General Estuarine Transport Model

Source Code and Test Case Documentation

Hans Burchard[1], Karsten Bolding[2] and Lars Umlauf[1]

1: Baltic Sea Research Institute Warnemünde, Germany
2: Bolding & Bruggeman ApS, Asperup, Denmark

Version pre_2.6.x (based on v2.5.x code)

August 5, 2021

Contents

1 What's new

- 2011-04-11 - 2012-04-01: Development of stable version v2.2

2 Introduction

2.1 What is GETM?

2.2 A short history of GETM

The idea for GETM was born in May 1997 in Arcachon, France during a workshop of the PhaSE project which was sponsored by the European Community in the framework of the MAST-III programme. It was planned to set up an idealised numerical model for the Eastern Scheldt, The Netherlands for simulating the effect of vertical mixing of nutrients on filter feeder growth rates. A discussion between the first author of this report, Peter Herman (NIOO, Yerseke, The Netherlands) and Walter Eifler (JRC Ispra, Italy) had the result that the associated processes were inherently three-dimensional (in space), and thus, only a three-dimensional model could give satisfying answers. Now the question arose, which numerical model to use. An old wadden sea model by *Burchard* (1995) including a two-equation turbulence model was written in z-coordinates with fixed geopotential layers (which could be added or removed for rising and sinking sea surface elevation, respectively) had proven to be too noisy for the applications in mind. Furthermore, the step-like bottom approximation typical for such models did not seem to be sufficient. Other Public Domain models did not allow for drying and flooding of inter-tidal flats, such as the Princeton Ocean Model (POM). There was thus the need for a new model. Most of the ingredients were however already there. The first author of this report had already written a k-ε turbulence model, see *Burchard and Baumert* (1995), the forerunner of GOTM. A two-dimensional code for general vertical coordinates had been written as well, see *Burchard and Petersen* (1997). And the first author of this report had already learned a lot about mode splitting models from Jean-Marie Beckers (University of Liege, Belgium). Back from Arcachon in Ispra, Italy at the Joint Research Centre of the European Community, the model was basically written during six weeks, after which an idealised tidal simulation for the Sylt-Rømø Bight in the wadden sea area between Germany and Denmark could be successfully simulated, see *Burchard* (1998). By that time this model had the little attractive name *MUDFLAT* which at least well accounted for the models ability to dry and flood inter-tidal flats. At the end of the PhaSE project in 1999, the idealised simulation of mussel growth in the Eastern Scheldt could be finished (not yet published, pers. comm. Francois Lamy and Peter Herman).

In May 1998 the second author of this report joined the development of *MUDFLAT*. He first fully rewrote the model from a one-file FORTRAN77 code to a modular FORTRAN90/95 code, made the interface to GOTM (such that the original k-ε model was not used any more), integrated the netCDF-library into the model, and prepared the parallelisation of the model. And a new name was created, GETM, General Estuarine Transport Model. As already in GOTM, the word "General" does not imply that the model is general, but indicates the motivation to make it more and more general.

At that time, GETM has actually been applied for simulating currents inside the Mururoa atoll in the Pacific Ocean, see *Mathieu et al.* (2002).

During the year 2001, GETM was then extended by the authors of this report to be a fully baroclinic model with transport of active and passive tracers, calculation of density, internal pressure gradient and stratification, surface heat and momentum fluxes and so forth. During a stay of the first author at the Université Catholique de Louvain, Institut d'Astronomie et de Géophysique George Lemaître, Belgium (we are grateful to Eric Deleersnijder for this invitation and many discussions) the high-order advection schemes have been written. During another invitation to Belgium, this time to the GHER at the Université de Liège, the first author had the opportunity to discuss numerical details of GETM with Jean-Marie Beckers, who originally motivated us to use the mode splitting technique.

The typical challenging application in mind of the authors was always a simulation of the tidal Elbe, where baroclinicity and drying and flooding of inter-tidal flats play an important role. Furthermore, the tidal Elbe is long, narrow and bended, such that the use of Cartesian coordinates would require an indexing of the horizontal fields, see e.g. *Duwe* (1988). Thus, the use of curvilinear coordinates which follow the course of the river has already been considered for a long time. However, the extensions just listed above, give the model also the ability to simulate shelf sea processes in fully baroclinic mode, such that the name General Estuarine Transport Model is already a bit too restrictive.

3 The physical equations behind GETM

3.1 Hydrodynamic equations

3.1.1 Three-dimensional momentum equations

For geophysical coastal sea and ocean dynamics, usually the three-dimensional hydrostatic equations of motion with the Boussinesq approximation and the eddy viscosity assumption are used (*Bryan* (1969), *Cox* (1984), *Blumberg and Mellor* (1987), *Haidvogel and Beckmann* (1999), *Kantha and Clayson* (2000b)). In the flux form, the dynamic equations of motion for the horizontal velocity components can be written in Cartesian coordinates as:

$$\begin{aligned}
&\partial_t u + \partial_z(uw) - \partial_z\left((\nu_t+\nu)\partial_z u\right) \\
&\quad +\alpha\Bigg(\partial_x(u^2) + \partial_y(uv) - \partial_x\left(2A_h^M\partial_x u\right) - \partial_y\left(A_h^M(\partial_y u + \partial_x v)\right) \\
&\quad -fv - \int_z^\zeta \partial_x b\, dz'\Bigg) = -g\partial_x\zeta,
\end{aligned} \tag{1}$$

$$\begin{aligned}
&\partial_t v + \partial_z(vw) - \partial_z\left((\nu_t+\nu)\partial_z v\right) \\
&\quad +\alpha\Bigg(\partial_x(vu) + \partial_y(v^2) - \partial_y\left(2A_h^M\partial_y v\right) - \partial_x\left(A_h^M(\partial_y u + \partial_x v)\right) \\
&\quad +fu - \int_z^\zeta \partial_y b\, dz'\Bigg) = -g\partial_y\zeta.
\end{aligned} \tag{2}$$

The vertical velocity is calculated by means of the incompressibility condition:

$$\partial_x u + \partial_y v + \partial_z w = 0. \tag{3}$$

Here, u, v and w are the ensemble averaged velocity components with respect to the x, y and z direction, respectively. The vertical coordinate z ranges from the bottom $-H(x,y)$ to the surface $\zeta(t,x,y)$ with t denoting time. ν_t is the vertical eddy viscosity, ν the kinematic viscosity, f the Coriolis parameter, and g is the gravitational acceleration. The horizontal mixing is parameterised by terms containing the horizontal eddy viscosity A_h^M, see *Blumberg and Mellor* (1987). The buoyancy b is defined as

$$b = -g\frac{\rho-\rho_0}{\rho_0} \tag{4}$$

with the density ρ and a reference density ρ_0. The last term on the left hand sides of equations (1) and (2) are the internal (due to density gradients) and the terms on the right hand sides are the external (due to surface slopes) pressure gradients. In the latter, the deviation of surface density from reference density is neglected (see *Burchard and Petersen* (1997)). The derivation of equations (1) - (3) has been shown in numerous publications, see e.g. *Pedlosky* (1987), *Haidvogel and Beckmann* (1999), *Burchard* (2002b).

In hydrostatic 3D models, the vertical velocity is calculated by means of equation (3) velocity equation. Due to this, mass conservation and free surface elevation can easily be obtained.

Drying and flooding of mud-flats is already incorporated in the physical equations by multiplying some terms with the non-dimensional number α which equals unity in regions where a critical

water depth D_{crit} is exceeded and approaches zero when the water depth D tends to a minimum value D_{min}:

$$\alpha = \min\left\{1, \frac{D - D_{min}}{D_{crit} - D_{min}}\right\}. \tag{5}$$

Thus, $\alpha = 1$ for $D \geq D_{crit}$, such that the usual momentum equation results except for very shallow water, where simplified physics are considered with a balance between tendency, friction and external pressure gradient. In a typical wadden sea application, D_{crit} is of the order of 0.1 m and $D_{\min}$ of the order of 0.02 m (see *Burchard* (1998), *Burchard et al.* (2004)).

3.1.2 Kinematic boundary conditions and surface elevation equation

At the surface and at the bottom, kinematic boundary conditions result from the requirement that the particles at the boundaries are moving along these boundaries:

$$w = \partial_t \zeta + u\partial_x \zeta + v\partial_y \zeta \qquad \text{for } z = \zeta, \tag{6}$$

$$w = -u\partial_x H - v\partial_y H \qquad \text{for } z = -H. \tag{7}$$

3.1.3 Dynamic boundary conditions

At the bottom boundaries, no-slip conditions are prescribed for the horizontal velocity components:

$$u = 0, \quad v = 0. \tag{8}$$

With (7), also $w = 0$ holds at the bottom. It should be noted already here, that the bottom boundary condition (8) is generally not directly used in numerical ocean models, since the near-bottom values of the horizontal velocity components are not located at the bed, but half a grid box above it. Instead, a logarithmic velocity profile is assumed in the bottom layer, leading to a quadratic friction law, see section 8.13.9.

At the surface, the dynamic boundary conditions read:

$$\begin{aligned} (\nu_t + \nu)\partial_z u &= \alpha \tau_s^x, \\ (\nu_t + \nu)\partial_z v &= \alpha \tau_s^y, \end{aligned} \tag{9}$$

The surface stresses (normalised by the reference density) τ_s^x and τ_s^y are calculated as functions of wind speed, wind direction, surface roughness etc. Also here, the drying parameter α is included in order to provide an easy handling of drying and flooding.

3.1.4 Lateral boundary conditions

Let G denote the lateral boundary of the model domain with the closed land boundary G^c and the open boundary G^o such that $G^c \cup G^o = G$ and $G^c \cap G^o = \emptyset$. Let further $\vec{u} = (u, v)$ denote the horizontal velocity vector and $\vec{u}_n = (-v, u)$ its normal vector. At closed boundaries, the flow must be parallel to the boundary:

$$\vec{u}_n \cdot \vec{\nabla} G^c = 0 \tag{10}$$

with $\vec{\nabla} = (\partial_x, \partial_y)$ being the gradient operator.

For an eastern or a western closed boundary with $\vec{\nabla}G^c = (0,1)$ this has the consequence that $u = 0$ and, equivalently, for a southern or a northern closed boundary with $\vec{\nabla}G^c = (1,0)$ this has the consequence that $v = 0$.

At open boundaries, the velocity gradients across the boundary vanish:

$$\vec{\nabla}_n u \cdot \vec{\nabla} G^o = 0, \quad \vec{\nabla}_n v \cdot \vec{\nabla} G^o = 0, \tag{11}$$

with $\vec{\nabla}_n = (-\partial_y, \partial_x)$ being the operator normal to the gradient operator.

For an eastern or a western open boundary with this has the consequence that $\partial_x u = \partial_x v = 0$ and, equivalently, for a southern or a northern open boundary this has the consequence that $\partial_y u = \partial_y v = 0$.

At so-called forced open boundaries, the sea surface elevation ζ is prescribed. At passive open boundaries, it is assumed that the curvature of the surface elevation normal to the boundary is zero, with the consequence that the spatial derivatives of the surface slopes normal to the boundaries vanish.

3.2 GETM as slice model

By chosing the compiler option `SLICE_MODEL` it is possible to operate GETM as a two-dimensional vertical (xz-)model under the assumption that all gradients in y-direction vanish. In order to do so, a bathymetry file with a width of 4 grid points has to be generated, with the outer ($j = 1$, $j = 4$) bathymetry values set to land, and the two inner ones being independent on j. The compiler option `SLICE_MODEL` then sets the transports, velocities, and sea surface elevations such that they are independent of y, i.e. they are forced to be identical for the same j-index. Especially, the V-transports and velocities in the walls ($j = 1$, $j = 3$) are set to the calculated value at index $j = 2$.

4 Transformations

4.1 General vertical coordinates

As a preparation of the discretisation, the physical space is vertically divided into N layers. This is done by introducing internal surfaces z_k, $k = 1, \ldots, N-1$ which do not intersect, each depending on the horizontal position (x, y) and time t. Let

$$-H(x,y) = z_0(x,y) < z_1(x,y,t) < \cdots < z_{N-1}(x,y,t) < z_N(x,y,t) = \zeta(x,y,t) \tag{12}$$

define the local layer depths h_k with

$$h_k = z_k - z_{k-1}. \tag{13}$$

for $1 \leq k \leq N$. For simplicity, the argument (x, y, t) is omitted in most of the cases.

The most simple layer distribution is given by the so-called σ transformation (see *Phillips* (1957) for a first application in meteorology and *Freeman et al.* (1972) for a first application in hydrodynamics) with

$$\sigma_k = \frac{k}{N} - 1 \tag{14}$$

and

$$z_k = D\sigma_k \tag{15}$$

for $0 \leq k \leq N$.

The σ-coordinates can also be refined towards the surface and the bed:

$$\beta_k = \frac{\tanh\left((d_l + d_u)(1 + \sigma_k) - d_l\right) + \tanh(d_l)}{\tanh(d_l) + \tanh(d_u)} - 1, \qquad k = 0, \ldots, N \tag{16}$$

such that z-levels are obtained as follows:

$$z_k = D\beta_k \tag{17}$$

for $0 \leq k \leq N$.

The grid is refined towards the surface for $d_u > 0$ and refined towards the bottom for $d_l > 0$. When both, d_u and d_l are larger than zero, then refinement towards surface and bed is obtained. For $d_u = d_l = 0$ the σ-transformation (14) with $\beta_k = \sigma_k$ is retained. Figure 1 shows four examples for vertical layer distributions obtained with the σ-transformation.

Due to the fact that all layer thicknesses are proportional to the water depth, the equidistant and also the non-equidistant σ-transformations, (14) and (16), have however one striking disadvantage. In order to sufficiently resolve the mixed layer also in deep water, many layers have to be located near the surface. The same holds for the bottom boundary layer. This problem of σ-coordinates has been discussed by several authors (see e.g. *Deleersnijder and Ruddick* (1992), *de Kok* (1992), *Gerdes* (1993), *Song and Haidvogel* (1994), *Burchard and Petersen* (1997)) who suggested methods for generalised vertical coordinates not resulting in layer thicknesses not proportional to the water depth.

The generalised vertical coordinate introduced here is a generalisation of the so-called mixed-layer transformation suggested by *Burchard and Petersen* (1997). It is a hybrid coordinate which interpolates between the equidistant and the non-equidistant σ-transformations given by (14) and (16). The weight for the interpolation depends on the ratio of a critical water depth D_γ (below which equidistant σ-coordinates are used) and the actual water depth:

$$z_k = D\left(\alpha_\gamma \sigma_k + (1 - \alpha_\gamma)\beta_k\right) \tag{18}$$

with

$$\alpha_\gamma = \min\left(\frac{(\beta_k - \beta_{k-1}) - \frac{D_\gamma}{D}(\sigma_k - \sigma_{k-1})}{(\beta_k - \beta_{k-1}) - (\sigma_k - \sigma_{k-1})}, 1\right). \tag{19}$$

and σ_k from (14) and β_k from (16).

For inserting $k = N$ in (19) and $d_l = 0$ and $d_u > 0$ in (16), the mixed layer transformation of *Burchard and Petersen* (1997) is retained, see the upper two panels in figure 2. Depending on the values for D_γ and d_u, some near-surface layer thicknesses will be constant in time and space, allowing for a good vertical resolution in the surface mixed layer.

The same is obtained for the bottom with the following settings: $k = 1$, $d_l > 0$ and $d_u = 0$, see the lower two panels in figure 2. This is recommended for reproducing sedimentation dynamics and other benthic processes. For $d_l = d_u > 0$ and $k = 1$ or $k = N$ a number of layers near the surface and near the bottom can be fixed to constant thickness. Intermediate states are obtained by intermediate settings, see figure 3. Some pathological settings are also possible, such as $k = 1$, $d_l = 1.5$ and $d_u = 5$, see figure 4.

The strong potential of the general vertical coordinates concept is the extendibility towards vertically adaptive grids. Since the layers may be redistributed after every baroclinic time step, one could adapt the coordinate distribution to the internal dynamics of the flow. One could for example concentrate more layers at vertical locations of high stratification and shear, or force certain layer interfaces towards certain isopycnals, or approximate Lagrangian vertical coordinates

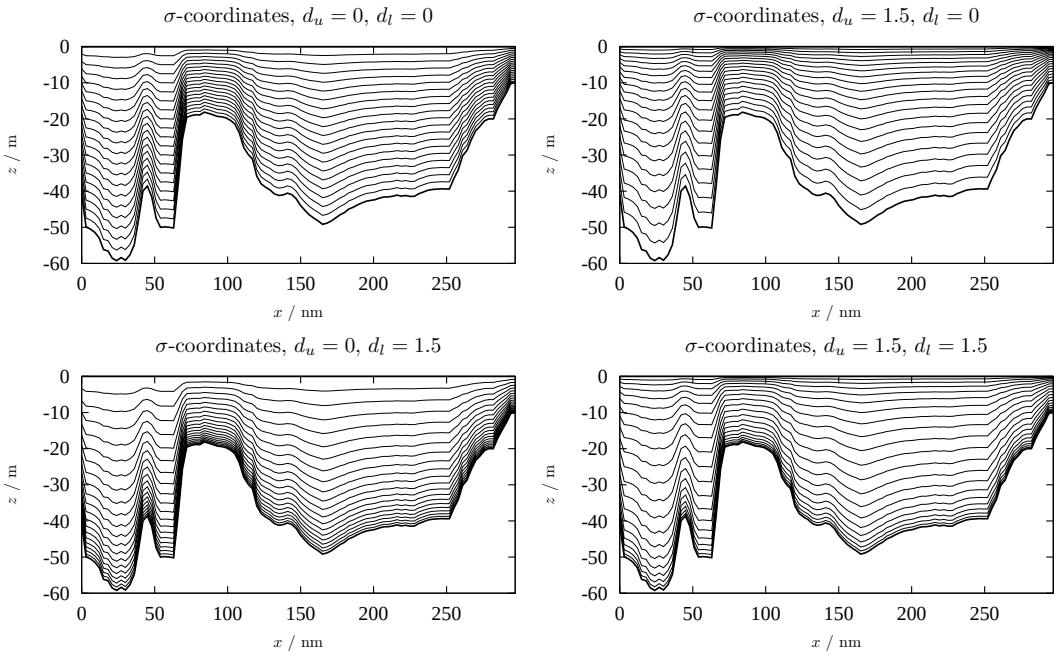

Figure 1: σ-transformation with four different zooming options. The plots show the vertical layer distribution for a cross section through the North Sea from Scarborough in England to Esbjerg in Denmark. The shallow area at about $x = 100$ nm is the Doggerbank.

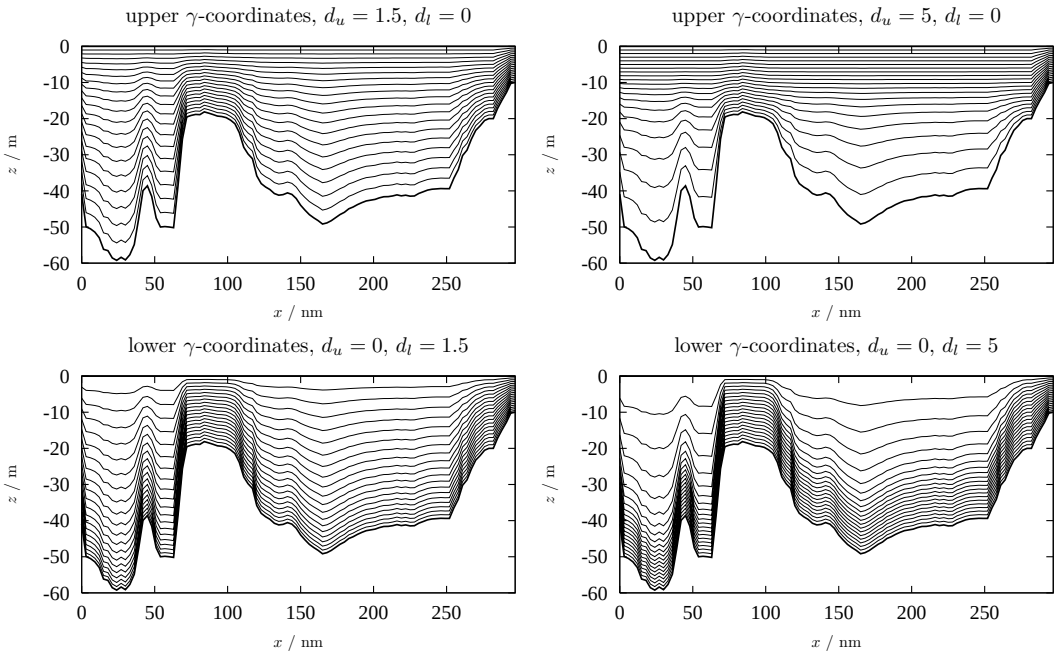

Figure 2: Boundary layer transformation (or γ transformation) with concentration of layers in the surface mixed layer (upper two panels) and with concentration of layers in the bottom mixed layer (lower two panels). The critical depth D_γ is here set to 20 m, such that at all shallower depths the equidistant σ-transformation is used. The same underlying bathymetry as in figure 1 has been used.

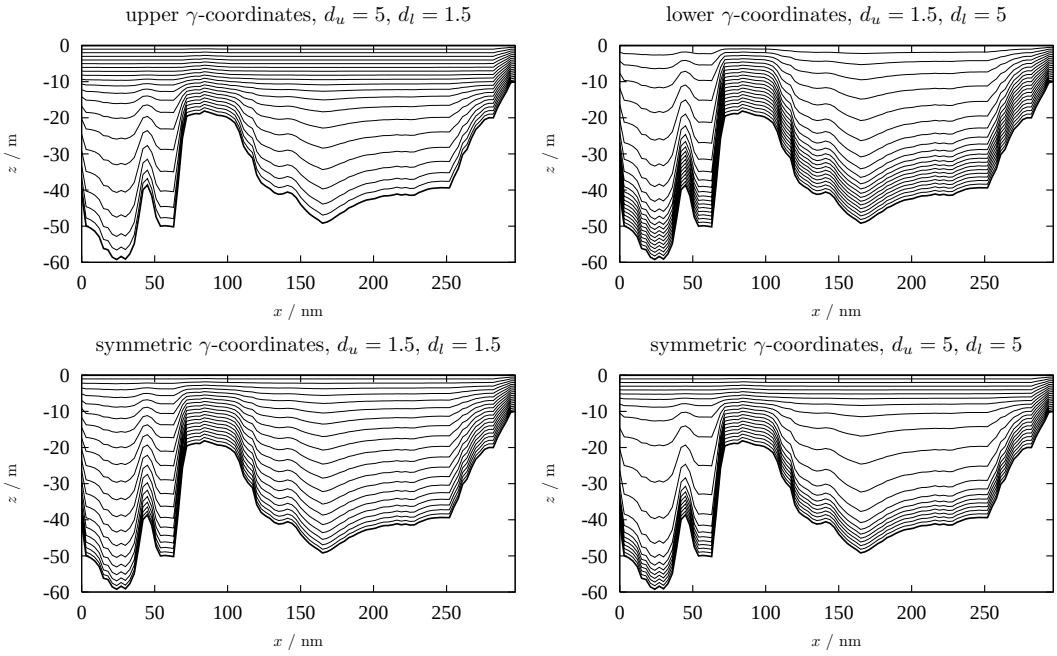

Figure 3: Boundary layer transformation (or γ transformation) with concentration of layers in both, the surface mixed layer and the bottom mixed layer. Four different realisations are shown. The critical depth D_γ is here set to 20 m, such that at all shallower depths the equidistant σ-transformation is used. The same underlying bathymetry as in figure 1 has been used.

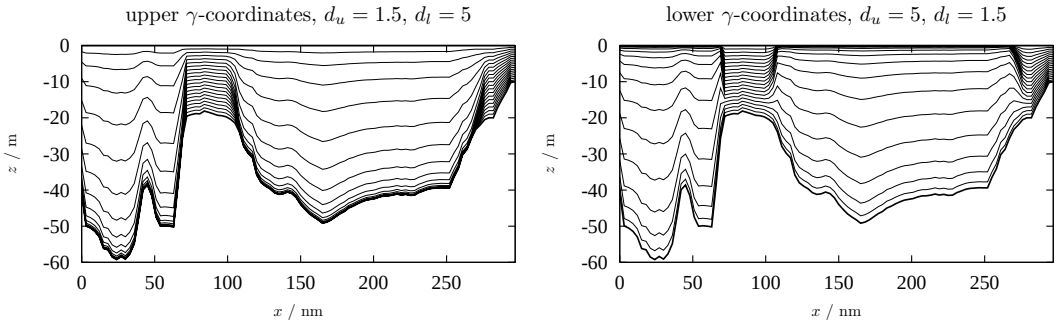

Figure 4: Two pathological examples for the boundary layer transformation. The critical depth D_γ is here set to 20 m, such that at all shallower depths the equidistant σ-transformation is used. The same underlying bathymetry as in figure 1 has been used.

by minimising the vertical advection through layer interfaces. The advantages of this concept have recently been demonstrated for one-dimensional water columns by *Burchard and Beckers* (2004). The three-dimensional generalisation of this concept of adaptive grids for GETM is currently under development.

4.2 Layer-integrated equations

There are two different ways to derive the layer-integrated equations. *Burchard and Petersen* (1997) transform first the equations into general vertical coordinate form (see *Deleersnijder and Ruddick* (1992)) and afterwards integrate the transformed equations over constant intervals in the transformed space. *Lander et al.* (1994) integrate the equations in the Cartesian space over surfaces z_k by considering the Leibniz rule

$$\int_{z_{k-1}}^{z_k} \partial_x f \, dz = \partial_x \int_{z_{k-1}}^{z_k} f \, dz - f(z_k)\partial_x z_k + f(z_{k-1})\partial_x z_{k-1} \tag{20}$$

for any function f. For the vertical staggering of the layer notation see figure 7.
More details about the layer integration are given in *Burchard and Petersen* (1997).
With the further definitions of layer integrated transport,

$$p_k := \int_{z_{k-1}}^{z_k} u \, dz, \qquad q_k := \int_{z_{k-1}}^{z_k} v \, dz, \tag{21}$$

layer mean velocities,

$$u_k := \frac{p_k}{h_k}, \qquad v_k := \frac{q_k}{h_k}, \tag{22}$$

and layer averaged tracer concentrations and buoyancy,

$$c_k^i := \frac{1}{h_k}\int_{z_{k-1}}^{z_k} c^i \, dz, \qquad b_k := \frac{1}{h_k}\int_{z_{k-1}}^{z_k} b \, dz, \tag{23}$$

and the grid related vertical velocity,

$$\bar{w}_k := (w - \partial_t z - u\partial_x z - v\partial_y z)_{z=z_k}, \tag{24}$$

the continuity equation (3) has the layer-integrated form:

$$\partial_t h_k + \partial_x p_k + \partial_y q_k + \bar{w}_k - \bar{w}_{k-1} = 0. \tag{25}$$

It should be noted that the grid related velocity is located on the layer interfaces. After this, the layer-integrated momentum equations read as:

$$\begin{aligned}
&\partial_t p_k + \bar{w}_k\tilde{u}_k - \bar{w}_{k-1}\tilde{u}_{k-1} - \tau_k^x + \tau_{k-1}^x \\
&+\alpha\Big\{\partial_x(u_k p_k) + \partial_y(v_k p_k) \\
&-\partial_x\left(2A_k^M h_k \partial_x u_k\right) - \partial_y\left(A_k^M h_k(\partial_y u_k + \partial_x v_k)\right) - f q_k \\
&-h_k\left(\frac{1}{2}h_N(\partial_x^* b)_N + \sum_{j=k}^{N-1}\frac{1}{2}(h_j + h_{j+1})(\partial_x^* b)_j\right)\Big\} = -g h_k \partial_x \zeta,
\end{aligned} \tag{26}$$

$$
\begin{aligned}
&\partial_t q_k + \bar{w}_k \tilde{v}_k - \bar{w}_{k-1}\tilde{v}_{k-1} - \tau^y_k + \tau^y_{k-1} \\
&+\alpha\Bigg\{ \partial_x(u_k q_k) + \partial_y(v_k q_k) \\
&-\partial_y\left(2A^M_k h_k \partial_y v_k\right) - \partial_x\left(A^M_k h_k(\partial_y u_k + \partial_x v_k)\right) + f p_k \\
&-h_k\left(\frac{1}{2}h_N(\partial^*_y b)_N + \sum_{j=k}^{N-1}\frac{1}{2}(h_j + h_{j+1})(\partial^*_y b)_j\right)\Bigg\} = -g h_k \partial_y \zeta
\end{aligned}
\tag{27}
$$

with suitably chosen advective horizontal velocities $\tilde{u}_k$ and $\tilde{v}_k$ (see section 8.13.7) on page 173, the shear stresses

$$
\tau^x_k = \left(\nu_t \partial_z u\right)_k , \tag{28}
$$

and

$$
\tau^y_k = \left(\nu_t \partial_z v\right)_k , \tag{29}
$$

and the horizontal buoyancy gradients

$$
(\partial^*_x b)_k = \frac{1}{2}(\partial_x b_{k+1} + \partial_x b_k) - \partial_x z_k \frac{b_{k+1} - b_k}{\frac{1}{2}(h_{k+1} + h_k)} \tag{30}
$$

and

$$
(\partial^*_y b)_k = \frac{1}{2}(\partial_y b_{k+1} + \partial_y b_k) - \partial_y z_k \frac{b_{k+1} - b_k}{\frac{1}{2}(h_{k+1} + h_k)}. \tag{31}
$$

The layer integration of the pressure gradient force is discussed in detail by *Burchard and Petersen* (1997).

A conservative formulation can be derived for the recalculation of the physical vertical velocity w which is convenient in the discrete space if w is evaluated at the layer centres (see *Deleersnijder and Ruddick* (1992)):

$$
w_k = \frac{1}{h_k}\left(\partial_t(h_k z_{k-1/2}) + \partial_x(p_k z_{k-1/2}) + \partial_y(q_k z_{k-1/2}) + \bar{w}_k z_k - \bar{w}_{k-1} z_{k-1}\right). \tag{32}
$$

It should be mentioned that w only needs to be evaluated for post-processing reasons.

For the layer-integrated tracer concentrations, we obtain the following expression:

$$
\begin{aligned}
&\partial_t(h_k c^i_k) + \partial_x(p_k c^i_k) + \partial_y(q_k c^i_k) + (\bar{w}_k + w^s_k)\tilde{c}^i_k - (\bar{w}_{k-1} + w^s_{k-1})\tilde{c}^i_{k-1} \\
&-(\nu'_t \partial_z c^i)_k + (\nu'_t \partial_z c^i)_{k-1} - \partial_x\left(A^T_k h_k \partial_x c^i_k\right) - \partial_y\left(A^T_k h_k \partial_y c^i_k\right) = Q^i_k.
\end{aligned}
\tag{33}
$$

It should be noted that the "horizontal" diffusion does no longer occur along geopotential surfaces but along horizontal coordinate lines. The properly transformed formulation would include some cross-diagonal terms which may lead to numerical instabilities due to violation of monotonicity. For an in-depth discussion of this problem, see *Beckers et al.* (1998) and *Beckers et al.* (2000).

4.3 Horizontal curvilinear coordinates

In this section, the layer-integrated equations from section 4 are transformed to horizontal orthogonal curvilinear coordinates. Similarly to general coordinates in the vertical, these allow for much more flexibility when optimising horizontal grids to coast-lines and bathymetry. Furthermore, this type of coordinates system includes spherical coordinates as a special case. The derivation of the transformed equations is carried out here according to *Haidvogel and Beckmann* (1999), see also *Arakawa and Lamb* (1977).

A rectangular domain with non-dimensional side lengths and with local Cartesian coordinates $\mathcal{X}$ and $\mathcal{Y}$ is mapped to a physical domain with four corners in such a way that the local coordinates of the physical space, (ξ_x, ξ_y) are orthogonal to each others everywhere:

$$\mathcal{X} \to \xi_x, \quad \mathcal{Y} \to \xi_y. \tag{34}$$

The infinitesimal increments in the physical space, $d\,\xi_x$ and $d\,\xi_y$ are related to the infinitesimal increments in the transformed space, $d\,\mathcal{X}$ and $d\,\mathcal{Y}$ by so-called metric coefficients $m(x,y)$ and $n(x,y)$:

$$d\,\xi_x = \left(\frac{1}{m}\right) d\,\mathcal{X}, \quad d\,\xi_y = \left(\frac{1}{n}\right) d\,\mathcal{Y}. \tag{35}$$

These metric coefficients have the physical unit of $[\mathrm{m}^{-1}]$. With $m = n =$const, Cartesian coordinates are retained, and with

$$m = \frac{1}{r_E \cos\phi}, \quad n = \frac{1}{r_E}, \tag{36}$$

spherical coordinates with $\mathcal{X} = \lambda$ and $\mathcal{Y} = \phi$ are retained (with the Earth's radius r_E, longitude λ and latitude ϕ).

