l
                  ↪ ) - u(ij+u_ldown,l-1))
           ENDDO
        ENDDO

        !--> flush wwuu
        !$OMP BARRIER

        ! Add vertical transport to du
        DO l=ll_begin,ll_end
           !DIR$ SIMD
           DO ij=ij_begin,ij_end
              du(ij+u_right, l )   = du(ij+u_right,l)  - (wwuu(ij+u_right,l+1)+ wwuu(ij+
                  ↪ u_right,l)) / (rhodz(ij,l)+rhodz(ij+t_right,l))
              du(ij+u_lup, l )     = du(ij+u_lup,l)    - (wwuu(ij+u_lup,l+1)  + wwuu(ij+u_lup
                  ↪ ,l))   / (rhodz(ij,l)+rhodz(ij+t_lup,l))
              du(ij+u_ldown, l )   = du(ij+u_ldown,l)  - (wwuu(ij+u_ldown,l+1)+ wwuu(ij+
                  ↪ u_ldown,l)) / (rhodz(ij,l)+rhodz(ij+t_ldown,l))
           ENDDO
        ENDDO

        !  DO l=ll_beginp1,ll_end
        !!DIR$ SIMD
        !      DO ij=ij_begin,ij_end
        !        wwuu_out(ij+u_right,l) = wwuu(ij+u_right,l)
        !        wwuu_out(ij+u_lup,l)   = wwuu(ij+u_lup,l)
        !        wwuu_out(ij+u_ldown,l) = wwuu(ij+u_ldown,l)
        !     ENDDO
        !   ENDDO

        CALL trace_end( compute_caldyn_vert )

    END SUBROUTINE compute_caldyn_vert
```

4 NICAM-based Mini IGCM

This mini IGCM is based on the NICAM code. NICAM is a global atmospheric model that uses an icosahedral grid. Recursive division of the edges of the icosahedron allow to gain fine resolution grids. An Arakawa-A grid is used then to localize variables at the vertices of the grid triangles. Hexagonal cells are generally formed by those triangular shapes. However, there are still 12 points resulting from the icosahedron structure that lead to pentagonal cells.
The benchmark provides a public documentation that can be accessed through the folder `docs/NICAM` in the root folder of the benchmark. The documentation can be built using the make utility using the Makefile at the `docs` folder.

4.1 Structure

The grid is divided into regions to allow execution on multiple nodes. Multiple regions can be assigned to a single process. Memory copy is done on node if two regions reside on the same process. Otherwise, communication is needed to handle the halo exchange, where MPI is used.

4.1.1 Problem Domain and Size

The problem domain of the NICAM-based mini-IGCM kernels is defined over the three dimensional space with two indices; one to address the horizontal, and another for the vertical (Listing 18).

Listing 18: NICAM mini-IGCM problem size definition

```
!--- region
integer, public             :: ADM_lall      = 1     ! number of regular region per
    ↪ process
integer, public, parameter  :: ADM_lall_pl   = 2     ! number of pole    region per
    ↪ process

!--- horizontal grid
integer, public             :: ADM_gall      = 16900 ! number of horizontal grid per
    ↪ regular region
integer, public             :: ADM_gall_1d   = 130   ! number of horizontal grid (1D)
integer, public             :: ADM_gmin      = 2     ! start index of 1D horizontal grid
integer, public             :: ADM_gmax      = 129   ! end   index of 1D horizontal grid

integer, public             :: ADM_gall_pl   = 6     ! number of horizontal grid for pole
    ↪  region
integer, public, parameter  :: ADM_gslf_pl   = 1     ! index for pole point
integer, public, parameter  :: ADM_gmin_pl   = 2     ! start index of grid around the
    ↪ pole point
integer, public             :: ADM_gmax_pl   = 6     ! end   index of grid around the
    ↪ pole point
integer, public, parameter  :: ADM_vlink     = 5     ! number of grid around the pole
    ↪ point

!--- vertical grid
integer, public             :: ADM_vlayer    = 40    ! number of vertical layer
integer, public             :: ADM_kall      = 42    ! number of vertical grid
integer, public             :: ADM_kmin      = 2     ! start index of vertical grid
integer, public             :: ADM_kmax      = 41    ! end   index of vertical grid
```

The shown snippet shows the problem size. A set of regions is processed by each process. Each region includes a provided number of grid elements. Special treatment is given for the pole regions.

4.1.2 I/O

The NICAM-based mini-IGCM provides a set of I/O functions to handle array storage:

- dumpio_fopen: open file IO stream
- dumpio_fclose: close file IO stream
- dumpio_write_data: write data array
- dumpio_read_data: read data array

4.1.3 Communication

The NICAM-based mini-IGCM includes public procedures to handle communication using MPI between the running processes (Listing 19). The procedures setup the communication and do the halo exchange between the processes.

Listing 19: NICAM mini-IGCM communication public procedures

```
  public :: COMM_setup
  public :: COMM_data_transfer
  public :: COMM_var
```

4.2 Running the mini-IGCM

The mini-IGCM can be downloaded from the already mentioned benchmark link. The NICAM-based mini-IGCM is located in the sub-folder `kernels/NICAM`. Each of the kernels is provided in its own folder. One sub-folder within each kernel's folder contains the source code of the kernel and the necessary additional code to make the kernel ready to run. Another sub-folder contains files to access test data for testing purposes as input to run the kernel. Also, a sub-folder contains scripts to run the kernel on Linux based machines, on the machine Mistral (in the German Climate Computing Center–DKRZ), and on the K computer (in RIKEN institute). Scripts are also included to run the kernels on Mistral and K computer with Scalasca. A fourth sub-folder is included in each kernel's folder, where log files are provided as reference for the purpose of validation.
To run a kernel, the input data can be first downloaded using the shell script `kernels/NICAM/kernel-name/data/download.sh`, where kernel-name is the name of the kernel to run. The script downloads the necessary data files to run the kernel. To check the integrity of the data files, there exists an md5 file in the same folder. To configure the communication, there is a set of files in the folder `kernels/NICAM/communication/data` to allow communication with different problem sizes. By default the $gl04rl00$ configuration is used, but also $gl04rl01$ and $gl04rl02$ are provided and their files can be renamed into `communication.cnf` to use those configurations. After the data files are downloaded, the kernel needs to be compiled. First, the environment variable IAB_SYS should be exported and assigned the name of one of the files in the `sysdep` folder at the root of the benchmark folder. One of this set of predefined files or an alternative modified version of them can be assigned to the environment variable. Next, the Makefile in the kernel source (`src`) folder can be used to build the code using the make tool.
After compiling a kernel, the shell scripts in `kernels/NICAM/kernel-name/run` can be used to run the kernel (as mentioned, there are five scripts to run the kernels on a Linux machines in general or on Mistral or K computer, and Mistral/K computer with Scalasca).
When the kernels are run, the results can be validated using the files in `kernels/NICAM/kernel-name/reference`.

4.3 Operators

In this section we present a set of procedures comprising the NICAM based mini-IGCM code.

4.3.1 OPRT_diffusion

This kernel is based on the subroutine *OPRT_diffusion* from NICAM.

Listing 20: OPRT_diffusion

```
  subroutine OPRT_diffusion( &
       dscl,       dscl_pl,       &
       scl,        scl_pl,        &
       kh,         kh_pl,         &
       coef_intp,  coef_intp_pl,  &
       coef_diff,  coef_diff_pl   )
!ESC!    use mod_adm, only: &
!ESC!        ADM_have_pl,  &
!ESC!        ADM_have_sgp, &
!ESC!        ADM_gall_1d,  &
!ESC!        ADM_gmin,     &
!ESC!        ADM_gmax,     &
!ESC!        ADM_gslf_pl,  &
!ESC!        ADM_gmin_pl,  &
!ESC!        ADM_gmax_pl
!ESC!    use mod_grd, only: &
```

```
!ESC!       XDIR => GRD_XDIR, &
!ESC!       YDIR => GRD_YDIR, &
!ESC!       ZDIR => GRD_ZDIR
    implicit none

    real(RP), intent(out) :: dscl        (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(out) :: dscl_pl     (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: scl         (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)  :: scl_pl      (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: kh          (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)  :: kh_pl       (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: coef_intp   (ADM_gall   ,1:3,     ADM_nxyz,TI:TJ,ADM_lall   )
    real(RP), intent(in)  :: coef_intp_pl(ADM_gall_pl,1:3,     ADM_nxyz,      ADM_lall_pl)
    real(RP), intent(in)  :: coef_diff   (ADM_gall   ,1:6,     ADM_nxyz,      ADM_lall   )
    real(RP), intent(in)  :: coef_diff_pl(              1:ADM_vlink,ADM_nxyz,
        ↪ ADM_lall_pl)

    real(RP) :: vt   (ADM_gall   ,ADM_nxyz,TI:TJ)
    real(RP) :: vt_pl(ADM_gall_pl,ADM_nxyz)
    real(RP) :: kf   (1:6)

    integer  :: gmin, gmax, iall, gall, kall, lall, nxyz, gminm1

    integer  :: ij
    integer  :: ip1j, ijp1, ip1jp1
    integer  :: im1j, ijm1, im1jm1

    integer  :: g, k, l, d, n, v
    !---------------------------------------------------------------------------

    call DEBUG_rapstart( 'OPRT_diffusion' )

    gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
    gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
    iall = ADM_gall_1d
    gall = ADM_gall
    kall = ADM_kall
    lall = ADM_lall
    nxyz = ADM_nxyz

    gminm1 = (ADM_gmin-1-1)*ADM_gall_1d + ADM_gmin-1

    !$omp parallel default(none),private(g,k,l,d,ij,ip1j,ip1jp1,ijp1,im1j,ijm1,im1jm1), &
    !$omp shared(ADM_have_sgp,gminm1,gmin,gmax,iall,gall,kall,lall,nxyz,dscl,scl,kh,kf,vt,
        ↪ coef_intp,coef_diff)
    do l = 1, lall
    do k = 1, kall

       do d = 1, nxyz
          !$omp do
          do g = gminm1, gmax
             ij     = g
             ip1j   = g + 1
             ip1jp1 = g + iall + 1
             ijp1   = g + iall

             vt(g,d,TI) = ( ( + 2.0_RP * coef_intp(g,1,d,TI,l) &
                              - 1.0_RP * coef_intp(g,2,d,TI,l) &
                              - 1.0_RP * coef_intp(g,3,d,TI,l) ) * scl(ij    ,k,l) &
                          + ( - 1.0_RP * coef_intp(g,1,d,TI,l) &
                              + 2.0_RP * coef_intp(g,2,d,TI,l) &
                              - 1.0_RP * coef_intp(g,3,d,TI,l) ) * scl(ip1j  ,k,l) &
                          + ( - 1.0_RP * coef_intp(g,1,d,TI,l) &
                              - 1.0_RP * coef_intp(g,2,d,TI,l) &
                              + 2.0_RP * coef_intp(g,3,d,TI,l) ) * scl(ip1jp1,k,l) &
                          ) / 3.0_RP
          enddo
          !$omp end do nowait

          !$omp do
          do g = gminm1, gmax
             ij     = g
             ip1j   = g + 1
             ip1jp1 = g + iall + 1
             ijp1   = g + iall
```

```
            vt(g,d,TJ) = ( ( + 2.0_RP * coef_intp(g,1,d,TJ,l) &
                             - 1.0_RP * coef_intp(g,2,d,TJ,l) &
                             - 1.0_RP * coef_intp(g,3,d,TJ,l) ) * scl(ij    ,k,l) &
                         + ( - 1.0_RP * coef_intp(g,1,d,TJ,l) &
                             + 2.0_RP * coef_intp(g,2,d,TJ,l) &
                             - 1.0_RP * coef_intp(g,3,d,TJ,l) ) * scl(ip1jp1,k,l) &
                         + ( - 1.0_RP * coef_intp(g,1,d,TJ,l) &
                             - 1.0_RP * coef_intp(g,2,d,TJ,l) &
                             + 2.0_RP * coef_intp(g,3,d,TJ,l) ) * scl(ijp1  ,k,l) &
                         ) / 3.0_RP
          enddo
          !$omp end do
        enddo

        if ( ADM_have_sgp(l) ) then ! pentagon
          !$omp master
          vt(gminm1,XDIR,TI) = vt(gminm1+1,XDIR,TJ)
          vt(gminm1,YDIR,TI) = vt(gminm1+1,YDIR,TJ)
          vt(gminm1,ZDIR,TI) = vt(gminm1+1,ZDIR,TJ)
          !$omp end master
        endif

!OCL XFILL
        !$omp do
        do g = 1, gmin-1
          dscl(g,k,l) = 0.0_RP
        enddo
        !$omp end do nowait

        !$omp do
        do g = gmin, gmax
          ij     = g
          ip1j   = g + 1
          ip1jp1 = g + iall + 1
          ijp1   = g + iall
          im1j   = g - 1
          im1jm1 = g - iall - 1
          ijm1   = g - iall

          kf(1) = 0.5_RP * ( kh(ij    ,k,l) + kh(ip1jp1,k,l) )
          kf(2) = 0.5_RP * ( kh(ij    ,k,l) + kh(ijp1  ,k,l) )
          kf(3) = 0.5_RP * ( kh(im1j  ,k,l) + kh(ij    ,k,l) )
          kf(4) = 0.5_RP * ( kh(im1jm1,k,l) + kh(ij    ,k,l) )
          kf(5) = 0.5_RP * ( kh(ijm1  ,k,l) + kh(ij    ,k,l) )
          kf(6) = 0.5_RP * ( kh(ij    ,k,l) + kh(ip1j  ,k,l) )

          dscl(g,k,l) = ( kf(1) * coef_diff(g,1,XDIR,l) * ( vt(ij    ,XDIR,TI) + vt(ij    ,
              ↪ XDIR,TJ) ) &
                      + kf(2) * coef_diff(g,2,XDIR,l) * ( vt(ij    ,XDIR,TJ) + vt(im1j  ,
                          ↪ XDIR,TI) ) &
                      + kf(3) * coef_diff(g,3,XDIR,l) * ( vt(im1j  ,XDIR,TI) + vt(im1jm1,
                          ↪ XDIR,TJ) ) &
                      + kf(4) * coef_diff(g,4,XDIR,l) * ( vt(im1jm1,XDIR,TJ) + vt(im1jm1,
                          ↪ XDIR,TI) ) &
                      + kf(5) * coef_diff(g,5,XDIR,l) * ( vt(im1jm1,XDIR,TI) + vt(ijm1  ,
                          ↪ XDIR,TJ) ) &
                      + kf(6) * coef_diff(g,6,XDIR,l) * ( vt(ijm1  ,XDIR,TJ) + vt(ij    ,
                          ↪ XDIR,TI) ) )
        enddo
        !$omp end do

