

IcoAtmosBenchmark
NICAM kernels

SPPEXA/AIMES Benchmarking team

May 17, 2018

Contents

1	Brief introduction of NICAM	5
1.1	NICAM is	5
1.2	Governing equations and the coordinate system	5
1.3	Time integration	6
1.4	Icosahedral grid and “glevel”	7
1.5	Upwind-Biased Advection Scheme	9
1.6	Vertical coordinate and discretization	10
1.7	Domain decomposition and “rlevel”	13
1.8	Horizontal data structure	13
1.9	Parallelization	14
1.10	Problem size	15
2	Description of each kernel	17
2.1	Overview and common stuff	17
2.1.1	Coding rule of <i>NICAM</i>	17
2.1.2	Data array for regular region	17
2.1.3	Data array for pole region	17
2.1.4	Kernelize	19
2.1.5	Problem size	20
2.1.6	MPI and OpenMP	20
2.1.7	Mesuring environment	20
2.2	<code>dyn_diffusion</code>	20
2.2.1	Description	20
2.2.2	Discretization and code	20
2.2.3	Input data and result	25
2.2.4	Sample of performance result	25
2.3	<code>dyn_divdamp</code>	25
2.3.1	Description	25
2.3.2	Discretization and code	25
2.3.3	Input data and result	30
2.3.4	Sample of performance result	31
2.4	<code>dyn_vi_rhow_solver</code>	31
2.4.1	Description	31
2.4.2	Discretization and code	31
2.4.3	Input data and result	34
2.4.4	Sample of performance result	34
2.5	<code>dyn_vert_adv_limiter</code>	34
2.5.1	Description	34
2.5.2	Discretization and code	35
2.5.3	Input data and result	38
2.5.4	Sample of performance result	38
2.6	<code>dyn_horiz_adv_flux</code>	38
2.6.1	Description	38

2.6.2	Discretization and code	38
2.6.3	Input data and result	43
2.6.4	Sample of performance result	43
2.7	dyn_horiz_adv_limiter	44
2.7.1	Description	44
2.7.2	Discretization and code	44
2.7.3	Input data and result	49
2.7.4	Sample of performance result	50
2.8	dyn_metrics	50
2.8.1	Description	50
2.8.2	Discretization and code	51
	(1) GMTR_p_setup	51
	(2) GMTR_t_setup	54
	(3) GMTR_a_setup	56
	(4) OPRT_divergence_setup	61
	(5) OPRT_rotation_setup	64
	(6) OPRT_gradient_setup	68
	(7) OPRT_laplacian_setup	71
	(8) OPRT_diffusion_setup	78
2.8.3	Input data and result	80
2.9	communication	81
2.9.1	Description	81
2.9.2	Discretization and code	82
	(1) COMM_setup	82
	(2) COMM_list_generate	83
	(3) COMM_sortdest	88
	(4) COMM_debugtest	93
2.9.3	Input data and result	98
2.9.4	Sample of performance result	98

Chapter 1

Brief introduction of NICAM

1.1 NICAM is

NICAM (Nonhydrostatic Atmospheric ICosahedral Model) is an global atmospheric model for the climate study, mainly developed by Japan Agency for Marine-Earth Science and Technology (JAMSTEC), Atmosphere And Ocean Research Institute (AORI) of U-Tokyo, and RIKEN. Since aiming high-resolution global atmospheric simulation, this model has several features, such as;

- cloud-system resolving approach,
- icosahedral grid system,
- targetting massively parallel supercomputer.

This manual describes the overview of *NICAM* and each kernel program briefly. For the details of *NICAM*, see Tomita and Satoh (2004), Satoh et al. (2008), Satoh et al. (2014), etc.

1.2 Governing equations and the coordinate system

The governing equations of *NICAM* are based on a fully compressible non-hydrostatic system, including acoustic waves, fast gravity waves and slow gravity waves. As a coordinate system, *NICAM* uses the Cartesian coordinate in 3-dimensional space. Introducing an orthogonal basis $\{e_1, e_2, e_3\}$, that is independent of space with e_3 being in the same direction as the angular velocity of the Earth, a 3-dimensional velocity v on the Earth's surface with reference to the bases e_1, e_2 and e_3 has the three components $\{v_1, v_2, v_3\}$. Furthermore, the velocity v is sometimes separated into a "horizontal element" and "vertical element" of that vector quantity. This treatment is useful in the horizontal-explicit-vertical-implicit scheme, which is used in *NICAM*. The wind component of radial direction is separated from v and named as a vertical wind w . The residual is horizontal wind and expressed as $v_h = \{v_x, v_y, v_z\}$.

The 3-dimensional Cartesian notations are used mainly in dynamical core of *NICAM*. In the physics package, more general wind velocity notations are used: the zonal wind u , meridional wind v , and vertical wind w . the same velocity v can be denoted as

$$v = u\hat{i} + v\hat{j} + w\hat{k} \quad (1.1)$$

where \hat{i} , \hat{j} , and \hat{k} are unit vectors in the longitudinal, latitudinal direction, and outward unit vector in the vertical direction at the given point on the sphere, respectively.

NICAM have a switch for the shallow-atmosphere approximation. The shallow-atmosphere approximation has inconsistency in the conservation of absolute angular momentum without several metric terms and the vertical Coriolis term. *NICAM* introduces a "deep" factor

$$\gamma \equiv r/a \quad (1.2)$$

where r is the distance from the center of the Planet, and a is the radius of the Planet at sea level. If we use the shallow-atmosphere approximation, γ is set to 1.

For vertical coordinate, *NICAM* employs a terrain-following coordinate with the following metrics:

$$\xi = \frac{\xi_T(z - z_s)}{z_T - z_s}, \quad G^{1/2} \equiv \left(\frac{\partial z}{\partial \xi}\right)_h, \quad \mathbf{G}^z \equiv \nabla_{h0} \xi = -\frac{\tilde{\nabla}_{h0} z}{G^{1/2}} \quad (1.3)$$

where z_T is the top of the model domain, z_s is the surface height, $(\partial/\partial \xi)_h$ denotes the derivative along the vertical direction, and $\tilde{\nabla}_{h0}$ denotes the spherical gradient operator along a constant ξ plane at sea level. Since $G^{1/2}\gamma^2$ is the factor of volume against the surface, *NICAM* treats the prognostic variable multiplied by this factor. For example, the perturbation density is $R = (\rho - \rho_{\text{ref}})G^{1/2}\gamma^2$, where subscript ref means the hydrostatic reference state, the horizontal and vertical momentum are $\mathbf{V}_h = \rho G^{1/2}\gamma^2 \mathbf{v}_h$ and $W = \rho G^{1/2}\gamma^2 w$, respectively, the internal energy is $E = \rho G^{1/2}\gamma^2 e$.

The governing equations are as follows:

$$\frac{\partial R}{\partial t} + \tilde{\nabla}_{h0} \cdot \frac{\mathbf{V}_h}{\gamma} + \frac{\partial}{\partial \xi} \left(\frac{\mathbf{V}_h}{\gamma} \cdot \mathbf{G}^z + \frac{W}{G^{1/2}} \right) = 0, \quad (1.4)$$

$$\frac{\partial \mathbf{V}_h}{\partial t} + \tilde{\nabla}_h \frac{P}{\gamma} + \frac{\partial}{\partial \xi} \left(\mathbf{G}^z \frac{P}{\gamma} \right) = -\tilde{\mathbf{A}}_h - \tilde{\mathbf{C}}_h, \quad (1.5)$$

$$\frac{\partial W}{\partial t} + \gamma^2 \frac{\partial}{\partial \xi} \left(\frac{P}{G^{1/2}\gamma^2} \right) + Rg = (-\tilde{A}_z - \tilde{C}_z), \quad (1.6)$$

$$\begin{aligned} & \frac{\partial E}{\partial t} + \tilde{\nabla}_{h0} \cdot \left(h \frac{\mathbf{V}_h}{\gamma} \right) + \frac{\partial}{\partial \xi} \left[h \left(\frac{\tilde{\mathbf{V}}_h}{\gamma} \cdot \mathbf{G}^z + \frac{W}{G^{1/2}} \right) \right] \\ & - \left[\mathbf{v}_h \cdot \left(\tilde{\nabla}_{h0} \frac{P}{\gamma} + \frac{\partial}{\partial \xi} \left(\mathbf{G}^z \frac{P}{\gamma} \right) \right) + w \left(\gamma^2 \frac{\partial}{\partial \xi} \left(\frac{P}{G^{1/2}\gamma^2} \right) + Rg \right) \right] + Wg = \tilde{Q}_{\text{heat}}, \end{aligned} \quad (1.7)$$

where $P = (p - p_{\text{ref}})G^{1/2}\gamma^2$ is the perturbation pressure, h is the enthalpy, g is the gravitational acceleration, and \tilde{Q}_{heat} is the heating rate. $\tilde{\mathbf{A}} (= \tilde{\mathbf{A}}_h + \tilde{A}_z \mathbf{k})$ and $\tilde{\mathbf{C}} (= \tilde{\mathbf{C}}_h + \tilde{C}_z \mathbf{k})$ are momentum advection term and Coriolis term, respectively. Using an orthogonal basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \}$, which is independent of space with \mathbf{e}_3 being in the same direction as the angular velocity of the Planet $(0, 0, \Omega)$. Similarly, the three-dimensional velocity \mathbf{v} is shown as (v_1, v_2, v_3) with regard to the basis $\mathbf{e}_1, \mathbf{e}_2$ and \mathbf{e}_3 , respectively. Using these $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{C}}$ can be expressed as

$$\tilde{\mathbf{A}} \equiv \sum_{i=1}^3 \left[\tilde{\nabla}_{h0} \cdot \left(v_i \frac{\mathbf{V}_h}{\gamma} \right) + \frac{\partial}{\partial \xi} \left[v_i \left(\frac{\mathbf{V}_h}{\gamma} \cdot \mathbf{G}^z + \frac{W}{G^{1/2}} \right) \right] \right] \mathbf{e}_i, \quad (1.8)$$

$$\tilde{\mathbf{C}} \equiv 2\Omega\rho G^{1/2}\gamma^2 (-v_2 \mathbf{e}_1 + v_1 \mathbf{e}_2). \quad (1.9)$$

1.3 Time integration

The set of governing equations that *NICAM* uses is an elastic system, so it contains all the waves, especially acoustic waves, high-frequency gravity waves and the Lamb waves. These are called as the ‘‘fast modes’’. The governing equations can be described schematically as

$$\frac{\partial \Psi}{\partial t} - F = S, \quad (1.10)$$

where Ψ , F and S represent the prognostic variable, the fast-mode term, and the slow-mode term, respectively. For small time step integration a forward-backward scheme based on the HEVI (horizontally explicit and vertically implicit) method is used, while for large time step integration the second- or third-stage Runge-Kutta method (Wicker and Skamarock, 2002) is used. This scheme treats fast waves implicitly in the vertical direction only, and Helmholtz equation is one-dimensional and can be solved directly as by the tridiagonal matrix solver (See section 1.6). The time step is restricted by the horizontal grid space, explicit scheme is

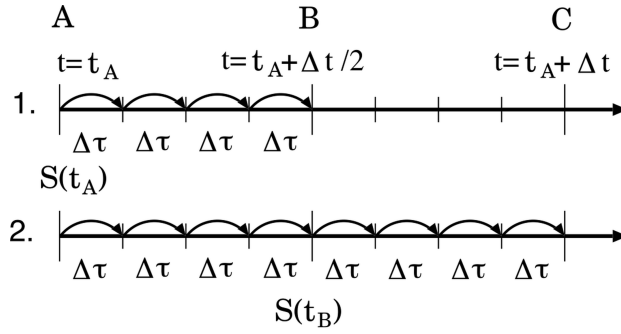


Figure 1.1: Schematic figure of temporal integration.

used in that direction. The fast-mode term are evaluated at every small time step $\Delta\tau$, and the slow-mode term are evaluated at larger time step Δt (Figure 1.1).

If Ψ at $t = t_A$ is given, the slow-mode tendency $S(t_A)$ can be evaluated. The variable is integrated from t_A to $t_B (> t_A)$ using $S(t_A) + F(t_A + m\Delta\tau)$ as the forcing function at $t = t_A + m\Delta\tau$, where $F(t_A + m\Delta\tau)$ is the fast-mode tendency being updated at every small step and m is the index of small time steps. Repeating integration with this small step, the temporary value of the prognostic variable Ψ^* at t_B can be obtained. The slow-mode tendency $S(t_B)$ is evaluated using this value. Finally, integrated variable Ψ at $t = t_C (> t_B)$ is calculated by $S(t_B) + F(t_A + m\Delta\tau)$.

1.4 Icosahedral grid and “glevel”

To overcome the pole problem which old-fashioned global models have, *NICAM* uses the icosahedral grid system. In this grid system, grid points are distributed quasi-homogeneously on the sphere. The icosahedral grid can be constructed by a recursive division method which is similar to that of [Stuhne and Peltier \(1996\)](#). The grid resolution obtained by i -th dividing operation is called “glevel- i ”. First, vertices of the spherical icosahedron construct glevel-0 (Figure 1.2a). By connecting the midpoint of each edge of the triangles, the next finer grid glevel-1 is generated as Figure 1.2b. Doing the same procedure, glevel-2 is generated(Figure 1.2c). For example, Figure 1.3 shows the glevel-5 grid.

All the variables are defined at the vertices of triangular grid elements. This arrangement is categorized into the Arakawa-A type grid. The control volume is defined as the polygon constructed by connecting the gravitational centers of neighboring triangular grid elements. The shape of control volume is hexagon, as shown in Figure 1.4, except that it is pentagon at only twelve points inherited from the original icosahedron.

NICAM adopts the finite volume method for the discretization of differential operators. In Figure 1.4,

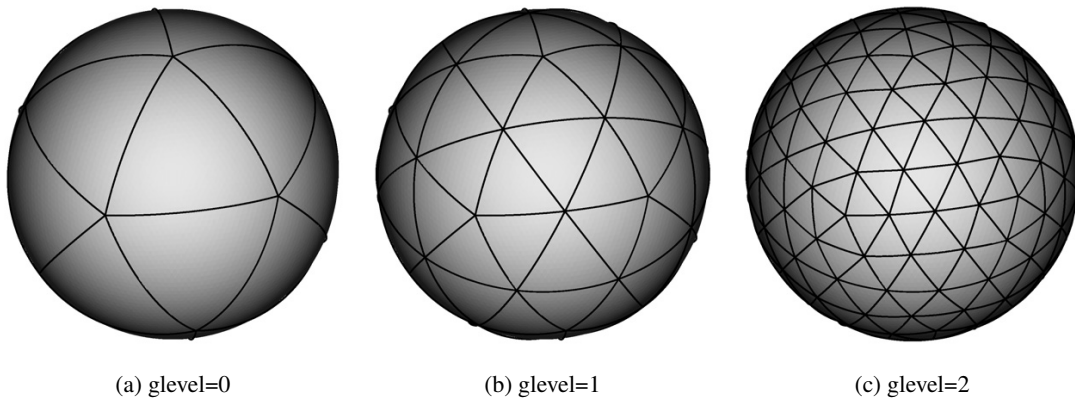


Figure 1.2: The method of horizontal grid refinement.

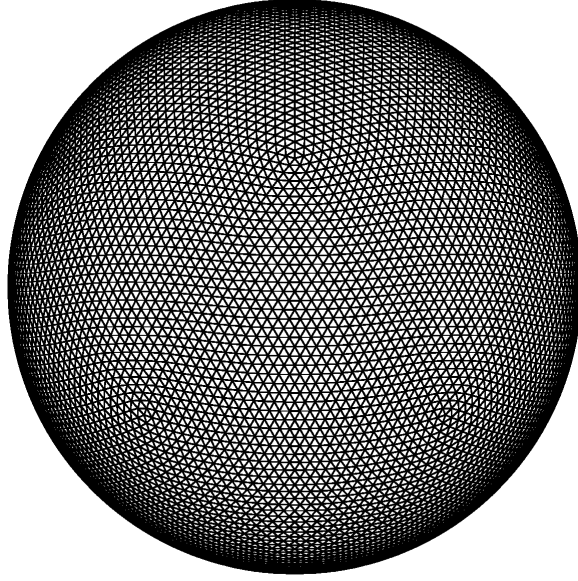


Figure 1.3: The icosahedral grid with glevel-5

some vector \mathbf{u} at the vertices of control volume Q_i are calculated by interpolation as^{*1)}

$$\mathbf{u}(Q_i) \simeq \frac{\alpha \mathbf{u}(P_0) + \beta \mathbf{u}(P_i) + \gamma \mathbf{u}(P_{i+1})}{\alpha + \beta + \gamma}, \quad (1.11)$$

where α , β and γ are the areas of the triangle $Q_i P_i P_{i+1}$, $Q_0 P_{i+1} P_0$ and $Q_i P_0 P_i$, respectively. The number 6 is replaced with 5 at the pentagonal control volumes for the singular points.

The divergence is calculated from the Gauss theorem as

$$\nabla \cdot \mathbf{u}(P_0) \simeq \frac{1}{a(P_0)} \sum_{i=1}^6 b_i \frac{\mathbf{u}(Q_i) + \mathbf{u}(Q_{i+1})}{2} \cdot \mathbf{n}_i, \quad (1.12)$$

where b_i and \mathbf{n}_i denote the geodesic arc length of $Q_i Q_{i+1}$ and the outward unit vector normal to this arc at the midpoint of $Q_i Q_{i+1}$. $a(P_0)$ is the area of control volume at the point P_0 . More than that, Equation 1.12 can be rewritten as a linear combination:

$$\nabla \cdot \mathbf{u}(P_0) = \sum_{i=1}^6 \mathbf{c}_i \cdot \mathbf{u}(P_i) \quad (1.13)$$

where \mathbf{c}_i are constant vector coefficients which can be pre-calculated once and stored for subsequent use. In Equation 1.13, the number of addition and multiplication is the same, this formulation is preferable for the CPU which has FMA.

The gradient operator to an arbitrary variable ϕ is calculated as

$$\nabla \phi(P_0) \simeq \frac{1}{a(P_0)} \sum_{i=1}^6 b_i \frac{\phi(Q_i) + \phi(Q_{i+1})}{2} \mathbf{n}_i - \frac{\phi_0}{a(P_0)} \sum_{i=1}^6 b_i \mathbf{n}_i, \quad (1.14)$$

where $\phi(Q_i)$ is interpolated by the similar way to the Equation 1.11. This scheme given as Equation 1.14 generates the vertical component because the allocation points are on the sphere. In practice, the vertical component is removed after the operation of Equation 1.14.

More precisely, *NICAM* used modified icosahedral grid by using spring dynamics in order to avoid severe problems on the numerical accuracy and stability. See Tomita et al. (2001) for the details.

^{*1)} P_{i+1} should be $P_{1+\text{mod}(i,6)}$, and the same hereinafter.

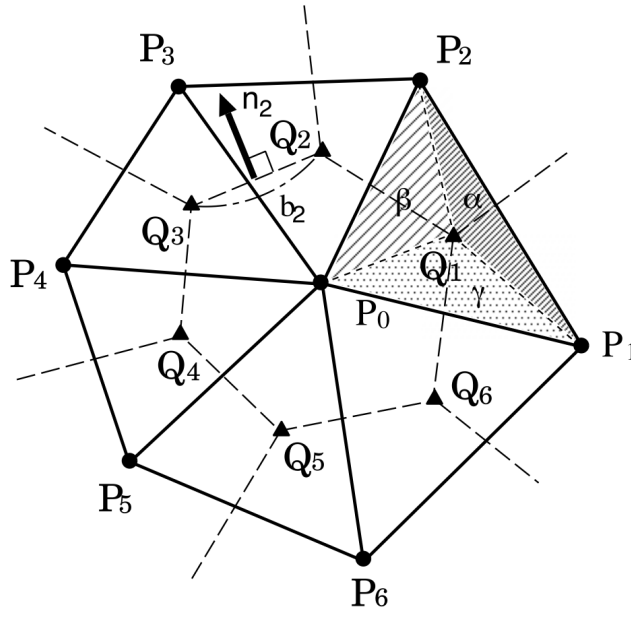


Figure 1.4: Schematic figure of control volume. (Tomita et al., 2008)

1.5 Upwind-Biased Advection Scheme

NICAM uses an upwind-biased advection scheme for icosahedral grid derived by Miura (2007). Figure 1.5 (a) shows arrangement of distorted hexagonal cells. Computational nodes are P_i ($i = 0, 2, \dots, 6$), hexagonal cell corners are Q_i , center of hexagonal cell faces are R_i , lengths of hexagonal cell faces are l_i , and unit vectors normal to cell faces are n_i . Figure 1.5 (b) shows schematic of the area swept by the arc $Q'_i Q'_{i+1}$ during a time interval. Here we assume that the arc $Q'_i Q'_{i+1}$ at time t moves with a constant velocity $v_{R_i}^{t+\Delta t/2}$ and coincides with the arc $Q_i Q_{i+1}$ at time $t + \Delta t$. The total amount of flux through the edge $Q_i Q_{i+1}$ during the time interval Δt is approximated by the amount of a tracer inside the parallelogram $Q'_i Q'_{i+1} Q_{i+1} Q_i$. Profiles of ρ and q , the density and the tracer mixing ratio, respectively, inside this parallelogram can be approximated by two-dimensional linear surface. Then the amount of a tracer inside this parallelogram is

$$\rho_{R_i} q_{R_i} = \frac{\int_{S_i} \rho q \, dS}{S_i} = \rho_{C_i} q_{C_i}, \quad (1.15)$$

where S_i and C_i denotes the area and the mass centroid of this parallelogram, respectively. Similarly, the continuity equation will yield

$$\rho_{R_i} = \rho_{C_i}. \quad (1.16)$$

Thus,

$$q_{R_i} = q_{C_i}. \quad (1.17)$$

The position of C_i can be computed as

$$C_i = R_i - v_{R_i}^{t+\Delta t/2} \frac{\Delta t}{2}. \quad (1.18)$$

Linear surfaces inside the parallelogram can be approximated by nodal values and gradients at a computational node that shares the cell face and is on the upwind side of the cell face. See Miura (2007) for more details.

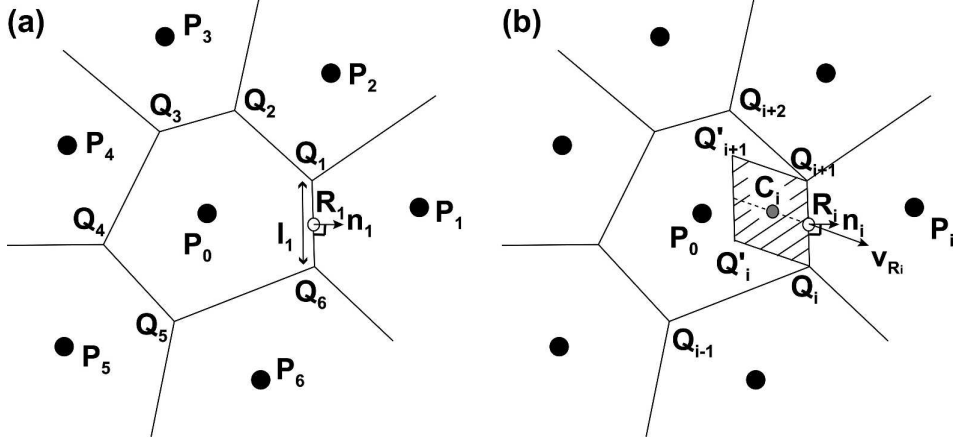


Figure 1.5: (a) Arrangement and notation of distorted hexagonal cells.(b) Schematic of the area swept by the arc $Q'_i Q'_{i+1}$ during a time interval.

1.6 Vertical coordinate and discretization

NICAM uses the Lorenz grid for the vertical grid configuration as shown in Figure 1.6, where W (vertical velocity with metrics) is defined at the half-integer levels and the other variables are defined at the integer levels. This model allows the grid stretching in the ξ coordinate as show in Figure 1.6. The integer levels are located at the mid-point of the upper and lower half-integer levels. The linear interpolation is used to obtain the values at the half-integer levels from the values at the integer levels, and vice versa:

$$\phi_{k+1/2} = a_{k+1/2} \phi_{k+1} + b_{k+1} \phi_k, \quad (1.19)$$

$$\phi_k = c_k \phi_{k+1/2} + d_k \phi_{k-1/2}, \quad (1.20)$$

where

$$a_{k+1/2} = \frac{\xi_{k+1/2} - \xi_k}{\xi_{k+1} - \xi_k}, \quad (1.21)$$

$$b_{k+1/2} = 1 - a_{k+1/2}, \quad (1.22)$$

$$c_k = \frac{\xi_k - \xi_{k-1/2}}{\xi_{k+1/2} - \xi_{k-1/2}}, \quad (1.23)$$

$$d_k = 1 - c_k. \quad (1.24)$$

$$(1.25)$$

One of main solvers is that of the Helmholtz equation, which can be expressed as

$$-\frac{\partial}{\partial \xi} \left[A^o \frac{\partial (A^i \tilde{W})}{\partial \xi} \right] - \frac{\partial (B \tilde{W})}{\partial \xi} - C^o \frac{\partial C^i \tilde{W}}{\partial \xi} + \alpha D \tilde{W} = S^{\text{all}} \quad (1.26)$$

where

$$\tilde{W} \equiv \frac{W^{*\tau + \Delta\tau}}{G^{1/2} \gamma^2} \left(= (\rho w)^{*\tau + \Delta\tau} \right) \quad (1.27)$$

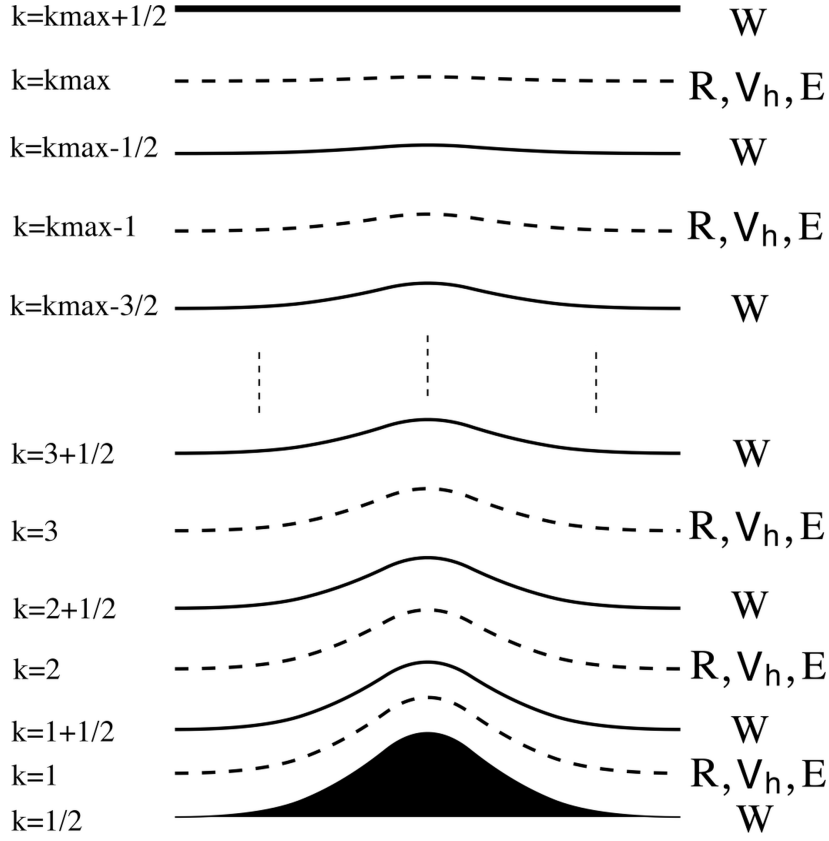


Figure 1.6: Schematic figure of vertical grid configuration.

and the coefficients are given as

$$A^o = \frac{1}{G^{1/2}\gamma^2}, \quad (1.28)$$

$$A^i = \gamma^2 h^t, \quad (1.29)$$

$$B = \tilde{g}^t, \quad (1.30)$$

$$C^o = \frac{C_v g}{R_d \gamma^2}, \quad (1.31)$$

$$C^i = \gamma^2, \quad (1.32)$$

$$D = \frac{C_v G^{1/2}}{R_d \Delta\tau^2}. \quad (1.33)$$

The source term is given as

$$S^{\text{all}} = \frac{C_v}{R_d \Delta\tau^2} \left(\frac{\alpha W^{*\tau} + \Delta\tau S^W}{\gamma^2} - \Delta\tau \frac{\partial}{\partial \xi} \left[\frac{1}{G^{1/2}\gamma^2} (P^{*\tau} + \Delta\tau S^P) \right] - \frac{\Delta\tau}{\gamma^2} (R^{*\tau} + \Delta\tau S^R) g \right). \quad (1.34)$$

Equation 1.26 is in discretized form at the level $k + 1/2$:

$$\begin{aligned}
& - \left[\frac{A_{k+1}^o \left(A_{k+3/2}^i \tilde{W}_{k+3/2} - A_{k+1/2}^i \tilde{W}_{k+1/2} \right)}{\Delta \xi_{k+1} \Delta \xi_{k+1/2}} - \frac{A_k^o \left(A_{k+1/2}^i \tilde{W}_{k+1/2} - A_{k-1/2}^i \tilde{W}_{k-1/2} \right)}{\Delta \xi_k \Delta_{k+1/2}} \right] \\
& - \left[\frac{\left(c_{k+1} B_{k+3/2} \tilde{W}_{k+3/2} + d_{k+1} B_{k+1/2} \tilde{W}_{k+1/2} \right) - \left(c_k B_{k+1/2} \tilde{W}_{k+1/2} + d_k B_{k-1/2} \tilde{W}_{k-1/2} \right)}{\Delta \xi_{k+1/2}} \right] \\
& - C_{k+1/2}^o \left[\frac{\left(c_{k+1} C_{k+3/2}^i \tilde{W}_{k+3/2} + d_{k+1} C_{k+1/2}^i \tilde{W}_{k+1/2} \right) - \left(c_k C_{k+1/2}^i \tilde{W}_{k+1/2} + d_k C_{k-1/2}^i \tilde{W}_{k-1/2} \right)}{\Delta \xi_{k+1/2}} \right] \\
& + \alpha D_{k+1/2} \tilde{W}_{k+1/2} \\
& = S_{k+1/2}^{\text{all}} \tag{1.35}
\end{aligned}$$

where

$$\begin{aligned}
S_{k+1/2}^{\text{all}} &= \frac{C_v}{R_d \Delta \tau^2} \left[\left(\frac{\alpha W_{k+1/2}^{*\tau} + \Delta \tau S_{k+1/2}^W}{\gamma_{k+1/2}^2} \right) \right. \\
& - \frac{\Delta \tau}{\Delta \xi_{k+1/2}} \left(\frac{P_{k+1}^{*\tau} + \Delta \tau S_{k+1}^P}{G_{k+1}^{1/2} \gamma_{k+1}^2} - \frac{P_k^{*\tau} + \Delta \tau S_k^P}{G_k^{1/2} \gamma_k^2} \right) \\
& \left. - g \Delta \tau \left(a_{k+1/2} \frac{R_{k+1}^{*\tau} + \Delta \tau S_{k+1}^R}{\gamma_{k+1}^2} + b_{k+1/2} \frac{R_k^{*\tau} + \Delta \tau S_k^R}{\gamma_k^2} \right) \right] \tag{1.36}
\end{aligned}$$

Equation 1.35 is a tridiagonal matrix system as follows:

$$\begin{aligned}
M_{k+1/2}^L \tilde{W}_{k-1/2} + M_{k+1/2}^C \tilde{W}_{k+1/2} + M_{k+1/2}^U \tilde{W}_{k+3/2} &= S_{k+1/2}^{\text{all}}, \\
\text{for } \frac{3}{2} \leq k + \frac{1}{2} \leq k_{\max} - \frac{1}{2}, & \tag{1.37}
\end{aligned}$$

where

$$\begin{aligned}
M_{k+1/2}^C &= \alpha D_{k+1/2} \\
& + \frac{1}{\Delta \xi_{k+1/2}} \left[\frac{A_{k+1}^o A_{k+1/2}^i}{\Delta \xi_{k+1}} + \frac{A_k^o A_{k+1/2}^i}{\Delta \xi_k} - (d_{k+1} - c_k) (B_{k+1/2} + C_{k+1/2}^o C_{k+1/2}^i) \right], \tag{1.38}
\end{aligned}$$

$$M_{k+1/2}^U = - \frac{A_{k+1}^o A_{k+3/2}^i}{\Delta \xi_{k+1} \Delta \xi_{k+1/2}} - \frac{c_{k+1} (B_{k+3/2} + C_{k+1/2}^o C_{k+3/2}^i)}{\Delta \xi_{k+1/2}}, \tag{1.39}$$

$$M_{k+1/2}^L = - \frac{A_{k+1}^o A_{k-1/2}^i}{\Delta \xi_{k+1} \Delta \xi_{k+1/2}} + \frac{d_k (B_{k-1/2} + C_{k+1/2}^o C_{k-1/2}^i)}{\Delta \xi_{k+1/2}}. \tag{1.40}$$

If the boundary conditions for \tilde{W} are given at $k = 1 - 1/2$ and $k_{\max} + 1/2$, this linear system can be written

explicitly as

$$\begin{pmatrix} M_{1+1/2}^C & M_{1+1/2}^U & 0 & \cdots & \cdots & \cdots & 0 \\ M_{2+1/2}^L & M_{2+1/2}^C & M_{2+1/2}^U & 0 & \cdots & \cdots & \vdots \\ 0 & M_{3+1/2}^L & M_{3+1/2}^C & M_{3+1/2}^U & 0 & \cdots & \vdots \\ \vdots & & \cdots & \cdots & \cdots & & 0 \\ \vdots & \cdots & \cdots & 0 & M_{k_{\max}-3/2}^L & M_{k_{\max}-3/2}^C & M_{k_{\max}-3/2}^U \\ 0 & \cdots & \cdots & \cdots & 0 & M_{k_{\max}-1/2}^L & M_{k_{\max}-1/2}^C \end{pmatrix} \quad (1.41)$$

$$\times \begin{pmatrix} \tilde{W}_{1+1/2} \\ \tilde{W}_{2+1/2} \\ \tilde{W}_{3+1/2} \\ \vdots \\ \tilde{W}_{k_{\max}-3/2} \\ \tilde{W}_{k_{\max}-1/2} \end{pmatrix} = \begin{pmatrix} S_{1+1/2}^{\text{all}} - M_{1+1/2}^L \tilde{W}_{1-1/2} \\ S_{2+1/2}^{\text{all}} \\ S_{3+1/2}^{\text{all}} \\ \vdots \\ S_{k_{\max}-3/2}^{\text{all}} \\ S_{k_{\max}-1/2}^{\text{all}} - M_{k_{\max}-1/2}^U \tilde{W}_{k_{\max}+1/2} \end{pmatrix}$$

Since A^i and B depend on large step values (see section 1.3), the compositions of the matrix (Equation 1.38, 1.39, 1.40, 1.41) are the updated only at the large step.

1.7 Domain decomposition and “rlevel”

For domain decomposition for parallel computing, *NICAM* adopt the concept of a region-division level, or “rlevel”, along with the grid-division level (Tomita et al., 2008). The ten rectangles in Figure 1.7a are the result of connecting two neighboring triangles of a spherical icosahedron. This structure is called “rlevel-0”. For each rectangle, four subrectangles are generated by connecting the diagonal midgridpoints (Figure 1.7b). This structure is called “rlevel-1”. This process is repeated until the desired number of regions is obtained.

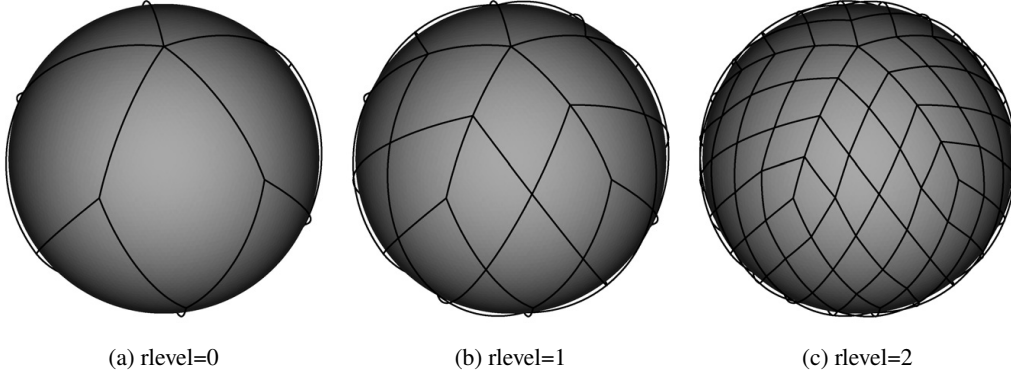


Figure 1.7: The method of region division. (Tomita et al., 2008)

1.8 Horizontal data structure

As described in section 1.4, the grid elements of an icosahedral grid are not rectangles. In *NICAM*, the shape of the control volume is hexagonal, except for the 12 points associated with the original icosahedron vertices. Therefore, the icosahedral grid is categorized as an unstructured grid. However, an icosahedral grid can actually be treated as a structured grid. An example of the grid structure in a region is shown in Figure 1.8a. The memory of each region in a single layer can be stored as a continuous one- or two-dimensional array. In

Figure 1.8a, the region manages the black gridpoints. The values indicated by the white circles are used only for reference and are provided from the neighboring regions by communication or memory copy. The values at the red circles are not used. If the west vertex of a region corresponds to a vertex of the original icosahedron, the two blue points shown in Figure 1.8a have the same values. With this data configuration, the black points cover all the icosahedral grid points except for the north and south poles.

Figure 1.8b shows data storage for the pole-region data. The gridpoint value included in this region is marked by a black circle. The other values marked by white circles are reference values, provided by communication or memory copy from the neighboring regular regions. The indices of the reference points are given in clockwise order. The pole regions are managed by the master process. Although this might cause load imbalance in parallel computing, the ratio of calculation required at the poles to that required in regular regions is small, and in practice the problem is negligible.

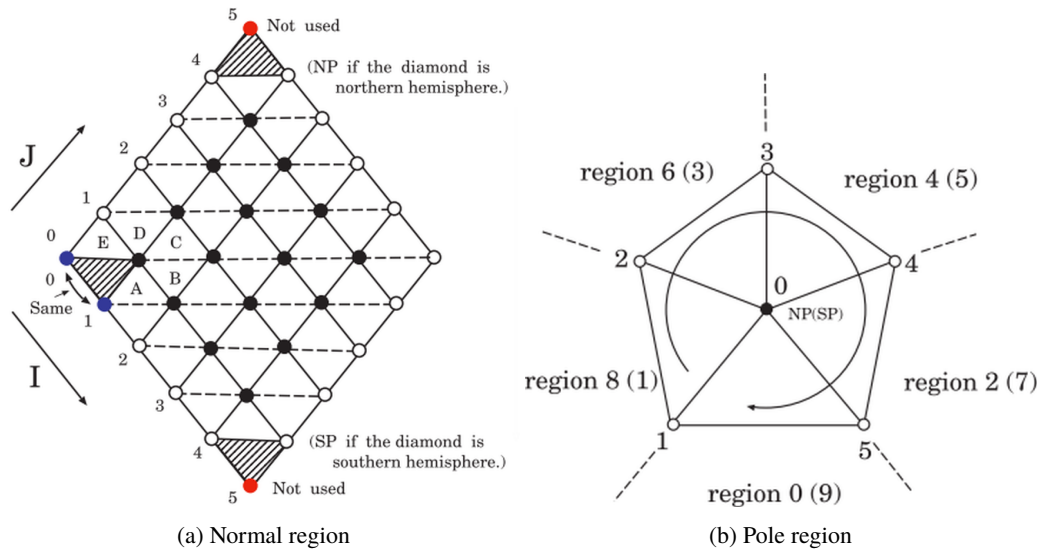


Figure 1.8: Grid structure of a region.(Tomita et al., 2008)

1.9 Parallelization

NICAM uses MPI for parallel execution. One MPI process manages one or several regions defined in previous section. For regions managed by the same process, exchanges of boundary values are done by memory copy, otherwise done by MPI communication. Figure 1.9 shows a schematic diagram of region management. In this case, rlevel is 1 and there are 40 regions, managed by 10 MPI processes. So a single process manages 4 regions, shown by the same color.

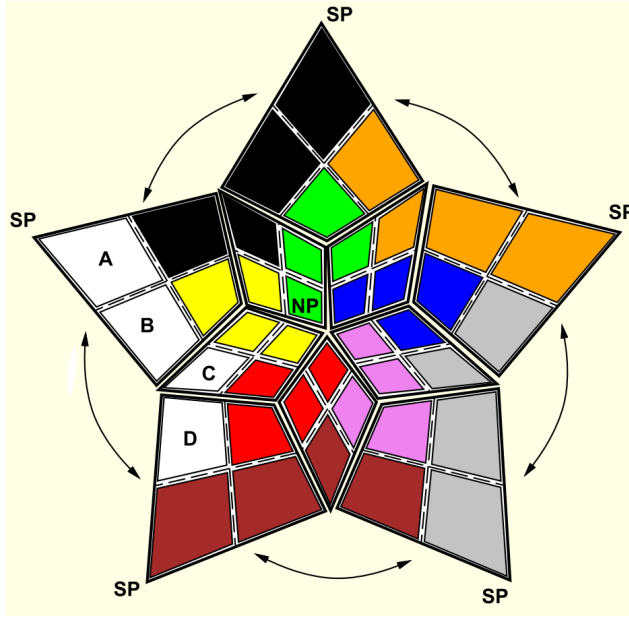


Figure 1.9: Schematic figure of parallelization. This figure shows the expansion of region geometry with rlevel-1.

1.10 Problem size

As shown in section 1.4, total number of grid points are defined by glevel. Total grid points N_g is calculated by Equation 1.42.

$$N_g = 10 \times 4^{gl} + 2, \quad (1.42)$$

where gl is glevel. +2 is corresponding to the pole points. Table 1.1 shows total number of grid points and average grid point distance for several glevels.

Table 1.1: Number of grid points and glevel

glevel	grid points	average grid interval[m]
0	12	7 142 126
1	42	3 571 063
2	162	1 785 432
...
8	655 362	27 899
9	2 621 442	13 949
10	10 485 762	6975
11	41 943 042	3487
...
13	2 684 354 562	872

Similarly, Total number of regions N_r is calculated by Equation 1.43.

$$N_r = 10 \times 4^{rl}, \quad (1.43)$$

where rl is rlevel. Table 1.2 shows the relation of rlevel and number of regions.

So the number of grid points in a single region (except polar region) N_p is calculated by Equation 1.44.

$$N_p = (2^{gl-rl} + 2)^2, \quad (1.44)$$

here +2 in parentheses denotes the halo grid points. For example, in case glevel is 5 and rlevel is 1, N_p is 324.

Table 1.2: Number of regions and rlevel

rlevel	number of regions numbers
0	10
1	40
2	160
3	640
4	2560

Chapter 2

Description of each kernel

2.1 Overview and common stuff

There are several things you should know when you read the kernel program codes.

2.1.1 Coding rule of *NICAM*

NICAM is written with Fortran90/95 standards, and uses module to modularize the program. Module name begins with `mod_`, such as `mod_adm`. Almost all source file defines only one module and have the same name with the module, such as `mod_adm.f90`. Several modules are used to define public parameters, variables, and subroutines, called “public object”. Name of such objects are prefixed its module name. For example, `ADM_gall` is defined in module `mod_adm` that is defined in `mod_adm.f90` in original *NICAM* source. Among the public objects, variables and parameters related to the problem size are moved and defined in `problem_size.inc`, that is included in `mod_misc.f90`

2.1.2 Data array for regular region

Figure 2.1 shows the schematic figure of data layout in a regular region. The grid points managed in the region are black circles, and white and blue circles are so called “halo points”, these are used only for the reference and their values are provided from the neighboring regions by the MPI communication or memory copy. If the west-most vertex of the region is the vertex of the original icosahedron, called singular points, two blue points have the same value. As you can see in Figure 2.1, all grid points can be specified by two index i and j , in the direction of southward and northward, respectively. For the computational efficiency, especially for vectorizing or using SIMD, the i and j dimension are merged, so usual variables are defined in a form shown in Table 2.1. The first dimensions corresponds to the horizontal index, the second is vertical index, and the third is region number (index) in each process.

The range of g is one from `ADM_gall`, that is the number of horizontal grid points including the halo points. Note that `ADM_gall = ADM_gall_1d*ADM_gall_1d`, where `ADM_gall_1d` is the number of grid points in i or j direction. `ADM_kall` is the actual number of vertical layers, and `ADM_lall` is the number of regions managed by a single MPI process. For example, if four regions are managed as Figure 1.9, `ADM_lall` is set to 4.

For some calculation, the values at the triangle center and those at the mid-points of arcs are required, called “the triangle point” and “the arc point”, respectively. Corresponding to the one grid point, there are two triangle points and three arc points. To specify these points, one more dimension m are used. Their value are `ADM_TI`, `ADM_TJ` for the triangle point, and `ADM_AI`, `ADM_AIJ` and `ADM_AJ` for the arc point, respectively (see Figure 2.1).

2.1.3 Data array for pole region

For pole region, Figure 2.2 shows the schematic figure of data storing. Only one grid point, the pole point, is managed, and the other point marked by the white circles are the halo points, these are used only for the

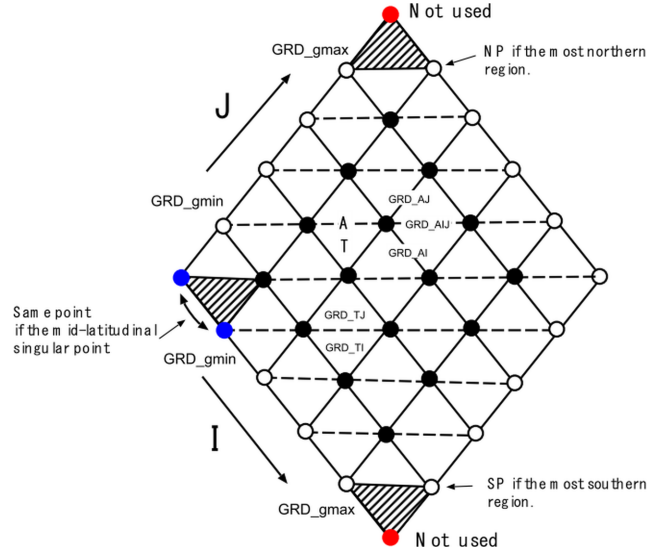


Figure 2.1: Schematic figure of data storing (regular region).