With these notations, the layer-integrated equations (25), (26), and (27) given in section 4 can be formulated as follows:

Continuity equation:

$$\partial_t \left(\frac{h_k}{mn}\right) + \partial_{\mathcal{X}} \left(\frac{p_k}{n}\right) + \partial_{\mathcal{Y}} \left(\frac{q_k}{m}\right) + \frac{\bar{w}_k - \bar{w}_{k-1}}{mn} = 0. \tag{37}$$

Momentum in ξ_x direction:

$$\begin{aligned}
&\partial_t \left(\frac{p_k}{mn}\right) + \frac{\bar{w}_k \tilde{u}_k - \bar{w}_{k-1}\tilde{u}_{k-1}}{mn} - \frac{\tau^{\mathcal{X}}_k - \tau^{\mathcal{X}}_{k-1}}{mn} \\
&+\alpha \Bigg\{ \partial_{\mathcal{X}} \left(\frac{u_k p_k}{n}\right) + \partial_{\mathcal{Y}} \left(\frac{v_k p_k}{m}\right) - q_k \left(\frac{f}{mn} + v_k \partial_{\mathcal{X}} \left(\frac{1}{n}\right) - u_k \partial_{\mathcal{Y}} \left(\frac{1}{m}\right)\right) \\
&-\partial_{\mathcal{X}} \left(\frac{2A^M_k h_k}{n} m \partial_{\mathcal{X}} u_k\right) - \partial_{\mathcal{Y}} \left(\frac{A^M_k h_k}{m}(n\partial_{\mathcal{Y}} u_k + m\partial_{\mathcal{X}} v_k)\right) \\
&-\frac{h_k}{n}\left(\frac{1}{2} h_N (\partial^*_{\mathcal{X}} b)_N + \sum_{j=k}^{N-1} \frac{1}{2}(h_j + h_{j+1})(\partial^*_{\mathcal{X}} b)_j\right) \Bigg\} = -g \frac{h_k}{n} \partial_{\mathcal{X}} \zeta.
\end{aligned} \tag{38}$$

Momentum in ξ_y direction:

$$\begin{aligned}
&\partial_t\left(\frac{q_k}{mn}\right)+\frac{\bar{w}_k\tilde{v}_k-\bar{w}_{k-1}\tilde{v}_{k-1}}{mn}-\frac{\tau^{\mathcal{Y}}_k-\tau^{\mathcal{Y}}_{k-1}}{mn}\\
&+\alpha\Bigg\{\partial_{\mathcal{X}}\left(\frac{u_kq_k}{n}\right)+\partial_{\mathcal{Y}}\left(\frac{v_kq_k}{m}\right)+p_k\left(\frac{f}{mn}+v_k\partial_{\mathcal{X}}\left(\frac{1}{n}\right)-u_k\partial_{\mathcal{Y}}\left(\frac{1}{m}\right)\right)\\
&-\partial_{\mathcal{Y}}\left(\frac{2A^M_k h_k}{m}n\partial_{\mathcal{Y}}v_k\right)-\partial_{\mathcal{X}}\left(\frac{A^M_k h_k}{n}(n\partial_{\mathcal{Y}}u_k+m\partial_{\mathcal{X}}v_k)\right)\\
&-\frac{h_k}{m}\left(\frac{1}{2}h_N(\partial^*_{\mathcal{Y}}b)_N+\sum_{j=k}^{N-1}\frac{1}{2}(h_j+h_{j+1})(\partial^*_{\mathcal{Y}}b)_j\right)\Bigg\}=-g\frac{h_k}{m}\partial_{\mathcal{Y}}\zeta.
\end{aligned}\tag{39}$$

In (38) and (39), the velocity and momentum components u_k and p_k are now pointing into the ξ_x-direction and v_k and q_k are pointing into the ξ_y-direction. The stresses $\tau^{\mathcal{X}}_k$ and $\tau^{\mathcal{Y}}_k$ are related to these directions as well. In order to account for this rotation of the velocity and momentum vectors, the rotational terms due to the Coriolis rotation are extended by terms related to the gradients of the metric coefficients. This rotation is here not considered for the horizontal diffusion terms in order not to unnecessarily complicate the equations. Instead we use the simplified formulation by *Kantha and Clayson* (2000b), who argue that it does not make sense to use complex formulations for minor processes with highly empirical parameterisations.

Finally, the tracer equation is of the following form after the transformation to curvilinear coordinates:

$$\begin{aligned}
&\partial_t\left(\frac{h_kc^i_k}{mn}\right)+\partial_{\mathcal{X}}\left(\frac{p_kc^i_k}{n}\right)+\partial_{\mathcal{Y}}\left(\frac{q_kc^i_k}{m}\right)+\frac{\bar{w}_k\tilde{c}^i_k-\bar{w}_{k-1}\tilde{c}^i_{k-1}}{mn}\\
&-\frac{(\nu'_t\partial_zc^i)_k-(\nu'_t\partial_zc^i)_{k-1}}{mn}\\
&-\partial_{\mathcal{X}}\left(\frac{A^T_k h_k}{n}m\partial_{\mathcal{X}}c^i_k\right)-\partial_{\mathcal{Y}}\left(\frac{A^T_k h_k}{m}n\partial_{\mathcal{Y}}c^i_k\right)=\frac{Q^i_k}{mn}.
\end{aligned}\tag{40}$$

5 Discretisation

5.1 Mode splitting

The external system consisting of the surface elevation equation (57) and the transport equations (61) and (62) underlies a strict time step constraint if the discretisation is carried out explicitly:

$$\Delta t<\left[\frac{1}{2}\left(\frac{1}{\Delta x}+\frac{1}{\Delta y}\right)\sqrt{2gD}\right]^{-1}.\tag{41}$$

In contrast to that, the time step of the internal system is only depending on the Courant number for advection,

$$\Delta t<\min\left\{\frac{\Delta x}{u_{\max}},\frac{\Delta y}{v_{\max}}\right\},\tag{42}$$

which in the case of sub-critical flow is a much weaker constraint. In order not to punish the whole model with a small time step resulting from the external system, two different approaches of mode splitting have been developed in the past.

The first approach, in which the external mode is calculated implicitly, has been proposed by *Madala and Piacsek* (1977). This method is numerically stable (if advection is absent) for unconditionally long time steps. The temporal approximation is of second order if semi-implicit treatment is chosen. In such models, the external and internal mode are generally calculated with the same time steps (see e.g. *Backhaus* (1985)). The introduction of interactions terms like (63) - (70) is thus not necessary in such models.

Another approach is to use different time steps for the internal (macro times steps Δt) and the external mode (micro time steps Δt_m). One of the first free surface models which has adopted this method is the Princeton Ocean Model (POM), see *Blumberg and Mellor* (1987). This method has the disadvantage that interaction terms are needed for the external mode and that the consistency between internal and external mode is difficult to obtain. The advantage of this method is that the free surface elevation is temporally well resolved which is a major requirement for models including flooding and drying. That is the reason why this method is adopted here.

The micro time step Δt_m has to be an integer fraction M of the macro time step Δt. Δt_m is limited by the speed of the surface waves (41), Δt is limited by the current speed (42). The time stepping principle is shown in figure 5. The vertically integrated transports are averaged over each macro time step:

$$\bar{U}_{i,j}^{n+1/2} = \frac{1}{M} \sum_{l=n+0.5/M}^{n+(M-0.5)/M} U_{i,j}^{l} \tag{43}$$

and

$$\bar{V}_{i,j}^{n+1/2} = \frac{1}{M} \sum_{l=n+0.5/M}^{n+(M-0.5)/M} V_{i,j}^{l} \tag{44}$$

such that

$$\frac{\zeta_{i,j}^{n+1} - \zeta_{i,j}^{n}}{\Delta t} = -\frac{\bar{U}_{i,j}^{n+1/2} - \bar{U}_{i-1,j}^{n+1/2}}{\Delta x} - \frac{\bar{V}_{i,j}^{n+1/2} - \bar{V}_{i,j-1}^{n+1/2}}{\Delta y}. \tag{45}$$

5.2 Spatial discretisation

For the spatial discretisation, a staggered C-grid is used, see *Arakawa and Lamb* (1977). The grid consists of prism-shaped finite volumes with the edges aligned with coordinates. The reference grid for the tracer points (from now on denoted by T-points) is shown in figures 6 and 7. The velocity points are located such that the corresponding velocity components are centralised on the surfaces of the T-point reference box, the u-velocity points (from now on U-points) at the western and eastern surfaces, the v-velocity points (from now on V-points) at the southern and northern surfaces and the w-velocity points (from now on W-points) at the lower and upper surfaces.

The indexing is carried out with i-indices in eastern ($\mathcal{X}$-) direction, with j-indices in northern ($\mathcal{Y}$-) direction and with k-indices in upward (z-) direction, such that each grid point is identified by a triple (i, j, k). A T-point and the corresponding eastern U-point, the northern V-point and the above W-point have always the same index, see figures 6 and 7. The different grid points cover the following index ranges:

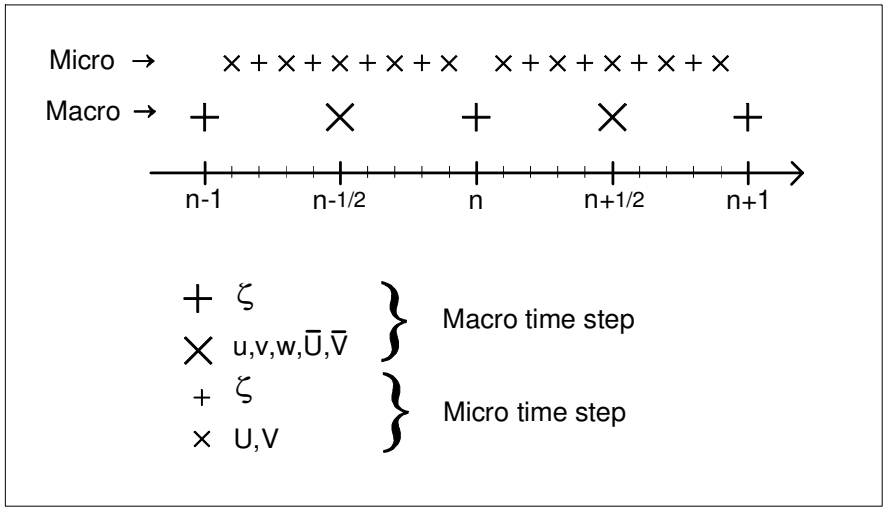

Figure 5: Sketch explaining the organisation of the time stepping.

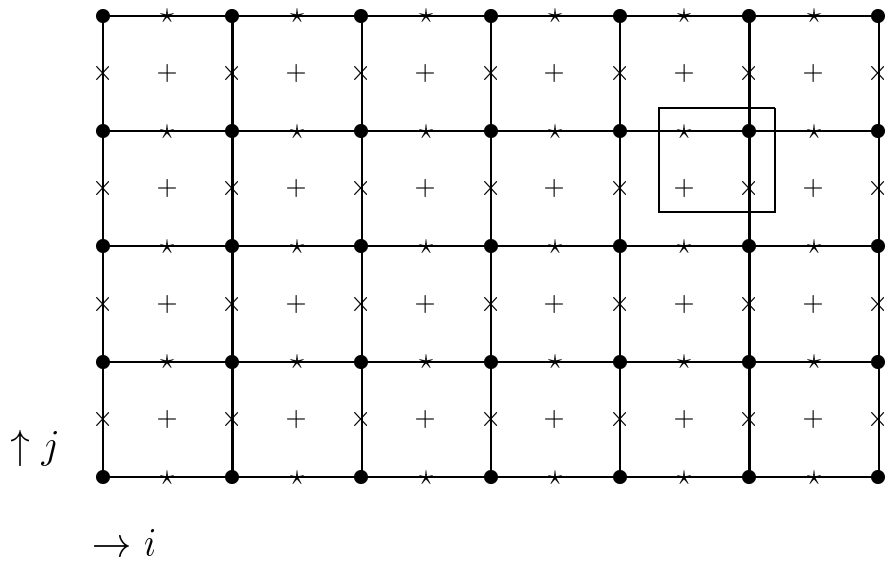

Figure 6: Layout of the model horizontal model grid in Cartesian coordinates. Shown are the reference boxes for the T-points. The following symbols are used: $+$: T-points; $\times$: U-points; $\star$: V-points; $\bullet$: X-points. The inserted box denotes grid points with the same index (i, j).

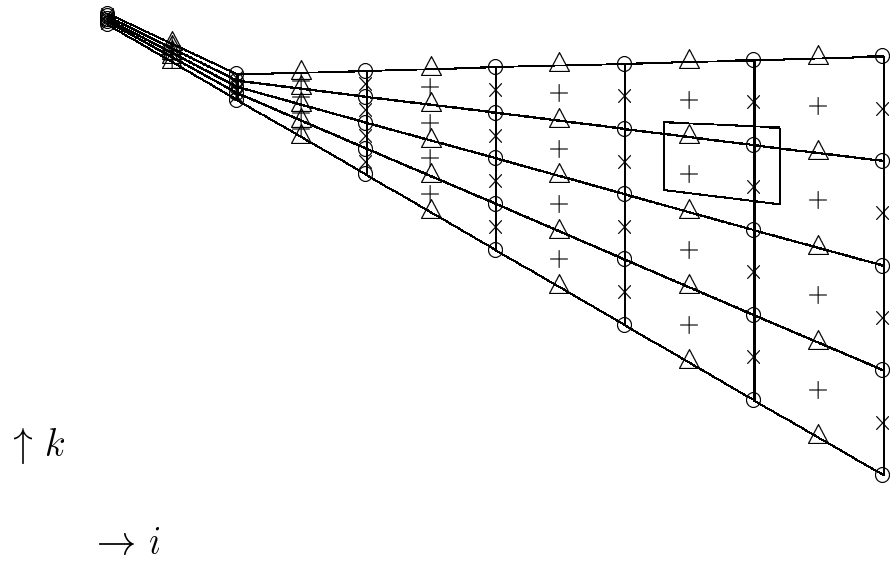

Figure 7: Layout of the model vertical model grid through the U-points. Shown are the reference boxes for the T-points. The following symbols are used: +: T-points; ×: U-points; △: W-points; ∘: X^u-points. The inserted box denotes grid points with the same index (i, k). The grid in the (j, k)-plane through the V-points is equivalent.

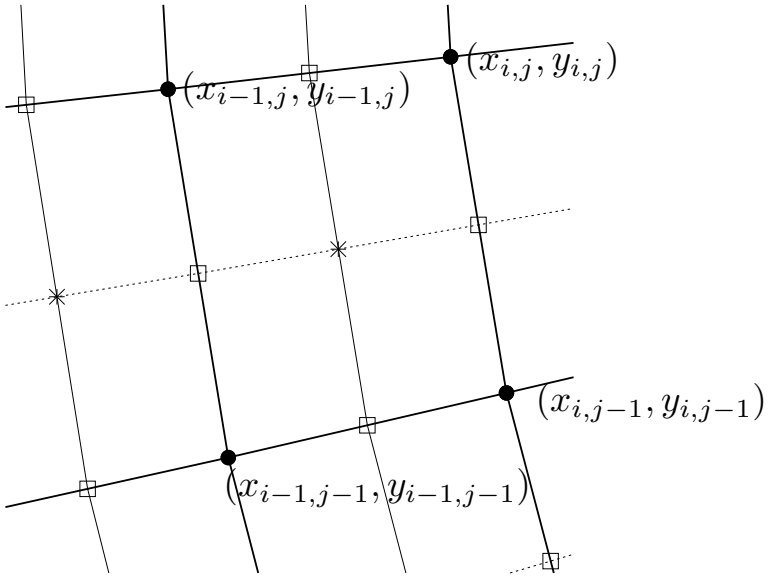

Figure 8: Grid layout and indexing of corner points for curvilinear grids.

$$\begin{array}{lllll} \text{T-points:} & 1 \le i \le i_{\max}, & 1 \le j \le j_{\max}, & 1 \le k \le k_{\max} \\ \text{U-points:} & 0 \le i \le i_{\max}, & 1 \le j \le j_{\max}, & 1 \le k \le k_{\max} \\ \text{V-points:} & 1 \le i \le i_{\max}, & 0 \le j \le j_{\max}, & 1 \le k \le k_{\max} \\ \text{W-points:} & 1 \le i \le i_{\max}, & 1 \le j \le j_{\max}, & 0 \le k \le k_{\max} \end{array} \tag{46}$$

On the T-points, all tracers such as temperature T, salinity S, the general tracers c^i and the density are located. All turbulent quantities such as eddy viscosity ν_t and eddy diffusivity ν_t' are located on the W-points.

For curvilinear grids, several arrays for spatial increments Δx and Δy have to be defined:

$$
\begin{aligned}
\Delta x^c_{i,j} &= \left|\left|\tfrac{1}{2}(X_{i,j-1} + X_{i,j} - X_{i-1,j-1} - X_{i-1,j})\right|\right| \\
\Delta x^u_{i,j} &= \left|\left|\tfrac{1}{4}(X_{i+1,j-1} + X_{i+1,j} - X_{i-1,j-1} - X_{i-1,j})\right|\right| \\
\Delta x^v_{i,j} &= ||X_{i,j} - X_{i-1,j}|| \\
\Delta x^+_{i,j} &= \left|\left|\tfrac{1}{2}(X_{i+1,j} - X_{i-1,j})\right|\right| \\
\Delta y^c_{i,j} &= \left|\left|\tfrac{1}{2}(X_{i-1,j} + X_{i,j} - X_{i-1,j-1} - X_{i,j-1})\right|\right| \\
\Delta y^u_{i,j} &= ||X_{i,j} - X_{i,j-1}|| \\
\Delta y^v_{i,j} &= \left|\left|\tfrac{1}{4}(X_{i-1,j+1} + X_{i,j+1} - X_{i-1,j-1} - X_{i,j-1})\right|\right| \\
\Delta y^+_{i,j} &= \left|\left|\tfrac{1}{2}(X_{i,j+1} - X_{i,j-1})\right|\right|
\end{aligned}
\tag{47}
$$

where $||X_{i,j} - X_{i-1,j}|| = \left((x_{i,j} - x_{i-1,j})^2 + (y_{i,j} - y_{i-1,j})^2\right)^{1/2}$. The superscripts $c, u, v, +$ in (47) indicate whether a Δx or Δy is centrered at a T-, U-, V-, or X-point, respectively. For the locations of the corner points $X_{i,j} = (x_{i,j}, y_{i,j})$, see figure 8.

5.3 Lateral boundary conditions

Usually, a land mask is defined on the horizontal numerical grid. This mask is denoted by a^z for T-points, a^u for U-points and a^v for V-points with a^z, a^u, and a^v being integer fields. A T-point is either a land point ($a^z = 0$) or a water point ($a^z > 0$). All U- and V-points surrounding a land point are defined as closed boundary and masked out: $a^u = 0$ and $a^v = 0$. The velocities on such closed boundaries are always set to 0.

Open boundaries are defined by $a^z > 1$ for T-points. Forced boundary points are marked by $a^z = 2$ and passive boundary points by $a^z = 3$. All other T-points are characterised by $a^z = 1$. For velocity points, three different types are defined at the open boundaries. U-points are classified by $a^u = 3$ if both the T-points east and west are open boundary points and by $a^u = 2$ if one adjacent T-point is an open boundary point and the other an open water point with $a^z = 1$. The same is carried out for V-points: They are classified by $a^v = 3$ if both the T-points south and north are open boundary points and by $a^v = 2$ if one adjacent T-point is an open boundary point and the other an open water point with $a^z = 1$. U-points which are adjacent to T-points with $a^z = 2$ and which are not denoted by $a^u = 2$ or $a^u = 3$ are the external U-points and are denoted by $a^u = 4$. The same holds for V-points: Those which are adjacent to T-points with $a^z = 2$ and which are not denoted by $a^v = 2$ or $a^v = 3$ are the external V-points and are denoted by $a^v = 4$. For a simple example of grid point classification, see figure 9.

When the barotropic boundary forcing is carried out by means of prescribed surface elevations only, then the surface elevation ζ is prescribed in all T-points with $a^z = 2$. For passive boundary conditions ($a^z = 3$), where the curvature of the surface elevation is zero normal to the boundary, the surface slope is simply extrapolated to the boundary points. For a boundary point (i, j) at the western boundary this results e.g. in the following calculation for the boundary point:

$$
\zeta_{i,j} = \zeta_{i+1,j} + (\zeta_{i+1,j} - \zeta_{i+2,j}) = 2\zeta_{i+1,j} - \zeta_{i+2,j}. \tag{48}
$$

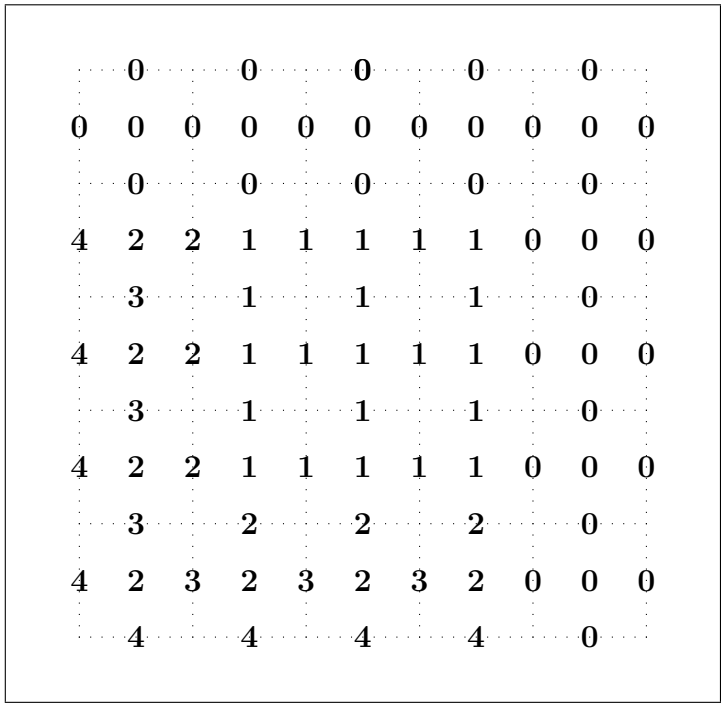

Figure 9: Classification of grid points for a simple 5×5 configuration ($i_{\max} = j_{\max} = 5$). On the locations for T-, U- and V-points, the values of a^z, a^u, and a^v, respectively, are written. The northern and eastern boundaries are closed and the western and southern boundaries are open and forced.

5.4 Bed friction

As already mentioned earlier in section 3.1.3, caution is needed when discretising the bottom boundary conditions for momentum, (8). They are an example for a physical condition which has to be modified for the numerical discretisation, since the discrete velocity point nearest to the bottom is half a grid box away from the point where the boundary condition is defined. Furthermore, due to the logarithmic law, high velocity gradients are typical near the bed. Simply setting the discrete bottom velocity to zero, would therefore lead to large discretisation errors. Instead, a flux condition using bottom stresses is derived from the law of the wall.

For the determination of the normalised bottom stresses

$$\frac{\tau_b^x}{\rho_0} = u_*^{bx} u_*^b, \tag{49}$$

$$\frac{\tau_b^y}{\rho_0} = u_*^{by} u_*^b \tag{50}$$

with the friction velocities $u_*^b = \sqrt{\tau_b/\rho_0}$ with $\tau_b = \sqrt{(\tau_b^x)^2 + (\tau_b^y)^2}$, assumptions about the structure of velocity inside the discrete bottom layer have to be made. We use here the logarithmic profile

$$\frac{u(z')}{u_*} = \frac{1}{\kappa} \ln\left(\frac{z' + z_0^b}{z_0^b}\right), \tag{51}$$

with the bottom roughness length z_0^b, the von Kármán constant $\kappa = 0.4$ and the distance from the bed, z'. Therefore, estimates for the velocities in the centre of the bottom layer can be achieved by:

$$u_b = \frac{u_*^{bx}}{\kappa} \ln\left(\frac{0.5h_1 + z_0^b}{z_0^b}\right), \tag{52}$$

$$v_b = \frac{u_*^{by}}{\kappa} \ln\left(\frac{0.5h_1 + z_0^b}{z_0^b}\right). \tag{53}$$

For $h_1 \to 0$, the original Dirichlet-type no-slip boundary conditions (8) are retained. Another possibility would be to specify the bottom velocities u_b and v_b such that they are equal to the layer-averaged log-law velocities (see *Baumert and Radach* (1992)). The calculation of this is however slightly more time consuming and does not lead to a higher accuracy.

5.5 Drying and flooding

The main requirement for drying and flooding is that the vertically integrated fluxes U and V are controlled such that at no point a negative water depth occurs. It is clear that parts of the physics which play an important role in very shallow water of a few centimetres depth like non-hydrostatic effects are not included in the equations. However, the model is designed in a way that the control of U and V in very shallow water is mainly motivated by the physics included in the equations rather than by defining complex drying and flooding algorithms. It is assumed that the major process in this situation is a balance between pressure gradient and bottom friction. Therefore, in the case of very shallow water, all other terms are multiplied with the non-dimensional factor α which approaches zero when a minimum water depth is reached.

By using formulation (71) for calculating the bottom drag coefficient R, it is guaranteed that R is exponentially growing if the water depth approaches very small values. This slows the flow down

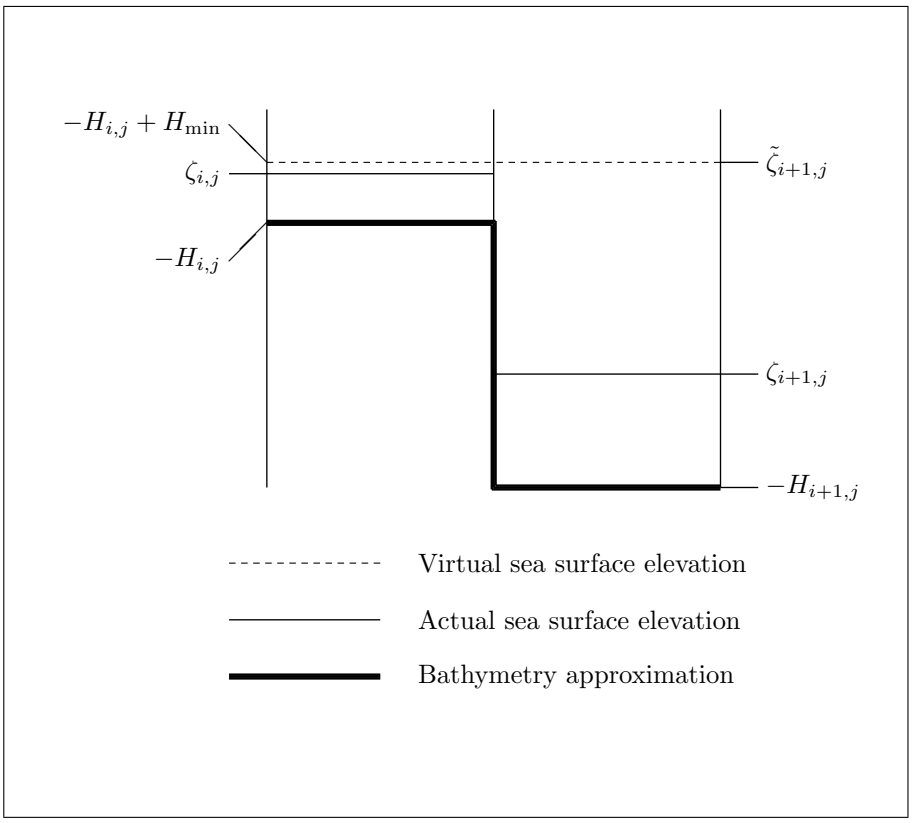

Figure 10: Sketch explaining the principle of pressure gradient minimisation during drying and flooding over sloping bathymetry.

when the water depth in a velocity point is sinking and also allows for flooding without further manipulation.

In this context, one important question is how to calculated the depth in the velocity points, H^u and H^v, since this determines how shallow the water in the velocity points may become on sloping beaches. In ocean models, usually, the depth in the velocity points is calculated as the mean of depths in adjacent elevation points (T-points):

$$H^u_{i,j} = \frac{1}{2}\left(H_{i,j} + H_{i+1,j}\right), \qquad H^v_{i,j} = \frac{1}{2}\left(H_{i,j} + H_{i,j+1}\right). \tag{54}$$

Other models which deal with drying and flooding such as the models of *Duwe* (1988) and *Casulli and Cattani* (1994) use the minimum of the adjacent depths in the T-points:

$$H^u_{i,j} = \min\{H_{i,j}, H_{i+1,j}\}, \qquad H^v_{i,j} = \min\{H_{i,j}, H_{i,j+1}\}. \tag{55}$$

This guarantees that all depths in the velocity points around a T-point are not deeper than the depth in the T-point. Thus, when the T-point depth is approaching the minimum depth, then all depths in the velocity points are also small and the friction coefficient correspondingly large.

Each of the methods has however drawbacks: When the mean is taken as in (54), the risk of negative water depths is relatively big, and thus higher values of $D_{\min}$ have to be chosen. When the minimum is taken, large mud-flats might need unrealistically long times for drying since all the water volume has to flow through relatively shallow velocity boxes. Also, velocities in these shallow boxes tend to be relatively high in order to provide sufficient transports. This might lead to numerical instabilities.

Therefore, GETM has both options, (54) and (55) and the addition of various other options such as depth depending weighting of the averaging can easily be added. These options are controlled by the GETM variable `vel_depth_method`, see section 6.1.9 (subroutine `uv_depths`) documented on page 44.

If a pressure point is dry (i.e. its bathymetry value is higher than a neighbouring sea surface elevation), the pressure gradient would be unnaturally high with the consequence of unwanted flow acceleration. Therefore this pressure gradient will be manipulated such that (only for the pressure gradient calculation) a virtual sea surface elevation $\tilde{\zeta}$ is assumed (see figure 10). In the situation shown in figure 10, the left pressure point is dry, and the sea surface elevation there is for numerical reasons even slightly below the critical value $-H_{i,j} + H_{\min}$. In order not to let more water flow out of the left cell, the pressure gradient between the two boxes shown is calculated with a manipulated sea surface elevation on the right, $\tilde{\zeta}_{i+1,j}$.

See also *Burchard et al.* (2004) for a description of drying and flooding numerics in GETM.

6 Introduction to the calculation domain

This module handles all tasks related to the definition of the computational domain - except reading in variables from file. The required information depends on the *grid_type* and also on the complexity of the model simulation to be done.
The mandatory varible *grid_type* read from the file containing the bathymetry and coordinate information (presently only NetCDF is supported) is guiding subsequent tasks. *grid_type* can take the following values:

1: equi-distant plane grid - dx, dy are constant - but not necessarily equal

2: equi-distant spherical grid - $dlon$, $dlat$ are constant - and again not necessarily equal

3: curvilinear grid in the plane - dx, dy are both functions of (i,j). The grid must be orthogonal

For all values of *grid_type* the bathymetry given on the T-points (see the GETM manual for definition) must be given.

Based on the value of *grid_type* the following additional variables are required:

1: proper monotone coordinate informtion in the xy-plane with equidistant spacing. The name of the coordinate variables are *xcord* and *ycord*.

2: proper monotone coordinate informtion on the sphere with equidistant spacing in longitude and latitude. The names of the coordinate variables are *xcord* and *ycord*.

3: position in the plane of the grid-vertices. These are called X-points in GETM. The names of these two variables are xx and yx.

In addition to the above required grid information the following information is necessary for specific model configurations:

A: *latu* and *latv*
If *f_plane* is false information about the latitude of U- and V-points are required for calculating the Coriolis term correctly. For $grid\_type = 1$ *latu* and *latv* are calculated based on an additional field *latc* i.e. the latitude of the T-points. For $grid\_type = 3$ $latx$ i.e. the latitude of the X-points will have to be provided in order to calculate *latu* and *latv*.