        !$omp do
        do g = gmin, gmax
          ij     = g
          ip1j   = g + 1
          ip1jp1 = g + iall + 1
          ijp1   = g + iall
          im1j   = g - 1
          im1jm1 = g - iall - 1
          ijm1   = g - iall

          kf(1) = 0.5_RP * ( kh(ij    ,k,l) + kh(ip1jp1,k,l) )
          kf(2) = 0.5_RP * ( kh(ij    ,k,l) + kh(ijp1  ,k,l) )
          kf(3) = 0.5_RP * ( kh(im1j  ,k,l) + kh(ij    ,k,l) )
```

```
            kf(4) = 0.5_RP * ( kh(im1jm1,k,l) + kh(ij    ,k,l) )
            kf(5) = 0.5_RP * ( kh(ijm1  ,k,l) + kh(ij    ,k,l) )
            kf(6) = 0.5_RP * ( kh(ij    ,k,l) + kh(ip1j  ,k,l) )

            dscl(g,k,l) = dscl(g,k,l) + ( kf(1) * coef_diff(g,1,YDIR,l) * ( vt(ij    ,XDIR,TI
                ↪ ) + vt(ij    ,YDIR,TJ) ) &
                                      + kf(2) * coef_diff(g,2,YDIR,l) * ( vt(ij    ,XDIR,TJ
                                          ↪ ) + vt(im1j  ,YDIR,TI) ) &
                                      + kf(3) * coef_diff(g,3,YDIR,l) * ( vt(im1j  ,XDIR,TI
                                          ↪ ) + vt(im1jm1,YDIR,TJ) ) &
                                      + kf(4) * coef_diff(g,4,YDIR,l) * ( vt(im1jm1,XDIR,TJ
                                          ↪ ) + vt(im1jm1,YDIR,TI) ) &
                                      + kf(5) * coef_diff(g,5,YDIR,l) * ( vt(im1jm1,XDIR,TI
                                          ↪ ) + vt(ijm1  ,YDIR,TJ) ) &
                                      + kf(6) * coef_diff(g,6,YDIR,l) * ( vt(ijm1  ,XDIR,TJ
                                          ↪ ) + vt(ij    ,YDIR,TI) ) )
         enddo
         !$omp end do

         !$omp do
         do g = gmin, gmax
            ij     = g
            ip1j   = g + 1
            ip1jp1 = g + iall + 1
            ijp1   = g + iall
            im1j   = g - 1
            im1jm1 = g - iall - 1
            ijm1   = g - iall

            kf(1) = 0.5_RP * ( kh(ij    ,k,l) + kh(ip1jp1,k,l) )
            kf(2) = 0.5_RP * ( kh(ij    ,k,l) + kh(ijp1  ,k,l) )
            kf(3) = 0.5_RP * ( kh(im1j  ,k,l) + kh(ij    ,k,l) )
            kf(4) = 0.5_RP * ( kh(im1jm1,k,l) + kh(ij    ,k,l) )
            kf(5) = 0.5_RP * ( kh(ijm1  ,k,l) + kh(ij    ,k,l) )
            kf(6) = 0.5_RP * ( kh(ij    ,k,l) + kh(ip1j  ,k,l) )

            dscl(g,k,l) = dscl(g,k,l) + ( kf(1) * coef_diff(g,1,ZDIR,l) * ( vt(ij    ,XDIR,TI
                ↪ ) + vt(ij    ,ZDIR,TJ) ) &
                                      + kf(2) * coef_diff(g,2,ZDIR,l) * ( vt(ij    ,XDIR,TJ
                                          ↪ ) + vt(im1j  ,ZDIR,TI) ) &
                                      + kf(3) * coef_diff(g,3,ZDIR,l) * ( vt(im1j  ,XDIR,TI
                                          ↪ ) + vt(im1jm1,ZDIR,TJ) ) &
                                      + kf(4) * coef_diff(g,4,ZDIR,l) * ( vt(im1jm1,XDIR,TJ
                                          ↪ ) + vt(im1jm1,ZDIR,TI) ) &
                                      + kf(5) * coef_diff(g,5,ZDIR,l) * ( vt(im1jm1,XDIR,TI
                                          ↪ ) + vt(ijm1  ,ZDIR,TJ) ) &
                                      + kf(6) * coef_diff(g,6,ZDIR,l) * ( vt(ijm1  ,XDIR,TJ
                                          ↪ ) + vt(ij    ,ZDIR,TI) ) )
         enddo
         !$omp end do nowait

!OCL XFILL
         !$omp do
         do g = gmax+1, gall
            dscl(g,k,l) = 0.0_RP
         enddo
         !$omp end do
      enddo ! loop k
      enddo ! loop l
      !$omp end parallel

      if ( ADM_have_pl ) then
         n = ADM_gslf_pl

         do l = 1, ADM_lall_pl
         do k = 1, ADM_kall

            do d = 1, ADM_nxyz
               do v = ADM_gmin_pl, ADM_gmax_pl
                  ij   = v
                  ijp1 = v + 1
                  if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl

                  vt_pl(ij,d) = ( ( + 2.0_RP * coef_intp_pl(v,1,d,l) &
                                    - 1.0_RP * coef_intp_pl(v,2,d,l) &
```

```
                                    - 1.0_RP * coef_intp_pl(v,3,d,l) ) * scl_pl(n   ,k,l) &
                                + ( - 1.0_RP * coef_intp_pl(v,1,d,l) &
                                    + 2.0_RP * coef_intp_pl(v,2,d,l) &
                                    - 1.0_RP * coef_intp_pl(v,3,d,l) ) * scl_pl(ij  ,k,l) &
                                + ( - 1.0_RP * coef_intp_pl(v,1,d,l) &
                                    - 1.0_RP * coef_intp_pl(v,2,d,l) &
                                    + 2.0_RP * coef_intp_pl(v,3,d,l) ) * scl_pl(ijp1,k,l) &
                                ) / 3.0_RP
                   enddo
                enddo

                dscl_pl(:,k,l) = 0.0_RP

                do v = ADM_gmin_pl, ADM_gmax_pl
                   ij   = v
                   ijm1 = v - 1
                   if( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl ! cyclic condition

                   dscl_pl(n,k,l) = dscl_pl(n,k,l) &
                                  + ( coef_diff_pl(v-1,XDIR,l) * ( vt_pl(ijm1,XDIR) + vt_pl(ij,
                                    ↪ XDIR) ) &
                                    + coef_diff_pl(v-1,YDIR,l) * ( vt_pl(ijm1,YDIR) + vt_pl(ij,
                                      ↪ YDIR) ) &
                                    + coef_diff_pl(v-1,ZDIR,l) * ( vt_pl(ijm1,ZDIR) + vt_pl(ij,
                                      ↪ ZDIR) ) &
                                    ) * 0.5_RP * ( kh_pl(n,k,l) + kh_pl(ij,k,l) )
                enddo

             enddo
             enddo
          else
             dscl_pl(:,:,:) = 0.0_RP
          endif

          call DEBUG_rapend( 'OPRT_diffusion')

          return
        end subroutine OPRT_diffusion
```

4.3.2 OPRT3D_divdamp

This kernel is based on the subroutine *OPRT3D_divdamp* from NICAM.

Listing 21: OPRT3D_divdamp

```
        subroutine OPRT3D_divdamp( &
             ddivdx,     ddivdx_pl,     &
             ddivdy,     ddivdy_pl,     &
             ddivdz,     ddivdz_pl,     &
             rhogvx,     rhogvx_pl,     &
             rhogvy,     rhogvy_pl,     &
             rhogvz,     rhogvz_pl,     &
             rhogw,      rhogw_pl,      &
             coef_intp,  coef_intp_pl,  &
             coef_diff,  coef_diff_pl   )
        !ESC!    use mod_adm, only: &
        !ESC!       TI   => ADM_TI,   &
        !ESC!       TJ   => ADM_TJ,   &
        !ESC!       ADM_nxyz,         &
        !ESC!       ADM_have_pl,      &
        !ESC!       ADM_have_sgp,     &
        !ESC!       ADM_vlink,        &
        !ESC!       ADM_lall,         &
        !ESC!       ADM_lall_pl,      &
        !ESC!       ADM_gall,         &
        !ESC!       ADM_gall_1d,      &
        !ESC!       ADM_gall_pl,      &
        !ESC!       ADM_kall,         &
        !ESC!       ADM_gmin,         &
        !ESC!       ADM_gmax,         &
        !ESC!       ADM_gslf_pl,      &
        !ESC!       ADM_gmin_pl,      &
        !ESC!       ADM_gmax_pl,      &
```

```
!ESC!       ADM_kmin,       &
!ESC!       ADM_kmax
!ESC!    use mod_grd, only: &
!ESC!       XDIR => GRD_XDIR, &
!ESC!       YDIR => GRD_YDIR, &
!ESC!       ZDIR => GRD_ZDIR, &
!ESC!       GRD_rdgz
!ESC!    use mod_vmtr, only: &
!ESC!       VMTR_RGSQRTH,     &
!ESC!       VMTR_RGSQRTH_pl,  &
!ESC!       VMTR_RGAM,        &
!ESC!       VMTR_RGAM_pl,     &
!ESC!       VMTR_RGAMH,       &
!ESC!       VMTR_RGAMH_pl,    &
!ESC!       VMTR_C2WfactGz,   &
!ESC!       VMTR_C2WfactGz_pl
    implicit none

    real(RP), intent(out) :: ddivdx      (ADM_gall   ,ADM_kall,ADM_lall   ) ! tendency
    real(RP), intent(out) :: ddivdx_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(out) :: ddivdy      (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(out) :: ddivdy_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(out) :: ddivdz      (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(out) :: ddivdz_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhogvx      (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho*vx { gam2
        ↪  x G^1/2 }
    real(RP), intent(in)  :: rhogvx_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhogvy      (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho*vy { gam2
        ↪  x G^1/2 }
    real(RP), intent(in)  :: rhogvy_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhogvz      (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho*vz { gam2
        ↪  x G^1/2 }
    real(RP), intent(in)  :: rhogvz_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhogw       (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho*w  { gam2
        ↪  x G^1/2 }
    real(RP), intent(in)  :: rhogw_pl    (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: coef_intp   (ADM_gall   ,1:3,ADM_nxyz,TI:TJ,ADM_lall   )
    real(RP), intent(in)  :: coef_intp_pl(ADM_gall_pl,1:3,ADM_nxyz,      ADM_lall_pl)
    real(RP), intent(in)  :: coef_diff   (ADM_gall,1:6         ,ADM_nxyz,ADM_lall   )
    real(RP), intent(in)  :: coef_diff_pl(         1:ADM_vlink,ADM_nxyz,ADM_lall_pl)

    real(RP) :: sclt   (ADM_gall   ,TI:TJ) ! scalar on the hexagon vertex
    real(RP) :: sclt_pl(ADM_gall_pl)
    real(RP) :: sclt_rhogw
    real(RP) :: sclt_rhogw_pl

    real(RP) :: rhogvx_vm   (ADM_gall   )                    ! rho*vx / vertical metrics
    real(RP) :: rhogvx_vm_pl(ADM_gall_pl)
    real(RP) :: rhogvy_vm   (ADM_gall   )                    ! rho*vy / vertical metrics
    real(RP) :: rhogvy_vm_pl(ADM_gall_pl)
    real(RP) :: rhogvz_vm   (ADM_gall   )                    ! rho*vz / vertical metrics
    real(RP) :: rhogvz_vm_pl(ADM_gall_pl)
    real(RP) :: rhogw_vm    (ADM_gall,   ADM_kall,ADM_lall   ) ! rho*w  / vertical metrics
    real(RP) :: rhogw_vm_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)

    integer  :: gmin, gmax, iall, gall, kall, kmin, kmax, lall, gminm1

    integer  :: ij
    integer  :: ip1j, ijp1, ip1jp1
    integer  :: im1j, ijm1, im1jm1

    integer  :: g, k, l, n, v
    !---------------------------------------------------------------------------

    call DEBUG_rapstart('OPRT3D_divdamp')

    gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
    gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
    iall = ADM_gall_1d
    gall = ADM_gall
    kall = ADM_kall
    kmin = ADM_kmin
    kmax = ADM_kmax
    lall = ADM_lall
```

```
      gminm1 = (ADM_gmin-1-1)*ADM_gall_1d + ADM_gmin-1

      !$omp parallel default(none),private(g,k,l), &
      !$omp shared(gall,kmin,kmax,lall,rhogw_vm,rhogvx,rhogvy,rhogvz,rhogw,VMTR_C2WfactGz,
          ↪ VMTR_RGSQRTH,VMTR_RGAMH)
      do l = 1, lall
         !$omp do
         do k = kmin+1, kmax
         do g = 1, gall
            rhogw_vm(g,k,l) = ( VMTR_C2WfactGz(g,k,1,l) * rhogvx(g,k  ,l) &
                              + VMTR_C2WfactGz(g,k,2,l) * rhogvx(g,k-1,l) &
                              + VMTR_C2WfactGz(g,k,3,l) * rhogvy(g,k  ,l) &
                              + VMTR_C2WfactGz(g,k,4,l) * rhogvy(g,k-1,l) &
                              + VMTR_C2WfactGz(g,k,5,l) * rhogvz(g,k  ,l) &
                              + VMTR_C2WfactGz(g,k,6,l) * rhogvz(g,k-1,l) &
                              ) * VMTR_RGAMH(g,k,l)                       & ! horizontal
                                  ↪ contribution
                            + rhogw(g,k,l) * VMTR_RGSQRTH(g,k,l)            ! vertical
                                ↪ contribution
         enddo
         enddo
         !$omp end do nowait