Table 2.1: Data array for the regular region.

$\text{var}(g,k,l) : \text{Grid point variables}$		
g	$= 1 \dots \text{ADM_gall}$	horizontal index
k	$= 1 \dots \text{ADM_kall}$	vertical index
l	$= 1 \dots \text{ADM_lall}$	region index
$\text{var}(g,k,l,m) : \text{Triangle point variables}$		
g	$= 1 \dots \text{ADM_gall}$	horizontal index
k	$= 1 \dots \text{ADM_kall}$	vertical index
l	$= 1 \dots \text{ADM_lall}$	region index
m	$= \text{ADM_TI}, \text{ADM_TJ}$	index of triangle position
$\text{var}(g,k,l,m) : \text{Arc point variables}$		
g	$= 1 \dots \text{ADM_gall}$	first horizontal index
k	$= 1 \dots \text{ADM_kall}$	vertical index
l	$= 1 \dots \text{ADM_lall}$	region index
m	$= \text{ADM_AI}, \text{ADM_AIJ}, \text{ADM_AJ}$	index of arc position

reference and their values are provided from the neighboring regions by the MPI communication or memory copy. The indices for these halo are in order of clockwise direction. The first dimension is horizontal suffix, and the size `ADM_GALL_PL` is set to 6, one for pole point and five for halo points. The second dimension is vertical suffix, which is the same with the regular region. The third dimension is region, suffix, and the range is 1 to `ADM_LALL_PL`, which is the number of pole regions, and set to 2 (North pole and South pole).

Different from the regular region, the number of triangle points and arc points are the same with the halo points, these are no need for more dimension, and stored as the same dimensions with the grid points. But the range of index are from `ADM_GMIN_PL(=2)` to `ADM_GMAX_PL(=6)`.

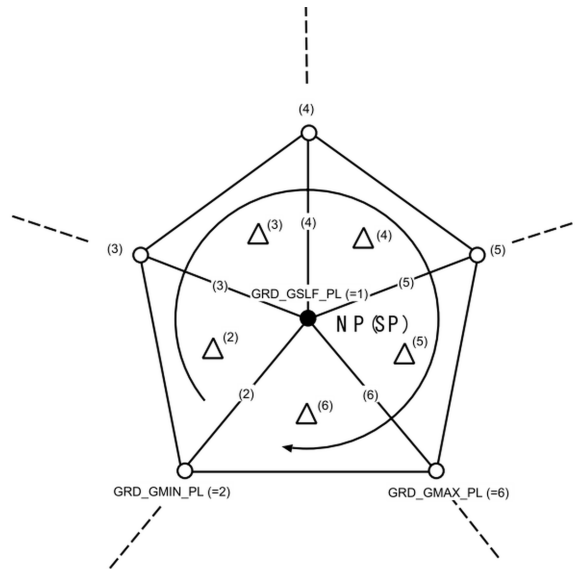


Figure 2.2: Schematic figure of data storing (pole region).

Table 2.2: Data array for the pole region.

<code>var_pl(n, k, l)</code> : Grid, triangle, arc point variables		
n	$= 1 \dots \text{ADM_GALL_PL}$	horizontal index
k	$= 1 \dots \text{ADM_kall}$	vertical index
l	$= 1 \dots \text{ADM_LALL_PL}$	region index

2.1.4 Kernelize

The kernels are single or multiple subroutines in original *NICAM* source code, and extracted and imposed into the wrapper for the kernel program. Values of input variables in the argument list of the kernel subroutine are stored as a data file just before the call in execution of the original *NICAM*, and they are read and given to the kernel subroutine in the kernel program. Similarly values of output variables in the argument list are stored just after the call in execution, and they are read and compared to the actual output values of kernel subroutine, the difference are written to the standard output for validation.

NICAM uses several “public object” defined in several modules, briefly described later. Some of them are moved to `problem_size.inc` and included in the `mod_misc`.

Kernel programs output several messages to the standard output, such as:

- min/max/sum of input data,
- min/max/sum of output data,
- min/max/sum of difference between output and validation data,

- computational time (elapsed time).

Elapsed time is measured using `omp_get_wtime()`.

There are sample output file for the reference in `reference/` directory of each kernel program, and also they are shown in “Input data and result” section of each kernel program in this document.

2.1.5 Problem size

In this kernel program package, problem size are set in `problem_size.inc` in each kernel program, except `communication`, as follows. Number of grid point of one side of the region `ADM_gall_1d` is 130 including two halo points, and the total number of grid points in whole region `ADM_gall` is 16900, which corresponds to `glevel - rlevel = 7`. Number of vertical layers `ADM_vlayer` is 40 and the actual size of k -direction `ADM_kall` is 42. The kernel program runs on a single process and this process manages only one normal region, ie. `ADM_lall` is 1, and also have two pole regions, ie. `ADM_have_pl` is `.true.` and `ADM_lall_pl` is 2. This normal region is to manage singular point, then `ADM_have_sgp(1)` is `.true.`

2.1.6 MPI and OpenMP

While original *NICAM* is parallelized by MPI and OpenMP, all kernel program in this package except `communication` are meant to be executed as one process and no threading. And you don’t need MPI library to compile/execute these kernel programs, but you need to make OpneMP enable in order to use `omp_get_wtime()`.

2.1.7 Mesuring environment

In the following sections, the example of performance result part of the log output file of each kernel program is shown. These were measured on the machine environment shown in [Table 2.3](#), with setting `export IAB_SYS=Ubuntu-gnu-mpi` on compilation (See `QuickStart.md`).

Table 2.3: Measuring environment

component	specification	notes
CPU	Xeon E5-2630v4 @2.2GHz (10cores) x2	HT disabled, TB enabled
Memory	256GB	
Storage	SSD (SATA)	
OS	Ubuntu 16.04.4 LTS	
Compiler	GNU 5.4.0	
MPI	OpenMPI 1.10.2	Ubuntu standard

2.2 dyn_diffusion

2.2.1 Description

Kernel `dyn_diffusion` is taken from the original subroutine `OPRT_diffusion` in *NICAM*. This subroutine is originally defined, as you can see in that name, in module `mod_oprt`. This module defines several differential operators on the sphere, such as divergence, gradient, etc. Subroutine `OPRT_diffusion` calculates diffusion term of given scalar field.

2.2.2 Discretization and code

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT_diffusion( &
2     dscl,      dscl_pl,    &
3     scl,       scl_pl,    &
4     kh,        kh_pl,    &
```

```

5     coef_intp, coef_intp_pl, &
6     coef_diff, coef_diff_pl )
7     implicit none
8
9     real(RP), intent(out) :: dsc1      (ADM_gall ,ADM_kall,ADM_lall )
10    real(RP), intent(out) :: dsc1_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl)
11    real(RP), intent(in)  :: scl      (ADM_gall ,ADM_kall,ADM_lall )
12    real(RP), intent(in)  :: scl_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl)
13    real(RP), intent(in)  :: kh       (ADM_gall ,ADM_kall,ADM_lall )
14    real(RP), intent(in)  :: kh_pl    (ADM_gall_pl,ADM_kall,ADM_lall_pl)
15    real(RP), intent(in)  :: coef_intp (ADM_gall ,1:3,      ADM_nxyz, TI:TJ, ADM_lall )
16    real(RP), intent(in)  :: coef_intp_pl(ADM_gall_pl,1:3,    ADM_nxyz,    ADM_lall_pl)
17    real(RP), intent(in)  :: coef_diff (ADM_gall ,1:6,      ADM_nxyz,    ADM_lall )
18    real(RP), intent(in)  :: coef_diff_pl(      1:ADM_vlink,ADM_nxyz,    ADM_lall_pl)
19
20    real(RP) :: vt (ADM_gall ,ADM_nxyz, TI:TJ)
21    real(RP) :: vt_pl(ADM_gall_pl,ADM_nxyz)
22    real(RP) :: kf (1:6)
23
24    integer :: gmin, gmax, iall, gall, kall, lall, nxyz, gminm1
25
26    integer :: ij
27    integer :: ip1j, ijp1, ip1jp1
28    integer :: im1j, imj1, im1jm1
29
30    integer :: g, k, l, d, n, v
31    !-----

```

Here `dsc1`, `dsc1_pl` are calculated diffusion of regular region and polar region, respectively, from some scalar field `scl`, `scl_pl` and diffusion coefficient `kh`, `kh_pl` at the triangular points. Other arguments `coef_intp`, `coef_intp_pl`, `coef_diff`, `coef_diff_pl` are various coefficients for finite difference calculation. These coefficients are calculated in advance in the subroutine `OPRT_diffusion_setup` also defined in the module (See section 2.8). These are supplied as arguments, not module variables, because of the computational optimization. The values are read from input data before executing this subroutine.

local variables `vt`, `vt_pl` are the differentiation of given scalar field at the gravitational center of triangles. Note that the range of the last dimension is `TI:TJ`. See below for details. `ADM_nxyz` is parameter and the value is 3, so this dimension shows spatial direction (X,Y,Z).

The first part of this subroutine is as follows.

```

32    call DEBUG_rapstart('OPRT_diffusion')
33
34    gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
35    gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
36    iall = ADM_gall_1d
37    gall = ADM_gall
38    kall = ADM_kall
39    lall = ADM_lall
40    nxyz = ADM_nxyz
41
42    gminm1 = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin-1
43
44    !$omp parallel default(none),private(g,k,l,d,ij,ip1j,ip1jp1,ijp1,im1j,imj1,im1jm1), &
45    !$omp shared(ADM_have_sgp,gminm1,gmin,gmax,iall,gall,kall,lall,nxyz,dsc1,scl,kh,kf,vt,coef_intp,coef_diff)
46    do l = 1, lall
47    do k = 1, kall
48
49        do d = 1, nxyz
50            !$omp do
51            do g = gminm1, gmax
52                ij = g
53                ip1j = g + 1
54                ip1jp1 = g + iall + 1
55                ijp1 = g + iall
56
57                vt(g,d, TI) = ( ( + 2.0_RP * coef_intp(g,1,d, TI,1) &
58                    - 1.0_RP * coef_intp(g,2,d, TI,1) &
59                    - 1.0_RP * coef_intp(g,3,d, TI,1) ) * scl(ij ,k,l) &
60                    + ( - 1.0_RP * coef_intp(g,1,d, TI,1) &
61                    + 2.0_RP * coef_intp(g,2,d, TI,1) &
62                    - 1.0_RP * coef_intp(g,3,d, TI,1) ) * scl(ip1j ,k,l) &
63                    + ( - 1.0_RP * coef_intp(g,1,d, TI,1) &
64                    - 1.0_RP * coef_intp(g,2,d, TI,1) &
65                    + 2.0_RP * coef_intp(g,3,d, TI,1) ) * scl(ip1jp1,k,l) &
66                    ) / 3.0_RP
67            enddo

```

```

68      !$omp end do nowait
69
70      !$omp do
71      do g = gminm1, gmax
72          ij = g
73          ip1j = g + 1
74          ip1jp1 = g + iall + 1
75          ijp1 = g + iall
76
77          vt(g,d,TJ) = ( ( + 2.0_RP * coef_intp(g,1,d,TJ,1) &
78                        - 1.0_RP * coef_intp(g,2,d,TJ,1) &
79                        - 1.0_RP * coef_intp(g,3,d,TJ,1) ) * scl(ij ,k,1) &
80                      + ( - 1.0_RP * coef_intp(g,1,d,TJ,1) &
81                        + 2.0_RP * coef_intp(g,2,d,TJ,1) &
82                        - 1.0_RP * coef_intp(g,3,d,TJ,1) ) * scl(ip1jp1,k,1) &
83                      + ( - 1.0_RP * coef_intp(g,1,d,TJ,1) &
84                        - 1.0_RP * coef_intp(g,2,d,TJ,1) &
85                        + 2.0_RP * coef_intp(g,3,d,TJ,1) ) * scl(ijp1 ,k,1) &
86                      ) / 3.0_RP
87
88      enddo
89      !$omp end do
90      enddo
91
92      if ( ADM_have_sgp(1) ) then ! pentagon
93          !$omp master
94          vt(gminm1,XDIR,TI) = vt(gminm1+1,XDIR,TJ)
95          vt(gminm1,YDIR,TI) = vt(gminm1+1,YDIR,TJ)
96          vt(gminm1,ZDIR,TI) = vt(gminm1+1,ZDIR,TJ)
97          !$omp end master
98      endif

```

In the first part, $vt(:, :, TI)$, $vt(:, :, TJ)$ are calculated from scl with interpolation. Figure 2.3 shows grid arrangements. scl are defined on white circle points in the figure, and vt are defined on black circle points. See Figure 2.4 for grid arrangement. vt is defined on the black circle points in the figure, while other physical quantities are defined on the white circle points.

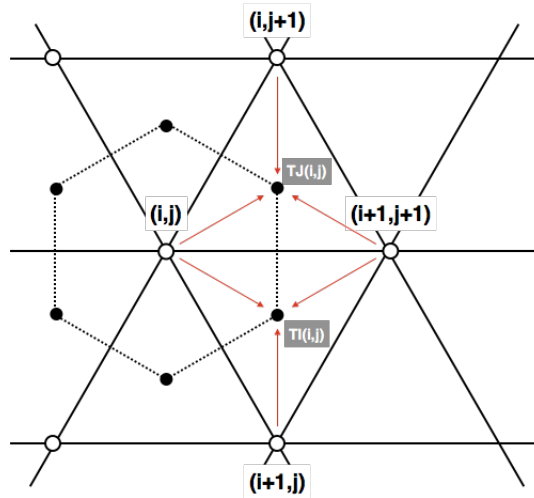


Figure 2.3: interpolation for vt

The second part of this subroutine is as follows.

```

98      !OCL XFILL
99      !$omp do
100     do g = 1, gmin-1
101         dscl(g,k,1) = 0.0_RP
102     enddo
103     !$omp end do nowait
104
105     !$omp do
106     do g = gmin, gmax
107         ij = g
108         ip1j = g + 1
109         ip1jp1 = g + iall + 1

```

```

110     ijp1 = g + iall
111     imlj = g - 1
112     imljm1 = g - iall - 1
113     ijm1 = g - iall
114
115     kf(1) = 0.5_RP * ( kh(ij ,k,l) + kh(ip1jp1,k,l) )
116     kf(2) = 0.5_RP * ( kh(ij ,k,l) + kh(ijp1 ,k,l) )
117     kf(3) = 0.5_RP * ( kh(imlj ,k,l) + kh(ij ,k,l) )
118     kf(4) = 0.5_RP * ( kh(imljm1,k,l) + kh(ij ,k,l) )
119     kf(5) = 0.5_RP * ( kh(ijm1 ,k,l) + kh(ij ,k,l) )
120     kf(6) = 0.5_RP * ( kh(ij ,k,l) + kh(ip1j ,k,l) )
121
122     dscl(g,k,l) = ( kf(1) * coef_diff(g,1,XDIR,l) * ( vt(ij ,XDIR,TI) + vt(ij ,XDIR,TJ) ) &
123 + kf(2) * coef_diff(g,2,XDIR,l) * ( vt(ij ,XDIR,TJ) + vt(imlj ,XDIR,TI) ) &
124 + kf(3) * coef_diff(g,3,XDIR,l) * ( vt(imlj ,XDIR,TI) + vt(imljm1,XDIR,TJ) ) &
125 + kf(4) * coef_diff(g,4,XDIR,l) * ( vt(imljm1,XDIR,TJ) + vt(imljm1,XDIR,TI) ) &
126 + kf(5) * coef_diff(g,5,XDIR,l) * ( vt(imljm1,XDIR,TI) + vt(ijm1 ,XDIR,TJ) ) &
127 + kf(6) * coef_diff(g,6,XDIR,l) * ( vt(ijm1 ,XDIR,TJ) + vt(ij ,XDIR,TI) ) )
128
129     !$omp end do
130
131     !$omp do
132     do g = gmin, gmax
133         ij = g
134         ip1j = g + 1
135         ip1jp1 = g + iall + 1
136         ijp1 = g + iall
137         imlj = g - 1
138         imljm1 = g - iall - 1
139         ijm1 = g - iall
140
141         kf(1) = 0.5_RP * ( kh(ij ,k,l) + kh(ip1jp1,k,l) )
142         kf(2) = 0.5_RP * ( kh(ij ,k,l) + kh(ijp1 ,k,l) )
143         kf(3) = 0.5_RP * ( kh(imlj ,k,l) + kh(ij ,k,l) )
144         kf(4) = 0.5_RP * ( kh(imljm1,k,l) + kh(ij ,k,l) )
145         kf(5) = 0.5_RP * ( kh(ijm1 ,k,l) + kh(ij ,k,l) )
146         kf(6) = 0.5_RP * ( kh(ij ,k,l) + kh(ip1j ,k,l) )
147
148         dscl(g,k,l) = dscl(g,k,l) + ( kf(1) * coef_diff(g,1,YDIR,l) * ( vt(ij ,XDIR,TI) + vt(ij ,YDIR,TJ) ) &
149 + kf(2) * coef_diff(g,2,YDIR,l) * ( vt(ij ,XDIR,TJ) + vt(imlj ,YDIR,TI) ) &
150 + kf(3) * coef_diff(g,3,YDIR,l) * ( vt(imlj ,XDIR,TI) + vt(imljm1,YDIR,TJ) ) &
151 + kf(4) * coef_diff(g,4,YDIR,l) * ( vt(imljm1,XDIR,TJ) + vt(imljm1,YDIR,TI) ) &
152 + kf(5) * coef_diff(g,5,YDIR,l) * ( vt(imljm1,XDIR,TI) + vt(ijm1 ,YDIR,TJ) ) &
153 + kf(6) * coef_diff(g,6,YDIR,l) * ( vt(ijm1 ,XDIR,TJ) + vt(ij ,YDIR,TI) ) )
154
155     !$omp end do
156
157     !$omp do
158     do g = gmin, gmax
159         ij = g
160         ip1j = g + 1
161         ip1jp1 = g + iall + 1
162         ijp1 = g + iall
163         imlj = g - 1
164         imljm1 = g - iall - 1
165         ijm1 = g - iall
166
167         kf(1) = 0.5_RP * ( kh(ij ,k,l) + kh(ip1jp1,k,l) )
168         kf(2) = 0.5_RP * ( kh(ij ,k,l) + kh(ijp1 ,k,l) )
169         kf(3) = 0.5_RP * ( kh(imlj ,k,l) + kh(ij ,k,l) )
170         kf(4) = 0.5_RP * ( kh(imljm1,k,l) + kh(ij ,k,l) )
171         kf(5) = 0.5_RP * ( kh(ijm1 ,k,l) + kh(ij ,k,l) )
172         kf(6) = 0.5_RP * ( kh(ij ,k,l) + kh(ip1j ,k,l) )
173
174         dscl(g,k,l) = dscl(g,k,l) + ( kf(1) * coef_diff(g,1,ZDIR,l) * ( vt(ij ,XDIR,TI) + vt(ij ,ZDIR,TJ) ) &
175 + kf(2) * coef_diff(g,2,ZDIR,l) * ( vt(ij ,XDIR,TJ) + vt(imlj ,ZDIR,TI) ) &
176 + kf(3) * coef_diff(g,3,ZDIR,l) * ( vt(imlj ,XDIR,TI) + vt(imljm1,ZDIR,TJ) ) &
177 + kf(4) * coef_diff(g,4,ZDIR,l) * ( vt(imljm1,XDIR,TJ) + vt(imljm1,ZDIR,TI) ) &
178 + kf(5) * coef_diff(g,5,ZDIR,l) * ( vt(imljm1,XDIR,TI) + vt(ijm1 ,ZDIR,TJ) ) &
179 + kf(6) * coef_diff(g,6,ZDIR,l) * ( vt(ijm1 ,XDIR,TJ) + vt(ij ,ZDIR,TI) ) )
180
181     enddo
182     !$omp end do nowait
183
184     !OCL XFILL
185     !$omp do
186     do g = gmax+1, gall
187         dscl(g,k,l) = 0.0_RP
188     enddo
189     !$omp end do
190
191     enddo ! loop k
192     enddo ! loop l
193
194     !$omp end parallel

```

In this part, objective variable $dsc1$ is calculated by interpolation of vt . Figure 2.4 shows the grid arrangements. vt are defined on black circle points as Figure 2.3, and $dsc1$ are defined on grey diamond points.

There are three similar do-loops, each of them calculates the contribution from X, Y, Z direction, respectively. Note that the third dimension of $coef_diff$ in the loop.

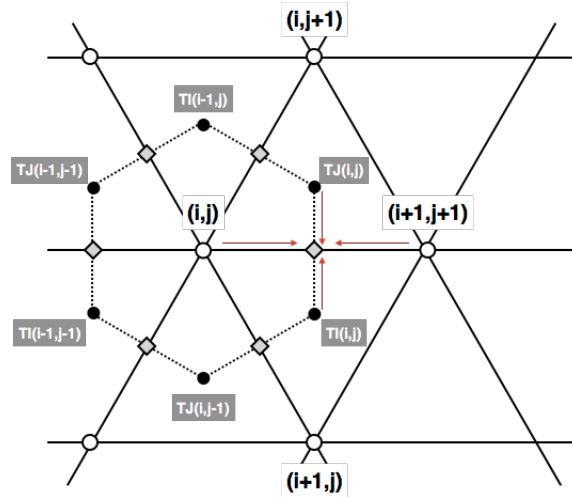


Figure 2.4: Interpolation for $dsc1$

The last part of this subroutine is as follows.

```

193 if ( ADM_have_pl ) then
194   n = ADM_gslf_pl
195
196   do l = 1, ADM_lall_pl
197     do k = 1, ADM_kall
198
199       do d = 1, ADM_nxyz
200         do v = ADM_gmin_pl, ADM_gmax_pl
201           ij = v
202           ijp1 = v + 1
203           if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
204
205           vt_pl(ij,d) = ( ( + 2.0_RP * coef_intp_pl(v,1,d,1) &
206             - 1.0_RP * coef_intp_pl(v,2,d,1) &
207             - 1.0_RP * coef_intp_pl(v,3,d,1) ) * scl_pl(n ,k,1) &
208             + ( - 1.0_RP * coef_intp_pl(v,1,d,1) &
209             + 2.0_RP * coef_intp_pl(v,2,d,1) &
210             - 1.0_RP * coef_intp_pl(v,3,d,1) ) * scl_pl(ij ,k,1) &
211             + ( - 1.0_RP * coef_intp_pl(v,1,d,1) &
212             - 1.0_RP * coef_intp_pl(v,2,d,1) &
213             + 2.0_RP * coef_intp_pl(v,3,d,1) ) * scl_pl(ijp1,k,1) &
214             ) / 3.0_RP
215
216         enddo
217       enddo
218
219       dsc1_pl(:,k,1) = 0.0_RP
220
221       do v = ADM_gmin_pl, ADM_gmax_pl
222         ij = v
223         ijm1 = v - 1
224         if( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl ! cyclic condition
225
226         dsc1_pl(n,k,1) = dsc1_pl(n,k,1) &
227           + ( coef_diff_pl(v-1,XDIR,1) * ( vt_pl(ijm1,XDIR) + vt_pl(ij,XDIR) ) &
228             + coef_diff_pl(v-1,YDIR,1) * ( vt_pl(ijm1,YDIR) + vt_pl(ij,YDIR) ) &
229             + coef_diff_pl(v-1,ZDIR,1) * ( vt_pl(ijm1,ZDIR) + vt_pl(ij,ZDIR) ) &
230             ) * 0.5_RP * ( kh_pl(n,k,1) + kh_pl(ij,k,1) )
231       enddo

```



```

232     enddo
233     enddo
234     else
235         dscl_pl(:, :, :) = 0.0_RP
236     endif
237
238     call DEBUG_rapend('OPRT_diffusion')
239
240     return
241 end subroutine OPRT_diffusion

```

The last part is for calculation for the pole region. Variable `ADM_have_pl` is `.true.` if this process manages pole region in original *NICAM*. For the kernel program, also set as `.true.` in `problem_size.inc`.

2.2.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of `$IAB_SYS=Ubuntu-gnu-mpi` case.

```

### Input ###
+check[check_dscl      ] max=  6.1941315670898286E-08,min= -7.0374144752795210E-08,sum= -2.7407850230958588E-07
+check[check_dscl_pl  ] max=  2.2139244811324830E-08,min= -6.0170678327656930E-11,sum=  1.8274579608905828E-07
+check[sc1             ] max=  1.6578626530298903E-11,min= -1.2670212860993856E-11,sum= -2.0289014286353776E-10
+check[sc1_pl         ] max=  1.9358849576453664E-11,min= -8.2853331106472899E-12,sum=  1.1445754720677440E-09
+check[kh             ] max=  2.8341305529772246E+12,min=  6.7659597088284981E+10,sum=  6.0439053980501018E+17
+check[kh_pl          ] max=  2.8334094314435532E+12,min=  6.7659597088284981E+10,sum=  4.3486317454839525E+14
### Output ###
+check[dscl           ] max=  6.1941315670898286E-08,min= -7.0374144752795210E-08,sum= -2.7407850230958588E-07
+check[dscl_pl        ] max=  2.2139244811324830E-08,min= -6.0170678327656930E-11,sum=  1.8274579608905828E-07
### Validation : point-by-point diff ###
+check[check_dscl     ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[check_dscl_pl  ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel

```

Check the lines below “Validation : point-by-point diff” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.2.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in [subsection 2.1.7](#). Note that in this program kernel part is iterated one time.

```

*** Computational Time Report
*** ID=001 : MAIN_dyn_diffusion          T=  0.028 N=  1
*** ID=002 : OPRT_diffusion              T=  0.028 N=  1

```

2.3 dyn_divdamp

2.3.1 Description

Kernel `dyn_divdamp` is taken from the original subroutine `OPRT3D_divdamp` in *NICAM*. This subroutine is originally defined, as you can see in that name, in `mod_oprt3d`. Subroutine `OPRT3D_divdamp` calculates the gradient of divergence of the vector $\{v_x, v_y, v_z\}$. If this vector represents the velocity vector, this term is called “divergence damping term”.

2.3.2 Discretization and code

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT3D_divdamp( &
2     ddivdx, ddivdx_pl, &
3     ddivdy, ddivdy_pl, &
4     ddivdz, ddivdz_pl, &
5     rhogvx, rhogvx_pl, &
6     rhogvy, rhogvy_pl, &
7     rhogvz, rhogvz_pl, &
8     rhogw, rhogw_pl, &
9     coef_intp, coef_intp_pl, &
10    coef_diff, coef_diff_pl )
11  implicit none
12
13  real(RP), intent(out) :: ddivdx (ADM_gall ,ADM_kall,ADM_lall ) ! tendency
14  real(RP), intent(out) :: ddivdx_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
15  real(RP), intent(out) :: ddivdy (ADM_gall ,ADM_kall,ADM_lall )
16  real(RP), intent(out) :: ddivdy_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
17  real(RP), intent(out) :: ddivdz (ADM_gall ,ADM_kall,ADM_lall )
18  real(RP), intent(out) :: ddivdz_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
19  real(RP), intent(in) :: rhogvx (ADM_gall ,ADM_kall,ADM_lall ) ! rho*vx { gam2 x G^-1/2 }
20  real(RP), intent(in) :: rhogvx_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
21  real(RP), intent(in) :: rhogvy (ADM_gall ,ADM_kall,ADM_lall ) ! rho*vy { gam2 x G^-1/2 }
22  real(RP), intent(in) :: rhogvy_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
23  real(RP), intent(in) :: rhogvz (ADM_gall ,ADM_kall,ADM_lall ) ! rho*vz { gam2 x G^-1/2 }
24  real(RP), intent(in) :: rhogvz_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
25  real(RP), intent(in) :: rhogw (ADM_gall ,ADM_kall,ADM_lall ) ! rho*w { gam2 x G^-1/2 }
26  real(RP), intent(in) :: rhogw_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
27  real(RP), intent(in) :: coef_intp (ADM_gall ,1:3,ADM_nxyz,TI:TJ,ADM_lall )
28  real(RP), intent(in) :: coef_intp_pl (ADM_gall_pl,1:3,ADM_nxyz, ADM_lall_pl)
29  real(RP), intent(in) :: coef_diff (ADM_gall,1:6 ,ADM_nxyz,ADM_lall )
30  real(RP), intent(in) :: coef_diff_pl( 1:ADM_vlink,ADM_nxyz,ADM_lall_pl)
31
32  real(RP) :: sclt (ADM_gall ,TI:TJ) ! scalar on the hexagon vertex
33  real(RP) :: sclt_pl(ADM_gall_pl)
34  real(RP) :: sclt_rhogw
35  real(RP) :: sclt_rhogw_pl
36
37  real(RP) :: rhogvx_vm (ADM_gall ) ! rho*vx / vertical metrics
38  real(RP) :: rhogvx_vm_pl (ADM_gall_pl)
39  real(RP) :: rhogvy_vm (ADM_gall ) ! rho*vy / vertical metrics
40  real(RP) :: rhogvy_vm_pl (ADM_gall_pl)
41  real(RP) :: rhogvz_vm (ADM_gall ) ! rho*vz / vertical metrics
42  real(RP) :: rhogvz_vm_pl (ADM_gall_pl)
43  real(RP) :: rhogw_vm (ADM_gall ,ADM_kall,ADM_lall ) ! rho*w / vertical metrics
44  real(RP) :: rhogw_vm_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
45
46  integer :: gmin, gmax, iall, gall, kall, kmin, kmax, lall, gminm1
47
48  integer :: ij
49  integer :: ip1j, ijp1, ip1jp1
50  integer :: im1j, imj1, im1jm1
51
52  integer :: g, k, l, n, v
53  !-----

```

Here `ddivdx`, `ddivdy`, `ddivdz` are calculated x, y, z component of the gradient of divergence, respectively. And these with `_pl` are those for the pole region. `rhogvx`, `rhogvy`, `rhogvz`, and `rhogw` are $G^{1/2}\gamma^2 \times \rho v_x$, $G^{1/2}\gamma^2 \times \rho v_y$, $G^{1/2}\gamma^2 \times \rho v_z$, and $G^{1/2}\gamma^2 \times \rho w$, respectively, where $\{v_x, v_y, v_z\}$ are the wind vector of horizontal wind component in 3-D Cartesian coordinates, w is vertical wind, and $G^{1/2}$ and γ are the metrics comes from the terrain-following coordinate described in [section 1.6](#) Other arguments `coef_intp`, `coef_diff` and those with `_pl` are various coefficients for finite difference calculation, the same with those in `dyn_diffusion`.

Local variable `sclt`, `sclt_pl` are the scalar value on the gravitational center of triangles, i.e. the vertices of the hexagonal control volume. `rhogvx_vm` etc. are `rhogvx` divided by the vertical metrics.

The first part of this subroutine is as follows.

```

54  call DEBUG_rapstart('OPRT3D_divdamp')
55
56  gmin = (ADM_gmin-1)*ADM_gall_id + ADM_gmin
57  gmax = (ADM_gmax-1)*ADM_gall_id + ADM_gmax
58  iall = ADM_gall_id
59  gall = ADM_gall
60  kall = ADM_kall
61  kmin = ADM_kmin
62  kmax = ADM_kmax

```

```

63  lall = ADM_lall
64
65  gminm1 = (ADM_gmin-1-1)*ADM_gall_id + ADM_gmin-1
66
67  !$omp parallel default(none),private(g,k,l), &
68  !$omp shared(gall,kmin,kmax,lall,rhogw_vm,rhogvx,rhogvy,rhogvz,rhogw,VMTR_C2WfactGz,VMTR_RGSQRTH,VMTR_RGAMH)
69  do l = 1, lall
70    !$omp do
71    do k = kmin+1, kmax
72    do g = 1, gall
73      rhogw_vm(g,k,l) = ( VMTR_C2WfactGz(g,k,1,l) * rhogvx(g,k ,l) &
74                        + VMTR_C2WfactGz(g,k,2,l) * rhogvx(g,k-1,l) &
75                        + VMTR_C2WfactGz(g,k,3,l) * rhogvy(g,k ,l) &
76                        + VMTR_C2WfactGz(g,k,4,l) * rhogvy(g,k-1,l) &
77                        + VMTR_C2WfactGz(g,k,5,l) * rhogvz(g,k ,l) &
78                        + VMTR_C2WfactGz(g,k,6,l) * rhogvz(g,k-1,l) &
79                        ) * VMTR_RGAMH(g,k,l) & ! horizontal contribution
80                        + rhogw(g,k,l) * VMTR_RGSQRTH(g,k,l) & ! vertical contribution
81
82    enddo
83  enddo
84  !$omp end do nowait
85
86  !OCL XFILL
87  !$omp do
88  do g = 1, gall
89    rhogw_vm(g,kmin ,l) = 0.0_RP
90    rhogw_vm(g,kmax+1,l) = 0.0_RP
91  enddo
92  !$omp end do
93  enddo
94  !$omp end parallel

```

This part calculates ρw_{vm} from 3 components of ρv and ρw (with the metrics). The values are located at the triangular points in horizontal direction, and at the half-integer levels in vertical direction. Coefficients prefixed with $VMTR_$ are defined and pre-calculated in module `mod_vmtr` in original *NICAM*. When kernelize this subroutine, these definition are moved to `mod_misc`, and read from the input data file.

Second part is pretty long as follows.

```

95  !$omp parallel default(none),private(g,k,l,i,j,ip1j,ip1jp1,ip1,im1j,ijm1,im1jm1,sclt_rhogw), &
96  !$omp shared(ADM_have_sgp,gminm1,gmin,gmax,gall,kmin,kmax,lall,iall,ddivdx,ddivdy,ddivdz,rhogvx,rhogvy,rhogvz, &
97  !$omp rhogvx_vm,rhogvy_vm,rhogvz_vm,rhogw_vm,sclt,coef_intp,coef_diff,GRD_rdgz,VMTR_RGAM)
98  do l = 1, lall
99  do k = kmin, kmax
100  !OCL XFILL
101  !$omp do
102  do g = 1, gall
103    rhogvx_vm(g) = rhogvx(g,k,l) * VMTR_RGAM(g,k,l)
104    rhogvy_vm(g) = rhogvy(g,k,l) * VMTR_RGAM(g,k,l)
105    rhogvz_vm(g) = rhogvz(g,k,l) * VMTR_RGAM(g,k,l)
106  enddo
107  !$omp end do
108
109  !$omp do
110  do g = gminm1, gmax
111    ij = g
112    ip1j = g + 1
113    ip1jp1 = g + iall + 1
114    ijp1 = g + iall
115
116    sclt_rhogw = ( ( rhogw_vm(ij,k+1,l) + rhogw_vm(ip1j,k+1,l) + rhogw_vm(ip1jp1,k+1,l) ) &
117                  - ( rhogw_vm(ij,k ,l) + rhogw_vm(ip1j,k ,l) + rhogw_vm(ip1jp1,k ,l) ) &
118                  ) / 3.0_RP * GRD_rdgz(k)
119
120    sclt(g,TI) = coef_intp(g,1,XDIR,TI,l) * rhogvx_vm(ij ) &
121                + coef_intp(g,2,XDIR,TI,l) * rhogvx_vm(ip1j ) &
122                + coef_intp(g,3,XDIR,TI,l) * rhogvx_vm(ip1jp1) &
123                + coef_intp(g,1,YDIR,TI,l) * rhogvy_vm(ij ) &
124                + coef_intp(g,2,YDIR,TI,l) * rhogvy_vm(ip1j ) &
125                + coef_intp(g,3,YDIR,TI,l) * rhogvy_vm(ip1jp1) &
126                + coef_intp(g,1,ZDIR,TI,l) * rhogvz_vm(ij ) &
127                + coef_intp(g,2,ZDIR,TI,l) * rhogvz_vm(ip1j ) &
128                + coef_intp(g,3,ZDIR,TI,l) * rhogvz_vm(ip1jp1) &
129                + sclt_rhogw
130  enddo
131  !$omp end do nowait
132
133  !$omp do

```

```

134 do g = gmin1, gmax
135   ij = g
136   ip1j = g + 1
137   ip1jp1 = g + iall + 1
138   ijp1 = g + iall
139
140   sclt_rhogw = ( ( rhogw_vm(ij,k+1,1) + rhogw_vm(ip1jp1,k+1,1) + rhogw_vm(ijp1,k+1,1) ) &
141     - ( rhogw_vm(ij,k ,1) + rhogw_vm(ip1jp1,k ,1) + rhogw_vm(ijp1,k ,1) ) &
142     ) / 3.0_RP * GRD_rdgz(k)
143
144   sclt(g,TJ) = coef_intp(g,1,XDIR,TJ,1) * rhogvx_vm(ij ) &
145     + coef_intp(g,2,XDIR,TJ,1) * rhogvx_vm(ip1jp1) &
146     + coef_intp(g,3,XDIR,TJ,1) * rhogvx_vm(ijp1 ) &
147     + coef_intp(g,1,YDIR,TJ,1) * rhogvy_vm(ij ) &
148     + coef_intp(g,2,YDIR,TJ,1) * rhogvy_vm(ip1jp1) &
149     + coef_intp(g,3,YDIR,TJ,1) * rhogvy_vm(ijp1 ) &
150     + coef_intp(g,1,ZDIR,TJ,1) * rhogvz_vm(ij ) &
151     + coef_intp(g,2,ZDIR,TJ,1) * rhogvz_vm(ip1jp1) &
152     + coef_intp(g,3,ZDIR,TJ,1) * rhogvz_vm(ijp1 ) &
153     + sclt_rhogw
154
155   !$omp end do
156
157   if ( ADM_have_sgp(1) ) then ! pentagon
158     !$omp master
159     sclt(gmin1,TI) = sclt(gmin1+1,TJ)
160     !$omp end master
161   endif
162
163   !OCL XFILL
164   !$omp do
165   do g = 1, gmin-1
166     ddivdx(g,k,1) = 0.0_RP
167     ddivdy(g,k,1) = 0.0_RP
168     ddivdz(g,k,1) = 0.0_RP
169   enddo
170   !$omp end do nowait
171
172   !$omp do
173   do g = gmin, gmax
174     ij = g
175     im1j = g - 1
176     im1jm1 = g - iall - 1
177     ijm1 = g - iall
178
179     ddivdx(g,k,1) = ( coef_diff(g,1,XDIR,1) * ( sclt(ij, TI) + sclt(ij, TJ) ) &
180       + coef_diff(g,2,XDIR,1) * ( sclt(ij, TJ) + sclt(im1j, TI) ) &
181       + coef_diff(g,3,XDIR,1) * ( sclt(im1j, TI) + sclt(im1jm1,TJ) ) &
182       + coef_diff(g,4,XDIR,1) * ( sclt(im1jm1,TJ) + sclt(im1jm1,TI) ) &
183       + coef_diff(g,5,XDIR,1) * ( sclt(im1jm1,TI) + sclt(ijm1 ,TJ) ) &
184       + coef_diff(g,6,XDIR,1) * ( sclt(ijm1, TJ) + sclt(ij, TI) ) )
185
186   enddo
187   !$omp end do nowait
188
189   !$omp do
190   do g = gmin, gmax
191     ij = g
192     im1j = g - 1
193     im1jm1 = g - iall - 1
194     ijm1 = g - iall
195
196     ddivdy(g,k,1) = ( coef_diff(g,1,YDIR,1) * ( sclt(ij, TI) + sclt(ij, TJ) ) &
197       + coef_diff(g,2,YDIR,1) * ( sclt(ij, TJ) + sclt(im1j, TI) ) &
198       + coef_diff(g,3,YDIR,1) * ( sclt(im1j, TI) + sclt(im1jm1,TJ) ) &
199       + coef_diff(g,4,YDIR,1) * ( sclt(im1jm1,TJ) + sclt(im1jm1,TI) ) &
200       + coef_diff(g,5,YDIR,1) * ( sclt(im1jm1,TI) + sclt(ijm1 ,TJ) ) &
201       + coef_diff(g,6,YDIR,1) * ( sclt(ijm1, TJ) + sclt(ij, TI) ) )
202
203   enddo
204   !$omp end do nowait
205
206   !$omp do
207   do g = gmin, gmax
208     ij = g
209     im1j = g - 1
210     im1jm1 = g - iall - 1
211     ijm1 = g - iall
212
213     ddivdz(g,k,1) = ( coef_diff(g,1,ZDIR,1) * ( sclt(ij, TI) + sclt(ij, TJ) ) &
214       + coef_diff(g,2,ZDIR,1) * ( sclt(ij, TJ) + sclt(im1j, TI) ) &
215       + coef_diff(g,3,ZDIR,1) * ( sclt(im1j, TI) + sclt(im1jm1,TJ) ) &
216       + coef_diff(g,4,ZDIR,1) * ( sclt(im1jm1,TJ) + sclt(im1jm1,TI) ) &
217       + coef_diff(g,5,ZDIR,1) * ( sclt(im1jm1,TI) + sclt(ijm1 ,TJ) ) &

```

```

216         + coef_diff(g,6,ZDIR,1) * ( sclt(ijm1, TJ) + sclt(ij, TI) ) )
217     enddo
218     !$omp end do nowait
219
220 !OCL XFILL
221 !$omp do
222     do g = gmax+1, gall
223         ddivdx(g,k,1) = 0.0_RP
224         ddivdy(g,k,1) = 0.0_RP
225         ddivdz(g,k,1) = 0.0_RP
226     enddo
227     !$omp end do
228     enddo ! loop k
229
230 !OCL XFILL
231 !$omp do
232     do g = 1, gall
233         ddivdx(g,kmin-1,1) = 0.0_RP
234         ddivdy(g,kmin-1,1) = 0.0_RP
235         ddivdz(g,kmin-1,1) = 0.0_RP
236         ddivdx(g,kmax+1,1) = 0.0_RP
237         ddivdy(g,kmax+1,1) = 0.0_RP
238         ddivdz(g,kmax+1,1) = 0.0_RP
239     enddo
240     !$omp end do
241     enddo ! loop l
242     !$omp end parallel
243

```

In the outer most l-loop(1.98) and k-loop(1.99), sclt at the triangular point TI(1.120) and TJ(1.144) are calculated separately by interpolation. and finally, in the g-loop begins at 1.173, desired ddivdx(1.179), ddivdy(1.195), and ddivdz(1.211) are calculated.