B: *lonc*, *latc* and *convc*
The longitude, latitude positions of the T-points are required when using forcing from a NWP-model. *lonc* and *latc* are used to do spatial interpolation from the meteo-grid to the GETM model and *convc* is the rotation of the local grid from true north.

In addition to the information above a few files are optionally read in *init_domain*(). Information about open boundaries, modifications to the bathymetry and the calculation masks are are done via simple ASCII files.

6.1 Fortran: Module Interface domain - sets up the calculation domain. (Source File: domain.F90)

INTERFACE:

```
module domain
```

DESCRIPTION:

This module provides all variables related to the bathymetry and model grid. The public subroutine *init_domain*() is called once and upon successful completion the bathymetry has been read and optionally modified, the calculation masks have been setup and all grid related variables have been initialised.
The *domain*-module depends on another module doing the actual reading of variables from files. This is provided through the generic subroutine *read_topo_file*. This subroutine takes two parameters - 1) a fileformat and 2) a filename. Adding a new input file format is thus straight forward and can be done without any changes to *domain*. Public variables defined in this module is used through out the code. **USES:**

```
use exceptions
use halo_zones,     only: update_2d_halo,wait_halo
use halo_zones,     only: H_TAG,U_TAG,V_TAG
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
integer                              :: bathy_format   = NETCDF

integer                              :: grid_type      = 1
integer                              :: vert_cord      = 1
integer                              :: il=-1,ih=-1,jl=-1,jh=-1
global index range
integer                              :: ilg=-1,ihg=-1,jlg=-1,jhg=-1
local index range
integer                              :: ill=-1,ihl=-1,jll=-1,jhl=-1

logical                              :: have_lonlat    = .true.
logical                              :: have_xy        = .true.

REALTYPE                             :: rearth

REALTYPE                             :: maxdepth       = -1.
REALTYPE                             :: ddu            = -_ONE_
REALTYPE                             :: ddl            = -_ONE_
REALTYPE                             :: d_gamma        = 20.
logical                              :: gamma_surf     = .true.
REALTYPE, allocatable, dimension(:) :: ga

integer                              :: NWB=-1,NNB=-1,NEB=-1,NSB=-1,NOB
integer                              :: calc_points
logical                              :: openbdy        = .false.

REALTYPE                             :: Hland=-10.0
REALTYPE                             :: min_depth,crit_depth

REALTYPE                             :: longitude      = _ZERO_
```

```
   REALTYPE                              :: latitude       = _ZERO_
   logical                               :: f_plane        = .true.
   logical                               :: check_cfl      = .true.

#ifdef STATIC
#include "static_domain.h"
#else
#include "dynamic_declarations_domain.h"
#endif
   integer                               :: nsbv

   integer                               :: ioff=0,joff=0
   integer, dimension(:), allocatable    :: bdy_2d_type
   integer, dimension(:), allocatable    :: bdy_3d_type
   integer, dimension(:), allocatable    :: wi,wfj,wlj
   integer, dimension(:), allocatable    :: nj,nfi,nli
   integer, dimension(:), allocatable    :: ei,efj,elj
   integer, dimension(:), allocatable    :: sj,sfi,sli
   integer, allocatable                  :: bdy_index(:),bdy_map(:,:)
   logical                               :: have_boundaries=.false.

   character(len=64)                     :: bdy_2d_desc(5)
   logical                               :: need_2d_bdy_elev = .false.
   logical                               :: need_2d_bdy_u    = .false.
   logical                               :: need_2d_bdy_v    = .false.

   REALTYPE                              :: cori= _ZERO_

   method for specifying bottom roughness (0=const, 1=from topo.nc)
   integer                               :: z0_method=0
   REALTYPE                              :: z0_const=0.01d0
```

DEFINED PARAMETERS:

```
   integer,           parameter          :: INNER          = 1
   REALTYPE, private, parameter          :: pi             = 3.141592654
   REALTYPE, private, parameter          :: deg2rad        = pi/180.
   REALTYPE, private, parameter          :: omega          = 2.*pi/86164.
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   REALTYPE, parameter                    :: rearth_default = 6378815.
```

6.1.1 init_domain() - initialise the computational domain

INTERFACE:

```
subroutine init_domain(input_dir,runtype)
IMPLICIT NONE
```

DESCRIPTION:

This routine is responsible for setting up the bathymetry and the grid information.
The following steps are done in *init_domain*():

1: partition of the calculation domain - important for parallel runs

2: reading bathymetry and grid information through the generic subroutine *read_topo_file*

3: optionally set minimum depth in regions

4: optionally adjust the depth in regions

5: optionally adjust the depth in regions

6: calculate the mask for T-points

7: optionally adjust the mask in regions

8: read boundary information and adjust masks

9: calculate masks for U-, V- and X-points

10: calculate additional grid-information - like *latu* and *latv*

11: calculate metrics - i.e. all necessary grid-spacings

12: calculate Coriolis parameter - can be constant or spatially varying

INPUT/OUTPUT PARAMETERS:

```
character(len=*)                    :: input_dir
integer, intent(in)                 :: runtype
```

REVISION HISTORY:

LOCAL VARIABLES:

```
integer                   :: rc
integer                   :: np,sz
integer                   :: i,j,n
integer                   :: kdum
character(len=PATH_MAX)   :: bathymetry               = 'topo.nc'
integer                   :: vel_depth_method=0
character(len=PATH_MAX)   :: bdyinfofile              = 'bdyinfo.dat'
character(len=PATH_MAX)   :: min_depth_file           = 'minimum_depth.dat'
character(len=PATH_MAX)   :: bathymetry_adjust_file   = 'bathymetry.adjust'
character(len=PATH_MAX)   :: mask_adjust_file         = 'mask.adjust'
namelist /domain/ &
          vert_cord,maxdepth,                                  &
          bathy_format,bathymetry,vel_depth_method,            &
          longitude,latitude,f_plane,openbdy,bdyinfofile,      &
          crit_depth,min_depth,kdum,ddu,ddl,                   &
          d_gamma,gamma_surf,il,ih,jl,jh,z0_method,z0_const,&
          check_cfl
```

6.1.2 x2uvc() - interpolate grid-points

INTERFACE:

```
subroutine x2uvc()
IMPLICIT NONE
```

DESCRIPTION:

This routine interpolates (latx,lonx), (xx,yx), and convx to the u-points, v-points, and the central T-points. The data at the T-points are only updated from values of the X-points if the logical flags `updateXYC`, `updateXYC`, and `updateXYC` are `.true.`. This is not necessary if data at the T-points have been read from the topo input file. **REVISION HISTORY:**

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
integer                 :: i,j,n
REALTYPE                :: x
```

6.1.3 metric() - calculate metric coefficients

INTERFACE:

```
subroutine metric()
IMPLICIT NONE
```

DESCRIPTION:

Computes the grid increments and areas related to the metric coefficients. **REVISION HISTORY:**

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
integer                   :: i,j
```

6.1.4 set_min_depth() - set the minimum depth in regions

INTERFACE:

```
subroutine set_min_depth(fn)
IMPLICIT NONE
```

DESCRIPTION:

Read region definitions and minimum depth for those regions. Adjust the bathymetry (variable H) accordingly. **INPUT PARAMETERS:**

```
character(len=*), intent(in)        :: fn
```

REVISION HISTORY:

LOCAL VARIABLES:

```
integer                   :: unit = 25 ! kbk
character(len=255)        :: line
integer                   :: iostat
integer                   :: i,j,k=0,n=-1
integer                   :: il,jl,ih,jh
integer                   :: i1,j1
REALTYPE                  :: dmin
```

6.1.5 adjust_bathymetry() - read mask adjustments from file.

INTERFACE:

```
subroutine adjust_bathymetry(fn)
IMPLICIT NONE
```

DESCRIPTION:

Read bathymetry adjustments from file. **INPUT PARAMETERS:**

```
character(len=*), intent(in)        :: fn
```

REVISION HISTORY:

LOCAL VARIABLES:

```
integer                   :: unit = 25 ! kbk
character(len=255)        :: line
integer                   :: iostat
integer                   :: i,j,k=0,n=-1
integer                   :: il,jl,ih,jh
REALTYPE                  :: x
```

6.1.6 adjust_mask() - read mask adjustments from file.

INTERFACE:

```
subroutine adjust_mask(fn)
IMPLICIT NONE
```

DESCRIPTION:

Read mask adjustments from file. The file format allows comments. Comment characters are ! or # - they MUST be in column 1. Lines with white-spaces are skipped. Conversion errors are caught and an error condition occurs. **INPUT PARAMETERS:**

```
character(len=*), intent(in)        :: fn
```

REVISION HISTORY:

LOCAL VARIABLES:

```
integer                   :: unit = 25 ! kbk
character(len=255)        :: line
integer                   :: iostat
integer                   :: i,j,k=0,n=-1
integer                   :: il,jl,ih,jh
```

6.1.7 print_mask() - prints a mask in readable format

INTERFACE:

```
subroutine print_mask(mask)
IMPLICIT NONE
```

DESCRIPTION:

Prints a integer mask in a human readable form. **INPUT PARAMETERS:**

```
integer, intent(in), dimension(E2DFIELD) :: mask
```

REVISION HISTORY:

```
22Apr99   Karsten Bolding & Hans Burchard  Initial code.
```

LOCAL VARIABLES:

```
integer                   :: i,j
```

6.1.8 part_domain() - partition the domain (Source File: part_domain.F90)

INTERFACE:

```
subroutine part_domain()
```

DESCRIPTION:

Set various integers defining the calculation domain. The settings depends on STATIC vs. DYNAMIC compilation and serial vs. parallel model run. **USES:**

```
   use domain, only: iextr,jextr
   use domain, only: imin,imax,jmin,jmax,kmax
   use domain, only: ioff,joff
#ifdef GETM_PARALLEL
   use halo_mpi, only: part_domain_mpi
#endif
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

6.1.9 uv_depths - calculate depths in u and v points.

INTERFACE:

```
subroutine uv_depths(vel_depth_method)
```

DESCRIPTION:

In this routine which is called once during the model initialisation, the bathymetry value in the U- and the V-points are calculated from the bathymetry values in the T-points. The interpolation depends on the value which is given to `vel_depth_method`:

$$H^u_{i,j} = \begin{cases} \frac{1}{2}\left(H_{i,j} + H_{i+1,j}\right), & \text{for } \texttt{vel\_depth\_method} = 0, \\ \min\left\{H_{i,j} + H_{i+1,j}\right\}, & \text{for } \texttt{vel\_depth\_method} = 1, \\ \min\left\{H_{i,j} + H_{i+1,j}\right\}, & \text{for } \texttt{vel\_depth\_method} = 2 \text{ and } \min\{H_{i,j}i, H_{i+1,j}\} < D_{crit} \\ \frac{1}{2}\left(H_{i,j} + H_{i+1,j}\right), & \text{for } \texttt{vel\_depth\_method} = 2 \text{ and } \min\{H_{i,j}, H_{i+1,j}\} \geq D_{crit} \end{cases} \tag{56}$$

The calculation of $H^v_{i,j}$ is done accordingly.
The options 1 and 2 for `vel_depth_method` may help to stabilise calculations when drying and flooding is involved. **USES:**

```
use exceptions
use domain, only: imin,imax,jmin,jmax,az,au,av,H,HU,HV
use getm_timers, only: tic,toc,TIM_UVDEPTHS
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: vel_depth_method
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j
REALTYPE                  :: d_crit=2.0
```

6.1.10 have_bdy - checks whether this node has boundaries. (Source File: have_bdy.F90)

INTERFACE:

```
subroutine have_bdy
```

DESCRIPTION:

This routine which is called in `domain.F90` checks whether the present node has open lateral boundaries. The integer field `bdy_index` is then set accordingly. **USES:**

```
use domain
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: i,j,k,m,n
integer                   :: nbdy
integer                   :: f,l
```

6.1.11 bdy_spec() - defines open boundaries (Source File: bdy_spec.F90)

INTERFACE:

```
subroutine bdy_spec(fn)
```

DESCRIPTION:

Read in the open boundary information from 'fn'. **USES:**

```
use exceptions
use domain, only: NWB,NNB,NEB,NSB,NOB
use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
use domain, only: bdy_index,bdy_map,nsbv
use domain, only: bdy_2d_type,bdy_3d_type
use domain, only: need_2d_bdy_elev,need_2d_bdy_u,need_2d_bdy_v
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fn
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
character(len=255)        :: line
integer                   :: iostat
integer                   :: i,j,k,l
integer                   :: n,rc
integer                   :: type_2d(4,10),type_3d(4,10)
```

6.1.12 print_bdy() - print open boundary info (Source File: print_bdy.F90)

INTERFACE:

```
subroutine print_bdy(header)
```

DESCRIPTION:

Print the open boundary information. This routine is called twice - first time with the global boundary infromation and second time with the local boundary information. In the case of a serial run the info is identical - in the case of a parallel run the open boundary information for a each sub-domain will be printed. **USES:**

```
use domain, only: NWB,NNB,NEB,NSB
use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
use domain, only: bdy_2d_type,bdy_3d_type
use domain, only: bdy_2d_desc
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: header
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: m,n
```

6.1.13 mirror_bdy_2d() - mirrors 2d variables (Source File: mirror_bdy_2d.F90)

INTERFACE:

```
subroutine mirror_bdy_2d(f,tag)
```

DESCRIPTION:

Some variables are mirrored outside the calculation domain in the vicinity of the open boundaries. This is to avoid if statements when calculating e.g. the Coriolis terms and advection. This routines mirrors 2d variables. **USES:**

```
use halo_zones, only : U_TAG,V_TAG,H_TAG
use domain, only: imin,imax,jmin,jmax
use domain, only: az,au,av
use domain, only: NWB,NNB,NEB,NSB
use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: tag
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE, intent(inout)             :: f(E2DFIELD)
```

OUTPUT PARAMETERS:

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: i,j,n
```

6.1.14 mirror_bdy_3d() - mirrors 3d vaiables (Source File: mirror_bdy_3d.F90)

INTERFACE:

```
subroutine mirror_bdy_3d(f,tag)
```

DESCRIPTION:

Some variables are mirrored outside the calculation domain in the vicinity of the open boundaries. This is to avoid if statements when calculating e.g. the Coriolis terms and advection. This routines mirrors 3d variables. **USES:**

```
use halo_zones, only : U_TAG,V_TAG,H_TAG,D_TAG
use domain, only: imin,imax,jmin,jmax,kmax
use domain, only: az,au,av
use domain, only: NWB,NNB,NEB,NSB
use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: tag
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE, intent(inout)             :: f(I3DFIELD)
```

OUTPUT PARAMETERS:

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: i,j,n
```

7 Introduction to 2d module

In the 2D module of GETM the vertically integrated mode is calculated, which is basically the vertically integrated momentum equations and the sea surface elevation equation. For the momentum equations, interaction terms with the baroclinic three-dimentional mode need to be considered. Those terms are here called the slow terms.

7.1 Vertically integrated mode

In order to provide the splitting of the model into an internal and an external mode, the continuity equation and the momentum equations are vertically integrated. The vertical integral of the continuity equation together with the kinematic boundary conditions (6) and (7) gives the sea surface elevation equation:

$$\partial_t \zeta = -\partial_x U - \partial_y V. \tag{57}$$

with

$$U = \int_{-H}^{\zeta} u\, dz, \qquad V = \int_{-H}^{\zeta} v\, dz. \tag{58}$$

Integrating the momentum equations (1) and (2) vertically results in:

$$\begin{aligned} &\partial_t U + \tau_b^x + \alpha\Bigg(\int_{-H}^{\zeta} \left(\partial_x u^2 + \partial_y (uv)\right)\, dz \\ &-\tau_s^x - \int_{-H}^{\zeta} \left(\partial_x \left(2A_h^M \partial_x u\right) - \partial_y \left(A_h^M (\partial_y u + \partial_x v)\right)\right) dz \\ &-fV - \int_{-H}^{\zeta} \int_{z}^{\zeta} \partial_x b\, dz'\, dz \Bigg) = -gD\partial_x \zeta \end{aligned} \tag{59}$$

and

$$\begin{aligned} &\partial_t V + \tau_b^y + \alpha\Bigg(\int_{-H}^{\zeta} \left(\partial_x (uv) + \partial_y v^2\right)\right) dz \\ &-\tau_s^y - \int_{-H}^{\zeta} \left(\partial_y \left(2A_h^M \partial_y v\right) - \partial_x \left(A_h^M (\partial_y u + \partial_x v)\right)\right) dz \\ &+fU - \int_{-H}^{\zeta} \int_{z}^{\zeta} \partial_y b\, dz'\, dz \Bigg) = -gD\partial_y \zeta. \end{aligned} \tag{60}$$

Here, τ_b^x and τ_b^y are bottom stresses. Their calculation is discussed in section 8.13.9. As a first preparation for the mode splitting, these integrals of the momentum equations can be formally rewritten as

$$\partial_t U + \frac{R}{D^2} U\sqrt{U^2+V^2} + S_F^x + \alpha\Bigg(\partial_x\left(\frac{U^2}{D}\right) + \partial_y\left(\frac{UV}{D}\right)$$
$$-\tau_s^x - \partial_x\left(2A_h^M D\partial_x\left(\frac{U}{D}\right)\right) - \partial_y\left(A_h^M D\left(\partial_y\left(\frac{U}{D}\right) + \partial_x\left(\frac{V}{D}\right)\right)\right) \tag{61}$$
$$-fV + S_A^x - S_D^x + S_B^x\Bigg) = -gD\partial_x\zeta$$

and

$$\partial_t V + \frac{R}{D^2} V\sqrt{U^2+V^2} + S_F^y + \alpha\Bigg(\partial_x\frac{UV}{D} + \partial_y\frac{V^2}{D}$$
$$-\tau_s^y - \partial_x\left(A_h^M D\left(\partial_y\left(\frac{U}{D}\right) + \partial_x\left(\frac{V}{D}\right)\right)\right) - \partial_y\left(2A_h^M D\partial_y\left(\frac{V}{D}\right)\right) \tag{62}$$
$$+fU + S_A^y - S_D^y + S_B^y\Bigg) = -gD\partial_y\zeta$$

with the so-called slow terms for bottom friction

$$S_F^x = \tau_b^x - \frac{R}{D^2} U\sqrt{U^2+V^2}, \tag{63}$$

$$S_F^y = \tau_b^y - \frac{R}{D^2} V\sqrt{U^2+V^2}, \tag{64}$$

horizontal advection

$$S_A^x = \int_{-H}^{\zeta}\left(\partial_x u^2 + \partial_y(uv)\right)\,dz - \partial_x\left(\frac{U^2}{D}\right) - \partial_y\left(\frac{UV}{D}\right), \tag{65}$$

$$S_A^y = \int_{-H}^{\zeta}\left(\partial_x(uv) + \partial_y v^2\right)\,dz - \partial_x\left(\frac{UV}{D}\right) - \partial_y\left(fracV^2D\right), \tag{66}$$

horizontal diffusion

$$S_D^x = \int_{-H}^{\zeta}\left(\partial_x\left(2A_h^M\partial_x u\right) - \partial_y\left(A_h^M(\partial_y u + \partial_x v)\right)\right)dz$$
$$-\partial_x\left(2A_h^M D\partial_x\left(\frac{U}{D}\right)\right) - \partial_y\left(A_h^M D\left(\partial_y\left(\frac{U}{D}\right) + \partial_x\left(\frac{V}{D}\right)\right)\right), \tag{67}$$

$$S_D^y = \int_{-H}^{\zeta}\left(\partial_y\left(2A_h^M\partial_y v\right) - \partial_x\left(A_h^M(\partial_y u + \partial_x v)\right)\right)dz$$
$$-\partial_y\left(2A_h^M D\partial_y\left(\frac{V}{D}\right)\right) - \partial_x\left(A_h^M D\left(\partial_y\left(\frac{U}{D}\right) + \partial_x\left(\frac{V}{D}\right)\right)\right), \tag{68}$$

and internal pressure gradients

$$S_B^x = -\int_{-H}^{\zeta}\int_{z}^{\zeta}\partial_x b\,dz'\,dz \tag{69}$$

and

$$S_B^y = -\int_{-H}^{\zeta}\int_{z}^{\zeta} \partial_y b \, dz' \, dz. \tag{70}$$

The drag coefficient R for the external mode is calculated as (this logarithmic dependence of the bottom drag from the water depth and the bottom roughness parameter z_b^0 is discussed in detail by *Burchard and Bolding* (2002)):

$$R = \left(\frac{\kappa}{\ln\left(\frac{\frac{D}{2}+z_0^b}{z_0^b}\right)} \right)^2 . \tag{71}$$

It should be noted that for numerical reasons, an additional explicit damping has been implemented into GETM. This method is based on diffusion of horizontal transports and is described in section 7.4.14 on page 83.

7.2 Fortran: Module Interface m2d - depth integrated hydrodynamical model (2D) (Source File: m2d.F90)

INTERFACE:

```
module m2d
```

DESCRIPTION:

This module contains declarations for all variables related to 2D hydrodynamical calculations. Information about the calculation domain is included from the `domain` module. The module contains public subroutines for initialisation, integration and clean up of the 2D model component. The actual calculation routines are called in `integrate_2d` and are linked in from the library `lib2d.a`. **USES:**

```
   use exceptions
   use time, only: julianday,secondsofday
   use parameters, only: avmmol
   use domain, only: imin,imax,jmin,jmax,az,au,av,H,min_depth
   use domain, only: ilg,ihg,jlg,jhg
   use domain, only: ill,ihl,jll,jhl
   use domain, only: openbdy,have_boundaries,z0_method,z0_const,z0
   use domain, only: check_cfl
   use domain, only: az,ax
 KB   use get_field, only: get_2d_field
   use advection, only: init_advection,print_adv_settings,NOADV
   use halo_zones, only: update_2d_halo,wait_halo,H_TAG
   use variables_2d

   IMPLICIT NONE

   interface

      subroutine uv_advect(U,V,DU,DV)
         use domain, only: imin,imax,jmin,jmax
         IMPLICIT NONE
         REALTYPE,dimension(E2DFIELD),intent(in)        :: U,V
         REALTYPE,dimension(E2DFIELD),target,intent(in) :: DU,DV
      end subroutine uv_advect

      subroutine uv_diffusion(An_method,U,V,D,DU,DV)
         use domain, only: imin,imax,jmin,jmax
         IMPLICIT NONE
         integer,intent(in)                       :: An_method
         REALTYPE,dimension(E2DFIELD),intent(in) :: U,V,D,DU,DV
      end subroutine uv_diffusion

      subroutine uv_diff_2dh(An_method,UEx,VEx,U,V,D,DU,DV,hsd_u,hsd_v)
         use domain, only: imin,imax,jmin,jmax
         IMPLICIT NONE
         integer,intent(in)                                  :: An_method
         REALTYPE,dimension(E2DFIELD),intent(in),optional  :: U,V,D,DU,DV
         REALTYPE,dimension(E2DFIELD),intent(inout)        :: UEx,VEx
         REALTYPE,dimension(E2DFIELD),intent(out),optional :: hsd_u,hsd_v
      end subroutine uv_diff_2dh
```

```
Temporary interface (should be read from module):
    subroutine get_2d_field(fn,varname,il,ih,jl,jh,break_on_missing,f)
       character(len=*),intent(in)   :: fn,varname
       integer, intent(in)           :: il,ih,jl,jh
       logical, intent(in)           :: break_on_missing
       REALTYPE, intent(out)         :: f(:,:)
    end subroutine get_2d_field
 end interface
```

PUBLIC DATA MEMBERS:

```
REALTYPE                  :: dtm
integer                   :: vel2d_adv_split=0
integer                   :: vel2d_adv_hor=1
REALTYPE                  :: Am=-_ONE_
method for specifying horizontal numerical diffusion coefficient
   (0=const, 1=from named nc-file)
integer                   :: An_method=0
REALTYPE                  :: An_const=-_ONE_
character(LEN = PATH_MAX) :: An_file

integer                   :: MM=1,residual=-1
integer                   :: sealevel_check=0
logical                   :: bdy2d=.false.
integer                   :: bdyfmt_2d,bdytype,bdy2d_ramp=-1
character(len=PATH_MAX)   :: bdyfile_2d
REAL_4B                   :: bdy_data(1500)
REAL_4B                   :: bdy_data_u(1500)
REAL_4B                   :: bdy_data_v(1500)
REAL_4B, allocatable      :: bdy_times(:)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: num_neighbors
REALTYPE                  :: An_sum
```

7.2.1 init_2d - initialise 2D related stuff.

INTERFACE:

```
subroutine init_2d(runtype,timestep,hotstart)
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: runtype
REALTYPE, intent(in)               :: timestep
logical, intent(in)                :: hotstart
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

Here, the `m2d` namelist is read from `getm.inp`, and the check for the fulfilment of the CFL criterium for shallow water theory `cfl_check` is called. A major part of this subroutine deals then with the setting of local bathymetry values and initial surface elevations in u- and v-points, also by calls to the subroutines `uv_depths` and `depth_update`. **LOCAL VARIABLES:**

```
integer                   :: rc
integer                   :: i,j
integer                   :: elev_method=1
REALTYPE                  :: elev_const=_ZERO_
character(LEN = PATH_MAX) :: elev_file='elev.nc'
namelist /m2d/ &
       elev_method,elev_const,elev_file,                    &
       MM,vel2d_adv_split,vel2d_adv_hor,                    &
       Am,An_method,An_const,An_file,residual,              &
       sealevel_check,bdy2d,bdyfmt_2d,bdy2d_ramp,bdyfile_2d
```

7.2.2 postinit_2d - re-initialise some 2D after hotstart read.

INTERFACE:

```
subroutine postinit_2d(runtype,timestep,hotstart)
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: runtype
REALTYPE, intent(in)               :: timestep
logical, intent(in)                :: hotstart
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

This routine provides possibility to reset/initialize 2D variables to ensure that velocities are correctly set on land cells after read of a hotstart file. **LOCAL VARIABLES:**

```
integer                    :: i,j, ischange
```

7.2.3 integrate_2d - sequence of calls to do 2D model integration

INTERFACE:

```
subroutine integrate_2d(runtype,loop,tausx,tausy,airp)
use getm_timers, only: tic, toc, TIM_INTEGR2D
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: runtype,loop
REALTYPE, intent(in)                :: tausx(E2DFIELD)
REALTYPE, intent(in)                :: tausy(E2DFIELD)
REALTYPE, intent(in)                :: airp(E2DFIELD)
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

Here, all 2D related subroutines are called. The major calls and their meaning are:

`call update_2d_bdy`	read in new lateral boundary conditions
`call bottom_friction`	update bottom friction
`call uv_advect`	calculate 2D advection terms
`call uv_diffusion`	calculate 2D diffusion terms
`call momentum`	iterate 2D momemtum equations
`call sealevel`	update sea surface elevation
`call depth_update`	update water depths
`call do_residual`	calculate intermdediate values for residual currents

It should be noted that some of these calls may be excluded for certain compiler options set in the `Makefile` of the application. **LOCAL VARIABLES:**

7.2.4 clean_2d - cleanup after 2D run.

INTERFACE:

```
subroutine clean_2d()
IMPLICIT NONE
```

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

This routine executes a final call to `do_residual` where the residual current calculations are finished. **LOCAL VARIABLES:**

7.3 Fortran: Module Interface variables_2d - global variables for 2D model (Source File: variables_2d.F90)

INTERFACE:

```
module variables_2d
```

DESCRIPTION:

This modules contains declarations for all variables related to 2D hydrodynamical calculations. Information about the calculation domain is included from the `domain` module. The module contains public subroutines to initialise and cleanup. Depending whether the compiler option STATIC is set or not, memory for 2D variables is statically or dynamically allocated, see `PUBLIC DATA MEMBERS`. **USES:**

```
use domain, only: imin,imax,jmin,jmax
use field_manager
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   integer, parameter                  :: rk = kind(_ONE_)
#ifdef STATIC
#include "static_2d.h"
#else
#include "dynamic_declarations_2d.h"
#endif
   integer                             :: size2d_field
   integer                             :: mem2d
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.3.1 init_variables_2d - initialise 2D related stuff.

INTERFACE:

```
subroutine init_variables_2d(runtype)
IMPLICIT NONE
```

DESCRIPTION:

Allocates memory (unless `STATIC` is set) for 2D related fields, by an include statement. Furthermore all public 2D variables are initialised to zero. Those are listed in table 1 on page 61.

`z`	sea surface elevation in T-point	[m]
`U`	x component of transport in U-point	[m^2s^{-1}]
`DU`	water depth in U-point	[m]
`fU`	Coriolis term for V-equation in V-point	[m^2s^{-2}]
`SlUx`	slow term for U-equation in U-point	[m^2s^{-2}]
`Slru`	slow bottom friction for U-equation in U-point	[m^2s^{-2}]
`V`	y component of transport in V-point	[m^2s^{-1}]
`DV`	water depth in V-point	[m]
`fV`	Coriolis term for U-equation in U-point	[m^2s^{-2}]
`SlVx`	slow term for V-equation in V-point	[m^2s^{-2}]
`Slrv`	slow bottom friction for V-equation in V-point	[m^2s^{-2}]
`Uint`	x-component of mean transport in U-point	[m^2s^{-1}]
`Vint`	y-component of mean transport in V-point	[m^2s^{-1}]
`UEx`	sum of explicit terms for for U-equation in U-point	[m^2s^{-2}]
`VEx`	sum of explicit terms for for V-equation in V-point	[m^2s^{-2}]
`ru`	bottom friction for U-equation in U-point	[m^2s^{-2}]
`rv`	bottom friction for V-equation in V-point	[m^2s^{-2}]
`res_du`	residual depth in U-point	[m]
`res_u`	x-component of residual transport in U-point	[m^2s^{-1}]
`res_dv`	residual depth in V-point	[m]
`res_v`	y-component of residual transport in V-point	[m^2s^{-1}]

Table 1: Public 2D variables.

INPUT PARAMETERS:

```
integer, intent(in)                 :: runtype
```

LOCAL VARIABLES:

```
integer                     :: rc
```

7.3.2 register_2d_variables() - register GETM variables. (Source File: variables_2d.F90)

INTERFACE:

```
subroutine register_2d_variables(fm)
```

DESCRIPTION:

USES:

```
 use variables_2d
IMPLICIT NONE
```

INPUT PARAMETERS:

```
type (type_field_manager) :: fm
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Jorn Bruggeman
```

LOCAL VARIABLES:

```
logical :: used
```

7.3.3 clean_variables_2d - cleanup after 2D run.

INTERFACE:

```
subroutine clean_variables_2d()
IMPLICIT NONE
```

DESCRIPTION:

This routine is currently empty. **LOCAL VARIABLES:**

7.4 Fortran: Module Interface 2D advection (Source File: advection.F90)

INTERFACE:

```
module advection
```

DESCRIPTION:

This module does lateral advection of scalars. It follows the same convention as the other modules in 'getm'. The module is initialised by calling 'init_advection()'. In the time-loop 'do_advection()' is called. 'do_advection' is a wrapper routine which - dependent on the actual advection scheme chosen - makes calls to the appropriate subroutines, which may be done as one-step or multiple-step schemes. The actual subroutines are coded in external FORTRAN files. New advection schemes are easily implemented - at least from a program point of view - since only this module needs to be changed. Additional work arrays can easily be added following the stencil given below. To add a new advection scheme three things must be done:

1. define a unique constant to identify the scheme (see e.g. `UPSTREAM` and `TVD`)
2. adopt the `select case` in `do_advection` and
3. write the actual subroutine.

USES:

```
   use domain, only: imin,imax,jmin,jmax
   IMPLICIT NONE

   private
```

PUBLIC DATA MEMBERS:

```
   public init_advection,do_advection,print_adv_settings
   public adv_split_u,adv_split_v,adv_upstream_2dh,adv_arakawa_j7_2dh,adv_fct_2dh
   public adv_interfacial_reconstruction

   type, public :: t_adv_grid
      logical,dimension(:,:),pointer,contiguous  :: mask_uflux,mask_vflux,mask_xflux
      logical,dimension(:,:),pointer,contiguous  :: mask_uupdate,mask_vupdate
      logical,dimension(:,:),pointer,contiguous  :: mask_finalise
      integer,dimension(:,:),pointer,contiguous  :: az
#if defined(SPHERICAL) || defined(CURVILINEAR)
      REALTYPE,dimension(:,:),pointer,contiguous :: dxu,dyu,dxv,dyv,arcd1
#endif
   end type t_adv_grid

   type(t_adv_grid),public,target :: adv_gridH,adv_gridU,adv_gridV

   integer,public,parameter           :: NOSPLIT=0,FULLSPLIT=1,HALFSPLIT=2
   character(len=64),public,parameter :: adv_splits(0:2) = &
                  (/"no split: one 2D uv step          ",  &
                    "full step splitting: u + v        ",  &
                    "half step splitting: u/2 + v + u/2"/)
   integer,public,parameter           :: NOADV=0,UPSTREAM=1,UPSTREAM_2DH=2
   integer,public,parameter           :: P2=3,SUPERBEE=4,MUSCL=5,P2_PDM=6
   integer,public,parameter           :: J7=7,FCT=8,P2_2DH=9
   character(len=64),public,parameter :: adv_schemes(0:9) = &
```