!OCL XFILL
         !$omp do
         do g = 1, gall
            rhogw_vm(g,kmin  ,l) = 0.0_RP
            rhogw_vm(g,kmax+1,l) = 0.0_RP
         enddo
         !$omp end do
      enddo
      !$omp end parallel

      !$omp parallel default(none),private(g,k,l,ij,ip1j,ip1jp1,ijp1,im1j,ijm1,im1jm1,
          ↪ sclt_rhogw), &
      !$omp shared(ADM_have_sgp,gminm1,gmin,gmax,gall,kmin,kmax,lall,iall,ddivdx,ddivdy,
          ↪ ddivdz,rhogvx,rhogvy,rhogvz, &
      !$omp        rhogvx_vm,rhogvy_vm,rhogvz_vm,rhogw_vm,sclt,coef_intp,coef_diff,GRD_rdgz,
          ↪ VMTR_RGAM)
      do l = 1, lall
         do k = kmin, kmax
!OCL XFILL
         !$omp do
         do g = 1, gall
            rhogvx_vm(g) = rhogvx(g,k,l) * VMTR_RGAM(g,k,l)
            rhogvy_vm(g) = rhogvy(g,k,l) * VMTR_RGAM(g,k,l)
            rhogvz_vm(g) = rhogvz(g,k,l) * VMTR_RGAM(g,k,l)
         enddo
         !$omp end do

         !$omp do
         do g = gminm1, gmax
            ij     = g
            ip1j   = g + 1
            ip1jp1 = g + iall + 1
            ijp1   = g + iall

            sclt_rhogw = ( ( rhogw_vm(ij,k+1,l) + rhogw_vm(ip1j,k+1,l) + rhogw_vm(ip1jp1,k
                ↪ +1,l) ) &
                       - ( rhogw_vm(ij,k  ,l) + rhogw_vm(ip1j,k  ,l) + rhogw_vm(ip1jp1,k
                           ↪   ,l) ) &
                       ) / 3.0_RP * GRD_rdgz(k)

            sclt(g,TI) = coef_intp(g,1,XDIR,TI,l) * rhogvx_vm(ij    ) &
                       + coef_intp(g,2,XDIR,TI,l) * rhogvx_vm(ip1j  ) &
                       + coef_intp(g,3,XDIR,TI,l) * rhogvx_vm(ip1jp1) &
                       + coef_intp(g,1,YDIR,TI,l) * rhogvy_vm(ij    ) &
                       + coef_intp(g,2,YDIR,TI,l) * rhogvy_vm(ip1j  ) &
                       + coef_intp(g,3,YDIR,TI,l) * rhogvy_vm(ip1jp1) &
                       + coef_intp(g,1,ZDIR,TI,l) * rhogvz_vm(ij    ) &
                       + coef_intp(g,2,ZDIR,TI,l) * rhogvz_vm(ip1j  ) &
                       + coef_intp(g,3,ZDIR,TI,l) * rhogvz_vm(ip1jp1) &
                       + sclt_rhogw
         enddo
         !$omp end do nowait
```

```
            !$omp do
            do g = gminm1, gmax
               ij     = g
               ip1j   = g + 1
               ip1jp1 = g + iall + 1
               ijp1   = g + iall

               sclt_rhogw = ( ( rhogw_vm(ij,k+1,l) + rhogw_vm(ip1jp1,k+1,l) + rhogw_vm(ijp1,k
                   ↪ +1,l) ) &
                            - ( rhogw_vm(ij,k  ,l) + rhogw_vm(ip1jp1,k  ,l) + rhogw_vm(ijp1,k
                               ↪   ,l) ) &
                            ) / 3.0_RP * GRD_rdgz(k)

               sclt(g,TJ) = coef_intp(g,1,XDIR,TJ,l) * rhogvx_vm(ij    ) &
                          + coef_intp(g,2,XDIR,TJ,l) * rhogvx_vm(ip1jp1) &
                          + coef_intp(g,3,XDIR,TJ,l) * rhogvx_vm(ijp1  ) &
                          + coef_intp(g,1,YDIR,TJ,l) * rhogvy_vm(ij    ) &
                          + coef_intp(g,2,YDIR,TJ,l) * rhogvy_vm(ip1jp1) &
                          + coef_intp(g,3,YDIR,TJ,l) * rhogvy_vm(ijp1  ) &
                          + coef_intp(g,1,ZDIR,TJ,l) * rhogvz_vm(ij    ) &
                          + coef_intp(g,2,ZDIR,TJ,l) * rhogvz_vm(ip1jp1) &
                          + coef_intp(g,3,ZDIR,TJ,l) * rhogvz_vm(ijp1  ) &
                          + sclt_rhogw
            enddo
            !$omp end do

            if ( ADM_have_sgp(l) ) then ! pentagon
               !$omp master
               sclt(gminm1,TI) = sclt(gminm1+1,TJ)
               !$omp end master
            endif

!OCL XFILL
            !$omp do
            do g = 1, gmin-1
               ddivdx(g,k,l) = 0.0_RP
               ddivdy(g,k,l) = 0.0_RP
               ddivdz(g,k,l) = 0.0_RP
            enddo
            !$omp end do nowait

            !$omp do
            do g = gmin, gmax
               ij     = g
               im1j   = g - 1
               im1jm1 = g - iall - 1
               ijm1   = g - iall

               ddivdx(g,k,l) = ( coef_diff(g,1,XDIR,l) * ( sclt(ij,    TI) + sclt(ij,    TJ)
                   ↪ ) &
                               + coef_diff(g,2,XDIR,l) * ( sclt(ij,    TJ) + sclt(im1j,  TI)
                                  ↪ ) &
                               + coef_diff(g,3,XDIR,l) * ( sclt(im1j,  TI) + sclt(im1jm1,TJ)
                                  ↪ ) &
                               + coef_diff(g,4,XDIR,l) * ( sclt(im1jm1,TJ) + sclt(im1jm1,TI)
                                  ↪ ) &
                               + coef_diff(g,5,XDIR,l) * ( sclt(im1jm1,TI) + sclt(ijm1  ,TJ)
                                  ↪ ) &
                               + coef_diff(g,6,XDIR,l) * ( sclt(ijm1,  TJ) + sclt(ij,    TI)
                                  ↪ ) )
            enddo
            !$omp end do nowait

            !$omp do
            do g = gmin, gmax
               ij     = g
               im1j   = g - 1
               im1jm1 = g - iall - 1
               ijm1   = g - iall

               ddivdy(g,k,l) = ( coef_diff(g,1,YDIR,l) * ( sclt(ij,    TI) + sclt(ij,    TJ)
                   ↪ ) &
                               + coef_diff(g,2,YDIR,l) * ( sclt(ij,    TJ) + sclt(im1j,  TI)
                                  ↪ ) &
```

```
                              + coef_diff(g,3,YDIR,l) * ( sclt(im1j,  TI) + sclt(im1jm1,TJ)
                                  ↪ ) &
                              + coef_diff(g,4,YDIR,l) * ( sclt(im1jm1,TJ) + sclt(im1jm1,TI)
                                  ↪ ) &
                              + coef_diff(g,5,YDIR,l) * ( sclt(im1jm1,TI) + sclt(ijm1  ,TJ)
                                  ↪ ) &
                              + coef_diff(g,6,YDIR,l) * ( sclt(ijm1,  TJ) + sclt(ij,    TI)
                                  ↪ ) )
          enddo
          !$omp end do nowait

          !$omp do
          do g = gmin, gmax
             ij     = g
             im1j   = g - 1
             im1jm1 = g - iall - 1
             ijm1   = g - iall

             ddivdz(g,k,l) = ( coef_diff(g,1,ZDIR,l) * ( sclt(ij,    TI) + sclt(ij,    TJ)
                 ↪ ) &
                              + coef_diff(g,2,ZDIR,l) * ( sclt(ij,    TJ) + sclt(im1j,  TI)
                                  ↪ ) &
                              + coef_diff(g,3,ZDIR,l) * ( sclt(im1j,  TI) + sclt(im1jm1,TJ)
                                  ↪ ) &
                              + coef_diff(g,4,ZDIR,l) * ( sclt(im1jm1,TJ) + sclt(im1jm1,TI)
                                  ↪ ) &
                              + coef_diff(g,5,ZDIR,l) * ( sclt(im1jm1,TI) + sclt(ijm1  ,TJ)
                                  ↪ ) &
                              + coef_diff(g,6,ZDIR,l) * ( sclt(ijm1,  TJ) + sclt(ij,    TI)
                                  ↪ ) )
          enddo
          !$omp end do nowait

!OCL XFILL
          !$omp do
          do g = gmax+1, gall
             ddivdx(g,k,l) = 0.0_RP
             ddivdy(g,k,l) = 0.0_RP
             ddivdz(g,k,l) = 0.0_RP
          enddo
          !$omp end do
       enddo ! loop k

!OCL XFILL
       !$omp do
       do g = 1, gall
          ddivdx(g,kmin-1,l) = 0.0_RP
          ddivdy(g,kmin-1,l) = 0.0_RP
          ddivdz(g,kmin-1,l) = 0.0_RP
          ddivdx(g,kmax+1,l) = 0.0_RP
          ddivdy(g,kmax+1,l) = 0.0_RP
          ddivdz(g,kmax+1,l) = 0.0_RP
       enddo
       !$omp end do
    enddo ! loop l
    !$omp end parallel

    if ( ADM_have_pl ) then
       n = ADM_gslf_pl

       do l = 1, ADM_lall_pl
          do k = ADM_kmin+1, ADM_kmax
          do g = 1, ADM_gall_pl
             rhogw_vm_pl(g,k,l) = ( VMTR_C2WfactGz_pl(g,k,1,l) * rhogvx_pl(g,k  ,l) &
                                  + VMTR_C2WfactGz_pl(g,k,2,l) * rhogvx_pl(g,k-1,l) &
                                  + VMTR_C2WfactGz_pl(g,k,3,l) * rhogvy_pl(g,k  ,l) &
                                  + VMTR_C2WfactGz_pl(g,k,4,l) * rhogvy_pl(g,k-1,l) &
                                  + VMTR_C2WfactGz_pl(g,k,5,l) * rhogvz_pl(g,k  ,l) &
                                  + VMTR_C2WfactGz_pl(g,k,6,l) * rhogvz_pl(g,k-1,l) &
                                  ) * VMTR_RGAMH_pl(g,k,l)                           & !
                                      ↪ horizontal contribution
                                + rhogw_pl(g,k,l) * VMTR_RGSQRTH_pl(g,k,l)             !
                                    ↪ vertical   contribution
          enddo
          enddo
```

```
         do g = 1, ADM_gall_pl
            rhogw_vm_pl(g,ADM_kmin  ,l) = 0.0_RP
            rhogw_vm_pl(g,ADM_kmax+1,l) = 0.0_RP
         enddo
      enddo

      do l = 1, ADM_lall_pl
         do k = ADM_kmin, ADM_kmax
            do v = 1, ADM_gall_pl
               rhogvx_vm_pl(v) = rhogvx_pl(v,k,l) * VMTR_RGAM_pl(v,k,l)
               rhogvy_vm_pl(v) = rhogvy_pl(v,k,l) * VMTR_RGAM_pl(v,k,l)
               rhogvz_vm_pl(v) = rhogvz_pl(v,k,l) * VMTR_RGAM_pl(v,k,l)
            enddo

            do v = ADM_gmin_pl, ADM_gmax_pl
               ij   = v
               ijp1 = v + 1
               if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl

               sclt_rhogw_pl = ( ( rhogw_vm_pl(n,k+1,l) + rhogw_vm_pl(ij,k+1,l) +
                   ↪ rhogw_vm_pl(ijp1,k+1,l) ) &
                             - ( rhogw_vm_pl(n,k  ,l) + rhogw_vm_pl(ij,k  ,l) +
                                 ↪ rhogw_vm_pl(ijp1,k  ,l) ) &
                             ) / 3.0_RP * GRD_rdgz(k)

               sclt_pl(ij) = coef_intp_pl(v,1,XDIR,l) * rhogvx_vm_pl(n   ) &
                           + coef_intp_pl(v,2,XDIR,l) * rhogvx_vm_pl(ij  ) &
                           + coef_intp_pl(v,3,XDIR,l) * rhogvx_vm_pl(ijp1) &
                           + coef_intp_pl(v,1,YDIR,l) * rhogvy_vm_pl(n   ) &
                           + coef_intp_pl(v,2,YDIR,l) * rhogvy_vm_pl(ij  ) &
                           + coef_intp_pl(v,3,YDIR,l) * rhogvy_vm_pl(ijp1) &
                           + coef_intp_pl(v,1,ZDIR,l) * rhogvz_vm_pl(n   ) &
                           + coef_intp_pl(v,2,ZDIR,l) * rhogvz_vm_pl(ij  ) &
                           + coef_intp_pl(v,3,ZDIR,l) * rhogvz_vm_pl(ijp1) &
                           + sclt_rhogw_pl
            enddo

            ddivdx_pl(:,k,l) = 0.0_RP
            ddivdy_pl(:,k,l) = 0.0_RP
            ddivdz_pl(:,k,l) = 0.0_RP

            do v = ADM_gmin_pl, ADM_gmax_pl
               ij   = v
               ijm1 = v - 1
               if( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl ! cyclic condition

               ddivdx_pl(n,k,l) = ddivdx_pl(n,k,l) + coef_diff_pl(v-1,XDIR,l) * ( sclt_pl(
                   ↪ ijm1) + sclt_pl(ij) )
               ddivdy_pl(n,k,l) = ddivdy_pl(n,k,l) + coef_diff_pl(v-1,YDIR,l) * ( sclt_pl(
                   ↪ ijm1) + sclt_pl(ij) )
               ddivdz_pl(n,k,l) = ddivdz_pl(n,k,l) + coef_diff_pl(v-1,ZDIR,l) * ( sclt_pl(
                   ↪ ijm1) + sclt_pl(ij) )
            enddo
         enddo

         ddivdx_pl(:,ADM_kmin-1,l) = 0.0_RP
         ddivdx_pl(:,ADM_kmax+1,l) = 0.0_RP
         ddivdy_pl(:,ADM_kmin-1,l) = 0.0_RP
         ddivdy_pl(:,ADM_kmax+1,l) = 0.0_RP
         ddivdz_pl(:,ADM_kmin-1,l) = 0.0_RP
         ddivdz_pl(:,ADM_kmax+1,l) = 0.0_RP
      enddo
   else
      ddivdx_pl(:,:,:) = 0.0_RP
      ddivdy_pl(:,:,:) = 0.0_RP
      ddivdz_pl(:,:,:) = 0.0_RP
   endif

   call DEBUG_rapend('OPRT3D_divdamp')

   return
end subroutine OPRT3D_divdamp
```

4.3.3 horizontal_flux

This kernel is based on the subroutine *horizontal_flux* from NICAM.

Listing 22: horizontal_flux