The final part is for the pole region, doing almost the same thing as the normal region described above.

```

244 if ( ADM_have_pl ) then
245     n = ADM_gslf_pl
246
247     do l = 1, ADM_lall_pl
248         do k = ADM_kmin+1, ADM_kmax
249             do g = 1, ADM_gall_pl
250                 rhogw_vm_pl(g,k,1) = ( VMTR_C2WfactGz_pl(g,k,1,1) * rhogvx_pl(g,k,1) &
251                     + VMTR_C2WfactGz_pl(g,k,2,1) * rhogvx_pl(g,k-1,1) &
252                     + VMTR_C2WfactGz_pl(g,k,3,1) * rhogvy_pl(g,k,1) &
253                     + VMTR_C2WfactGz_pl(g,k,4,1) * rhogvy_pl(g,k-1,1) &
254                     + VMTR_C2WfactGz_pl(g,k,5,1) * rhogvz_pl(g,k,1) &
255                     + VMTR_C2WfactGz_pl(g,k,6,1) * rhogvz_pl(g,k-1,1) &
256                     + VMTR_RGAMH_pl(g,k,1) & ! horizontal contribution
257                     + rhogw_pl(g,k,1) * VMTR_RGSQRTH_pl(g,k,1) & ! vertical contribution
258                 )
259             enddo
260
261             do g = 1, ADM_gall_pl
262                 rhogw_vm_pl(g,ADM_kmin,1) = 0.0_RP
263                 rhogw_vm_pl(g,ADM_kmax+1,1) = 0.0_RP
264             enddo
265         enddo
266
267         do l = 1, ADM_lall_pl
268             do k = ADM_kmin, ADM_kmax
269                 do v = 1, ADM_gall_pl
270                     rhogvx_vm_pl(v) = rhogvx_pl(v,k,1) * VMTR_RGAM_pl(v,k,1)
271                     rhogvy_vm_pl(v) = rhogvy_pl(v,k,1) * VMTR_RGAM_pl(v,k,1)
272                     rhogvz_vm_pl(v) = rhogvz_pl(v,k,1) * VMTR_RGAM_pl(v,k,1)
273                 enddo
274
275                 do v = ADM_gmin_pl, ADM_gmax_pl
276                     ij = v
277                     ijp1 = v + 1
278                     if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
279
280                     sclt_rhogw_pl = ( ( rhogw_vm_pl(n,k+1,1) + rhogw_vm_pl(ij,k+1,1) + rhogw_vm_pl(ijp1,k+1,1) ) &
281                         - ( rhogw_vm_pl(n,k,1) + rhogw_vm_pl(ij,k,1) + rhogw_vm_pl(ijp1,k,1) ) &
282                         ) / 3.0_RP * GRD_rdgz(k)
283
284                     sclt_pl(ij) = coef_intp_pl(v,1,XDIR,1) * rhogvx_vm_pl(n,1) &
285                         + coef_intp_pl(v,2,XDIR,1) * rhogvx_vm_pl(ij,1) &
286                         + coef_intp_pl(v,3,XDIR,1) * rhogvx_vm_pl(ijp1,1) &
287                         + coef_intp_pl(v,1,YDIR,1) * rhogvy_vm_pl(n,1) &

```

```

288         + coef_intp_pl(v,2,YDIR,1) * rhogvy_vm_pl(ij ) &
289         + coef_intp_pl(v,3,YDIR,1) * rhogvy_vm_pl(ijpl) &
290         + coef_intp_pl(v,1,ZDIR,1) * rhogvz_vm_pl(n ) &
291         + coef_intp_pl(v,2,ZDIR,1) * rhogvz_vm_pl(ij ) &
292         + coef_intp_pl(v,3,ZDIR,1) * rhogvz_vm_pl(ijpl) &
293         + sclt_rhogw_pl
294     enddo
295
296     ddivdx_pl(:,k,l) = 0.0_RP
297     ddivdy_pl(:,k,l) = 0.0_RP
298     ddivdz_pl(:,k,l) = 0.0_RP
299
300     do v = ADM_gmin_pl, ADM_gmax_pl
301         ij = v
302         ijm1 = v - 1
303         if( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl ! cyclic condition
304
305         ddivdx_pl(n,k,l) = ddivdx_pl(n,k,l) + coef_diff_pl(v-1,XDIR,1) * ( sclt_pl(ijm1) + sclt_pl(ij) )
306         ddivdy_pl(n,k,l) = ddivdy_pl(n,k,l) + coef_diff_pl(v-1,YDIR,1) * ( sclt_pl(ijm1) + sclt_pl(ij) )
307         ddivdz_pl(n,k,l) = ddivdz_pl(n,k,l) + coef_diff_pl(v-1,ZDIR,1) * ( sclt_pl(ijm1) + sclt_pl(ij) )
308     enddo
309 enddo
310
311     ddivdx_pl(:,ADM_kmin-1,l) = 0.0_RP
312     ddivdx_pl(:,ADM_kmax+1,l) = 0.0_RP
313     ddivdy_pl(:,ADM_kmin-1,l) = 0.0_RP
314     ddivdy_pl(:,ADM_kmax+1,l) = 0.0_RP
315     ddivdz_pl(:,ADM_kmin-1,l) = 0.0_RP
316     ddivdz_pl(:,ADM_kmax+1,l) = 0.0_RP
317 enddo
318 else
319     ddivdx_pl(:, :, :) = 0.0_RP
320     ddivdy_pl(:, :, :) = 0.0_RP
321     ddivdz_pl(:, :, :) = 0.0_RP
322 endif
323
324 call DEBUG_rapend('OPRT3D_divdamp')
325
326 return
327 end subroutine OPRT3D_divdamp

```

2.3.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of \$IAB_SYS=Ubuntu-gnu-mpi case.

```

### Input ###
+check[check_ddivdx ] max= 3.0131744015374420E-11,min= -5.2988229203876260E-11,sum= -1.6975403236890680E-11
+check[check_ddivdx_pl ] max= 2.6717761683260969E-11,min= -5.3219224048058505E-11,sum= 4.8275913249408648E-11
+check[check_ddivdy ] max= 2.8051192990274010E-11,min= -2.3223214708700599E-11,sum= -4.5483269668124264E-11
+check[check_ddivdy_pl ] max= 4.4686835079822753E-11,min= -4.1286060234480353E-11,sum= 6.8817604250940581E-11
+check[check_ddivdz ] max= 3.4380463226247279E-11,min= -1.3010875841849350E-11,sum= -7.2052908719075937E-11
+check[check_ddivdz_pl ] max= 3.2276474319270210E-13,min= -8.5477300588035169E-14,sum= 3.9437219768394675E-12
+check[rhogvx ] max= 2.2205470252374623E-01,min= -2.0636880346843767E-01,sum= -1.7381945003843336E+02
+check[rhogvx_pl ] max= 1.4368235944041274E-01,min= -1.9062928892442133E-01,sum= -5.6937904377485706E+00
+check[rhogvy ] max= 2.4120815977222165E-01,min= -2.5907366818620919E-01,sum= 1.7927398128296358E+02
+check[rhogvy_pl ] max= 8.8938663283927272E-02,min= -9.6299249362234787E-02,sum= -9.3645167747781997E-03
+check[rhogvz ] max= 2.089984486710360E-01,min= -1.9461784957198358E-01,sum= 2.9192900535497301E+02
+check[rhogvz_pl ] max= 1.2715227826262342E-03,min= -6.6561175383069663E-04,sum= 2.2146129568583188E-02
+check[rhogw ] max= 1.2675708942695349E-03,min= -4.2678569215437437E-03,sum= -2.65257154444058587E-02
+check[rhogw_pl ] max= 9.9481836269239257E-04,min= -4.2678569215437437E-03,sum= -8.3801034534239580E-02
### Output ###
+check[ddivdx ] max= 3.0131744015374420E-11,min= -5.2988229203876260E-11,sum= -1.6975403236890680E-11
+check[ddivdx_pl ] max= 2.6717761683260969E-11,min= -5.3219224048058505E-11,sum= 4.8275913249408648E-11
+check[ddivdy ] max= 2.8051192990274010E-11,min= -2.3223214708700599E-11,sum= -4.5483269668124264E-11
+check[ddivdy_pl ] max= 4.4686835079822753E-11,min= -4.1286060234480353E-11,sum= 6.8817604250940581E-11
+check[ddivdz ] max= 3.4380463226247279E-11,min= -1.3010875841849350E-11,sum= -7.2052908719075937E-11
+check[ddivdz_pl ] max= 3.2276474319270210E-13,min= -8.5477300588035169E-14,sum= 3.9437219768394675E-12
### Validation : point-by-point diff ###
+check[check_ddivdx ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_ddivdx_pl ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_ddivdy ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_ddivdy_pl ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_ddivdz ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_ddivdz_pl ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
*** Finish kernel

```

Check the lines below “Validation : point-by-point diff” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.3.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in [subsection 2.1.7](#). Note that in this program kernel part is iterated one time.

```
*** Computational Time Report
*** ID=001 : MAIN_dyn_divdamp          T= 0.028 N= 1
*** ID=002 : OPRT3D_divdamp          T= 0.028 N= 1
```

2.4 dyn_vi_rhow_solver

2.4.1 Description

Kernel `dyn_vi_rhow_solver` is taken from the original subroutine `vi_rhow_solver` in *NICAM*. This subrouine is originally defined in `mod_vi`. Subroutine `vi_rhow_solver` is to solve the tridiagonal matrix equations related to the vertical implicit scheme. See [section 1.6](#) for the detail of this calculation.

2.4.2 Discretization and code

Argument lists and local variables definition part of this subroutine is as follows.

```
1  !-----
2  !> Tridiagonal matrix solver
3  subroutine vi_rhow_solver( &
4     rhogw, rhogw_pl, &
5     rhogw0, rhogw0_pl, &
6     preg0, preg0_pl, &
7     rhog0, rhog0_pl, &
8     Srho, Srho_pl, &
9     Sw, Sw_pl, &
10    Spre, Spre_pl, &
11    dt
12    )
13    !$ use omp_lib
14    implicit none
15    real(RP), intent(inout) :: rhogw (ADM_gall ,ADM_kall,ADM_lall ) ! rho*w ( G~1/2 x gam2 ), n+1
16    real(RP), intent(inout) :: rhogw_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
17
18    real(RP), intent(in) :: rhogw0 (ADM_gall ,ADM_kall,ADM_lall ) ! rho*w ( G~1/2 x gam2 )
19    real(RP), intent(in) :: rhogw0_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
20    real(RP), intent(in) :: preg0 (ADM_gall ,ADM_kall,ADM_lall ) ! pressure prime ( G~1/2 x gam2 )
21    real(RP), intent(in) :: preg0_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
22    real(RP), intent(in) :: rhog0 (ADM_gall ,ADM_kall,ADM_lall ) ! rho ( G~1/2 x gam2 )
23    real(RP), intent(in) :: rhog0_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
24    real(RP), intent(in) :: Srho (ADM_gall ,ADM_kall,ADM_lall ) ! source term for rho at the full level
25    real(RP), intent(in) :: Srho_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
26    real(RP), intent(in) :: Sw (ADM_gall ,ADM_kall,ADM_lall ) ! source term for rhow at the half level
27    real(RP), intent(in) :: Sw_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
28    real(RP), intent(in) :: Spre (ADM_gall ,ADM_kall,ADM_lall ) ! source term for pres at the full level
29    real(RP), intent(in) :: Spre_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
30    real(RP), intent(in) :: dt
31
32    real(RP) :: Sall (ADM_gall, ADM_kall)
33    real(RP) :: Sall_pl (ADM_gall_pl,ADM_kall)
34    real(RP) :: beta (ADM_gall )
35    real(RP) :: beta_pl (ADM_gall_pl)
36    real(RP) :: gamma (ADM_gall, ADM_kall)
37    real(RP) :: gamma_pl (ADM_gall_pl,ADM_kall)
38
39    integer :: gall, kmin, kmax, lall
40    real(RP) :: grav
41    real(RP) :: CVovRt2 ! Cv / R / dt**2
42    real(RP) :: alpha
```

```

43
44 integer :: g, k, l
45 integer :: gstr, gend
46 !$ integer :: n_per_thread
47 !$ integer :: n_thread
48 !-----
49

```

Here rhogw is $\rho \times w$ with metric $G^{1/2}\gamma^2$ multiplied at new time step $n + 1$. rhogw0 , preg0 , rhog0 , are $\rho \times w$, pressure, ρ with metric multiplied at time step n , respectively. Srho , Sw , Spre are source term for ρ at the full level, source term for $\rho \times w$ at the half level, source term for pressure at the full level, respectively. Other arguments with suffix $_p1$ are for the pole region. dt is a time step for fast-mode.

Among local variables, alpha is the flag for non-hydrostatic/hydrostatic. In this kernel, set to 1 in `problem_size.inc`.

Main part of this subroutine is as follows.

```

50 call DEBUG_rapstart('____vi_rhov_solver')
51
52 gall = ADM_gall
53 kmin = ADM_kmin
54 kmax = ADM_kmax
55 lall = ADM_lall
56
57 grav = CONST_GRAV
58 CVovRt2 = CONST_CVdry / CONST_Rdry / (dt*dt)
59 alpha = real(NON_HYDRO_ALPHA,kind=RP)
60
61 !$omp parallel default(none),private(g,k,l), &
62 !$omp private(gstr,gend,n_thread,n_per_thread) &
63 !$omp shared(gall,kmin,kmax,lall,rhogw,rhogw0,preg0,rhog0,Srho,Sw,Spre,dt,Sall,beta,gamma,Mu,Mc,Ml, &
64 !$omp GRD_afact,GRD_bfact,GRD_rdgzh,VMTR_GSGAM2H,VMTR_RGAM,VMTR_RGAMH,VMTR_RSGAM2,VMTR_RSGAM2H,grav,alpha, CVovRt2)
65 gstr = 1
66 gend = gall
67 !$ n_thread = omp_get_num_threads()
68 !$ n_per_thread = gall / n_thread + int( 0.5_RP + sign(0.5_RP,mod(gall,n_thread)-0.5_RP) )
69 !$ gstr = n_per_thread * omp_get_thread_num() + 1
70 !$ gend = min( gstr+n_per_thread-1, gall )
71
72 do l = 1, lall
73 ! calc Sall
74 do k = kmin+1, kmax
75 do g = gstr, gend
76 Sall(g,k) = ( ( rhogw0(g,k, l)*alpha + dt * Sw (g,k, l) ) * VMTR_RGAMH (g,k, l)**2 &
77 - ( ( preg0 (g,k, l) + dt * Spre(g,k, l) ) * VMTR_RSGAM2(g,k, l) &
78 - ( preg0 (g,k-1,l) + dt * Spre(g,k-1,l) ) * VMTR_RSGAM2(g,k-1,l) &
79 ) * dt * GRD_rdgzh(k) &
80 - ( ( rhog0 (g,k, l) + dt * Srho(g,k, l) ) * VMTR_RGAM(g,k, l)**2 * GRD_afact(k) &
81 + ( rhog0 (g,k-1,l) + dt * Srho(g,k-1,l) ) * VMTR_RGAM(g,k-1,l)**2 * GRD_bfact(k) &
82 ) * dt * grav &
83 ) * CVovRt2
84
85 enddo
86
87 ! boundary conditions
88 do g = gstr, gend
89 rhogw(g,kmin, l) = rhogw(g,kmin, l) * VMTR_RSGAM2H(g,kmin, l)
90 rhogw(g,kmax+1,l) = rhogw(g,kmax+1,l) * VMTR_RSGAM2H(g,kmax+1,l)
91 Sall (g,kmin+1) = Sall (g,kmin+1) - Ml (g,kmin+1,l) * rhogw(g,kmin, l)
92 Sall (g,kmax ) = Sall (g,kmax ) - Mu(g,kmax, l) * rhogw(g,kmax+1,l)
93 enddo
94
95 !---< solve tri-daigonal matrix >
96
97 ! condition at kmin+1
98 k = kmin+1
99 do g = gstr, gend
100 beta (g) = Mc(g,k,l)
101 rhogw(g,k,l) = Sall(g,k) / beta(g)
102 enddo
103
104 ! forward
105 do k = kmin+2, kmax
106 do g = gstr, gend
107 gamma(g,k) = Mu(g,k-1,l) / beta(g)
108 beta (g) = Mc(g,k,l) - Ml(g,k,l) * gamma(g,k) ! update beta
109 rhogw(g,k,l) = ( Sall(g,k) - Ml(g,k,l) * rhogw(g,k-1,l) ) / beta(g)

```



```

110 enddo
111 enddo
112
113 ! backward
114 do k = kmax-1, kmin+1, -1
115 do g = gstr, gend
116   rhogw(g,k ,1) = rhogw(g,k ,1) - gamma(g,k+1) * rhogw(g,k+1,1)
117   rhogw(g,k+1,1) = rhogw(g,k+1,1) * VMTR_GSGAM2H(g,k+1,1) ! return value ( G~1/2 x gam2 )
118 enddo
119 enddo
120
121 ! boundary treatment
122 do g = gstr, gend
123   rhogw(g,kmin ,1) = rhogw(g,kmin ,1) * VMTR_GSGAM2H(g,kmin ,1)
124   rhogw(g,kmin+1,1) = rhogw(g,kmin+1,1) * VMTR_GSGAM2H(g,kmin+1,1)
125   rhogw(g,kmax+1,1) = rhogw(g,kmax+1,1) * VMTR_GSGAM2H(g,kmax+1,1)
126 enddo
127 enddo
128 !$omp end parallel

```

Inside of the long l -loop(1.72), the first k -loop(1.73) calculates total source term S_{all} . The section after setting the boundary condition at $kmin$ and $kmax$ solves tri-diagonal matrix. Its first part(1.97 to 1.111) is for forward elimination and the second part(1.114 to 1.119) is for backward substitution.

Note that elements of the tridiagonal matrix Mc , Mu and Ml are calculated in advance by other subroutine in the original module, and they are read from input data file in this kernel program.

The last part of this subroutine is for the pole region, doing almost the same process as the normal region.

```

129 if ( ADM_have_pl ) then
130 do l = 1, ADM_lall_pl
131 do k = ADM_kmin+1, ADM_kmax
132 do g = 1, ADM_gall_pl
133   Sall_pl(g,k) = ( ( rhogw0_pl(g,k, 1)*alpha + dt * Sw_pl (g,k, 1) ) * VMTR_RGAMH_pl (g,k, 1)**2           &
134                 - ( ( preg0_pl (g,k, 1) + dt * Spre_pl(g,k, 1) ) * VMTR_RSGAM2_pl(g,k, 1)           &
135                 - ( preg0_pl (g,k-1,1) + dt * Spre_pl(g,k-1,1) ) * VMTR_RSGAM2_pl(g,k-1,1)           &
136                 ) * dt * GRD_rdgzh(k) ) * dt * grav ) * CVovRt2 ) * dt * grav ) * CVovRt2           &
137                 - ( ( rhog0_pl (g,k, 1) + dt * Srho_pl(g,k, 1) ) * VMTR_RGAM_pl(g,k, 1)**2 * GRD_afact(k) &
138                 + ( rhog0_pl (g,k-1,1) + dt * Srho_pl(g,k-1,1) ) * VMTR_RGAM_pl(g,k-1,1)**2 * GRD_bfact(k) &
139                 ) * dt * grav ) * CVovRt2 ) * dt * grav ) * CVovRt2           &
140 enddo
141 enddo
142 enddo
143
144 do g = 1, ADM_gall_pl
145   rhogw_pl(g,ADM_kmin, 1) = rhogw_pl(g,ADM_kmin, 1) * VMTR_RSGAM2H_pl(g,ADM_kmin, 1)
146   rhogw_pl(g,ADM_kmax+1,1) = rhogw_pl(g,ADM_kmax+1,1) * VMTR_RSGAM2H_pl(g,ADM_kmax+1,1)
147   Sall_pl (g,ADM_kmin+1) = Sall_pl (g,ADM_kmin+1) - Ml_pl(g,ADM_kmin+1,1) * rhogw_pl(g,ADM_kmin, 1)
148   Sall_pl (g,ADM_kmax ) = Sall_pl (g,ADM_kmax ) - Mu_pl(g,ADM_kmax, 1) * rhogw_pl(g,ADM_kmax+1,1)
149 enddo
150
151 k = ADM_kmin+1
152 do g = 1, ADM_gall_pl
153   beta_pl (g) = Mc_pl(g,k,1)
154   rhogw_pl(g,k,1) = Sall_pl(g,k) / beta_pl(g)
155 enddo
156
157 do k = ADM_kmin+2, ADM_kmax
158 do g = 1, ADM_gall_pl
159   gamma_pl(g,k) = Mu_pl(g,k-1,1) / beta_pl(g)
160   beta_pl (g) = Mc_pl(g,k,1) - Ml_pl(g,k,1) * gamma_pl(g,k) ! update beta
161   rhogw_pl(g,k,1) = ( Sall_pl(g,k) - Ml_pl(g,k,1) * rhogw_pl(g,k-1,1) ) / beta_pl(g)
162 enddo
163 enddo
164
165 ! backward
166 do k = ADM_kmax-1, ADM_kmin+1, -1
167 do g = 1, ADM_gall_pl
168   rhogw_pl(g,k ,1) = rhogw_pl(g,k ,1) - gamma_pl(g,k+1) * rhogw_pl(g,k+1,1)
169   rhogw_pl(g,k+1,1) = rhogw_pl(g,k+1,1) * VMTR_GSGAM2H_pl(g,k+1,1) ! return value ( G~1/2 x gam2 )
170 enddo
171 enddo
172
173 ! boundary treatment
174 do g = 1, ADM_gall_pl
175   rhogw_pl(g,ADM_kmin ,1) = rhogw_pl(g,ADM_kmin ,1) * VMTR_GSGAM2H_pl(g,ADM_kmin ,1)
176   rhogw_pl(g,ADM_kmin+1,1) = rhogw_pl(g,ADM_kmin+1,1) * VMTR_GSGAM2H_pl(g,ADM_kmin+1,1)
177   rhogw_pl(g,ADM_kmax+1,1) = rhogw_pl(g,ADM_kmax+1,1) * VMTR_GSGAM2H_pl(g,ADM_kmax+1,1)
178 enddo

```

```

179     enddo
180   endif
181
182   call DEBUG_rapend('____vi_rhow_solver')
183
184   return
185 end subroutine vi_rhow_solver

```

2.4.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of \$IAB_SYS=Ubuntu-gnu-mpi case.

```

### Input ###
+check[rhogw_prev ] max= 1.0733675425174699E-14,min= -2.5212277839542070E-18,sum= 2.1443525022460023E-14
+check[rhogw_prev_pl ] max= 1.0733675425174699E-14,min= -3.0929371635479732E-15,sum= 7.6415689931329447E-15
+check[check_rhogw ] max= 6.3321830580406713E-01,min= -5.5759247708875415E-01,sum= 3.9071354372140144E+02
+check[check_rhogw_pl ] max= 9.9023353298473032E-02,min= -7.4852014128754477E-03,sum= 2.1725659627743767E+00
+check[rhogw0 ] max= 1.2675708942695349E-03,min= -4.2678569215437437E-03,sum= -2.6525715444058587E-02
+check[rhogw0_pl ] max= 9.9481836269239257E-04,min= -4.2678569215437437E-03,sum= -8.3801034534239580E-02
+check[prego ] max= 1.4047194007847271E+01,min= -2.6115802085359952E+01,sum= -1.0687083035942905E+02
+check[prego_pl ] max= 1.3572400586711277E+01,min= -2.6115802085359952E+01,sum= 3.4640462790019097E+05
+check[Srho ] max= 3.3317357550311205E-04,min= -3.9691441465327759E-04,sum= -2.7468694177252501E-01
+check[Srho_pl ] max= 4.3441154740222974E-05,min= -2.2770470374092324E-05,sum= -3.8685936849254728E-05
+check[Sw ] max= 2.4651449651180712E-04,min= -3.2516247664737818E-04,sum= -1.6092196259156019E-01
+check[Sw_pl ] max= 2.8812604579719903E-04,min= -5.5442162236740700E-04,sum= -1.1046915056782920E-02
+check[Spre ] max= 3.6234075910381584E+01,min= -3.7498614557267203E+01,sum= -2.4763893276893483E+04
+check[Spre_pl ] max= 4.4863109100507081E+00,min= -2.0915571645010926E+00,sum= 1.0496606720811164E+00
+check[Mc ] max= 1.7037935104756485E+00,min= 0.0000000000000000E+00,sum= 8.2325595392291399E+05
+check[Mc_pl ] max= 1.4569727548844720E+00,min= 2.3177911966477871-310,sum= 6.0165678214473996E+02
+check[Ml ] max= 0.0000000000000000E+00,min= -8.3019108752152282E-01,sum= -3.9659607793632336E+05
+check[Ml_pl ] max= 3.2282782818256430E+02,min= -6.9953788050738130E-01,sum= 3.4604237063825449E+03
+check[Mu ] max= 0.0000000000000000E+00,min= -8.6888851023079583E-01,sum= -4.2616616191494197E+05
+check[Mu_pl ] max= 9.7233873292775215E+01,min= -7.5264063505825385E-01,sum= 8.5169724995286754E+02
### Output ###
+check[rhogw ] max= 6.3321830580406713E-01,min= -5.5759247708875415E-01,sum= 3.9071354372140144E+02
+check[rhogw_pl ] max= 9.9023353298473032E-02,min= -7.4852014128754477E-03,sum= 2.1725659627743767E+00
### Validation : point-by-point diff ###
+check[check_rhogw ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_rhogw_pl ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
*** Finish kernel

```

Check the lines below “Validation : point-by-point diff” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.4.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in [subsection 2.1.7](#). Note that in this program kernel part is iterated one time.

```

*** Computational Time Report
*** ID=001 : MAIN_dyn_vi_rhow_solver          T=    0.016 N=    1
*** ID=002 : ____vi_rhow_solver              T=    0.016 N=    1

```

2.5 dyn_vert_adv_limiter

2.5.1 Description

Kernel `dyn_vert_adv_limiter` is taken from the original subroutine `vertical_limiter_thuburn` in *NICAM*. This subroutine is originally defined in `mod_src_tracer`, that is to contain several subroutines for tracer advection. Subroutine `vertical_limiter_thuburn` is to ensure distribution of tracer quantities’ monotonicity in advection scheme, using the flux limiter proposed by [Thuburn \(1996\)](#). This subroutine is for

vertical advection only and horizontal advection is treated by other subroutine `horizontal_limiter_thuburn`, which is also kernelized in this package (See section 2.7). See section 4. in Tomita et al. (2010) for details of the tracer scheme in NICAM.

2.5.2 Discretization and code

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine vertical_limiter_thuburn( &
2     q_h, q_h_pl, &
3     q,  q_pl,  &
4     d,  d_pl,  &
5     ck, ck_pl )
6     !ESC! use mod_const, only: &
7     !ESC!   CONST_HUGE, &
8     !ESC!   CONST_EPS
9     !ESC! use mod_adm, only: &
10    !ESC!   ADM_have_pl, &
11    !ESC!   ADM_gall, &
12    !ESC!   ADM_gall_pl, &
13    !ESC!   ADM_lall, &
14    !ESC!   ADM_lall_pl, &
15    !ESC!   ADM_kall, &
16    !ESC!   ADM_kmin, &
17    !ESC!   ADM_kmax
18    implicit none
19
20    real(RP), intent(inout) :: q_h (ADM_gall ,ADM_kall,ADM_lall )
21    real(RP), intent(inout) :: q_h_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl)
22    real(RP), intent(in)    :: q  (ADM_gall ,ADM_kall,ADM_lall )
23    real(RP), intent(in)    :: q_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
24    real(RP), intent(in)    :: d  (ADM_gall ,ADM_kall,ADM_lall )
25    real(RP), intent(in)    :: d_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
26    real(RP), intent(in)    :: ck  (ADM_gall ,ADM_kall,ADM_lall ,2)
27    real(RP), intent(in)    :: ck_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl,2)
28
29    real(RP) :: Qout_min_k
30    real(RP) :: Qout_max_k
31    real(RP) :: Qout_min_km1(ADM_gall)
32    real(RP) :: Qout_max_km1(ADM_gall)
33    real(RP) :: Qout_min_pl(ADM_gall_pl,ADM_kall)
34    real(RP) :: Qout_max_pl(ADM_gall_pl,ADM_kall)
35
36    real(RP) :: Qin_minL, Qin_maxL
37    real(RP) :: Qin_minU, Qin_maxU
38    real(RP) :: qnext_min, qnext_max
39    real(RP) :: Cin, Cout
40    real(RP) :: CQin_min, CQin_max
41    real(RP) :: inflagL, inflagU
42    real(RP) :: zerosw
43
44    integer :: gall, kmin, kmax
45    real(RP) :: EPS, BIG
46
47    integer :: g, k, l
48    !-----

```

Here q_h is q at half level of the vertical layer, which modified by the flux limiter. q is q at grid point, ck is Courant number, d is a correction factor derived from an artificial viscosity for the total density. Note that ck has the 4th dimension whose size is 2, which specify lower/upper face, i.e. half integer level.

The first section of the subroutine is as follows.

```

1  call DEBUG_rapstart('___vertical_adv_limiter')
2
3  gall = ADM_gall
4  kmin = ADM_kmin
5  kmax = ADM_kmax
6
7  EPS = CONST_EPS
8  BIG = CONST_HUGE
9
10 do l = 1, ADM_lall
11     !$omp parallel default(none), &
12     !$omp private(g,k,zerosw,inflagL,inflagU,Qin_minL,Qin_minU,Qin_maxL,Qin_maxU, &

```

```

13      !$omp      qnext_min,qnext_max,Cin,Cout,CQin_min,CQin_max,Qout_min_k,Qout_max_k) , &
14      !$omp shared(l,gall,kmin,kmax,q_h,ck,q,d,Qout_min_kml,Qout_max_kml,EPS,BIG)
15
16      !OCL XFILL
17      !$omp do
18      do g = 1, gall
19      k = kmin ! peeling
20
21      inflagL = 0.5_RP - sign(0.5_RP,ck(g,k ,l,1)) ! incoming flux: flag=1
22      inflagU = 0.5_RP + sign(0.5_RP,ck(g,k+1,l,1)) ! incoming flux: flag=1
23
24      Qin_minL = min( q(g,k,l), q(g,k-1,l) ) + ( 1.0_RP-inflagL ) * BIG
25      Qin_minU = min( q(g,k,l), q(g,k+1,l) ) + ( 1.0_RP-inflagU ) * BIG
26      Qin_maxL = max( q(g,k,l), q(g,k-1,l) ) - ( 1.0_RP-inflagL ) * BIG
27      Qin_maxU = max( q(g,k,l), q(g,k+1,l) ) - ( 1.0_RP-inflagU ) * BIG
28
29      qnext_min = min( Qin_minL, Qin_minU, q(g,k,l) )
30      qnext_max = max( Qin_maxL, Qin_maxU, q(g,k,l) )
31
32      Cin      = (      inflagL ) * ck(g,k,l,1) &
33      + (      inflagU ) * ck(g,k,l,2)
34      Cout     = ( 1.0_RP-inflagL ) * ck(g,k,l,1) &
35      + ( 1.0_RP-inflagU ) * ck(g,k,l,2)
36
37      CQin_min = (      inflagL ) * ck(g,k,l,1) * Qin_minL &
38      + (      inflagU ) * ck(g,k,l,2) * Qin_minU
39      CQin_max = (      inflagL ) * ck(g,k,l,1) * Qin_maxL &
40      + (      inflagU ) * ck(g,k,l,2) * Qin_maxU
41
42      zerosw = 0.5_RP - sign(0.5_RP,abs(Cout)-EPS) ! if Cout = 0, sw = 1
43
44      Qout_min_k = ( ( q(g,k,l) - qnext_max ) + qnext_max*(Cin+Cout-d(g,k,l)) - CQin_max ) &
45      / ( Cout + zerosw ) * ( 1.0_RP - zerosw ) &
46      + q(g,k,l) * zerosw
47      Qout_max_k = ( ( q(g,k,l) - qnext_min ) + qnext_min*(Cin+Cout-d(g,k,l)) - CQin_min ) &
48      / ( Cout + zerosw ) * ( 1.0_RP - zerosw ) &
49      + q(g,k,l) * zerosw
50
51      Qout_min_kml(g) = Qout_min_k
52      Qout_max_kml(g) = Qout_max_k
53      enddo
54      !$omp end do
55
56      do k = kmin+1, kmax
57      !OCL XFILL
58      !$omp do
59      do g = 1, gall
60      inflagL = 0.5_RP - sign(0.5_RP,ck(g,k ,l,1)) ! incoming flux: flag=1
61      inflagU = 0.5_RP + sign(0.5_RP,ck(g,k+1,l,1)) ! incoming flux: flag=1
62
63      Qin_minL = min( q(g,k,l), q(g,k-1,l) ) + ( 1.0_RP-inflagL ) * BIG
64      Qin_minU = min( q(g,k,l), q(g,k+1,l) ) + ( 1.0_RP-inflagU ) * BIG
65      Qin_maxL = max( q(g,k,l), q(g,k-1,l) ) - ( 1.0_RP-inflagL ) * BIG
66      Qin_maxU = max( q(g,k,l), q(g,k+1,l) ) - ( 1.0_RP-inflagU ) * BIG
67
68      qnext_min = min( Qin_minL, Qin_minU, q(g,k,l) )
69      qnext_max = max( Qin_maxL, Qin_maxU, q(g,k,l) )
70
71      Cin      = (      inflagL ) * ck(g,k,l,1) &
72      + (      inflagU ) * ck(g,k,l,2)
73      Cout     = ( 1.0_RP-inflagL ) * ck(g,k,l,1) &
74      + ( 1.0_RP-inflagU ) * ck(g,k,l,2)
75
76      CQin_min = (      inflagL ) * ck(g,k,l,1) * Qin_minL &
77      + (      inflagU ) * ck(g,k,l,2) * Qin_minU
78      CQin_max = (      inflagL ) * ck(g,k,l,1) * Qin_maxL &
79      + (      inflagU ) * ck(g,k,l,2) * Qin_maxU
80
81      zerosw = 0.5_RP - sign(0.5_RP,abs(Cout)-EPS) ! if Cout = 0, sw = 1
82
83      Qout_min_k = ( ( q(g,k,l) - qnext_max ) + qnext_max*(Cin+Cout-d(g,k,l)) - CQin_max ) &
84      / ( Cout + zerosw ) * ( 1.0_RP - zerosw ) &
85      + q(g,k,l) * zerosw
86      Qout_max_k = ( ( q(g,k,l) - qnext_min ) + qnext_min*(Cin+Cout-d(g,k,l)) - CQin_min ) &
87      / ( Cout + zerosw ) * ( 1.0_RP - zerosw ) &
88      + q(g,k,l) * zerosw
89
90      q_h(g,k,l) = (      inflagL ) * max( min( q_h(g,k,l), Qout_max_kml(g) ), Qout_min_kml(g) ) &
91      + ( 1.0_RP-inflagL ) * max( min( q_h(g,k,l), Qout_max_k      ), Qout_min_k      )
92
93      Qout_min_kml(g) = Qout_min_k
94      Qout_max_kml(g) = Qout_max_k

```

```

95         enddo
96         !$omp end do
97     enddo
98
99     !$omp end parallel
100 enddo
101

```

In the long l -loop, there seems to be two blocks, but they are almost the same, except that the first one is only for k_{\min} , that is the lowest level, and the second one is the rest of k to the top level. inflagL and inflagU are flag that takes the value 1 if there is an incoming flux to the current layer through the lower/upper face. Q_{in}^* are the smaller/larger values of q at lower/upper neighboring layer, that is meaningful only if inflag at lower/upper is 1. C_{in} and C_{out} are the sum of Courant number at both face of the layer, for example, C_{in} is the sum of ck at lower face and upper face, if both of inflagL and inflagU is 1, which means that there are inflow through both lower/upper face. CQ_{in}^* are the min/max of C_{in} times Q_{in}^* , which specify the minimum/maximum of inflow. Then CQ_{out}^* are calculated. Finally q_h is calculated, which is bounded by Q_{out}^* .

The second section of this subroutine is as follows.

```

1  if ( ADM_have_pl ) then
2      do l = 1, ADM_lall_pl
3
4          do k = ADM_kmin, ADM_kmax
5              do g = 1, ADM_gall_pl
6                  inflagL = 0.5_RP - sign(0.5_RP,ck_pl(g,k ,1,1)) ! incoming flux: flag=1
7                  inflagU = 0.5_RP + sign(0.5_RP,ck_pl(g,k+1,1,1)) ! incoming flux: flag=1
8
9                  Qin_minL = min( q_pl(g,k,1), q_pl(g,k-1,1) ) + ( 1.0_RP-inflagL ) * BIG
10                 Qin_minU = min( q_pl(g,k,1), q_pl(g,k+1,1) ) + ( 1.0_RP-inflagU ) * BIG
11                 Qin_maxL = max( q_pl(g,k,1), q_pl(g,k-1,1) ) - ( 1.0_RP-inflagL ) * BIG
12                 Qin_maxU = max( q_pl(g,k,1), q_pl(g,k+1,1) ) - ( 1.0_RP-inflagU ) * BIG
13
14                 qnext_min = min( Qin_minL, Qin_minU, q_pl(g,k,1) )
15                 qnext_max = max( Qin_maxL, Qin_maxU, q_pl(g,k,1) )
16
17                 Cin      = (      inflagL ) * ( ck_pl(g,k,1,1) ) &
18                     + (      inflagU ) * ( ck_pl(g,k,1,2) )
19                 Cout     = ( 1.0_RP-inflagL ) * ( ck_pl(g,k,1,1) ) &
20                     + ( 1.0_RP-inflagU ) * ( ck_pl(g,k,1,2) )
21
22                 CQin_max = (      inflagL ) * ( ck_pl(g,k,1,1) * Qin_maxL ) &
23                     + (      inflagU ) * ( ck_pl(g,k,1,2) * Qin_maxU )
24                 CQin_min = (      inflagL ) * ( ck_pl(g,k,1,1) * Qin_minL ) &
25                     + (      inflagU ) * ( ck_pl(g,k,1,2) * Qin_minU )
26
27                 zerosw = 0.5_RP - sign(0.5_RP,abs(Cout)-EPS) ! if Cout = 0, sw = 1
28
29                 Qout_min_pl(g,k) = ( ( q_pl(g,k,1) - qnext_max ) + qnext_max*(Cin+Cout-d_pl(g,k,1)) - CQin_max ) &
30                     / ( Cout + zerosw ) * ( 1.0_RP - zerosw ) &
31                     + q_pl(g,k,1) * zerosw
32                 Qout_max_pl(g,k) = ( ( q_pl(g,k,1) - qnext_min ) + qnext_min*(Cin+Cout-d_pl(g,k,1)) - CQin_min ) &
33                     / ( Cout + zerosw ) * ( 1.0_RP - zerosw ) &
34                     + q_pl(g,k,1) * zerosw
35             enddo
36         enddo
37
38         do k = ADM_kmin+1, ADM_kmax
39             do g = 1, ADM_gall_pl
40                 inflagL = 0.5_RP - sign(0.5_RP,ck_pl(g,k,1,1)) ! incoming flux: flag=1
41
42                 q_h_pl(g,k,1) = (      inflagL ) * max( min( q_h_pl(g,k,1), Qout_max_pl(g,k-1) ), Qout_min_pl(g,k-1) ) &
43                     + ( 1.0_RP-inflagL ) * max( min( q_h_pl(g,k,1), Qout_max_pl(g,k) ), Qout_min_pl(g,k) )
44             enddo
45         enddo
46
47     enddo
48 endif
49
50 call DEBUG_rapend ( '___vertical_adv_limiter' )
51
52 return
53 end subroutine vertical_limiter_thuburn

```

This section is for the pole region, and doing almost the same procedure with the regular region.

2.5.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of \$IAB_SYS=Ubuntu-gnu-mpi case.

```
### Input ###
+check[q_h_prev] ] max= 7.2663804548391786E+00,min= 0.0000000000000000E+00,sum= 2.4959084727640115E+06
+check[q_h_prev_pl] ] max= 6.4080655438269076E+00,min= 0.0000000000000000E+00,sum= 1.8117616747458535E+03
+check[check_q_h] ] max= 7.3763287914601054E+00,min= 0.0000000000000000E+00,sum= 2.4996023860691669E+06
+check[check_q_h_pl] ] max= 7.0217121722230473E+00,min= 0.0000000000000000E+00,sum= 1.8122948877569047E+03
+check[q] ] max= 7.3763287914601054E+00,min= 0.0000000000000000E+00,sum= 2.4959084727641600E+06
+check[q_pl] ] max= 7.0217121722230473E+00,min= 0.0000000000000000E+00,sum= 1.8117616747458526E+03
+check[d] ] max= -0.0000000000000000E+00,min= -0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[d_pl] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[ck] ] max= 3.1666977358687308E-02,min= -3.3842023700197510E-02,sum= -3.3888707532669278E+00
+check[ck_pl] ] max= 2.8817336749971920E-02,min= -3.0878557008837175E-02,sum= -2.9483051886829596E-02
### Output ###
+check[q_h] ] max= 7.3763287914601054E+00,min= 0.0000000000000000E+00,sum= 2.4996023860691669E+06
+check[q_h_pl] ] max= 7.0217121722230473E+00,min= 0.0000000000000000E+00,sum= 1.8122948877569047E+03
### Validation : point-by-point diff ###
+check[check_q_h] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_q_h_pl] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
*** Finish kernel
```

Check the lines below “Validation : point-by-point diff” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in reference/ in each kernel program directory, for reference purpose.

2.5.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in subsection 2.1.7. Note that in this program kernel part is iterated one time.

```
*** Computational Time Report
*** ID=001 : MAIN_dyn_vert_adv_limiter      T= 0.032 N= 1
*** ID=002 : ___vertical_adv_limiter        T= 0.032 N= 1
```

2.6 dyn_horiz_adv_flux

2.6.1 Description

Kernel `dyn_horiz_adv_flux` is taken from the original subroutine `horizontal_flux` in *NICAM*. This subroutine is originally defined in `mod_src_tracer`. Subroutine `horizontal_flux` is to calculate horizontal advection term, i.e. horizontal mass flux and mass centroid position of an area, which is used for the estimation of mass flux passing through an edge of control cell during one time step. In *NICAM*, a third order upwind scheme proposed by [Miura \(2007\)](#) is used for the horizontal tracer advection on the icosahedral grid, briefly described in [section 1.5](#). See section 4. in [Tomita et al. \(2010\)](#) for details of tracer scheme in *NICAM*, too.

2.6.2 Discretization and code

Argument lists and local variables definition part of this subroutine is as follows.

```
1 subroutine horizontal_flux( &
2     flx_h, flx_h_pl, &
3     GRD_xc, GRD_xc_pl, &
4     rho, rho_pl, &
5     rhovx, rhovx_pl, &
6     rhovy, rhovy_pl, &
7     rhovz, rhovz_pl, &
8     dt
9     implicit none
```

```

10
11 real(RP), intent(out) :: flx_h (ADM_gall ,ADM_kall,ADM_lall ,6) ! horizontal mass flux
12 real(RP), intent(out) :: flx_h_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl )
13 real(RP), intent(out) :: GRD_xc (ADM_gall ,ADM_kall,ADM_lall ,AI:AJ,XDIR:ZDIR) ! mass centroid position
14 real(RP), intent(out) :: GRD_xc_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl, XDIR:ZDIR)
15 real(RP), intent(in) :: rho (ADM_gall ,ADM_kall,ADM_lall ) ! rho at cell center
16 real(RP), intent(in) :: rho_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
17 real(RP), intent(in) :: rhovx (ADM_gall ,ADM_kall,ADM_lall )
18 real(RP), intent(in) :: rhovx_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
19 real(RP), intent(in) :: rhovy (ADM_gall ,ADM_kall,ADM_lall )
20 real(RP), intent(in) :: rhovy_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
21 real(RP), intent(in) :: rhovz (ADM_gall ,ADM_kall,ADM_lall )
22 real(RP), intent(in) :: rhovz_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl)
23 real(RP), intent(in) :: dt
24
25 real(RP) :: rhot_TI (ADM_gall ) ! rho at cell vertex
26 real(RP) :: rhot_TJ (ADM_gall ) ! rho at cell vertex
27 real(RP) :: rhot_pl (ADM_gall_pl)
28 real(RP) :: rhovxt_TI(ADM_gall )
29 real(RP) :: rhovxt_TJ(ADM_gall )
30 real(RP) :: rhovxt_pl(ADM_gall_pl)
31 real(RP) :: rhovyt_TI(ADM_gall )
32 real(RP) :: rhovyt_TJ(ADM_gall )
33 real(RP) :: rhovyt_pl(ADM_gall_pl)
34 real(RP) :: rhovzt_TI(ADM_gall )
35 real(RP) :: rhovzt_TJ(ADM_gall )
36 real(RP) :: rhovzt_pl(ADM_gall_pl)
37
38 real(RP) :: rhovxt2
39 real(RP) :: rhovyt2
40 real(RP) :: rhovzt2
41 real(RP) :: flux
42 real(RP) :: rrhoa2
43
44 integer :: gmin, gmax, kall, iall
45 real(RP) :: EPS
46
47 integer :: ij
48 integer :: ip1j, ijp1, ip1jp1
49 integer :: im1j, ijm1
50
51 integer :: i, j, k, l, n, v
52 !-----

```

Output variable `flx_h` is horizontal mass flux and `GRD_xc` is the spatial position C_i in Figure 1.5(b). Note that `flx_h` has 4th dimension whose size is 6, specifying 6 edges of hexagon control volume. Similarly, `GRD_xc` has 4th dimension ranged (AI:AJ) and 5th dimension ranged (XDIR:ZDIR), the former specifies three arc points and the latter specifies 3 coordinates of the position. Input variable `rho`, `rhovx`, `rhovy` and `rhovz` are ρ , ρv_x , ρv_y , and ρv_z at cell center with metrics multiplied, respectively. Other arguments with suffix `_pl` are for the pole region.

Among local variables, `rhot_TI`, `rhot_TJ` are interpolated ρ at the gravitational center of downward triangle and upward triangle, respectively. Other variables `rho*` with suffix `_TI`, `_TJ` are the same.

The first half of main part is as follows.

```

53 call DEBUG_rapstart('___horizontal_adv_flux')
54
55 gmin = ADM_gmin
56 gmax = ADM_gmax
57 kall = ADM_kall
58 iall = ADM_gall_1d
59
60 EPS = CONST_EPS
61
62 do l = 1, ADM_lall
63   !$omp parallel default(none), &
64   !$omp private(i,j,k,ij,ip1j,ip1jp1,ijp1,im1j,ijm1, &
65   !$omp rrhoa2,rhovxt2,rhovyt2,rhovzt2,flux), &
66   !$omp shared(l,ADM_have_sgp,gmin,gmax,kall,iall,rho,rhovx,rhovy,rhovz,flx_h,dt, &
67   !$omp rhot_TI,rhovxt_TI,rhovyt_TI,rhovzt_TI,rhot_TJ,rhovxt_TJ,rhovyt_TJ,rhovzt_TJ, &
68   !$omp GRD_xc,GRD_xr,GMTR_p,GMTR_t,GMTR_a,EPS)
69   do k = 1, kall
70
71     ! (i,j), (i+1,j)
72     !$omp do
73     do j = gmin-1, gmax
74     do i = gmin-1, gmax

```

```

75     ij      = (j-1)*iall + i
76     ip1j   = ij + 1
77
78     rhot_TI (ij) = rho (ij ,k,l) * GMTR_t(ij,KO,1,TI,W1) &
79               + rho (ip1j,k,l) * GMTR_t(ij,KO,1,TI,W2)
80     rhovxt_TI(ij) = rhovx(ij ,k,l) * GMTR_t(ij,KO,1,TI,W1) &
81               + rhovx(ip1j,k,l) * GMTR_t(ij,KO,1,TI,W2)
82     rhovyt_TI(ij) = rhovy(ij ,k,l) * GMTR_t(ij,KO,1,TI,W1) &
83               + rhovy(ip1j,k,l) * GMTR_t(ij,KO,1,TI,W2)
84     rhovzt_TI(ij) = rhovz(ij ,k,l) * GMTR_t(ij,KO,1,TI,W1) &
85               + rhovz(ip1j,k,l) * GMTR_t(ij,KO,1,TI,W2)
86
87     rhot_TJ (ij) = rho (ij ,k,l) * GMTR_t(ij,KO,1,TJ,W1)
88     rhovxt_TJ(ij) = rhovx(ij ,k,l) * GMTR_t(ij,KO,1,TJ,W1)
89     rhovyt_TJ(ij) = rhovy(ij ,k,l) * GMTR_t(ij,KO,1,TJ,W1)
90     rhovzt_TJ(ij) = rhovz(ij ,k,l) * GMTR_t(ij,KO,1,TJ,W1)
91
92     enddo
93     enddo
94     !$omp end do
95
96     ! (i,j+1), (i+1,j+1)
97     !$omp do
98     do j = gmin-1, gmax
99     do i = gmin-1, gmax
100        ij      = (j-1)*iall + i
101        ip1j1  = ij + iall
102        ip1jp1 = ij + iall + 1
103
104        rhot_TI (ij) = rhot_TI (ij) + rho (ip1jp1,k,l) * GMTR_t(ij,KO,1,TI,W3)
105        rhovxt_TI(ij) = rhovxt_TI(ij) + rhovx(ip1jp1,k,l) * GMTR_t(ij,KO,1,TI,W3)
106        rhovyt_TI(ij) = rhovyt_TI(ij) + rhovy(ip1jp1,k,l) * GMTR_t(ij,KO,1,TI,W3)
107        rhovzt_TI(ij) = rhovzt_TI(ij) + rhovz(ip1jp1,k,l) * GMTR_t(ij,KO,1,TI,W3)
108
109        rhot_TJ (ij) = rhot_TJ (ij) + rho (ip1jp1,k,l) * GMTR_t(ij,KO,1,TJ,W2) &
110               + rho (ip1j1 ,k,l) * GMTR_t(ij,KO,1,TJ,W3)
111        rhovxt_TJ(ij) = rhovxt_TJ(ij) + rhovx(ip1jp1,k,l) * GMTR_t(ij,KO,1,TJ,W2) &
112               + rhovx(ip1j1 ,k,l) * GMTR_t(ij,KO,1,TJ,W3)
113        rhovyt_TJ(ij) = rhovyt_TJ(ij) + rhovy(ip1jp1,k,l) * GMTR_t(ij,KO,1,TJ,W2) &
114               + rhovy(ip1j1 ,k,l) * GMTR_t(ij,KO,1,TJ,W3)
115        rhovzt_TJ(ij) = rhovzt_TJ(ij) + rhovz(ip1jp1,k,l) * GMTR_t(ij,KO,1,TJ,W2) &
116               + rhovz(ip1j1 ,k,l) * GMTR_t(ij,KO,1,TJ,W3)
117
118     enddo
119     enddo
120     !$omp end do
121
122     if ( ADM_have_sgp(1) ) then
123         !$omp master
124         j = gmin-1
125         i = gmin-1
126
127         ij      = (j-1)*iall + i
128         ip1j   = ij + 1
129
130         rhot_TI (ij) = rhot_TJ (ip1j)
131         rhovxt_TI(ij) = rhovxt_TJ(ip1j)
132         rhovyt_TI(ij) = rhovyt_TJ(ip1j)
133         rhovzt_TI(ij) = rhovzt_TJ(ip1j)
134         !$omp end master
135     endif

```

In long l -loop and k -loop, first part calculate ρ , ρv_x , ρv_y , ρv_z at two center points of triangle at TI and TJ. Note that two triangular points are surrounded by 4 grid points (i, j) , $(i + 1, j)$, $(i + 1, j + 1)$ and $(i, j + 1)$. The first i, j -double loop(1.73-) calculates contribution from the former two grid points, and the second i, j -double loop(1.97-) does from the latter two grid points. IF clause from l.120 is treatment for the singular point. GMTR_t is the metrics for triangle linear interpolation from three triangular vertices to gravitational center point of triangle. In original *NICAM*, this array is defined as GMTR_T_var in module mod_gmtr. In this kernel program, this is read from input data file. Note that TI, TJ, W1, W2 and W3 are not loop index but constant defined in problem_size.inc, those are originally defined in mod_gmtr in *NICAM*.

The second half of main part is as follows.