```
      (/"advection disabled                              ",  &
        "upstream advection (first-order, monotone)      ",  &
        "2DH-upstream advection with forced monotonicity",  &
        "P2 advection (third-order, non-monotone)        ",  &
        "TVD-Superbee advection (second-order, monotone)",  &
        "TVD-MUSCL advection (second-order, monotone)    ",  &
        "TVD-P2-PDM advection (third-order, monotone)    ",  &
        "2DH-J7 advection (Arakawa and Lamb, 1977)       ",  &
        "2DH-FCT advection                               ",  &
        "2DH-P2 advection                                "/)
```

LOCAL VARIABLES:

```
#ifdef STATIC
   logical,dimension(E2DFIELD),target         :: mask_updateH
   logical,dimension(E2DFIELD),target         :: mask_uflux,mask_vflux,mask_xflux
   logical,dimension(E2DFIELD),target         :: mask_uupdateU,mask_vupdateV
#else
   logical,dimension(:,:),allocatable,target  :: mask_updateH
   logical,dimension(:,:),allocatable,target  :: mask_uflux,mask_vflux,mask_xflux
   logical,dimension(:,:),allocatable,target  :: mask_uupdateU,mask_vupdateV
#endif
```

REVISION HISTORY:

```
   Original author(s): Knut Klingbeil
```

7.4.1 init_advection

INTERFACE:

```
   subroutine init_advection()
```

DESCRIPTION:

Allocates memory and sets up masks and lateral grid increments. **USES:**

```
   use domain, only: az,au,av,ax
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxc,dyc,arcd1,dxu,dyu,arud1,dxv,dyv,arvd1,dxx,dyx
#endif
   IMPLICIT NONE
```

LOCAL VARIABLES:

```
   integer :: rc
```

7.4.2 do_advection - 2D advection schemes

INTERFACE:

```
subroutine do_advection(dt,f,U,V,DU,DV,Do,Dn,split,scheme,AH,tag, &
                        Dires,advres)
```

DESCRIPTION:

Laterally advects a 2D quantity. The location of the quantity on the grid (either T-, U- or V-points) must be specified by the argument `tag`. The transports through the interfaces of the corresponding Finite-Volumes and their different height information (all relative to the given quantity) must be provided as well. Depending on `split` and `scheme` several fractional steps (Strang splitting) with different options for the calculation of the interfacial fluxes are carried out.
The options for `split` are:

`split = NOSPLIT:`	no split (one 2D uv step)
`split = FULLSPLIT:`	full step splitting (u + v)
`split = HALFSPLIT:`	half step splitting (u/2 + v + u/2)

The options for `scheme` are:

`scheme = NOADV:`	advection disabled
`scheme = UPSTREAM:`	first-order upstream (monotone)
`scheme = UPSTREAM_2DH:`	2DH upstream with forced monotonicity
`scheme = P2:`	third-order polynomial (non-monotone)
`scheme = SUPERBEE:`	second-order TVD (monotone)
`scheme = MUSCL:`	second-order TVD (monotone)
`scheme = P2_PDM:`	third-order ULTIMATE-QUICKEST (monotone)
`scheme = J7:`	2DH Arakawa J7
`scheme = FCT:`	2DH FCT with forced monotonicity
`scheme = P2_2DH:`	2DH P2 with forced monotonicity

With the compiler option `SLICE_MODEL`, the advection in meridional direction is not executed.
USES:

```
use halo_zones, only: update_2d_halo,wait_halo,D_TAG,H_TAG,U_TAG,V_TAG
use getm_timers, only: tic,toc,TIM_ADV,TIM_ADVH
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in)                                 :: dt,AH
REALTYPE,dimension(E2DFIELD),intent(in)             :: U,V,Do,Dn,DU,DV
integer,intent(in)                                  :: split,scheme,tag
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE,dimension(E2DFIELD),intent(inout)          :: f
```

OUTPUT PARAMETERS:

```
REALTYPE,dimension(E2DFIELD),target,intent(out),optional :: Dires,advres
```

LOCAL VARIABLES:

```
type(t_adv_grid),pointer                    :: adv_grid
REALTYPE,dimension(E2DFIELD),target         :: fi,Di,adv
REALTYPE,dimension(:,:),pointer,contiguous :: p_Di,p_adv
integer                                     :: i,j
```

7.4.3 print_adv_settings

INTERFACE:

```
subroutine print_adv_settings(split,scheme,AH)
```

DESCRIPTION:

Checks and prints out settings for 2D advection. **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,intent(in)  :: split,scheme
REALTYPE,intent(in) :: AH
```

LOCAL VARIABLES:

7.4.4 adv_interfacial_reconstruction -

INTERFACE:

```
REALTYPE function adv_interfacial_reconstruction(scheme,cfl,fuu,fu,fd)
```

DESCRIPTION:

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,intent(in)  :: scheme
REALTYPE,intent(in) :: cfl,fuu,fu,fd
```

LOCAL VARIABLES:

```
REALTYPE           :: ratio,limiter,x,deltaf,deltafu
REALTYPE,parameter :: one6th=_ONE_/6
```

REVISION HISTORY:

```
Original author(s): Knut Klingbeil
```

7.4.5 adv_interfacial_reconstruction_p2 -

INTERFACE:

```
REALTYPE function adv_interfacial_reconstruction_p2(cfl,fu,deltafu,deltaf)
```

DESCRIPTION:

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in) :: cfl,fu,deltafu,deltaf
```

LOCAL VARIABLES:

```
REALTYPE           :: x
REALTYPE,parameter :: one6th=_ONE_/6
```

REVISION HISTORY:

```
Original author(s): Knut Klingbeil
```

7.4.6 test_pointer_remap -

INTERFACE:

```
logical function test_pointer_remap()
```

DESCRIPTION:

Tests the support of pointer remapping. **USES:**

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
REALTYPE,dimension(3,2),target  :: t2d
REALTYPE,dimension(:,:),pointer :: p2d
```

7.4.7 adv_split_u - zonal advection of 2D quantities

INTERFACE:

```
   subroutine adv_split_u(dt,f,fi,Di,adv,U,DU,   &
#if defined(SPHERICAL) || defined(CURVILINEAR)
                          dxu,dyu,arcd1,         &
#endif
                          splitfac,scheme,AH,    &
                          mask_flux,mask_update)
   Note (KK): Keep in sync with interface in advection.F90
```

DESCRIPTION:

Executes an advection step in zonal direction for a 2D quantity. The 1D advection equation

$$D^n_{i,j}c^n_{i,j} = D^o_{i,j}c^o_{i,j} - \Delta t \frac{U_{i,j}\tilde{c}^u_{i,j}\Delta y^u_{i,j} - U_{i-1,j}\tilde{c}^u_{i-1,j}\Delta y^u_{i-1,j}}{\Delta x^c_{i,j}\Delta y^c_{i,j}}, \tag{72}$$

is accompanied by an fractional step for the 1D continuity equation

$$D^n_{i,j} = D^o_{i,j} - \Delta t \frac{U_{i,j}\Delta y^u_{i,j} - U_{i-1,j}\Delta y^u_{i-1,j}}{\Delta x^c_{i,j}\Delta y^c_{i,j}}. \tag{73}$$

Here, n and o denote values before and after this operation, respectively, n denote intermediate values when other 1D advection steps come after this and o denotes intermediate values when other 1D advection steps came before this. Furthermore, when this u-directional split step is repeated during the total time step (Strang splitting), the time step Δt denotes a fraction of the full time step.
The interfacial fluxes $\tilde{c}^u_{i,j}$ are calculated according to the third-order polynomial scheme (so-called P_2 scheme), cast in Lax-Wendroff form by:

$$\tilde{c}_{i,j} = \begin{cases} \left(c_{i,j} + \frac{1}{2}\tilde{c}^+_{i,j}(1-|C_{i,j}|)(c_{i+1,j}-c_{i,j})\right) & \text{for } U_{i,j} \geq 0, \\ \left(c_{i+1,j} + \frac{1}{2}\tilde{c}^-_{i,j}(1-|C_{i,j}|)(c_{i,j}-c_{i+1,j})\right) & \text{else,} \end{cases} \tag{74}$$

with the Courant number $C_{i,j} = u_{i,j}\Delta t/\Delta x$ and

$$\tilde{c}^+_{i,j} = \alpha_{i,j} + \beta_{i,j}r^+_{i,j}, \quad \tilde{c}^-_{i,j} = \alpha_{i,j} + \beta_{i,j}r^-_{i,j}, \tag{75}$$

where

$$\alpha_{i,j} = \frac{1}{2} + \frac{1}{6}(1-2|C_{i,j}|), \quad \beta_{i,j} = \frac{1}{2} - \frac{1}{6}(1-2|C_{i,j}|), \tag{76}$$

and

$$r^+_{i,j} = \frac{c_{i,j}-c_{i-1,j}}{c_{i+1,j}-c_{i,j}}, \quad r^-_{i,j} = \frac{c_{i+2,j}-c_{i+1,j}}{c_{i+1,j}-c_{i,j}}. \tag{77}$$

It should be noted, that for $\tilde{c}^+_{i,j} = \tilde{c}^-_{i,j} = 1$ the original Lax-Wendroff scheme and for $\tilde{c}^+_{i,j} = \tilde{c}^-_{i,j} = 0$ the first-oder upstream scheme can be recovered.
In order to obtain a monotonic and positive scheme, the factors $\tilde{c}^+_{i,j}$ are limited in the following way:

$$\tilde{c}^+_{i,j} \rightarrow \max\left[0, \min\left(\tilde{c}^+_{i,j}, \frac{2}{1-|C_{i,j}|}, \frac{2r^+_{i,j}}{|C_{i,j}|}\right)\right], \tag{78}$$

and, equivalently, for $\tilde{c}^-_{i,j}$. This so-called PDM-limiter has been described in detail by *Leonard* (1991), who named the PDM-limited P_2 scheme also ULTIMATE QUICKEST (quadratic upstream interpolation for convective kinematics with estimated stream terms).
Some simpler limiters which do not exploit the third-order polynomial properties of the discretisation (74) have been listed by *Zalezak* (1987). Among those are the MUSCL scheme by *van Leer* (1979),

$$\tilde{c}^+_{i,j} \to \max\left[0, \min\left(2, 2r^+_{i,j}, \frac{1+r^+_{i,j}}{2}\right)\right], \tag{79}$$

and the Superbee scheme by *Roe* (1985),

$$\tilde{c}^+_{i,j} \to \max\left[0, \min(1, 2r^+_{i,j}), \min(r^+_{i,j}, 2)\right]. \tag{80}$$

The selector for the schemes is `scheme`:

`scheme = UPSTREAM`:	first-order upstream (monotone)
`scheme = P2`:	third-order polynomial (non-monotone)
`scheme = SUPERBEE`:	second-order TVD (monotone)
`scheme = MUSCL`:	second-order TVD (monotone)
`scheme = P2_PDM`:	third-order ULTIMATE-QUICKEST (monotone)

Furthermore, the horizontal diffusion in zonal direction with the constant diffusion coefficient `AH` is carried out here by means of a central difference second-order scheme. **USES:**

```
   use domain, only: imin,imax,jmin,jmax
#if !( defined(SPHERICAL) || defined(CURVILINEAR) )
   use domain, only: dx,dy,ard1
#endif
   use advection, only: adv_interfacial_reconstruction
   use advection, only: UPSTREAM
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   Note (KK): in general dxu, dyu and mask_flux do only have valid data
              within (_IRANGE_HALO_-1,_JRANGE_HALO_). In some cases the
              original field extension may even be _IRANGE_HALO_. Then
              explicit declared array bounds _IRANGE_HALO_-1 require a
              provision of the corresponding subarray and will cause
              copying of the non-contiguously data into a temporarily
              array. Therefore they are declared as pointers here. This
              however requires, that the provided pointers already carry
              the correct bounds.
   REALTYPE,intent(in)                          :: dt,splitfac,AH
   REALTYPE,dimension(E2DFIELD),intent(in)      :: f,U,DU
#if defined(SPHERICAL) || defined(CURVILINEAR)
   REALTYPE,dimension(:,:),pointer,contiguous,intent(in) :: dxu,dyu
   REALTYPE,dimension(E2DFIELD),intent(in)      :: arcd1
#endif
   integer,intent(in)                           :: scheme
   logical,dimension(:,:),pointer,contiguous,intent(in)  :: mask_flux
   logical,dimension(E2DFIELD),intent(in)       :: mask_update
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE,dimension(E2DFIELD),intent(inout) :: fi,Di,adv
```

LOCAL VARIABLES:

```
REALTYPE,dimension(E2DFIELD) :: uflux
logical                      :: use_limiter,use_AH
integer                      :: i,j,isub
REALTYPE                     :: dti,Dio,advn,cfl,fuu,fu,fd
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

7.4.8 adv_split_v - meridional advection of 2D quantities

INTERFACE:

```
   subroutine adv_split_v(dt,f,fi,Di,adv,V,DV,   &
#if defined(SPHERICAL) || defined(CURVILINEAR)
                          dxv,dyv,arcd1,         &
#endif
                          splitfac,scheme,AH,    &
                          mask_flux,mask_update)
   Note (KK): Keep in sync with interface in advection.F90
```

DESCRIPTION:

Executes an advection step in meridional direction for a 2D quantity in analogy to routine `adv_u_split` (see section 7.4.7 on page 73). **USES:**

```
   use domain, only: imin,imax,jmin,jmax
#if !( defined(SPHERICAL) || defined(CURVILINEAR) )
   use domain, only: dx,dy,ard1
#endif
   use advection, only: adv_interfacial_reconstruction
   use advection, only: UPSTREAM
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE,intent(in)                                          :: dt,splitfac,AH
   REALTYPE,dimension(E2DFIELD),intent(in)                      :: f,V,DV
#if defined(SPHERICAL) || defined(CURVILINEAR)
   REALTYPE,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in) :: dxv,dyv
   REALTYPE,dimension(E2DFIELD),intent(in)                      :: arcd1
#endif
   integer,intent(in)                                           :: scheme
   logical,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in)  :: mask_flux
   logical,dimension(E2DFIELD),intent(in)                       :: mask_update
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),intent(inout)                   :: fi,Di,adv
```

LOCAL VARIABLES:

```
   REALTYPE,dimension(E2DFIELD) :: vflux
   logical            :: use_limiter,use_AH
   integer            :: i,j,jsub
   REALTYPE           :: dti,Dio,advn,cfl,fuu,fu,fd
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

7.4.9 adv_arakawa_j7_2dh - 2DH Arakawa J7 advection of 2D quantities

INTERFACE:

```
   subroutine adv_arakawa_j7_2dh(dt,f,fi,Di,adv,vfU,vfV,Dn,DU,DV,  &
#if defined(SPHERICAL) || defined(CURVILINEAR)
                                 dxv,dyu,dxu,dyv,arcd1,            &
#endif
                                 AH,az,                            &
                                 mask_uflux,mask_vflux,mask_xflux)
   Note (KK): Keep in sync with interface in advection.F90
```

DESCRIPTION:

USES:

```
   use domain, only: imin,imax,jmin,jmax
#if !( defined(SPHERICAL) || defined(CURVILINEAR) )
   use domain, only: dx,dy,ard1
#endif
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE,intent(in)                                          :: dt,AH
   REALTYPE,dimension(E2DFIELD),target,intent(in)               :: f
   REALTYPE,dimension(E2DFIELD),intent(in)                      :: vfU,vfV,Dn,DU,DV
#if defined(SPHERICAL) || defined(CURVILINEAR)
   REALTYPE,dimension(:,:),pointer,contiguous,intent(in)        :: dxu,dyu
   REALTYPE,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in) :: dxv,dyv
   REALTYPE,dimension(E2DFIELD),intent(in)                      :: arcd1
#endif
   integer,dimension(E2DFIELD),intent(in)                       :: az
   logical,dimension(:,:),pointer,contiguous,intent(in)         :: mask_uflux,mask_xflux
   logical,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in)  :: mask_vflux
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),target,intent(inout)            :: fi,Di,adv
```

LOCAL VARIABLES:

```
   logical                      :: use_AH
   integer                      :: i,j,matsuno_it
   REALTYPE                     :: Dio,advn
   REALTYPE,dimension(:,:),pointer,contiguous :: faux,p_fiaux,p_Diaux,p_advaux
   REALTYPE,dimension(E2DFIELD) :: flux_e,flux_n,flux_ne,flux_nw
   REALTYPE,dimension(E2DFIELD) :: f_e,f_n,f_ne,f_nw
   REALTYPE,dimension(E2DFIELD),target :: fiaux,Diaux,advaux
   REALTYPE,dimension(E2DFIELD) :: uflux,vflux
   REALTYPE,parameter           :: one3rd = _ONE_/_THREE_
   REALTYPE,parameter           :: one6th = one3rd/_TWO_
```

REVISION HISTORY:

```
   Original author(s): Knut Klingbeil
```

7.4.10 adv_upstream_2dh - 2DH upstream advection of 2D quantities

INTERFACE:

```
   subroutine adv_upstream_2dh(dt,f,fi,Di,adv,U,V,Dn,DU,DV, &
#if defined(SPHERICAL) || defined(CURVILINEAR)
                               dxv,dyu,dxu,dyv,arcd1,       &
#endif
                               AH,az)
   Note (KK): Keep in sync with interface in advection.F90
```

DESCRIPTION:

In this routine, the first-order upstream advection scheme is applied for the two horizontal directions in one step. The scheme should be positive definite and of high resolution. In order to remove truncation errors which might in Wadden Sea applications cause non-monotonicity, a truncation of over- and undershoots is carried out at the end of this subroutine. Such two-dimensional schemes are advantageous in Wadden Sea applications, since one-dimensional directional-split schemes might compute negative intermediate depths. **USES:**

```
   use domain, only: imin,imax,jmin,jmax
#if !( defined(SPHERICAL) || defined(CURVILINEAR) )
   use domain, only: dx,dy,ard1
#endif
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE,intent(in)                                          :: dt,AH
   REALTYPE,dimension(E2DFIELD),intent(in)                      :: f,U,V,Dn,DU,DV
#if defined(SPHERICAL) || defined(CURVILINEAR)
   REALTYPE,dimension(:,:),pointer,contiguous,intent(in)        :: dxu,dyu
   REALTYPE,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in) :: dxv,dyv
   REALTYPE,dimension(E2DFIELD),intent(in)                      :: arcd1
#endif
   integer,dimension(E2DFIELD),intent(in)                       :: az
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),intent(inout)                   :: fi,Di,adv
```

LOCAL VARIABLES:

```
   integer                      :: i,j,ii,jj
   REALTYPE                     :: Dio,advn
   REALTYPE,dimension(E2DFIELD) :: uflux,vflux
   REALTYPE,dimension(E2DFIELD) :: cmin,cmax
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

7.4.11 adv_fct_2dh - 2DH FCT advection of 2D quantities

INTERFACE:

```
   subroutine adv_fct_2dh(fct,dt,f,fi,Di,adv,U,V,Dn,DU,DV, &
#if defined(SPHERICAL) || defined(CURVILINEAR)
                          dxv,dyu,dxu,dyv,arcd1,           &
#endif
                          AH,az,                           &
                          mask_uflux,mask_vflux)
   Note (KK): keep in sync with interface in advection.F90
```

DESCRIPTION:

In this routine, the flux corrected transport advection scheme by *Zalezak* (1979) is applied for the two horizontal directions in one step. For details of this type of operator splitting, see section 7.4.10 on page 78).
The monotone low-order flux is the first-order upstream scheme, the high-order flux is the third-order ULTIMATE QUICKEST scheme by *Leonard et al.* (1995). The scheme should thus be positive definite and of high resolution. In order to remove truncation errors which might in Wadden Sea applications cause non-monotonicity, a truncation of over- and undershoots is carried out at the end of this subroutine. Such two-dimensional schemes are advantageous in Wadden Sea applications, since one-dimensional directioal-split schemes might compute negative intermediate solutions. Extra checks for boundaries including mirroring out of the transported quantities are performed in order to account for the third-order large stencils.
If GETM is executed as slice model (compiler option `SLICE_MODEL`) the advection step for the y direction is not executed. **USES:**

```
   use domain, only: imin,imax,jmin,jmax
#if !( defined(SPHERICAL) || defined(CURVILINEAR) )
   use domain, only: dx,dy,ard1
#endif
   use halo_zones, only : update_2d_halo,wait_halo,z_TAG
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   logical,intent(in)                                          :: fct
   REALTYPE,intent(in)                                         :: dt,AH
   REALTYPE,dimension(E2DFIELD),intent(in)                     :: f,U,V,Dn,DU,DV
#if defined(SPHERICAL) || defined(CURVILINEAR)
   REALTYPE,dimension(:,:),pointer,contiguous,intent(in)       :: dxu,dyu
   REALTYPE,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in) :: dxv,dyv
   REALTYPE,dimension(E2DFIELD),intent(in)                     :: arcd1
#endif
   integer,dimension(E2DFIELD),intent(in)                      :: az
   logical,dimension(:,:),pointer,contiguous,intent(in)        :: mask_uflux
   logical,dimension(_IRANGE_HALO_,_JRANGE_HALO_-1),intent(in) :: mask_vflux
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),intent(inout)                  :: fi,Di,adv
```

LOCAL VARIABLES:

```
   integer         :: i,j
   REALTYPE,dimension(E2DFIELD) :: Dio
```

```
   REALTYPE,dimension(E2DFIELD) :: uflux,flx
#ifndef SLICE_MODEL
   REALTYPE,dimension(E2DFIELD) :: vflux,fly
#endif
   REALTYPE,dimension(E2DFIELD) :: faux,rp,rm,cmin,cmax
   REALTYPE        :: CNW,CW,CSW,CSSW,CWW,CSWW,CC,CS
   REALTYPE        :: advn,uuu,vvv,CExx,Cl,Cu,fac
   REALTYPE,parameter :: one12th=_ONE_/12,one6th=_ONE_/6,one3rd=_ONE_/3
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

7.4.12 bottom_friction - calculates the 2D bottom friction. (Source File: bottom_friction.F90)

INTERFACE:

```
subroutine bottom_friction(runtype)
```

DESCRIPTION:

In this routine the bottom friction for the external (vertically integrated) mode is calculated. This is done separately for the U-equation in the U-points and for the V-equation in the V-points. The drag coefficient R for the external mode is given in eq. (71) on page 53. For `runtype=1` (only vertically integrated calculations), the bottom roughness length is depending on the bed friction velocity u_*^b and the molecular viscosity ν:

$$z_0^b = 0.1\frac{\nu}{u_*^b} + \left(z_0^b\right)_{\min}, \tag{81}$$

see e.g. *Kagan* (1995), i.e. the given roughness may be increased by viscous effects. After this, the drag coefficient is multiplied by the absolute value of the local velocity, which is alculated by dividing the local transports by the local water depths and by properly interpolating these velocities to the U- and V-points. The resulting fields are `ru`, representing $R\sqrt{u^2+v^2}$ on the U-points and `rv`, representing this quantity on the V-points. **USES:**

```
   use parameters, only: kappa,avmmol
   use domain, only: imin,imax,jmin,jmax,au,av,min_depth
   use variables_2d
   use getm_timers,  only: tic, toc, TIM_BOTTFRICT
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: runtype
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   integer                   :: i,j
   REALTYPE                  :: uloc(E2DFIELD),vloc(E2DFIELD)
   REALTYPE                  :: HH(E2DFIELD),fricvel(E2DFIELD)
```

7.4.13 uv_advect - 2D advection of momentum (Source File: uv_advect.F90)

INTERFACE:

```
subroutine uv_advect(U,V,DU,DV)

Note (KK): keep in sync with interface in m2d.F90
```

DESCRIPTION:

Wrapper to prepare and do calls to `do_advection` (see section 7.4.2 on page 67) to calculate the advection terms of the depth-averaged velocities. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,az,au,av,ax
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxv,dyu
#else
   use domain, only: dx,dy
#endif
   use m2d, only: dtm,vel2d_adv_split,vel2d_adv_hor
   use variables_2d, only: UEx,VEx
   use advection, only: NOADV,UPSTREAM,J7,do_advection
   use halo_zones, only: update_2d_halo,wait_halo,U_TAG,V_TAG
   use getm_timers, only: tic,toc,TIM_UVADV,TIM_UVADVH
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),intent(in)        :: U,V
   REALTYPE,dimension(E2DFIELD),target,intent(in) :: DU,DV
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer :: i,j
   REALTYPE,dimension(E2DFIELD)               :: fadv,Uadv,Vadv,DUadv,DVadv
   REALTYPE,dimension(E2DFIELD),target        :: Dadv
   REALTYPE,dimension(:,:),pointer,contiguous :: pDadv
```

7.4.14 uv_diffusion - lateral diffusion of depth-averaged velocity (Source File: uv_diffusion.F90)

INTERFACE:

```
subroutine uv_diffusion(An_method,U,V,D,DU,DV)

Note (KK): keep in sync with interface in m2d.F90
```

DESCRIPTION:

This wrapper calls routine `uv_diff_2dh` (see section 7.4.15 on page 84). **USES:**

```
use domain, only: imin,imax,jmin,jmax
use m2d, only: uv_diff_2dh
use m2d, only: Am
use variables_2d, only: UEx,VEx
use getm_timers,  only: tic,toc,TIM_UVDIFF

IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer,intent(in)                      :: An_method
REALTYPE,dimension(E2DFIELD),intent(in) :: U,V,D,DU,DV
```

REVISION HISTORY:

```
Original author(s): Hans Burchard
```

LOCAL VARIABLES:

7.4.15 uv_diff_2dh - lateral diffusion of velocity

INTERFACE:

```
subroutine uv_diff_2dh(An_method,UEx,VEx,U,V,D,DU,DV,hsd_u,hsd_v)

Note (KK): keep in sync with interface in m2d.F90
```

DESCRIPTION:

Here, the diffusion terms for the vertically integrated transports are calculated by means of central differences, following the finite volume approach. They are added to the advection terms into the terms `UEx` and `VEx` for the U- and the V-equation, respectively. The physical diffusion with the given eddy viscosity coefficient A_h^M is based on velocity gradients, whereas an additional numerical damping of the barotropic mode is based on gradients of the transports with the damping coefficient A_h^N, see the example given as equations (90) and (91).
First diffusion term in (61):

$$\left(mn\,\partial_{\mathcal{X}} \left(2A_h^M D \partial_{\mathcal{X}} \left(\frac{U}{D} \right) + A_h^N \partial_{\mathcal{X}} U \right) \right)_{i,j} \approx \frac{\mathcal{F}^{Dxx}_{i+1,j} - \mathcal{F}^{Dxx}_{i,j}}{\Delta x^u_{i,j} \Delta y^u_{i,j}} \tag{82}$$

with diffusive fluxes

$$\mathcal{F}^{Dxx}_{i,j} = \left(2A_h^M D_{i,j} \left(\frac{U_{i,j}}{D^u_{i,j}} - \frac{U_{i-1,j}}{D^u_{i-1,j}} \right) + A_h^N \left(U_{i,j} - U_{i-1,j} \right) \right) \frac{\Delta y^c_{i,j}}{\Delta x^c_{i,j}}. \tag{83}$$

Second diffusion term in (61):

$$\left(mn\,\partial_{\mathcal{Y}} \left(A_h^M D \left(\partial_{\mathcal{Y}} \left(\frac{U}{D} \right) + \partial_{\mathcal{X}} \left(\frac{V}{D} \right) \right) + A_h^N \partial_{\mathcal{Y}} U \right) \right)_{i,j} \approx \frac{\mathcal{F}^{Dxy}_{i,j} - \mathcal{F}^{Dxy}_{i,j-1}}{\Delta x^x_{i,j} \Delta y^x_{i,j}} \tag{84}$$

with diffusive fluxes

$$\begin{aligned} \mathcal{F}^{Dxy}_{i,j} &= A_h^M \frac{1}{2} \left(D^u_{i,j} + D^u_{i,j+1} \right) \Delta x^x_{i,j} \left(\left(\frac{U_{i,j+1}}{D^u_{i,j+1}} - \frac{U_{i,j}}{D^u_{i,j}} \right) \frac{1}{\Delta y^x_{i,j}} + \left(\frac{V_{i+1,j}}{D^v_{i+1,j}} - \frac{V_{i,j}}{D^v_{i,j}} \right) \frac{1}{\Delta x^x_{i,j}} \right) \\ &\quad + A_h^N \left(U_{i,j+1} - U_{i,j} \right) \frac{\Delta x^x_{i,j}}{\Delta y^x_{i,j}}. \end{aligned} \tag{85}$$

First diffusion term in (62):

$$\left(mn\,\partial_{\mathcal{X}} \left(A_h^M D \left(\partial_{\mathcal{Y}} \left(\frac{U}{D} \right) + \partial_{\mathcal{X}} \left(\frac{V}{D} \right) \right) + A_h^N \partial_{\mathcal{X}} V \right) \right)_{i,j} \approx \frac{\mathcal{F}^{Dyx}_{i,j} - \mathcal{F}^{Dyx}_{i-1,j}}{\Delta x^x_{i,j} \Delta y^x_{i,j}} \tag{86}$$

with diffusive fluxes

$$\begin{aligned} \mathcal{F}^{Dyx}_{i,j} &= A_h^M \frac{1}{2} \left(D^v_{i,j} + D^v_{i+1,j} \right) \Delta y^x_{i,j} \left(\left(\frac{U_{i,j+1}}{D^u_{i,j+1}} - \frac{U_{i,j}}{D^u_{i,j}} \right) \frac{1}{\Delta y^x_{i,j}} + \left(\frac{V_{i+1,j}}{D^v_{i+1,j}} - \frac{V_{i,j}}{D^v_{i,j}} \right) \frac{1}{\Delta x^x_{i,j}} \right) \\ &\quad + A_h^N \left(V_{i+1,j} - V_{i,j} \right) \frac{\Delta y^x_{i,j}}{\Delta x^x_{i,j}}. \end{aligned} \tag{87}$$

Second diffusion term in (62):

$$\left(mn\,\partial_{\mathcal{Y}} \left(2A_h^M D \partial_{\mathcal{Y}} \left(\frac{V}{D} \right) + A_h^N \partial_{\mathcal{Y}} V \right) \right)_{i,j} \approx \frac{\mathcal{F}^{Dyy}_{i,j+1} - \mathcal{F}^{Dyy}_{i,j}}{\Delta x^v_{i,j} \Delta y^v_{i,j}} \tag{88}$$

with diffusive fluxes

$$\mathcal{F}^{Dyy}_{i,j} = \left(2A_h^M D_{i,j} \left(\frac{V_{i,j}}{D^v_{i,j}} - \frac{V_{i,j-1}}{D^v_{i,j-1}} \right) + A_h^N \left(V_{i,j} - V_{i,j-1} \right) \right) \frac{\Delta x^c_{i,j}}{\Delta y^c_{i,j}}. \tag{89}$$

The role of the additional diffusion of U and V with the diffusion coefficient A_h^N is best demonstrated by means of a simplified set of vertically integrated equations:

$$\begin{aligned} \partial_t \eta &= -\partial_x U - \partial_y V \\ \partial_t U &= -gD\partial_x \eta + A_h^N \left(\partial_{xx} U + \partial_{yy} U \right) \\ \partial_t V &= -gD\partial_y \eta + A_h^N \left(\partial_{xx} V + \partial_{yy} V \right), \end{aligned} \tag{90}$$

which can be transformed into an equation for $\partial_t \eta$ by derivation of the η-equation with respect to t, of the U-equation with respect to x and the V-equation with respect to y and subsequent elimination of U and V:

$$\partial_t \left(\partial_t \eta \right) = gD \left(\partial_{xx} \eta + \partial_{yy} \eta \right) + A_h^N \left(\partial_{xx} \left(\partial_t \eta \right) + \partial_{yy} \left(\partial_t \eta \right) \right), \tag{91}$$

which can be interpreted as a wave equation with a damping on $\partial_t \eta$. This introduces an explicit damping of free surface elevation oscillations in a momentum-conservative manner. Hydrodynamic models with implicit treatment of the barotropic mode do not need to apply this method due to the implicit damping of those models, see e.g. *Backhaus* (1985). The implementation of this explicit damping described here has been suggested by Jean-Marie Beckers, Liége (Belgium).
When working with the option `SLICE_MODEL`, the calculation of all gradients in y-direction is suppressed. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,az,au,av,ax
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dyc,arud1,dxx,dyx,arvd1,dxc
#else
   use domain, only: dx,dy,ard1
#endif
   use m2d, only: Am
   use variables_2d, only: An,AnX
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer,intent(in)                                  :: An_method
   REALTYPE,dimension(E2DFIELD),intent(in),optional  :: U,V,D,DU,DV
```

INPUT/OUTPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),intent(inout)        :: UEx,VEx
```

OUTPUT PARAMETERS:

```
   REALTYPE,dimension(E2DFIELD),intent(out),optional :: hsd_u,hsd_v
```

REVISION HISTORY:

```
Original author(s): Hans Burchard
Modified by       : Knut Klingbeil
```

LOCAL VARIABLES:

```
REALTYPE,dimension(E2DFIELD) :: work2d
logical :: use_Am
integer :: i,j
```

7.4.16 momentum - 2D-momentum for all interior points. (Source File: momentum.F90)

INTERFACE:

```
subroutine momentum(n,tausx,tausy,airp)
```

DESCRIPTION:

This small routine calls the U-equation and the V-equation in an alternating sequence (UVVU-UVVUUVVU), in order to provide higher accuracy and energy conservation for the explicit numerical treatment of the Coriolis term. **USES:**

```
use domain, only: imin,imax,jmin,jmax
! For timer here: Only clock what is not taken at "next" level.
use getm_timers, only: tic, toc, TIM_MOMENTUM
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: n
REALTYPE, intent(in)                :: tausx(E2DFIELD)
REALTYPE, intent(in)                :: tausy(E2DFIELD)
REALTYPE, intent(in)                :: airp(E2DFIELD)
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
logical                   :: ufirst=.false.
```

7.4.17 umomentum - 2D-momentum for all interior points.