```
  subroutine horizontal_flux( &
       flx_h,  flx_h_pl,  &
       GRD_xc, GRD_xc_pl, &
       rho,    rho_pl,    &
       rhovx,  rhovx_pl,  &
       rhovy,  rhovy_pl,  &
       rhovz,  rhovz_pl,  &
       dt                 )
!ESC!    use mod_const, only: &
!ESC!       CONST_EPS
!ESC!    use mod_adm, only: &
!ESC!       ADM_have_pl,    &
!ESC!       ADM_have_sgp,   &
!ESC!       ADM_lall,       &
!ESC!       ADM_lall_pl,    &
!ESC!       ADM_gall,       &
!ESC!       ADM_gall_pl,    &
!ESC!       ADM_kall,       &
!ESC!       ADM_gall_1d,    &
!ESC!       ADM_gmin,       &
!ESC!       ADM_gmax,       &
!ESC!       ADM_gslf_pl,    &
!ESC!       ADM_gmin_pl,    &
!ESC!       ADM_gmax_pl
!ESC!    use mod_grd, only: &
!ESC!       GRD_xr,   &
!ESC!       GRD_xr_pl
!ESC!    use mod_gmtr, only: &
!ESC!       GMTR_p,    &
!ESC!       GMTR_p_pl, &
!ESC!       GMTR_t,    &
!ESC!       GMTR_t_pl, &
!ESC!       GMTR_a,    &
!ESC!       GMTR_a_pl
    implicit none

    real(RP), intent(out) :: flx_h    (ADM_gall   ,ADM_kall,ADM_lall   ,6)                !
        ↪ horizontal mass flux
    real(RP), intent(out) :: flx_h_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl  )
    real(RP), intent(out) :: GRD_xc   (ADM_gall   ,ADM_kall,ADM_lall   ,AI:AJ,XDIR:ZDIR) !
        ↪ mass centroid position
    real(RP), intent(out) :: GRD_xc_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,      XDIR:ZDIR)
    real(RP), intent(in)  :: rho      (ADM_gall   ,ADM_kall,ADM_lall   )                 !
        ↪ rho at cell center
    real(RP), intent(in)  :: rho_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhovx    (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)  :: rhovx_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhovy    (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)  :: rhovy_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: rhovz    (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)  :: rhovz_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)  :: dt

    real(RP) :: rhot_TI  (ADM_gall   ) ! rho at cell vertex
    real(RP) :: rhot_TJ  (ADM_gall   ) ! rho at cell vertex
    real(RP) :: rhot_pl  (ADM_gall_pl)
    real(RP) :: rhovxt_TI(ADM_gall   )
    real(RP) :: rhovxt_TJ(ADM_gall   )
    real(RP) :: rhovxt_pl(ADM_gall_pl)
    real(RP) :: rhovyt_TI(ADM_gall   )
    real(RP) :: rhovyt_TJ(ADM_gall   )
    real(RP) :: rhovyt_pl(ADM_gall_pl)
    real(RP) :: rhovzt_TI(ADM_gall   )
    real(RP) :: rhovzt_TJ(ADM_gall   )
    real(RP) :: rhovzt_pl(ADM_gall_pl)

    real(RP) :: rhovxt2
    real(RP) :: rhovyt2
    real(RP) :: rhovzt2
```

```
      real(RP) :: flux
      real(RP) :: rrhoa2

      integer  :: gmin, gmax, kall, iall
      real(RP) :: EPS

      integer  :: ij
      integer  :: ip1j, ijp1, ip1jp1
      integer  :: im1j, ijm1

      integer  :: i, j, k, l, n, v
      !-----------------------------------------------------------------------------

      call DEBUG_rapstart('____horizontal_adv_flux')

      gmin = ADM_gmin
      gmax = ADM_gmax
      kall = ADM_kall
      iall = ADM_gall_1d

      EPS  = CONST_EPS

      do l = 1, ADM_lall
         !$omp parallel default(none), &
         !$omp private(i,j,k,ij,ip1j,ip1jp1,ijp1,im1j,ijm1,
             ↪                                            &
         !$omp          rrhoa2,rhovxt2,rhovyt2,rhovzt2,flux),
             ↪                                           &
         !$omp shared(l,ADM_have_sgp,gmin,gmax,kall,iall,rho,rhovx,rhovy,rhovz,flx_h,dt,
             ↪            &
         !$omp         rhot_TI,rhovxt_TI,rhovyt_TI,rhovzt_TI,rhot_TJ,rhovxt_TJ,rhovyt_TJ,
             ↪ rhovzt_TJ, &
         !$omp         GRD_xc,GRD_xr,GMTR_p,GMTR_t,GMTR_a,EPS)
         do k = 1, kall

            ! (i,j),(i+1,j)
            !$omp do
            do j = gmin-1, gmax
            do i = gmin-1, gmax
               ij     = (j-1)*iall + i
               ip1j   = ij + 1

               rhot_TI  (ij) = rho  (ij  ,k,l) * GMTR_t(ij,K0,l,TI,W1) &
                             + rho  (ip1j,k,l) * GMTR_t(ij,K0,l,TI,W2)
               rhovxt_TI(ij) = rhovx(ij  ,k,l) * GMTR_t(ij,K0,l,TI,W1) &
                             + rhovx(ip1j,k,l) * GMTR_t(ij,K0,l,TI,W2)
               rhovyt_TI(ij) = rhovy(ij  ,k,l) * GMTR_t(ij,K0,l,TI,W1) &
                             + rhovy(ip1j,k,l) * GMTR_t(ij,K0,l,TI,W2)
               rhovzt_TI(ij) = rhovz(ij  ,k,l) * GMTR_t(ij,K0,l,TI,W1) &
                             + rhovz(ip1j,k,l) * GMTR_t(ij,K0,l,TI,W2)

               rhot_TJ  (ij) = rho  (ij  ,k,l) * GMTR_t(ij,K0,l,TJ,W1)
               rhovxt_TJ(ij) = rhovx(ij  ,k,l) * GMTR_t(ij,K0,l,TJ,W1)
               rhovyt_TJ(ij) = rhovy(ij  ,k,l) * GMTR_t(ij,K0,l,TJ,W1)
               rhovzt_TJ(ij) = rhovz(ij  ,k,l) * GMTR_t(ij,K0,l,TJ,W1)
            enddo
            enddo
            !$omp end do

            ! (i,j+1),(i+1,j+1)
            !$omp do
            do j = gmin-1, gmax
            do i = gmin-1, gmax
               ij     = (j-1)*iall + i
               ijp1   = ij + iall
               ip1jp1 = ij + iall + 1

               rhot_TI  (ij) = rhot_TI  (ij) + rho  (ip1jp1,k,l) * GMTR_t(ij,K0,l,TI,W3)
               rhovxt_TI(ij) = rhovxt_TI(ij) + rhovx(ip1jp1,k,l) * GMTR_t(ij,K0,l,TI,W3)
               rhovyt_TI(ij) = rhovyt_TI(ij) + rhovy(ip1jp1,k,l) * GMTR_t(ij,K0,l,TI,W3)
               rhovzt_TI(ij) = rhovzt_TI(ij) + rhovz(ip1jp1,k,l) * GMTR_t(ij,K0,l,TI,W3)

               rhot_TJ  (ij) = rhot_TJ  (ij) + rho  (ip1jp1,k,l) * GMTR_t(ij,K0,l,TJ,W2) &
                                             + rho  (ijp1  ,k,l) * GMTR_t(ij,K0,l,TJ,W3)
               rhovxt_TJ(ij) = rhovxt_TJ(ij) + rhovx(ip1jp1,k,l) * GMTR_t(ij,K0,l,TJ,W2) &
```

```
                                          + rhovx(ijp1   ,k,l) * GMTR_t(ij,K0,l,TJ,W3)
            rhovyt_TJ(ij) = rhovyt_TJ(ij) + rhovy(ip1jp1,k,l) * GMTR_t(ij,K0,l,TJ,W2) &
                                          + rhovy(ijp1   ,k,l) * GMTR_t(ij,K0,l,TJ,W3)
            rhovzt_TJ(ij) = rhovzt_TJ(ij) + rhovz(ip1jp1,k,l) * GMTR_t(ij,K0,l,TJ,W2) &
                                          + rhovz(ijp1   ,k,l) * GMTR_t(ij,K0,l,TJ,W3)
         enddo
         enddo
         !$omp end do

         if ( ADM_have_sgp(l) ) then
            !$omp master
            j = gmin-1
            i = gmin-1

            ij   = (j-1)*iall + i
            ip1j = ij + 1

            rhot_TI  (ij) = rhot_TJ  (ip1j)
            rhovxt_TI(ij) = rhovxt_TJ(ip1j)
            rhovyt_TI(ij) = rhovyt_TJ(ip1j)
            rhovzt_TI(ij) = rhovzt_TJ(ip1j)
            !$omp end master
         endif

         !--- calculate flux and mass centroid position

!OCL XFILL
         !$omp do
         do j = 1, iall
         do i = 1, iall
            if (      i < gmin .OR. i > gmax &
                 .OR. j < gmin .OR. j > gmax ) then
               ij = (j-1)*iall + i

               flx_h(ij,k,l,1) = 0.0_RP
               flx_h(ij,k,l,2) = 0.0_RP
               flx_h(ij,k,l,3) = 0.0_RP
               flx_h(ij,k,l,4) = 0.0_RP
               flx_h(ij,k,l,5) = 0.0_RP
               flx_h(ij,k,l,6) = 0.0_RP

               GRD_xc(ij,k,l,AI ,XDIR) = 0.0_RP
               GRD_xc(ij,k,l,AI ,YDIR) = 0.0_RP
               GRD_xc(ij,k,l,AI ,ZDIR) = 0.0_RP
               GRD_xc(ij,k,l,AIJ,XDIR) = 0.0_RP
               GRD_xc(ij,k,l,AIJ,YDIR) = 0.0_RP
               GRD_xc(ij,k,l,AIJ,ZDIR) = 0.0_RP
               GRD_xc(ij,k,l,AJ ,XDIR) = 0.0_RP
               GRD_xc(ij,k,l,AJ ,YDIR) = 0.0_RP
               GRD_xc(ij,k,l,AJ ,ZDIR) = 0.0_RP
            endif
         enddo
         enddo
         !$omp end do

         !$omp do
         do j = gmin  , gmax
         do i = gmin-1, gmax
            ij     = (j-1)*iall + i
            ip1j   = ij + 1
            ijm1   = ij - iall

            rrhoa2  = 1.0_RP / max( rhot_TJ(ijm1) + rhot_TI(ij), EPS ) ! doubled
            rhovxt2 = rhovxt_TJ(ijm1) + rhovxt_TI(ij)
            rhovyt2 = rhovyt_TJ(ijm1) + rhovyt_TI(ij)
            rhovzt2 = rhovzt_TJ(ijm1) + rhovzt_TI(ij)

            flux = 0.5_RP * ( rhovxt2 * GMTR_a(ij,k0,l,AI ,HNX) &
                            + rhovyt2 * GMTR_a(ij,k0,l,AI ,HNY) &
                            + rhovzt2 * GMTR_a(ij,k0,l,AI ,HNZ) )

            flx_h(ij   ,k,l,1) =  flux * GMTR_p(ij   ,k0,l,P_RAREA) * dt
            flx_h(ip1j,k,l,4) = -flux * GMTR_p(ip1j,k0,l,P_RAREA) * dt
```

```
      GRD_xc(ij,k,l,AI,XDIR) = GRD_xr(ij,K0,l,AI,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5
          ↪ _RP
      GRD_xc(ij,k,l,AI,YDIR) = GRD_xr(ij,K0,l,AI,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5
          ↪ _RP
      GRD_xc(ij,k,l,AI,ZDIR) = GRD_xr(ij,K0,l,AI,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5
          ↪ _RP
    enddo
    enddo
    !$omp end do

    !$omp do
    do j = gmin-1, gmax
    do i = gmin-1, gmax
      ij     = (j-1)*iall + i
      ip1jp1 = ij + iall + 1

      rrhoa2  = 1.0_RP / max( rhot_TI(ij) + rhot_TJ(ij), EPS ) ! doubled
      rhovxt2 = rhovxt_TI(ij) + rhovxt_TJ(ij)
      rhovyt2 = rhovyt_TI(ij) + rhovyt_TJ(ij)
      rhovzt2 = rhovzt_TI(ij) + rhovzt_TJ(ij)

      flux = 0.5_RP * ( rhovxt2 * GMTR_a(ij,k0,l,AIJ,HNX) &
                      + rhovyt2 * GMTR_a(ij,k0,l,AIJ,HNY) &
                      + rhovzt2 * GMTR_a(ij,k0,l,AIJ,HNZ) )

      flx_h(ij    ,k,l,2) =  flux * GMTR_p(ij    ,k0,l,P_RAREA) * dt
      flx_h(ip1jp1,k,l,5) = -flux * GMTR_p(ip1jp1,k0,l,P_RAREA) * dt