```

135     !--- calculate flux and mass centroid position
136
137     !OCL XFILL
138     !$omp do
139     do j = 1, iall

```



```

140     do i = 1, iall
141         if ( i < gmin .OR. i > gmax &
142             .OR. j < gmin .OR. j > gmax ) then
143             ij = (j-1)*iall + i
144
145             flx_h(ij,k,l,1) = 0.0_RP
146             flx_h(ij,k,l,2) = 0.0_RP
147             flx_h(ij,k,l,3) = 0.0_RP
148             flx_h(ij,k,l,4) = 0.0_RP
149             flx_h(ij,k,l,5) = 0.0_RP
150             flx_h(ij,k,l,6) = 0.0_RP
151
152             GRD_xc(ij,k,l,AI ,XDIR) = 0.0_RP
153             GRD_xc(ij,k,l,AI ,YDIR) = 0.0_RP
154             GRD_xc(ij,k,l,AI ,ZDIR) = 0.0_RP
155             GRD_xc(ij,k,l,AIJ,XDIR) = 0.0_RP
156             GRD_xc(ij,k,l,AIJ,YDIR) = 0.0_RP
157             GRD_xc(ij,k,l,AIJ,ZDIR) = 0.0_RP
158             GRD_xc(ij,k,l,AJ ,XDIR) = 0.0_RP
159             GRD_xc(ij,k,l,AJ ,YDIR) = 0.0_RP
160             GRD_xc(ij,k,l,AJ ,ZDIR) = 0.0_RP
161         endif
162     enddo
163 enddo
164 !$omp end do
165
166 !$omp do
167 do j = gmin , gmax
168 do i = gmin-1, gmax
169     ij = (j-1)*iall + i
170     ip1j = ij + 1
171     ijm1 = ij - iall
172
173     rrhoa2 = 1.0_RP / max( rhot_TJ(ijm1) + rhot_TI(ij), EPS ) ! doubled
174     rhovxt2 = rhovxt_TJ(ijm1) + rhovxt_TI(ij)
175     rhovyt2 = rhovyt_TJ(ijm1) + rhovyt_TI(ij)
176     rhovzt2 = rhovzt_TJ(ijm1) + rhovzt_TI(ij)
177
178     flux = 0.5_RP * ( rhovxt2 * GMTR_a(ij,k0,l,AI ,HNY) &
179                     + rhovyt2 * GMTR_a(ij,k0,l,AI ,HNY) &
180                     + rhovzt2 * GMTR_a(ij,k0,l,AI ,HNZ) )
181
182     flx_h(ij ,k,l,1) = flux * GMTR_p(ij ,k0,l,P_RAREA) * dt
183     flx_h(ip1j,k,l,4) = -flux * GMTR_p(ip1j,k0,l,P_RAREA) * dt
184
185     GRD_xc(ij,k,l,AI,XDIR) = GRD_xr(ij,KO,l,AI,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5_RP
186     GRD_xc(ij,k,l,AI,YDIR) = GRD_xr(ij,KO,l,AI,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5_RP
187     GRD_xc(ij,k,l,AI,ZDIR) = GRD_xr(ij,KO,l,AI,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5_RP
188 enddo
189 enddo
190 !$omp end do
191
192 !$omp do
193 do j = gmin-1, gmax
194 do i = gmin-1, gmax
195     ij = (j-1)*iall + i
196     ip1jp1 = ij + iall + 1
197
198     rrhoa2 = 1.0_RP / max( rhot_TI(ij) + rhot_TJ(ij), EPS ) ! doubled
199     rhovxt2 = rhovxt_TI(ij) + rhovxt_TJ(ij)
200     rhovyt2 = rhovyt_TI(ij) + rhovyt_TJ(ij)
201     rhovzt2 = rhovzt_TI(ij) + rhovzt_TJ(ij)
202
203     flux = 0.5_RP * ( rhovxt2 * GMTR_a(ij,k0,l,AIJ,HNY) &
204                     + rhovyt2 * GMTR_a(ij,k0,l,AIJ,HNY) &
205                     + rhovzt2 * GMTR_a(ij,k0,l,AIJ,HNZ) )
206
207     flx_h(ij ,k,l,2) = flux * GMTR_p(ij ,k0,l,P_RAREA) * dt
208     flx_h(ip1jp1,k,l,5) = -flux * GMTR_p(ip1jp1,k0,l,P_RAREA) * dt
209
210     GRD_xc(ij,k,l,AIJ,XDIR) = GRD_xr(ij,KO,l,AIJ,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5_RP
211     GRD_xc(ij,k,l,AIJ,YDIR) = GRD_xr(ij,KO,l,AIJ,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5_RP
212     GRD_xc(ij,k,l,AIJ,ZDIR) = GRD_xr(ij,KO,l,AIJ,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5_RP
213 enddo
214 enddo
215 !$omp end do
216
217 !$omp do
218 do j = gmin-1, gmax
219 do i = gmin , gmax
220     ij = (j-1)*iall + i
221     ipj1 = ij + iall

```

```

222     im1j = ij - 1
223
224     rrhoa2 = 1.0_RP / max( rhot_TJ(ij) + rhot_TI(im1j), EPS ) ! doubled
225     rhovxt2 = rhovxt_TJ(ij) + rhovxt_TI(im1j)
226     rhovyt2 = rhovyt_TJ(ij) + rhovyt_TI(im1j)
227     rhovzt2 = rhovzt_TJ(ij) + rhovzt_TI(im1j)
228
229     flux = 0.5_RP * ( rhovxt2 * GMTR_a(ij,k0,l,AJ ,HNX) &
230                   + rhovyt2 * GMTR_a(ij,k0,l,AJ ,HNY) &
231                   + rhovzt2 * GMTR_a(ij,k0,l,AJ ,HNZ) )
232
233     flx_h(ij ,k,l,3) = flux * GMTR_p(ij ,k0,l,P_RAREA) * dt
234     flx_h(ijp1,k,l,6) = -flux * GMTR_p(ijp1,k0,l,P_RAREA) * dt
235
236     GRD_xc(ij,k,l,AJ,XDIR) = GRD_xr(ij,KO,l,AJ,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5_RP
237     GRD_xc(ij,k,l,AJ,YDIR) = GRD_xr(ij,KO,l,AJ,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5_RP
238     GRD_xc(ij,k,l,AJ,ZDIR) = GRD_xr(ij,KO,l,AJ,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5_RP
239 enddo
240 enddo
241 !$omp end do
242
243 if ( ADM_have_sgp(1) ) then
244     !$omp master
245     j = gmin
246     i = gmin
247
248     ij = (j-1)*iall + i
249
250     flx_h(ij,k,l,6) = 0.0_RP
251     !$omp end master
252 endif
253
254 enddo
255 !$omp end parallel
256 enddo
257

```

There are 4 i, j -double loops in the long k and l loop continued from previous section. After setting halo region as 0.0, each 3 loops calculates flx_h and GRD_xc at each 3 arc points specified by AI, AIJ, AJ. Here $GMTR_a$ is the metrics for the arc points (the normal vector on the arc point). In original *NICAM*, the array is defined as $GMTR_A_var$ in module `mod_gmtr`. In this kernel program, this is read from input data file. Similarly, $GMTR_p$ is the metrics for grid points (the reciprocal number of the area of the control cell).

The last part is for the pole region, doing almost same calculation with the normal region.

```

258 if ( ADM_have_pl ) then
259     n = ADM_gslf_pl
260
261     do l = 1, ADM_lall_pl
262     do k = 1, ADM_kall
263
264         do v = ADM_gmin_pl, ADM_gmax_pl
265             ij = v
266             ijp1 = v + 1
267             if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
268
269             rhot_pl (v) = rho_pl (n ,k,l) * GMTR_t_pl(ij,KO,l,W1) &
270                   + rho_pl (ij ,k,l) * GMTR_t_pl(ij,KO,l,W2) &
271                   + rho_pl (ijp1,k,l) * GMTR_t_pl(ij,KO,l,W3) &
272             rhovxt_pl(v) = rhovx_pl(n ,k,l) * GMTR_t_pl(ij,KO,l,W1) &
273                   + rhovx_pl(ij ,k,l) * GMTR_t_pl(ij,KO,l,W2) &
274                   + rhovx_pl(ijp1,k,l) * GMTR_t_pl(ij,KO,l,W3) &
275             rhovyt_pl(v) = rhovy_pl(n ,k,l) * GMTR_t_pl(ij,KO,l,W1) &
276                   + rhovy_pl(ij ,k,l) * GMTR_t_pl(ij,KO,l,W2) &
277                   + rhovy_pl(ijp1,k,l) * GMTR_t_pl(ij,KO,l,W3) &
278             rhovzt_pl(v) = rhovz_pl(n ,k,l) * GMTR_t_pl(ij,KO,l,W1) &
279                   + rhovz_pl(ij ,k,l) * GMTR_t_pl(ij,KO,l,W2) &
280                   + rhovz_pl(ijp1,k,l) * GMTR_t_pl(ij,KO,l,W3) &
281         enddo
282
283         do v = ADM_gmin_pl, ADM_gmax_pl
284             ij = v
285             ijm1 = v - 1
286             if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl
287
288             rrhoa2 = 1.0_RP / max( rhot_pl(ijm1) + rhot_pl(ij), EPS ) ! doubled
289             rhovxt2 = rhovxt_pl(ijm1) + rhovxt_pl(ij)
290             rhovyt2 = rhovyt_pl(ijm1) + rhovyt_pl(ij)

```

```

291      rhovzt2 = rhovzt_pl(ijm1) + rhovzt_pl(ij)
292
293      flux = 0.5_RP * ( rhovxt2 * GMTR_a_pl(ij,KO,1,HNX) &
294                    + rhovyt2 * GMTR_a_pl(ij,KO,1,HNY) &
295                    + rhovzt2 * GMTR_a_pl(ij,KO,1,HNZ) )
296
297      flx_h_pl(v,k,l) = flux * GMTR_p_pl(n,KO,1,P_RAREA) * dt
298
299      GRD_xc_pl(v,k,l,XDIR) = GRD_xr_pl(v,KO,1,XDIR) - rhovxt2 * rrhoa2 * dt * 0.5_RP
300      GRD_xc_pl(v,k,l,YDIR) = GRD_xr_pl(v,KO,1,YDIR) - rhovyt2 * rrhoa2 * dt * 0.5_RP
301      GRD_xc_pl(v,k,l,ZDIR) = GRD_xr_pl(v,KO,1,ZDIR) - rhovzt2 * rrhoa2 * dt * 0.5_RP
302  enddo
303
304  enddo
305  enddo
306  endif
307
308  call DEBUG_rapend ('___horizontal_adv_flux')
309
310  return
311  end subroutine horizontal_flux

```

2.6.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of \$IAB_SYS=Ubuntu-gnu-mpi case.

```

### Input ###
+check[flx_h] ] max= 1.5610710092943020E-01,min= -1.5670677438306818E-01,sum= 2.5627807148368471E+00
+check[flx_h_pl] ] max= 1.1002449397299904E-01,min= -1.0379423223736732E-01,sum= 2.3428578735122479E-02
+check[GRD_xc] ] max= 6.3711740974481208E+06,min= -2.8421843880731151E+06,sum= 2.0008127660775812E+13
+check[GRD_xc_pl] ] max= 6.3711707368004546E+06,min= -6.3711702446065033E+06,sum= -5.2402711763852555E+05
+check[rhog_mean] ] max= 1.3664531579209602E+00,min= 3.0688241919459083E-02,sum= 3.0079615977439255E+05
+check[rhog_mean_pl] ] max= 1.3664531579209602E+00,min= 3.0688131204860365E-02,sum= 2.1686973266477614E+02
+check[rhogvx] ] max= 2.4807620495454326E+01,min= -2.6743430307618826E+01,sum= -7.2284486916298035E+05
+check[rhogvx_pl] ] max= 2.3291561831945440E+01,min= -1.7023283571413913E+01,sum= 7.6579748418940346E+02
+check[rhogvy] ] max= 3.6016622784129858E+01,min= -3.3418704823922688E+01,sum= 8.9473900044437428E+05
+check[rhogvy_pl] ] max= 8.7323000593655760E+00,min= -9.4486634372500102E+00,sum= 1.2151517834227268E+03
+check[rhogvz] ] max= 2.3202617358688446E+01,min= -2.4977800359078525E+01,sum= 1.1280540293791931E+05
+check[rhogvz_pl] ] max= 1.6015604765102806E-01,min= -1.3081853271482025E-01,sum= 1.6285945760677518E+00
+check[GRD_xr] ] max= 4.0592444288400000E+13,min= -9.9998999999999996E+30,sum= -1.5719842799999647E+34
+check[GRD_xr_pl] ] max= 6.3711570243787766E+06,min= -9.9998999999999996E+30,sum= -5.9999399999999993E+31
+check[GMTR_p] ] max= 3.1937623126446486E+09,min= -1.0000000000000000E+00,sum= 5.2568771941580258E+13
+check[GMTR_p_pl] ] max= 2.6945919723895960E+09,min= -1.0000000000000000E+00,sum= 3.0761871439307461E+10
+check[GMTR_t] ] max= 1.598890413474729E+09,min= 0.0000000000000000E+00,sum= 5.1787428410119117E+13
+check[GMTR_t_pl] ] max= 1.9722549493151660E+09,min= 0.0000000000000000E+00,sum= 1.9722549503025471E+10
+check[GMTR_a] ] max= 6.3362880879999531E+04,min= -6.0857794015098916E+04,sum= 7.9258961023823655E+08
+check[GMTR_a_pl] ] max= 5.7873883883004550E+04,min= -5.7872753043726123E+04,sum= -4.3064210331067443E-07
### Output ###
+check[flx_h] ] max= 1.5610710092943020E-01,min= -1.5670677438306818E-01,sum= 2.5627807148368471E+00
+check[flx_h_pl] ] max= 1.1002449397299904E-01,min= -1.0379423223736732E-01,sum= 2.3428578735122479E-02
+check[GRD_xc] ] max= 6.3711740974481208E+06,min= -2.8421843880731151E+06,sum= 2.0008127660775812E+13
+check[GRD_xc_pl] ] max= 6.3711707368004546E+06,min= -6.3711702446065033E+06,sum= -5.2402711763852555E+05
### Validation : point-by-point diff ###
+check[check_flx_h] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_flx_h_pl] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_GRD_xc] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_GRD_xc_pl] ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
*** Finish kernel

```

Check the lines below “Validation : point-by-point diff” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in reference/ in each kernel program directory, for reference purpose.

2.6.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in [subsection 2.1.7](#). Note that in this program kernel part is iterated one time.

```

*** Computational Time Report
*** ID=001 : MAIN_dyn_horiz_adv_flux      T=    0.040 N=    1
*** ID=002 : ____horizontal_adv_flux     T=    0.040 N=    1

```

2.7 dyn_horiz_adv_limiter

2.7.1 Description

Kernel `dyn_horiz_adv_limiter` is taken from the original subroutine `horizontal_limiter_thuburn` in *NICAM*. This subroutine is originally defined in `mod_src_tracer`, that is to contain several subroutines for tracer advection. Subroutine `horizontal_limiter_thuburn` is to ensure distribution of tracer quantities' monotonicity in advection scheme, using the flux limiter proposed by [Thuburn \(1996\)](#). This subroutine is for horizontal advection only and vertical advection is treated by other subroutine `vertical_limiter_thuburn`, which is also kernelized in this package (See [section 2.5](#)). In *NICAM*, a third order upwind scheme proposed by [Miura \(2007\)](#) is used for the horizontal tracer advection on the icosahedral grid. See [section 4.](#) in [Tomita et al. \(2010\)](#) for details of the tracer scheme in *NICAM*, too.

2.7.2 Discretization and code

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine horizontal_limiter_thuburn( &
2     q_a,    q_a_pl,  &
3     q,     q_pl,    &
4     d,     d_pl,    &
5     ch,    ch_pl,   &
6     cmask, cmask_pl, &
7     Qout_prev, Qout_prev_pl, & ! KERNEL
8     Qout_post, Qout_post_pl ) ! KERNEL
9     !ESC! use mod_const, only: &
10    !ESC!   CONST_HUGE, &
11    !ESC!   CONST_EPS
12    !ESC! use mod_adm, only: &
13    !ESC!   ADM_have_pl, &
14    !ESC!   ADM_have_sgp, &
15    !ESC!   ADM_lall, &
16    !ESC!   ADM_lall_pl, &
17    !ESC!   ADM_gall, &
18    !ESC!   ADM_gall_pl, &
19    !ESC!   ADM_kall, &
20    !ESC!   ADM_gall_id, &
21    !ESC!   ADM_gmin, &
22    !ESC!   ADM_gmax, &
23    !ESC!   ADM_gslf_pl, &
24    !ESC!   ADM_gmin_pl, &
25    !ESC!   ADM_gmax_pl
26    !ESC! use mod_comm, only: &
27    !ESC!   COMM_data_transfer
28    !ESC!   implicit none
29
30    real(RP), intent(inout) :: q_a    (ADM_gall ,ADM_kall,ADM_lall ,6)
31    real(RP), intent(inout) :: q_a_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl )
32    real(RP), intent(in)   :: q      (ADM_gall ,ADM_kall,ADM_lall )
33    real(RP), intent(in)   :: q_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl )
34    real(RP), intent(in)   :: d      (ADM_gall ,ADM_kall,ADM_lall )
35    real(RP), intent(in)   :: d_pl   (ADM_gall_pl,ADM_kall,ADM_lall_pl )
36    real(RP), intent(in)   :: ch     (ADM_gall ,ADM_kall,ADM_lall ,6)
37    real(RP), intent(in)   :: ch_pl  (ADM_gall_pl,ADM_kall,ADM_lall_pl )
38    real(RP), intent(in)   :: cmask  (ADM_gall ,ADM_kall,ADM_lall ,6)
39    real(RP), intent(in)   :: cmask_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl )
40    real(RP), intent(out)  :: Qout_prev (ADM_gall ,ADM_kall,ADM_lall ,2) ! before communication (for check)
41    real(RP), intent(out)  :: Qout_prev_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,2) !
42    real(RP), intent(in)   :: Qout_post (ADM_gall ,ADM_kall,ADM_lall ,2) ! after communication (additional input)
43    real(RP), intent(in)   :: Qout_post_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,2) !
44
45    real(RP) :: q_min_AI, q_min_AIJ, q_min_AJ, q_min_pl
46    real(RP) :: q_max_AI, q_max_AIJ, q_max_AJ, q_max_pl
47
48    real(RP) :: qnext_min , qnext_min_pl
49    real(RP) :: qnext_max , qnext_max_pl

```

```

50 real(RP) :: Cin_sum      , Cin_sum_pl
51 real(RP) :: Cout_sum    , Cout_sum_pl
52 real(RP) :: CQin_max_sum, CQin_max_sum_pl
53 real(RP) :: CQin_min_sum, CQin_min_sum_pl
54
55 integer, parameter :: I_min = 1
56 integer, parameter :: I_max = 2
57 real(RP) :: Qin (ADM_gall ,ADM_kall,ADM_lall ,2,6)
58 real(RP) :: Qin_pl (ADM_gall_pl,ADM_kall,ADM_lall_pl,2,2)
59 real(RP) :: Qout (ADM_gall ,ADM_kall,ADM_lall ,2 )
60 real(RP) :: Qout_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,2 )
61
62 real(RP) :: ch_masked1
63 real(RP) :: ch_masked2
64 real(RP) :: ch_masked3
65 real(RP) :: ch_masked4
66 real(RP) :: ch_masked5
67 real(RP) :: ch_masked6
68 real(RP) :: ch_masked
69 real(RP) :: zerosw
70
71 integer :: gmin, gmax, kall, iall
72 real(RP) :: EPS, BIG
73
74 integer :: ij
75 integer :: ip1j, ijp1, ip1jp1, ip2jp1
76 integer :: im1j, ijm1
77
78 integer :: i, j, k, l, n, v
79 !-----
80

```

q_a is q at the edge of hexagonal control cell, which modified by the flux limiter. Note that q_a has 4-th dimension and its size is 6, that specifies 6 edges of hexagon control volume. q is q at grid point, d is a correction factor derived from an artificial viscosity for the total density. ch is Courant number, $cmask$ is upwind direction mask. In original subroutine in NICAM, $Qout_prev$ and $Qout_post$ are not exist. These additional arguments is prepared to avoid halo communication, which appeared in the middle of this scheme. The detail is discussed below.

The first section of the subroutine is as follows.

```

81 call DEBUG_rapstart('___horizontal_adv_limiter')
82
83 gmin = ADM_gmin
84 gmax = ADM_gmax
85 kall = ADM_kall
86 iall = ADM_gall_1d
87
88 EPS = CONST_EPS
89 BIG = CONST_HUGE
90
91 do l = 1, ADM_lall
92   !$omp parallel default(none), &
93   !$omp private(i,j,k,ij,ip1j,ip1jp1,ijp1,im1j,ijm1,ip2jp1, &
94   !$omp q_min_AI,q_min_AIJ,q_min_AJ,q_max_AI,q_max_AIJ,q_max_AJ,zerosw, &
95   !$omp ch_masked1,ch_masked2,ch_masked3,ch_masked4,ch_masked5,ch_masked6, &
96   !$omp qnext_min,qnext_max,Cin_sum,Cout_sum,CQin_min_sum,CQin_max_sum), &
97   !$omp shared(l,ADM_have_sgp,gmin,gmax,kall,iall,q,cmask,d,ch,Qin,Qout,EPS,BIG)
98   do k = 1, kall
99     !---< (i) define inflow bounds, eq.(32)&(33) >---
100   !OCL XFILL
101     !$omp do
102       do j = gmin-1, gmax
103         do i = gmin-1, gmax
104           ij = (j-1)*iall + i
105           ip1j = ij + 1
106           ip1jp1 = ij + iall + 1
107           ijp1 = ij + iall
108           im1j = ij - 1
109           ijm1 = ij - iall
110
111           im1j = max( im1j , 1 )
112           ijm1 = max( ijm1 , 1 )
113
114           q_min_AI = min( q(ij,k,l), q(ijm1,k,l), q(ip1j,k,l), q(ip1jp1,k,l) )
115           q_max_AI = max( q(ij,k,l), q(ijm1,k,l), q(ip1j,k,l), q(ip1jp1,k,l) )
116           q_min_AIJ = min( q(ij,k,l), q(ip1j,k,l), q(ip1jp1,k,l), q(ijp1,k,l) )
117           q_max_AIJ = max( q(ij,k,l), q(ip1j,k,l), q(ip1jp1,k,l), q(ijp1,k,l) )

```

```

118     q_min_AJ = min( q(ij,k,l), q(ip1jp1,k,l), q(ijp1,k,l), q(im1j,k,l) )
119     q_max_AJ = max( q(ij,k,l), q(ip1jp1,k,l), q(ijp1,k,l), q(im1j,k,l) )
120
121     Qin(ij,    k,l,I_min,1) = (      cmask(ij,k,l,1) ) * q_min_AI &
122                               + ( 1.0_RP-cmask(ij,k,l,1) ) * BIG
123     Qin(ip1j,  k,l,I_min,4) = (      cmask(ij,k,l,1) ) * BIG      &
124                               + ( 1.0_RP-cmask(ij,k,l,1) ) * q_min_AI
125     Qin(ij,    k,l,I_max,1) = (      cmask(ij,k,l,1) ) * q_max_AI &
126                               + ( 1.0_RP-cmask(ij,k,l,1) ) * (-BIG)
127     Qin(ip1j,  k,l,I_max,4) = (      cmask(ij,k,l,1) ) * (-BIG) &
128                               + ( 1.0_RP-cmask(ij,k,l,1) ) * q_max_AI
129
130     Qin(ij,    k,l,I_min,2) = (      cmask(ij,k,l,2) ) * q_min_AIJ &
131                               + ( 1.0_RP-cmask(ij,k,l,2) ) * BIG
132     Qin(ip1jp1,k,l,I_min,5) = (      cmask(ij,k,l,2) ) * BIG      &
133                               + ( 1.0_RP-cmask(ij,k,l,2) ) * q_min_AIJ
134     Qin(ij,    k,l,I_max,2) = (      cmask(ij,k,l,2) ) * q_max_AIJ &
135                               + ( 1.0_RP-cmask(ij,k,l,2) ) * (-BIG)
136     Qin(ip1jp1,k,l,I_max,5) = (      cmask(ij,k,l,2) ) * (-BIG) &
137                               + ( 1.0_RP-cmask(ij,k,l,2) ) * q_max_AIJ
138
139     Qin(ij,    k,l,I_min,3) = (      cmask(ij,k,l,3) ) * q_min_AJ &
140                               + ( 1.0_RP-cmask(ij,k,l,3) ) * BIG
141     Qin(ijp1,  k,l,I_min,6) = (      cmask(ij,k,l,3) ) * BIG      &
142                               + ( 1.0_RP-cmask(ij,k,l,3) ) * q_min_AJ
143     Qin(ij,    k,l,I_max,3) = (      cmask(ij,k,l,3) ) * q_max_AJ &
144                               + ( 1.0_RP-cmask(ij,k,l,3) ) * (-BIG)
145     Qin(ijp1,  k,l,I_max,6) = (      cmask(ij,k,l,3) ) * (-BIG) &
146                               + ( 1.0_RP-cmask(ij,k,l,3) ) * q_max_AJ
147
148     enddo
149     enddo
150     !$omp end do
151
152     if ( ADM_have_sgp(1) ) then
153         !$omp master
154         j = gmin-1
155         i = gmin-1
156
157         ij      = (j-1)*iall + i
158         ip1j    = ij + iall
159         ip1jp1  = ij + iall + 1
160         ip2jp1  = ij + iall + 2
161
162         q_min_AIJ = min( q(ij,k,l), q(ip1jp1,k,l), q(ip2jp1,k,l), q(ijp1,k,l) )
163         q_max_AIJ = max( q(ij,k,l), q(ip1jp1,k,l), q(ip2jp1,k,l), q(ijp1,k,l) )
164
165         Qin(ij,    k,l,I_min,2) = (      cmask(ij,k,l,2) ) * q_min_AIJ &
166                               + ( 1.0_RP-cmask(ij,k,l,2) ) * BIG
167     Qin(ip1jp1,k,l,I_min,5) = (      cmask(ij,k,l,2) ) * BIG      &
168                               + ( 1.0_RP-cmask(ij,k,l,2) ) * q_min_AIJ
169     Qin(ij,    k,l,I_max,2) = (      cmask(ij,k,l,2) ) * q_max_AIJ &
170                               + ( 1.0_RP-cmask(ij,k,l,2) ) * (-BIG)
171     Qin(ip1jp1,k,l,I_max,5) = (      cmask(ij,k,l,2) ) * (-BIG) &
172                               + ( 1.0_RP-cmask(ij,k,l,2) ) * q_max_AIJ
173
174         !$omp end master
175     endif

```

The first section above and second section below are in long l - and k - double loop. In that there are 2 main i, j -double loops and one IF clause and a small i, j -loop. The i, j -loop(l.102) calculates the inflow bounds. First, minimum and maximum of q at arc points (specified by AI, AIJ and AJ) are set, then actual bound of inflow Q_{in} is calculated. IF clause at l.151 is a treatment for the singular points.

Second section is as follows.

```

175     !---< (iii) define allowable range of q at next step, eq.(42)&(43) >---
176     !OCL XFILL
177     !$omp do
178     do j = gmin, gmax
179     do i = gmin, gmax
180         ij = (j-1)*iall + i
181
182         qnext_min = min( q(ij,k,l),      &
183                       Qin(ij,k,l,I_min,1), &
184                       Qin(ij,k,l,I_min,2), &
185                       Qin(ij,k,l,I_min,3), &
186                       Qin(ij,k,l,I_min,4), &
187                       Qin(ij,k,l,I_min,5), &
188                       Qin(ij,k,l,I_min,6) )

```

```

189
190     qnext_max = max( q(ij,k,l),           &
191                    Qin(ij,k,l,I_max,1), &
192                    Qin(ij,k,l,I_max,2), &
193                    Qin(ij,k,l,I_max,3), &
194                    Qin(ij,k,l,I_max,4), &
195                    Qin(ij,k,l,I_max,5), &
196                    Qin(ij,k,l,I_max,6) )
197
198     ch_masked1 = min( ch(ij,k,l,1), 0.0_RP )
199     ch_masked2 = min( ch(ij,k,l,2), 0.0_RP )
200     ch_masked3 = min( ch(ij,k,l,3), 0.0_RP )
201     ch_masked4 = min( ch(ij,k,l,4), 0.0_RP )
202     ch_masked5 = min( ch(ij,k,l,5), 0.0_RP )
203     ch_masked6 = min( ch(ij,k,l,6), 0.0_RP )
204
205     Cin_sum    = ch_masked1 &
206                + ch_masked2 &
207                + ch_masked3 &
208                + ch_masked4 &
209                + ch_masked5 &
210                + ch_masked6
211
212     Cout_sum   = ch(ij,k,l,1) - ch_masked1 &
213                + ch(ij,k,l,2) - ch_masked2 &
214                + ch(ij,k,l,3) - ch_masked3 &
215                + ch(ij,k,l,4) - ch_masked4 &
216                + ch(ij,k,l,5) - ch_masked5 &
217                + ch(ij,k,l,6) - ch_masked6
218
219     CQin_min_sum = ch_masked1 * Qin(ij,k,l,I_min,1) &
220                  + ch_masked2 * Qin(ij,k,l,I_min,2) &
221                  + ch_masked3 * Qin(ij,k,l,I_min,3) &
222                  + ch_masked4 * Qin(ij,k,l,I_min,4) &
223                  + ch_masked5 * Qin(ij,k,l,I_min,5) &
224                  + ch_masked6 * Qin(ij,k,l,I_min,6)
225
226     CQin_max_sum = ch_masked1 * Qin(ij,k,l,I_max,1) &
227                  + ch_masked2 * Qin(ij,k,l,I_max,2) &
228                  + ch_masked3 * Qin(ij,k,l,I_max,3) &
229                  + ch_masked4 * Qin(ij,k,l,I_max,4) &
230                  + ch_masked5 * Qin(ij,k,l,I_max,5) &
231                  + ch_masked6 * Qin(ij,k,l,I_max,6)
232
233     zerosw = 0.5_RP - sign(0.5_RP,abs(Cout_sum)-EPS) ! if Cout_sum = 0, sw = 1
234
235     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)) ) &
236                        / ( Cout_sum + zerosw ) * ( 1.0_RP - zerosw )           &
237                        + q(ij,k,l) * zerosw
238     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)) ) &
239                        / ( Cout_sum + zerosw ) * ( 1.0_RP - zerosw )           &
240                        + q(ij,k,l) * zerosw
241
242     enddo
243     enddo
244     !$omp end do
245
246     !OCL XFILL
247     !$omp do
248     do j = 1, iall
249     do i = 1, iall
250     if ( i < gmin .OR. i > gmax &
251         .OR. j < gmin .OR. j > gmax ) then
252         ij = (j-1)*iall + i
253
254         Qout(ij,k,l,I_min) = q(ij,k,l)
255         Qout(ij,k,l,I_min) = q(ij,k,l)
256         Qout(ij,k,l,I_max) = q(ij,k,l)
257         Qout(ij,k,l,I_max) = q(ij,k,l)
258     endif
259     enddo
260     !$omp end do
261
262     enddo ! k loop
263     !$omp end parallel
264     enddo ! l loop
265

```

There are two i, j -double loops. In the first one, allowable range of q is defined as q_{next_min} , q_{next_max} , then Q_{out} is calculated. And the second double loop is to set halo region.

The third section is as follows. This section is for the pole region, doing almost the same calculation with the normal region.

```

266 if ( ADM_have_pl ) then
267   n = ADM_gslf_pl
268
269   do l = 1, ADM_lall_pl
270     do k = 1, ADM_kall
271       do v = ADM_gmin_pl, ADM_gmax_pl
272         ij = v
273         ijp1 = v + 1
274         ijm1 = v - 1
275         if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
276         if( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl
277
278         q_min_pl = min( q_pl(n,k,l), q_pl(ij,k,l), q_pl(ijm1,k,l), q_pl(ijp1,k,l) )
279         q_max_pl = max( q_pl(n,k,l), q_pl(ij,k,l), q_pl(ijm1,k,l), q_pl(ijp1,k,l) )
280
281         Qin_pl(ij,k,l,I_min,1) = ( cmask_pl(ij,k,l) ) * q_min_pl &
282                               + ( 1.0_RP-cmask_pl(ij,k,l) ) * BIG
283         Qin_pl(ij,k,l,I_min,2) = ( cmask_pl(ij,k,l) ) * BIG &
284                               + ( 1.0_RP-cmask_pl(ij,k,l) ) * q_min_pl
285         Qin_pl(ij,k,l,I_max,1) = ( cmask_pl(ij,k,l) ) * q_max_pl &
286                               + ( 1.0_RP-cmask_pl(ij,k,l) ) * (-BIG)
287         Qin_pl(ij,k,l,I_max,2) = ( cmask_pl(ij,k,l) ) * (-BIG) &
288                               + ( 1.0_RP-cmask_pl(ij,k,l) ) * q_max_pl
289
290       enddo
291
292       qnext_min_pl = q_pl(n,k,l)
293       qnext_max_pl = q_pl(n,k,l)
294       do v = ADM_gmin_pl, ADM_gmax_pl
295         qnext_min_pl = min( qnext_min_pl, Qin_pl(v,k,l,I_min,1) )
296         qnext_max_pl = max( qnext_max_pl, Qin_pl(v,k,l,I_max,1) )
297       enddo
298
299       Cin_sum_pl = 0.0_RP
300       Cout_sum_pl = 0.0_RP
301       CQin_max_sum_pl = 0.0_RP
302       CQin_min_sum_pl = 0.0_RP
303       do v = ADM_gmin_pl, ADM_gmax_pl
304         ch_masked = cmask_pl(v,k,l) * ch_pl(v,k,l)
305
306         Cin_sum_pl = Cin_sum_pl + ch_masked
307         Cout_sum_pl = Cout_sum_pl - ch_masked + ch_pl(v,k,l)
308         CQin_min_sum_pl = CQin_min_sum_pl + ch_masked * Qin_pl(v,k,l,I_min,1)
309         CQin_max_sum_pl = CQin_max_sum_pl + ch_masked * Qin_pl(v,k,l,I_max,1)
310       enddo
311
312       zerosw = 0.5_RP - sign(0.5_RP,abs(Cout_sum_pl)-EPS) ! if Cout_sum_pl = 0, sw = 1
313
314       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)) ) &
315                             / ( Cout_sum_pl + zerosw ) * ( 1.0_RP - zerosw ) &
316                             + q_pl(n,k,l) * zerosw
317       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)) ) &
318                             / ( Cout_sum_pl + zerosw ) * ( 1.0_RP - zerosw ) &
319                             + q_pl(n,k,l) * zerosw
320
321     enddo
322   enddo
323

```

In the original subroutine, halo exchange using communication subroutine COMM_data_transfer here. In kernelization process, this communication is omitted. Values to be sent is output for the purpose of validation, and values to be received are given as an argument, read from input data file in the main program of this kernel program.

```

324 !#####KERNEL
325 call DEBUG_rapend ('___horizontal_adv_limiter')
326 Qout_pl(ADM_gmin_pl:ADM_gmax_pl, :, :) = 0.0_RP
327
328 Qout_prev (:, :, :) = Qout (:, :, :)
329 Qout_prev_pl (:, :, :) = Qout_pl (:, :, :)
330 !call COMM_data_transfer( Qout(:, :, :), Qout_pl(:, :, :) )
331 Qout (:, :, :) = Qout_post (:, :, :)
332 Qout_pl (:, :, :) = Qout_post_pl(:, :, :)
333 call DEBUG_rapstart ('___horizontal_adv_limiter')

```



```

334 !#####KERNEL
335

```

The next section is as follows.

```

336 !---- apply inflow/outflow limiter
337 do l = 1, ADM_lall
338 !$omp parallel do default(none),private(i,j,k,ij,ip1j,ip1jp1,ijp1), &
339 !$omp shared(l,gmin,gmax,kall,iall,q_a,cmask,Qin,Qout)
340 do k = 1, kall
341 do j = gmin-1, gmax
342 do i = gmin-1, gmax
343 ij = (j-1)*iall + i
344 ip1j = ij + 1
345 ip1jp1 = ij + iall + 1
346 ijp1 = ij + iall
347
348 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) ) &
349 + ( 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) ) )
350 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 ) ) &
351 + ( 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 ) ) )
352 q_a(ip1j,k,l,4) = q_a(ij,k,l,1)
353
354 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) ) &
355 + ( 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) ) )
356 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 ) ) &
357 + ( 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 ) ) )
358 q_a(ip1jp1,k,l,5) = q_a(ij,k,l,2)
359
360 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) ) &
361 + ( 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) ) )
362 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 ) ) &
363 + ( 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 ) ) )
364 q_a(ijp1,k,l,6) = q_a(ij,k,l,3)
365 enddo
366 enddo
367 enddo
368 !$omp end parallel do
369 enddo
370

```

In new l - and k - double loop, there is a i, j -double loop, calculating limiter applied tracer quantities at each cell faces, i.e. edges of the hexagonal control volume. Note that only 1st, 2nd, 3rd edge must be calculated, because other 3 edges' value is the same with other grid's 1st, 2nd, or 3rd edge.

The last part of this subroutine is as follows.

```

371 if ( ADM_have_pl ) then
372 n = ADM_gslf_pl
373
374 do l = 1, ADM_lall_pl
375 do k = 1, ADM_kall
376 do v = ADM_gmin_pl, ADM_gmax_pl
377 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) ) &
378 + ( 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) ) )
379 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 ) ) &
380 + ( 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 ) ) )
381 enddo
382 enddo
383 enddo
384 endif
385
386 call DEBUG_rapend ( '___horizontal_adv_limiter' )
387
388 return
389 end subroutine horizontal_limiter_thuburn

```

In this part, q_a in the pole region are calculated, with almost the same procedure with the normal region.

2.7.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of \$IAB_SYS=Ubuntu-gnu-mpi case.

```

### Input ###
+check[q_a_prev ] max= 3.1443592333952495E+02,min= -3.3659800625927818E+02,sum= 1.4867286349717416E+07
+check[q_a_prev_pl ] max= 7.5768030875957253E+00,min= -5.6211573453294959E-15,sum= 1.5762773946695211E+03
+check[check_q_a ] max= 3.0706553064103554E+02,min= -2.3487181740471829E+02,sum= 1.4857441189951191E+07
+check[check_q_a_pl ] max= 6.5571419615446418E+00,min= -5.6211573453294959E-15,sum= 1.5683664645246540E+03
+check[q ] max= 7.3394566876316425E+00,min= 0.0000000000000000E+00,sum= 2.4959622235983950E+06
+check[q_pl ] max= 6.5571419615446418E+00,min= 0.0000000000000000E+00,sum= 1.8536812127383619E+03
+check[d ] max= 1.7307542155642468E-05,min= -1.9433424502805651E-05,sum= 6.2175140984963743E-06
+check[d_pl ] max= 6.2231074370491122E-06,min= -2.0788320527281462E-05,sum= -1.1073365231406677E-03
+check[ch ] max= 2.5115202433354633E-01,min= -2.8726813406847629E-01,sum= -6.8752454869001423E-01
+check[ch_pl ] max= 1.9806049765030476E-01,min= -1.2208152245920290E+01,sum= -3.1022849107616275E+02
+check[cmask ] max= 1.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 2.1729060000000000E+06
+check[cmask_pl ] max= 1.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 2.1800000000000000E+02
+check[check_Qout_prev ] max= 2.5201719369799781E+05,min= -7.0121044696269173E+05,sum= 4.7462261170534119E+06
+check[check_Qout_prev_pl ] max= 3.4849498839973911E+01,min= -3.6368061161685844E+01,sum= 7.2956220495347202E+02
+check[Qout_post ] max= 2.5201719369799781E+05,min= -7.0121044696269173E+05,sum= 4.7387405453230906E+06
+check[Qout_post_pl ] max= 5.9738451374182347E+01,min= -3.6368061161685844E+01,sum= 4.0063169963931473E+03
### Output ###
+check[q_a ] max= 3.0706553064103554E+02,min= -2.3487181740471829E+02,sum= 1.4857441189951191E+07
+check[q_a_pl ] max= 6.5571419615446418E+00,min= -5.6211573453294959E-15,sum= 1.5683664645246540E+03
+check[Qout_prev ] max= 2.5201719369799781E+05,min= -7.0121044696269173E+05,sum= 4.7462261170534119E+06
+check[Qout_prev_pl ] max= 3.4849498839973911E+01,min= -3.6368061161685844E+01,sum= 7.2956220495347202E+02
### Validation : point-by-point diff ###
+check[check_q_a ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_q_a_pl ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_Qout_prev ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
+check[check_Qout_prev_pl ] max= 0.0000000000000000E+00,min= 0.0000000000000000E+00,sum= 0.0000000000000000E+00
*** Finish kernel

```

Check the lines below “Validation : point-by-point diff” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.7.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in [subsection 2.1.7](#). Note that in this program kernel part is iterated one time.

```

*** Computational Time Report
*** ID=001 : MAIN_dyn_horiz_adv_limiter      T= 0.180 N= 1
*** ID=002 : ____horizontal_adv_limiter     T= 0.164 N= 2

```

2.8 dyn_metrics

2.8.1 Description

Kernel `dyn_metrics` gathers calculation part of various metric terms in several subroutines listed below:

- `GMTR_p_setup`
- `GMTR_t_setup`
- `GMTR_a_setup`
- `OPRT_divergence_setup`
- `OPRT_rotation_setup`
- `OPRT_gradient_setup`
- `OPRT_laplacian_setup`
- `OPRT_diffusion_setup`

These subroutines are defined as the same name in module `mod_gmtr` and `mod_oprt` in original *NICAM*.

2.8.2 Discretization and code

(1) GMTR_p_setup

mod_gmtr contains public objects related to the metrics. This subroutine is to setup metrics for the cell point. Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine GMTR_p_setup( &
2     GRD_x,  GRD_x_pl,  &
3     GRD_xt, GRD_xt_pl, &
4     GRD_s,  GRD_s_pl,  &
5     GMTR_p, GMTR_p_pl, &
6     GRD_rscale
7     )
8     !ESC! use mod_adm, only: &
9     !ESC!     ADM_nxyz,      &
10    !ESC!     ADM_have_pl,   &
11    !ESC!     ADM_have_sgp,  &
12    !ESC!     ADM_vlink,    &
13    !ESC!     ADM_gmin,     &
14    !ESC!     ADM_gmax,     &
15    !ESC!     ADM_gslf_pl
16    !ESC! use mod_grd, only: &
17    !ESC!     GRD_XDIR,      &
18    !ESC!     GRD_YDIR,      &
19    !ESC!     GRD_ZDIR,      &
20    !ESC!     GRD_LON,       &
21    !ESC!     GRD_grid_type_on_plane, &
22    !ESC!     GRD_grid_type
23    use mod_vector, only: &
24    VECTR_triangle,      &
25    VECTR_triangle_plane
26    implicit none
27
28    real(RP), intent(in) :: GRD_x (ADM_gall ,k0,ADM_lall , ADM_nxyz)
29    real(RP), intent(in) :: GRD_x_pl (ADM_gall_pl,k0,ADM_lall_pl, ADM_nxyz)
30    real(RP), intent(in) :: GRD_xt (ADM_gall ,k0,ADM_lall ,TI:TJ,ADM_nxyz)
31    real(RP), intent(in) :: GRD_xt_pl (ADM_gall_pl,k0,ADM_lall_pl, ADM_nxyz)
32    real(RP), intent(in) :: GRD_s (ADM_gall ,k0,ADM_lall , 2)
33    real(RP), intent(in) :: GRD_s_pl (ADM_gall_pl,k0,ADM_lall_pl, 2)
34    real(RP), intent(out) :: GMTR_p (ADM_gall ,k0,ADM_lall ,GMTR_p_nmax)
35    real(RP), intent(out) :: GMTR_p_pl (ADM_gall_pl,k0,ADM_lall_pl,GMTR_p_nmax)
36    real(RP), intent(in) :: GRD_rscale
37
38    real(RP) :: wk (ADM_nxyz,0:7,ADM_gall)
39    real(RP) :: wk_pl (ADM_nxyz,0:ADM_vlink+1)
40
41    real(RP) :: area
42    real(RP) :: cos_lambda, sin_lambda
43
44    integer :: ij
45    integer :: ip1j, ijp1, ip1jp1
46    integer :: im1j, ijm1, im1jm1
47
48    integer :: i, j, l, d, v, n
49    !-----

```

GRD_x and GRD_xt are the coordinates in 3-D Cartesian of the center points and the vertex points of control cell, respectively. And GRD_s is the coordinates in the spherical coordinate on the planet i.e. latitude and longitude in radian. Those with suffix _pl are for the pole region. These coordinate values are given as arguments, read from input data file in this kernel program. GMTR_p and GMTR_p_pl are the metrics this subroutine calculates for normal region and pole region, respectively. The last dimension of these have the size of GMTR_p_nmax, each element of this dimension specify the kind of various metrics. For example, GMTR_p(:, :, :, GMTR_p-AREA) means area of the hexagonal controll cell GRD_rscale is a scaling factor for the radius of the sphere, and set as 6.37122E+6_RP [m] for this kernel program in problem_size.inc.

There is one long *l*-loop in this subroutine, divided to 3 sections. The first section is as follows.