INTERFACE:

```
subroutine umomentum(tausx,airp)
```

DESCRIPTION:

Here, the vertically integrated U-momentum equation (61) given on page 52 including a number of slow terms is calculated. One slight modification is that for better stability of drying and flooding processes the slow friction term S_F^x is now also multiplied with the parameter α defined in eq. (5). Furthermore, the horizontal pressure gradient $\partial_x^*\zeta$ is modified in order to support drying and flooding, see figure 10 on page 31 and the explanations in section 5.5. $\partial_x^*\zeta$ is now also considering the atmospheric pressure gradient at sea surface height.
For numerical stability reasons, the U-momentum equation is here discretised in time such that the bed friction is treated explicitely:

$$U^{n+1} = \frac{U^n - \Delta t_m(gD\partial_x^*\zeta + \alpha(-\frac{\tau_x^s}{\rho_0} - fV^n + U_{Ex} + S_A^x - S_D^x + S_B^x + S_F^x))}{1 + \Delta t_m \frac{R}{D^2}\sqrt{(U^n)^2 + (V^n)^2}}, \tag{92}$$

with U_{Ex} combining advection and diffusion of U, see routines `uv_advect` (section 7.4.13 on page 82) and `uv_diffusion` (section 7.4.14 on page 83). The slow terms are calculated in the routine `slow_terms` documented in section 8.13.11 on page 177. In (92), U^{n+1} denotes the transport on the new and U^n and V^n the transports on the old time level.
The Coriolis term fU for the subsequent V-momentum is also calculated here, by directly interpolating the U-transports to the V-points or by a method suggested by *Espelid et al.* (2000) which takes the varying water depths into account.
Some provisions for proper behaviour of the U-transports when GETM runs as slice model are made as well, see section 3.2 on page 15. **USES:**

```
   use parameters, only: g,rho_0
   use domain, only: imin,imax,jmin,jmax
   use domain, only: H,au,av,min_depth,dry_u,Cori,corv
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxu,arvd1,dxc,dyx
   use variables_2d, only: V
#else
   use domain, only: dx
#endif
   use m2d, only: dtm
   use variables_2d, only: D,z,UEx,U,DU,fV,SlUx,Slru,ru,fU,DV
   use getm_timers,  only: tic, toc, TIM_MOMENTUMH
   use halo_zones, only : update_2d_halo,wait_halo,U_TAG
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: tausx(E2DFIELD),airp(E2DFIELD)
```

LOCAL VARIABLES:

```
   integer                   :: i,j
   REALTYPE                  :: zp,zm,zx,tausu,Slr,Uloc
   REALTYPE                  :: gamma=rho_0*g
   REALTYPE                  :: cord_curv=_ZERO_
   REALTYPE                  :: gammai
```

7.4.18 vmomentum - 2D-momentum for all interior points.

INTERFACE:

```
subroutine vmomentum(tausy,airp)
```

DESCRIPTION:

Here, the vertically integrated V-momentum equation (62) given on page 52 including a number of slow terms is calculated. One slight modification is that for better stability of drying and flooding processes the slow friction term S_F^y is now also multiplied with the parameter α defined in eq. (5). Furthermore, the horizontal pressure gradient $\partial_y^*\zeta$ is modified in order to support drying and flooding, see figure 10 on page 31 and the explanations in section 5.5. $\partial_y^*\zeta$ is now also considering the atmospheric pressure gradient at sea surface height.
For numerical stability reasons, the V-momentum equation is here discretised in time such that the bed friction is treated explicitely:

$$V^{n+1} = \frac{V^n - \Delta t_m(gD\partial_y^*\zeta + \alpha(-\frac{\tau_y^s}{\rho_0} + fU^n + V_{Ex} + S_A^y - S_D^y + S_B^y + S_F^y))}{1 + \Delta t_m \frac{R}{D^2}\sqrt{(U^n)^2 + (V^n)^2}}, \tag{93}$$

with V_{Ex} combining advection and diffusion of V, see routines `uv_advect` (section 7.4.13 on page 82) and `uv_diffusion` (section 7.4.14 on page 83). The slow terms are calculated in the routine `slow_terms` documented in section 8.13.11 on page 177. In (93), V^{n+1} denotes the transport on the new and U^n and V^n the transports on the old time level.
The Coriolis term fV for the subsequent U-momentum is also calculated here, by directly interpolating the U-transports to the U-points or by a method suggested by *Espelid et al.* (2000) which takes the varying water depths into account.
Some provisions for proper behaviour of the V-transports when GETM runs as slice model are made as well, see section 3.2 on page 15. **USES:**

```
   use parameters, only: g,rho_0
   use domain, only: imin,imax,jmin,jmax
   use domain, only: H,au,av,min_depth,dry_v,Cori,coru
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dyv,arud1,dxx,dyc
   use m2d, only: U
#else
   use domain, only: dy
#endif
   use m2d, only: dtm
   use variables_2d, only: D,z,VEx,V,DV,fU,SlVx,Slrv,rv,fV,DU
   use getm_timers,  only: tic, toc, TIM_MOMENTUMH
   use halo_zones, only : update_2d_halo,wait_halo,V_TAG
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)                :: tausy(E2DFIELD),airp(E2DFIELD)
```

LOCAL VARIABLES:

```
   integer                   :: i,j
   REALTYPE                  :: zp,zm,zy,tausv,Slr,Vloc
   REALTYPE                  :: gamma=rho_0*g
   REALTYPE                  :: cord_curv=_ZERO_
   REALTYPE                  :: gammai
```

7.4.19 sealevel - using the cont. eq. to get the sealevel. (Source File: sealevel.F90)

INTERFACE:

```
   subroutine sealevel(loop)
```

DESCRIPTION:

Here, the sea surface elevation is iterated according to the vertically integrated continuity equation given in (57) on page 51.
When working with the option SLICE_MODEL, the elevations at $j = 2$ are copied to $j = 3$.
Now with consideration of fresh water fluxes (precipitation and evaporation). Positive for flux into the water. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,az,H
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only : arcd1,dxv,dyu
#else
   use domain, only : dx,dy,ard1
#endif
   use m2d, only: dtm,sealevel_check
   use variables_2d, only: z,zo,U,V,fwf
   use getm_timers, only: tic, toc, TIM_SEALEVEL, TIM_SEALEVELH
   use halo_zones, only : update_2d_halo,wait_halo,z_TAG
#ifdef USE_BREAKS
   use halo_zones, only : nprocs,set_flag,u_TAG,v_TAG
   use variables_2d, only: break_mask,break_stat
   use domain, only : min_depth,au,av
#endif
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: loop
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer                   :: i,j
#ifdef USE_BREAKS
   integer                   :: n,break_flag,break_flags(nprocs)
#endif
```

7.4.20 sealevel_nan_check - Sweep the sealevel (z) for NaN values

INTERFACE:

```
subroutine sealevel_nan_check
```

DESCRIPTION:

The sea surface elevation (2d) variable is sweeped scanning for not-a-number (NaN). NaN values indicate that the integration has become unstable and that it really should be stopped. First time a NaN value is found, a warning is issued and possibly the code is stopped. After the first encounter, the sweep is suspended.
The behaviour of this routine is controlled by the `sealevel_check` parameter in the `m2d` namelist.

USES:

```
use domain, only: imin,imax,jmin,jmax,ioff,joff
use m2d, only: sealevel_check
use variables_2d, only: z
use exceptions, only: getm_error
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Bjarne B\"uchmann
```

LOCAL VARIABLES:

```
integer, save :: Ncall = 0
integer, save :: can_check   = 0
integer, save :: have_warned = 0
integer       :: num_nan
integer       :: i,j,inan,jnan, idum
REALTYPE      :: ahuge,zdum
```

7.4.21 sealevel_nandum. Helper routine to spot NaNs

INTERFACE:

```
subroutine sealevel_nandum(a,b,idum)
```

USES:

INPUT PARAMETERS:

```
REALTYPE, intent(in)       :: a,b
```

OUTPUT PARAMETERS:

```
integer, intent(out)      :: idum
```

DESCRIPTION:

This routine is a kind of dummy routine primarily to provide a means to spot NaN values. Output is 1 or 2, based on which is smaller (a or b, respectively). The default is 2, and the idea is that "imin=2" should be returned also if a is NaN. If b=HUGE(b), then this provides a means to detect if a is a denormal number.

7.4.22 depth_update - adjust the depth to new elevations. (Source File: depth_update.F90)

INTERFACE:

```
subroutine depth_update
```

DESCRIPTION:

This routine which is called at every micro time step updates all necessary depth related information. These are the water depths in the T-, U- and V-points, `D`, `DU` and `DV`, respectively, and the drying value α defined in equation (5) on page 14 in the T-, the U- and the V-points (`dry_z`, `dry_u` and `dry_v`).
When working with the option `SLICE_MODEL`, the water depths in the V-points are mirrored from $j = 2$ to $j = 1$ and $j = 3$. **USES:**

```
use domain, only: imin,imax,jmin,jmax,H,HU,HV,min_depth,crit_depth
use domain, only: az,au,av,dry_z,dry_u,dry_v
use variables_2d, only: D,z,zo,DU,DV
use getm_timers,  only: tic, toc, TIM_DPTHUPDATE
$ use omp_lib
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j
REALTYPE                  :: d1,d2i,x
```

7.4.23 update_2d_bdy - update 2D boundaries every time step. (Source File: update_2d_bdy.F90)

INTERFACE:

```
subroutine update_2d_bdy(loop,bdyramp)
```

DESCRIPTION:

In this routine sea surface elevation boundary conditions are read in from a file, interpolated to the actual time step, and distributed to the open boundary grid boxes. Only for a special test case (SYLT_TEST), ascii data reading is supported. For a few special simple cases, analytical calculation of boundary elevations is supported. The generic way is reading in boundary data from a netcdf file, which is managed in get_2d_bdy via get_2d_bdy_ncdf. **USES:**

```
   use domain, only: NWB,NNB,NEB,NSB,H,min_depth,imin,imax,jmin,jmax,az
   use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
   use domain, only: bdy_index,nsbv
   use domain, only: bdy_2d_type
   use m2d, only: dtm,bdyfmt_2d,bdy_data,bdy_data_u,bdy_data_v
   use variables_2d, only: z,D,U,DU,V,DV
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxc,dyc
#else
   use domain, only: dx,dy
#endif
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                :: loop,bdyramp
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   logical, save             :: first=.true.
   REALTYPE, save            :: time_array(1000),zbo(1000),zbn(1000)
   REALTYPE, save            :: t,t1,t2
   REALTYPE                  :: a,ratio,fac
   integer                   :: i,j,k,l,n
   REALTYPE, parameter       :: FOUR=4.*_ONE_
```

7.4.24 do_residual - barotropic residual currents. (Source File: residual.F90)

INTERFACE:

```
subroutine do_residual(finish)
```

DESCRIPTION:

Here, the residual transports and depths are integrated up every time step. At the end of the simulation, the Eulerian residual currents are calculated from:

$$u_{res} = \frac{\int_{t_0}^{t_1} U \, \mathrm{d}\tau}{\int_{t_0}^{t_1} D^u \, \mathrm{d}\tau}, \quad v_{res} = \frac{\int_{t_0}^{t_1} V \, \mathrm{d}\tau}{\int_{t_0}^{t_1} D^v \, \mathrm{d}\tau}, \tag{94}$$

where t_0 is the time when the residual calculation begins (to be chosen from namelist) and t_1 is the finishing time of the model simulation.

USES:

```
use variables_2d, only: u,v,res_du,res_u,res_dv,res_v,du,dv
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: finish
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

7.4.25 cfl_check - check for explicit barotropic time step. (Source File: cfl_check.F90)

INTERFACE:

```
subroutine cfl_check()
```

DESCRIPTION:

This routine loops over all horizontal grid points and calculated the maximum time step according to the CFL criterium by *Beckers and Deleersnijder* (1993):

$$\Delta t_{\max} = \min_{i,j} \left\{ \frac{\Delta x_{i,j} \Delta y_{i,j}}{\sqrt{2} c_{i,j} \sqrt{\Delta x_{i,j}^2 + \Delta y_{i,j}^2}} \right\} \tag{95}$$

with the local shallow water wave speed

$$c_{i,j} = \sqrt{g H_{i,j}}, \tag{96}$$

where g is the gravitational acceleration and $H_{i,j}$ is the local bathymetry value. In case that the chosen micro time step Δt_m is larger than $\Delta t_{\max}$, the program will be aborted. In any case the CFL diagnostics will be written to standard output.

USES:

```
   use parameters, only: g
   use domain, only: imin,imax,jmin,jmax,H,az
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dyc,dxc
#else
   use domain, only: dy,dx
#endif
   use m2d, only: dtm
   IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   integer                   :: pos(2),max_pos(2),rc,i,j
   REALTYPE                  :: h_max=-99.,c,max_dt,dtt
   logical, dimension(:,:), allocatable :: lmask
```

8 Introduction to 3d module

8.1 Overview over 3D routines in GETM

This module contains the physical core of GETM. All three-dimensional equations are iterated here, which are currently the equations for

quantity	description	unit	variable	routine name	page
p_k	layer-int. u-transport	$\mathrm{m^2 s^{-1}}$	`uu`	`uu_momentum`	168
q_k	layer-int. v-transport	$\mathrm{m^2 s^{-1}}$	`vv`	`vv_momentum`	170
θ	potential temperature	°C	`T`	`do_temperature`	130
S	salinity	psu	`S`	`do_salinity`	134
C	suspended matter	$\mathrm{kg\,m^{-3}}$	`spm`	`do_spm`	166

The vertical grid for GETM, i.e. the layer thicknesses in all U-, V- and T-points, are defined in the routine `coordinates`, see section 8.5.4 on page 8.5.4.
The grid-related vertical velocity $\bar{w}_k$ is calculated directly from the layer-integrated continuity equation (25) which here done in the routine `ww_momentum` described on page 172.
The physics of the horizontal momentum equations is given in section 3.1.1, and their transformation to general vertical coordinates in section 4.2. Their numerical treatment will be discussed in the routines for the individual terms, see below. The forcing terms of the horizontal momentum equations are calculated in various routines, such as `uv_advect_3d` for the three-dimensional advection (which in turn calls `advection_3d` in case that higher order positive definite advection schemes are chosen for the momentum equation), `uv_diffusion_3d.F90` for the horizontal diffusion, `bottom_friction_3d` for the bottom friction applied to the lowest layer, and `internal_pressure` for the calculation of the internal pressure gradients.
The major tracer equations in any ocean model are those for potential temperature and salinity. They are calculated in the routines `do_temperature` and `do_salinity`. A further hard-coded tracer equation is the suspended matter equation, see `do_spm`.
In the near future (the present text is typed in February 2006), a general interface to the biogeochemical module of GOTM (also not yet released) will be available. This allow to add tracer equations of arbitrary complexity to GETM, ranging from completely passive tracer equations to complex ecosystem models such as ERSEM (*Baretta et al.* (1995)). The interfacing between this so-called GOTM-BIO to GETM is made in a similar manner than the interfacing between GETM and the GOTM turbulence module described in `gotm` on page 182. The basic structure of GOTM-BIO has been recently presented by *Burchard et al.* (2006). Some more details about the tracer equations currently included in GETM is given in section 8.2.
The entire turbulence model, which basically provides eddy viscosity ν_t and eddy diffusivity ν_t' is provided from the General Ocean Turbulence Model (GOTM, see *Umlauf et al.* (2005) for the source code documentation and `http://www.gotm.net` download of source code, docomentation and test scenarios). The turbulence module of GOTM (which is a complete one-dimensional water column model) is coupled to GETM via the interfacing routine `gotm` described in section `gotm` on page 182. Major input to the turbulence model are the shear squared $M^2 = (\partial_z u)^2 + (\partial_z u)^2$ and the buoyancy frequency squared $N^2 = \partial_z b$ with the buoyancy b from (4). Those are calculated and interpolated to the T-points where the turbulence model columns are located in the routine `ss_nn` described on page 179.
The surface and bottom stresses which need to be passed to the turbulence module as well, are interpolated to T-points in the routine `stresses_3d`, see page 181.
The module `rivers` (see section 8.12 on page 155) organises the riverine input of fresh water from any number of rivers.
Three-dimensional boundary conditions for temperature and salinity are provided by means of the module `bdy-3d`, see section 8.11 described on page 152.
The remaining routines in the module `3d` deal with the coupling of the external and the internal

mode. The basic idea of the mode splitting has already been discussed in section 5.1. The consistency of the two modes is given through the so-called slow terms, which are mode interaction terms resulting from subtracting vertically integrated equations with parameterised advection, diffusion, bottom friction and internal pressure gradient from vertically integrated equations with explicit vertical resolution of these processes. These slow terms which are updated every macro time step only (that is why we call them slow terms) need to be considered for the external mode included in module `2d`. Those slow terms are calculated here in the `3d` module at the end of `integrate_3d` and in the routine `slow_bottom_friction`, and they are added together in `slow_terms`, see the descriptions in sections 8.4.3, 8.13.10 and 8.13.11 on pages 105, 176, and 177, respectively.
One other important measure of coupling the two modes is to add to all calculated u- and v-velocity profiles the difference between their vertical integral and the time-average of the vertically integrated transport from the previous set of micro time steps. This shifting is done in the routines `uu_momentum_3d` and `vv_momentum_3d` and the time-average of the vertically integrated transport is updated in the `2d` module in the routine `m2d` and divided by the number of micro time steps per macro time step in `start_macro`. Further basic calculations performed in `start_macro` (see description in section 8.13.3 on page 167) are the updates of the *old* and *new* sea surface elevations with respect to the actual macro time step. The routine `stop_macro` (see description in section 8.13.12 on page 178) which called at the end of each macro time step simply resets the variables for the time-averaged transports to zero.

8.2 Tracer equations

The general conservation equation for tracers c^i with $1 \leq i \leq N_c$ (with N_c being the number of tracers), which can e.g. be temperature, salinity, nutrients, phytoplankton, zoo-plankton, suspended matter, chemical concentrations etc. is given as:

$$\begin{aligned} &\partial_t c^i + \partial_x(uc^i) + \partial_y(vc^i) + \partial_z((w+\alpha w_s^i)c^i) - \partial_z(\nu_t'\partial_z c^i) \\ &\quad -\partial_x(A_h^T\partial_x c^i) - \partial_y(A_h^T\partial_y c^i) = Q^i. \end{aligned} \tag{97}$$

Here, ν_t' denotes the vertical eddy diffusivity and A_h^T the horizontal diffusivity. Vertical migration of concentration with migration velocity w_s^i (positive for upward motion) is considered as well. This could be i.e. settling of suspended matter or active migration of phytoplankton. In order to avoid stability problems with vertical advection when intertidal flats are drying, the settling of SPM is linearly reduced towards zero when the water depth is between the critical and the minimum water depth. This is done by means of multiplication of the settling velocity with α, (see the definition in equation (5)). Q^i denotes all internal sources and sinks of the tracer c^i. This might e.g. be for the temperature equation the heating of water due to absorption of solar radiation in the water column.
Surface of bottom boundary conditions for tracers are usually given by prescribed fluxes:

$$-\alpha w_s^i c^i + \nu_t'\partial_z c^i = F_s^i \qquad \text{for } z = \zeta \tag{98}$$

and

$$-\alpha w_s^i c^i + \nu_t'\partial_z c^i = -F_b^i \qquad \text{for } z = -H, \tag{99}$$

with surface and bottom fluxes F_s^n and F_b^n directed into the domain, respectively.
At open lateral boundaries, the tracers c^n are prescribed for the horizontal velocity normal to the open boundary flowing into the domain. In case of outflow, a zero-gradient condition is used.
All tracer equations except those for temperature, salinity and suspended matter will be treated in the future by means of GOTM-BIO.
The two most important tracer equations which are hard-coded in GETM are the transport equations for potential temperature θ in °C and salinity S in psu (practical salinity units):

$$\partial_t\theta + \partial_x(u\theta) + \partial_y(v\theta) + \partial_z(w\theta) - \partial_z(\nu'_t\partial_z\theta)$$
$$-\partial_x(A^\theta_h\partial_x\theta) - \partial_y(A^\theta_h\partial_y\theta) = \frac{\partial_z I}{c'_p\rho_0}, \tag{100}$$

$$\partial_t S + \partial_x(uS) + \partial_y(vS) + \partial_z(wS) - \partial_z(\nu'_t\partial_z S)$$
$$-\partial_x(A^S_h\partial_x S) - \partial_y(A^S_h\partial_y S) = 0. \tag{101}$$

On the right hand side of the temperature equation (100) is a source term for absorption of solar radiation with the solar radiation at depth z, I, and the specific heat capacity of water, c'_p. According to *Paulson and Simpson* (1977) the radiation I in the upper water column may be parameterised by

$$I(z) = I_0\left(ae^{-\eta_1 z} + (1-a)e^{-\eta_2 z}\right). \tag{102}$$

Here, I_0 is the albedo corrected radiation normal to the sea surface. The weighting parameter a and the attenuation lengths for the longer and the shorter fraction of the short-wave radiation, η_1 and η_2, respectively, depend on the turbidity of the water. *Jerlov* (1968) defined 6 different classes of water from which *Paulson and Simpson* (1977) calculated weighting parameter a and attenuation coefficients η_1 and η_2.
At the surface, flux boundary conditions for T and S have to be prescribed. For the potential temperature, it is of the following form:

$$\nu'_t\partial_z T = \frac{Q_s + Q_l + Q_b}{c'_p\rho_0}, \qquad \text{for } z = \zeta, \tag{103}$$

with the sensible heat flux, Q_s, the latent heat flux, Q_l and the long wave back radiation, Q_b. Here, the *Kondo* (1975) bulk formulae have been used for calculating the momentum and temperature surface fluxes due to air-sea interactions. In the presence of sea ice, these air-sea fluxes have to be considerably changed, see e.g. *Kantha and Clayson* (2000b). Since there is no sea-ice model coupled to GETM presently, the surface heat flux is limited to positive values, when the sea surface temperature T_s reaches the freezing point

$$T_f = -0.0575\,S_s + 1.710523\cdot 10^{-3}\,S_s^{1.5} - 2.154996\cdot 10^{-4}\,S_s^2 \approx -0.0575\,S_s \tag{104}$$

with the sea surface salinity S_s, see e.g. *Kantha and Clayson* (2000a):

$$Q_{surf} = \begin{cases} Q_s + Q_l + Q_b, & \text{for } T_s > T_f, \\ \max\{0, Q_s + Q_l + Q_b\}, & \text{else.} \end{cases} \tag{105}$$

For the surface freshwater flux, which defines the salinity flux, the difference between evaporation Q_E (from bulk formulae) and precipitation Q_P (from observations or atmospheric models) is calculated:

$$\nu'_t\partial_z S = \frac{S(Q_E - Q_P)}{\rho_0(0)}, \qquad \text{for } z = \zeta, \tag{106}$$

where $\rho_0(0)$ is the density of freshwater at sea surface temperature. In the presence of sea-ice, the calculation of freshwater flux is more complex, see e.g. *Large et al.* (1994). However, for many short term calculations, the freshwater flux can often be neglected compared to the surface heat flux.
A complete revision of the surface flux calculation is currently under development. It will be the idea to have the same surface flux calculations for GOTM and GETM. In addition to the older

bulk formulae by *Kondo* (1975) we will also implement the more recent formlations by *Fairall et al.* (1996).
Heat and salinity fluxes at the bottom are set to zero.

8.3 Equation of state

The coupling between the potential temperature and salinity equations and the momentum equations is due to an algebraic equation of state:

$$\rho = \rho(\theta, S, p_0) \tag{107}$$

with $p_0 = g\rho_0(\zeta - z)$ being the hydrostatic reference pressure. In order to obtain potential density from the equation of state, p_0 needs to be set to zero, which is the default in GETM.
Currently the equation of state by *Fofonoff and Millard* (1983) is implemented into GETM, but the more recent and more consistent equation of state by *Jackett et al.* (2005) which is already contained in GOTM will be added as an option in the near future.
For the equation of state, also linearised version are implemented into GETM, for details, see section 8.9 on page 135.
For convinient use in other subroutines the buoyancy b as defined in (4) is calculated and stored in the GETM variable `buoy`.

8.4 Fortran: Module Interface m3d - 3D model component (Source File: m3d.F90)

INTERFACE:

```
module m3d
```

DESCRIPTION:

This module contains declarations for all variables related to 3D hydrodynamical calculations. Information about the calculation domain is included from the `domain` module. The module contains public subroutines for initialisation, integration and clean up of the 3D model component. The `m3d` module is initialised in the routine `init_3d`, see section 8.4.1 described on page 103. The actual calculation routines are called in `integrate_3d` (see section 8.4.3 on page 105). and are linked in from the library `lib3d.a`. After the simulation, the module is closed in `clean_3d`, see section 8.4.4 on page 107. **USES:**

```
   use exceptions
   use parameters, only: avmmol
   use domain, only: openbdy,maxdepth,vert_cord,az
   use m2d, only: uv_advect,uv_diffusion
   use variables_2d, only: z,Uint,Vint,UEx,VEx
#ifndef NO_BAROCLINIC
   use temperature,only: init_temperature, do_temperature, &
            init_temperature_field
   use salinity,   only: init_salinity, do_salinity, init_salinity_field
   use eqstate,    only: init_eqstate, do_eqstate
   use internal_pressure, only: init_internal_pressure, do_internal_pressure
   use internal_pressure, only: ip_method
#endif
   use variables_3d
   use advection, only: NOADV
   use advection_3d, only: init_advection_3d,print_adv_settings_3d,adv_ver_iterations
   use bdy_3d, only: init_bdy_3d, do_bdy_3d
   use bdy_3d, only: bdy3d_tmrlx, bdy3d_tmrlx_ucut, bdy3d_tmrlx_max, bdy3d_tmrlx_min
   Necessary to use halo_zones because update_3d_halos() have been moved out
   temperature.F90 and salinity.F90 - should be changed at a later stage
   use halo_zones, only: update_3d_halo,wait_halo,D_TAG

   IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   integer                          :: M=1
   REALTYPE                         :: cord_relax=_ZERO_
   integer                          :: vel3d_adv_split=0
   integer                          :: vel3d_adv_hor=1
   integer                          :: vel3d_adv_ver=1
   integer                          :: turb_adv_split=0
   integer                          :: turb_adv_hor=0
   integer                          :: turb_adv_ver=0
   logical                          :: calc_temp=.true.
   logical                          :: calc_salt=.true.
   logical                          :: bdy3d=.false.
   integer                          :: bdyfmt_3d,bdy3d_ramp
   character(len=PATH_MAX)          :: bdyfile_3d
```

```
   REALTYPE                            :: ip_fac=_ONE_
   integer                             :: vel_check=0
   REALTYPE                            :: min_vel=-4*_ONE_,max_vel=4*_ONE_
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   logical         :: advect_turbulence=.false.
#ifdef NO_BAROCLINIC
   integer         :: ip_method
#endif
   integer         :: ip_ramp=-1
```

8.4.1 init_3d - initialise 3D related stuff

INTERFACE:

```
subroutine init_3d(runtype,timestep,hotstart)
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: runtype
REALTYPE, intent(in)                :: timestep
logical, intent(in)                 :: hotstart
```

DESCRIPTION:

Here, the `m3d` namelist is read from `getm.inp`, and the initialisation of variables is called (see routine `init_variables` described on page 111). Furthermore, a number of consistency checks are made for the choices of the momentum advection schemes. When higher-order advection schemes are chosen for the momentum advection, the compiler option `UV_TVD` has to be set. Here, the macro time step Δt is calculated from the micro time step Δt_m and the split factor `M`. Then, in order to have the vertical coordinate system present already here, `coordinates` (see page 114) needs to be called, in order to enable proper interpolation of initial values for potential temperature θ and salinity S for cold starts. Those initial values are afterwards read in via the routines `init_temperature` (page 128) and `init_salinity` (page 132). Finally, in order to prepare for the first time step, the momentum advection and internal pressure gradient routines are initialised and the internal pressure gradient routine is called. **LOCAL VARIABLES:**

```
integer         :: rc
NAMELIST /m3d/ &
          M,cnpar,cord_relax,adv_ver_iterations,       &
          bdy3d,bdyfmt_3d,bdy3d_ramp,bdyfile_3d,      &
          bdy3d_tmrlx,bdy3d_tmrlx_ucut,               &
          bdy3d_tmrlx_max,bdy3d_tmrlx_min,            &
          vel3d_adv_split,vel3d_adv_hor,vel3d_adv_ver, &
          turb_adv_split,turb_adv_hor,turb_adv_ver,   &
          calc_temp,calc_salt,                        &
          avmback,avhback,                            &
          ip_method,ip_ramp,                          &
          vel_check,min_vel,max_vel
```

8.4.2 postinit_3d - re-initialise some 3D after hotstart read.