      GRD_xc(ij,k,l,AIJ,XDIR) = GRD_xr(ij,K0,l,AIJ,XDIR) - rhovxt2 * rrhoa2 * dt *
          ↪ 0.5_RP
      GRD_xc(ij,k,l,AIJ,YDIR) = GRD_xr(ij,K0,l,AIJ,YDIR) - rhovyt2 * rrhoa2 * dt *
          ↪ 0.5_RP
      GRD_xc(ij,k,l,AIJ,ZDIR) = GRD_xr(ij,K0,l,AIJ,ZDIR) - rhovzt2 * rrhoa2 * dt *
          ↪ 0.5_RP
    enddo
    enddo
    !$omp end do

    !$omp do
    do j = gmin-1, gmax
    do i = gmin  , gmax
      ij     = (j-1)*iall + i
      ijp1   = ij + iall
      im1j   = ij - 1

      rrhoa2  = 1.0_RP / max( rhot_TJ(ij) + rhot_TI(im1j), EPS ) ! doubled
      rhovxt2 = rhovxt_TJ(ij) + rhovxt_TI(im1j)
      rhovyt2 = rhovyt_TJ(ij) + rhovyt_TI(im1j)
      rhovzt2 = rhovzt_TJ(ij) + rhovzt_TI(im1j)

      flux = 0.5_RP * ( rhovxt2 * GMTR_a(ij,k0,l,AJ ,HNX) &
                      + rhovyt2 * GMTR_a(ij,k0,l,AJ ,HNY) &
                      + rhovzt2 * GMTR_a(ij,k0,l,AJ ,HNZ) )

      flx_h(ij  ,k,l,3) =  flux * GMTR_p(ij  ,k0,l,P_RAREA) * dt
      flx_h(ijp1,k,l,6) = -flux * GMTR_p(ijp1,k0,l,P_RAREA) * dt

      GRD_xc(ij,k,l,AJ,XDIR) = GRD_xr(ij,K0,l,AJ,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5
          ↪ _RP
      GRD_xc(ij,k,l,AJ,YDIR) = GRD_xr(ij,K0,l,AJ,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5
          ↪ _RP
      GRD_xc(ij,k,l,AJ,ZDIR) = GRD_xr(ij,K0,l,AJ,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5
          ↪ _RP
    enddo
    enddo
    !$omp end do

    if ( ADM_have_sgp(l) ) then
      !$omp master
      j = gmin
      i = gmin

      ij = (j-1)*iall + i

      flx_h(ij,k,l,6) = 0.0_RP
```

```
            !$omp end master
         endif

      enddo
      !$omp end parallel
   enddo

   if ( ADM_have_pl ) then
      n = ADM_gslf_pl

      do l = 1, ADM_lall_pl
      do k = 1, ADM_kall

         do v = ADM_gmin_pl, ADM_gmax_pl
            ij   = v
            ijp1 = v + 1
            if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl

            rhot_pl  (v) = rho_pl  (n   ,k,l) * GMTR_t_pl(ij,K0,l,W1) &
                         + rho_pl  (ij  ,k,l) * GMTR_t_pl(ij,K0,l,W2) &
                         + rho_pl  (ijp1,k,l) * GMTR_t_pl(ij,K0,l,W3)
            rhovxt_pl(v) = rhovx_pl(n   ,k,l) * GMTR_t_pl(ij,K0,l,W1) &
                         + rhovx_pl(ij  ,k,l) * GMTR_t_pl(ij,K0,l,W2) &
                         + rhovx_pl(ijp1,k,l) * GMTR_t_pl(ij,K0,l,W3)
            rhovyt_pl(v) = rhovy_pl(n   ,k,l) * GMTR_t_pl(ij,K0,l,W1) &
                         + rhovy_pl(ij  ,k,l) * GMTR_t_pl(ij,K0,l,W2) &
                         + rhovy_pl(ijp1,k,l) * GMTR_t_pl(ij,K0,l,W3)
            rhovzt_pl(v) = rhovz_pl(n   ,k,l) * GMTR_t_pl(ij,K0,l,W1) &
                         + rhovz_pl(ij  ,k,l) * GMTR_t_pl(ij,K0,l,W2) &
                         + rhovz_pl(ijp1,k,l) * GMTR_t_pl(ij,K0,l,W3)
         enddo

         do v = ADM_gmin_pl, ADM_gmax_pl
            ij   = v
            ijm1 = v - 1
            if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl

            rrhoa2  = 1.0_RP / max( rhot_pl(ijm1) + rhot_pl(ij), EPS ) ! doubled
            rhovxt2 = rhovxt_pl(ijm1) + rhovxt_pl(ij)
            rhovyt2 = rhovyt_pl(ijm1) + rhovyt_pl(ij)
            rhovzt2 = rhovzt_pl(ijm1) + rhovzt_pl(ij)

            flux = 0.5_RP * ( rhovxt2 * GMTR_a_pl(ij,K0,l,HNX) &
                            + rhovyt2 * GMTR_a_pl(ij,K0,l,HNY) &
                            + rhovzt2 * GMTR_a_pl(ij,K0,l,HNZ) )

            flx_h_pl(v,k,l) = flux * GMTR_p_pl(n,K0,l,P_RAREA) * dt

            GRD_xc_pl(v,k,l,XDIR) = GRD_xr_pl(v,K0,l,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5
                ↪ _RP
            GRD_xc_pl(v,k,l,YDIR) = GRD_xr_pl(v,K0,l,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5
                ↪ _RP
            GRD_xc_pl(v,k,l,ZDIR) = GRD_xr_pl(v,K0,l,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5
                ↪ _RP
         enddo

      enddo
      enddo
   endif

   call DEBUG_rapend  ( '____horizontal_adv_flux' )

   return
end subroutine horizontal_flux
```

4.3.4 horizontal_limiter_thuburn

This kernel is based on the subroutine *horizontal_limiter_thuburn* from NICAM.

Listing 23: horizontal_limiter_thuburn

```
   subroutine horizontal_limiter_thuburn( &
        q_a,    q_a_pl,  &
```

```
        q,       q_pl,     &
        d,       d_pl,     &
        ch,      ch_pl,    &
        cmask,   cmask_pl, &
        Qout_prev, Qout_prev_pl, & ! KERNEL
        Qout_post, Qout_post_pl  ) ! KERNEL
!ESC!    use mod_const, only: &
!ESC!       CONST_HUGE, &
!ESC!       CONST_EPS
!ESC!    use mod_adm, only: &
!ESC!       ADM_have_pl,    &
!ESC!       ADM_have_sgp,   &
!ESC!       ADM_lall,       &
!ESC!       ADM_lall_pl,    &
!ESC!       ADM_gall,       &
!ESC!       ADM_gall_pl,    &
!ESC!       ADM_kall,       &
!ESC!       ADM_gall_1d,    &
!ESC!       ADM_gmin,       &
!ESC!       ADM_gmax,       &
!ESC!       ADM_gslf_pl,    &
!ESC!       ADM_gmin_pl,    &
!ESC!       ADM_gmax_pl
!ESC!    use mod_comm, only: &
!ESC!       COMM_data_transfer
!ESC!    implicit none

    real(RP), intent(inout) :: q_a     (ADM_gall   ,ADM_kall,ADM_lall   ,6)
    real(RP), intent(inout) :: q_a_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl  )
    real(RP), intent(in)    :: q       (ADM_gall   ,ADM_kall,ADM_lall    )
    real(RP), intent(in)    :: q_pl    (ADM_gall_pl,ADM_kall,ADM_lall_pl  )
    real(RP), intent(in)    :: d       (ADM_gall   ,ADM_kall,ADM_lall    )
    real(RP), intent(in)    :: d_pl    (ADM_gall_pl,ADM_kall,ADM_lall_pl  )
    real(RP), intent(in)    :: ch      (ADM_gall   ,ADM_kall,ADM_lall   ,6)
    real(RP), intent(in)    :: ch_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl  )
    real(RP), intent(in)    :: cmask   (ADM_gall   ,ADM_kall,ADM_lall   ,6)
    real(RP), intent(in)    :: cmask_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl  )
    real(RP), intent(out)   :: Qout_prev   (ADM_gall   ,ADM_kall,ADM_lall   ,2 ) ! before
        ↪ communication (for check)
    real(RP), intent(out)   :: Qout_prev_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,2 ) !
    real(RP), intent(in)    :: Qout_post   (ADM_gall   ,ADM_kall,ADM_lall   ,2 ) ! after
        ↪ communication (additional input)
    real(RP), intent(in)    :: Qout_post_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,2 ) !

    real(RP) :: q_min_AI, q_min_AIJ, q_min_AJ, q_min_pl
    real(RP) :: q_max_AI, q_max_AIJ, q_max_AJ, q_max_pl

    real(RP) :: qnext_min   , qnext_min_pl
    real(RP) :: qnext_max   , qnext_max_pl
    real(RP) :: Cin_sum     , Cin_sum_pl
    real(RP) :: Cout_sum    , Cout_sum_pl
    real(RP) :: CQin_max_sum, CQin_max_sum_pl
    real(RP) :: CQin_min_sum, CQin_min_sum_pl

    integer, parameter :: I_min = 1
    integer, parameter :: I_max = 2
    real(RP) :: Qin    (ADM_gall   ,ADM_kall,ADM_lall   ,2,6)
    real(RP) :: Qin_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl,2,2)
    real(RP) :: Qout   (ADM_gall   ,ADM_kall,ADM_lall   ,2  )
    real(RP) :: Qout_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,2  )

    real(RP) :: ch_masked1
    real(RP) :: ch_masked2
    real(RP) :: ch_masked3
    real(RP) :: ch_masked4
    real(RP) :: ch_masked5
    real(RP) :: ch_masked6
    real(RP) :: ch_masked
    real(RP) :: zerosw

    integer  :: gmin, gmax, kall, iall
    real(RP) :: EPS, BIG

    integer  :: ij
    integer  :: ip1j, ijp1, ip1jp1, ip2jp1
```

```
      integer  :: im1j, ijm1

      integer  :: i, j, k, l, n, v
      !-------------------------------------------------------------------------

      call DEBUG_rapstart( '___horizontal_adv_limiter' )

      gmin = ADM_gmin
      gmax = ADM_gmax
      kall = ADM_kall
      iall = ADM_gall_1d

      EPS  = CONST_EPS
      BIG  = CONST_HUGE

      do l = 1, ADM_lall
         !$omp parallel default(none), &
         !$omp private(i,j,k,ij,ip1j,ip1jp1,ijp1,im1j,ijm1,ip2jp1,                          &
         !$omp          q_min_AI,q_min_AIJ,q_min_AJ,q_max_AI,q_max_AIJ,q_max_AJ,zerosw,    &
         !$omp          ch_masked1,ch_masked2,ch_masked3,ch_masked4,ch_masked5,ch_masked6, &
         !$omp          qnext_min,qnext_max,Cin_sum,Cout_sum,CQin_min_sum,CQin_max_sum),   &
         !$omp shared(l,ADM_have_sgp,gmin,gmax,kall,iall,q,cmask,d,ch,Qin,Qout,EPS,BIG)
         do k = 1, kall
            !---< (i) define inflow bounds, eq.(32)&(33) >---
!OCL XFILL
            !$omp do
            do j = gmin-1, gmax
            do i = gmin-1, gmax
               ij     = (j-1)*iall + i
               ip1j   = ij + 1
               ip1jp1 = ij + iall + 1
               ijp1   = ij + iall
               im1j   = ij - 1
               ijm1   = ij - iall

               im1j   = max( im1j  , 1 )
               ijm1   = max( ijm1  , 1 )

               q_min_AI  = min( q(ij,k,l), q(ijm1,k,l), q(ip1j,k,l), q(ip1jp1,k,l) )
               q_max_AI  = max( q(ij,k,l), q(ijm1,k,l), q(ip1j,k,l), q(ip1jp1,k,l) )
               q_min_AIJ = min( q(ij,k,l), q(ip1j,k,l), q(ip1jp1,k,l), q(ijp1,k,l) )
               q_max_AIJ = max( q(ij,k,l), q(ip1j,k,l), q(ip1jp1,k,l), q(ijp1,k,l) )
               q_min_AJ  = min( q(ij,k,l), q(ip1jp1,k,l), q(ijp1,k,l), q(im1j,k,l) )
               q_max_AJ  = max( q(ij,k,l), q(ip1jp1,k,l), q(ijp1,k,l), q(im1j,k,l) )

               Qin(ij,    k,l,I_min,1) = (        cmask(ij,k,l,1) ) * q_min_AI &
                                       + ( 1.0_RP-cmask(ij,k,l,1) ) * BIG
               Qin(ip1j,  k,l,I_min,4) = (        cmask(ij,k,l,1) ) * BIG      &
                                       + ( 1.0_RP-cmask(ij,k,l,1) ) * q_min_AI
               Qin(ij,    k,l,I_max,1) = (        cmask(ij,k,l,1) ) * q_max_AI &
                                       + ( 1.0_RP-cmask(ij,k,l,1) ) * (-BIG)
               Qin(ip1j,  k,l,I_max,4) = (        cmask(ij,k,l,1) ) * (-BIG)   &
                                       + ( 1.0_RP-cmask(ij,k,l,1) ) * q_max_AI

               Qin(ij,    k,l,I_min,2) = (        cmask(ij,k,l,2) ) * q_min_AIJ &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * BIG
               Qin(ip1jp1,k,l,I_min,5) = (        cmask(ij,k,l,2) ) * BIG       &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * q_min_AIJ
               Qin(ij,    k,l,I_max,2) = (        cmask(ij,k,l,2) ) * q_max_AIJ &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * (-BIG)
               Qin(ip1jp1,k,l,I_max,5) = (        cmask(ij,k,l,2) ) * (-BIG)    &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * q_max_AIJ