```

50    !if( IO_L ) write(IO_FID_LOG,*) '*** setup metrics for hexagonal/pentagonal mesh'
51
52    GMTR_p ( : , : , : , : ) = 0.0_RP
53    GMTR_p_pl( : , : , : , : ) = 0.0_RP
54
55    do l = 1, ADM_lall

```

```

56 do j = ADM_gmin, ADM_gmax
57 do i = ADM_gmin, ADM_gmax
58   ij = suf(i ,j )
59   ip1j = suf(i+1,j )
60   ip1jp1 = suf(i+1,j+1)
61   ijp1 = suf(i ,j+1)
62   im1j = suf(i-1,j )
63   im1jm1 = suf(i-1,j-1)
64   ijm1 = suf(i ,j-1)
65
66   !--- prepare 1 center and 6 vertices
67   do d = 1, ADM_nxyz
68     wk(d,0,ij) = GRD_x(ij,k0,l,d)
69
70     wk(d,1,ij) = GRD_xt(ijm1 ,k0,l,TJ,d)
71     wk(d,2,ij) = GRD_xt(ij ,k0,l,TI,d)
72     wk(d,3,ij) = GRD_xt(ij ,k0,l,TJ,d)
73     wk(d,4,ij) = GRD_xt(im1j ,k0,l,TI,d)
74     wk(d,5,ij) = GRD_xt(im1jm1,k0,l,TJ,d)
75     wk(d,6,ij) = GRD_xt(im1jm1,k0,l,TI,d)
76     wk(d,7,ij) = wk(d,1,ij)
77   enddo
78 enddo ! i loop
79 enddo ! j loop
80
81 if ( ADM_have_sgp(1) ) then ! pentagon
82   wk(:,6,suf(ADM_gmin,ADM_gmin)) = wk(:,1,suf(ADM_gmin,ADM_gmin))
83   wk(:,7,suf(ADM_gmin,ADM_gmin)) = wk(:,1,suf(ADM_gmin,ADM_gmin))
84 endif
85

```

In this first section, the coordinates of center and 6 vertices of a control volume are set to a temporary array `wk`. `ADM_nxyz` is 3 and the inner-most loop index `d` specifies (X, Y, Z) direction of 3-D Cartesian coordinates. As for the last IF clause(1.81), if `ADM_have_sgp(1)` is true, the l -th region has the singular point and the control volume of that point is not a hexagon but a pentagon.

The second section of the main l -loop is as follows.

```

86 !--- calc control area
87 if ( GRD_grid_type == GRD_grid_type_on_plane ) then
88   do j = ADM_gmin, ADM_gmax
89   do i = ADM_gmin, ADM_gmax
90     ij = suf(i,j)
91
92     area = 0.0_RP
93     do v = 1, 6
94       area = area + VECTR_triangle_plane( wk(:,0,ij), wk(:,v,ij), wk(:,v+1,ij) )
95     enddo
96
97     GMTR_p(ij,k0,l,GMTR_p_AREA) = area
98     GMTR_p(ij,k0,l,GMTR_p_RAREA) = 1.0_RP / GMTR_p(ij,k0,l,GMTR_p_AREA)
99
100   enddo ! i loop
101   enddo ! j loop
102 else
103   do j = ADM_gmin, ADM_gmax
104   do i = ADM_gmin, ADM_gmax
105     ij = suf(i,j)
106
107     wk(:, :,ij) = wk(:, :,ij) / GRD_rscale
108
109     area = 0.0_RP
110     do v = 1, 6
111       area = area + VECTR_triangle( wk(:,0,ij), wk(:,v,ij), wk(:,v+1,ij), GMTR_polygon_type, GRD_rscale )
112     enddo
113
114     GMTR_p(ij,k0,l,GMTR_p_AREA) = area
115     GMTR_p(ij,k0,l,GMTR_p_RAREA) = 1.0_RP / GMTR_p(ij,k0,l,GMTR_p_AREA)
116
117   enddo ! i loop
118   enddo ! j loop
119 endif
120

```

This section calculates the area size of control volume, by summing up 6 triangles that consist of the hexagon. `GMTR_p(:, :, :, GMTR_p_RAREA)` is a reciprocal of the size of area.

The last part of this main *l*-loop is as follows.

```

121      !--- calc coefficient between xyz <-> latlon
122      if ( GRD_grid_type == GRD_grid_type_on_plane ) then
123          GMTR_p(:,k0,l,GMTR_p_IX) = 1.0_RP
124          GMTR_p(:,k0,l,GMTR_p_IY) = 0.0_RP
125          GMTR_p(:,k0,l,GMTR_p_IZ) = 0.0_RP
126          GMTR_p(:,k0,l,GMTR_p_JX) = 0.0_RP
127          GMTR_p(:,k0,l,GMTR_p_JY) = 1.0_RP
128          GMTR_p(:,k0,l,GMTR_p_JZ) = 0.0_RP
129      else
130          do j = ADM_gmin, ADM_gmax
131              do i = ADM_gmin, ADM_gmax
132                  ij = suf(i,j)
133
134                  sin_lambda = sin( GRD_s(ij,k0,l,GRD_LON) )
135                  cos_lambda = cos( GRD_s(ij,k0,l,GRD_LON) )
136
137                  GMTR_p(ij,k0,l,GMTR_p_IX) = -sin_lambda
138                  GMTR_p(ij,k0,l,GMTR_p_IY) = cos_lambda
139                  GMTR_p(ij,k0,l,GMTR_p_IZ) = 0.0_RP
140                  GMTR_p(ij,k0,l,GMTR_p_JX) = -( GRD_x(ij,k0,l,ZDIR) * cos_lambda ) / GRD_rscales
141                  GMTR_p(ij,k0,l,GMTR_p_JY) = -( GRD_x(ij,k0,l,ZDIR) * sin_lambda ) / GRD_rscales
142                  GMTR_p(ij,k0,l,GMTR_p_JZ) = ( GRD_x(ij,k0,l,XDIR) * cos_lambda &
143                      + GRD_x(ij,k0,l,YDIR) * sin_lambda ) / GRD_rscales
144              enddo ! i loop
145          enddo ! j loop
146      endif
147      enddo ! l loop
148

```

This section calculates the coefficients used by conversion from 3-D Cartesian coordinates to the spherical coordinates and *vice versa*. Note that `GRD_grid_type == GRD_grid_type_on_plane` is set as false for this kernel program in `problem_size.inc`.

The remaining part of this subroutine is as follows.

```

149      if ( ADM_have_pl ) then
150          n = ADM_gslf_pl
151
152          do l = 1, ADM_lall_pl
153              !--- prepare l center and * vertices
154              do d = 1, ADM_nxyz
155                  wk_pl(d,0) = GRD_x_pl(n,k0,l,d)
156                  do v = 1, ADM_vlink ! (ICO=5)
157                      wk_pl(d,v) = GRD_xt_pl(v+1,k0,l,d)
158                  enddo
159                  wk_pl(d,ADM_vlink+1) = wk_pl(d,1)
160              enddo
161
162              wk_pl(:,0) = wk_pl(:,0) / GRD_rscales
163
164              !--- calc control area
165              area = 0.0_RP
166              do v = 1, ADM_vlink ! (ICO=5)
167                  area = area + VECTR_triangle( wk_pl(:,0), wk_pl(:,v), wk_pl(:,v+1), GMTR_polygon_type, GRD_rscales )
168              enddo
169
170              GMTR_p_pl(n,k0,l,GMTR_p_AREA) = area
171              GMTR_p_pl(n,k0,l,GMTR_p_RAREA) = 1.0_RP / GMTR_p_pl(n,k0,l,GMTR_p_AREA)
172
173              !--- calc coefficient between xyz <-> latlon
174              sin_lambda = sin( GRD_s_pl(n,k0,l,GRD_LON) )
175              cos_lambda = cos( GRD_s_pl(n,k0,l,GRD_LON) )
176
177              GMTR_p_pl(n,k0,l,GMTR_p_IX) = -sin_lambda
178              GMTR_p_pl(n,k0,l,GMTR_p_IY) = cos_lambda
179              GMTR_p_pl(n,k0,l,GMTR_p_IZ) = 0.0_RP
180              GMTR_p_pl(n,k0,l,GMTR_p_JX) = -( GRD_x_pl(n,k0,l,ZDIR) * cos_lambda ) / GRD_rscales
181              GMTR_p_pl(n,k0,l,GMTR_p_JY) = -( GRD_x_pl(n,k0,l,ZDIR) * sin_lambda ) / GRD_rscales
182              GMTR_p_pl(n,k0,l,GMTR_p_JZ) = ( GRD_x_pl(n,k0,l,XDIR) * cos_lambda &
183                  + GRD_x_pl(n,k0,l,YDIR) * sin_lambda ) / GRD_rscales
184          enddo ! l loop
185      endif
186
187      return
188      end subroutine GMTR_p_setup

```

This part is to calculate for the pole region. Note that the pole region is a pentagon.

(2) GMTR_t_setup

This subroutine is to setup metrics for the cell vertices or the triangles.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine GMTR_t_setup( &
2     GRD_x,  GRD_x_pl,  &
3     GRD_xt, GRD_xt_pl, &
4     GMTR_t, GMTR_t_pl, &
5     GRD_rscale
6     )
7     !ESC! use mod_adm, only: &
8     !ESC!   ADM_nxyz,      &
9     !ESC!   ADM_have_pl,  &
10    !ESC!   ADM_have_sgp, &
11    !ESC!   ADM_gmin,     &
12    !ESC!   ADM_gmax,     &
13    !ESC!   ADM_gslf_pl,  &
14    !ESC!   ADM_gmin_pl,  &
15    !ESC!   ADM_gmax_pl
16    !ESC! use mod_grd, only: &
17    !ESC!   GRD_grid_type_on_plane, &
18    !ESC!   GRD_grid_type
19    use mod_vector, only: &
20    VECTR_triangle,      &
21    VECTR_triangle_plane
22    implicit none
23
24    real(RP), intent(in) :: GRD_x   (ADM_gall ,k0,ADM_lall ,      ADM_nxyz)
25    real(RP), intent(in) :: GRD_x_pl (ADM_gall_pl,k0,ADM_lall_pl,  ADM_nxyz)
26    real(RP), intent(in) :: GRD_xt   (ADM_gall ,k0,ADM_lall ,TI:TJ,ADM_nxyz)
27    real(RP), intent(in) :: GRD_xt_pl (ADM_gall_pl,k0,ADM_lall_pl,  ADM_nxyz)
28    real(RP), intent(out) :: GMTR_t   (ADM_gall ,k0,ADM_lall ,TI:TJ,GMTR_t_nmax)
29    real(RP), intent(out) :: GMTR_t_pl (ADM_gall_pl,k0,ADM_lall_pl,  GMTR_t_nmax)
30    real(RP), intent(in) :: GRD_rscale
31
32    real(RP) :: wk   (ADM_nxyz,0:3,ADM_gall,TI:TJ)
33    real(RP) :: wk_pl (ADM_nxyz,0:3)
34
35    real(RP) :: area, area1, area2, area3
36
37    integer :: ij
38    integer :: ip1j, ijp1, ip1jp1
39
40    integer :: i, j, l, d, v, n, t
41    !-----

```

Input arguments are the same with the previous GMTR_p_setup. GMTR_t and GMTR_t_pl are the metrics for the vertex points of control cell for the normal region and the pole region, respectively. In the last dimension of these arrays whose size is GMTR_t_nmax, each element specifies the kind of various metrics. For example, GMTR_t (:, :, :, :, GMTR_t_AREA) means the area of the upward and downward triangle, which contains vertex points of control cell as the gravitational center.

Main part of this subroutine is a single *l*-loop, divided to two sections. The first section is as follows.

```

42    !if( IO_L ) write(IO_FID_LOG,*) '*** setup metrics for triangle mesh'
43
44    GMTR_t   (:, :, :, :, :) = 0.0_RP
45    GMTR_t_pl (:, :, :, :) = 0.0_RP
46
47    do l = 1, ADM_lall
48        do j = ADM_gmin-1, ADM_gmax
49            do i = ADM_gmin-1, ADM_gmax
50                ij = suf(i , j )
51                ip1j = suf(i+1,j )
52                ip1jp1 = suf(i+1,j+1)
53                ijp1 = suf(i ,j+1)
54
55                !--- prepare 1 center and 3 vertices for 2 triangles
56                do d = 1, ADM_nxyz
57                    wk(d,0,ij,TI) = GRD_xt(ij,k0,l,TI,d)
58
59                    wk(d,1,ij,TI) = GRD_x(ij ,k0,l,d)

```

```

60      wk(d,2,ij,TI) = GRD_x(ip1j ,k0,1,d)
61      wk(d,3,ij,TI) = GRD_x(ip1jp1,k0,1,d)
62
63      wk(d,0,ij,TJ) = GRD_xt(ij,k0,1,TJ,d)
64
65      wk(d,1,ij,TJ) = GRD_x(ij ,k0,1,d)
66      wk(d,2,ij,TJ) = GRD_x(ip1jp1,k0,1,d)
67      wk(d,3,ij,TJ) = GRD_x(ipj1 ,k0,1,d)
68      enddo
69      enddo
70      enddo
71

```

In this section, coordinates of the center and three vertices for two triangles represents. The meaning of ADM_nxyz and the loop index d are the same with them in the previous subroutine GMTR_p_setup.

The second section is as follows.

```

72      !--- treat unused triangle
73      wk(:, :, suf(ADM_gmax,ADM_gmin-1),TI) = wk(:, :, suf(ADM_gmax,ADM_gmin-1),TJ)
74      wk(:, :, suf(ADM_gmin-1,ADM_gmax),TJ) = wk(:, :, suf(ADM_gmin-1,ADM_gmax),TI)
75
76      if ( ADM_have_sgp(1) ) then ! pentagon
77      wk(:, :, suf(ADM_gmin-1,ADM_gmin-1),TI) = wk(:, :, suf(ADM_gmin,ADM_gmin-1),TJ)
78      endif
79
80      if ( GRD_grid_type == GRD_grid_type_on_plane ) then
81      do t = TI,TJ
82      do j = ADM_gmin-1, ADM_gmax
83      do i = ADM_gmin-1, ADM_gmax
84      ij = suf(i,j)
85
86      area1 = VECTR_triangle_plane( wk(:,0,ij,t), wk(:,2,ij,t), wk(:,3,ij,t) )
87      area2 = VECTR_triangle_plane( wk(:,0,ij,t), wk(:,3,ij,t), wk(:,1,ij,t) )
88      area3 = VECTR_triangle_plane( wk(:,0,ij,t), wk(:,1,ij,t), wk(:,2,ij,t) )
89
90      area = area1 + area2 + area3
91
92      GMTR_t(ij,k0,1,t,GMTR_t_AREA) = area
93      GMTR_t(ij,k0,1,t,GMTR_t_RAREA) = 1.0_RP / area
94
95      GMTR_t(ij,k0,1,t,GMTR_t_W1) = area1 / area
96      GMTR_t(ij,k0,1,t,GMTR_t_W2) = area2 / area
97      GMTR_t(ij,k0,1,t,GMTR_t_W3) = area3 / area
98      enddo
99      enddo
100     enddo
101     else
102     do t = TI,TJ
103     do j = ADM_gmin-1, ADM_gmax
104     do i = ADM_gmin-1, ADM_gmax
105     ij = suf(i,j)
106
107     wk(:, :, ij,t) = wk(:, :, ij,t) / GRD_rscale
108
109     area1 = VECTR_triangle( wk(:,0,ij,t), wk(:,2,ij,t), wk(:,3,ij,t), GMTR_polygon_type, GRD_rscale )
110     area2 = VECTR_triangle( wk(:,0,ij,t), wk(:,3,ij,t), wk(:,1,ij,t), GMTR_polygon_type, GRD_rscale )
111     area3 = VECTR_triangle( wk(:,0,ij,t), wk(:,1,ij,t), wk(:,2,ij,t), GMTR_polygon_type, GRD_rscale )
112
113     area = area1 + area2 + area3
114
115     GMTR_t(ij,k0,1,t,GMTR_t_AREA) = area
116     GMTR_t(ij,k0,1,t,GMTR_t_RAREA) = 1.0_RP / area
117
118     GMTR_t(ij,k0,1,t,GMTR_t_W1) = area1 / area
119     GMTR_t(ij,k0,1,t,GMTR_t_W2) = area2 / area
120     GMTR_t(ij,k0,1,t,GMTR_t_W3) = area3 / area
121     enddo
122     enddo
123     enddo
124     endif
125
126     enddo
127

```

This section calculates an area of the triangle each triangle point represents. Note that GRD_grid_type == GRD_grid_type_on_plane is false in this kernel program. The triangle is divided to three small triangles

by connecting three vertices and the triangle point. The area of whole triangle is calculated by summing up these three small triangles. $GMTR_t(:, :, :, :, GMTR_t_RAREA)$ means the reciprocal of the area of whole triangle, and $GMTR_t(:, :, :, :, GMTR_t_W1)$ etc. are the area fraction of each of three small triangles. These are used for the triangle linear interpolation from the grid point (the center of control cell) to the vertex point.

The remaining part of this subroutine is for the pole region and is as follows.

```

128  if ( ADM_have_pl ) then
129      n = ADM_gslf_pl
130
131      do l = 1,ADM_lall_pl
132          do v = ADM_gmin_pl, ADM_gmax_pl
133              ij = v
134              ijp1 = v + 1
135              if ( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
136
137              do d = 1, ADM_nxyz
138                  wk_pl(d,0) = GRD_xt_pl(ij,k0,l,d)
139
140                  wk_pl(d,1) = GRD_x_pl(n ,k0,l,d)
141                  wk_pl(d,2) = GRD_x_pl(ij ,k0,l,d)
142                  wk_pl(d,3) = GRD_x_pl(ijp1,k0,l,d)
143              enddo
144
145              wk_pl(:,:) = wk_pl(:,:) / GRD_rsacle
146
147              area1 = VECTR_triangle( wk_pl(:,0), wk_pl(:,2), wk_pl(:,3), GMTR_polygon_type, GRD_rsacle )
148              area2 = VECTR_triangle( wk_pl(:,0), wk_pl(:,3), wk_pl(:,1), GMTR_polygon_type, GRD_rsacle )
149              area3 = VECTR_triangle( wk_pl(:,0), wk_pl(:,1), wk_pl(:,2), GMTR_polygon_type, GRD_rsacle )
150
151              area = area1 + area2 + area3
152
153              GMTR_t_pl(ij,k0,l,GMTR_t_AREA) = area
154              GMTR_t_pl(ij,k0,l,GMTR_t_RAREA) = 1.0_RP / area
155
156              GMTR_t_pl(ij,k0,l,GMTR_t_W1) = area1 / area
157              GMTR_t_pl(ij,k0,l,GMTR_t_W2) = area2 / area
158              GMTR_t_pl(ij,k0,l,GMTR_t_W3) = area3 / area
159          enddo
160      enddo
161  endif
162
163  return
164 end subroutine GMTR_t_setup

```

The procedure of calculation is almost the same with for the normal region, but note that for the pole region, there is only one triangle point for one grid point, and there is no distinction between TI and TJ.

(3) GMTR_a_setup

This subroutine is to setup metrics for the cell edge points or the arcs of triangles.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine GMTR_a_setup( &
2      GRD_x,  GRD_x_pl,  &
3      GRD_xt, GRD_xt_pl, &
4      GMTR_a, GMTR_a_pl, &
5      GRD_rsacle
6      )
7      !ESC! use mod_adm, only: &
8          ADM_nxyz, &
9          ADM_have_pl, &
10         ADM_have_sgp, &
11         ADM_gmin, &
12         ADM_gmax, &
13         ADM_gslf_pl, &
14         ADM_gmin_pl, &
15         ADM_gmax_pl
16         !ESC! use mod_grd, only: &
17         GRD_grid_type_on_plane, &
18         GRD_grid_type
19         implicit none
20         real(RP), intent(in) :: GRD_x (ADM_gall ,k0,ADM_lall , ADM_nxyz)

```



```

21 real(RP), intent(in) :: GRD_x_pl (ADM_gall_pl,k0,ADM_lall_pl, ADM_nxyz)
22 real(RP), intent(in) :: GRD_xt (ADM_gall ,k0,ADM_lall ,TI:TJ,ADM_nxyz)
23 real(RP), intent(in) :: GRD_xt_pl(ADM_gall_pl,k0,ADM_lall_pl, ADM_nxyz)
24 real(RP), intent(out) :: GMTR_a (ADM_gall ,k0,ADM_lall ,AI:AJ,GMTR_a_nmax )
25 real(RP), intent(out) :: GMTR_a_pl(ADM_gall_pl,k0,ADM_lall_pl, GMTR_a_nmax_pl)
26 real(RP), intent(in) :: GRD_rscale
27
28 real(RP) :: wk (ADM_nxyz,2,ADM_gall)
29 real(RP) :: wk_pl(ADM_nxyz,2)
30
31 real(RP) :: Tvec(3), Nvec(3)
32
33 integer :: ij
34 integer :: ip1j, ijp1, ip1jp1
35 integer :: im1j, ijm1
36
37 integer :: i, j, l, d, v, n
38 !-----
39

```

Input arguments are the same with the previous GMTR_p_setup. GMTR_a and GMTR_a_pl are the metrics for the edge of the triangle or the hexagonal/pentagonal control cell for the normal region and the pole region, respectively. In the last dimension of these arrays whose size is GMTR_a_nmax, each element specifies the kind of various metrics.

Main part of this subroutine is consist of two *l*-loops. The first loop is divided by three sections, and is as follows.

```

40 !if( IO_L ) write(IO_FID_LOG,*) '*** setup metrics for cell arcs'
41
42 GMTR_a ( : , : , : , : ) = 0.0_RP
43 GMTR_a_pl( : , : , : , : ) = 0.0_RP
44
45 !--- Triangle
46 do l = 1, ADM_lall
47
48 !--- AI
49 do j = ADM_gmin-1, ADM_gmax+1
50 do i = ADM_gmin-1, ADM_gmax
51 ij = suf(i , j )
52 ip1j = suf(i+1,j )
53
54 do d = 1, ADM_nxyz
55 wk(d,1,ij) = GRD_x(ij ,k0,1,d)
56 wk(d,2,ij) = GRD_x(ip1j,k0,1,d)
57 enddo
58 enddo
59 enddo
60
61 ! treat arc of unused triangle
62 wk(:,1,suf(ADM_gmax ,ADM_gmin-1)) = GRD_x(suf(ADM_gmax ,ADM_gmin-1),k0,1,:)
63 wk(:,2,suf(ADM_gmax ,ADM_gmin-1)) = GRD_x(suf(ADM_gmax ,ADM_gmin ),k0,1,:)
64 wk(:,1,suf(ADM_gmin-1,ADM_gmax+1)) = GRD_x(suf(ADM_gmin ,ADM_gmax+1),k0,1,:)
65 wk(:,2,suf(ADM_gmin-1,ADM_gmax+1)) = GRD_x(suf(ADM_gmin ,ADM_gmax ),k0,1,:)
66
67 if ( ADM_have_sgp(1) ) then ! pentagon
68 wk(:,1,suf(ADM_gmin-1,ADM_gmin-1)) = GRD_x(suf(ADM_gmin ,ADM_gmin-1),k0,1,:)
69 wk(:,2,suf(ADM_gmin-1,ADM_gmin-1)) = GRD_x(suf(ADM_gmin+1,ADM_gmin ),k0,1,:)
70 endif
71
72 do j = ADM_gmin-1, ADM_gmax+1
73 do i = ADM_gmin-1, ADM_gmax
74 ij = suf(i,j)
75
76 call GMTR_TNvec( Tvec(:), Nvec(:), & ! [OUT]
77 wk(:,1,ij), wk(:,2,ij), & ! [IN]
78 GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
79
80 GMTR_a(ij,k0,1,AI,GMTR_a_TNX) = Nvec(1)
81 GMTR_a(ij,k0,1,AI,GMTR_a_TNY) = Nvec(2)
82 GMTR_a(ij,k0,1,AI,GMTR_a_TNZ) = Nvec(3)
83 GMTR_a(ij,k0,1,AI,GMTR_a_TTX) = Tvec(1)
84 GMTR_a(ij,k0,1,AI,GMTR_a_TTY) = Tvec(2)
85 GMTR_a(ij,k0,1,AI,GMTR_a_TTZ) = Tvec(3)
86 enddo
87 enddo
88
89 !--- AIJ

```

```

90 do j = ADM_gmin-1, ADM_gmax
91 do i = ADM_gmin-1, ADM_gmax
92   ij = suf(i ,j )
93   ip1jp1 = suf(i+1,j+1)
94
95   do d = 1, ADM_nxyz
96     wk(d,1,ij) = GRD_x(ij ,k0,1,d)
97     wk(d,2,ij) = GRD_x(ip1jp1,k0,1,d)
98   enddo
99 enddo
100 enddo
101
102 do j = ADM_gmin-1, ADM_gmax
103 do i = ADM_gmin-1, ADM_gmax
104   ij = suf(i,j)
105
106   call GMTR_TNvec( Tvec(:), Nvec(:), & ! [OUT]
107                   wk(:,1,ij), wk(:,2,ij), & ! [IN]
108                   GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
109
110   GMTR_a(ij,k0,1,AIJ,GMTR_a_TNX) = Nvec(1)
111   GMTR_a(ij,k0,1,AIJ,GMTR_a_TNY) = Nvec(2)
112   GMTR_a(ij,k0,1,AIJ,GMTR_a_TNZ) = Nvec(3)
113   GMTR_a(ij,k0,1,AIJ,GMTR_a_TTX) = Tvec(1)
114   GMTR_a(ij,k0,1,AIJ,GMTR_a_TTY) = Tvec(2)
115   GMTR_a(ij,k0,1,AIJ,GMTR_a_TTZ) = Tvec(3)
116 enddo
117 enddo
118
119 !--- AJ
120 do j = ADM_gmin-1, ADM_gmax
121 do i = ADM_gmin-1, ADM_gmax+1
122   ij = suf(i ,j )
123   ijp1 = suf(i ,j+1)
124
125   do d = 1, ADM_nxyz
126     wk(d,1,ij) = GRD_x(ij ,k0,1,d)
127     wk(d,2,ij) = GRD_x(ijp1,k0,1,d)
128   enddo
129 enddo
130 enddo
131
132 ! treat arc of unused triangle
133 wk(:,1,suf(ADM_gmax+1,ADM_gmin-1)) = GRD_x(suf(ADM_gmax+1,ADM_gmin),k0,1,:)
134 wk(:,2,suf(ADM_gmax+1,ADM_gmin-1)) = GRD_x(suf(ADM_gmax ,ADM_gmin),k0,1,:)
135 wk(:,1,suf(ADM_gmin-1,ADM_gmax )) = GRD_x(suf(ADM_gmin-1,ADM_gmax),k0,1,:)
136 wk(:,2,suf(ADM_gmin-1,ADM_gmax )) = GRD_x(suf(ADM_gmin ,ADM_gmax),k0,1,:)
137
138 do j = ADM_gmin-1, ADM_gmax
139 do i = ADM_gmin-1, ADM_gmax+1
140   ij = suf(i,j)
141
142   call GMTR_TNvec( Tvec(:), Nvec(:), & ! [OUT]
143                   wk(:,1,ij), wk(:,2,ij), & ! [IN]
144                   GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
145
146   GMTR_a(ij,k0,1,AJ,GMTR_a_TNX) = Nvec(1)
147   GMTR_a(ij,k0,1,AJ,GMTR_a_TNY) = Nvec(2)
148   GMTR_a(ij,k0,1,AJ,GMTR_a_TNZ) = Nvec(3)
149   GMTR_a(ij,k0,1,AJ,GMTR_a_TTX) = Tvec(1)
150   GMTR_a(ij,k0,1,AJ,GMTR_a_TTY) = Tvec(2)
151   GMTR_a(ij,k0,1,AJ,GMTR_a_TTZ) = Tvec(3)
152 enddo
153 enddo
154
155 enddo ! l loop

```

These three sections calculate the metrics for the edge points of three arcs of triangle represented by AI, AIJ and AJ, respectively. After setting the coordinates of two endpoints of the arc (wk), subroutine GMTR_TNvec calculates a normal vector and a tangent vector of the edge.

The second loop is also divided by three sections, and is as follows.

```

156 !--- Hexagon/Pentagon
157 do l = 1, ADM_lall
158
159   !--- AI
160   do j = ADM_gmin, ADM_gmax
161   do i = ADM_gmin-1, ADM_gmax

```

```

162     ij = suf(i ,j )
163     ijm1 = suf(i ,j-1)
164
165     do d = 1, ADM_nxyz
166         wk(d,1,ij) = GRD_xt(ij ,k0,1,TI,d)
167         wk(d,2,ij) = GRD_xt(ijm1,k0,1,TJ,d)
168     enddo
169 enddo
170
171
172 do j = ADM_gmin, ADM_gmax
173 do i = ADM_gmin-1, ADM_gmax
174     ij = suf(i,j)
175
176     call GMTR_TNvec( Tvec(:), Nvec(:), & ! [OUT]
177                    wk(:,1,ij), wk(:,2,ij), & ! [IN]
178                    GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
179
180     GMTR_a(ij,k0,1,AI,GMTR_a_HNX) = Nvec(1)
181     GMTR_a(ij,k0,1,AI,GMTR_a_HNY) = Nvec(2)
182     GMTR_a(ij,k0,1,AI,GMTR_a_HNZ) = Nvec(3)
183     GMTR_a(ij,k0,1,AI,GMTR_a_HTX) = Tvec(1)
184     GMTR_a(ij,k0,1,AI,GMTR_a_HTY) = Tvec(2)
185     GMTR_a(ij,k0,1,AI,GMTR_a_HTZ) = Tvec(3)
186 enddo
187 enddo
188
189 !--- AIJ
190 do j = ADM_gmin-1, ADM_gmax
191 do i = ADM_gmin-1, ADM_gmax
192     ij = suf(i ,j )
193
194     do d = 1, ADM_nxyz
195         wk(d,1,ij) = GRD_xt(ij ,k0,1,TJ,d)
196         wk(d,2,ij) = GRD_xt(ij ,k0,1,TI,d)
197     enddo
198 enddo
199 enddo
200
201 ! treat arc of unused hexagon
202 wk(:,1,suf(ADM_gmax ,ADM_gmin-1)) = GRD_xt(suf(ADM_gmax ,ADM_gmin-1),k0,1,TJ,:)
203 wk(:,2,suf(ADM_gmax ,ADM_gmin-1)) = GRD_xt(suf(ADM_gmax ,ADM_gmin ),k0,1,TI,:)
204 wk(:,1,suf(ADM_gmin-1,ADM_gmax )) = GRD_xt(suf(ADM_gmin ,ADM_gmax ),k0,1,TJ,:)
205 wk(:,2,suf(ADM_gmin-1,ADM_gmax )) = GRD_xt(suf(ADM_gmin-1,ADM_gmax ),k0,1,TI,:)
206
207 do j = ADM_gmin-1, ADM_gmax
208 do i = ADM_gmin-1, ADM_gmax
209     ij = suf(i,j)
210
211     call GMTR_TNvec( Tvec(:), Nvec(:), & ! [OUT]
212                    wk(:,1,ij), wk(:,2,ij), & ! [IN]
213                    GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
214
215     GMTR_a(ij,k0,1,AIJ,GMTR_a_HNX) = Nvec(1)
216     GMTR_a(ij,k0,1,AIJ,GMTR_a_HNY) = Nvec(2)
217     GMTR_a(ij,k0,1,AIJ,GMTR_a_HNZ) = Nvec(3)
218     GMTR_a(ij,k0,1,AIJ,GMTR_a_HTX) = Tvec(1)
219     GMTR_a(ij,k0,1,AIJ,GMTR_a_HTY) = Tvec(2)
220     GMTR_a(ij,k0,1,AIJ,GMTR_a_HTZ) = Tvec(3)
221 enddo
222 enddo
223
224 !--- AJ
225 do j = ADM_gmin-1, ADM_gmax
226 do i = ADM_gmin, ADM_gmax
227     ij = suf(i ,j )
228     im1j = suf(i-1,j )
229
230     do d = 1, ADM_nxyz
231         wk(d,1,ij) = GRD_xt(im1j,k0,1,TI,d)
232         wk(d,2,ij) = GRD_xt(ij ,k0,1,TJ,d)
233     enddo
234 enddo
235 enddo
236
237 if ( ADM_have_sgp(1) ) then ! pentagon
238     wk(:,1,suf(ADM_gmin ,ADM_gmin-1)) = GRD_xt(suf(ADM_gmin ,ADM_gmin ),k0,1,TI,:)
239     wk(:,2,suf(ADM_gmin ,ADM_gmin-1)) = GRD_xt(suf(ADM_gmin ,ADM_gmin-1),k0,1,TJ,:)
240 endif
241
242 do j = ADM_gmin-1, ADM_gmax
243 do i = ADM_gmin, ADM_gmax

```

```

244     ij = suf(i,j)
245
246     call GMTR_TNvec( Tvec(:), Nvec(:),                & ! [OUT]
247                   wk(:,1,ij), wk(:,2,ij),           & ! [IN]
248                   GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
249
250     GMTR_a(ij,k0,1,AJ,GMTR_a_HNX) = Nvec(1)
251     GMTR_a(ij,k0,1,AJ,GMTR_a_HNY) = Nvec(2)
252     GMTR_a(ij,k0,1,AJ,GMTR_a_HNZ) = Nvec(3)
253     GMTR_a(ij,k0,1,AJ,GMTR_a_HTX) = Tvec(1)
254     GMTR_a(ij,k0,1,AJ,GMTR_a_HTY) = Tvec(2)
255     GMTR_a(ij,k0,1,AJ,GMTR_a_HTZ) = Tvec(3)
256 enddo
257 enddo
258
259 enddo ! 1 loop
260

```

These three sections also calculate the metrics for the three arc points represented by AI, AIJ and AJ, respectively, but these sections are for the edges of hexagonal/pentagonal control cell, i.e. the line crossing the arc of triangle. After setting the coordinates of two vertex points, subroutine GMTR_TNvec calculates a normal vector and a tangent vector of the line connecting two endpoints.

The remaining part of this subroutine is for the pole region, and is as follows.

```

261 if ( ADM_have_pl ) then
262   n = ADM_gslf_pl
263
264   do l = 1, ADM_lall_pl
265
266     !--- Triangle (arc 1)
267     do v = ADM_gmin_pl, ADM_gmax_pl
268       ij = v
269
270       do d = 1, ADM_nxyz
271         wk_pl(d,1) = GRD_x_pl(n ,k0,1,d)
272         wk_pl(d,2) = GRD_x_pl(ij,k0,1,d)
273       enddo
274
275       call GMTR_TNvec( Tvec(:), Nvec(:),                & ! [OUT]
276                     wk_pl(:,1), wk_pl(:,2),           & ! [IN]
277                     GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
278
279       GMTR_a_pl(ij,k0,1,GMTR_a_TNX) = Nvec(1)
280       GMTR_a_pl(ij,k0,1,GMTR_a_TNY) = Nvec(2)
281       GMTR_a_pl(ij,k0,1,GMTR_a_TNZ) = Nvec(3)
282       GMTR_a_pl(ij,k0,1,GMTR_a_TTX) = Tvec(1)
283       GMTR_a_pl(ij,k0,1,GMTR_a_TTY) = Tvec(2)
284       GMTR_a_pl(ij,k0,1,GMTR_a_TTZ) = Tvec(3)
285     enddo
286
287     !--- Triangle (arc 2)
288     do v = ADM_gmin_pl, ADM_gmax_pl
289       ij = v
290       ijp1 = v+1
291       if ( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
292
293       do d = 1, ADM_nxyz
294         wk_pl(d,1) = GRD_x_pl(ij ,k0,1,d)
295         wk_pl(d,2) = GRD_x_pl(ijp1,k0,1,d)
296       enddo
297
298       call GMTR_TNvec( Tvec(:), Nvec(:),                & ! [OUT]
299                     wk_pl(:,1), wk_pl(:,2),           & ! [IN]
300                     GRD_grid_type, GMTR_polygon_type, GRD_rscale ) ! [IN]
301
302       GMTR_a_pl(ij,k0,1,GMTR_a_TN2X) = Nvec(1)
303       GMTR_a_pl(ij,k0,1,GMTR_a_TN2Y) = Nvec(2)
304       GMTR_a_pl(ij,k0,1,GMTR_a_TN2Z) = Nvec(3)
305       GMTR_a_pl(ij,k0,1,GMTR_a_TT2X) = Tvec(1)
306       GMTR_a_pl(ij,k0,1,GMTR_a_TT2Y) = Tvec(2)
307       GMTR_a_pl(ij,k0,1,GMTR_a_TT2Z) = Tvec(3)
308     enddo
309
310     !--- Pentagon
311     do v = ADM_gmin_pl, ADM_gmax_pl
312       ij = v
313       ijm1 = v-1
314       if ( ijm1 == ADM_gmin_pl-1 ) ijm1 = ADM_gmax_pl

```

```

315
316     do d = 1, ADM_nxyz
317         wk_pl(d,1) = GRD_xt_pl(ijm1,k0,1,d)
318         wk_pl(d,2) = GRD_xt_pl(ij ,k0,1,d)
319     enddo
320
321     call GMTR_TNvec( Tvec(:), Nvec(:),                & ! [OUT]
322                   wk_pl(:,1), wk_pl(:,2),           & ! [IN]
323                   GRD_grid_type, GMTR_polygon_type, GRD_rscales ) ! [IN]
324
325     GMTR_a_pl(ij,k0,1,GMTR_a_HNX) = Nvec(1)
326     GMTR_a_pl(ij,k0,1,GMTR_a_HNY) = Nvec(2)
327     GMTR_a_pl(ij,k0,1,GMTR_a_HNZ) = Nvec(3)
328     GMTR_a_pl(ij,k0,1,GMTR_a_HTX) = Tvec(1)
329     GMTR_a_pl(ij,k0,1,GMTR_a_HTY) = Tvec(2)
330     GMTR_a_pl(ij,k0,1,GMTR_a_HTZ) = Tvec(3)
331 enddo
332
333     enddo
334 endif
335
336     return
337 end subroutine GMTR_a_setup

```

The normal vector and the tangent vector are calculated between the pole point and neighboring grid point, and between the two neighboring grid points. Then the metrics for the edge of the pentagonal control cell are calculated by almost the same procedure using subroutine GMTR_TNvec.

(4) OPRT_divergence_setup

Module mod_oprt contains public objects related to the vector operator, such as divergence, rotation, laplacian, etc. Calculation of these vector operator is discretized to form stencil calculation, which needs stencil coefficients. This subroutine calculates the coefficients for the divergence operator, subroutine OPRT_divergence, in advance.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT_divergence_setup( &
2      GMTR_p, GMTR_p_pl, &
3      GMTR_t, GMTR_t_pl, &
4      GMTR_a, GMTR_a_pl, &
5      coef_div, coef_div_pl )
6      !ESC! use mod_adm, only: &
7      !ESC! ADM_have_pl, &
8      !ESC! ADM_have_sgp, &
9      !ESC! ADM_gall_id, &
10     !ESC! ADM_gmin, &
11     !ESC! ADM_gmax, &
12     !ESC! ADM_gslf_pl, &
13     !ESC! ADM_gmin_pl, &
14     !ESC! ADM_gmax_pl
15     !ESC! use mod_gmtr, only: &
16     !ESC! P_RAREA => GMTR_p_RAREA, &
17     !ESC! W1 => GMTR_t_W1, &
18     !ESC! W2 => GMTR_t_W2, &
19     !ESC! W3 => GMTR_t_W3, &
20     !ESC! HNX => GMTR_a_HNX, &
21     !ESC! GMTR_p_nmax, &
22     !ESC! GMTR_t_nmax, &
23     !ESC! GMTR_a_nmax, &
24     !ESC! GMTR_a_nmax_pl
25     implicit none
26
27     real(RP), intent(in) :: GMTR_p (ADM_gall ,KO,ADM_lall , GMTR_p_nmax )
28     real(RP), intent(in) :: GMTR_p_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_p_nmax )
29     real(RP), intent(in) :: GMTR_t (ADM_gall ,KO,ADM_lall ,TI:TJ,GMTR_t_nmax )
30     real(RP), intent(in) :: GMTR_t_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_t_nmax )
31     real(RP), intent(in) :: GMTR_a (ADM_gall ,KO,ADM_lall ,AI:AJ,GMTR_a_nmax )
32     real(RP), intent(in) :: GMTR_a_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_a_nmax_pl)
33     real(RP), intent(out) :: coef_div (ADM_gall,0:6 ,ADM_nxyz,ADM_lall )
34     real(RP), intent(out) :: coef_div_pl( 0:ADM_vlink,ADM_nxyz,ADM_lall_pl)
35
36     integer :: gmin, gmax, iall, gall, nxyz, lall
37
38     integer :: ij
39     integer :: ip1j, ijp1, ip1jp1

```

```

40 integer :: imlj, ijm1, im1jm1
41
42 real(RP) :: coef
43 integer :: g, l, d, n, v, hn
44 -----
45

```

Input arguments are the metrics calculated by the subroutine described before in this section. Output arguments `coef_div` and `coef_div_pl` are the coefficients for the divergence operator for the normal region and the pole region, respectively. The second dimension of `coef_div` has the range 0:6, which corresponds to the 7-point stencil calculation. Also the second dimension of `coef_div_pl` has the range 0:ADM_vlink where `ADM_vlink` is 5, corresponds to the 6-point stencil calculation. Note that the pole region is a pentagon.

The procedure of this subroutine is consist of three parts. The first part is as follows.

```

46 !if( IO_L ) write(IO_FID_LOG,*) '*** setup coefficient of divergence operator'
47
48 gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
49 gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
50 iall = ADM_gall_1d
51 gall = ADM_gall
52 nxyz = ADM_nxyz
53 lall = ADM_lall
54
55 !$omp parallel workshare
56 coef_div (:,:,,:) = 0.0_RP
57 !$omp end parallel workshare
58 coef_div_pl( :,:, ) = 0.0_RP
59
60 !$omp parallel default(none),private(g,d,l,hn,ij,ip1j,ip1jp1,ijp1,imlj,ijm1,im1jm1), &
61 !$omp shared(ADM_have_sgp,gmin,gmax,iall,gall,nxyz,lall,coef_div,GMTR_p,GMTR_t,GMTR_a)
62 do l = 1, lall
63 do d = 1, nxyz
64 hn = d + HNX - 1
65
66 !$omp do
67 do g = gmin, gmax
68 ij = g
69 ip1j = g + 1
70 ip1jp1 = g + iall + 1
71 ijp1 = g + iall
72 imlj = g - 1
73 im1jm1 = g - iall - 1
74 ijm1 = g - iall
75
76 ! ij
77 coef_div(ij,0,d,l) = ( + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
78 + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
79 + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
80 + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
81 + GMTR_t(imlj ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
82 - GMTR_t(imlj ,k0,1,TI,W2) * GMTR_a(imlj ,k0,1,AI ,hn) & ! Q3 * b3
83 - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(imlj ,k0,1,AI ,hn) & ! Q4 * b3
84 - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
85 - GMTR_t(im1jm1,k0,1,TI,W3) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q5 * b4
86 - GMTR_t(im1jm1,k0,1,TI,W3) * GMTR_a(ijm1 ,k0,1,AJ ,hn) & ! Q5 * b5
87 - GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(ijm1 ,k0,1,AJ ,hn) & ! Q6 * b5
88 + GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
89 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
90
91 ! ip1j
92 coef_div(ij,1,d,l) = ( - GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(ijm1 ,k0,1,AJ ,hn) & ! Q6 * b5
93 + GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
94 + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
95 + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
96 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
97
98 ! ip1jp1
99 coef_div(ij,2,d,l) = ( + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
100 + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
101 + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
102 + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
103 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
104
105 ! ijp1
106 coef_div(ij,3,d,l) = ( + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
107 + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
108 + GMTR_t(imlj ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
109 - GMTR_t(imlj ,k0,1,TI,W3) * GMTR_a(imlj ,k0,1,AI ,hn) & ! Q3 * b3
110 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
111
112 ! imlj

```

```

109     coef_div(ij,4,d,1) = ( + GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
110                       - GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
111                       - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
112                       - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
113                       ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
114     ! im1jm1
115     coef_div(ij,5,d,1) = ( - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
116                       - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
117                       - GMTR_t(im1jm1,k0,1,TI,W1) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q5 * b4
118                       - GMTR_t(im1jm1,k0,1,TI,W1) * GMTR_a(ijm1 ,k0,1,AJ ,hn) & ! Q5 * b5
119                       ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
120     ! ijm1
121     coef_div(ij,6,d,1) = ( - GMTR_t(im1jm1,k0,1,TI,W2) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q5 * b4
122                       - GMTR_t(im1jm1,k0,1,TI,W2) * GMTR_a(ijm1 ,k0,1,AJ ,hn) & ! Q5 * b5
123                       - GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(ijm1 ,k0,1,AJ ,hn) & ! Q6 * b5
124                       + GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
125                       ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
126     enddo
127     !$omp end do
128

```

The first part above and the second part below are in a long l -loop. In the g -loop represents horizontal index, 7 coefficients are calculated separately from the various metrics.