INTERFACE:

```
subroutine postinit_3d(runtype,timestep,hotstart)
```

USES:

```
use domain, only: imin,imax,jmin,jmax, az,au,av
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: runtype
REALTYPE, intent(in)               :: timestep
logical, intent(in)                :: hotstart
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

This routine provides possibility to reset/initialize 3D variables to ensure that velocities are correctly set on land cells after read of a hotstart file. **LOCAL VARIABLES:**

```
integer                    :: i,j,rc
```

8.4.3 integrate_3d - calls to do 3D model integration

INTERFACE:

```
   subroutine integrate_3d(runtype,n)
   use getm_timers, only: tic, toc, TIM_INTEGR3D
#ifndef NO_BAROCLINIC
   use getm_timers, only: TIM_TEMPH, TIM_SALTH
#endif
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: runtype,n
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

This is a wrapper routine to call all 3D related subroutines. The call position for the `coordinates` routine depends on the compiler option `MUDFLAT`: If it is defined, then the call to `coordinates` construction is made such that drying and flooding is stable. If `MUDFLAT` is not defined, then the adaptive grids with Lagrangian component which are currently under development are supported. Both, drying and flooding and Lagrangian coordinates does not go together yet. The call sequence is as follows:

`start_macro`	initialising a 3d step	see page 167
`do_bdy_3d`	boundary conditions for θ and S	see page 154
`coordinates`	layer heights (`MUTFLAT` defined)	see page 114
`bottom_friction_3d`	bottom friction	see page 175
`do_internal_pressure`	internal pressure gradient	see page 144
`uu_momentum_3d`	layer-integrated u-velocity	see page 168
`vv_momentum_3d`	layer-integrated v-velocity	see page 170
`coordinates`	layer heights (`MUTFLAT` not defined)	see page 114
`ww_momentum_3d`	grid-related vertical velocity	see page 172
`uv_advect_3d`	momentum advection	see page 173
`uv_diffusion_3d`	momentum diffusion	see page 174
`stresses_3d`	stresses (for GOTM)	see page 181
`ss_nn`	shear and stratification (for GOTM)	see page 179
`gotm`	interface and call to GOTM	see page 182
`do_temperature`	potential temperature equation	see page 130
`do_salinity`	salinity equation	see page 134
`do_eqstate`	equation of state	see page 137
`do_spm`	suspended matter equation	see page 166
`do_getm_bio`	call to GOTM-BIO (not yet released)	
`slow_bottom_friction`	slow bottom friction	see page 176
`slow_terms`	sum of slow terms	see page 177
`stop_macro`	finishing a 3d step	see page 178

Several calls are only executed for certain compiler options. At each time step the call sequence for the horizontal momentum equations is changed in order to allow for higher order accuracy for the Coriolis rotation. **LOCAL VARIABLES:**

```
logical, save              :: ufirst=.true.
```

8.4.4 clean_3d - cleanup after 3D run

INTERFACE:

```
subroutine clean_3d()
IMPLICIT NONE
```

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

Here, a call to the routine `clean_variables_3d` which howewer does not do anything yet. **LOCAL VARIABLES:**

8.5 Fortran: Module Interface variables_3d - global 3D related variables (Source File: variables_3d.F90)

INTERFACE:

```
module variables_3d
```

DESCRIPTION:

This modules contains declarations for all variables related to 3D hydrodynamical calculations. Information about the calculation domain is included from the `domain` module. The variables are either statically defined in `static_3d.h` or dynamically allocated in `dynamic_declarations_3d.h`. The variables which need to be declared have the following dimensions, units and meanings:

`kmin`	2D	[-]	lowest index in T-point
`kumin`	2D	[-]	lowest index in U-point
`kvmin`	2D	[-]	lowest index in V-point
`kmin_pmz`	2D	[-]	lowest index in T-point (poor man's z-coordinate)
`kumin_pmz`	2D	[-]	lowest index in U-point (poor man's z-coordinate)
`kvmin_pmz`	2D	[-]	lowest index in V-point (poor man's z-coordinate)
`uu`	3D	$[\mathrm{m^2 s^{-1}}]$	layer integrated u transport p_k
`vv`	3D	$[\mathrm{m^2 s^{-1}}]$	layer integrated v transport q_k
`ww`	3D	$[\mathrm{m\,s^{-1}}]$	grid-related vertical velocity $\bar{w}_k$
`ho`	3D	[m]	old layer height in T-point
`hn`	3D	[m]	new layer height in T-point
`huo`	3D	[m]	old layer height in U-point
`hun`	3D	[m]	new layer height in U-point
`hvo`	3D	[m]	old layer height in V-point
`hvn`	3D	[m]	new layer height in V-point
`hcc`	3D	[-]	hydrostatic consistency index in T-points
`uuEx`	3D	$[\mathrm{m^2 s^{-2}}]$	sum of advection and diffusion for u-equation
`vvEx`	3D	$[\mathrm{m^2 s^{-2}}]$	sum of advection and diffusion for v-equation
`num`	3D	$[\mathrm{m^2 s^{-1}}]$	eddy viscosity on w-points ν_t
`nuh`	3D	$[\mathrm{m^2 s^{-1}}]$	eddy diffusivity on w-points ν_t'
`tke`	3D	$[\mathrm{m^2 s^{-2}}]$	turbulent kinetic energy k
`eps`	3D	$[\mathrm{m^2 s^{-3}}]$	turbulent dissipation rate ε
`SS`	3D	$[\mathrm{s^{-2}}]$	shear-frequency squared M^2
`NN`	3D	$[\mathrm{s^{-2}}]$	Brunt-Väisälä frequency squaredN^2
`S`	3D	[psu]	salinity S
`T`	3D	[°C]	potential temperature θ
`rad`	3D	$[\mathrm{W m^{-2}}]$	Short wave penetration
`rho`	3D	$[\mathrm{kg\,m^{-3}}]$	density rho
`buoy`	3D	$[\mathrm{m\,s^{-2}}]$	buoyancy b
`idpdx`	3D	$[\mathrm{m^2 s^{-2}}]$	x-component of internal pressure gradient
`idpdy`	3D	$[\mathrm{m^2 s^{-2}}]$	y-component of internal pressure gradient
`spm`	3D	$[\mathrm{kg\,m^{-3}}]$	suspended matter concentration
`spm_ws`	3D	$[\mathrm{m\,s^{-1}}]$	settling velocity of suspended matter
`spm_pool`	2D	$[\mathrm{kg\,m^{-2}}]$	bottom pool of suspended matter
`uadv`	3D	$[\mathrm{m\,s^{-1}}]$	interpolated x-component of momentum advection velocity
`vadv`	3D	$[\mathrm{m\,s^{-1}}]$	interpolated y-component of momentum advection velocity
`wadv`	3D	$[\mathrm{m\,s^{-1}}]$	interpolated vertical component of momentum advection velocity

huadv	3D	[m]	interpolated height of advective flux layer (x-component)
hvadv	3D	[m]	interpolated height of advective flux layer (y-component)
hoadv	3D	[m]	old height of advective finite volume cell
hnadv	3D	[m]	new height of advective finite volume cell
sseo	2D	[m]	sea surface elevation before macro time step (T-point)
ssen	2D	[m]	sea surface elevation after macro time step (T-point)
ssuo	2D	[m]	sea surface elevation before macro time step (U-point)
ssun	2D	[m]	sea surface elevation after macro time step (U-point)
ssvo	2D	[m]	sea surface elevation before macro time step (V-point)
ssvn	2D	[m]	sea surface elevation after macro time step (V-point)
rru	2D	$[\mathrm{m\,s^{-1}}]$	drag coefficient times curret speed in U-point
rrv	2D	$[\mathrm{m\,s^{-1}}]$	drag coefficient times curret speed in V-point
taus	2D	$[\mathrm{m^2 s^{-2}}]$	normalised surface stress (T-point)
taub	2D	$[\mathrm{m^2 s^{-2}}]$	normalised bottom stress (T-point)

It should be noted that depending on compiler options and runtype not all these variables are defined.
The module contains public subroutines to initialise (see `init_variables_3d`) and cleanup (see `clean_variables_3d`). **USES:**

```
   use domain,     only: imin,imax,jmin,jmax,kmax
   use field_manager
   IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   integer, parameter                  :: rk = kind(_ONE_)
   REALTYPE                            :: dt,cnpar=0.9
   REALTYPE                            :: avmback=_ZERO_,avhback=_ZERO_
   logical                             :: do_numerical_analyses=.false.
#ifdef STATIC
#include "static_3d.h"
#else
#include "dynamic_declarations_3d.h"
#endif

  REALTYPE, dimension(:,:,:), allocatable   :: numdis3d
  REALTYPE, dimension(:,:), allocatable     :: numdis2d
  REALTYPE, dimension(:,:,:), allocatable   :: nummix3d_S,nummix3d_T
  REALTYPE, dimension(:,:,:), allocatable   :: phymix3d_S,phymix3d_T
  REALTYPE, dimension(:,:), allocatable     :: nummix2d_S,nummix2d_T
  REALTYPE, dimension(:,:), allocatable     :: phymix2d_S,phymix2d_T

#ifdef GETM_BIO
   REALTYPE, allocatable              :: cc3d(:,:,:,:)
   REALTYPE, allocatable              :: ws3d(:,:,:,:)
#endif
#ifdef _FABM_
   REALTYPE, allocatable, dimension(:,:,:,:) :: fabm_pel,fabm_diag
   REALTYPE, allocatable, dimension(:,:,:)   :: fabm_ben,fabm_diag_hz
#endif
   integer                             :: size3d_field
   integer                             :: mem3d
   integer                             :: preadapt
```

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard

8.5.1 init_variables_3d - initialise 3D related stuff

INTERFACE:

```
subroutine init_variables_3d(runtype)
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: runtype
```

DESCRIPTION:

Dynamic allocation of memory for 3D related fields via `dynamic_allocations_3d.h` (unless the compiler option `STATIC` is set). Furthermore, most variables are initialised here. **LOCAL VARIABLES:**

```
integer                    :: rc
```

8.5.2 register_3d_variables() - register GETM variables. (Source File: variables_3d.F90)

INTERFACE:

```
subroutine register_3d_variables(fm,runtype)
```

DESCRIPTION:

USES:

```
KB   use variables_3d
  IMPLICIT NONE
```

INPUT PARAMETERS:

```
type (type_field_manager) :: fm
integer, intent(in)       :: runtype
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Jorn Bruggeman
```

LOCAL VARIABLES:

8.5.3 clean_variables_3d - cleanup after 3D run.

INTERFACE:

```
subroutine clean_variables_3d()
IMPLICIT NONE
```

DESCRIPTION:

This routine cleans up after a 3D integrationby doing nothing so far.

8.5.4 coordinates - defines the vertical coordinate (Source File: coordinates.F90)

INTERFACE:

```
subroutine coordinates(hotstart)
```

DESCRIPTION:

Here, the vertical layer distribution in T-, U- and V-points is updated during every macro time step. This is done for the old and the new layer thicknesses at every point. Calculation of the layer distribution in the U- and V-points is done indepently from the calculation in the T-points, since different methods for the calculation of the bathymetry values in the U- and V-points are possible, see routine `uv_depths` described on page 44.
The different methods for the vertical layer distribution are initialised and called to be chosen by the namelist paramter `vert_cord`:

`vert_cord=1`: sigma coordinates (section 8.5.5)
`vert_cord=2`: z-level (not coded yet)
`vert_cord=3`: general vertical coordinates (gvc, section 8.5.6)
`vert_cord=5`: adaptive vertical coordinates (section 8.5.7)

USES:

```
   use domain, only: imin,imax,jmin,jmax,kmax,H
#ifdef SLICE_MODEL
   use variables_3d, only: kvmin,hvo,hvn
#endif
   use getm_timers, only: tic, toc,TIM_COORDS
   use m3d
   use domain, only: vert_cord
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
    integer, intent(in)                 :: cord_type
    REALTYPE, intent(in)                :: cord_relax
    REALTYPE, intent(in)                :: maxdepth
   logical, intent(in)                 :: hotstart
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   logical, save   :: first=.true.
   integer         :: ii
    integer         :: preadapt=0
   integer          :: i,j,k
```

8.5.5 equidistant and zoomed sigma-coordinates

INTERFACE:

```
subroutine sigma_coordinates(first)
```

DESCRIPTION:

Here, the sigma coordinates layer distribution in T-, U- and V-points is calculated. The layer interfaces for each layer index have a fixed relative position σ_k in the water column, which may be even equidistant or non-equidistant, see equations (14) and (16). The surface and bottom zooming factors d_u and d_l are read in via the `domain` namelist in `getm.inp` as `ddu` and `ddl`. In the first call to the `sigma_coordinates`, the relative interface positions `dga` are calculated as a one-dimensional vector (in case of non-equidistant σ coordinates), and those are then multiplied with the water depths in all T-, U- and V-points to get the layer thicknesses. **USES:**

```
use domain, only: imin,imax,jmin,jmax,kmax,H,HU,HV
use domain, only: ga,ddu,ddl
use variables_3d, only: kmin,kumin,kvmin,ho,hn,huo,hun,hvo,hvn
use variables_3d, only: sseo,ssen,ssuo,ssun,ssvo,ssvn
IMPLICIT NONE
```

INPUT PARAMETERS:

```
logical, intent(in)                 :: first
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer          :: i,j,k,rc
REALTYPE         :: kmaxm1
logical, save    :: equiv_sigma=.false.
REALTYPE, save, dimension(:), allocatable  :: dga
```

8.5.6 general vertical coordinates

INTERFACE:

```
subroutine general_coordinates(first,cord_relax,maxdepth)
```

DESCRIPTION:

Here, the general vertical coordinates layer distribution in T-, U- and V-points is calculated. The general vertical coordinates as discussed in section 4.1, see equations (14) - (19), are basically an interpolation between equidistant and non-equaidistant σ coordinates. During the first call, a three-dimensional field `gga` containing the relative interface positions is calculated, which further down used together with the actual water depth in the T-, U- and V-points for calculating the updated old and new layer thicknesses.

USES:

```
use domain, only: ga,ddu,ddl,d_gamma,gamma_surf
use domain, only: imin,imax,jmin,jmax,kmax,H,HU,HV,az,au,av,min_depth
use variables_3d, only: dt,kmin,kumin,kvmin,ho,hn,huo,hun,hvo,hvn
use variables_3d, only: sseo,ssen,ssuo,ssun,ssvo,ssvn
$ use omp_lib
IMPLICIT NONE
```

INPUT PARAMETERS:

```
logical, intent(in)                :: first
REALTYPE, intent(in)               :: cord_relax
REALTYPE, intent(in)               :: maxdepth
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer         :: i,j,k,rc,kk
REALTYPE        :: alpha
REALTYPE        :: HH,zz,r
REALTYPE, save, dimension(:),     allocatable  :: dga,be,sig
REALTYPE, save, dimension(:,:,:), allocatable  :: gga
```

8.5.7 adaptive vertical coordinates

INTERFACE:

```
subroutine adaptive_coordinates(first,hotstart)
```

DESCRIPTION:

The vertical grid adaptivity is partially given by a vertical diffusion equation for the vertical layer positions, with diffusivities being proportional to shear, stratification and distance from the boundaries. In the horizontal, the grid can be smoothed with respect to z-levels, grid layer slope and density. Lagrangian tendency of the grid movement is supported. The adaptive terrain-following grid can be set to be an Eulerian-Lagrangian grid, a hybrid σ-ρ or σ-z grid and combinations of these with great flexibility. With this, internal flow structures such as thermoclines can be well resolved and followed by the grid. A set of idealised examples is presented in Hofmeister et al. (2009), which show that the introduced adaptive grid strategy reduces pressure gradient errors and numerical mixing significantly.
For the configuration of parameters, a seperate namelist file adaptcoord.inp has to be given with parameters as following:
faclag - Factor on Lagrangian coords., 0.le.faclag.le.1
facdif - Factor on thickness filter, 0.le.faclag.le.1
fachor - Factor on position filter, 0.le.faclag.le.1
cNN - dependence on stratification
cSS - dependence on shear
cdd - dep. on distance from surface and bottom
d_vel - Typical velocity difference for scaling cNN adaption
d_dens - Typical density difference for scaling cSS adaption
dsurf - reference value for surface/bottom distance [m]
tgrid - Time scale of grid adaptation [s]
preadapt - number of iterations for pre-adaptation

The parameters cNN,cSS,cdd are used for the vertical adaption and have to be less or equal 1 in sum. The difference to 1 is describing a background value which forces the coordinates back to a sigma distribution. The values ddu and ddl from the domain namelist are used for weighting the zooming to surface and bottom if cdd>0. The option preadapt allows for a pre-adaption of coordinates to the initial density field and bathymetry. The number defines the number of iterations (change coordinates, vertically advect tracer, calculate vertical gradients) used for the preadaption. The initial temperature and salinity fields are re-interpolated onto the adapted grid afterwards.
USES:

```
   use domain, only: ga,imin,imax,jmin,jmax,kmax,H,HU,HV,az,au,av
   use variables_3d, only: dt,kmin,kumin,kvmin,ho,hn,huo,hvo,hun,hvn
   use variables_3d, only: sseo,ssen,ssuo,ssun,ssvo,ssvn
   use variables_3d, only: kmin_pmz,kumin_pmz,kvmin_pmz
   use variables_3d, only: preadapt

  ADAPTIVE-BEGIN
   use  parameters,  only: g,rho_0
   use variables_3d, only: uu,vv,SS
#ifndef NO_BAROCLINIC
   use variables_3d, only: NN
   use variables_3d, only: rho
#endif
   use domain,       only: ddu,ddl
   use halo_zones, only: update_3d_halo,wait_halo
   use halo_zones, only: H_TAG,U_TAG,V_TAG
```

```
#if defined CURVILINEAR || defined SPHERICAL
   use domain,       only: dxv,dyu,arcd1
#else
   use domain,       only: dx,dy,ard1
#endif

 ADAPTIVE-END

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   logical, intent(in)                :: first
   logical, intent(in)                :: hotstart
```

OUTPUT PARAMETERS:

```
    integer, intent(out)               :: preadapt
```

REVISION HISTORY:

```
   Original author(s): Richard Hofmeister and Hans Burchard
```

LOCAL VARIABLES:

```
   integer         :: i,j,k,rc
   REALTYPE        :: kmaxm1
   REALTYPE        :: deltaiso
   REALTYPE, save, dimension(:),     allocatable  :: be
   REALTYPE, save, dimension(:),     allocatable  :: NNloc ! local NN vector
   REALTYPE, save, dimension(:),     allocatable  :: SSloc ! local SS vector
   REALTYPE, save, dimension(:),     allocatable  :: gaa   ! new relative coord.
   REALTYPE, save, dimension(:),     allocatable  :: gaaold! old relative coord.
   REALTYPE, save, dimension(:),     allocatable  :: aav ! total grid diffus.
   REALTYPE, save, dimension(:),     allocatable  :: avn ! NN-rel. grid diffus.
   REALTYPE, save, dimension(:),     allocatable  :: avs ! SS-rel. grid diffus.
   REALTYPE, save, dimension(:),     allocatable  :: avd ! dist.-rel. grid diff.
   REALTYPE, save, dimension(:,:,:), allocatable  :: zpos ! new pos. of z-levels
   REALTYPE, save, dimension(:,:,:), allocatable  :: zposo! old pos. of z-levels
   REALTYPE, save, dimension(:,:,:), allocatable  :: work2,work3
   REALTYPE     :: faclag=_ZERO_    ! Factor on Lagrangian coords., 0.le.faclag.le.1
   REALTYPE     :: facdif=3*_TENTH_ ! Factor on thickness filter,   0.le.faclag.le.1
   REALTYPE     :: fachor=_TENTH_   ! Factor on position filter,  0.le.faclag.le.1
   REALTYPE     :: faciso=_ZERO_    ! Factor for isopycnal tendency
   REALTYPE     :: depthmin=_ONE_/5
   REALTYPE     :: Ncrit=_ONE_/1000000
   integer      :: mhor=1    ! this number is experimental - it has to be 1 for now-
   integer      :: iw=2      ! stencil for isopycnal tendency
   REALTYPE     :: rm
   INTEGER      :: im,iii,jjj,ii
   integer      :: split=1           ! splits the vertical adaption into #split steps
   REALTYPE     :: c1ad=_ONE_/5      ! dependence on NN
   REALTYPE     :: c2ad=_ZERO_       ! dependence on SS
   REALTYPE     :: c3ad=_ONE_/5      ! distance from surface and bottom
   REALTYPE     :: c4ad=6*_TENTH_    ! background value
   REALTYPE     :: d_vel=_TENTH_     ! Typical value of absolute shear
   REALTYPE     :: d_dens=_HALF_     ! Typical value of BVF squared
```

```
   REALTYPE     :: dsurf=20*_ONE_    ! reference value for surface/bottom distance
   REALTYPE     :: tgrid=21600*_ONE_ ! Time scale of grid adaptation
   REALTYPE     :: dtgrid
   REALTYPE     :: aau(0:kmax),bu(0:kmax)
   REALTYPE     :: cu(0:kmax),du(0:kmax)
   REALTYPE     :: facupper=_ONE_
   REALTYPE     :: faclower=_ONE_
   REALTYPE     :: cNN,cSS,cdd,csum
   REALTYPE     :: cbg=6*_TENTH_
   REALTYPE     :: tfac_hor=3600*_ONE_ ! factor introducing a hor. adaption timescale = dt/tgrid_
   integer      :: iip

   integer,save :: count=0
      namelist /adapt_coord/   faclag,facdif,fachor,faciso,  &
                           depthmin,Ncrit, &
                           cNN,cSS,cdd,cbg,d_vel,d_dens, &
                           dsurf,tgrid,split,preadapt
#if (defined GETM_PARALLEL && defined INPUT_DIR)
   character(len=PATH_MAX)   :: input_dir=INPUT_DIR
#else
   character(len=PATH_MAX)   :: input_dir='./'
#endif
```

8.5.8 hcc_check - hydrostatic consistency criteria

INTERFACE:

```
subroutine hcc_check()
```

DESCRIPTION:

This diagnostic routine calculates the hydrostatic consistency h^c in each T-point and each layer. h^c is defined as:

$$h^c_{i,j,k} = \max\left\{|\partial_x z_k| \frac{\Delta x}{\frac{1}{2}(h_{i,j,k} + h_{i+1,j,k})}, |\partial_y z_k| \frac{\Delta y}{\frac{1}{2}(h_{i,j,k} + h_{i,j+1,k})}\right\}. \tag{108}$$

For the numerical calculation it is used here that Δx and Δy can be cancelled out each. For $h^c \leq 1$, the grid box is hydrostatically consistent, else it is called hydrostatically inconsistent. In the latter case, numerical problems can be expected for terrain-following coordinates when stratification is strong.
h^c is stored in the 3d netcdf output file. **USES:**

```
use domain, only: imin,imax,jmin,jmax,kmax,az,au,av,HU,HV
use variables_3d, only: hn,hun,hvn,hcc
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                 :: i,j,k
REALTYPE                :: du1,du2,dv1,dv2
REALTYPE                :: x,y
```

8.6 Fortran: Module Interface 3D advection (Source File: advection_3d.F90)

INTERFACE:

```
module advection_3d
```

DESCRIPTION:

This module does 3D advection of scalars. The module follows the same convention as the other modules in 'getm'. The module is initialised by calling 'init_advection_3d()'. In the time-loop 'do_advection_3d' is called. 'do_advection_3d' is a wrapper routine which - dependent on the actual advection scheme chosen - makes calls to the appropriate subroutines, which may be done as one-step or multiple-step schemes. The actual subroutines are coded in external FORTRAN files. New advection schemes are easily implemented - at least from a program point of view - since only this module needs to be changed. Additional work arrays can easily be added following the stencil given below. To add a new advection scheme three things must be done:

1. define a unique constant to identify the scheme (see e.g. `UPSTREAM` and `TVD`)
2. adopt the `select case` in `do_advection_3d` and
3. write the actual subroutine.

USES:

```
use domain, only: imin,imax,jmin,jmax,kmax
use advection
IMPLICIT NONE
private
```

PUBLIC DATA MEMBERS:

```
public init_advection_3d, do_advection_3d,print_adv_settings_3d
integer,public                          :: adv_ver_iterations=1
integer,public,parameter                :: HVSPLIT=3,W_TAG=33
character(len=64),public,parameter      :: adv_splits_3d(0:3) = &
         (/"no split: one 3D uvw step                      ",  &
           "full step splitting: u + v + w                 ",  &
           "half step splitting: u/2 + v/2 + w + v/2 + u/2",  &
           "hor/ver splitting: uv + w                      "/)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.6.1 init_advection_3d

INTERFACE:

```
subroutine init_advection_3d()
```

DESCRIPTION:

Allocates memory. **USES:**

```
IMPLICIT NONE
```

8.6.2 do_advection_3d - 3D advection schemes

INTERFACE:

```
subroutine do_advection_3d(dt,f,uu,vv,ww,hu,hv,ho,hn,    &
                           split,hscheme,vscheme,AH,tag, &
                           hires,advres)
```

DESCRIPTION:

Here, advection terms for all three-dimensional state variables are calculated by means of a finite-volume approach (an exception is the possibility to directly calculate the momentum advection by a one-step three-dimensional upstream scheme, see `uv_advection_3d`) and the advection step is carried out as a fractional advection time step. Those 3D variables may be defined on T-, U-, V- and W-points. The latter option is interesting for turbulent quantities. Inside this advection routine, it does not matter, wehre the advected variable is located on the grid. All finite volume fluxes and geometric coefficients need to be calculated before `do_advection_3d` is called.
Originally, this 3D advection routine has been written for tracer equations. There, after multiplying the layer-integrated and transformed to curvilinear coordinates tracer equation (40) with mn, the advective terms in this equation are discretised as follows.
First advection term in (40):

$$\left(mn\,\partial_{\mathcal{X}}\left(\frac{p_k c_k}{n}\right)\right)_{i,j} \approx \frac{p_{i,j,k}\tilde{c}^u_{i,j,k}\Delta y^u_{i,j} - p_{i-1,j,k}\tilde{c}^u_{i-1,j,k}\Delta y^u_{i-1,j}}{\Delta x^c_{i,j}\Delta y^c_{i,j}} \tag{109}$$

Second advection term in (40):

$$\left(mn\,\partial_{\mathcal{Y}}\left(\frac{q_k c_k}{m}\right)\right)_{i,j} \approx \frac{q_{i,j,k}\tilde{c}^v_{i,j,k}\Delta y^v_{i,j} - q_{i,j-1,k}\tilde{c}^v_{i,j-1,k}\Delta y^v_{i,j-1}}{\Delta x^c_{i,j}\Delta y^c_{i,j}} \tag{110}$$

Vertical advective fluxes in (40):

$$(\bar{w}_k\tilde{c}_k)_{i,j} \approx w_{i,j,k}\tilde{c}^w_{i,j,k}. \tag{111}$$

The interfacial concentrations $\tilde{c}_{i,j,k}$ are calculated according to upwind or higher order directional split schemes, which are discussed in detail below and in sections 7.4.2 and 8.6.4.
However, as said above, in the same way these routines may be applied to quantities on U-, V-, and W-points, if the transports are properly calculated.
There are various combinations of advection schemes possible.
The options for `split` are:

`split = NOSPLIT`:	no split (one 3D uvw step)
`split = FULLSPLIT`:	full step splitting (u + v + w)
`split = HALFSPLIT`:	half step splitting (u/2 + v/2 + w + v/2 + u/2)
`split = HVSPLIT`:	hor./ver. splitting (uv + w)

The options for the horizontal scheme `hscheme` are:

`scheme = NOADV:`	advection disabled
`scheme = UPSTREAM:`	first-order upstream (monotone)
`scheme = UPSTREAM_2DH:`	2DH upstream with forced monotonicity
`scheme = P2:`	third-order polynomial (non-monotone)
`scheme = SUPERBEE:`	second-order TVD (monotone)
`scheme = MUSCL:`	second-order TVD (monotone)
`scheme = P2_PDM:`	third-order ULTIMATE-QUICKEST (monotone)
`scheme = J7:`	2DH Arakawa J7
`scheme = FCT:`	2DH FCT with forced monotonicity
`scheme = P2_2DH:`	2DH P2 with forced monotonicity

The options for the vertical scheme `vscheme` are:

`scheme = NOADV:`	advection disabled
`scheme = UPSTREAM:`	first-order upstream (monotone)
`scheme = P2:`	third-order polynomial (non-monotone)
`scheme = SUPERBEE:`	second-order TVD (monotone)
`scheme = MUSCL:`	second-order TVD (monotone)
`scheme = P2_PDM:`	third-order ULTIMATE-QUICKEST (monotone)

With the compiler option `SLICE_MODEL`, the advection in meridional direction is not executed.
USES:

```
use halo_zones, only: update_3d_halo,wait_halo,D_TAG,H_TAG,U_TAG,V_TAG
use getm_timers, only: tic,toc,TIM_ADV3D,TIM_ADV3DH
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in)                                 :: dt,AH
REALTYPE,dimension(I3DFIELD),intent(in)             :: uu,vv,ww,ho,hn,hu,hv
integer,intent(in)                                  :: split,hscheme,vscheme,tag
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE,dimension(I3DFIELD),intent(inout)          :: f
```

OUTPUT PARAMETERS:

```
REALTYPE,dimension(I3DFIELD),target,intent(out),optional :: hires,advres
```

LOCAL VARIABLES:

```
type(t_adv_grid),pointer                        :: adv_grid
REALTYPE,dimension(I3DFIELD),target             :: fi,hi,adv
REALTYPE,dimension(:,:,:),pointer,contiguous :: p_hi,p_adv
integer                                         :: tag2d,i,j,k
```

8.6.3 print_adv_settings_3d

INTERFACE:

```
subroutine print_adv_settings_3d(split,hscheme,vscheme,AH)
```

DESCRIPTION:

Checks and prints out settings for 3D advection.
!USES IMPLICIT NONE **INPUT PARAMETERS:**

```
integer,intent(inout):: split
integer,intent(in)   :: hscheme,vscheme
REALTYPE,intent(in)  :: AH
```

LOCAL VARIABLES:

8.6.4 adv_split_w - vertical advection of 3D quantities

INTERFACE:

```
subroutine adv_split_w(dt,f,fi,hi,adv,ww,      &
                       splitfac,scheme,tag,az, &
                       itersmax)
Note (KK): Keep in sync with interface in advection_3d.F90
```

DESCRIPTION:

Executes an advection step in vertical direction. The 1D advection equation

$$h^n_{i,j,k}c^n_{i,j,k} = h^o_{i,j,k}c^o_{i,j,k} - \Delta t \left(w_{i,j,k}\tilde{c}^w_{i,j,k} - w_{i,j,k-1}\tilde{c}^w_{i,j,k-1} \right), \tag{112}$$

is accompanied by an fractional step for the 1D continuity equation

$$h^n_{i,j,k} = h^o_{i,j,k} - \Delta t \left(w_{i,j,k}\tilde{} w_{i,j,k-1} \right). \tag{113}$$

Here, n and o denote values before and after this operation, respectively, n denote intermediate values when other 1D advection steps come after this and o denotes intermediate values when other 1D advection steps came before this.
The interfacial fluxes $\tilde{c}^w_{i,j,k}$ are calculated by means of monotone and non-monotone schemes which are described in detail in section 7.4.7 on page 73. **USES:**

```
use domain, only: imin,imax,jmin,jmax,kmax,ioff,joff
use advection, only: adv_interfacial_reconstruction
use advection, only: NOADV,UPSTREAM
use advection_3d, only: W_TAG
use halo_zones, only: U_TAG,V_TAG
$ use omp_lib
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE,intent(in)                                :: dt,splitfac
REALTYPE,dimension(I3DFIELD),intent(in),target     :: f
REALTYPE,dimension(I3DFIELD),intent(in)            :: ww
integer,intent(in)                                 :: scheme,tag,itersmax
integer,dimension(E2DFIELD),intent(in)             :: az
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE,dimension(I3DFIELD),target,intent(inout) :: fi,hi,adv
```

LOCAL VARIABLES:

```
logical            :: iterate,use_limiter,allocated_aux
integer            :: i,j,k,kshift,it,iters,iters_new,rc
REALTYPE           :: itersm1,dti,dtik,hio,advn,fuu,fu,fd,splitfack
REALTYPE,dimension(:),allocatable        :: wflux
REALTYPE,dimension(:),allocatable,target :: cfl0
REALTYPE,dimension(:),pointer            :: fo,faux,fiaux,hiaux,advaux,cfls
REALTYPE,dimension(:),pointer            :: p_fiaux,p_hiaux,p_advaux
REALTYPE,dimension(:),pointer            :: p1d
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

8.7 Fortran: Module Interface temperature (Source File: temperature.F90)

INTERFACE:

```
module temperature
```

DESCRIPTION:

In this module, the temperature equation is processed by reading in the namelist `temp` and initialising the temperature field (this is done in `init_temperature`), and calculating the advection-diffusion-equation, which includes penetrating short-wave radiation as source term (see `do_temperature`).