               Qin(ij,    k,l,I_min,3) = (        cmask(ij,k,l,3) ) * q_min_AJ &
                                       + ( 1.0_RP-cmask(ij,k,l,3) ) * BIG
               Qin(ijp1,  k,l,I_min,6) = (        cmask(ij,k,l,3) ) * BIG      &
                                       + ( 1.0_RP-cmask(ij,k,l,3) ) * q_min_AJ
               Qin(ij,    k,l,I_max,3) = (        cmask(ij,k,l,3) ) * q_max_AJ &
                                       + ( 1.0_RP-cmask(ij,k,l,3) ) * (-BIG)
               Qin(ijp1,  k,l,I_max,6) = (        cmask(ij,k,l,3) ) * (-BIG)   &
                                       + ( 1.0_RP-cmask(ij,k,l,3) ) * q_max_AJ
            enddo
            enddo
            !$omp end do
```

```
            if ( ADM_have_sgp(l) ) then
               !$omp master
               j = gmin-1
               i = gmin-1

               ij     = (j-1)*iall + i
               ijp1   = ij + iall
               ip1jp1 = ij + iall + 1
               ip2jp1 = ij + iall + 2

               q_min_AIJ = min( q(ij,k,l), q(ip1jp1,k,l), q(ip2jp1,k,l), q(ijp1,k,l) )
               q_max_AIJ = max( q(ij,k,l), q(ip1jp1,k,l), q(ip2jp1,k,l), q(ijp1,k,l) )

               Qin(ij,    k,l,I_min,2) = (        cmask(ij,k,l,2) ) * q_min_AIJ &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * BIG
               Qin(ip1jp1,k,l,I_min,5) = (        cmask(ij,k,l,2) ) * BIG       &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * q_min_AIJ
               Qin(ij,    k,l,I_max,2) = (        cmask(ij,k,l,2) ) * q_max_AIJ &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * (-BIG)
               Qin(ip1jp1,k,l,I_max,5) = (        cmask(ij,k,l,2) ) * (-BIG)    &
                                       + ( 1.0_RP-cmask(ij,k,l,2) ) * q_max_AIJ
               !$omp end master
            endif

            !---< (iii) define allowable range of q at next step, eq.(42)&(43) >---
!OCL XFILL
            !$omp do
            do j = gmin, gmax
            do i = gmin, gmax
               ij = (j-1)*iall + i

               qnext_min = min( q(ij,k,l),           &
                                Qin(ij,k,l,I_min,1), &
                                Qin(ij,k,l,I_min,2), &
                                Qin(ij,k,l,I_min,3), &
                                Qin(ij,k,l,I_min,4), &
                                Qin(ij,k,l,I_min,5), &
                                Qin(ij,k,l,I_min,6)  )

               qnext_max = max( q(ij,k,l),           &
                                Qin(ij,k,l,I_max,1), &
                                Qin(ij,k,l,I_max,2), &
                                Qin(ij,k,l,I_max,3), &
                                Qin(ij,k,l,I_max,4), &
                                Qin(ij,k,l,I_max,5), &
                                Qin(ij,k,l,I_max,6)  )

               ch_masked1 = min( ch(ij,k,l,1), 0.0_RP )
               ch_masked2 = min( ch(ij,k,l,2), 0.0_RP )
               ch_masked3 = min( ch(ij,k,l,3), 0.0_RP )
               ch_masked4 = min( ch(ij,k,l,4), 0.0_RP )
               ch_masked5 = min( ch(ij,k,l,5), 0.0_RP )
               ch_masked6 = min( ch(ij,k,l,6), 0.0_RP )

               Cin_sum      = ch_masked1 &
                            + ch_masked2 &
                            + ch_masked3 &
                            + ch_masked4 &
                            + ch_masked5 &
                            + ch_masked6

               Cout_sum     = ch(ij,k,l,1) - ch_masked1 &
                            + ch(ij,k,l,2) - ch_masked2 &
                            + ch(ij,k,l,3) - ch_masked3 &
                            + ch(ij,k,l,4) - ch_masked4 &
                            + ch(ij,k,l,5) - ch_masked5 &
                            + ch(ij,k,l,6) - ch_masked6

               CQin_min_sum = ch_masked1 * Qin(ij,k,l,I_min,1) &
                            + ch_masked2 * Qin(ij,k,l,I_min,2) &
                            + ch_masked3 * Qin(ij,k,l,I_min,3) &
                            + ch_masked4 * Qin(ij,k,l,I_min,4) &
                            + ch_masked5 * Qin(ij,k,l,I_min,5) &
                            + ch_masked6 * Qin(ij,k,l,I_min,6)
```

```
               CQin_max_sum = ch_masked1 * Qin(ij,k,l,I_max,1) &
                            + ch_masked2 * Qin(ij,k,l,I_max,2) &
                            + ch_masked3 * Qin(ij,k,l,I_max,3) &
                            + ch_masked4 * Qin(ij,k,l,I_max,4) &
                            + ch_masked5 * Qin(ij,k,l,I_max,5) &
                            + ch_masked6 * Qin(ij,k,l,I_max,6)

               zerosw = 0.5_RP - sign(0.5_RP,abs(Cout_sum)-EPS) ! if Cout_sum = 0, sw = 1

               Qout(ij,k,l,I_min) = ( q(ij,k,l) - CQin_max_sum - qnext_max*(1.0_RP-Cin_sum-
                   ↪ Cout_sum+d(ij,k,l)) ) &
                                  / ( Cout_sum + zerosw ) * ( 1.0_RP - zerosw )
                                      ↪                                          &
                                  + q(ij,k,l) * zerosw
               Qout(ij,k,l,I_max) = ( q(ij,k,l) - CQin_min_sum - qnext_min*(1.0_RP-Cin_sum-
                   ↪ Cout_sum+d(ij,k,l)) ) &
                                  / ( Cout_sum + zerosw ) * ( 1.0_RP - zerosw )
                                      ↪                                          &
                                  + q(ij,k,l) * zerosw
            enddo
            enddo
            !$omp end do

!OCL XFILL
            !$omp do
            do j = 1, iall
            do i = 1, iall
               if (      i < gmin .OR. i > gmax &
                    .OR. j < gmin .OR. j > gmax ) then
                  ij = (j-1)*iall + i

                  Qout(ij,k,l,I_min) = q(ij,k,l)
                  Qout(ij,k,l,I_min) = q(ij,k,l)
                  Qout(ij,k,l,I_max) = q(ij,k,l)
                  Qout(ij,k,l,I_max) = q(ij,k,l)
               endif
            enddo
            enddo
            !$omp end do

         enddo ! k loop
         !$omp end parallel
      enddo ! l loop

      if ( ADM_have_pl ) then
         n = ADM_gslf_pl

         do l = 1, ADM_lall_pl
            do k = 1, ADM_kall
               do v = ADM_gmin_pl, ADM_gmax_pl
                  ij   = v
                  ijp1 = v + 1
                  ijm1 = v - 1
                  if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
                  if( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl

                  q_min_pl = min( q_pl(n,k,l), q_pl(ij,k,l), q_pl(ijm1,k,l), q_pl(ijp1,k,l) )
                  q_max_pl = max( q_pl(n,k,l), q_pl(ij,k,l), q_pl(ijm1,k,l), q_pl(ijp1,k,l) )

                  Qin_pl(ij,k,l,I_min,1) = (         cmask_pl(ij,k,l) ) * q_min_pl &
                                         + ( 1.0_RP-cmask_pl(ij,k,l) ) * BIG
                  Qin_pl(ij,k,l,I_min,2) = (         cmask_pl(ij,k,l) ) * BIG      &
                                         + ( 1.0_RP-cmask_pl(ij,k,l) ) * q_min_pl
                  Qin_pl(ij,k,l,I_max,1) = (         cmask_pl(ij,k,l) ) * q_max_pl &
                                         + ( 1.0_RP-cmask_pl(ij,k,l) ) * (-BIG)
                  Qin_pl(ij,k,l,I_max,2) = (         cmask_pl(ij,k,l) ) * (-BIG)   &
                                         + ( 1.0_RP-cmask_pl(ij,k,l) ) * q_max_pl
               enddo

               qnext_min_pl = q_pl(n,k,l)
               qnext_max_pl = q_pl(n,k,l)
               do v = ADM_gmin_pl, ADM_gmax_pl
                  qnext_min_pl = min( qnext_min_pl, Qin_pl(v,k,l,I_min,1) )
                  qnext_max_pl = max( qnext_max_pl, Qin_pl(v,k,l,I_max,1) )
               enddo
```

```
            Cin_sum_pl      = 0.0_RP
            Cout_sum_pl     = 0.0_RP
            CQin_max_sum_pl = 0.0_RP
            CQin_min_sum_pl = 0.0_RP
            do v = ADM_gmin_pl, ADM_gmax_pl
               ch_masked = cmask_pl(v,k,l) * ch_pl(v,k,l)

               Cin_sum_pl      = Cin_sum_pl      + ch_masked
               Cout_sum_pl     = Cout_sum_pl     - ch_masked + ch_pl (v,k,l)
               CQin_min_sum_pl = CQin_min_sum_pl + ch_masked * Qin_pl(v,k,l,I_min,1)
               CQin_max_sum_pl = CQin_max_sum_pl + ch_masked * Qin_pl(v,k,l,I_max,1)
            enddo

            zerosw = 0.5_RP - sign(0.5_RP,abs(Cout_sum_pl)-EPS) ! if Cout_sum_pl = 0, sw =
                ↪  1

            Qout_pl(n,k,l,I_min) = ( q_pl(n,k,l) - CQin_max_sum_pl - qnext_max_pl*(1.0_RP-
                ↪ Cin_sum_pl-Cout_sum_pl+d_pl(n,k,l)) ) &
                                 / ( Cout_sum_pl + zerosw ) * ( 1.0_RP - zerosw )
                                   ↪                                                  &
                                 + q_pl(n,k,l) * zerosw
            Qout_pl(n,k,l,I_max) = ( q_pl(n,k,l) - CQin_min_sum_pl - qnext_min_pl*(1.0_RP-
                ↪ Cin_sum_pl-Cout_sum_pl+d_pl(n,k,l)) ) &
                                 / ( Cout_sum_pl + zerosw ) * ( 1.0_RP - zerosw )
                                   ↪                                                  &
                                 + q_pl(n,k,l) * zerosw

         enddo
      enddo
   endif

   !##########################################################################KERNEL
   call DEBUG_rapend  ('___horizontal_adv_limiter')
   Qout_pl(ADM_gmin_pl:ADM_gmax_pl,:,:,:) = 0.0_RP

   Qout_prev   (:,:,:,:) = Qout        (:,:,:,:)
   Qout_prev_pl(:,:,:,:) = Qout_pl     (:,:,:,:)
   !call COMM_data_transfer( Qout(:,:,:,:), Qout_pl(:,:,:,:) )
   Qout        (:,:,:,:) = Qout_post   (:,:,:,:)
   Qout_pl     (:,:,:,:) = Qout_post_pl(:,:,:,:)
   call DEBUG_rapstart('___horizontal_adv_limiter')
   !##########################################################################KERNEL

   !---- apply inflow/outflow limiter
   do l = 1, ADM_lall
      !$omp parallel do default(none),private(i,j,k,ij,ip1j,ip1jp1,ijp1), &
      !$omp shared(l,gmin,gmax,kall,iall,q_a,cmask,Qin,Qout)
      do k = 1, kall
         do j = gmin-1, gmax
         do i = gmin-1, gmax
            ij     = (j-1)*iall + i
            ip1j   = ij + 1
            ip1jp1 = ij + iall + 1
            ijp1   = ij + iall

            q_a(ij,k,l,1) = (        cmask(ij,k,l,1) ) * min( max( q_a(ij,k,l,1), Qin (ij
                ↪    ,k,l,I_min,1) ), Qin (ij    ,k,l,I_max,1) ) &
                          + ( 1.0_RP-cmask(ij,k,l,1) ) * min( max( q_a(ij,k,l,1), Qin (
                              ↪ ip1j  ,k,l,I_min,4) ), Qin (ip1j  ,k,l,I_max,4) )
            q_a(ij,k,l,1) = (        cmask(ij,k,l,1) ) * max( min( q_a(ij,k,l,1), Qout(
                ↪ ip1j  ,k,l,I_max  ) ), Qout(ip1j  ,k,l,I_min  ) ) &
                          + ( 1.0_RP-cmask(ij,k,l,1) ) * max( min( q_a(ij,k,l,1), Qout(ij
                              ↪    ,k,l,I_max  ) ), Qout(ij    ,k,l,I_min  ) )
            q_a(ip1j,k,l,4) = q_a(ij,k,l,1)

            q_a(ij,k,l,2) = (        cmask(ij,k,l,2) ) * min( max( q_a(ij,k,l,2), Qin (ij
                ↪    ,k,l,I_min,2) ), Qin (ij    ,k,l,I_max,2) ) &
                          + ( 1.0_RP-cmask(ij,k,l,2) ) * min( max( q_a(ij,k,l,2), Qin (
                              ↪ ip1jp1,k,l,I_min,5) ), Qin (ip1jp1,k,l,I_max,5) )
            q_a(ij,k,l,2) = (        cmask(ij,k,l,2) ) * max( min( q_a(ij,k,l,2), Qout(
                ↪ ip1jp1,k,l,I_max  ) ), Qout(ip1jp1,k,l,I_min  ) ) &
                          + ( 1.0_RP-cmask(ij,k,l,2) ) * max( min( q_a(ij,k,l,2), Qout(ij
                              ↪    ,k,l,I_max  ) ), Qout(ij    ,k,l,I_min  ) )
            q_a(ip1jp1,k,l,5) = q_a(ij,k,l,2)
```