The second part is as follows.

```

129     if ( ADM_have_sgp(1) ) then ! pentagon
130         !$omp master
131         ij = gmin
132         ip1j = gmin + 1
133         ip1jp1 = gmin + iall + 1
134         ijp1 = gmin + iall
135         im1j = gmin - 1
136         im1jm1 = gmin - iall - 1
137         ijm1 = gmin - iall
138
139         ! ij
140         coef_div(ij,0,d,1) = ( + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
141                           + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
142                           + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
143                           + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
144                           + GMTR_t(im1j ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
145                           - GMTR_t(im1j ,k0,1,TI,W2) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
146                           - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
147                           - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
148                           - GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q6 * b4
149                           + GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
150                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
151
152         ! ip1j
153         coef_div(ij,1,d,1) = ( - GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q6 * b4
154                           + GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
155                           + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
156                           + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
157                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
158
159         ! ip1jp1
160         coef_div(ij,2,d,1) = ( + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
161                           + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
162                           + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
163                           + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
164                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
165
166         ! ijp1
167         coef_div(ij,3,d,1) = ( + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
168                           + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
169                           - GMTR_t(im1j ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
170                           - GMTR_t(im1j ,k0,1,TI,W3) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
171                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
172
173         ! im1j
174         coef_div(ij,4,d,1) = ( + GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
175                           - GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
176                           - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
177                           - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
178                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
179
180         ! im1jm1
181         coef_div(ij,5,d,1) = ( - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
182                           - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
183                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
184
185         ! ijm1
186         coef_div(ij,6,d,1) = ( - GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q6 * b4
187                           + GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
188                           ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)

```

```

183      !$omp end master
184      endif
185
186      enddo ! loop d
187      enddo ! loop l
188      !$omp end parallel

```

This section is for the singular point. Note that `ADM_have_sgp(1)` is true.

The last section of this subroutine is for the pole region and is as follows.

```

189      if ( ADM_have_pl ) then
190          n = ADM_gslf_pl
191          do l = 1, ADM_lall_pl
192              do d = 1, ADM_nxyz
193                  hn = d + HNX - 1
194
195                  coef = 0.0_RP
196                  do v = ADM_gmin_pl, ADM_gmax_pl
197                      ij = v
198                      ijp1 = v + 1
199                      if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
200
201                      coef = coef + ( GMTR_t_pl(ij,k0,l,W1) * GMTR_a_pl(ij ,k0,l,hn) &
202                                  + GMTR_t_pl(ij,k0,l,W1) * GMTR_a_pl(ijp1,k0,l,hn) )
203
204                      enddo
205                      coef_div_pl(0,d,l) = coef * 0.5_RP * GMTR_p_pl(n,k0,l,P_RAREA)
206
207                      do v = ADM_gmin_pl, ADM_gmax_pl
208                          ij = v
209                          ijp1 = v + 1
210                          ijm1 = v - 1
211                          if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
212                          if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl
213
214                          coef_div_pl(v-1,d,l) = ( + GMTR_t_pl(ijm1,k0,l,W3) * GMTR_a_pl(ijm1,k0,l,hn) &
215                                                  + GMTR_t_pl(ijm1,k0,l,W3) * GMTR_a_pl(ij ,k0,l,hn) &
216                                                  + GMTR_t_pl(ij ,k0,l,W2) * GMTR_a_pl(ij ,k0,l,hn) &
217                                                  + GMTR_t_pl(ij ,k0,l,W2) * GMTR_a_pl(ijp1,k0,l,hn) &
218                                                  ) * 0.5_RP * GMTR_p_pl(n,k0,l,P_RAREA)
219
220                      enddo ! loop d
221                      enddo ! loop l
222                  endif
223              return
224          end subroutine OPRT_divergence_setup

```

Note the data layout of coefficient is slightly different between the normal region and the pole region. First dimension of `coef_div_pl` corresponds to the second dimension of `coef_div`. The range of index `v` of the inner-most loop are `ADM_gmin_pl` and `ADM_gmax_pl`, which means the five grid points surrounding the pole point.

(5) OPRT_rotation_setup

This subroutine is similar to the previous one, but this one is for the rotation operator, subroutine `OPRT_rotation`.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT_rotation_setup( &
2      GMTR_p, GMTR_p_pl, &
3      GMTR_t, GMTR_t_pl, &
4      GMTR_a, GMTR_a_pl, &
5      coef_rot, coef_rot_pl )
6      !ESC! use mod_adm, only: &
7      !ESC! ADM_have_pl, &
8      !ESC! ADM_have_sgp, &
9      !ESC! ADM_gall_id, &
10     !ESC! ADM_gmin, &
11     !ESC! ADM_gmax, &
12     !ESC! ADM_gslf_pl, &
13     !ESC! ADM_gmin_pl, &
14     !ESC! ADM_gmax_pl
15     !ESC! use mod_gmtr, only: &
16     !ESC! P_RAREA => GMTR_p_RAREA, &

```



```

17 !ESC!      W1      => GMTR_t_w1,  &
18 !ESC!      W2      => GMTR_t_w2,  &
19 !ESC!      W3      => GMTR_t_w3,  &
20 !ESC!      HTX     => GMTR_a_HTX, &
21 !ESC!      GMTR_p_nmax, &
22 !ESC!      GMTR_t_nmax, &
23 !ESC!      GMTR_a_nmax, &
24 !ESC!      GMTR_a_nmax_pl
25 implicit none
26
27 real(RP), intent(in) :: GMTR_p (ADM_gall ,KO,ADM_lall , GMTR_p_nmax )
28 real(RP), intent(in) :: GMTR_p_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_p_nmax )
29 real(RP), intent(in) :: GMTR_t (ADM_gall ,KO,ADM_lall ,TI:TJ,GMTR_t_nmax )
30 real(RP), intent(in) :: GMTR_t_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_t_nmax )
31 real(RP), intent(in) :: GMTR_a (ADM_gall ,KO,ADM_lall ,AI:AJ,GMTR_a_nmax )
32 real(RP), intent(in) :: GMTR_a_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_a_nmax_pl)
33 real(RP), intent(out) :: coef_rot (ADM_gall,0:6 ,ADM_nxyz,ADM_lall )
34 real(RP), intent(out) :: coef_rot_pl( 0:ADM_vlink,ADM_nxyz,ADM_lall_pl)
35
36 integer :: gmin, gmax, iall, gall, nxyz, lall
37
38 integer :: ij
39 integer :: ip1j, ijp1, ip1jp1
40 integer :: im1j, imj1, im1jm1
41
42 real(RP) :: coef
43 integer :: g, l, d, n, v, ht
44 !-----
45

```

Input arguments are the same with subroutine OPRT_divergence_setup, the metrics calculated by the subroutine described before in this section. Output arguments coef_rot and coef_rot_pl are the coefficients for the rotation operator for the normal region and the pole region, respectively. The second dimension of coef_rot has the range 0:6, which corresponds to the 7-point stencil calculation. Also the second dimension of coef_rot_pl has the range 0:ADM_vlink where ADM_vlink is 5, corresponds to the 6-point stencil calculation.

The procedure of this subroutine is consist of three parts. The first part is as follows.

```

46 !if( IO_L ) write(IO_FID_LOG,*) '*** setup coefficient of rotation operator'
47
48 gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
49 gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
50 iall = ADM_gall_1d
51 gall = ADM_gall
52 nxyz = ADM_nxyz
53 lall = ADM_lall
54
55 !$omp parallel workshare
56 coef_rot (:,:,,:) = 0.0_RP
57 !$omp end parallel workshare
58 coef_rot_pl( :,:,) = 0.0_RP
59
60 !$omp parallel default(none),private(g,d,l,ht,ij,ip1j,ip1jp1,ijp1,im1j,imj1,im1jm1), &
61 !$omp shared(ADM_have_sgp,gmin,gmax,iall,gall,nxyz,lall,coef_rot,GMTR_p,GMTR_t,GMTR_a)
62 do l = 1, lall
63 do d = 1, nxyz
64 ht = d + HTX - 1
65
66 !$omp do
67 do g = gmin, gmax
68 ij = g
69 ip1j = g + 1
70 ip1jp1 = g + iall + 1
71 ijp1 = g + iall
72 im1j = g - 1
73 im1jm1 = g - iall - 1
74 imj1 = g - iall
75
76 ! ij
77 coef_rot(ij,0,d,l) = ( + GMTR_t(ij ,k0,l,TI,W1) * GMTR_a(ij ,k0,l,AI ,ht) & ! Q1 * b6
78 + GMTR_t(ij ,k0,l,TI,W1) * GMTR_a(ij ,k0,l,AIJ,ht) & ! Q1 * b1
79 + GMTR_t(ij ,k0,l,TJ,W1) * GMTR_a(ij ,k0,l,AIJ,ht) & ! Q2 * b1
80 + GMTR_t(ij ,k0,l,TJ,W1) * GMTR_a(ij ,k0,l,AJ ,ht) & ! Q2 * b2
81 + GMTR_t(im1j ,k0,l,TI,W2) * GMTR_a(ij ,k0,l,AJ ,ht) & ! Q3 * b2
82 - GMTR_t(im1j ,k0,l,TI,W2) * GMTR_a(im1j ,k0,l,AI ,ht) & ! Q3 * b3
83 - GMTR_t(im1jm1,k0,l,TJ,W2) * GMTR_a(im1j ,k0,l,AI ,ht) & ! Q4 * b3
84 - GMTR_t(im1jm1,k0,l,TJ,W2) * GMTR_a(im1jm1,k0,l,AIJ,ht) & ! Q4 * b4

```

```

85 - GMTR_t(im1jm1,k0,1,TI,W3) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q5 * b4
86 - GMTR_t(im1jm1,k0,1,TI,W3) * GMTR_a(im1jm1 ,k0,1,AJ ,ht) & ! Q5 * b5
87 - GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(im1jm1 ,k0,1,AJ ,ht) & ! Q6 * b5
88 + GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q6 * b6
89 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
90
91 ! ip1j
92 coef_rot(ij,1,d,1) = ( - GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(im1jm1 ,k0,1,AJ ,ht) & ! Q6 * b5
93 + GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q6 * b6
94 + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q1 * b6
95 + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q1 * b1
96 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
97
98 ! ip1jpl
99 coef_rot(ij,2,d,1) = ( + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q1 * b6
100 + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q1 * b1
101 + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q2 * b1
102 + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AJ ,ht) & ! Q2 * b2
103 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
104
105 ! ijpl
106 coef_rot(ij,3,d,1) = ( + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q2 * b1
107 + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AJ ,ht) & ! Q2 * b2
108 + GMTR_t(im1j ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AJ ,ht) & ! Q3 * b2
109 - GMTR_t(im1j ,k0,1,TI,W3) * GMTR_a(im1j ,k0,1,AI ,ht) & ! Q3 * b3
110 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
111
112 ! im1j
113 coef_rot(ij,4,d,1) = ( + GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AJ ,ht) & ! Q3 * b2
114 - GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(im1j ,k0,1,AI ,ht) & ! Q3 * b3
115 - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1j ,k0,1,AI ,ht) & ! Q4 * b3
116 - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q4 * b4
117 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
118
119 ! im1jm1
120 coef_rot(ij,5,d,1) = ( - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1j ,k0,1,AI ,ht) & ! Q4 * b3
121 - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q4 * b4
122 - GMTR_t(im1jm1,k0,1,TI,W1) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q5 * b4
123 - GMTR_t(im1jm1,k0,1,TI,W1) * GMTR_a(ijm1 ,k0,1,AJ ,ht) & ! Q5 * b5
124 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
125
126 ! ijm1
127 coef_rot(ij,6,d,1) = ( - GMTR_t(im1jm1,k0,1,TI,W2) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q5 * b4
128 - GMTR_t(im1jm1,k0,1,TI,W2) * GMTR_a(ijm1 ,k0,1,AJ ,ht) & ! Q5 * b5
129 - GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(ijm1 ,k0,1,AJ ,ht) & ! Q6 * b5
130 + GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q6 * b6
131 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
132
133 enddo
134 !$omp end do
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158

```

The first part above and the second part below are in a long l -loop. In the g -loop represents horizontal index, 7 coefficients are calculated separately from the various metrics.

The second part is as follows.

```

129 if ( ADM_have_sgp(l) ) then ! pentagon
130 !$omp master
131 ij = gmin
132 ip1j = gmin + 1
133 ip1jpl = gmin + iall + 1
134 ijpl = gmin + iall
135 im1j = gmin - 1
136 im1jm1 = gmin - iall - 1
137 ijm1 = gmin - iall
138
139 ! ij
140 coef_rot(ij,0,d,1) = ( + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q1 * b6
141 + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q1 * b1
142 + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q2 * b1
143 + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AJ ,ht) & ! Q2 * b2
144 + GMTR_t(im1j ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AJ ,ht) & ! Q3 * b2
145 - GMTR_t(im1j ,k0,1,TI,W2) * GMTR_a(im1j ,k0,1,AI ,ht) & ! Q3 * b3
146 - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1j ,k0,1,AI ,ht) & ! Q4 * b3
147 - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q4 * b4
148 - GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q6 * b4
149 + GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q6 * b6
150 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
151
152 ! ip1j
153 coef_rot(ij,1,d,1) = ( - GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q6 * b4
154 + GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q6 * b6
155 + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q1 * b6
156 + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AIJ,ht) & ! Q1 * b1
157 ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
158
159 ! ip1jpl
160 coef_rot(ij,2,d,1) = ( + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AI ,ht) & ! Q1 * b6

```

```

159         + GMTR_t(ij      ,k0,1,TI,W3) * GMTR_a(ij      ,k0,1,AIJ,ht) & ! Q1 * b1
160         + GMTR_t(ij      ,k0,1,TJ,W2) * GMTR_a(ij      ,k0,1,AIJ,ht) & ! Q2 * b1
161         + GMTR_t(ij      ,k0,1,TJ,W2) * GMTR_a(ij      ,k0,1,AJ ,ht) & ! Q2 * b2
162         ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
163     ! ijp1
164     coef_rot(ij,3,d,1) = ( + GMTR_t(ij      ,k0,1,TJ,W3) * GMTR_a(ij      ,k0,1,AIJ,ht) & ! Q2 * b1
165         + GMTR_t(ij      ,k0,1,TJ,W3) * GMTR_a(ij      ,k0,1,AJ ,ht) & ! Q2 * b2
166         + GMTR_t(im1j    ,k0,1,TI,W3) * GMTR_a(ij      ,k0,1,AJ ,ht) & ! Q3 * b2
167         - GMTR_t(im1j    ,k0,1,TI,W3) * GMTR_a(im1j    ,k0,1,AI ,ht) & ! Q3 * b3
168         ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
169     ! im1j
170     coef_rot(ij,4,d,1) = ( + GMTR_t(im1j    ,k0,1,TI,W1) * GMTR_a(ij      ,k0,1,AJ ,ht) & ! Q3 * b2
171         - GMTR_t(im1j    ,k0,1,TI,W1) * GMTR_a(im1j    ,k0,1,AI ,ht) & ! Q3 * b3
172         - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1j    ,k0,1,AI ,ht) & ! Q4 * b3
173         - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q4 * b4
174         ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
175     ! im1jm1
176     coef_rot(ij,5,d,1) = ( - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1j    ,k0,1,AI ,ht) & ! Q4 * b3
177         - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q4 * b4
178         ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
179     ! ijm1
180     coef_rot(ij,6,d,1) = ( - GMTR_t(ijm1    ,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,ht) & ! Q6 * b4
181         + GMTR_t(ijm1    ,k0,1,TJ,W1) * GMTR_a(ij      ,k0,1,AI ,ht) & ! Q6 * b6
182         ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
183     !$omp end master
184     endif
185
186     enddo ! loop d
187     enddo ! loop l
188     !$omp end parallel
189

```

This section is for the singular point. Note that `ADM_have_sgp(1)` is true.

The last section of this subroutine is for the pole region and is as follows.

```

190     if ( ADM_have_pl ) then
191         n = ADM_gs1f_pl
192         do l = 1, ADM_lall_pl
193             do d = 1, ADM_nxyz
194                 ht = d + HTX - 1
195
196                 coef = 0.0_RP
197                 do v = ADM_gmin_pl, ADM_gmax_pl
198                     ij = v
199                     ijp1 = v + 1
200                     if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
201
202                     coef = coef + ( GMTR_t_pl(ij,k0,1,W1) * GMTR_a_pl(ij ,k0,1,ht) &
203                         + GMTR_t_pl(ij,k0,1,W1) * GMTR_a_pl(ijp1,k0,1,ht) )
204                 enddo
205                 coef_rot_pl(0,d,1) = coef * 0.5_RP * GMTR_p_pl(n,k0,1,P_RAREA)
206
207                 do v = ADM_gmin_pl, ADM_gmax_pl
208                     ij = v
209                     ijp1 = v + 1
210                     ijm1 = v - 1
211                     if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
212                     if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl
213
214                     coef_rot_pl(v-1,d,1) = ( + GMTR_t_pl(ijm1,k0,1,W3) * GMTR_a_pl(ijm1,k0,1,ht) &
215                         + GMTR_t_pl(ijm1,k0,1,W3) * GMTR_a_pl(ij ,k0,1,ht) &
216                         + GMTR_t_pl(ij ,k0,1,W2) * GMTR_a_pl(ij ,k0,1,ht) &
217                         + GMTR_t_pl(ij ,k0,1,W2) * GMTR_a_pl(ijp1,k0,1,ht) &
218                         ) * 0.5_RP * GMTR_p_pl(n,k0,1,P_RAREA)
219                 enddo
220             enddo ! loop d
221         enddo ! loop l
222     endif
223
224     return
225 end subroutine OPRT_rotation_setup

```

Note the data layout of coefficient is slightly different between the normal region and the pole region, First dimension of `coef_rot_pl` corresponds to the second dimension of `coef_rot`. The range of index `v` of the inner-most loop are `ADM_gmin_pl` and `ADM_gmax_pl`, which means the five grid points surrounding the pole point.

(6) OPRT_gradient_setup

This subroutine is also similar to the previous subroutines, but this one is for the gradient operator, subroutine OPRT_gradient.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT_gradient_setup( &
2     GMTR_p,  GMTR_p_pl,  &
3     GMTR_t,  GMTR_t_pl,  &
4     GMTR_a,  GMTR_a_pl,  &
5     coef_grad, coef_grad_pl )
6  !ESC!      use mod_adm, only: &
7  !ESC!      ADM_have_pl,  &
8  !ESC!      ADM_have_sgp, &
9  !ESC!      ADM_gall_id,  &
10 !ESC!      ADM_gmin,    &
11 !ESC!      ADM_gmax,    &
12 !ESC!      ADM_gslf_pl, &
13 !ESC!      ADM_gmin_pl, &
14 !ESC!      ADM_gmax_pl
15 !ESC!      use mod_gmtr, only: &
16 !ESC!      P_RAREA => GMTR_p_RAREA, &
17 !ESC!      W1      => GMTR_t_W1,    &
18 !ESC!      W2      => GMTR_t_W2,    &
19 !ESC!      W3      => GMTR_t_W3,    &
20 !ESC!      HNX     => GMTR_a_HNX,    &
21 !ESC!      GMTR_p_nmax, &
22 !ESC!      GMTR_t_nmax, &
23 !ESC!      GMTR_a_nmax, &
24 !ESC!      GMTR_a_nmax_pl
25 implicit none
26
27 real(RP), intent(in) :: GMTR_p      (ADM_gall ,KO,ADM_lall , GMTR_p_nmax )
28 real(RP), intent(in) :: GMTR_p_pl  (ADM_gall_pl,KO,ADM_lall_pl, GMTR_p_nmax )
29 real(RP), intent(in) :: GMTR_t      (ADM_gall ,KO,ADM_lall ,TI:TJ,GMTR_t_nmax )
30 real(RP), intent(in) :: GMTR_t_pl  (ADM_gall_pl,KO,ADM_lall_pl, GMTR_t_nmax )
31 real(RP), intent(in) :: GMTR_a      (ADM_gall ,KO,ADM_lall ,AI:AJ,GMTR_a_nmax )
32 real(RP), intent(in) :: GMTR_a_pl  (ADM_gall_pl,KO,ADM_lall_pl, GMTR_a_nmax_pl)
33 real(RP), intent(out) :: coef_grad  (ADM_gall,0:6 ,ADM_nxyz,ADM_lall )
34 real(RP), intent(out) :: coef_grad_pl( 0:ADM_vlink,ADM_nxyz,ADM_lall_pl)
35
36 integer :: gmin, gmax, iall, gall, nxyz, lall
37
38 integer :: ij
39 integer :: ip1j, ijp1, ip1jp1
40 integer :: im1j, ijm1, im1jm1
41
42 real(RP) :: coef
43 integer :: g, l, d, n, v, hn
44 !-----
45

```

Input arguments are the same with subroutine OPRT_divergence_setup and OPRT_rotation_setup, the metrics calculated by the subroutines described before in this section. Output arguments `coef_grad` and `coef_grad_pl` are the coefficients for the gradient operator for the normal region and the pole region, respectively. The second dimension of `coef_grad` has the range 0:6, which corresponds to the 7-point stencil calculation. Also the second dimension of `coef_grad_pl` has the range 0:ADM_vlink where ADM_vlink is 5, corresponds to the 6-point stencil calculation. Note that the pole region is a pentagon.

The procedure of this subroutine is consist of three parts. The first part is as follows.

```

46  !if( IO_L ) write(IO_FID_LOG,*) '*** setup coefficient of gradient operator'
47
48  gmin = (ADM_gmin-1)*ADM_gall_id + ADM_gmin
49  gmax = (ADM_gmax-1)*ADM_gall_id + ADM_gmax
50  iall = ADM_gall_id
51  gall = ADM_gall
52  nxyz = ADM_nxyz
53  lall = ADM_lall
54
55  !$omp parallel workshare
56  coef_grad  (:,:,,:) = 0.0_RP
57  !$omp end parallel workshare
58  coef_grad_pl( :,:, ) = 0.0_RP
59

```

```

60 ! $omp parallel default (none), private (g, d, l, hn, ij, ip1j, ip1jp1, ijp1, im1j, im1jm1), &
61 ! $omp shared (ADM_have_sgp, gmin, gmax, iall, gall, nxyz, lall, coef_grad, GMTR_p, GMTR_t, GMTR_a)
62 do l = 1, lall
63 do d = 1, nxyz
64   hn = d + HNX - 1
65
66   ! $omp do
67   do g = gmin, gmax
68     ij = g
69     ip1j = g + 1
70     ip1jp1 = g + iall + 1
71     ijp1 = g + iall
72     im1j = g - 1
73     im1jm1 = g - iall - 1
74     ijm1 = g - iall
75
76     ! ij
77     coef_grad(ij, 0, d, l) = ( + GMTR_t(ij, k0, 1, TI, W1) * GMTR_a(ij, k0, 1, AI, hn) & ! Q1 * b6
78       + GMTR_t(ij, k0, 1, TI, W1) * GMTR_a(ij, k0, 1, AIJ, hn) & ! Q1 * b1
79       + GMTR_t(ij, k0, 1, TJ, W1) * GMTR_a(ij, k0, 1, AIJ, hn) & ! Q2 * b1
80       + GMTR_t(ij, k0, 1, TJ, W1) * GMTR_a(ij, k0, 1, AJ, hn) & ! Q2 * b2
81       + GMTR_t(im1j, k0, 1, TI, W2) * GMTR_a(ij, k0, 1, AJ, hn) & ! Q3 * b2
82       - GMTR_t(im1j, k0, 1, TI, W2) * GMTR_a(im1j, k0, 1, AI, hn) & ! Q3 * b3
83       - GMTR_t(im1jm1, k0, 1, TJ, W2) * GMTR_a(im1j, k0, 1, AI, hn) & ! Q4 * b3
84       - GMTR_t(im1jm1, k0, 1, TI, W3) * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! Q4 * b4
85       - GMTR_t(im1jm1, k0, 1, TJ, W2) * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! Q5 * b4
86       - GMTR_t(im1jm1, k0, 1, TI, W3) * GMTR_a(im1jm1, k0, 1, AJ, hn) & ! Q5 * b5
87       - GMTR_t(ijm1, k0, 1, TJ, W3) * GMTR_a(ijm1, k0, 1, AJ, hn) & ! Q6 * b5
88       + GMTR_t(ijm1, k0, 1, TJ, W3) * GMTR_a(ij, k0, 1, AI, hn) & ! Q6 * b6
89       - 2.0_RP * GMTR_a(ij, k0, 1, AIJ, hn) & ! P0 * b1
90       - 2.0_RP * GMTR_a(ij, k0, 1, AJ, hn) & ! P0 * b2
91       + 2.0_RP * GMTR_a(im1j, k0, 1, AI, hn) & ! P0 * b3
92       + 2.0_RP * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! P0 * b4
93       + 2.0_RP * GMTR_a(ijm1, k0, 1, AJ, hn) & ! P0 * b5
94       - 2.0_RP * GMTR_a(ij, k0, 1, AI, hn) & ! P0 * b6
95     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
96
97     ! ip1j
98     coef_grad(ij, 1, d, l) = ( - GMTR_t(ijm1, k0, 1, TJ, W2) * GMTR_a(ijm1, k0, 1, AJ, hn) & ! Q6 * b5
99       + GMTR_t(ijm1, k0, 1, TJ, W2) * GMTR_a(ij, k0, 1, AI, hn) & ! Q6 * b6
100      + GMTR_t(ij, k0, 1, TI, W2) * GMTR_a(ij, k0, 1, AI, hn) & ! Q1 * b6
101      + GMTR_t(ij, k0, 1, TI, W2) * GMTR_a(ij, k0, 1, AIJ, hn) & ! Q1 * b1
102     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
103
104     ! ip1jp1
105     coef_grad(ij, 2, d, l) = ( + GMTR_t(ij, k0, 1, TI, W3) * GMTR_a(ij, k0, 1, AI, hn) & ! Q1 * b6
106       + GMTR_t(ij, k0, 1, TI, W3) * GMTR_a(ij, k0, 1, AIJ, hn) & ! Q1 * b1
107       + GMTR_t(ij, k0, 1, TJ, W2) * GMTR_a(ij, k0, 1, AIJ, hn) & ! Q2 * b1
108       + GMTR_t(ij, k0, 1, TJ, W2) * GMTR_a(ij, k0, 1, AJ, hn) & ! Q2 * b2
109     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
110
111     ! ijp1
112     coef_grad(ij, 3, d, l) = ( + GMTR_t(ij, k0, 1, TJ, W3) * GMTR_a(ij, k0, 1, AIJ, hn) & ! Q2 * b1
113       + GMTR_t(ij, k0, 1, TJ, W3) * GMTR_a(ij, k0, 1, AJ, hn) & ! Q2 * b2
114       + GMTR_t(im1j, k0, 1, TI, W3) * GMTR_a(ij, k0, 1, AJ, hn) & ! Q3 * b2
115       - GMTR_t(im1j, k0, 1, TI, W3) * GMTR_a(im1j, k0, 1, AI, hn) & ! Q3 * b3
116     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
117
118     ! im1j
119     coef_grad(ij, 4, d, l) = ( + GMTR_t(im1j, k0, 1, TI, W1) * GMTR_a(ij, k0, 1, AJ, hn) & ! Q3 * b2
120       - GMTR_t(im1j, k0, 1, TI, W1) * GMTR_a(im1j, k0, 1, AI, hn) & ! Q3 * b3
121       - GMTR_t(im1jm1, k0, 1, TJ, W3) * GMTR_a(im1j, k0, 1, AI, hn) & ! Q4 * b3
122       - GMTR_t(im1jm1, k0, 1, TJ, W3) * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! Q4 * b4
123     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
124
125     ! im1jm1
126     coef_grad(ij, 5, d, l) = ( - GMTR_t(im1jm1, k0, 1, TJ, W1) * GMTR_a(im1j, k0, 1, AI, hn) & ! Q4 * b3
127       - GMTR_t(im1jm1, k0, 1, TJ, W1) * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! Q4 * b4
128       - GMTR_t(im1jm1, k0, 1, TI, W1) * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! Q5 * b4
129       - GMTR_t(im1jm1, k0, 1, TI, W1) * GMTR_a(ijm1, k0, 1, AJ, hn) & ! Q5 * b5
130     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
131
132     ! ijm1
133     coef_grad(ij, 6, d, l) = ( - GMTR_t(im1jm1, k0, 1, TI, W2) * GMTR_a(im1jm1, k0, 1, AIJ, hn) & ! Q5 * b4
134       - GMTR_t(im1jm1, k0, 1, TI, W2) * GMTR_a(ijm1, k0, 1, AJ, hn) & ! Q5 * b5
135       - GMTR_t(ijm1, k0, 1, TJ, W1) * GMTR_a(ijm1, k0, 1, AJ, hn) & ! Q6 * b5
136       + GMTR_t(ijm1, k0, 1, TJ, W1) * GMTR_a(ij, k0, 1, AI, hn) & ! Q6 * b6
137     ) * 0.5_RP * GMTR_p(ij, k0, 1, P_RAREA)
138
139   enddo
140 ! $omp end do

```

The first part above and the second part below are in a long l -loop. In the g -loop represents horizontal index, 7 coefficients are calculated separately from the various metrics.

The second part is as follows.

```

135 if ( ADM_have_sgp(1) ) then ! pentagon
136   !$omp master
137   ij = gmin
138   ip1j = gmin + 1
139   ip1j1 = gmin + iall + 1
140   ijp1 = gmin + iall
141   im1j = gmin - 1
142   im1jm1 = gmin - iall - 1
143   ijm1 = gmin - iall
144
145   ! ij
146   coef_grad(ij,0,d,1) = ( + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
147     + GMTR_t(ij ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
148     + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
149     + GMTR_t(ij ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
150     + GMTR_t(im1j ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
151     - GMTR_t(im1j ,k0,1,TI,W2) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
152     - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
153     - GMTR_t(im1jm1,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
154     - GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q6 * b4
155     + GMTR_t(ijm1 ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
156     - 2.0_RP * GMTR_a(ij ,k0,1,AIJ,hn) & ! PO * b1
157     - 2.0_RP * GMTR_a(ij ,k0,1,AJ ,hn) & ! PO * b2
158     + 2.0_RP * GMTR_a(im1j ,k0,1,AI ,hn) & ! PO * b3
159     + 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! PO * b4
160     - 2.0_RP * GMTR_a(ij ,k0,1,AI ,hn) & ! PO * b6
161     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
162
163   ! ip1j
164   coef_grad(ij,1,d,1) = ( - GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q6 * b4
165     + GMTR_t(ijm1 ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
166     + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
167     + GMTR_t(ij ,k0,1,TI,W2) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
168     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
169
170   ! ip1j1
171   coef_grad(ij,2,d,1) = ( + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q1 * b6
172     + GMTR_t(ij ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q1 * b1
173     + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
174     + GMTR_t(ij ,k0,1,TJ,W2) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
175     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
176
177   ! ijp1
178   coef_grad(ij,3,d,1) = ( + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AIJ,hn) & ! Q2 * b1
179     + GMTR_t(ij ,k0,1,TJ,W3) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q2 * b2
180     + GMTR_t(im1j ,k0,1,TI,W3) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
181     - GMTR_t(im1j ,k0,1,TI,W3) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
182     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
183
184   ! im1j
185   coef_grad(ij,4,d,1) = ( + GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(ij ,k0,1,AJ ,hn) & ! Q3 * b2
186     - GMTR_t(im1j ,k0,1,TI,W1) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q3 * b3
187     - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
188     - GMTR_t(im1jm1,k0,1,TJ,W3) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
189     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
190
191   ! im1jm1
192   coef_grad(ij,5,d,1) = ( - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1j ,k0,1,AI ,hn) & ! Q4 * b3
193     - GMTR_t(im1jm1,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q4 * b4
194     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
195
196   ! ijm1
197   coef_grad(ij,6,d,1) = ( - GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(im1jm1,k0,1,AIJ,hn) & ! Q6 * b4
198     + GMTR_t(ijm1 ,k0,1,TJ,W1) * GMTR_a(ij ,k0,1,AI ,hn) & ! Q6 * b6
199     ) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
200
201   !$omp end master
202   endif
203
204   enddo ! loop d
205   enddo ! loop l
206   !$omp end parallel

```

This section is for the singular point. Note that ADM_have_sgp(1) is true.

The last section of this subroutine is for the pole region and is as follows.

```

201 if ( ADM_have_pl ) then
202   n = ADM_gslf_pl
203
204   do l = 1, ADM_lall_pl
205     do d = 1, ADM_nxyz
206       hn = d + HNX - 1
207
208       coef = 0.0_RP

```

```

209     do v = ADM_gmin_pl, ADM_gmax_pl
210         ij = v
211         ijp1 = v + 1
212         if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
213
214         coef = coef + 2.0_RP * ( GMTR_t_pl(ij,k0,l,W1) - 1.0_RP ) * GMTR_a_pl(ijp1,k0,l,hn)
215     enddo
216     coef_grad_pl(0,d,l) = coef * 0.5_RP * GMTR_p_pl(n,k0,l,P_RAREA)
217
218     do v = ADM_gmin_pl, ADM_gmax_pl
219         ij = v
220         ijp1 = v + 1
221         ijm1 = v - 1
222         if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
223         if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl
224
225         coef_grad_pl(v-1,d,l) = ( + GMTR_t_pl(ijm1,k0,l,W3) * GMTR_a_pl(ijm1,k0,l,hn) &
226                                 + GMTR_t_pl(ijm1,k0,l,W3) * GMTR_a_pl(ij ,k0,l,hn) &
227                                 + GMTR_t_pl(ij ,k0,l,W2) * GMTR_a_pl(ij ,k0,l,hn) &
228                                 + GMTR_t_pl(ij ,k0,l,W2) * GMTR_a_pl(ijp1,k0,l,hn) &
229                                 ) * 0.5_RP * GMTR_p_pl(n,k0,l,P_RAREA)
230     enddo
231     enddo ! loop d
232     enddo ! loop l
233 endif
234
235     return
236 end subroutine OPRT_gradient_setup

```

Note the data layout of coefficient is slightly different between the normal region and the pole region, First dimension of `coef_grad_pl` corresponds to the second dimension of `coef_grad`. The range of index `v` of the inner-most loop are `ADM_gmin_pl` and `ADM_gmax_pl`, which means the five grid points surrounding the pole point.

(7) OPRT_laplacian_setup

This subroutine is also similar to the previous three subroutines, but this one is for the laplacian operator, subroutine `OPRT_laplacian`.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT_laplacian_setup( &
2      GMTR_p,  GMTR_p_pl,  &
3      GMTR_t,  GMTR_t_pl,  &
4      GMTR_a,  GMTR_a_pl,  &
5      coef_lap, coef_lap_pl )
6      !ESC! use mod_adm, only: &
7      !ESC!     ADM_have_pl,  &
8      !ESC!     ADM_have_sgp, &
9      !ESC!     ADM_gall_id,  &
10     !ESC!     ADM_gmin,    &
11     !ESC!     ADM_gmax,    &
12     !ESC!     ADM_gslf_pl, &
13     !ESC!     ADM_gmin_pl,  &
14     !ESC!     ADM_gmax_pl
15     !ESC! use mod_gmtr, only: &
16     !ESC!     P_RAREA => GMTR_p_RAREA, &
17     !ESC!     T_RAREA => GMTR_t_RAREA, &
18     !ESC!     HNX    => GMTR_a_HNX,  &
19     !ESC!     TNX    => GMTR_a_TNX,  &
20     !ESC!     TN2X   => GMTR_a_TN2X, &
21     !ESC!     GMTR_p_nmax, &
22     !ESC!     GMTR_t_nmax, &
23     !ESC!     GMTR_a_nmax,  &
24     !ESC!     GMTR_a_nmax_pl
25     implicit none
26
27     real(RP), intent(in) :: GMTR_p (ADM_gall ,KO,ADM_lall , GMTR_p_nmax )
28     real(RP), intent(in) :: GMTR_p_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_p_nmax )
29     real(RP), intent(in) :: GMTR_t (ADM_gall ,KO,ADM_lall ,TI:TJ,GMTR_t_nmax )
30     real(RP), intent(in) :: GMTR_t_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_t_nmax )
31     real(RP), intent(in) :: GMTR_a (ADM_gall ,KO,ADM_lall ,AI:AJ,GMTR_a_nmax )
32     real(RP), intent(in) :: GMTR_a_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_a_nmax_pl)
33     real(RP), intent(out) :: coef_lap (ADM_gall,0:6 ,ADM_lall )
34     real(RP), intent(out) :: coef_lap_pl( 0:ADM_vlink,ADM_lall_pl)
35

```

```

36 integer :: gmin, gmax, iall, gall, nxyz, lall
37
38 integer :: ij
39 integer :: ip1j, ijp1, ip1jp1
40 integer :: im1j, ijm1, im1jm1
41
42 integer :: g, l, d, n, v, hn, tn, tn2
43 !-----
44

```

Input arguments are the same with the previous three subroutines, the metrics calculated by the subroutines described before in this section. But the output argument `coef_lap` and `coef_lap_pl` have the different shape with the previous ones, such as `coef_div` and `coef_div_pl`. Since the laplacian operator is the second order differentiation, different from the divergence, rotation, gradient, these are the first order, and the result is a scalar, not a vector, there are no need to specify the 3-D direction, which are the third dimension of such as `coef_div`. Instead of that, calculation of the coefficient must include the contribution of more grid points than the operator of the first order differentiation.

The procedure of this subroutine is consist of three parts. The first part is as follows.

```

45 !if( IO_L ) write(IO_FID_LOG,*) '*** setup coefficient of laplacian operator'
46
47 gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
48 gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
49 iall = ADM_gall_1d
50 gall = ADM_gall
51 nxyz = ADM_nxyz
52 lall = ADM_lall
53
54 !$omp parallel workshare
55 coef_lap ( : , : , : ) = 0.0_RP
56 !$omp end parallel workshare
57 coef_lap_pl( : , : ) = 0.0_RP
58
59 !$omp parallel default(none),private(g,d,l,hn,tn,ij,ip1j,ip1jp1,ijp1,im1j,ijm1,im1jm1), &
60 !$omp shared(ADM_have_sgp,gmin,gmax,iall,gall,nxyz,lall,coef_lap,GMTR_p,GMTR_t,GMTR_a)
61 do l = 1, lall
62
63     do d = 1, nxyz
64         hn = d + HNX - 1
65         tn = d + TNX - 1
66
67         !$omp do
68         do g = gmin, gmax
69             ij = g
70             ip1j = g + 1
71             ip1jp1 = g + iall + 1
72             ijp1 = g + iall
73             im1j = g - 1
74             im1jm1 = g - iall - 1
75             ijm1 = g - iall
76
77             ! ij
78             coef_lap(ij,0,l) = coef_lap(ij,0,l) &
79                 + GMTR_t(ij ,k0,l,TI,T_RAREA) &
80                 * ( - 1.0_RP * GMTR_a(ij ,k0,l,AI ,tn) * GMTR_a(ij ,k0,l,AI ,hn) &
81                   + 2.0_RP * GMTR_a(ip1j ,k0,l,AJ ,tn) * GMTR_a(ij ,k0,l,AI ,hn) &
82                   + 1.0_RP * GMTR_a(ij ,k0,l,AIJ,tn) * GMTR_a(ij ,k0,l,AI ,hn) &
83                   - 1.0_RP * GMTR_a(ij ,k0,l,AI ,tn) * GMTR_a(ij ,k0,l,AIJ,hn) &
84                   + 2.0_RP * GMTR_a(ip1j ,k0,l,AJ ,tn) * GMTR_a(ij ,k0,l,AIJ,hn) &
85                   + 1.0_RP * GMTR_a(ij ,k0,l,AIJ,tn) * GMTR_a(ij ,k0,l,AIJ,hn) )
86
87             coef_lap(ij,0,l) = coef_lap(ij,0,l) &
88                 + GMTR_t(ij ,k0,l,TJ,T_RAREA) &
89                 * ( - 1.0_RP * GMTR_a(ij ,k0,l,AIJ,tn) * GMTR_a(ij ,k0,l,AIJ,hn) &
90                   - 2.0_RP * GMTR_a(ip1j ,k0,l,AI ,tn) * GMTR_a(ij ,k0,l,AIJ,hn) &
91                   + 1.0_RP * GMTR_a(ij ,k0,l,AJ ,tn) * GMTR_a(ij ,k0,l,AIJ,hn) &
92                   - 1.0_RP * GMTR_a(ij ,k0,l,AIJ,tn) * GMTR_a(ij ,k0,l,AJ ,hn) &
93                   - 2.0_RP * GMTR_a(ip1j ,k0,l,AI ,tn) * GMTR_a(ij ,k0,l,AJ ,hn) &
94                   + 1.0_RP * GMTR_a(ij ,k0,l,AJ ,tn) * GMTR_a(ij ,k0,l,AJ ,hn) )
95
96             coef_lap(ij,0,l) = coef_lap(ij,0,l) &
97                 + GMTR_t(im1j ,k0,l,II,T_RAREA) &
98                 * ( - 1.0_RP * GMTR_a(ij ,k0,l,AJ ,tn) * GMTR_a(ij ,k0,l,AJ ,hn) &
99                   - 2.0_RP * GMTR_a(im1j ,k0,l,AIJ,tn) * GMTR_a(ij ,k0,l,AJ ,hn) &
100                  - 1.0_RP * GMTR_a(im1j ,k0,l,AI ,tn) * GMTR_a(ij ,k0,l,AJ ,hn) &
101                  + 1.0_RP * GMTR_a(ij ,k0,l,AJ ,tn) * GMTR_a(im1j ,k0,l,AI ,hn) &

```



```

102         + 2.0_RP * GMTR_a(im1j ,k0,1,AIJ,tn) * GMTR_a(im1j ,k0,1,AI ,hn) &
103         + 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(im1j ,k0,1,AI ,hn) )
104
105     coef_lap(ij,0,1) = coef_lap(ij,0,1) &
106     + GMTR_t(im1jm1,k0,1,TJ,T_RAREA) &
107     * ( - 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(im1j ,k0,1,AI ,hn) &
108     + 2.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1j ,k0,1,AI ,hn) &
109     + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j ,k0,1,AI ,hn) &
110     - 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
111     + 2.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
112     + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) )
113
114     coef_lap(ij,0,1) = coef_lap(ij,0,1) &
115     + GMTR_t(im1jm1,k0,1,TI,T_RAREA) &
116     * ( - 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
117     - 2.0_RP * GMTR_a(im1jm1,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
118     + 1.0_RP * GMTR_a(im1jm1 ,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
119     - 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1 ,k0,1,AJ ,tn) &
120     - 2.0_RP * GMTR_a(im1jm1,k0,1,AI ,tn) * GMTR_a(im1jm1 ,k0,1,AJ ,tn) &
121     + 1.0_RP * GMTR_a(im1jm1 ,k0,1,AJ ,tn) * GMTR_a(im1jm1 ,k0,1,AJ ,tn) )
122
123     coef_lap(ij,0,1) = coef_lap(ij,0,1) &
124     + GMTR_t(ijm1 ,k0,1,TJ,T_RAREA) &
125     * ( - 1.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(ijm1 ,k0,1,AJ ,hn) &
126     - 2.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(ijm1 ,k0,1,AJ ,hn) &
127     - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ijm1 ,k0,1,AJ ,hn) &
128     + 1.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
129     + 2.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
130     + 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,hn) )
131
132     ! ip1j
133     coef_lap(ij,1,1) = coef_lap(ij,1,1) &
134     + GMTR_t(ijm1 ,k0,1,TJ,T_RAREA) &
135     * ( - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ijm1 ,k0,1,AJ ,hn) &
136     + 2.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(ijm1 ,k0,1,AJ ,hn) &
137     + 1.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(ijm1 ,k0,1,AJ ,hn) &
138     + 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
139     - 2.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
140     - 1.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,hn) )
141
142     coef_lap(ij,1,1) = coef_lap(ij,1,1) &
143     + GMTR_t(ij ,k0,1,TI,T_RAREA) &
144     * ( - 1.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
145     - 2.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
146     - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
147     - 1.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
148     - 2.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
149     - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) )
150
151     ! ip1jp1
152     coef_lap(ij,2,1) = coef_lap(ij,2,1) &
153     + GMTR_t(ij ,k0,1,TI,T_RAREA) &
154     * ( + 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
155     + 2.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
156     - 1.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
157     + 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
158     + 2.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
159     - 1.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) )
160
161     coef_lap(ij,2,1) = coef_lap(ij,2,1) &
162     + GMTR_t(ij ,k0,1,TJ,T_RAREA) &
163     * ( + 1.0_RP * GMTR_a(ijp1 ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
164     - 2.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
165     - 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
166     + 1.0_RP * GMTR_a(ijp1 ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
167     - 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
168     - 2.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) )
169
170     ! ijp1
171     coef_lap(ij,3,1) = coef_lap(ij,3,1) &
172     + GMTR_t(ij ,k0,1,TJ,T_RAREA) &
173     * ( + 1.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
174     + 2.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
175     + 1.0_RP * GMTR_a(ijp1 ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
176     + 1.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
177     + 2.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
178     + 1.0_RP * GMTR_a(ijp1 ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) )
179
180     coef_lap(ij,3,1) = coef_lap(ij,3,1) &
181     + GMTR_t(im1j ,k0,1,TI,T_RAREA) &
182     * ( + 1.0_RP * GMTR_a(im1j ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AJ ,hn) &
183     + 2.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AJ ,hn) &

```

```

184         - 1.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
185         - 1.0_RP * GMTR_a(im1j   ,k0,1,AIJ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
186         - 2.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
187         + 1.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) )
188
189     ! im1j
190     coef_lap(ij,4,1) = coef_lap(ij,4,1) &
191     + GMTR_t(im1j   ,k0,1,TT,T_RAREA) &
192     * ( - 1.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
193     + 2.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
194     + 1.0_RP * GMTR_a(im1j   ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
195     + 1.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
196     - 2.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
197     - 1.0_RP * GMTR_a(im1j   ,k0,1,AIJ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) )
198
199     coef_lap(ij,4,1) = coef_lap(ij,4,1) &
200     + GMTR_t(im1jm1,k0,1,TJ,T_RAREA) &
201     * ( - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
202     - 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
203     - 1.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
204     - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
205     - 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
206     - 1.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) )
207
208     ! im1jm1
209     coef_lap(ij,5,1) = coef_lap(ij,5,1) &
210     + GMTR_t(im1jm1,k0,1,TJ,T_RAREA) &
211     * ( + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
212     + 2.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
213     - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1j   ,k0,1,AI ,hn) &
214     + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
215     + 2.0_RP * GMTR_a(im1j   ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
216     - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) )
217
218     coef_lap(ij,5,1) = coef_lap(ij,5,1) &
219     + GMTR_t(im1jm1,k0,1,TT,T_RAREA) &
220     * ( + 1.0_RP * GMTR_a(im1jm1,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
221     - 2.0_RP * GMTR_a(im1j   ,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
222     - 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
223     + 1.0_RP * GMTR_a(im1jm1,k0,1,AI ,tn) * GMTR_a(im1j   ,k0,1,AJ ,hn) &
224     - 2.0_RP * GMTR_a(im1j   ,k0,1,AJ ,tn) * GMTR_a(im1j   ,k0,1,AJ ,hn) &
225     - 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j   ,k0,1,AJ ,hn) )
226
227     ! ijm1
228     coef_lap(ij,6,1) = coef_lap(ij,6,1) &
229     + GMTR_t(im1jm1,k0,1,TT,T_RAREA) &
230     * ( + 1.0_RP * GMTR_a(ijm1   ,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
231     + 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
232     + 1.0_RP * GMTR_a(im1jm1,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
233     + 1.0_RP * GMTR_a(ijm1   ,k0,1,AJ ,tn) * GMTR_a(ijm1   ,k0,1,AJ ,hn) &
234     + 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(ijm1   ,k0,1,AJ ,hn) &
235     + 1.0_RP * GMTR_a(im1jm1,k0,1,AI ,tn) * GMTR_a(ijm1   ,k0,1,AJ ,hn) )
236
237     coef_lap(ij,6,1) = coef_lap(ij,6,1) &
238     + GMTR_t(ijm1   ,k0,1,TJ,T_RAREA) &
239     * ( + 1.0_RP * GMTR_a(ijm1   ,k0,1,AIJ,tn) * GMTR_a(ijm1   ,k0,1,AJ ,hn) &
240     + 2.0_RP * GMTR_a(ij      ,k0,1,AI ,tn) * GMTR_a(ijm1   ,k0,1,AJ ,hn) &
241     - 1.0_RP * GMTR_a(ijm1   ,k0,1,AJ ,tn) * GMTR_a(ijm1   ,k0,1,AJ ,hn) &
242     - 1.0_RP * GMTR_a(ijm1   ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
243     - 2.0_RP * GMTR_a(ij      ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
244     + 1.0_RP * GMTR_a(ijm1   ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) )
245
246     enddo
247     !$omp end do
248     enddo ! loop d

```

The first part above and the second part below are in a long l -loop. In the g -loop represents horizontal index, 7 coefficients are calculated separately from the various metrics. Note that each 7 coefficients contains contribution from more grid points than previous operator subroutines.

The second part is as follows.