```
            q_a(ij,k,l,3) = (        cmask(ij,k,l,3) ) * min( max( q_a(ij,k,l,3), Qin (ij
                ↪    ,k,l,I_min,3) ), Qin (ij    ,k,l,I_max,3) ) &
                          + ( 1.0_RP-cmask(ij,k,l,3) ) * min( max( q_a(ij,k,l,3), Qin (
                              ↪ ijp1  ,k,l,I_min,6) ), Qin (ijp1  ,k,l,I_max,6) )
            q_a(ij,k,l,3) = (        cmask(ij,k,l,3) ) * max( min( q_a(ij,k,l,3), Qout(
                ↪ ijp1  ,k,l,I_max  ) ), Qout(ijp1  ,k,l,I_min  ) ) &
                          + ( 1.0_RP-cmask(ij,k,l,3) ) * max( min( q_a(ij,k,l,3), Qout(ij
                              ↪    ,k,l,I_max  ) ), Qout(ij    ,k,l,I_min  ) )
            q_a(ijp1,k,l,6) = q_a(ij,k,l,3)
         enddo
         enddo
      enddo
      !$omp end parallel do
   enddo

   if ( ADM_have_pl ) then
      n = ADM_gslf_pl

      do l = 1, ADM_lall_pl
      do k = 1, ADM_kall
      do v = ADM_gmin_pl, ADM_gmax_pl
         q_a_pl(v,k,l) = (        cmask_pl(v,k,l) ) * min(max(q_a_pl(v,k,l), Qin_pl (v,k,l
             ↪ ,I_min,1)), Qin_pl (v,k,l,I_max,1)) &
                       + ( 1.0_RP-cmask_pl(v,k,l) ) * min(max(q_a_pl(v,k,l), Qin_pl (v,k,l
                           ↪ ,I_min,2)), Qin_pl (v,k,l,I_max,2))
         q_a_pl(v,k,l) = (        cmask_pl(v,k,l) ) * max(min(q_a_pl(v,k,l), Qout_pl(v,k,l
             ↪ ,I_max  )), Qout_pl(v,k,l,I_min  )) &
                       + ( 1.0_RP-cmask_pl(v,k,l) ) * max(min(q_a_pl(v,k,l), Qout_pl(n,k,l
                           ↪ ,I_max  )), Qout_pl(n,k,l,I_min  ))
      enddo
      enddo
      enddo
   endif

   call DEBUG_rapend  ('____horizontal_adv_limiter')

   return
 end subroutine horizontal_limiter_thuburn
```

4.3.5 vertical_limiter_thuburn

This kernel is based on the subroutine *vertical_limiter_thuburn* from NICAM.

Listing 24: vertical_limiter_thuburn

```
  subroutine vertical_limiter_thuburn( &
       q_h, q_h_pl, &
       q,   q_pl,   &
       d,   d_pl,   &
       ck,  ck_pl   )
!ESC!    use mod_const, only: &
!ESC!       CONST_HUGE, &
!ESC!       CONST_EPS
!ESC!    use mod_adm, only: &
!ESC!       ADM_have_pl, &
!ESC!       ADM_gall,    &
!ESC!       ADM_gall_pl, &
!ESC!       ADM_lall,    &
!ESC!       ADM_lall_pl, &
!ESC!       ADM_kall,    &
!ESC!       ADM_kmin,    &
!ESC!       ADM_kmax
    implicit none

    real(RP), intent(inout) :: q_h   (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(inout) :: q_h_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: q     (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)    :: q_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: d     (ADM_gall   ,ADM_kall,ADM_lall   )
    real(RP), intent(in)    :: d_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: ck    (ADM_gall   ,ADM_kall,ADM_lall   ,2)
    real(RP), intent(in)    :: ck_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl,2)
```

```
      real(RP) :: Qout_min_k
      real(RP) :: Qout_max_k
      real(RP) :: Qout_min_km1(ADM_gall)
      real(RP) :: Qout_max_km1(ADM_gall)
      real(RP) :: Qout_min_pl(ADM_gall_pl,ADM_kall)
      real(RP) :: Qout_max_pl(ADM_gall_pl,ADM_kall)

      real(RP) :: Qin_minL, Qin_maxL
      real(RP) :: Qin_minU, Qin_maxU
      real(RP) :: qnext_min, qnext_max
      real(RP) :: Cin, Cout
      real(RP) :: CQin_min, CQin_max
      real(RP) :: inflagL, inflagU
      real(RP) :: zerosw

      integer  :: gall, kmin, kmax
      real(RP) :: EPS, BIG

      integer  :: g, k, l
      !-----------------------------------------------------------------------------

      call DEBUG_rapstart('____vertical_adv_limiter')

      gall = ADM_gall
      kmin = ADM_kmin
      kmax = ADM_kmax

      EPS  = CONST_EPS
      BIG  = CONST_HUGE

      do l = 1, ADM_lall
         !$omp parallel default(none), &
         !$omp private(g,k,zerosw,inflagL,inflagU,Qin_minL,Qin_minU,Qin_maxL,Qin_maxU, &
         !$omp         qnext_min,qnext_max,Cin,Cout,CQin_min,CQin_max,Qout_min_k,Qout_max_k),
            ↪    &
         !$omp shared(l,gall,kmin,kmax,q_h,ck,q,d,Qout_min_km1,Qout_max_km1,EPS,BIG)

!OCL XFILL
         !$omp do
         do g = 1, gall
            k = kmin ! peeling

            inflagL = 0.5_RP - sign(0.5_RP,ck(g,k  ,l,1)) ! incoming flux: flag=1
            inflagU = 0.5_RP + sign(0.5_RP,ck(g,k+1,l,1)) ! incoming flux: flag=1

            Qin_minL = min( q(g,k,l), q(g,k-1,l) ) + ( 1.0_RP-inflagL ) * BIG
            Qin_minU = min( q(g,k,l), q(g,k+1,l) ) + ( 1.0_RP-inflagU ) * BIG
            Qin_maxL = max( q(g,k,l), q(g,k-1,l) ) - ( 1.0_RP-inflagL ) * BIG
            Qin_maxU = max( q(g,k,l), q(g,k+1,l) ) - ( 1.0_RP-inflagU ) * BIG

            qnext_min = min( Qin_minL, Qin_minU, q(g,k,l) )
            qnext_max = max( Qin_maxL, Qin_maxU, q(g,k,l) )

            Cin      = (        inflagL ) * ck(g,k,l,1) &
                     + (        inflagU ) * ck(g,k,l,2)
            Cout     = ( 1.0_RP-inflagL ) * ck(g,k,l,1) &
                     + ( 1.0_RP-inflagU ) * ck(g,k,l,2)

            CQin_min = (        inflagL ) * ck(g,k,l,1) * Qin_minL &
                     + (        inflagU ) * ck(g,k,l,2) * Qin_minU
            CQin_max = (        inflagL ) * ck(g,k,l,1) * Qin_maxL &
                     + (        inflagU ) * ck(g,k,l,2) * Qin_maxU

            zerosw = 0.5_RP - sign(0.5_RP,abs(Cout)-EPS) ! if Cout = 0, sw = 1

            Qout_min_k = ( ( q(g,k,l) - qnext_max ) + qnext_max*(Cin+Cout-d(g,k,l)) -
                ↪ CQin_max ) &
                       / ( Cout + zerosw ) * ( 1.0_RP - zerosw )
                           ↪                                    &
                       + q(g,k,l) * zerosw
            Qout_max_k = ( ( q(g,k,l) - qnext_min ) + qnext_min*(Cin+Cout-d(g,k,l)) -
                ↪ CQin_min ) &
                       / ( Cout + zerosw ) * ( 1.0_RP - zerosw )
                           ↪                                    &
```

```
                        + q(g,k,l) * zerosw

            Qout_min_km1(g) = Qout_min_k
            Qout_max_km1(g) = Qout_max_k
         enddo
         !$omp end do

         do k = kmin+1, kmax
!OCL XFILL
         !$omp do
         do g = 1, gall
            inflagL = 0.5_RP - sign(0.5_RP,ck(g,k  ,l,1)) ! incoming flux: flag=1
            inflagU = 0.5_RP + sign(0.5_RP,ck(g,k+1,l,1)) ! incoming flux: flag=1

            Qin_minL = min( q(g,k,l), q(g,k-1,l) ) + ( 1.0_RP-inflagL ) * BIG
            Qin_minU = min( q(g,k,l), q(g,k+1,l) ) + ( 1.0_RP-inflagU ) * BIG
            Qin_maxL = max( q(g,k,l), q(g,k-1,l) ) - ( 1.0_RP-inflagL ) * BIG
            Qin_maxU = max( q(g,k,l), q(g,k+1,l) ) - ( 1.0_RP-inflagU ) * BIG

            qnext_min = min( Qin_minL, Qin_minU, q(g,k,l) )
            qnext_max = max( Qin_maxL, Qin_maxU, q(g,k,l) )

            Cin      = (        inflagL ) * ck(g,k,l,1) &
                     + (        inflagU ) * ck(g,k,l,2)
            Cout     = ( 1.0_RP-inflagL ) * ck(g,k,l,1) &
                     + ( 1.0_RP-inflagU ) * ck(g,k,l,2)

            CQin_min = (        inflagL ) * ck(g,k,l,1) * Qin_minL &
                     + (        inflagU ) * ck(g,k,l,2) * Qin_minU
            CQin_max = (        inflagL ) * ck(g,k,l,1) * Qin_maxL &
                     + (        inflagU ) * ck(g,k,l,2) * Qin_maxU

            zerosw = 0.5_RP - sign(0.5_RP,abs(Cout)-EPS) ! if Cout = 0, sw = 1

            Qout_min_k = ( ( q(g,k,l) - qnext_max ) + qnext_max*(Cin+Cout-d(g,k,l)) -
                ↪ CQin_max ) &
                       / ( Cout + zerosw ) * ( 1.0_RP - zerosw )
                           ↪                                    &
                       + q(g,k,l) * zerosw
            Qout_max_k = ( ( q(g,k,l) - qnext_min ) + qnext_min*(Cin+Cout-d(g,k,l)) -
                ↪ CQin_min ) &
                       / ( Cout + zerosw ) * ( 1.0_RP - zerosw )
                           ↪                                    &
                       + q(g,k,l) * zerosw

            q_h(g,k,l) = (        inflagL ) * max( min( q_h(g,k,l), Qout_max_km1(g) ),
                ↪ Qout_min_km1(g) ) &
                       + ( 1.0_RP-inflagL ) * max( min( q_h(g,k,l), Qout_max_k      ),
                           ↪ Qout_min_k      )

            Qout_min_km1(g) = Qout_min_k
            Qout_max_km1(g) = Qout_max_k
         enddo
         !$omp end do
         enddo

         !$omp end parallel
      enddo

      if ( ADM_have_pl ) then
         do l = 1, ADM_lall_pl

         do k = ADM_kmin, ADM_kmax
         do g = 1, ADM_gall_pl
            inflagL = 0.5_RP - sign(0.5_RP,ck_pl(g,k  ,l,1)) ! incoming flux: flag=1
            inflagU = 0.5_RP + sign(0.5_RP,ck_pl(g,k+1,l,1)) ! incoming flux: flag=1

            Qin_minL = min( q_pl(g,k,l), q_pl(g,k-1,l) ) + ( 1.0_RP-inflagL ) * BIG
            Qin_minU = min( q_pl(g,k,l), q_pl(g,k+1,l) ) + ( 1.0_RP-inflagU ) * BIG
            Qin_maxL = max( q_pl(g,k,l), q_pl(g,k-1,l) ) - ( 1.0_RP-inflagL ) * BIG
            Qin_maxU = max( q_pl(g,k,l), q_pl(g,k+1,l) ) - ( 1.0_RP-inflagU ) * BIG

            qnext_min = min( Qin_minL, Qin_minU, q_pl(g,k,l) )
            qnext_max = max( Qin_maxL, Qin_maxU, q_pl(g,k,l) )
```

```
                Cin      = (         inflagL ) * ( ck_pl(g,k,l,1) ) &
                         + (         inflagU ) * ( ck_pl(g,k,l,2) )
                Cout     = ( 1.0_RP-inflagL ) * ( ck_pl(g,k,l,1) ) &
                         + ( 1.0_RP-inflagU ) * ( ck_pl(g,k,l,2) )

                CQin_max = (         inflagL ) * ( ck_pl(g,k,l,1) * Qin_maxL ) &
                         + (         inflagU ) * ( ck_pl(g,k,l,2) * Qin_maxU )
                CQin_min = (         inflagL ) * ( ck_pl(g,k,l,1) * Qin_minL ) &
                         + (         inflagU ) * ( ck_pl(g,k,l,2) * Qin_minU )

                zerosw = 0.5_RP - sign(0.5_RP,abs(Cout)-EPS) ! if Cout = 0, sw = 1

                Qout_min_pl(g,k) = ( ( q_pl(g,k,l) - qnext_max ) + qnext_max*(Cin+Cout-d_pl(g,
                    ↪ k,l)) - CQin_max ) &
                                 / ( Cout + zerosw ) * ( 1.0_RP - zerosw )
                                   ↪                                          &
                                 + q_pl(g,k,l) * zerosw
                Qout_max_pl(g,k) = ( ( q_pl(g,k,l) - qnext_min ) + qnext_min*(Cin+Cout-d_pl(g,
                    ↪ k,l)) - CQin_min ) &
                                 / ( Cout + zerosw ) * ( 1.0_RP - zerosw )
                                   ↪                                          &
                                 + q_pl(g,k,l) * zerosw
             enddo
             enddo

             do k = ADM_kmin+1, ADM_kmax
             do g = 1, ADM_gall_pl
                inflagL = 0.5_RP - sign(0.5_RP,ck_pl(g,k,l,1)) ! incoming flux: flag=1

                q_h_pl(g,k,l) = (         inflagL ) * max( min( q_h_pl(g,k,l), Qout_max_pl(g,k
                    ↪ -1) ), Qout_min_pl(g,k-1) ) &
                              + ( 1.0_RP-inflagL ) * max( min( q_h_pl(g,k,l), Qout_max_pl(g,k
                                  ↪  ) ), Qout_min_pl(g,k  ) )
             enddo
             enddo

          enddo
       endif

       call DEBUG_rapend  ( '____vertical_adv_limiter' )

       return
     end subroutine vertical_limiter_thuburn
```

4.3.6 vi_rhow_solver

This kernel is based on the subroutine *vi_rhow_solver* from NICAM.

Listing 25: vi_rhow_solver