```

249     if ( ADM_have_sgp(1) ) then ! pentagon
250         !$omp master
251         coef_lap(gmin,0,1) = 0.0_RP
252         coef_lap(gmin,1,1) = 0.0_RP
253         coef_lap(gmin,2,1) = 0.0_RP
254         coef_lap(gmin,3,1) = 0.0_RP
255         coef_lap(gmin,4,1) = 0.0_RP

```

```

256 coef_lap(gmin,5,1) = 0.0_RP
257 coef_lap(gmin,6,1) = 0.0_RP
258
259 do d = 1, ADM_nxyz
260   hn = d + HNX - 1
261   tn = d + TNX - 1
262
263   ij = gmin
264   ip1j = gmin + 1
265   ip1jp1 = gmin + iall + 1
266   ijp1 = gmin + iall
267   im1j = gmin - 1
268   im1jm1 = gmin - iall - 1
269   ijm1 = gmin - iall
270
271   ! ij
272   coef_lap(ij,0,1) = coef_lap(ij,0,1) &
273     + GMTR_t(ij ,k0,1,TI,T_RAREA) &
274     * ( - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
275       + 2.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
276       + 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,hn) &
277       - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
278       + 2.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
279       + 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) )
280
281   coef_lap(ij,0,1) = coef_lap(ij,0,1) &
282     + GMTR_t(ij ,k0,1,TJ,T_RAREA) &
283     * ( - 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
284       - 2.0_RP * GMTR_a(ip1j ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
285       + 1.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
286       - 1.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
287       - 2.0_RP * GMTR_a(ip1j ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
288       + 1.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) )
289
290   coef_lap(ij,0,1) = coef_lap(ij,0,1) &
291     + GMTR_t(im1j ,k0,1,TI,T_RAREA) &
292     * ( - 1.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
293       - 2.0_RP * GMTR_a(im1j ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
294       - 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AJ ,tn) &
295       + 1.0_RP * GMTR_a(ij ,k0,1,AJ ,tn) * GMTR_a(im1j ,k0,1,AI ,tn) &
296       + 2.0_RP * GMTR_a(im1j ,k0,1,AIJ,tn) * GMTR_a(im1j ,k0,1,AI ,tn) &
297       + 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(im1j ,k0,1,AI ,tn) )
298
299   coef_lap(ij,0,1) = coef_lap(ij,0,1) &
300     + GMTR_t(im1jm1,k0,1,TJ,T_RAREA) &
301     * ( - 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(im1j ,k0,1,AI ,tn) &
302       + 2.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1j ,k0,1,AI ,tn) &
303       + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j ,k0,1,AI ,tn) &
304       - 1.0_RP * GMTR_a(im1j ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
305       + 2.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
306       + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) )
307
308   coef_lap(ij,0,1) = coef_lap(ij,0,1) &
309     + GMTR_t(ijm1 ,k0,1,TJ,T_RAREA) &
310     * ( - 1.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
311       - 2.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
312       - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
313       + 1.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
314       + 2.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
315       + 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,tn) )
316
317   ! ip1j
318   coef_lap(ij,1,1) = coef_lap(ij,1,1) &
319     + GMTR_t(ijm1 ,k0,1,TJ,T_RAREA) &
320     * ( + 1.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
321       + 2.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
322       - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,tn) &
323       - 1.0_RP * GMTR_a(ijm1 ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
324       - 2.0_RP * GMTR_a(ijm1 ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
325       + 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,tn) )
326
327   coef_lap(ij,1,1) = coef_lap(ij,1,1) &
328     + GMTR_t(ij ,k0,1,TI,T_RAREA) &
329     * ( - 1.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
330       - 2.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
331       - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AI ,tn) &
332       - 1.0_RP * GMTR_a(ip1j ,k0,1,AJ ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
333       - 2.0_RP * GMTR_a(ij ,k0,1,AIJ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) &
334       - 1.0_RP * GMTR_a(ij ,k0,1,AI ,tn) * GMTR_a(ij ,k0,1,AIJ,tn) )
335
336   ! ip1jp1
337   coef_lap(ij,2,1) = coef_lap(ij,2,1) &

```

```

338     + GMTR_t(ij      ,k0,1,TT,T_RAREA) &
339     * ( + 1.0_RP * GMTR_a(ij      ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
340         + 2.0_RP * GMTR_a(ij      ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
341         - 1.0_RP * GMTR_a(ip1j    ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
342         + 1.0_RP * GMTR_a(ij      ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
343         + 2.0_RP * GMTR_a(ij      ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
344         - 1.0_RP * GMTR_a(ip1j    ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) )
345
346     coef_lap(ij,2,1) = coef_lap(ij,2,1) &
347     + GMTR_t(ij      ,k0,1,TJ,T_RAREA) &
348     * ( + 1.0_RP * GMTR_a(ij1p1   ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
349         - 2.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
350         - 1.0_RP * GMTR_a(ij      ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
351         + 1.0_RP * GMTR_a(ij1p1   ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
352         - 2.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
353         - 1.0_RP * GMTR_a(ij      ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) )
354
355     ! ijp1
356     coef_lap(ij,3,1) = coef_lap(ij,3,1) &
357     + GMTR_t(ij      ,k0,1,TJ,T_RAREA) &
358     * ( + 1.0_RP * GMTR_a(ij1p1   ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
359         + 2.0_RP * GMTR_a(ij      ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
360         + 1.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AIJ,hn) &
361         + 1.0_RP * GMTR_a(ij1p1   ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
362         + 2.0_RP * GMTR_a(ij      ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
363         + 1.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) )
364
365     coef_lap(ij,3,1) = coef_lap(ij,3,1) &
366     + GMTR_t(im1j     ,k0,1,TT,T_RAREA) &
367     * ( + 1.0_RP * GMTR_a(im1j     ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
368         + 2.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
369         - 1.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
370         - 1.0_RP * GMTR_a(im1j     ,k0,1,AIJ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
371         - 2.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
372         + 1.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) )
373
374     ! im1j
375     coef_lap(ij,4,1) = coef_lap(ij,4,1) &
376     + GMTR_t(im1j     ,k0,1,TT,T_RAREA) &
377     * ( + 1.0_RP * GMTR_a(im1j     ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
378         + 2.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
379         - 1.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AJ ,hn) &
380         - 1.0_RP * GMTR_a(im1j     ,k0,1,AIJ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
381         - 2.0_RP * GMTR_a(ij      ,k0,1,AJ ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
382         + 1.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) )
383
384     coef_lap(ij,4,1) = coef_lap(ij,4,1) &
385     + GMTR_t(im1jm1,k0,1,TJ,T_RAREA) &
386     * ( - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
387         - 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
388         - 1.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
389         - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
390         - 2.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
391         - 1.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) )
392
393     ! im1jm1
394     coef_lap(ij,5,1) = coef_lap(ij,5,1) &
395     + GMTR_t(im1jm1,k0,1,TJ,T_RAREA) &
396     * ( - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
397         + 2.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
398         + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1j     ,k0,1,AI ,hn) &
399         - 1.0_RP * GMTR_a(im1jm1,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
400         + 2.0_RP * GMTR_a(im1j     ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
401         + 1.0_RP * GMTR_a(im1jm1,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) )
402
403     ! ijm1
404     coef_lap(ij,6,1) = coef_lap(ij,6,1) &
405     + GMTR_t(ijm1     ,k0,1,TJ,T_RAREA) &
406     * ( + 1.0_RP * GMTR_a(ijm1     ,k0,1,AIJ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
407         + 2.0_RP * GMTR_a(ij      ,k0,1,AI ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
408         - 1.0_RP * GMTR_a(ijm1     ,k0,1,AJ ,tn) * GMTR_a(im1jm1,k0,1,AIJ,hn) &
409         - 1.0_RP * GMTR_a(ijm1     ,k0,1,AIJ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
410         - 2.0_RP * GMTR_a(ij      ,k0,1,AI ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) &
411         + 1.0_RP * GMTR_a(ijm1     ,k0,1,AJ ,tn) * GMTR_a(ij      ,k0,1,AI ,hn) )
412
413     enddo ! loop d
414     !$omp end master
415
416     !$omp do
417     do g = 1, gall
418     coef_lap(g,0,1) = coef_lap(g,0,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP
419     coef_lap(g,1,1) = coef_lap(g,1,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP

```

```

420     coef_lap(g,2,1) = coef_lap(g,2,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP
421     coef_lap(g,3,1) = coef_lap(g,3,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP
422     coef_lap(g,4,1) = coef_lap(g,4,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP
423     coef_lap(g,5,1) = coef_lap(g,5,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP
424     coef_lap(g,6,1) = coef_lap(g,6,1) * GMTR_p(g,k0,1,P_RAREA) / 12.0_RP
425     enddo
426     !$omp end do
427
428     enddo ! loop l
429     !$omp end parallel
430

```

This section is for the singular point. Note that ADM_have_sgp(1) is true.

The last section of this subroutine is for the pole region and is as follows.

```

431     if ( ADM_have_pl ) then
432         n = ADM_gslf_pl
433
434         do l = 1,ADM_lall_pl
435
436             do d = 1, ADM_nxyz
437                 hn = d + HNX - 1
438                 tn = d + TNX - 1
439                 tn2 = d + TN2X - 1
440
441                 do v = ADM_gmin_pl, ADM_gmax_pl
442                     ij = v
443                     ijp1 = v + 1
444                     ijm1 = v - 1
445                     if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
446                     if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl
447
448                     coef_lap_pl(0,1) = coef_lap_pl(0,1) &
449                         + GMTR_t_pl(ijm1,k0,1,T_RAREA) &
450                         * ( + 1.0_RP * GMTR_a_pl(ijm1,k0,1,tn) * GMTR_a_pl(ij,k0,1,hn) &
451                             - 2.0_RP * GMTR_a_pl(ijm1,k0,1,tn2) * GMTR_a_pl(ij,k0,1,hn) &
452                             - 1.0_RP * GMTR_a_pl(ij ,k0,1,tn) * GMTR_a_pl(ij,k0,1,hn) )
453
454                     coef_lap_pl(0,1) = coef_lap_pl(0,1) &
455                         + GMTR_t_pl(ij ,k0,1,T_RAREA) &
456                         * ( + 1.0_RP * GMTR_a_pl(ij ,k0,1,tn) * GMTR_a_pl(ij,k0,1,hn) &
457                             - 2.0_RP * GMTR_a_pl(ij ,k0,1,tn2) * GMTR_a_pl(ij,k0,1,hn) &
458                             - 1.0_RP * GMTR_a_pl(ijp1,k0,1,tn) * GMTR_a_pl(ij,k0,1,hn) )
459
460                     enddo
461
462                 do v = ADM_gmin_pl, ADM_gmax_pl
463                     ij = v
464                     ijp1 = v + 1
465                     ijm1 = v - 1
466                     if( ijp1 == ADM_gmax_pl + 1 ) ijp1 = ADM_gmin_pl
467                     if( ijm1 == ADM_gmin_pl - 1 ) ijm1 = ADM_gmax_pl
468
469                     coef_lap_pl(v-1,1) = coef_lap_pl(v-1,1) &
470                         + GMTR_t_pl(ijm1,k0,1,T_RAREA) &
471                         * ( - 2.0_RP * GMTR_a_pl(ijm1,k0,1,tn) * GMTR_a_pl(ijm1,k0,1,hn) &
472                             + 1.0_RP * GMTR_a_pl(ijm1,k0,1,tn2) * GMTR_a_pl(ijm1,k0,1,hn) &
473                             - 1.0_RP * GMTR_a_pl(ij ,k0,1,tn) * GMTR_a_pl(ijm1,k0,1,hn) &
474                             - 2.0_RP * GMTR_a_pl(ijm1,k0,1,tn) * GMTR_a_pl(ij ,k0,1,hn) &
475                             + 1.0_RP * GMTR_a_pl(ijm1,k0,1,tn2) * GMTR_a_pl(ij ,k0,1,hn) &
476                             - 1.0_RP * GMTR_a_pl(ij ,k0,1,tn) * GMTR_a_pl(ij ,k0,1,hn) )
477
478                     coef_lap_pl(v-1,1) = coef_lap_pl(v-1,1) &
479                         + GMTR_t_pl(ij ,k0,1,T_RAREA) &
480                         * ( + 1.0_RP * GMTR_a_pl(ij ,k0,1,tn) * GMTR_a_pl(ij ,k0,1,hn) &
481                             + 1.0_RP * GMTR_a_pl(ij ,k0,1,tn2) * GMTR_a_pl(ij ,k0,1,hn) &
482                             + 2.0_RP * GMTR_a_pl(ijp1,k0,1,tn) * GMTR_a_pl(ij ,k0,1,hn) &
483                             + 1.0_RP * GMTR_a_pl(ij ,k0,1,tn) * GMTR_a_pl(ijp1,k0,1,hn) &
484                             + 1.0_RP * GMTR_a_pl(ij ,k0,1,tn2) * GMTR_a_pl(ijp1,k0,1,hn) &
485                             + 2.0_RP * GMTR_a_pl(ijp1,k0,1,tn) * GMTR_a_pl(ijp1,k0,1,hn) )
486
487                     enddo
488                 enddo ! d loop
489
490                 do v = ADM_gslf_pl, ADM_gmax_pl
491                     coef_lap_pl(v-1,1) = coef_lap_pl(v-1,1) * GMTR_p_pl(n,k0,1,P_RAREA) / 12.0_RP
492                 enddo
493
494                 enddo ! l loop
495             endif

```

```

495     return
496 end subroutine OPRT_laplacian_setup

```

Note the data layout of coefficient is slightly different between the normal region and the pole region. First dimension of `coef_lap_pl` corresponds to the second dimension of `coef_lap`. The range of index `v` of the inner-most loop are `ADM_gmin_pl` and `ADM_gmax_pl`, which means the five grid points surrounding the pole point.

(8) OPRT_diffusion_setup

This subroutine is to calculate the coefficients for the diffusion operator, subroutine `OPRT_diffusion`, which is used in the `dyn_diffusion` kernel program.

Argument lists and local variables definition part of this subroutine is as follows.

```

1  subroutine OPRT_diffusion_setup( &
2     GMTR_p,   GMTR_p_pl,   &
3     GMTR_t,   GMTR_t_pl,   &
4     GMTR_a,   GMTR_a_pl,   &
5     coef_intp, coef_intp_pl, &
6     coef_diff, coef_diff_pl )
7     !ESC! use mod_adm, only: &
8     !ESC!     ADM_have_pl, &
9     !ESC!     ADM_have_sgp, &
10    !ESC!     ADM_gall_id, &
11    !ESC!     ADM_gmin, &
12    !ESC!     ADM_gmax, &
13    !ESC!     ADM_gslf_pl, &
14    !ESC!     ADM_gmin_pl, &
15    !ESC!     ADM_gmax_pl
16    !ESC! use mod_gmtr, only: &
17    !ESC!     P_RAREA => GMTR_p_RAREA, &
18    !ESC!     T_RAREA => GMTR_t_RAREA, &
19    !ESC!     HNX     => GMTR_a_HNX, &
20    !ESC!     TNX     => GMTR_a_TNX, &
21    !ESC!     TN2X    => GMTR_a_TN2X, &
22    !ESC!     GMTR_p_nmax, &
23    !ESC!     GMTR_t_nmax, &
24    !ESC!     GMTR_a_nmax, &
25    !ESC!     GMTR_a_nmax_pl
26    implicit none
27
28    real(RP), intent(in) :: GMTR_p   (ADM_gall ,KO,ADM_lall , GMTR_p_nmax )
29    real(RP), intent(in) :: GMTR_p_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_p_nmax )
30    real(RP), intent(in) :: GMTR_t   (ADM_gall ,KO,ADM_lall ,TI:TJ,GMTR_t_nmax )
31    real(RP), intent(in) :: GMTR_t_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_t_nmax )
32    real(RP), intent(in) :: GMTR_a   (ADM_gall ,KO,ADM_lall ,AI:AJ,GMTR_a_nmax )
33    real(RP), intent(in) :: GMTR_a_pl (ADM_gall_pl,KO,ADM_lall_pl, GMTR_a_nmax_pl)
34    real(RP), intent(out) :: coef_intp (ADM_gall ,1:3,ADM_nxyz,TI:TJ,ADM_lall )
35    real(RP), intent(out) :: coef_intp_pl(ADM_gall_pl,1:3,ADM_nxyz, ADM_lall_pl)
36    real(RP), intent(out) :: coef_diff (ADM_gall,1:6 ,ADM_nxyz,ADM_lall )
37    real(RP), intent(out) :: coef_diff_pl( 1:ADM_vlink,ADM_nxyz,ADM_lall_pl)
38
39    integer :: gmin, gmax, iall, gall, nxyz, lall, gminm1
40
41    integer :: ij
42    integer :: ip1j, ip1j
43    integer :: im1j, im1j, im1jm1
44
45    integer :: g, l, d, n, v, hn, tn, tn2
46    !-----
47

```

Input arguments are the same with the previous subroutines, the metrics calculated by the subroutines described before in this section. But the different from them, output arguments this subroutine calculates are two kinds, `coef_intp`, `coef_diff`, and those for the pole region. `coef_intp` is used to calculate the interpolated value at the vertex points of the control cell from the center point of the cell, and `coef_diff` is used to calculate flux convergence by using the value at the mid-point of the edge of the cell. These coefficients are used not only for the diffusion operator, but also the operator of horizontal divergence damping. See the source code of subroutine `OPRT_diffusion` in [section 2.2](#).

Main part of this subroutine is consist of three section. The first section is as follows.

```

48      !if( IO_L ) write(IO_FTD_LOG,*) '*** setup coefficient of diffusion operator'
49
50      gmin = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin
51      gmax = (ADM_gmax-1)*ADM_gall_1d + ADM_gmax
52      iall = ADM_gall_1d
53      gall = ADM_gall
54      nxyz = ADM_nxyz
55      lall = ADM_lall
56
57      !$omp parallel workshare
58      coef_intp (:,:,,:) = 0.0_RP
59      coef_diff (:,:,) = 0.0_RP
60      !$omp end parallel workshare
61      coef_intp_pl(:,:,) = 0.0_RP
62      coef_diff_pl( :,:,) = 0.0_RP
63
64      gminm1 = (ADM_gmin-1)*ADM_gall_1d + ADM_gmin-1
65
66      !$omp parallel do default(none),private(g,d,l,tn,ij,ip1,ijp1), &
67      !$omp shared(gminm1,gmax,iall,gall,nxyz,lall,coef_intp,GMTR_t,GMTR_a), &
68      !$omp collapse(2)
69      do l = 1, lall
70      do d = 1, nxyz
71          tn = d + TNX - 1
72
73          do g = gminm1, gmax
74              ij = g
75              ip1j = g + 1
76              ijp1 = g + iall
77
78              coef_intp(ij,1,d,TI,1) = ( + GMTR_a(ij ,k0,1,AIJ,tn) - GMTR_a(ij ,k0,1,AI ,tn) ) &
79              * 0.5_RP * GMTR_t(ij,k0,1,TI,T_RAREA)
80              coef_intp(ij,2,d,TI,1) = ( - GMTR_a(ij ,k0,1,AI ,tn) - GMTR_a(ip1j,k0,1,AJ ,tn) ) &
81              * 0.5_RP * GMTR_t(ij,k0,1,TI,T_RAREA)
82              coef_intp(ij,3,d,TI,1) = ( - GMTR_a(ip1j,k0,1,AJ ,tn) + GMTR_a(ij ,k0,1,AIJ,tn) ) &
83              * 0.5_RP * GMTR_t(ij,k0,1,TI,T_RAREA)
84
85              coef_intp(ij,1,d,TJ,1) = ( + GMTR_a(ij ,k0,1,AJ ,tn) - GMTR_a(ij ,k0,1,AIJ,tn) ) &
86              * 0.5_RP * GMTR_t(ij,k0,1,TJ,T_RAREA)
87              coef_intp(ij,2,d,TJ,1) = ( - GMTR_a(ij ,k0,1,AIJ,tn) + GMTR_a(ijp1,k0,1,AI ,tn) ) &
88              * 0.5_RP * GMTR_t(ij,k0,1,TJ,T_RAREA)
89              coef_intp(ij,3,d,TJ,1) = ( + GMTR_a(ijp1,k0,1,AI ,tn) + GMTR_a(ij ,k0,1,AJ ,tn) ) &
90              * 0.5_RP * GMTR_t(ij,k0,1,TJ,T_RAREA)
91          enddo
92      enddo ! loop d
93      enddo ! loop l
94      !$omp end parallel do
95

```

In this section coef_intp is calculated.

The second section is as follows.

```

96      !$omp parallel default(none),private(g,d,l,hn,ij,im1j,ijm1,im1jm1), &
97      !$omp shared(ADM_have_sgp,gmin,gmax,iall,gall,nxyz,lall,coef_diff,GMTR_p,GMTR_a)
98      do l = 1, lall
99      do d = 1, nxyz
100          hn = d + HNX - 1
101
102          !$omp do
103          do g = gmin, gmax
104              ij = g
105              im1j = g - 1
106              im1jm1 = g - iall - 1
107              ijm1 = g - iall
108
109              coef_diff(ij,1,d,1) = + GMTR_a(ij ,k0,1,AIJ,hn) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
110              coef_diff(ij,2,d,1) = + GMTR_a(ij ,k0,1,AJ ,hn) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
111              coef_diff(ij,3,d,1) = - GMTR_a(im1j ,k0,1,AI ,hn) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
112              coef_diff(ij,4,d,1) = - GMTR_a(im1jm1,k0,1,AIJ,hn) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
113              coef_diff(ij,5,d,1) = - GMTR_a(ijm1 ,k0,1,AJ ,hn) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
114              coef_diff(ij,6,d,1) = + GMTR_a(ij ,k0,1,AI ,hn) * 0.5_RP * GMTR_p(ij,k0,1,P_RAREA)
115          enddo
116          !$omp end do
117
118          if ( ADM_have_sgp(1) ) then ! pentagon
119              !$omp master
120              coef_diff(gmin,5,d,1) = 0.0_RP
121              !$omp end master

```

```

122     endif
123   enddo ! loop d
124   enddo ! loop l
125   !$omp end parallel
126

```

In this section coef_diff is calculated.

The last section is as follows.

```

127   if ( ADM_have_pl ) then
128     n = ADM_gsif_pl
129
130     do l = 1, ADM_lall_pl
131
132       do d = 1, ADM_nxyz
133         hn = d + HNX - 1
134         tn = d + TNX - 1
135         tn2 = d + TN2X - 1
136
137         do v = ADM_gmin_pl, ADM_gmax_pl
138           ij = v
139           ijp1 = v + 1
140           if( ijp1 == ADM_gmax_pl+1 ) ijp1 = ADM_gmin_pl
141
142           coef_intp_pl(v,1,d,l) = - GMTR_a_pl(ijp1,k0,l,tn ) + GMTR_a_pl(ij ,k0,l,tn )
143           coef_intp_pl(v,2,d,l) = + GMTR_a_pl(ij ,k0,l,tn ) + GMTR_a_pl(ij ,k0,l,tn2)
144           coef_intp_pl(v,3,d,l) = + GMTR_a_pl(ij ,k0,l,tn2) - GMTR_a_pl(ijp1,k0,l,tn )
145
146           coef_intp_pl(v,:,d,l) = coef_intp_pl(v,:,d,l) * 0.5_RP * GMTR_t_pl(v,k0,l,T_RAREA)
147
148           coef_diff_pl(v-1,d,l) = GMTR_a_pl(v,k0,l,hn) * 0.5_RP * GMTR_p_pl(n,k0,l,P_RAREA)
149         enddo
150       enddo
151
152     enddo ! l loop
153   endif
154
155   return
156 end subroutine OPRT_diffusion_setup

```

This section is for the pole region.

2.8.3 Input data and result

Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of \$IAB_SYS=Ubuntu-gnu-mpi case.

```

### Input ###
+check[GRD_x ] max= 6.371220000000000E+06,min= -2.8628180616668505E+06,sum= 1.6166957233794562E+11
+check[GRD_x_pl ] max= 6.371220000000000E+06,min= -6.371220000000000E+06,sum= -3.6880373954772949E-07
+check[GRD_xt ] max= 4.059244428840000E+13,min= -2.8414384541049926E+06,sum= 6.3080977453379920E+16
+check[GRD_xt_pl ] max= 6.371220000000000E+06,min= -6.371220000000000E+06,sum= -2.6449561119079590E-07
+check[GRD_s ] max= 1.5707963267948966E+00,min= -1.25666370614358098E+00,sum= 1.9488544754099312E+04
+check[GRD_s_pl ] max= 3.1415926535897931E+00,min= -2.5132741228746029E+00,sum= 3.1415926535791883E+00
+check[check_GMTR_p ] max= 3.1937623126102595E+09,min= -1.000000000000000E+00,sum= 5.2568771941077820E+13
+check[check_GMTR_p_pl ] max= 2.6945919723204341E+09,min= -1.000000000000000E+00,sum= 3.0761871438935734E+10
+check[check_GMTR_t ] max= 1.5988904134394393E+09,min= 0.000000000000000E+00,sum= 5.1787428409617836E+10
+check[check_GMTR_t_pl ] max= 1.9722549492980933E+09,min= 0.000000000000000E+00,sum= 1.9722549502848648E+10
+check[check_GMTR_a ] max= 6.3362880879999531E+04,min= -6.0857794015098916E+04,sum= 7.9258961023823655E+08
+check[check_GMTR_a_pl ] max= 5.7873883883004550E+04,min= -5.7872753043726123E+04,sum= -4.3064210331067443E-07
+check[check_OPRT_coef_] max= 7.7272073615122900E-06,min= -7.7272073610225610E-06,sum= -3.8056606893040025E-03
+check[check_OPRT_coef_] max= 6.2151734035229484E-06,min= -6.2403951822043815E-06,sum= 4.0763460586613204E-21
+check[check_OPRT_coef_] max= 8.1249192032094562E-06,min= -7.0395716054136067E-06,sum= 2.3097297195178346E-03
+check[check_OPRT_coef_] max= 6.2404355543912712E-06,min= -6.2404355543546362E-06,sum= 2.0117032497291439E-21
+check[check_OPRT_coef_] max= 7.7272073615122900E-06,min= -1.2825075745577413E-05,sum= 1.8320196095290668E-15
+check[check_OPRT_coef_] max= 6.2151734035229484E-06,min= -6.2403951822043815E-06,sum= -8.3793581340078915E-17
+check[check_OPRT_coef_] max= 3.5452458387465835E-10,min= -1.7726229192139183E-09,sum= -2.7069899136104107E-20
+check[check_OPRT_coef_] max= 6.4022976413393270E-11,min= -3.2011488205582575E-10,sum= -1.0339757656912846E-25
+check[check_OPRT_coef_] max= 3.6115131385825805E-05,min= -3.3754836603619669E-05,sum= -1.5562860020283464E-02
+check[check_OPRT_coef_] max= 1.4672104347081161E-05,min= -1.4672104347198630E-05,sum= 6.2882931910771206E-19
+check[check_OPRT_coef_] max= 8.2996786251499792E-06,min= -8.2996786252745083E-06,sum= -1.9028303446517866E-03
+check[check_OPRT_coef_] max= 8.7267985478094788E-06,min= -8.7267985478605143E-06,sum= 5.4196873734474373E-21
### Output ###
+check[GMTR_p ] max= 3.1937623126102595E+09,min= -1.000000000000000E+00,sum= 5.2568771941077820E+13

```



```

+check[GMTR_p_pl ] max= 2.6945919723204341E+09,min= -1.000000000000000E+00,sum= 3.0761871438935734E+10
+check[GMTR_t ] max= 1.5988904134394393E+09,min= 0.000000000000000E+00,sum= 5.1787428409617836E+13
+check[GMTR_t_pl ] max= 1.9722549492980933E+09,min= 0.000000000000000E+00,sum= 1.9722549502848648E+10
+check[GMTR_a ] max= 6.3362880879999531E+04,min= -6.0857794015098916E+04,sum= 7.9258961023823655E+08
+check[GMTR_a_pl ] max= 5.7873883883004550E+04,min= -5.7872753043726123E+04,sum= -4.3064210331067443E-07
+check[OPRT_coef_div ] max= 7.7272073615122900E-06,min= -7.7272073610225610E-06,sum= -3.8056606893040025E-03
+check[OPRT_coef_div_pl] max= 6.2151734035229484E-06,min= -6.2403951822043815E-06,sum= 4.0763460586613204E-21
+check[OPRT_coef_rot ] max= 8.1249192032094562E-06,min= -7.0395716054136067E-06,sum= 2.3097297195178346E-03
+check[OPRT_coef_rot_pl] max= 6.2404355543912712E-06,min= -6.2404355543546362E-06,sum= 2.0117032497291439E-21
+check[OPRT_coef_grad ] max= 7.7272073615122900E-06,min= -1.2825075745577413E-05,sum= 1.8320196095290668E-15
+check[OPRT_coef_grad_pl] max= 6.2151734035229484E-06,min= -6.2403951822043815E-06,sum= -8.3793581340078915E-17
+check[OPRT_coef_lap ] max= 3.5452458387465835E-10,min= -1.7726229192139183E-09,sum= -2.7069899136104107E-20
+check[OPRT_coef_lap_pl] max= 6.4022976413393270E-11,min= -3.2011488205582575E-10,sum= -1.0339757656912846E-25
+check[OPRT_coef_intp ] max= 3.6115131385825805E-05,min= -3.3754836603619669E-05,sum= -1.5562860020283464E-02
+check[OPRT_coef_intp_pl] max= 1.4672104347081161E-05,min= -1.4672104347198630E-05,sum= 6.2882931910771206E-19
+check[OPRT_coef_diff ] max= 8.2996786251499792E-06,min= -8.2996786252745083E-06,sum= -1.9028303446517686E-03
+check[OPRT_coef_diff_pl] max= 8.7267985478094788E-06,min= -8.7267985478605143E-06,sum= 5.4196873734474373E-21
### Validation : point-by-point diff ###
+check[check_GMTR_p ] max= 3.3306690738754696E-16,min= -3.3306690738754696E-16,sum= 8.0781261129099502E-15
+check[check_GMTR_p_pl ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_GMTR_t ] max= 5.5511151231257827E-17,min= -2.3841857910156250E-07,sum= -2.3841857899054020E-07
+check[check_GMTR_t_pl ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_GMTR_a ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_GMTR_a_pl ] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_] max= 8.4703294725430034E-22,min= -5.7240898388669515E-22,sum= 2.2830184906463564E-21
+check[check_OPRT_coef_pl] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_] max= 5.7271917661640254E-22,min= -8.4703294725430034E-22,sum= -9.7783088161424784E-22
+check[check_OPRT_coef_pl] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_] max= 8.4703294725430034E-22,min= -2.2896359355467806E-21,sum= 4.2351647362715017E-22
+check[check_OPRT_coef_pl] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_] max= 5.1698788284564230E-26,min= 0.000000000000000E+00,sum= 7.7548182426846345E-26
+check[check_OPRT_coef_pl] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_] max= 1.6940658945086007E-21,min= -3.3881317890172014E-21,sum= -2.9646153153900512E-21
+check[check_OPRT_coef_pl] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
+check[check_OPRT_coef_pl] max= 0.000000000000000E+00,min= 0.000000000000000E+00,sum= 0.000000000000000E+00
*** Finish kernel

```

Check the lines below `\src{''Validation : point-by-point diff''}` line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable.

There are sample output log files in `\file{reference/}` in each kernel program directory, for reference purpose.

`\subsection{Sample of performance result}`

Here's an example of the performance result part of the log output. Below is an example executed with the machine environment described in `\autoref{s:measuring_env}`.
%
Note that in this program kernel part is iterated one time.

```

\begin{LstLog}
*** Computational Time Report
*** ID=001 : MAIN_dyn_metrics          T=    0.108 N=    1

```

2.9 communication

2.9.1 Description

Kernel communication is taken from the original subroutine `COMM_setup` in *NICAM*. This subroutine is originally defined in module `mod_comm`. This module defines all communication stuff. Subroutine `COMM_setup` prepares necessary information for communication, such as connection list of regions.

As described in [subsection 2.1.6](#) this kernel only needs MPI library to compile/execute.

2.9.2 Discretization and code

(1) COMM_setup

Main subroutine of this kernel program COMM_setup is as follows.

```
1  subroutine COMM_setup
2  use mod_process, only: &
3  PRC_MPIstop
4  use mod_adm, only: &
5  RGNMNG_r2p_pl, &
6  I_NPL, &
7  I_SPL
8  implicit none
9
10 namelist / COMMPARAM / &
11 COMM_apply_barrier, &
12 COMM_varmax, &
13 debug, &
14 testonly
15
16 integer :: ierr
17 !-----
18
19 !--- read parameters
20 write(IO_FID_LOG,*)
21 write(IO_FID_LOG,*) '+++_Module[comm]/Category[common_share]'
22 rewind(IO_FID_CONF)
23 read(IO_FID_CONF,nml=COMMPARAM,iostat=ierr)
24 if ( ierr < 0 ) then
25 write(IO_FID_LOG,*) '***_COMMPARAM_is_not_specified._use_default.'
26 elseif( ierr > 0 ) then
27 write(*,*) 'xxx_Not_appropriate_names_in_namelist_COMMPARAM_STOP.'
28 write(IO_FID_LOG,*) 'xxx_Not_appropriate_names_in_namelist_COMMPARAM_STOP.'
29 call PRC_MPIstop
30 endif
31 write(IO_FID_LOG,nml=COMMPARAM)
32
33 if ( RP == DP ) then
34 COMM_datatype = MPI_DOUBLE_PRECISION
35 elseif( RP == SP ) then
36 COMM_datatype = MPI_REAL
37 else
38 write(*,*) 'xxx_precision_is_not_supported'
39 call PRC_MPIstop
40 endif
41
42 if ( RGNMNG_r2p_pl(I_NPL) < 0 &
43 .AND. RGNMNG_r2p_pl(I_SPL) < 0 ) then
44 COMM_pl = .false.
45 endif
46
47 write(IO_FID_LOG,*)
48 write(IO_FID_LOG,*) '====_communication_information_===='
49
50 call COMM_list_generate
51
52 call COMM_sortdest
53 call COMM_sortdest_pl
54 call COMM_sortdest_singular
55
56 allocate( REQ_list( Recv_nmax_r2r + Send_nmax_r2r &
57 + Recv_nmax_p2r + Send_nmax_p2r &
58 + Recv_nmax_r2p + Send_nmax_r2p ) )
59
60 if( testonly ) call COMM_debugtest
61
62 return
63 end subroutine COMM_setup
```

This subroutine calls five private subroutines after reading control namelist COMMPARAM. Note that as shown below, control variable `testonly` is set as true, `COMM_debugtest` is also called.

(2) COMM_list_generate

Subroutine COMM_list_generate is to setup an array rellist, that contains several information about the relation between grid points including halo.

Initial part of this subroutine is as follows.

```
1  !-----
2  !> Generate inner grid -> halo communication list
3  subroutine COMM_list_generate
4  use mod_adm, only: &
5     ADM_prc_me,      &
6     ADM_lall,       &
7     ADM_gall,       &
8     ADM_gmax,       &
9     ADM_gmin,       &
10    RGNMNG_r2lp,     &
11    RGNMNG_l2r,      &
12    RGNMNG_edge_tab, &
13    RGNMNG_vert_num, &
14    RGNMNG_vert_tab, &
15    I_prc,           &
16    I_RGNID,        &
17    I_DIR,          &
18    I_SW,           &
19    I_NW,           &
20    I_NE,           &
21    I_SE,           &
22    I_W,            &
23    I_N,            &
24    I_E,            &
25    I_S
26  implicit none
27
28  integer :: ginnar
29
30  integer :: prc, prc_rmt
31  integer :: rgnid, rgnid_rmt
32  integer :: i, j, i_rmt, j_rmt
33
34  integer :: n, l, cnt
35  !-----
36
37  ginnar = ADM_gmax - ADM_gmin + 1
38
39  allocate( rellist(rellist_vindex,ADM_gall*ADM_lall) )
40
```

This subroutine is private in module, there are no arguments, but using use to include from other module. Variables with prefix ADM_ define problem size, and variables with prefix RGNMNG_ define tables to manage the relation between regions and processes. Variables with prefix I_ are the constant index and are used to specify the kind of quantity. For example, RGNMNG_r2lp(I_prc,rgnid) means the process number that manage given region with rgnid

The main part of this subroutine is a pretty long l-loop, that is consist of two sections. The first section is as follows.

```
41  cnt = 0
42  do l = 1, ADM_lall
43    rgnid = RGNMNG_l2r(l)
44    prc = ADM_prc_me
45
46    !---< South West >---
47
48    ! NE -> SW halo
49    if ( RGNMNG_edge_tab(I_DIR,I_SW,rgnid) == I_NE ) then
50      rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_SW,rgnid)
51      prc_rmt = RGNMNG_r2lp(I_prc,rgnid_rmt)
52
53      do n = 1, ginnar
54        cnt = cnt + 1
55
56        i = ADM_gmin - 1 + n
57        j = ADM_gmin - 1
58        i_rmt = ADM_gmin - 1 + n
59        j_rmt = ADM_gmax
```

```

60
61     rellist(I_recv_grid,cnt) = suf(i,j)
62     rellist(I_recv_rgn, cnt) = rgnid
63     rellist(I_recv_prc, cnt) = prc
64     rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
65     rellist(I_send_rgn, cnt) = rgnid_rmt
66     rellist(I_send_prc, cnt) = prc_rmt
67   enddo
68   endif
69
70   ! SE -> SW halo (Southern Hemisphere, Edge of diamond)
71   if ( RGNMNG_edge_tab(I_DIR,I_SW,rgnid) == I_SE ) then
72     rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_SW,rgnid)
73     prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
74
75     do n = 1, ginnar
76       cnt = cnt + 1
77
78       i   = ADM_gmin - 1 + n
79       j   = ADM_gmin - 1
80       i_rmt = ADM_gmax
81       j_rmt = ADM_gmax + 1 - n ! reverse order
82
83       rellist(I_recv_grid,cnt) = suf(i,j)
84       rellist(I_recv_rgn, cnt) = rgnid
85       rellist(I_recv_prc, cnt) = prc
86       rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
87       rellist(I_send_rgn, cnt) = rgnid_rmt
88       rellist(I_send_prc, cnt) = prc_rmt
89     enddo
90   endif
91
92   !---< North West >---
93
94   ! SE -> NW
95   if ( RGNMNG_edge_tab(I_DIR,I_NW,rgnid) == I_SE ) then
96     rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_NW,rgnid)
97     prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
98
99     do n = 1, ginnar
100      cnt = cnt + 1
101
102      i   = ADM_gmin - 1
103      j   = ADM_gmin - 1 + n
104      i_rmt = ADM_gmax
105      j_rmt = ADM_gmin - 1 + n
106
107      rellist(I_recv_grid,cnt) = suf(i,j)
108      rellist(I_recv_rgn, cnt) = rgnid
109      rellist(I_recv_prc, cnt) = prc
110      rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
111      rellist(I_send_rgn, cnt) = rgnid_rmt
112      rellist(I_send_prc, cnt) = prc_rmt
113    enddo
114  endif
115
116  ! NE -> NW (Northern Hemisphere, Edge of diamond)
117  if ( RGNMNG_edge_tab(I_DIR,I_NW,rgnid) == I_NE ) then
118    rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_NW,rgnid)
119    prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
120
121    do n = 1, ginnar
122      cnt = cnt + 1
123
124      i   = ADM_gmin - 1
125      j   = ADM_gmin - 1 + n
126      i_rmt = ADM_gmax + 1 - n ! reverse order
127      j_rmt = ADM_gmax
128
129      rellist(I_recv_grid,cnt) = suf(i,j)
130      rellist(I_recv_rgn, cnt) = rgnid
131      rellist(I_recv_prc, cnt) = prc
132      rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
133      rellist(I_send_rgn, cnt) = rgnid_rmt
134      rellist(I_send_prc, cnt) = prc_rmt
135    enddo
136  endif
137
138  !---< North East >---
139
140  ! SW -> NE
141  if ( RGNMNG_edge_tab(I_DIR,I_NE,rgnid) == I_SW ) then

```

```

142   rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_NE,rgnid)
143   prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
144
145   do n = 1, ginnar
146     cnt = cnt + 1
147
148     i   = ADM_gmin - 1 + n
149     j   = ADM_gmax + 1
150     i_rmt = ADM_gmin - 1 + n
151     j_rmt = ADM_gmin
152
153     rellist(I_recv_grid,cnt) = suf(i,j)
154     rellist(I_recv_rgn, cnt) = rgnid
155     rellist(I_recv_prc, cnt) = prc
156     rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
157     rellist(I_send_rgn, cnt) = rgnid_rmt
158     rellist(I_send_prc, cnt) = prc_rmt
159   enddo
160 endif
161
162   ! Nw -> NE (Northern Hemisphere, Edge of diamond)
163   if ( RGNMNG_edge_tab(I_DIR,I_NE,rgnid) == I_NW ) then
164     rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_NE,rgnid)
165     prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
166
167     do n = 1, ginnar
168       cnt = cnt + 1
169
170       i   = ADM_gmin   + n ! shift 1 grid
171       j   = ADM_gmax + 1
172       i_rmt = ADM_gmin
173       j_rmt = ADM_gmax + 1 - n ! reverse order
174
175       rellist(I_recv_grid,cnt) = suf(i,j)
176       rellist(I_recv_rgn, cnt) = rgnid
177       rellist(I_recv_prc, cnt) = prc
178       rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
179       rellist(I_send_rgn, cnt) = rgnid_rmt
180       rellist(I_send_prc, cnt) = prc_rmt
181     enddo
182   endif
183
184   !---< South East >---
185
186   ! Nw -> SE
187   if ( RGNMNG_edge_tab(I_DIR,I_SE,rgnid) == I_NW ) then
188     rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_SE,rgnid)
189     prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
190
191     do n = 1, ginnar
192       cnt = cnt + 1
193
194       i   = ADM_gmax + 1
195       j   = ADM_gmin - 1 + n
196       i_rmt = ADM_gmin
197       j_rmt = ADM_gmin - 1 + n
198
199       rellist(I_recv_grid,cnt) = suf(i,j)
200       rellist(I_recv_rgn, cnt) = rgnid
201       rellist(I_recv_prc, cnt) = prc
202       rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
203       rellist(I_send_rgn, cnt) = rgnid_rmt
204       rellist(I_send_prc, cnt) = prc_rmt
205     enddo
206   endif
207
208   ! Sw -> SE (Southern Hemisphere, Edge of diamond)
209   if ( RGNMNG_edge_tab(I_DIR,I_SE,rgnid) == I_SW ) then
210     rgnid_rmt = RGNMNG_edge_tab(I_RGNID,I_SE,rgnid)
211     prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
212
213     do n = 1, ginnar
214       cnt = cnt + 1
215
216       i   = ADM_gmax + 1
217       j   = ADM_gmin   + n ! shift 1 grid
218       i_rmt = ADM_gmax + 1 - n ! reverse order
219       j_rmt = ADM_gmin
220
221       rellist(I_recv_grid,cnt) = suf(i,j)
222       rellist(I_recv_rgn, cnt) = rgnid
223       rellist(I_recv_prc, cnt) = prc

```

```

224         rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
225         rellist(I_send_rgn, cnt) = rgnid_rmt
226         rellist(I_send_prc, cnt) = prc_rmt
227     enddo
228 endif
229

```

This section sets `rellist` at four edges of each region. The first dimension of `rellist` has the size 6, they specify the pair of source and destination grid point. The data is transferred from source grid to destination grid in halo communication. The second dimension is for the serial number of the grid. That is, `rellist` manages 6 kind of quantities for each inner grid, first three are one-dimensioned grid index from (i, j) , region number and process number, respectively. The grid points identified by these three indices are the destination of data transfer. The latter three are the same but of the informations of the source grid points.

Remember that the shape of the region in *NICAM* is a diamond, and setting of `rellist` is separated for each edge of a diamond. For example, `(RGNMNG_edge_tab(I_DIR,I_SW,rgnid) == I_NE)`(1.49) means that whether the south-west edge of the region with `rgnid` is linked with the north-east edge of neighbouring region. Therefore, this IF-clause sets `rellist` as that the halo grid points of this region are linked with the corresponding inner grid points of the region `rgnid_rmt` which is managed by the process `prc_rmt`.

The continuing section is as follows.