```
     subroutine vi_rhow_solver( &
          rhogw,  rhogw_pl,  &
          rhogw0, rhogw0_pl, &
          preg0,  preg0_pl,  &
          rhog0,  rhog0_pl,  &
          Srho,   Srho_pl,   &
          Sw,     Sw_pl,     &
          Spre,   Spre_pl,   &
          dt                 )
!ESC!    use mod_adm, only: &
!ESC!       ADM_have_pl, &
!ESC!       ADM_gall,    &
!ESC!       ADM_gall_pl, &
!ESC!       ADM_lall,    &
!ESC!       ADM_lall_pl, &
!ESC!       ADM_kall,    &
!ESC!       ADM_kmin,    &
!ESC!       ADM_kmax
!ESC!    use mod_const, only: &
!ESC!       CONST_GRAV, &
!ESC!       CONST_Rdry, &
!ESC!       CONST_CVdry
!ESC!    use mod_grd, only: &
```

```
!ESC!       GRD_rdgzh, &
!ESC!       GRD_afact, &
!ESC!       GRD_bfact
!ESC!    use mod_vmtr, only: &
!ESC!       VMTR_GSGAM2H,     &
!ESC!       VMTR_GSGAM2H_pl,  &
!ESC!       VMTR_RGAM,        &
!ESC!       VMTR_RGAM_pl,     &
!ESC!       VMTR_RGAMH,       &
!ESC!       VMTR_RGAMH_pl,    &
!ESC!       VMTR_RGSGAM2,     &
!ESC!       VMTR_RGSGAM2_pl,  &
!ESC!       VMTR_RGSGAM2H,    &
!ESC!       VMTR_RGSGAM2H_pl
!ESC!    use mod_runconf, only: &
!ESC!       NON_HYDRO_ALPHA
    !$ use omp_lib
    implicit none

    real(RP), intent(inout) :: rhogw    (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho*w
    ↪          ( G^1/2 x gam2 ), n+1
    real(RP), intent(inout) :: rhogw_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)

    real(RP), intent(in)    :: rhogw0   (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho*w
    ↪          ( G^1/2 x gam2 )
    real(RP), intent(in)    :: rhogw0_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: preg0    (ADM_gall   ,ADM_kall,ADM_lall   ) ! pressure prime
    ↪  ( G^1/2 x gam2 )
    real(RP), intent(in)    :: preg0_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: rhog0    (ADM_gall   ,ADM_kall,ADM_lall   ) ! rho
    ↪            ( G^1/2 x gam2 )
    real(RP), intent(in)    :: rhog0_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: Srho     (ADM_gall   ,ADM_kall,ADM_lall   ) ! source term
    ↪ for rho  at the full level
    real(RP), intent(in)    :: Srho_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: Sw       (ADM_gall   ,ADM_kall,ADM_lall   ) ! source term
    ↪ for rhow at the half level
    real(RP), intent(in)    :: Sw_pl    (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: Spre     (ADM_gall   ,ADM_kall,ADM_lall   ) ! source term
    ↪ for pres at the full level
    real(RP), intent(in)    :: Spre_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl)
    real(RP), intent(in)    :: dt

    real(RP) :: Sall    (ADM_gall,   ADM_kall)
    real(RP) :: Sall_pl (ADM_gall_pl,ADM_kall)
    real(RP) :: beta    (ADM_gall   )
    real(RP) :: beta_pl (ADM_gall_pl)
    real(RP) :: gamma   (ADM_gall,   ADM_kall)
    real(RP) :: gamma_pl(ADM_gall_pl,ADM_kall)

    integer  :: gall, kmin, kmax, lall
    real(RP) :: grav
    real(RP) :: CVovRt2 ! Cv / R / dt**2
    real(RP) :: alpha

    integer  :: g, k, l
    integer  :: gstr, gend
    !$ integer  :: n_per_thread
    !$ integer  :: n_thread
    !---------------------------------------------------------------------------

    call DEBUG_rapstart( '____vi_rhow_solver' )

    gall = ADM_gall
    kmin = ADM_kmin
    kmax = ADM_kmax
    lall = ADM_lall

    grav    = CONST_GRAV
    CVovRt2 = CONST_CVdry / CONST_Rdry / (dt*dt)
    alpha   = real(NON_HYDRO_ALPHA,kind=RP)

    !$omp parallel default(none),private(g,k,l), &
    !$omp private(gstr,gend,n_thread,n_per_thread) &
```

```
!$omp shared(gall,kmin,kmax,lall,rhogw,rhogw0,preg0,rhog0,Srho,Sw,Spre,dt,Sall,beta,
    ↪ gamma,Mu,Mc,Ml, &
!$omp         GRD_afact,GRD_bfact,GRD_rdgzh,VMTR_GSGAM2H,VMTR_RGAM,VMTR_RGAMH,
    ↪ VMTR_RGSGAM2,VMTR_RGSGAM2H,grav,alpha,CVovRt2)
gstr = 1
gend = gall
!$ n_thread      = omp_get_num_threads()
!$ n_per_thread = gall / n_thread + int( 0.5_RP + sign(0.5_RP,mod(gall,n_thread)-0.5_RP
    ↪ ) )
!$ gstr          = n_per_thread * omp_get_thread_num() + 1
!$ gend          = min( gstr+n_per_thread-1, gall )

do l = 1, lall
   ! calc Sall
   do k = kmin+1, kmax
   do g = gstr, gend
      Sall(g,k) = (   ( rhogw0(g,k,  l)*alpha + dt * Sw  (g,k,  l) ) * VMTR_RGAMH  (g,k
          ↪ ,  l)**2             &
                  - ( ( preg0 (g,k,  l)       + dt * Spre(g,k,  l) ) * VMTR_RGSGAM2(g,k
                      ↪ ,  l)                &
                    - ( preg0 (g,k-1,l)       + dt * Spre(g,k-1,l) ) * VMTR_RGSGAM2(g,k
                        ↪ -1,l)                &
                    ) * dt * GRD_rdgzh(k)
                        ↪
                        ↪ &
                  - ( ( rhog0 (g,k,  l)       + dt * Srho(g,k,  l) ) * VMTR_RGAM(g,k,
                      ↪ l)**2 * GRD_afact(k) &
                    + ( rhog0 (g,k-1,l)       + dt * Srho(g,k-1,l) ) * VMTR_RGAM(g,k-1,
                        ↪ l)**2 * GRD_bfact(k) &
                    ) * dt * grav
                        ↪
                        ↪ &
                  ) * CVovRt2
   enddo
   enddo

   ! boundary conditions
   do g = gstr, gend
      rhogw(g,kmin,  l) = rhogw(g,kmin,  l) * VMTR_RGSGAM2H(g,kmin,  l)
      rhogw(g,kmax+1,l) = rhogw(g,kmax+1,l) * VMTR_RGSGAM2H(g,kmax+1,l)
      Sall (g,kmin+1)   = Sall (g,kmin+1) - Ml(g,kmin+1,l) * rhogw(g,kmin,  l)
      Sall (g,kmax  )   = Sall (g,kmax  ) - Mu(g,kmax,  l) * rhogw(g,kmax+1,l)
   enddo

   !---< solve tri-daigonal matrix >

   ! condition at kmin+1
   k = kmin+1
   do g = gstr, gend
      beta (g)     = Mc(g,k,l)
      rhogw(g,k,l) = Sall(g,k) / beta(g)
   enddo

   ! forward
   do k = kmin+2, kmax
   do g = gstr, gend
      gamma(g,k)   = Mu(g,k-1,l) / beta(g)
      beta (g)     = Mc(g,k,l) - Ml(g,k,l) * gamma(g,k) ! update beta
      rhogw(g,k,l) = ( Sall(g,k) - Ml(g,k,l) * rhogw(g,k-1,l) ) / beta(g)
   enddo
   enddo

   ! backward
   do k = kmax-1, kmin+1, -1
   do g = gstr, gend
      rhogw(g,k  ,l) = rhogw(g,k  ,l) - gamma(g,k+1) * rhogw(g,k+1,l)
      rhogw(g,k+1,l) = rhogw(g,k+1,l) * VMTR_GSGAM2H(g,k+1,l) ! return value ( G^1/2 x
          ↪ gam2 )
   enddo
   enddo

   ! boundary treatment
   do g = gstr, gend
      rhogw(g,kmin  ,l) = rhogw(g,kmin  ,l) * VMTR_GSGAM2H(g,kmin  ,l)
      rhogw(g,kmin+1,l) = rhogw(g,kmin+1,l) * VMTR_GSGAM2H(g,kmin+1,l)
```

```
            rhogw(g,kmax+1,l) = rhogw(g,kmax+1,l) * VMTR_GSGAM2H(g,kmax+1,l)
         enddo
      enddo
      !$omp end parallel

      if ( ADM_have_pl ) then
         do l = 1, ADM_lall_pl
            do k  = ADM_kmin+1, ADM_kmax
            do g = 1, ADM_gall_pl
               Sall_pl(g,k) = (   ( rhogw0_pl(g,k,  l)*alpha + dt * Sw_pl  (g,k,  l) ) *
                   ↪ VMTR_RGAMH_pl  (g,k,  l)**2              &
                             - ( ( preg0_pl (g,k,  l)       + dt * Spre_pl(g,k,  l) ) *
                                 ↪ VMTR_RGSGAM2_pl(g,k,  l)                &
                               - ( preg0_pl (g,k-1,l)       + dt * Spre_pl(g,k-1,l) ) *
                                   ↪ VMTR_RGSGAM2_pl(g,k-1,l)                &
                               ) * dt * GRD_rdgzh(k)
                                 ↪
                                 ↪ &
                             - ( ( rhog0_pl (g,k,  l)       + dt * Srho_pl(g,k,  l) ) *
                                 ↪ VMTR_RGAM_pl(g,k,  l)**2 * GRD_afact(k) &
                               + ( rhog0_pl (g,k-1,l)       + dt * Srho_pl(g,k-1,l) ) *
                                   ↪ VMTR_RGAM_pl(g,k-1,l)**2 * GRD_bfact(k) &
                               ) * dt * grav
                                 ↪
                                 ↪ &
                             ) * CVovRt2
            enddo
            enddo

            do g = 1, ADM_gall_pl
               rhogw_pl(g,ADM_kmin,  l) = rhogw_pl(g,ADM_kmin,  l) * VMTR_RGSGAM2H_pl(g,
                   ↪ ADM_kmin,  l)
               rhogw_pl(g,ADM_kmax+1,l) = rhogw_pl(g,ADM_kmax+1,l) * VMTR_RGSGAM2H_pl(g,
                   ↪ ADM_kmax+1,l)
               Sall_pl (g,ADM_kmin+1)   = Sall_pl (g,ADM_kmin+1) - Ml_pl(g,ADM_kmin+1,l) *
                   ↪ rhogw_pl(g,ADM_kmin,  l)
               Sall_pl (g,ADM_kmax  )   = Sall_pl (g,ADM_kmax  ) - Mu_pl(g,ADM_kmax,  l) *
                   ↪ rhogw_pl(g,ADM_kmax+1,l)
            enddo

            k = ADM_kmin+1
            do g = 1, ADM_gall_pl
               beta_pl (g)     = Mc_pl(g,k,l)
               rhogw_pl(g,k,l) = Sall_pl(g,k) / beta_pl(g)
            enddo

            do k = ADM_kmin+2, ADM_kmax
            do g = 1, ADM_gall_pl
               gamma_pl(g,k)   = Mu_pl(g,k-1,l) / beta_pl(g)
               beta_pl (g)     = Mc_pl(g,k,l) - Ml_pl(g,k,l) * gamma_pl(g,k) ! update beta
               rhogw_pl(g,k,l) = ( Sall_pl(g,k) - Ml_pl(g,k,l) * rhogw_pl(g,k-1,l) ) /
                   ↪ beta_pl(g)
            enddo
            enddo

            ! backward
            do k = ADM_kmax-1, ADM_kmin+1, -1
            do g = 1, ADM_gall_pl
               rhogw_pl(g,k  ,l) = rhogw_pl(g,k  ,l) - gamma_pl(g,k+1) * rhogw_pl(g,k+1,l)
               rhogw_pl(g,k+1,l) = rhogw_pl(g,k+1,l) * VMTR_GSGAM2H_pl(g,k+1,l) ! return
                   ↪ value ( G^1/2 x gam2 )
            enddo
            enddo

            ! boundary treatment
            do g = 1, ADM_gall_pl
               rhogw_pl(g,ADM_kmin  ,l) = rhogw_pl(g,ADM_kmin  ,l) * VMTR_GSGAM2H_pl(g,
                   ↪ ADM_kmin  ,l)
               rhogw_pl(g,ADM_kmin+1,l) = rhogw_pl(g,ADM_kmin+1,l) * VMTR_GSGAM2H_pl(g,
                   ↪ ADM_kmin+1,l)
               rhogw_pl(g,ADM_kmax+1,l) = rhogw_pl(g,ADM_kmax+1,l) * VMTR_GSGAM2H_pl(g,
                   ↪ ADM_kmax+1,l)
            enddo
         enddo
      endif
```

```
    call DEBUG_rapend( '___vi_rhow_solver' )

    return
  end subroutine vi_rhow_solver
```

5 Summary and Conclusions

We have described in this document a testbed application that represents icosahedral modeling codes. It is a mini application that is ready to run with a reducded effort compared to running the real icosahedral modeling. A set of kernels grouped into three groups, as they were derived based on codes from three real icosahedral models, are shown. The differnt kernel groups allow to deal with different icosahedral grid structures, e.g. triangular and hexagonal tesselations, semi-structured and unstructured grids. Each group of kernels is delivered within the necessary data structures and code to run it.

Acknowledgement

This work was supported by the German Research Foundation (DFG) through the Priority Programme 1648 „Software for Exascale Computing“ (SPPEXA).