```

230     !---< Vertex : link to the next next region >---
231
232     ! West Vertex
233     if ( RGNMNG_vert_num(I_W,rgnid) == 4 ) then
234         rgnid_rmt = RGNMNG_vert_tab(I_RGNID,I_W,rgnid,3)
235         prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
236
237         cnt = cnt + 1
238
239         i = ADM_gmin - 1
240         j = ADM_gmin - 1
241
242         if ( RGNMNG_vert_tab(I_DIR,I_W,rgnid,3) == I_N ) then
243             i_rmt = ADM_gmin
244             j_rmt = ADM_gmax
245         elseif( RGNMNG_vert_tab(I_DIR,I_W,rgnid,3) == I_E ) then
246             i_rmt = ADM_gmax
247             j_rmt = ADM_gmax
248         elseif( RGNMNG_vert_tab(I_DIR,I_W,rgnid,3) == I_S ) then
249             i_rmt = ADM_gmax
250             j_rmt = ADM_gmin
251         endif
252
253         rellist(I_recv_grid,cnt) = suf(i,j)
254         rellist(I_recv_rgn, cnt) = rgnid
255         rellist(I_recv_prc, cnt) = prc
256         rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
257         rellist(I_send_rgn, cnt) = rgnid_rmt
258         rellist(I_send_prc, cnt) = prc_rmt
259     endif
260
261     ! North Vertex
262     if ( RGNMNG_vert_num(I_N,rgnid) == 4 ) then
263         rgnid_rmt = RGNMNG_vert_tab(I_RGNID,I_N,rgnid,3)
264         prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
265
266         ! known as north pole point
267         if ( RGNMNG_vert_tab(I_DIR,I_N,rgnid,3) == I_W ) then
268             cnt = cnt + 1
269
270             i = ADM_gmin
271             j = ADM_gmax + 1
272             i_rmt = ADM_gmin
273             j_rmt = ADM_gmin
274
275             rellist(I_recv_grid,cnt) = suf(i,j)
276             rellist(I_recv_rgn, cnt) = rgnid
277             rellist(I_recv_prc, cnt) = prc
278             rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
279             rellist(I_send_rgn, cnt) = rgnid_rmt
280             rellist(I_send_prc, cnt) = prc_rmt
281         endif
282

```

```

283      ! unused vertex point
284      cnt = cnt + 1
285
286      i = ADM_gmin - 1
287      j = ADM_gmax + 1
288
289      if ( RGNMNG_vert_tab(I_DIR,I_N,rgnid,3) == I_W ) then
290          i_rmt = ADM_gmin
291          j_rmt = ADM_gmin + 1
292      elseif( RGNMNG_vert_tab(I_DIR,I_N,rgnid,3) == I_N ) then
293          i_rmt = ADM_gmin
294          j_rmt = ADM_gmax
295      elseif( RGNMNG_vert_tab(I_DIR,I_N,rgnid,3) == I_E ) then
296          i_rmt = ADM_gmax
297          j_rmt = ADM_gmax
298      elseif( RGNMNG_vert_tab(I_DIR,I_N,rgnid,3) == I_S ) then
299          i_rmt = ADM_gmax
300          j_rmt = ADM_gmin
301      endif
302
303      rellist(I_recv_grid,cnt) = suf(i,j)
304      rellist(I_recv_rgn, cnt) = rgnid
305      rellist(I_recv_prc, cnt) = prc
306      rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
307      rellist(I_send_rgn, cnt) = rgnid_rmt
308      rellist(I_send_prc, cnt) = prc_rmt
309  endif
310
311  ! East Vertex
312  if ( RGNMNG_vert_num(I_E,rgnid) == 4 ) then
313      if ( RGNMNG_vert_tab(I_DIR,I_E,rgnid,3) == I_W ) then
314          rgnid_rmt = RGNMNG_vert_tab(I_RGNID,I_E,rgnid,3)
315          prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
316
317          cnt = cnt + 1
318
319          i   = ADM_gmax + 1
320          j   = ADM_gmax + 1
321          i_rmt = ADM_gmin
322          j_rmt = ADM_gmin
323
324          rellist(I_recv_grid,cnt) = suf(i,j)
325          rellist(I_recv_rgn, cnt) = rgnid
326          rellist(I_recv_prc, cnt) = prc
327          rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
328          rellist(I_send_rgn, cnt) = rgnid_rmt
329          rellist(I_send_prc, cnt) = prc_rmt
330      endif
331  endif
332
333  ! South Vertex
334  if ( RGNMNG_vert_num(I_S,rgnid) == 4 ) then
335      rgnid_rmt = RGNMNG_vert_tab(I_RGNID,I_S,rgnid,3)
336      prc_rmt   = RGNMNG_r2lp(I_prc,rgnid_rmt)
337
338      ! known as south pole point
339      if ( RGNMNG_vert_tab(I_DIR,I_S,rgnid,3) == I_W ) then
340          cnt = cnt + 1
341
342          i   = ADM_gmax + 1
343          j   = ADM_gmin
344          i_rmt = ADM_gmin
345          j_rmt = ADM_gmin
346
347          rellist(I_recv_grid,cnt) = suf(i,j)
348          rellist(I_recv_rgn, cnt) = rgnid
349          rellist(I_recv_prc, cnt) = prc
350          rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
351          rellist(I_send_rgn, cnt) = rgnid_rmt
352          rellist(I_send_prc, cnt) = prc_rmt
353      endif
354
355      ! unused vertex point
356      cnt = cnt + 1
357
358      i = ADM_gmax + 1
359      j = ADM_gmin - 1
360
361      if ( RGNMNG_vert_tab(I_DIR,I_S,rgnid,3) == I_W ) then
362          i_rmt = ADM_gmin + 1
363          j_rmt = ADM_gmin
364      elseif( RGNMNG_vert_tab(I_DIR,I_S,rgnid,3) == I_N ) then

```

```

365         i_rmt = ADM_gmin
366         j_rmt = ADM_gmax
367         elseif( RGNMNG_vert_tab(I_DIR,I_S,rgnid,3) == I_E ) then
368             i_rmt = ADM_gmax
369             j_rmt = ADM_gmax
370         elseif( RGNMNG_vert_tab(I_DIR,I_S,rgnid,3) == I_S ) then
371             i_rmt = ADM_gmax
372             j_rmt = ADM_gmin
373         endif
374
375         rellist(I_recv_grid,cnt) = suf(i,j)
376         rellist(I_recv_rgn, cnt) = rgnid
377         rellist(I_recv_prc, cnt) = prc
378         rellist(I_send_grid,cnt) = suf(i_rmt,j_rmt)
379         rellist(I_send_rgn, cnt) = rgnid_rmt
380         rellist(I_send_prc, cnt) = prc_rmt
381     endif
382 enddo ! loop l
383
384 rellist_nmax = cnt
385
386 write(IO_FID_LOG,*)
387 write(IO_FID_LOG,*) "***_rellist_nmax:", rellist_nmax
388
389 if ( debug ) then
390     write(IO_FID_LOG,*) "---_Relation_Table"
391     write(IO_FID_LOG,'(7(A10))') 'Count', '|recv_grid', '|_recv_rgn', '|_recv_prc', &
392                                     '|send_grid', '|_send_rgn', '|_send_prc'
393     do cnt = 1, rellist_nmax
394         write(IO_FID_LOG,'(7(I10))') cnt, rellist(:,cnt)
395     enddo
396 endif
397
398 return
399 end subroutine COMM_list_generate

```

Similar to the previous section, this section sets `rellist` additional source-destination relationships of each region. Around the vertices, the data can be transferred from next-next region. In the IF-clause at l.233, (`RGNMNG_vert_num(I_W,rgnid) == 4`) means whether the number of regions around the western vertex of the region with `rgnid` is four, i.e. this vertex is not a singular point, that has only three region links. And (`RGNMNG_vert_tab(I_DIR,I_W,rgnid,3) == I_N`)(l.242) means whether the northern vertex of the next-next region is connected to the western vertex of current region. The regions around the vertex are searched clockwise. Other two IF-clauses(l.245 and l.248) are the same meaning but for the east or west vertex of the next-next region.

At the northern and southern vertices, there are the cases applying special treatment. If the region does not contain north pole at the halo, and the north-western edge is connected to the north-eastern edge of neighboring region, We should take one grid from the next-next region into account. Additionally, in northern and southern vertex section, northern-most and southern-most vertex in each region is not used, respectively (See Figure 1.8 in section 1.8). These points are filled by the value of the grid in the self region.

After the `l`-loop, contents of `rellist` is written to the log (standard output). See subsection 2.9.3 for example.

Also note that even if one region is managed by a single process, all process has the same `rellist` that contains the information of *ALL* regions.

(3) COMM_sortdest

Subroutine `COMM_sortdest` is to sort the region to region connection info.

The first part is as follows.

```

1  !-----
2  !> Sort data destination for region <-> region
3  subroutine COMM_sortdest
4      use mod_process, only: &
5          PRC_LOCAL_COMM_WORLD, &
6          PRC_nprocs
7      use mod_adm, only: &
8          ADM_prc_me, &
9          ADM_kall, &
10         ADM_gall_1d, &

```



```

11     RGNMNG_lp2r,    &
12     RGNMNG_r2lp,    &
13     I_l
14     implicit none
15
16     integer :: sendbuf1(1)
17     integer :: recvbuf1(1)
18
19     integer, allocatable :: sendbuf_info(:)
20     integer, allocatable :: recvbuf_info(:)
21     integer, allocatable :: sendbuf_list(:, :, :)
22     integer, allocatable :: recvbuf_list(:, :, :)
23     integer, allocatable :: REQ_list_r2r(:)
24
25     integer :: Recv_nglobal_r2r
26     integer :: Send_size_nglobal
27
28     integer :: cnt, irank, ipos
29     integer :: totalsize, rank, tag
30
31     integer :: i_from, j_from, r_from, g_from, l_from, p_from
32     integer :: i_to, j_to, r_to, g_to, l_to, p_to
33
34     integer :: ierr
35     integer :: n, p
36     !-----
37
38     allocate( Copy_info_r2r(info_vindex) )
39     allocate( Recv_info_r2r(info_vindex,Recv_nlim) )
40     allocate( Send_info_r2r(info_vindex,Send_nlim) )
41     Copy_info_r2r(:) = -1
42     Recv_info_r2r(:, :) = -1
43     Send_info_r2r(:, :) = -1
44     Copy_info_r2r(I_size) = 0
45     Recv_info_r2r(I_size, :) = 0
46     Send_info_r2r(I_size, :) = 0
47
48     allocate( Copy_list_r2r(list_vindex,rellist_nmax) )
49     allocate( Recv_list_r2r(list_vindex,rellist_nmax,Recv_nlim) )
50     allocate( Send_list_r2r(list_vindex,rellist_nmax,Send_nlim) )
51     Copy_list_r2r(:, :) = -1
52     Recv_list_r2r(:, :, :) = -1
53     Send_list_r2r(:, :, :) = -1
54

```

This subroutine is also private in this module, there are no arguments. Variables with the prefix `sendbuf` and `recvbuf` are temporal containers used for MPI communication. `Send_info_r2r` and `Recv_info_r2r` specify the number of transferring grid points between two precesses, for the sending side and receiving side, respectively. `Copy_info_r2r` is the number of grid points to copy within the same process. `Copy_info_r2r` is one-dimensional and the size is `info_vindex` that is 3, and the index takes one of `I_size`, `I_prc_from` or `I_prc_to`. `Copy_info_r2r(I_size)` is the number of halo-grid points that exchange with the grid points in other region by memory-copy, and `Copy_info_r2r(I_prc_from)` and `Copy_info_r2r(i_prc_to)` are the process number that the process this region exchange from/to, of course this value is the same with `ADM_prc_me`, the process this region is managed.

On the other hand, `Recv_info_r2r` and `Send_info_r2r` are two-dimensional, the first dimension of them are the same with `Copy_info_r2r`. The size of the second dimension of these are `Recv_nlim` and `Send_nlim`, respectively, they are both 10, and this dimension specifies that which process this process receive from/send to.

`Copy_list_r2r` is two-dimensional, the size of these dimension are `list_vindex`, which is 4, and `rellist_nmax`, which is the number of grid points `rellist` contains. First dimension takes one of `I_grid_from`, `I_l_from`, `I_grid_to` or `I_l_to`, they specify the grid point and the (local) region number the value comes from, the grid point and the (local) region number the value goes to, respectively. And second dimension specifies the grid point in this process whose value is exchanged by the memory-copy.

`Recv_list_r2r` and `Send_list_r2r` are similar, but have another dimension, whose size are `Recv_nlim` and `Send_nlim`, respectively. This last dimension has the same meaning with the last dimension of `Recv_info_r2r` and `Send_info_r2r`.

These arrays are set in the next section, as follows.

```

55     ! sorting list according to destination

```

```

56 do cnt = 1, rellist_nmax
57
58   if ( rellist(I_recv_prc,cnt) == rellist(I_send_prc,cnt) ) then ! no communication
59     Copy_info_r2r(I_size) = Copy_info_r2r(I_size) + 1
60     ipos                    = Copy_info_r2r(I_size)
61
62     Copy_list_r2r(I_grid_from,ipos) = rellist(I_send_grid,cnt)
63     Copy_list_r2r(I_l_from ,ipos) = RGNMNG_r2lp(I_l,rellist(I_send_rgn,cnt))
64     Copy_list_r2r(I_grid_to ,ipos) = rellist(I_recv_grid,cnt)
65     Copy_list_r2r(I_l_to ,ipos) = RGNMNG_r2lp(I_l,rellist(I_recv_rgn,cnt))
66   else ! node-to-node communication
67     !--- search existing rank id (identify by prc_from)
68     irank = -1
69     do n = 1, Recv_nmax_r2r
70       if ( Recv_info_r2r(I_prc_from,n) == rellist(I_send_prc,cnt) ) then
71         irank = n
72         exit
73       endif
74     enddo
75
76     if ( irank < 0 ) then ! register new rank id
77       Recv_nmax_r2r = Recv_nmax_r2r + 1
78       irank         = Recv_nmax_r2r
79
80       Recv_info_r2r(I_prc_from,irank) = rellist(I_send_prc,cnt)
81       Recv_info_r2r(I_prc_to ,irank) = rellist(I_recv_prc,cnt)
82     endif
83
84     Recv_info_r2r(I_size,irank) = Recv_info_r2r(I_size,irank) + 1
85     ipos                    = Recv_info_r2r(I_size,irank)
86
87     Recv_list_r2r(I_grid_from,ipos,irank) = rellist(I_send_grid,cnt)
88     Recv_list_r2r(I_l_from ,ipos,irank) = RGNMNG_r2lp(I_l,rellist(I_send_rgn,cnt))
89     Recv_list_r2r(I_grid_to ,ipos,irank) = rellist(I_recv_grid,cnt)
90     Recv_list_r2r(I_l_to ,ipos,irank) = RGNMNG_r2lp(I_l,rellist(I_recv_rgn,cnt))
91   endif
92 enddo
93
94 if ( Copy_info_r2r(I_size) > 0 ) then
95   Copy_nmax_r2r = 1
96   Copy_info_r2r(I_prc_from) = ADM_prc_me
97   Copy_info_r2r(I_prc_to ) = ADM_prc_me
98 endif
99
100
101

```

In the next section there are some MPI communication, as follows.

```

102 ! get maximum number of rank to communication
103 sendbuf1(1) = Recv_nmax_r2r
104
105 call MPI_Allreduce( sendbuf1(1),      & ! [IN]
106                   rcvbuf1(1),      & ! [OUT]
107                   1,                & ! [IN]
108                   MPI_INTEGER,     & ! [IN]
109                   MPI_MAX,         & ! [IN]
110                   PRC_LOCAL_COMM_WORLD, & ! [IN]
111                   ierr              ) ! [OUT]
112
113 Recv_nglobal_r2r = rcvbuf1(1)
114
115 ! Distribute receive request from each rank to all
116 allocate( sendbuf_info(info_vindex*Recv_nglobal_r2r) )
117 allocate( rcvbuf_info(info_vindex*Recv_nglobal_r2r*PRC_nprocs) )
118 sendbuf_info(:) = -1
119
120 do irank = 1, Recv_nmax_r2r
121   n = (irank-1) * info_vindex
122
123   sendbuf_info(n+I_size ) = Recv_info_r2r(I_size ,irank)
124   sendbuf_info(n+I_prc_from) = Recv_info_r2r(I_prc_from,irank)
125   sendbuf_info(n+I_prc_to ) = Recv_info_r2r(I_prc_to ,irank)
126 enddo
127
128 totalsize = info_vindex * Recv_nglobal_r2r
129
130 call MPI_Allgather( sendbuf_info(1), & ! [IN]
131                  totalsize,        & ! [IN]
132                  MPI_INTEGER,      & ! [IN]

```

```

133         recvbuf_info(1),      & ! [OUT]
134         totalsize,           & ! [IN]
135         MPI_INTEGER,         & ! [IN]
136         PRC_LOCAL_COMM_WORLD, & ! [IN]
137         ierr                  ) ! [OUT]
138
139     Send_size_nglobal = 0
140
141     ! Accept receive request to my rank
142     do p = 1, Recv_nglobal_r2r*PRC_nprocs
143         n = (p-1) * info_vindex
144
145         if ( recvbuf_info(n+I_prc_from) == ADM_prc_me ) then
146             Send_nmax_r2r = Send_nmax_r2r + 1
147             irank          = Send_nmax_r2r
148
149             Send_info_r2r(I_size, irank) = recvbuf_info(n+I_size )
150             Send_info_r2r(I_prc_from, irank) = recvbuf_info(n+I_prc_from)
151             Send_info_r2r(I_prc_to, irank) = recvbuf_info(n+I_prc_to )
152         endif
153
154         Send_size_nglobal = max( Send_size_nglobal, recvbuf_info(n+I_size) )
155     enddo
156
157     write(IO_FID_LOG,*)
158     write(IO_FID_LOG,*) '***_Recv_nmax_r2r(global)_{u}', Recv_nglobal_r2r
159     write(IO_FID_LOG,*) '***_Recv_nmax_r2r(local)_{u}', Recv_nmax_r2r
160     write(IO_FID_LOG,*) '***_Send_nmax_r2r(local)_{u}', Send_nmax_r2r
161     write(IO_FID_LOG,*) '***_Send_size_r2r(global)_{u}', Send_size_nglobal
162     write(IO_FID_LOG,*)
163     write(IO_FID_LOG,'(A)')          "|-----"
164     write(IO_FID_LOG,'(A)')          "|_{u}size_{u}prc_from_{u}prc_to"
165     write(IO_FID_LOG,'(A10,3(I10))') "|_{u}Copy_r2r", Copy_info_r2r(:)
166     do irank = 1, Recv_nmax_r2r
167         write(IO_FID_LOG,'(A10,3(I10))') "|_{u}Recv_r2r", Recv_info_r2r(:,irank)
168     enddo
169     do irank = 1, Send_nmax_r2r
170         write(IO_FID_LOG,'(A10,3(I10))') "|_{u}Send_r2r", Send_info_r2r(:,irank)
171     enddo
172

```

In this section, shown as several write statements, Recv_nglobal_r2r, Recv_nmax_r2r, Send_nmax_r2r and Send_size_nglobal are calculated. Recv_nglobal_r2r is the maximum of Recv_nmax_r2r in all processes, number of processes to communicate. This value is calculated by MPI_Allreduce with MPI_MAX operation. Then contents of Recv_info_r2r are MPI_Allgathered and the contents of Recv_info_r2r which related to the current process are additionally stored to the Send_info_r2r.

The next section is as follows.

```

173     ! Communicate detailed information in each pair
174     allocate( REQ_list_r2r(Recv_nmax_r2r+Send_nmax_r2r) )
175
176     allocate( sendbuf_list(list_vindex,Send_size_nglobal,Recv_nmax_r2r) )
177     allocate( recvbuf_list(list_vindex,Send_size_nglobal,Send_nmax_r2r) )
178     sendbuf_list(:, :, :) = -1
179
180     REQ_count = 0
181
182     do irank = 1, Send_nmax_r2r
183         REQ_count = REQ_count + 1
184         totalsize = Send_info_r2r(I_size, irank) * list_vindex
185         rank      = Send_info_r2r(I_prc_to, irank) - 1 ! rank = prc - 1
186         tag       = Send_info_r2r(I_prc_from, irank) - 1 ! rank = prc - 1
187
188         call MPI_IRECV( recvbuf_list(1,1,irank), & ! [OUT]
189             totalsize, & ! [IN]
190             MPI_INTEGER, & ! [IN]
191             rank, & ! [IN]
192             tag, & ! [IN]
193             PRC_LOCAL_COMM_WORLD, & ! [IN]
194             REQ_list_r2r(REQ_count), & ! [OUT]
195             ierr ) ! [OUT]
196     enddo
197
198     do irank = 1, Recv_nmax_r2r
199         do ipos = 1, Recv_info_r2r(I_size, irank)
200             sendbuf_list(:, ipos, irank) = Recv_list_r2r(:, ipos, irank)
201         enddo

```

```

202
203     REQ_count = REQ_count + 1
204     totalsize = Recv_info_r2r(I_size,irank) * list_vindex
205     rank      = Recv_info_r2r(I_prc_from,irank) - 1 ! rank = prc - 1
206     tag       = Recv_info_r2r(I_prc_from,irank) - 1 ! rank = prc - 1
207
208     call MPI_ISEND( sendbuf_list(1,1,irank), & ! [IN]
209                   totalsize,                & ! [IN]
210                   MPI_INTEGER,              & ! [IN]
211                   rank,                     & ! [IN]
212                   tag,                      & ! [IN]
213                   PRC_LOCAL_COMM_WORLD,    & ! [IN]
214                   REQ_list_r2r(REQ_count), & ! [OUT]
215                   ierr                       ) ! [OUT]
216
217     enddo
218
219     !--- wait packets
220     call MPI_WAITALL( Recv_nmax_r2r+Send_nmax_r2r, & ! [IN]
221                    REQ_list_r2r(:),             & ! [IN]
222                    MPI_STATUSES_IGNORE,        & ! [OUT]
223                    ierr                          ) ! [OUT]
224
225     do irank = 1, Send_nmax_r2r
226     do ipos = 1, Send_info_r2r(I_size,irank)
227         Send_list_r2r(:,ipos,irank) = recvbuf_list(:,ipos,irank)
228     enddo
229     enddo

```

In this section each process send the contents of Recv_info_r2r to the corresponding process, then receive and store them to the corresponding location of Send_info_r2r and Send_list_r2r. Note that this procedure is a kind of all to all communication, implemented asynchronous communication, MPI_Irecv, MPI_Isend and MPI_Waitall.

Final section of this subroutine is as follows.

```

230 if ( debug ) then
231     write(IO_FID_LOG,*)
232     write(IO_FID_LOG,*) "---_Copy_list_r2r"
233     write(IO_FID_LOG,*)
234     write(IO_FID_LOG,'(13(A6))') "number", &
235           "|ifrom","jfrom","rfrom","gfrom","lfrom","pfrom", &
236           "|_ujto","_ujto","_urto","_ugto","_ulto","_upto"
237
238     do ipos = 1, Copy_info_r2r(I_size)
239     g_from = Copy_list_r2r(I_grid_from,ipos)
240     l_from = Copy_list_r2r(I_l_from,ipos)
241     p_from = Copy_info_r2r(I_prc_from)
242     i_from = mod(g_from-1,ADM_gall_id) + 1
243     j_from = (g_from-i_from) / ADM_gall_id + 1
244     r_from = RGNMNG_lp2r(l_from,p_from)
245     g_to   = Copy_list_r2r(I_grid_to,ipos)
246     l_to   = Copy_list_r2r(I_l_to,ipos)
247     p_to   = Copy_info_r2r(I_prc_to)
248     i_to   = mod(g_to-1,ADM_gall_id) + 1
249     j_to   = (g_to-i_to) / ADM_gall_id + 1
250     r_to   = RGNMNG_lp2r(l_to,p_to)
251
252     write(IO_FID_LOG,'(13(I6))') ipos, i_from, j_from, r_from, g_from, l_from, p_from, &
253           i_to , j_to , r_to , g_to , l_to , p_to
254
255     enddo
256
257     write(IO_FID_LOG,*)
258     write(IO_FID_LOG,*) "---_Recv_list_r2r"
259     do irank = 1, Recv_nmax_r2r
260     write(IO_FID_LOG,'(13(A6))') "number", &
261           "|ifrom","jfrom","rfrom","gfrom","lfrom","pfrom", &
262           "|_ujto","_ujto","_urto","_ugto","_ulto","_upto"
263
264     do ipos = 1, Recv_info_r2r(I_size,irank)
265     g_from = Recv_list_r2r(I_grid_from,ipos,irank)
266     l_from = Recv_list_r2r(I_l_from,ipos,irank)
267     p_from = Recv_info_r2r(I_prc_from,irank)
268     i_from = mod(g_from-1,ADM_gall_id) + 1
269     j_from = (g_from-i_from) / ADM_gall_id + 1
270     r_from = RGNMNG_lp2r(l_from,p_from)
271     g_to   = Recv_list_r2r(I_grid_to,ipos,irank)
272     l_to   = Recv_list_r2r(I_l_to,ipos,irank)
273     p_to   = Recv_info_r2r(I_prc_to,irank)
274     i_to   = mod(g_to-1,ADM_gall_id) + 1
275     j_to   = (g_to-i_to) / ADM_gall_id + 1

```

```

273     r_to = RGNMNG_lp2r(l_to,p_to)
274
275     write(IO_FID_LOG,'(13(I6))') ipos, i_from, j_from, r_from, g_from, l_from, p_from, &
276         i_to , j_to , r_to , g_to , l_to , p_to
277
278     enddo
279
280     write(IO_FID_LOG,*)
281     write(IO_FID_LOG,*) "---_Send_list_r2r"
282     do irank = 1, Send_nmax_r2r
283         write(IO_FID_LOG,'(13(A6))') "number", &
284             "lfrom","jfrom","rfrom","gfrom","lfrom","pfrom", &
285             "lto","jto","rto","gto","lto","pto"
286
287         do ipos = 1, Send_info_r2r(I_size,irank)
288             g_from = Send_list_r2r(I_grid_from,ipos,irank)
289             l_from = Send_list_r2r(I_l_from,ipos,irank)
290             p_from = Send_info_r2r(I_prc_from,irank)
291             i_from = mod(g_from-1,ADM_gall_id) + 1
292             j_from = (g_from-i_from) / ADM_gall_id + 1
293             r_from = RGNMNG_lp2r(l_from,p_from)
294             g_to = Send_list_r2r(I_grid_to,ipos,irank)
295             l_to = Send_list_r2r(I_l_to,ipos,irank)
296             p_to = Send_info_r2r(I_prc_to,irank)
297             i_to = mod(g_to-1,ADM_gall_id) + 1
298             j_to = (g_to-i_to) / ADM_gall_id + 1
299             r_to = RGNMNG_lp2r(l_to,p_to)
300
301             write(IO_FID_LOG,'(13(I6))') ipos, i_from, j_from, r_from, g_from, l_from, p_from, &
302                 i_to , j_to , r_to , g_to , l_to , p_to
303
304         enddo
305     enddo
306
307     allocate( sendbuf_r2r(Send_size_nglobal*ADM_kall*COMM_varmax,Send_nmax_r2r) )
308     allocate( recvbuf_r2r(Send_size_nglobal*ADM_kall*COMM_varmax,Recv_nmax_r2r) )
309
310     return
311 end subroutine COMM_sortdest

```

This section writes the contents of Copy_list_r2r, Recv_list_r2r, Send_list_r2r, in the form of table, respectively. Note that debug is true in this kernel program. See [subsection 2.9.3](#) for the example of this output.

Regarding to subroutine COMM_sortdest_pl and COMM_sortdest_singular, they do similar procedure with COMM_sortdest, omit here.

(4) COMM_debugtest

COMM_debugtest is to test actual halo-exchange communication routine, COMM_data_transfer and COMM_var.

The first section is as follows.

```

1  subroutine COMM_debugtest
2  use mod_process, only: &
3  PRC_MPIfinish
4  use mod_adm, only: &
5  ADM_prc_me, &
6  ADM_gall, &
7  ADM_gall_pl, &
8  ADM_kall, &
9  ADM_lall, &
10 ADM_lall_pl, &
11 ADM_lall_pl, &
12 ADM_gall_id, &
13 ADM_gmin, &
14 ADM_gmax, &
15 ADM_kmin, &
16 ADM_kmax, &
17 ADM_vlink, &
18 ADM_have_sgp, &
19 RGNMNG_l2r, &
20 RGNMNG_vert_num, &
21 I_N, &
22 I_S
23 implicit none
24

```

```

25 real(RP) :: var (ADM_gall ,ADM_kall,ADM_lall ,4)
26 real(RP) :: var_pl(ADM_gall_pl,ADM_kall,ADM_lall_pl,4)
27
28 integer :: i, j, k, l, ij, rgnid, prc
29 !-----
30

```

As the other subroutines in this section, this is also private in this module, and there are no arguments. var and var_pl are temporal array used in communication.

Main part are separated by two section. The first section is as follows.

```

31 write(IO_FID_LOG,*)
32 write(IO_FID_LOG,*) '+++_TEST_start'
33
34 var (:,:,,:) = -999.DO
35 var_pl(:,:,,:) = -999.DO
36
37 do l = 1, ADM_lall
38   rgnid = RGNMNG_12r(1)
39   prc = ADM_prc_me
40
41   do k = ADM_kmin, ADM_kmax
42     do j = ADM_gmin, ADM_gmax
43       do i = ADM_gmin, ADM_gmax
44         ij = ADM_gall_id * (j-1) + i
45
46         var(ij,k,1,1) = real(prc, kind=RP)
47         var(ij,k,1,2) = real(rgnid,kind=RP)
48         var(ij,k,1,3) = real(i, kind=RP)
49         var(ij,k,1,4) = real(j, kind=RP)
50       enddo
51     enddo
52   enddo
53
54   if ( ADM_have_sgp(1) ) then
55     do k = ADM_kmin, ADM_kmax
56       var(1,k,1,:) = -1.DO
57     enddo
58   endif
59 enddo
60
61 do l = 1, ADM_lall_pl
62   rgnid = 1
63   prc = ADM_prc_me
64
65   do k = ADM_kmin, ADM_kmax
66     do ij = 1, ADM_gall_pl
67       var_pl(ij,k,1,1) = real(-prc, kind=RP)
68       var_pl(ij,k,1,2) = real(-rgnid,kind=RP)
69       var_pl(ij,k,1,3) = real(-ij, kind=RP)
70       var_pl(ij,k,1,4) = real(-ij, kind=RP)
71     enddo
72   enddo
73 enddo
74
75 write(IO_FID_LOG,*) "#####_(prc,rgnid)_#####"
76 do l = 1, ADM_lall
77   do k = ADM_kmin, ADM_kmin
78     write(IO_FID_LOG,*)
79     write(IO_FID_LOG,'(A9)',advance='no') "          |"
80     do i = 1, ADM_gall_id
81       write(IO_FID_LOG,'(I9)',advance='no') i
82     enddo
83     write(IO_FID_LOG,*)
84
85     do j = ADM_gall_id, 1, -1
86       write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
87       do i = 1, ADM_gall_id
88         ij = ADM_gall_id * (j-1) + i
89         write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
90           '( ',int(var(ij,k,1,1)),', ',int(var(ij,k,1,2)),')'
91       enddo
92       write(IO_FID_LOG,*)
93     enddo
94   enddo
95 enddo
96
97 do l = 1, ADM_lall_pl
98   do k = ADM_kmin, ADM_kmin

```

```

99 write(IO_FID_LOG,*)
100 write(IO_FID_LOG,'(A9)',advance='no') "_____|"
101 do i = 1, ADM_gall_pl
102   write(IO_FID_LOG,'(I9)',advance='no') i
103 enddo
104 write(IO_FID_LOG,*)
105
106 write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
107 do ij = 1, ADM_gall_pl
108   write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
109     '(?,int(var_pl(ij,k,1,1)),',',int(var_pl(ij,k,1,2)),')'
110   enddo
111   write(IO_FID_LOG,*)
112 enddo
113 enddo
114
115 write(IO_FID_LOG,*) "#####_(i,j)_#####"
116 do l = 1, ADM_lall
117 do k = ADM_kmin, ADM_kmin
118   write(IO_FID_LOG,*)
119   write(IO_FID_LOG,'(A9)',advance='no') "_____|"
120   do i = 1, ADM_gall_1d
121     write(IO_FID_LOG,'(I9)',advance='no') i
122   enddo
123   write(IO_FID_LOG,*)
124
125   do j = ADM_gall_1d, 1, -1
126     write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
127     do i = 1, ADM_gall_1d
128       ij = ADM_gall_1d * (j-1) + i
129       write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
130         '(?,int(var(ij,k,1,3)),',',int(var(ij,k,1,4)),')'
131     enddo
132     write(IO_FID_LOG,*)
133   enddo
134 enddo
135 enddo
136
137 do l = 1, ADM_lall_pl
138 do k = ADM_kmin, ADM_kmin
139   write(IO_FID_LOG,*)
140   write(IO_FID_LOG,'(A9)',advance='no') "_____|"
141   do i = 1, ADM_gall_pl
142     write(IO_FID_LOG,'(I9)',advance='no') i
143   enddo
144   write(IO_FID_LOG,*)
145
146   write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
147   do ij = 1, ADM_gall_pl
148     write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
149       '(?,int(var_pl(ij,k,1,3)),',',int(var_pl(ij,k,1,4)),')'
150   enddo
151   write(IO_FID_LOG,*)
152 enddo
153 enddo
154
155
156
157 write(IO_FID_LOG,*)
158 write(IO_FID_LOG,*) '+++Communication_start'
159
160 call COMM_data_transfer( var(:,:,:), var_pl(:,:,:) )
161
162 write(IO_FID_LOG,*) "#####_(prc,rgnid)_#####"
163 do l = 1, ADM_lall
164 do k = ADM_kmin, ADM_kmin
165   write(IO_FID_LOG,*)
166   write(IO_FID_LOG,'(A9)',advance='no') "_____|"
167   do i = 1, ADM_gall_1d
168     write(IO_FID_LOG,'(I9)',advance='no') i
169   enddo
170   write(IO_FID_LOG,*)
171
172   do j = ADM_gall_1d, 1, -1
173     write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
174     do i = 1, ADM_gall_1d
175       ij = ADM_gall_1d * (j-1) + i
176       write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
177         '(?,int(var(ij,k,1,1)),',',int(var(ij,k,1,2)),')'
178     enddo
179     write(IO_FID_LOG,*)
180   enddo

```

```

181  enddo
182  enddo
183
184  do l = 1, ADM_lall_pl
185  do k = ADM_kmin, ADM_kmin
186      write(IO_FID_LOG,*)
187      write(IO_FID_LOG,'(A9)',advance='no') " |"
188      do i = 1, ADM_gall_pl
189          write(IO_FID_LOG,'(I9)',advance='no') i
190      enddo
191      write(IO_FID_LOG,*)
192
193      write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
194      do ij = 1, ADM_gall_pl
195          write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
196              '(,int(var_pl(ij,k,1,1)),',',int(var_pl(ij,k,1,2)),')'
197      enddo
198      write(IO_FID_LOG,*)
199  enddo
200  enddo
201
202  write(IO_FID_LOG,*) "#####_(i,j)_#####"
203  do l = 1, ADM_lall
204  do k = ADM_kmin, ADM_kmin
205      write(IO_FID_LOG,*)
206      write(IO_FID_LOG,'(A9)',advance='no') " |"
207      do i = 1, ADM_gall_1d
208          write(IO_FID_LOG,'(I9)',advance='no') i
209      enddo
210      write(IO_FID_LOG,*)
211
212      do j = ADM_gall_1d, 1, -1
213          write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
214          do i = 1, ADM_gall_1d
215              ij = ADM_gall_1d * (j-1) + i
216              write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
217                  '(,int(var(ij,k,1,3)),',',int(var(ij,k,1,4)),')'
218          enddo
219          write(IO_FID_LOG,*)
220      enddo
221  enddo
222  enddo
223
224  do l = 1, ADM_lall_pl
225  do k = ADM_kmin, ADM_kmin
226      write(IO_FID_LOG,*)
227      write(IO_FID_LOG,'(A9)',advance='no') " |"
228      do i = 1, ADM_gall_pl
229          write(IO_FID_LOG,'(I9)',advance='no') i
230      enddo
231      write(IO_FID_LOG,*)
232
233      write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
234      do ij = 1, ADM_gall_pl
235          write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
236              '(,int(var_pl(ij,k,1,3)),',',int(var_pl(ij,k,1,4)),')'
237      enddo
238      write(IO_FID_LOG,*)
239  enddo
240  enddo
241
242
243

```

This section does the test of COMM_data_transfer. As test data, process number prc, region number rgnid, and the grid index i, j are set to var and var_pl. The values of these arrays of before and after the calling COMM_data_transfer are written to the log (standard out).

The latter section is as follows.

```

244  do l = 1, ADM_lall
245      rgnid = RGNMNG_l2r(l)
246      prc = ADM_prc_me
247
248      if ( RGNMNG_vert_num(I_N,rgnid) == ADM_vlink ) then
249          do k = ADM_kmin, ADM_kmax
250              j = ADM_gmax+1
251              i = ADM_gmin
252              ij = ADM_gall_1d * (j-1) + i

```



```

253
254     var(ij,k,1,1) = real(prc, kind=RP)
255     var(ij,k,1,2) = real(rgnid,kind=RP)
256     var(ij,k,1,3) = real(i, kind=RP)
257     var(ij,k,1,4) = real(j, kind=RP)
258     enddo
259   endif
260
261   if ( RGNMNG_vert_num(I_S,rgnid) == ADM_vlink ) then
262     do k = ADM_kmin, ADM_kmax
263       j = ADM_gmin
264       i = ADM_gmax+1
265       ij = ADM_gall_1d * (j-1) + i
266
267       var(ij,k,1,1) = real(prc, kind=RP)
268       var(ij,k,1,2) = real(rgnid,kind=RP)
269       var(ij,k,1,3) = real(i, kind=RP)
270       var(ij,k,1,4) = real(j, kind=RP)
271     enddo
272   endif
273
274 enddo
275
276 write(IO_FID_LOG,*)
277 write(IO_FID_LOG,*) '+++pole_fill_start'
278
279 call COMM_var( var(:,:,:), var_pl(:,:,:), ADM_kall, 4 )
280
281 write(IO_FID_LOG,*) "#####_(prc,rgnid)_#####"
282 do l = 1, ADM_lall
283 do k = ADM_kmin, ADM_kmin
284 write(IO_FID_LOG,*)
285 write(IO_FID_LOG,'(A9)',advance='no') "_____|"
286 do i = 1, ADM_gall_1d
287 write(IO_FID_LOG,'(I9)',advance='no') i
288 enddo
289 write(IO_FID_LOG,*)
290
291 do j = ADM_gall_1d, 1, -1
292 write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
293 do i = 1, ADM_gall_1d
294 ij = ADM_gall_1d * (j-1) + i
295 write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
296 '( ',int(var(ij,k,1,1)),', ',int(var(ij,k,1,2)),')'
297 enddo
298 write(IO_FID_LOG,*)
299 enddo
300 enddo
301
302
303 do l = 1, ADM_lall_pl
304 do k = ADM_kmin, ADM_kmin
305 write(IO_FID_LOG,*)
306 write(IO_FID_LOG,'(A9)',advance='no') "_____|"
307 do i = 1, ADM_gall_pl
308 write(IO_FID_LOG,'(I9)',advance='no') i
309 enddo
310 write(IO_FID_LOG,*)
311
312 write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
313 do ij = 1, ADM_gall_pl
314 write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
315 '( ',int(var_pl(ij,k,1,1)),', ',int(var_pl(ij,k,1,2)),')'
316 enddo
317 write(IO_FID_LOG,*)
318 enddo
319 enddo
320
321 write(IO_FID_LOG,*) "#####_(i,j)_#####"
322 do l = 1, ADM_lall
323 do k = ADM_kmin, ADM_kmin
324 write(IO_FID_LOG,*)
325 write(IO_FID_LOG,'(A9)',advance='no') "_____|"
326 do i = 1, ADM_gall_1d
327 write(IO_FID_LOG,'(I9)',advance='no') i
328 enddo
329 write(IO_FID_LOG,*)
330
331 do j = ADM_gall_1d, 1, -1
332 write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
333 do i = 1, ADM_gall_1d
334 ij = ADM_gall_1d * (j-1) + i

```

```

335         write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
336             '( ',int(var(ij,k,1,3)),', ',int(var(ij,k,1,4)),')'
337     enddo
338     write(IO_FID_LOG,*)
339 enddo
340 enddo
341 enddo
342
343 do l = 1, ADM_lall_pl
344 do k = ADM_kmin, ADM_kmin
345     write(IO_FID_LOG,*)
346     write(IO_FID_LOG,'(A9)',advance='no') "          |"
347     do i = 1, ADM_gall_pl
348         write(IO_FID_LOG,'(I9)',advance='no') i
349     enddo
350     write(IO_FID_LOG,*)
351
352     write(IO_FID_LOG,'(I8,A1)',advance='no') j, "|"
353     do ij = 1, ADM_gall_pl
354         write(IO_FID_LOG,'(A1,I3,A1,I3,A1)',advance='no') &
355             '( ',int(var_pl(ij,k,1,3)),', ',int(var_pl(ij,k,1,4)),')'
356     enddo
357     write(IO_FID_LOG,*)
358 enddo
359 enddo
360
361 return
362 end subroutine COMM_debugtest

```

This section does the test of `COMM_var`. The procedure is the same with the case of `COMM_data_transfer`, using `prc`, `rgnid`, `i` and `j` as a test data, writes out these values the before and after calling this subroutine. See [subsection 2.9.3](#) for example.

2.9.3 Input data and result

This kernel program doesn't need any input data, except configuring namelist file `communication.cnf`. Several configuration is prepared in data directory. The default one is as follows.

```

#####
&ADMPARAM
  glevel = 4,
  rlevel = 0,
  vlayer = 20,
  debug = .true.,
/

&COMPPARAM
  debug = .true.,
  testonly = .true.,
/

#####

```

Script files in run directory executes this kernel program with two MPI processes. Each process outputs logfile named `msg.pe000000` and `msg.pe000001`.

These files are so long and skip here. There are example output files in `reference` directory. These outputs must be the same except white space changes.

2.9.4 Sample of performance result

Here's an example of the performance result part of the log output. Below is an example executed with the machine environment described in [subsection 2.1.7](#). Note that in this program kernel part is iterated one time.

```

*** Computational Time Report
*** Rap level is      2
*** ID=001 : MAIN_Kernel_ALL           T=   0.020 N=   1
*** ID=002 : MAIN_COMM_data_transfer   T=   0.001 N=   2
*** ID=003 : MAIN_COMM_var             T=   0.000 N=   1

```

Bibliography

- H. Tomita and M. Satoh. A new dynamical framework of nonhydrostatic global model using the icosahedral grid. *Fluid Dynamics Research*, 34(6):357–400, 2004.
- M. Satoh, T. Matsuno, H. Tomita, H. Miura, T. Nasuno, and S. Iga. Nonhydrostatic icosahedral atmospheric model (NICAM) for global cloud resolving simulations. *Journal of Computational Physics*, 227(7): 3486–3514, 2008.
- M. Satoh, H. Tomita, H. Yashiro, H. Miura, C. Kodama, T. Seiki, A. T. Noda, Y. Yamada, D. Goto, M. Sawada, T. Miyoshi, Y. Niwa, M. Hara, T. Ohno, S. Iga, T. Arakawa, T. Inoue, and H. Kubokawa. The Non-hydrostatic Icosahedral Atmospheric Model: description and development. *Progress in Earth and Planetary Science*, 1(1):2293, 2014.
- L. J. Wicker and W. C. Skamarock. Time-Splitting Methods for Elastic Models Using Forward Time Schemes. *Monthly Weather Review*, 130(8):2088–2097, 2002.
- G. R. Stuhne and W. R. Peltier. Vortex Erosion and Amalgamation in a New Model of Large Scale Flow on the Sphere. *Journal of Computational Physics*, 128(1):58–81, 1996.
- H. Tomita, K. Goto, and M. Satoh. A new approach to atmospheric general circulation model: Global cloud resolving model NICAM and its computational performance. *SIAM Journal on Scientific Computing*, 30(6):2755–2776, 2008.
- H. Tomita, M. Tsugawa, M. Satoh, and K. Goto. Shallow Water Model on a Modified Icosahedral Geodesic Grid by Using Spring Dynamics. *Journal of Computational Physics*, 174(2):579–613, 2001.
- H. Miura. An Upwind-Biased Conservative Advection Scheme for Spherical Hexagonal–Pentagonal Grids. *Monthly Weather Review*, 135(12):4038–4044, 2007.
- J. Thuburn. Multidimensional flux-limited advection schemes. *Journal of Computational Physics*, 123(1): 74–83, 1996.
- H. Tomita, M. Satoh, H. Miura, and Y. Niwa. Current status of nonhydrostatic modeling at nicam. In *ECMWF Workshop on Non-hydrostatic Modelling, 8-10 November 2010*, pages 171–182, Shinfield Park, Reading, 2010. ECMWF, ECMWF.

IcoAtmosBenchmark NICAM kernels

Author & Editor

SPPEXA/AIMES Benchmarking team

contact : Hisashi Yashiro (RIKEN) h.yashiro@riken.jp

Copyright ©RIKEN, 2017-2018. All rights reserved.