USES:

```
use exceptions
use domain, only: imin,jmin,imax,kmax,jmax,H,az,dry_z
use domain, only: ill,ihl,jll,jhl
use domain, only: ilg,ihg,jlg,jhg
KB   use get_field, only: get_3d_field
use variables_3d, only: rk,T,rad,hn,kmin,A,g1,g2
use halo_zones, only: update_3d_halo,wait_halo,D_TAG,H_TAG
IMPLICIT NONE
private
```

PUBLIC DATA MEMBERS:

```
public init_temperature, do_temperature, init_temperature_field
!PRIVATE DATA MEMBERS:
integer                    :: temp_method=1,temp_format=2
character(len=PATH_MAX)    :: temp_file="t_and_s.nc"
integer                    :: temp_field_no=1
character(len=32)          :: temp_name='temp'
REALTYPE                   :: temp_const=20.
integer                    :: temp_adv_split=0
integer                    :: temp_adv_hor=1
integer                    :: temp_adv_ver=1
REALTYPE                   :: temp_AH=-_ONE_
integer                    :: attenuation_method=0,jerlov=1
character(len=PATH_MAX)    :: attenuation_file="attenuation.nc"
integer                    :: ncid=-1,A_id,g1_id,g2_id
integer, allocatable       :: varids(:)
character(len=50), allocatable :: varnames(:)
integer                    :: old_month=-1
REALTYPE                   :: A_const=0.58,g1_const=0.35,g2_const=23.0
REALTYPE                   :: swr_bot_refl_frac=-_ONE_
REALTYPE                   :: swr_min_bot_frac=0.01
integer                    :: temp_check=0
REALTYPE                   :: min_temp=-2.,max_temp=35.
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.7.1 init_temperature - initialisation of temperature

INTERFACE:

```
subroutine init_temperature()
```

DESCRIPTION:

Here, the temperature equation is initialised. First, the namelist `temp` is read from `getm.inp`. Then, depending on the `temp_method`, the temperature field is read from a hotstart file (`temp_method`=0), initialised with a constant value (`temp_method`=1), initialised and interpolated with horizontally homogeneous temperature from a given temperature profile (`temp_method`=2), or read in and interpolated from a 3D netCDF field (`temp_method`=3). Finally, a number of sanity checks are performed for the chosen temperature advection schemes. **USES:**

```
use advection, only: J7
use advection_3d, only: print_adv_settings_3d
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                   :: k,i,j,n
integer                   :: status
namelist /temp/ &
         temp_method,temp_const,temp_file,                &
         temp_format,temp_name,temp_field_no,             &
         temp_adv_split,temp_adv_hor,temp_adv_ver,temp_AH, &
         attenuation_method,attenuation_file,jerlov,      &
         A_const,g1_const,g2_const,                       &
         swr_bot_refl_frac, swr_min_bot_frac,             &
         temp_check,min_temp,max_temp
```

8.7.2 init_temperature_field - initialisation of temperature field

INTERFACE:

```
subroutine init_temperature_field()
```

DESCRIPTION:

Initialise the temperature field as specified with temp_method and exchange the HALO zones
USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

```
integer                   :: k,i,j,n
integer, parameter        :: nmax=10000
REALTYPE                  :: zlev(nmax),prof(nmax)
integer                   :: status
```

8.7.3 do_temperature - temperature equation

INTERFACE:

```
subroutine do_temperature(n)
```

DESCRIPTION:

Here, one time step for the temperature equation is performed. First, preparations for the call to the advection schemes are made, i.e. calculating the necessary metric coefficients. After the call to the advection schemes, which actually perform the advection (and horizontal diffusion) step as an operational split step, the solar radiation at the interfaces (`rad(k)`) is calculated from given surface radiation (`swr_loc`) by means of a double exponential approach, see equation (102) on page 99). An option to reflect part of the short wave radiation that reaches the bottom has been implemented. In very shallow waters - or with very clear waters - a significant part of the incoming radiation will reach the bottom. Setting swr_bot_refl_frac to a value between 0 and 1 will reflect this fraction of what ever the value of SWR is at the bottom. The default value of swr_bot_refl_frac is 0. The reflection is only done if the ratio between the surface and bottom values of SWR is greater than swr_min_bot_frac (default 0.01). Furthermore, the surface heat flux `sfl_loc` is given a value. The sea surface temperature is limited by the freezing point temperature (as a most primitive sea ice model). The next step is to set up the tri-diagonal matrix for calculating the new temperature by means of a semi-implicit central scheme for the vertical diffusion. Source terms which appear on the right hand sides are due to the divergence of the solar radiation at the interfaces. The subroutine is completed by solving the tri-diagonal linear equation by means of a tri-diagonal solver. **USES:**

```
use time, only: month,timestr
use advection_3d, only: do_advection_3d
use variables_3d, only: dt,cnpar,hn,ho,nuh,uu,vv,ww,hun,hvn,S
use domain,       only: imin,imax,jmin,jmax,kmax,az
use meteo,        only: swr,shf
use parameters,   only: rho_0,cp
use parameters, only: avmolt
use getm_timers, only: tic, toc, TIM_TEMP, TIM_MIXANALYSIS
use variables_3d, only: do_numerical_analyses
use variables_3d, only: nummix3d_T,nummix2d_T
use variables_3d, only: phymix3d_T,phymix2d_T
$ use omp_lib
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in) :: n
```

LOCAL VARIABLES:

```
integer                   :: i,j,k,rc
REALTYPE                  :: T2(I3DFIELD)
OMP-NOTE: The pointer declarations is to allow each omp thread to
 have its own work storage (over a vertical).
REALTYPE, POINTER         :: Res(:)
REALTYPE, POINTER         :: auxn(:),auxo(:)
REALTYPE, POINTER         :: a1(:),a2(:),a3(:),a4(:)
REALTYPE, POINTER         :: rad1d(:)
REALTYPE                  :: zz,swr_loc,shf_loc
REALTYPE                  :: swr_refl
REALTYPE                  :: rho_0_cpi
integer                   :: status
```

8.8 Fortran: Module Interface Salinity (Source File: salinity.F90)

INTERFACE:

```
module salinity
```

DESCRIPTION:

In this module, the salinity equation is processed by reading in the namelist `salt` and initialising the salinity field (this is done in `init_salinity`), and calculating the advection-diffusion-equation (see `do_salinity`). **USES:**

```
use exceptions
use domain, only: imin,jmin,imax,jmax,kmax,ioff,joff
use domain, only: H,az
KB   use get_field, only: get_3d_field
use variables_2d, only: fwf_int
use variables_3d, only: rk,S,hn,kmin
use halo_zones, only: update_3d_halo,wait_halo,D_TAG,H_TAG
IMPLICIT NONE
private
```

PUBLIC DATA MEMBERS:

```
public init_salinity, do_salinity, init_salinity_field
!PRIVATE DATA MEMBERS:
integer                    :: salt_method=1,salt_format=2
character(len=PATH_MAX)    :: salt_file="t_and_s.nc"
integer                    :: salt_field_no=1
character(len=32)          :: salt_name='salt'
REALTYPE                   :: salt_const=35*_ONE_
integer                    :: salt_adv_split=0
integer                    :: salt_adv_hor=1
integer                    :: salt_adv_ver=1
REALTYPE                   :: salt_AH=-_ONE_
integer                    :: salt_check=0
REALTYPE                   :: min_salt=_ZERO_,max_salt=40*_ONE_
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.8.1 init_salinity - initialisation of salinity

INTERFACE:

```
subroutine init_salinity()
```

DESCRIPTION:

Here, the salinity equation is initialised. First, the namelist `salt` is read from `getm.inp`. Then, depending on the `salt_method`, the salinity field is read from a hotstart file (`salt_method`=0), initialised with a constant value (`salt_method`=1), initialised and interpolated with horizontally homogeneous salinity from a given salinity profile (`salt_method`=2), or read in and interpolated from a 3D netCDF field (`salt_method`=3). Finally, a number of sanity checks are performed for the chosen salinity advection schemes.
Apart from this, there are various options for specific initial conditions which are selected by means of compiler options. **USES:**

```
use advection, only: J7
use advection_3d, only: print_adv_settings_3d
IMPLICIT NONE
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

```
integer                   :: i,j,k,n
integer                   :: status
NAMELIST /salt/                                              &
         salt_method,salt_const,salt_file,                   &
         salt_format,salt_name,salt_field_no,                &
         salt_adv_split,salt_adv_hor,salt_adv_ver,salt_AH, &
         salt_check,min_salt,max_salt
```

8.8.2 init_salinity_field - initialisation of the salinity field

INTERFACE:

```
subroutine init_salinity_field()
```

DESCRIPTION:

Initialisation of the salinity field as specified by salt_method and exchange of the HALO zones
USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

```
integer                   :: i,j,k,n
integer, parameter        :: nmax=10000
REALTYPE                  :: zlev(nmax),prof(nmax)
integer                   :: status
```

8.8.3 do_salinity - salinity equation

INTERFACE:

```
subroutine do_salinity(n)
```

DESCRIPTION:

Here, one time step for the salinity equation is performed. First, preparations for the call to the advection schemes are made, i.e. calculating the necessary metric coefficients. After the call to the advection schemes, which actually perform the advection (and horizontal diffusion) step as an operational split step, the tri-diagonal matrix for calculating the new salinity by means of a semi-implicit central scheme for the vertical diffusion is set up. There are no source terms on the right hand sides. The subroutine is completed by solving the tri-diagonal linear equation by means of a tri-diagonal solver.
Also here, there are some specific options for single test cases selected by compiler options. **USES:**

```
use advection_3d, only: do_advection_3d
use variables_3d, only: dt,cnpar,hn,ho,nuh,uu,vv,ww,hun,hvn
use domain,       only: imin,imax,jmin,jmax,kmax,az
use parameters, only: avmols
use getm_timers, only: tic, toc, TIM_SALT, TIM_MIXANALYSIS
use variables_3d, only: do_numerical_analyses
use variables_3d, only: nummix3d_S,nummix2d_S
use variables_3d, only: phymix3d_S,phymix2d_S
$ use omp_lib
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in) :: n
```

LOCAL VARIABLES:

```
integer                   :: i,j,k,rc
REALTYPE, POINTER         :: Res(:)
REALTYPE, POINTER         :: auxn(:),auxo(:)
REALTYPE, POINTER         :: a1(:),a2(:),a3(:),a4(:)
REALTYPE                  :: S2(I3DFIELD)
integer                   :: status
```

8.9 Fortran: Module Interface eqstate (Source File: eqstate.F90)

INTERFACE:

```
module eqstate
```

DESCRIPTION:

Documentation will follow when the equation of state calculations are updated. The idea is to use the respective routines from GOTM. **USES:**

```
use domain, only: imin,imax,jmin,jmax,kmax,az
use  parameters, only:   g,rho_0
use  variables_3d, only: T,S,rho
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
public init_eqstate, do_eqstate
!PRIVATE DATA MEMBERS:
integer                   :: eqstate_method=3
REALTYPE                  :: T0 = 10., S0 = 33.75, p0 = 0.
REALTYPE                  :: dtr0 = -0.17, dsr0 = 0.78
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

8.9.1 init_eqstate

INTERFACE:

```
subroutine init_eqstate()
IMPLICIT NONE
```

DESCRIPTION:

Reads the namelist and makes calls to the init functions of the various model components. **LOCAL VARIABLES:**

```
namelist /eqstate/ eqstate_method,T0,S0,p0,dtr0,dsr0
```

8.9.2 do_eqstate - equation of state

INTERFACE:

```
subroutine do_eqstate()
```

DESCRIPTION:

Here, the equation of state is calculated for every 3D grid point. **USES:**

```
use domain, only: imin,imax,jmin,jmax,kmax,az
use variables_3d, only: kmin,T,S,rho,buoy,hn,alpha,beta
use getm_timers, only: tic, toc, TIM_EQSTATE
$ use omp_lib
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: x
REALTYPE                  :: p1,s1,t1
REALTYPE                  :: th,densp
```

8.9.3 rho_from_theta_unesco80

INTERFACE:

```
subroutine rho_from_theta_unesco80(T,S,rho)
```

DESCRIPTION:

Here, the equation of state is calculated using the UNESCO 1980 code. **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)      :: T   ! potential temperature degC
REALTYPE, intent(in)      :: S   ! salinity PSU
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)     :: rho ! density [kg/m3]
```

REVISION HISTORY:

LOCAL VARIABLES:

```
REALTYPE                  :: x,T1,T2,T3,T4,T5,S1,S2,S3
```

8.9.4 rho_from_theta

INTERFACE:

```
subroutine rho_from_theta(s,th,p,dens0,densp)
```

DESCRIPTION:

Here, the equation of state is calculated Uses Jackett ea 2006 algorithm specific for potential temperature Checkvalue S=35 T=25 p=10000 rho_from_theta = 1062.53817
s : salinity (psu) th : potential temperature (deg C, ITS-90) p : gauge pressure (dbar) (absolute pressure - 10.1325 dbar)
rho_from_theta : in-situ density (kg m^{-3})
check value : rho_from_theta(20,20,1000) = 1017.728868019642
based on DRJ on 10/12/03 **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)      :: th  ! potential temperature degC
REALTYPE, intent(in)      :: s  ! in situ salinity PSU
REALTYPE, intent(in)      :: p  ! pressure in dbars
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)     :: dens0  ! density at 0.0 dbars
REALTYPE, intent(out)     :: densp  ! density at p dbars
```

REVISION HISTORY:

```
AS 2009 based on code provided by Jackett 2005
See the log for the module
!LOCAL VARIABLES
REALTYPE th2,sqrts,anum,aden,pth
```

8.9.5 eosall_from_theta

INTERFACE:

```
subroutine eosall_from_theta(s,th,p,rho_s,rho_th)
```

DESCRIPTION:

in-situ density and its derivatives (only 2) as functions of salinity, potential temperature and pressure as in Jackett, McDougall, Feistel, Wright and Griffies (2006), JAOT
s : salinity (psu) th : potential temperature (deg C, ITS-90) p : gauge pressure (dbar) (absolute pressure - 10.1325 dbar)
rho : in-situ density (kg m^{-3}) rho_s : partial derivative wrt s (kg m^{-3} psu^{-1}) rho_th : partial derivative wrt th (kg m^{-3} deg C^{-1})
check values : eosall_from_theta(20,20,1000,...) gives
rho = 1017.728868019642 rho_s = 0.7510471164699279 rho_th = -0.2570255211349140
based on DRJ on 10/12/03 **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)      :: th ! potential temperature degC
REALTYPE, intent(in)      :: s  ! in situ salinity PSU
REALTYPE, intent(in)      :: p  ! pressure in dbars
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)     :: rho_s   ! partial derivative wrt s
REALTYPE, intent(out)     :: rho_th  ! partial derivative wrt th
```

REVISION HISTORY:

```
AS 2009 based on code provided by Jackett 2005
See the log for the module
!LOCAL VARIABLES
REALTYPE        :: th2,sqrts,anum,aden,pth
REALTYPE        :: rho,anum_s,aden_s,anum_th,aden_th,rec_aden
```

8.10 Fortran: Module Interface internal_pressure (Source File: internal_pressure.F90)

INTERFACE:

```
module internal_pressure
```

DESCRIPTION:

In GETM, various methods are provided for the calculation of the internal pressure gradients terms in x- and y-direction. These terms which appear as layer-integrated terms in the equations for the layer-integrated momentum are for the eastward momentum p_k (see equation (26)):

$$h_k \left(\frac{1}{2} h_N (\partial_x^* b)_N + \sum_{j=k}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial_x^* b)_j \right) \tag{114}$$

and for the northward layer-integrated momentum q_k (see equation (27)):

$$h_k \left(\frac{1}{2} h_N (\partial_y^* b)_N + \sum_{j=k}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial_y^* b)_j \right) \tag{115}$$

The major problem is how to calculate the horizontal (with respect to isogeopotentials) buoyancy gradients $\partial_x^* b$ and $\partial_y^* b$, which need to be defined at the interfaces positioned vertically between two velocity points.
The methods for calculating the internal pressure gradient included in GETM are currently:

1. Method by *Mellor et al.* (1994), see routine `ip_blumberg_mellor`
2. Modified *Mellor et al.* (1994) method, exact for linear density profiles with z-dependence only, see routine `ip_blumberg_mellor_lin`
3. Calculation by mean of linear interpolation to z-levels, see routine `ip_z_interpol`
4. Method by *Song* (1998), see routine `ip_song_wright`
5. Method by *Chu and Fan* (2003), see routine `ip_chu_fan`
6. Method by *Shchepetkin and McWilliams* (2003), see routine `ip_shchepetkin_mcwilliams`
7. Method by *Stelling and van Kester* (1994), see routine `ip_stelling_vankester.F90`

It is possible, by setting the compiler option `SUBSTR_INI_PRESS`, to substract the initial pressure gradient from all pressure gradients. This is only advisable for strong stratification without any initial internal pressure gradients. In this case any non-zero values of the resulting numerical initial pressure gradient are due to discretisation errors. **USES:**

```
   use exceptions
   use domain, only: imin,imax,jmin,jmax,kmax,az,au,av,H,HU,HV
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxu,dyv
#else
   use domain, only: dx,dy
#endif
   use variables_3d, only: kmin,hun,hvn,idpdx,idpdy,buoy,ssun,ssvn,ssen
#ifdef MUDFLAT
   use variables_3d, only: hn=>ho
```

```
#else
   use variables_3d, only: hn
#endif
   IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   public init_internal_pressure, do_internal_pressure
   integer, public           :: ip_method=1
#ifdef STATIC
   REALTYPE                  :: zz(I3DFIELD)
#ifdef SUBSTR_INI_PRESS
   REALTYPE                  :: idpdx0(I3DFIELD),idpdy0(I3DFIELD)
#endif
#else
   REALTYPE, allocatable     :: zz(:,:,:)
#ifdef SUBSTR_INI_PRESS
   REALTYPE, allocatable     :: idpdx0(:,:,:),idpdy0(:,:,:)
#endif
#endif
  !PRIVATE DATA MEMBERS:
   integer, private, parameter         :: BLUMBERG_MELLOR=1
   integer, private, parameter         :: BLUMBERG_MELLOR_LIN=2
   integer, private, parameter         :: Z_INTERPOL=3
   integer, private, parameter         :: SONG_WRIGHT=4
   integer, private, parameter         :: CHU_FAN=5
   integer, private, parameter         :: SHCHEPETKIN_MCWILLIAMS=6
   integer, private, parameter         :: STELLING_VANKESTER=7
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

8.10.1 init_internal_pressure - initialising internal pressure gradient

INTERFACE:

```
subroutine init_internal_pressure()
IMPLICIT NONE
```

DESCRIPTION:

Here, some necessary memory is allocated (in case of the compiler option `STATIC`), and information is written to the log-file of the simulation.
!LOCAL VARIABLES integer :: rc

8.10.2 do_internal_pressure - internal pressure gradient

INTERFACE:

```
subroutine do_internal_pressure()
```

DESCRIPTION:

Here, the chosen internal pressure gradient method is selected and (in case that the compiler option `SUBSTR_INI_PRESS` is set), the initial pressure is calculated and subtracted from the updated internal pressure gradient.
If GETM is executed as slice model (compiler option `SLICE_MODEL` is set, the internal pressure gradient for $j = 2$ is copied to $j = 3$. **USES:**

```
use getm_timers, only: tic, toc, TIM_INTPRESS
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                 :: i,j,k
logical, save           :: first=.true.
```

8.10.3 ip_blumberg_mellor -

INTERFACE:

```
subroutine ip_blumberg_mellor()
```

DESCRIPTION:

Here, the internal part of the pressure gradient is discretised according to *Mellor et al.* (1994). The crucial part of this term, which is $(\partial_x^* b)_k$ (in the case of the u-equation), is discretised between two vertically adjacent velocity points:

$$\begin{aligned}
&\frac{1}{2}(h_{i,j,k} + h_{i,j,k+1})\,(m\,\partial_{\mathcal{X}}^* b)_{i,j,k} \\
&\approx \frac{1}{2}(h^u_{i,j,k} + h^u_{i,j,k+1}) \frac{\frac{1}{2}(b_{i+1,j,k+1} + b_{i+1,j,k}) - \frac{1}{2}(b_{i,j,k+1} + b_{i,j,k})}{\Delta x^u_{i,j}} \\
&- \frac{z^i_{i+1,j,k} - z^i_{i,j,k}}{\Delta x^u_{i,j}} \left(\frac{1}{2}(b_{i+1,j,k+1} + b_{i,j,k+1}) - \frac{1}{2}(b_{i+1,j,k} + b_{i,j,k}) \right),
\end{aligned} \tag{116}$$

where $z^i_{i,j,k}$ is the z-coordinate of the interface in the T-point above the grid box with the index (i, j, k).
The discretisation of $(\partial_y^* b)_k$ for the v-equation is done accordingly.
In this routine, as a first step, the interface heights are calculated in the T-points, in order to allow for the calculation of the coordinate slopes in the U- and V-points. In a second step, the expression (116) equivalent formulation for the y-direction are integrated up downwards, beginning from the surface. **USES:**

```
  use internal_pressure
$ use omp_lib
  IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard, Adolf Stips, Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: dxm1,dym1
REALTYPE                  :: grdl,grdu,buoyl,buoyu,prgr,dxz,dyz
```

8.10.4 ip_blumberg_mellor_lin

INTERFACE:

```
subroutine ip_blumberg_mellor_lin()
```

DESCRIPTION:

Here, the internal pressure gradient calculation is carried out on the basis of the same buoyancy stencil than in the method according to *Mellor et al.* (1994) (see routine `ip_blumberg_mellor`), but in such a way that the pressure gradient numerically vanishes for linear stratification without horizontal gradients.

$$\begin{aligned}
&\frac{1}{2}(h_{i,j,k}+h_{i,j,k+1})\,(m\,\partial^*_{\mathcal{X}}b)_{i,j,k}\\
&\approx \frac{1}{2}(h^u_{i,j,k}+h^u_{i,j,k+1})\left[\frac{\frac{1}{2}(b_{i+1,j,k+1}+b_{i+1,j,k})-\frac{1}{2}(b_{i,j,k+1}+b_{i,j,k})}{\Delta x^u_{i,j}}\right.\\
&\quad -\frac{1}{2}\left(\frac{\frac{1}{2}(z^c_{i+1,j,k+1}+z^c_{i+1,j,k})-\frac{1}{2}(z^c_{i,j,k+1}+z^c_{i,j,k})}{\Delta x^u_{i,j}}\right)\\
&\quad \left.\frac{1}{2}\left(\frac{b_{i+1,j,k+1}-b_{i+1,j,k}}{z^c_{i+1,j,k+1}-z^c_{i+1,j,k}}+\frac{b_{i,j,k+1}-b_{i,j,k}}{z^c_{i,j,k+1}-z^c_{i,j,k}}\right)\right],
\end{aligned} \tag{117}$$

where $z^c_{i,j,k}$ is the z-coordinate of the centre of the grid box with the index (i,j,k).
The discretisation of $(\partial^*_y b)_k$ for the v-equation is done accordingly. **USES:**

```
   use internal_pressure
   use variables_3d, only: kumin_pmz,kvmin_pmz
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: dxm1,dym1
REALTYPE                  :: prgr,dxzu,dxzl,dyzu,dyzl
REALTYPE                  :: dzr2,dzr1,dxru,dxrl,dyru,dyrl,aa,bb,cc
```

8.10.5 ip_z_interpol

INTERFACE:

```
subroutine ip_z_interpol()
```

DESCRIPTION:

Here, the horizontal gradients of buoyancy, $(\partial_x^* b)_k$ and $(\partial_y^* b)_k$, are directly calculated in z-coordinates by linearly interpolating the buoyancies in the vertical to the evaluation point (which is the interface vertically located between the velocity points). In the case that extrapolations become necessary near the sloping surface (or more likely) near the sloping bottom, then the last regular buoyancy value (surface value or bottom value) is used. **USES:**

```
   use internal_pressure
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,k, rc
REALTYPE                  :: dxm1,dym1
REALTYPE                  :: grdl,grdu,buoyl,prgr,dxz,dyz
integer                   :: kplus,kminus
REALTYPE, POINTER         :: zx(:)
REALTYPE                  :: buoyplus,buoyminus
```

8.10.6 ip_song_wright

INTERFACE:

```
subroutine ip_song_wright()
```

DESCRIPTION:

Here, the pressure gradient is calculating according to an energy-conserving method suggested by *Song* (1998), which for the pressure gradient in x-direction looks as:

$$\begin{aligned}
&\frac{1}{2}(h_{i,j,k}+h_{i,j,k+1})\,(m\,\partial^*_{\mathcal{X}}b)_{i,j,k}\\
&\approx \frac{\frac{1}{4}(b_{i+1,j,k+1}+b_{i+1,j,k})(h^c_{i+1,j,k+1}+h^c_{i+1,j,k})-\frac{1}{4}(b_{i,j,k+1}+b_{i,j,k})(h^c_{i,j,k+1}+h^c_{i,j,k})}{\Delta x^u_{i,j}}\\
&\quad-\left[\frac{1}{2}(b_{i+1,j,k+1}+b_{i,j,k+1})\frac{z^c_{i+1,j,k+1}-z^c_{i,j,k+1}}{\Delta x^u_{i,j}}\right.\\
&\quad\left.-\frac{1}{2}(b_{i+1,j,k}+b_{i,j,k})\frac{z^c_{i+1,j,k}-z^c_{i,j,k}}{\Delta x^u_{i,j}}\right],
\end{aligned}\tag{118}$$

where $z^c_{i,j,k}$ is the z-coordinate of the centre of the grid box with the index (i,j,k).
The discretisation of $(\partial^*_y b)_k$ for the v-equation is done accordingly. **USES:**

```
   use internal_pressure
$  use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: dxm1,dym1
REALTYPE                  :: grdl,grdu,buoyl,buoyu,prgr,dxz,dyz
```

8.10.7 ip_chu_fan

INTERFACE:

```
subroutine ip_chu_fan()
```

DESCRIPTION:

This routine calculates the internal pressure gradient based on the classical approach by *Mellor et al.* (1994), extended by the hydrostatic extension by *Chu and Fan* (2003). **USES:**

```
   use internal_pressure
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding & Adolf Stips
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: dxm1,dym1,x,y,x1,y1,hc
REALTYPE                  :: grdl,grdu,buoyl,buoyu,prgr,dxz,dyz
REALTYPE, PARAMETER       :: SIXTH=_ONE_/6
```

8.10.8 ip_shchepetkin_mcwilliams

INTERFACE:

```
subroutine ip_shchepetkin_mcwilliams()
```

DESCRIPTION:

Here, the pressure gradient is calculated according to the method and the algorithm suggested by Shchepetkin and McWilliams, 2003. This method uses a nonconservative Density-Jacobian scheme, based on cubic polynomial fits for the bouyancy "buoy" and "zz", the vertical position of rho-points, as functions of its respective array indices. The cubic polynomials are monotonized by using harmonic mean instead of linear averages to interpolate slopes. Exact anti-symmetry of the density Jacobian

$$J(rho, zz) = -J(zz, rho) \tag{119}$$

is retained for the density/bouyancy Jacobian in the pressure gradient formulation in x-direction for a non aligned vertical coordinate σ, the atmospheric pressure p_0 and the sea surface elevation η:

$$-\frac{1}{\rho_0}\partial_x p = \underbrace{-\frac{1}{\rho_0}\partial_x p_0 - g\partial_x \eta}_{uu\_momentum} + \underbrace{buoy(\eta)\partial_x \eta + \int_z^\eta J(buoy, zz)\mathrm{d}\sigma}_{idpdx} \tag{120}$$

Details about the calculation of the integral over the Jacobian in (120) can be found in Shchepetkin and McWilliams, 2003.
If parameter OneFifth (below) is set to zero, the scheme should become identical to standard Jacobian. **USES:**

```
  use internal_pressure
  use variables_3d, only: buoy,sseo
  use domain, only: H,az,au,av
$ use omp_lib
  IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Richard Hofmeister
```

LOCAL VARIABLES:

```
integer                 :: i,j,k
REALTYPE                :: dR(I3DFIELD)
REALTYPE                :: dZ(I3DFIELD)
REALTYPE                :: P(I3DFIELD)
REALTYPE                :: dxm1,dym1,cff,cff1,cff2
REALTYPE                :: AJ
REALTYPE                :: eps=1.e-10
REALTYPE                :: OneFifth = 0.2
REALTYPE                :: FC(I2DFIELD)
REALTYPE                :: dZx(I2DFIELD)
REALTYPE                :: dRx(I2DFIELD)
```

8.10.9 ip_stelling_vankester

INTERFACE:

```
subroutine ip_stelling_vankester()
```

DESCRIPTION:

Here, the horizontal gradients of buoyancy, $(\partial_x^* b)_k$ and $(\partial_y^* b)_k$, are calculated as suggested in Stelling and vanKester (1994). The horizontal gradient of buoyancy is calculated with defining kmax non-sloping control volumes in each water column and evaluating the horizontal gradients at the intersections of neighbouring control volumes. For each intersection, the buoyancy gradient is evaluated by linear interpolation of the buoyancy profile in the neighbour column at the T-depth of the actual column for both directions. The minimum of the absolute value of the buoyancy gradient for both directions is used then for the internal pressure calculation. If both gradients point inconcistently in different directions, the buoyancy gradient in an intersection does not contribute to the internal pressure (as happens for violated hydrostatic consistency and strong stratification)

USES:

```
   use internal_pressure
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Richard Hofmeister
```

LOCAL VARIABLES:

```
integer                  :: i,j,k,l,kcount, rc
REALTYPE                 :: dxm1,dym1
REALTYPE                 :: prgr,dyz,dzz,zlm
integer                  :: klower,kupper
integer                  :: lnum
REALTYPE                 :: db,dcn,dcm
logical                  :: changed
REALTYPE                 :: zltmp
REALTYPE                 :: buoyplus,buoyminus
REALTYPE                 :: zi(I3DFIELD)
REALTYPE, POINTER        :: zx(:)
REALTYPE, POINTER        :: zl(:)
REALTYPE, POINTER        :: dzl(:)
REALTYPE, POINTER        :: dzfrac(:)
integer, POINTER         :: lvel(:)
integer, POINTER         :: m(:)
integer, POINTER         :: n(:)
```

8.11 Fortran: Module Interface bdy_3d - 3D boundary conditions (Source File: bdy_3d.F90)

INTERFACE:

```
   module bdy_3d
```

DESCRIPTION:

Here, the three-dimensional boundary conditions for temperature and salinity are handled. **USES:**

```
   use halo_zones, only : H_TAG,U_TAG,V_TAG
   use domain, only: imin,jmin,imax,jmax,kmax,H,az,au,av
   use domain, only: nsbv,NWB,NNB,NEB,NSB,bdy_index
   use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
   use variables_3d
#ifdef _FABM_
   use getm_fabm, only: fabm_calc,model,fabm_pel,fabm_ben
#endif
   IMPLICIT NONE
   private
```

PUBLIC DATA MEMBERS:

```
   public init_bdy_3d, do_bdy_3d
   REALTYPE, public, allocatable       :: S_bdy(:,:),T_bdy(:,:)
#ifdef _FABM_
   REALTYPE, public, allocatable       :: bio_bdy(:,:,:)
   integer, public, allocatable        :: have_bio_bdy_values(:)
#endif
   logical,  public                    :: bdy3d_tmrlx=.false.
   REALTYPE, public                    :: bdy3d_tmrlx_ucut=_ONE_/50
   REALTYPE, public                    :: bdy3d_tmrlx_max=_ONE_/4
   REALTYPE, public                    :: bdy3d_tmrlx_min=_ZERO_
  !PRIVATE DATA MEMBERS:
   REALTYPE,         allocatable       :: bdyvertS(:), bdyvertT(:)
   REALTYPE,         allocatable       :: rlxcoef(:)
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

8.11.1 init_bdy_3d - initialising 3D boundary conditions

INTERFACE:

```
subroutine init_bdy_3d()
```

DESCRIPTION:

Here, the necessary fields `S_bdy` and `T_bdy` for salinity and temperature, respectively, are allocated.

USES:

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                   :: rc,i,j,k,n
```

8.11.2 do_bdy_3d - updating 3D boundary conditions

INTERFACE:

```
subroutine do_bdy_3d(tag,field)
```

DESCRIPTION:

Here, the boundary conditions for salinity and temperature are copied to the boundary points and relaxed to the near boundary points by means of the flow relaxation scheme by *Martinsen and Engedahl* (1987).
As an extention to the flow relaxation scheme, it is possible to relax the boundary point values to the specified boundary condition in time, thus giving more realistic situations especially for outgoing flow conditions. This nudging is implemented to depend on the local (3D) current velocity perpendicular to the boundary. For strong outflow, the boundary condition is turned off, while for inflows it is given a high impact. **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                 :: tag
```

INPUT/OUTPUT PARAMETERS:

```
REALTYPE, intent(inout)             :: field(I3DFIELD)
```

LOCAL VARIABLES:

```
integer                     :: i,j,k,l,n,o,ii,jj,kk
REALTYPE                    :: sp(1:4),rat
REALTYPE                    :: bdy3d_tmrlx_umin
REALTYPE                    :: wsum
```

8.12 Fortran: Module Interface rivers (Source File: rivers.F90)

INTERFACE:

```
   module rivers
```

DESCRIPTION:

This module includes support for river input. Rivers are treated the same way as meteorology, i.e. as external module to the hydrodynamic model itself. The module follows the same scheme as all other modules, i.e. `init_rivers` sets up necessary information, and `do_rivers` updates the relevant variables. `do_river` is called in `getm/integration.F90` between the `2d` and `3d` routines as it only updates the sea surface elevation (in `2d`) and sea surface elevation, and optionally salinity and temperature (in `3d`). At present the momentum of the river water is not include, the model however has a direct response to the river water because of the pressure gradient introduced.

USES:

```
   use domain, only: imin,jmin,imax,jmax,ioff,joff
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: H,az,kmax,arcd1
#else
   use domain, only: H,az,kmax,ard1
#endif
   use m2d, only: dtm
   use variables_2d, only: z
#ifndef NO_BAROCLINIC
   use m3d, only: calc_salt,calc_temp
   use variables_3d, only: hn,ssen,T,S
#endif
#ifdef GETM_BIO
   use bio, only: bio_calc
   use bio_var, only: numc
   use variables_3d, only: cc3d
#endif
#ifdef _FABM_
   use getm_fabm, only: model,fabm_pel
#endif
   IMPLICIT NONE
   private
```

PUBLIC DATA MEMBERS:

```
   public init_rivers, do_rivers, clean_rivers
#ifdef GETM_BIO
   public init_rivers_bio
#endif
#ifdef _FABM_
   public init_rivers_fabm
#endif
   integer, public                     :: river_method=0,nriver=0,rriver=0
   logical,public                      :: use_river_temp = .false.
   logical,public                      :: use_river_salt = .false.
   character(len=64), public           :: river_data="rivers.nc"
   character(len=64), public, allocatable  :: river_name(:)
   character(len=64), public, allocatable  :: real_river_name(:)
   integer, public, allocatable        :: ok(:)
```

```
   REALTYPE, public, allocatable       :: river_flow(:)
   REALTYPE, public, allocatable       :: river_salt(:)
   REALTYPE, public, allocatable       :: river_temp(:)
   integer, public                     :: river_ramp= -1
   REALTYPE, public                    :: river_factor= _ONE_
   REALTYPE, public,parameter          :: temp_missing=-9999.0
   REALTYPE, public,parameter          :: salt_missing=-9999.0
   integer,  public, allocatable       :: river_split(:)
#ifdef GETM_BIO
   REALTYPE, public, allocatable       :: river_bio(:,:)
   REALTYPE, public, parameter         :: bio_missing=-9999.0
#endif
#ifdef _FABM_
   REALTYPE, public, allocatable       :: river_fabm(:,:)
#endif
  !PRIVATE DATA MEMBERS:
   integer                   :: river_format=2
   character(len=64)         :: river_info="riverinfo.dat"
   integer, allocatable      :: ir(:),jr(:)
   REALTYPE, allocatable     :: rzl(:),rzu(:)
   REALTYPE, allocatable     :: irr(:)
   REALTYPE, allocatable     :: macro_height(:)
   REALTYPE, allocatable     :: flow_fraction(:),flow_fraction_rel(:)
   logical                   :: river_outflow_properties_follow_source_cell=.true.
```

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard

8.12.1 init_rivers

INTERFACE:

```
subroutine init_rivers
```

DESCRIPTION:

First of all, the namelist `rivers` is read from getm.F90 and a number of vectors with the length of `nriver` (number of rivers) is allocated. Then, by looping over all rivers, the ascii file `river_info` is read, and checked for consistency. The number of used rivers `rriver` is calculated and it is checked whether they are on land (which gives a warning) or not. When a river name occurs more than once in `river_info`, it means that its runoff is split among several grid boxed (for wide river mouths). **USES:**

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                   :: i,j,n,nn,ni,rc,m,iriver,jriver,numcells
logical                   :: outside,outsidehalo
REALTYPE                  :: bathy, area, total_weight
character(len=255)        :: line,xxx
NAMELIST /rivers/ &
         river_method,river_info,river_format,river_data,river_ramp, &
         river_factor,use_river_salt,use_river_temp,river_outflow_properties_follow_source_cel
```

8.12.2 read_river_info

INTERFACE:

```
subroutine read_river_info()
```

DESCRIPTION:

Read global indices for river positions, the river name and optionally depth range over which to distribute the water - zl:zu. Negative values imply 'bottom' for zl and 'surface' for zu. **USES:**

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
logical                  :: exist
integer                  :: unit = 25 ! kbk
integer                  :: n,rc,ios
character(len=255)       :: line
```

8.12.3 init_rivers_bio

INTERFACE:

```
subroutine init_rivers_bio()
```

DESCRIPTION:

First, memory for storing the biological loads from rivers is allocated. The variable - `river_bio` - is initialised to - `bio_missing`. **USES:**

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                   :: rc
```

8.12.4 init_rivers_fabm

INTERFACE:

```
subroutine init_rivers_fabm()
```

DESCRIPTION:

First, memory for storing the biological loads from rivers is allocated. The variable - `river_fabm` - is initialised to - variable- specific missing values obtained provided by FABM. **USES:**

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                   :: rc,m
```

8.12.5 do_rivers - updating river points

INTERFACE:

```
subroutine do_rivers(do_3d)
```

DESCRIPTION:

Here, the temperature, salinity, sea surface elevation and layer heights are updated in the river inflow grid boxes. Temperature and salinity are mixed with riverine values proportional to the old volume and the river inflow volume at that time step, sea surface elevation is simply increased by the inflow volume divided by the grid box area, and the layer heights are increased proportionally.

USES:

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
logical, intent(in)                :: do_3d
```

LOCAL VARIABLES:

```
integer                   :: i,j,k,m,n
integer                   :: kl,kh
integer, save             :: nn=0
REALTYPE                  :: ramp=_ONE_
REALTYPE                  :: rvol,height
REALTYPE                  :: river_depth,x
```

8.12.6 clean_rivers

INTERFACE:

```
subroutine clean_rivers
```

DESCRIPTION:

This routine closes the river handling by writing the integrated river run-off for each river to standard output. **USES:**

```
IMPLICIT NONE
```

LOCAL VARIABLES:

```
integer                 :: i,j,n
REALTYPE                :: tot=_ZERO_
```

8.13 Fortran: Module Interface suspended_matter (Source File: spm.F90)

INTERFACE:

```
module suspended_matter
```

DESCRIPTION:

This model for Suspended Particulate Matter (SPM) considers a single class of non-cohesive SPM particles that do not interact with the mean flow (no density effect of SPM is taken into account by default). The concentration C of SPM is modelled with the tracer equation. At the bottom, the net SPM flux is the residual of erosion and sedimentation fluxes:

$$-w_s C - \partial_z(\nu_t' \partial_z C) = F_e - F_s, \tag{121}$$

where erosion and sedimentation fluxes are modelled following *Krone* (1962) as functions of the bottom shear stress τ_b. In (121), w_s is a positive settling velocity. So far, GETM is only coded for constant settling velocities. The erosion flux is only non-zero when the bottom shear stress exceeds a critical shear stress τ_{ce}:

$$F_e = \begin{cases} \max\left\{\dfrac{c_e}{\rho_0}(|\tau_b| - \tau_{ce}), 0\right\}, & \text{for } B > 0 \text{ and } |\tau_b| > \tau_{ce} \\ \\ 0, & \text{else} \end{cases} \tag{122}$$

with c_e erosion constant with units $\mathrm{kg\,s\,m^{-4}}$ and the fluff layer SPM content B (see below). The sedimentation flux is only non-zero for bottom shear stresses smaller than a critical shear stress τ_{cs}. This flux is limited by the near bottom concentration C_b:

$$F_s = \max\left\{\frac{w_s C_b}{\tau_{cs}}(\tau_{cs} - |\tau_b|), 0\right\}. \tag{123}$$

Critical shear stresses for erosion and sedimentation (τ_{ce} and τ_{cs} have as units $\mathrm{N\,m^{-2}}$). However, the SPM flux between the water column and the bed may be switched off by setting `spm_method` in `spm.inp` to zero. A pool B of non-dynamic particulate matter (fluff layer) is assumed in order to take into account the effects of depletion of erodible material at the bottom. A horizontally homogeneous distribution with $B = B_0\ \mathrm{kg\,m^{-2}}$ is initially assumed. Sedimentation and erosion fill and empty this pool, respectively:

$$\partial_t(B) = F_s - F_e \tag{124}$$

and the erosion flux is constricted by the availability of SPM from the pool (see eq. (122)). The erosion and sedimentation fluxes are discretised using the quasi-implicit *Patankar* (1980) approach, which guarantees positivity of SPM, but only in the diffusion step, negative values might appear after the advection step, although these negative values should be small. The settling of SPM is linearly reduced towards zero when the water depth is between the critical and the minimum water depth. This is done by means of multiplication of the settling velocity with α, (see the definition in equation (5)).
It is possible to take into account the impact of sediments on density by setting `spm_dens` to `.true.` The modified density is computed as:

$$\rho = \rho_{T,S,p} + \left(1 - \frac{\rho_{T,S,p}}{\rho_{spm}}\right) C. \tag{125}$$

USES:

```
use exceptions
use domain, only: imin,jmin,imax,jmax,kmax,ioff,joff
```

```
#ifdef TRACER_POSITIVE
   use m2d, only : z,D
#endif
   use domain, only: H,az
   use parameters, only: rho_0,g
   use variables_3d, only: hn,taub,spm,spm_ws,spm_pool
   use halo_zones, only: update_3d_halo,wait_halo,D_TAG,H_TAG
   IMPLICIT NONE
   private
```

PUBLIC DATA MEMBERS:

```
   public init_spm, do_spm
   logical, public           :: spm_calc=.false.
   logical, public           :: spm_save=.true.
   logical, public           :: spm_hotstart=.false.
  !PRIVATE DATA MEMBERS:
   integer                 :: spm_method=1
   integer                 :: spm_init_method=1, spm_format=2
   character(len=PATH_MAX) :: spm_file="spm.nc"
   character(len=32)       :: spm_name='spm'
   integer                 :: spm_adv_split=0
   integer                 :: spm_adv_hor=1
   integer                 :: spm_adv_ver=1
   REALTYPE                :: spm_AH = -_ONE_
   REALTYPE                :: spm_const= _ZERO_
   REALTYPE                :: spm_init= _ZERO_
   integer                 :: spm_ws_method = 0
   REALTYPE                :: spm_ws_const=0.001
   REALTYPE                :: spm_erosion_const, spm_tauc_sedimentation
   REALTYPE                :: spm_tauc_erosion, spm_pool_init
   REALTYPE                :: spm_porosity=_ZERO_
   REALTYPE                :: spm_rho= 2650.
   logical                 :: spm_dens=.false.
   For erosion-sedimentation flux
   REALTYPE                :: Erosion_flux , Sedimentation_flux
   logical                 :: erosed_flux =.false.
  For flocculation (not yet in namelist)
   REALTYPE                :: spm_gellingC=0.08        !(g/l or kg/m3)
   REALTYPE                :: spm_part_density=2650.   !(g/l or kg/m3)
   integer                 :: spm_mfloc=4
```

REVISION HISTORY:

```
   Original author(s): Manuel Ruiz Villarreal, Karsten Bolding
                       and Hans Burchard
```

8.13.1 init_spm

INTERFACE:

```
subroutine init_spm(nml_file,runtype)
```

DESCRIPTION:

Here, the suspended matter equation is initialised. First, the namelist `spm` is read from `getm.inp`. Then, depending on the `spm_init_method`, the suspended matter field is read from a hotstart file (`spm_init_method`=0), initialised with a constant value (`spm_init_method`=1), initialised and interpolated with horizontally homogeneous suspended matter from a given suspended matter profile (`spm_init_method`=2), or read in and interpolated from a 3D netCDF field (`spm_init_method`=3). Then, some specifications for the SPM bottom pool are given, such as that there should be no initial SPM pool on tidal flats.
As the next step, a number of sanity checks is performed for the chosen suspended matter advection schemes.
Finally, the settling velocity is directly prescibed or calculated by means of the *Zanke* (1977) formula. **USES:**

```
For initialization of spm in intertidal flats
use domain,only: min_depth
use advection, only: J7
use advection_3d, only: print_adv_settings_3d
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)   :: nml_file
 logical                    :: hotstart_spm
integer, intent(in)            :: runtype
```

REVISION HISTORY:

```
See revision for the module
```

LOCAL VARIABLES:

```
integer         :: i,j,k,n
integer         :: rc
integer, parameter     :: nmax=100
REALTYPE        :: zlev(nmax),prof(nmax)
No initial pool of spm at intertidal flats
logical         :: intertidal_spm0=.false.
namelist /spm_nml/   spm_calc,spm_save,spm_method,spm_init_method,  &
                     spm_const,spm_format,spm_file,spm_name,        &
                     spm_adv_split,spm_adv_hor,spm_adv_ver,         &
                     spm_AH,spm_ws_method,spm_ws_const,             &
                     spm_erosion_const, spm_tauc_sedimentation,     &
                     spm_tauc_erosion, spm_porosity, spm_pool_init, &
                     spm_rho,spm_dens
```

8.13.2 do_spm - suspended matter equation

INTERFACE:

```
subroutine do_spm()
```

DESCRIPTION:

Here, one time step for the suspended matter equation is performed. First, preparations for the call to the advection schemes are made, i.e. calculating the necessary metric coefficients and the relevant vertical velocity, which is here composed of the grid-related vertical flow velocity and the settling velocity. Some lines of code allow here for consideration of flocculation processes. After the call to the advection schemes, which actually perform the advection (and horizontal diffusion) step as an operational split step, the fluxes between bottom SPM pool and the suspended matter in the water column are calculated. Afterwards, the tri-diagonal matrix for calculating the new suspended matter by means of a semi-implicit central scheme for the vertical diffusion is set up. There are no source terms on the right hand sides. The subroutine is completed by solving the tri-diagonal linear equation by means of a tri-diagonal solver.
Optionally, the density of the sediment-laden water may be corrected by the sediment density, see eq. (125).
Finally, some special settings for single test cases are made via compiler options. **USES:**

```
   use advection_3d, only: do_advection_3d
   use variables_3d, only: dt,cnpar,hun,hvn,ho,nuh,uu,vv,ww
#ifndef NO_BAROCLINIC
   use variables_3d, only: rho
#endif
   use domain, only: dry_z
   IMPLICIT NONE
```

LOCAL VARIABLES:

```
   integer         :: i,j,k,rc
   REALTYPE,dimension(I3DFIELD) :: wwadv
   REALTYPE        :: spmtot
   REALTYPE        :: Res(0:kmax)
   REALTYPE        :: auxn(1:kmax-1),auxo(1:kmax-1)
   REALTYPE        :: a1(0:kmax),a2(0:kmax)
   REALTYPE        :: a3(0:kmax),a4(0:kmax)
   REALTYPE        :: bed_flux
   REALTYPE        :: c
   REALTYPE        :: volCmud,volCpart
   integer         :: k2
   logical         :: patankar=.true.
#ifdef TRACER_POSITIVE
   logical         :: kk
#endif
```

8.13.3 start_macro - initialise the macro loop (Source File: start_macro.F90)

INTERFACE:

```
subroutine start_macro()
```

DESCRIPTION:

This routine needs to be called from `m3d` at the beginning of each macro time step. Here, the sea surface elevations at the before and after the macro time step are updated at the T-, U- and V-points.the sea surface elevations at the before and after the macro time step are updated at the T-, U- and V-points, their notation is `sseo`, `ssen`, `ssuo`, `ssun`, `ssvo` and `ssvn`, where `e`, `u` and `v` stand for T-, U- and V-point and `o` and `n` for old and new, respectively, see also the description of `variables_3d` in section 8.5 on page 108.
Furthermore, the vertically integrated transports `Uint` and `Vint` are here divided by the number of micro time steps per macro time step, `M`, in order to obtain the time-averaged transports.

USES:

```
use domain, only: imin,imax,jmin,jmax,H,HU,HV,min_depth
use m2d, only: z,Uint,Vint
use m3d, only: M
use variables_3d, only: sseo,ssen,ssuo,ssun,ssvo,ssvn,Dn,Dun,Dvn
use variables_3d, only: Uavg, Vavg
use getm_timers, only: tic, toc, TIM_STARTMCR
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j
REALTYPE                  :: split
```

8.13.4 uu_momentum_3d - x-momentum eq. (Source File: uu_momentum_3d.F90)

INTERFACE:

```
subroutine uu_momentum_3d(n,bdy3d)
```

DESCRIPTION:

Here, the budget equation for layer-averaged momentum in eastern direction, p_k, is calculated. The physical equation is given as equation (1), the layer-integrated equation as (26), and after curvilinear transformation as (38). In this routine, first the Coriolis rotation term, fq_k is calculated, either as direct transport averaging, or following *Espelid et al.* (2000) by using velocity averages (in case the compiler option `NEW_CORI` is set).
As a next step, explicit forcing terms (advection, diffusion, internal pressure gradient, surface stresses) are added up (into the variable `ex(k)`), the eddy viscosity is horizontally interpolated to the U-point, and the barotropic pressure gradient is calculated (the latter includes the pressure gradient correction for drying points, see section 5.5). Afterwards, the matrix is set up for each water column, and it is solved by means of a tri-diagonal matrix solver.
In case that the compiler option `STRUCTURE_FRICTION` is switched on, the frictional effect of structures in the water column is calculated by adding the quadratic frictional term $Cu\sqrt{u^2+v^2}$ (with a minus sign on the right hand side) numerically implicitly to the u-equation, with the friction coefficient C. The explicit part of this term, $C\sqrt{u^2+v^2}$, is calculated in the routine `structure_friction_3d.F90`.
Finally, the new velocity profile is shifted such that its vertical integral is identical to the time integral of the vertically integrated transport. If the compiler option `MUDFLAT` is defined, this fitting of profiles is made with respect to the new surface elevation, otherwise to the old surface elevation.
When GETM is run as a slice model (compiler option `SLICE_MODEL` is activated), the result for $j = 2$ is copied to $j = 3$. **USES:**

```
   use exceptions
   use parameters, only: g,avmmol,rho_0
   use domain, only: imin,imax,jmin,jmax,kmax,H,HU,min_depth
   use domain, only: dry_u,coru,au,av,az
#if defined CURVILINEAR || defined SPHERICAL
   use domain, only: dxu,arud1,dxx,dyc,dyx,dxc
#else
   use domain, only: dx,dy
#endif
   use variables_2d, only: Uint,D
   use bdy_3d, only: do_bdy_3d
   use variables_3d, only: dt,cnpar,kumin,uu,vv,huo,hun,hvo,uuEx,ww,hvn
   use variables_3d, only: num,nuh,sseo,ssun,rru
   use variables_3d, only: ssuo
#ifdef _MOMENTUM_TERMS_
   use variables_3d, only: tdv_u,cor_u,ipg_u,epg_u,vsd_u,hsd_u,adv_u
#endif
#ifdef STRUCTURE_FRICTION
   use variables_3d, only: sf
#endif
#ifndef NO_BAROCLINIC
   use variables_3d, only: idpdx
#endif
   use halo_zones, only: update_3d_halo,wait_halo,U_TAG
   use meteo, only: tausx,airp
   use m3d, only: ip_fac
```

```
   use m3d, only: vel_check,min_vel,max_vel
   use getm_timers, only: tic, toc, TIM_UUMOMENTUM, TIM_UUMOMENTUMH
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: n
   logical, intent(in)                 :: bdy3d
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer                   :: i,j,k,rc
#ifdef NEW_CORI
   REALTYPE,dimension(I3DFIELD) :: work3d
#endif
   REALTYPE, POINTER         :: dif(:)
   REALTYPE, POINTER         :: auxn(:),auxo(:)
   REALTYPE, POINTER         :: a1(:),a2(:)
   REALTYPE, POINTER         :: a3(:),a4(:)
   REALTYPE, POINTER         :: Res(:),ex(:)
   REALTYPE                  :: zp,zm,zx,ResInt,Diff,Vloc
   REALTYPE                  :: gamma=g*rho_0
   REALTYPE                  :: cord_curv=_ZERO_
   REALTYPE                  :: gammai,rho_0i
   integer                   :: status
```

8.13.5 vv_momentum_3d - y-momentum eq. (Source File: vv_momentum_3d.F90)

INTERFACE:

```
subroutine vv_momentum_3d(n,bdy3d)
```

DESCRIPTION:

Here, the budget equation for layer-averaged momentum in northern direction, q_k, is calculated. The physical equation is given as equation (2), the layer-integrated equation as (27), and after curvilinear transformation as (39). In this routine, first the Coriolis rotation term, fp_k is calculated, either as direct transport averaging, or following *Espelid et al.* (2000) by using velocity averages (in case the compiler option `NEW_CORI` is set).
As a next step, explicit forcing terms (advection, diffusion, internal pressure gradient, surface stresses) are added up (into the variable `ex(k)`), the eddy viscosity is horizontally interpolated to the V-point, and the barotropic pressure gradient is calculated (the latter includes the pressure gradient correction for drying points, see section 5.5). Afterwards, the matrix is set up for each water column, and it is solved by means of a tri-diagonal matrix solver.
In case that the compiler option `STRUCTURE_FRICTION` is switched on, the frictional effect of structures in the water column is calculated by adding the quadratic frictional term $Cv\sqrt{u^2+v^2}$ (with a minus sign on the right hand side) numerically implicitly to the v-equation, with the friction coefficient C. The explicit part of this term, $C\sqrt{u^2+v^2}$, is calculated in the routine `structure_friction_3d.F90`.
Finally, the new velocity profile is shifted such that its vertical integral is identical to the time integral of the vertically integrated transport. If the compiler option `MUDFLAT` is defined, this fitting of profiles is made with respect to the new surface elevation, otherwise to the old surface elevation.
When GETM is run as a slice model (compiler option `SLICE_MODEL` is activated), the result for $j = 2$ is copied to $j = 1$ and $j = 3$. **USES:**

```
   use exceptions
   use parameters, only: g,avmmol,rho_0
   use domain, only: imin,imax,jmin,jmax,kmax,H,HV,min_depth
   use domain, only: dry_v,corv,au,av,az
#if defined CURVILINEAR || defined SPHERICAL
   use domain, only: dyv,arvd1,dxc,dyx,dyc,dxx
#else
   use domain, only: dx,dy
#endif
   use variables_2d, only: Vint,D
   use bdy_3d, only: do_bdy_3d
   use variables_3d, only: dt,cnpar,kvmin,uu,vv,huo,hvo,hvn,vvEx,ww,hun
   use variables_3d, only: num,nuh,sseo,ssvn,rrv
   use variables_3d, only: ssvo
#ifdef _MOMENTUM_TERMS_
   use variables_3d, only: tdv_v,cor_v,ipg_v,epg_v,vsd_v,hsd_v,adv_v
#endif
#ifdef STRUCTURE_FRICTION
   use variables_3d, only: sf
#endif
#ifndef NO_BAROCLINIC
   use variables_3d, only: idpdy
#endif
   use halo_zones, only: update_3d_halo,wait_halo,V_TAG
   use meteo, only: tausy,airp
   use m3d, only: ip_fac
```

```
   use m3d, only: vel_check,min_vel,max_vel
   use getm_timers, only: tic, toc, TIM_VVMOMENTUM, TIM_VVMOMENTUMH
 $ use omp_lib
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: n
   logical, intent(in)                 :: bdy3d
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer                   :: i,j,k,rc
#ifdef NEW_CORI
   REALTYPE,dimension(I3DFIELD) :: work3d
#endif
   REALTYPE, POINTER         :: dif(:)
   REALTYPE, POINTER         :: auxn(:),auxo(:)
   REALTYPE, POINTER         :: a1(:),a2(:)
   REALTYPE, POINTER         :: a3(:),a4(:)
   REALTYPE, POINTER         :: Res(:),ex(:)
   REALTYPE                  :: zp,zm,zy,ResInt,Diff,Uloc
   REALTYPE                  :: gamma=g*rho_0
   REALTYPE                  :: cord_curv=_ZERO_
   REALTYPE                  :: gammai,rho_0i
   integer                   :: status
```

8.13.6 ww_momentum_3d - continuity eq. (Source File: ww_momentum_3d.F90)

INTERFACE:

```
subroutine ww_momentum_3d()
```

DESCRIPTION:

Here, the local continuity equation is calculated in order to obtain the grid-related vertical velocity $\bar{w}_k$. An layer-integrated equation for this quantity is given as equation (25) which has been derived from the differential formulation (3).
Since the kinematic boundary condition must hold (and is used for the derivation of (25)), the grid-related vertical velocity at the surface muzst be zero, i.e. $\bar{w}_{k_{\max}} = 0$. This is a good consistence check for the mode splitting, since this is only fulfilled if the vertically integrated continuity equation (which is the sea surface elevation equation calculated on the micro time step) and this local continuity equation are compatible.
The physical vertical velocity is then recalculated from the grid-related vertical velocity by means of (32), ... which should soon be coded in the routine `tow` in the directory `futils`. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: arcd1,dxv,dyu
#else
   use domain, only: dx,dy,ard1
#endif
   use variables_3d, only: dt,kmin,uu,vv,ww,ho,hn
   #define CALC_HALO_WW
#ifndef CALC_HALO_WW
   use domain, only: az
   use halo_zones, only: update_3d_halo,wait_halo,z_TAG
#endif
   use getm_timers, only: tic, toc, TIM_WWMOMENTUM, TIM_WWMOMENTUMH
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
REALTYPE                  :: dtm1
integer                   :: i,j,k
```

8.13.7 uv_advect_3d - 3D momentum advection (Source File: uv_advect_3d.F90)

INTERFACE:

```
subroutine uv_advect_3d()
```

DESCRIPTION:

Wrapper to prepare and do calls to routine `do_advection_3d` (see section 8.6.2 on page 123) to calculate the advection terms of the 3D velocities.
If `save_numerical_analyses` is set to `.true.`, the numerical dissipation is calculated using the method suggested by *Burchard* (2012). **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax,az,au,av,ax
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxv,dyu
#else
   use domain, only: dx,dy
#endif
   use m3d, only: vel3d_adv_split,vel3d_adv_hor,vel3d_adv_ver
   use variables_3d, only: dt,uu,vv,ww,ho,hn,hun,hvn,uuEx,vvEx
   use advection, only: NOADV,UPSTREAM,J7
   use advection_3d, only: do_advection_3d
   use halo_zones, only: update_3d_halo,wait_halo,U_TAG,V_TAG
   use variables_3d, only: do_numerical_analyses
   use variables_3d, only: numdis3d,numdis2d
#ifdef _MOMENTUM_TERMS_
   use variables_3d, only: adv_u,adv_v
#endif
   use getm_timers, only: tic,toc,TIM_UVADV3D,TIM_UVADV3DH
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer                                   :: i,j,k
   REALTYPE,dimension(I3DFIELD)              :: fadv3d,uuadv,vvadv,wwadv,huadv,hvadv
   REALTYPE,dimension(I3DFIELD),target       :: hnadv
   REALTYPE,dimension(:,:,:),pointer,contiguous :: phadv
   REALTYPE,dimension(I3DFIELD)              :: work3d,hires
```

8.13.8 uv_diffusion_3d - lateral diffusion of 3D velocity (Source File: uv_diffusion_3d.F90)

INTERFACE:

```
subroutine uv_diffusion_3d()
```

DESCRIPTION:

This wrapper calls routine `uv_diff_2dh` (see section 7.4.15 on page 84) for each layer. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax
   use m2d, only: uv_diff_2dh
   use m2d, only: Am
   use variables_3d, only: uu,vv,uuEx,vvEx,hn,hun,hvn
#ifdef _MOMENTUM_TERMS_
   use variables_3d, only: hsd_u,hsd_v
#endif
   use getm_timers, only: tic, toc, TIM_UVDIFF3D
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Knut Klingbeil
```

LOCAL VARIABLES:

```
integer :: i,j,k
```

8.13.9 bottom_friction_3d - bottom friction (Source File: bottom_friction_3d.F90)

INTERFACE:

```
subroutine bottom_friction_3d
```

DESCRIPTION:

Based on the assumption that the velocity distribution in the bottom layer is logarithmic, the product of the drag coefficient with the absolute value of the current speed in the bottom layer,

$$r\sqrt{u_b^2 + v_b^2} \tag{126}$$

with the velocity components of the bottom layer, u_b and v_b, and the drag coefficient

$$r = \left(\frac{\kappa}{\ln\left(\frac{0.5h_1 + z_0^b}{z_0^b}\right)} \right)^2 , \tag{127}$$

is calculated and provided as output parameters `rru` (for U-points) and `rrv` (for V-points). The layer height h_1 in (127) is set to the thickness of the bottom layer in the respective U- or V-point. There are some experimental options for the interested user included here. It is possible to change the interpolation of u to V-points and of v to U-points from velocity-based interpolation (as done presently) to transport-based averaging (commented out). Furthermore, the user may activate some outcommented lines which allow the consideration of flow-depending bottom roughness length z_0^b according to (81), see page 81.
For a derivation of (127), see section 5.4 on page 30. **USES:**

```
   use parameters, only: kappa,avmmol
   use domain, only: imin,imax,jmin,jmax,kmax,au,av,min_depth
   use variables_2d, only: zub,zvb,zub0,zvb0
   use variables_3d, only: kumin,kvmin,uu,vv,huo,hun,hvo,hvn,rru,rrv
   use getm_timers, only: tic, toc, TIM_BOTTFRICT3D
$ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,kk
REALTYPE                  :: r,hh,fricvel
logical, save             :: first=.true.
REALTYPE                  :: uuloc(I2DFIELD)
REALTYPE                  :: uvloc(I2DFIELD)
REALTYPE                  :: vuloc(I2DFIELD)
REALTYPE                  :: vvloc(I2DFIELD)
```

8.13.10 slow_bottom_friction - slow bed friction (Source File: slow_bottom_friction.F90)

INTERFACE:

```
subroutine slow_bottom_friction
```

DESCRIPTION:

This routine basically calculates the bed friction, as it would come out if the vertically and macro timestep averaged velocity would be used. The output of this subroutine is thus $R\sqrt{u^2+v^2}$ on the U-points (see variable `ruu`) and on the V-points (see `rvv`) with the vertically and macro timestep averaged velocity components on the old time step, u and v, which are in the code denoted by `Ui` and `Vi`, respectively. The drag coefficient R is given by eq. (71) on page 53. The results for the variables `ruu` and `rvv` will then be used in the routine `slow_terms` described on page 177 for the calculation of the slow terms S_F^x and S_F^y, see section 7.1.

USES:

```
use parameters, only: kappa
use domain, only: imin,imax,jmin,jmax,HU,HV,min_depth,au,av
use variables_2d, only: zub,zvb,ru,rv,Uinto,Vinto
use variables_3d, only: ssuo,ssun,ssvo,ssvn
use getm_timers, only: tic, toc, TIM_SLOWBFRICT
use exceptions, only: getm_error
$ use omp_lib
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j
REALTYPE                  :: uloc,vloc,HH
logical,save              :: first=.true.
REALTYPE                  :: Ui(I2DFIELD)
REALTYPE                  :: Vi(I2DFIELD)
REALTYPE                  :: ruu(I2DFIELD)
REALTYPE                  :: rvv(I2DFIELD)
```

8.13.11 slow_terms - calculation of slow terms (Source File: slow_terms.F90)

INTERFACE:

```
subroutine slow_terms
```

DESCRIPTION:

Here, the calculation of the so-called slow terms (which are the interaction terms between the barotropic and the baroclinic mode) is completed. The mathematical form of these slow terms is given by equations (63) - (70), see section 7.1. These calculations have been prepared in the routines `integrate_3d` and `slow_bottom_friction`. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax,HU,HV,au,av
   use variables_2d, only: Uint,Vint,UEx,VEx,Slru,Slrv,SlUx,SlVx,ru,rv
   use variables_3d, only: kumin,kvmin,uu,vv,huo,hun,hvo,hvn
   use variables_3d, only: ssuo,ssun,ssvo,ssvn,uuEx,vvEx,rru,rrv
   use m3d, only: ip_fac
   use getm_timers, only: tic, toc, TIM_SLOWTERMS
#ifndef NO_BAROCLINIC
   use variables_3d, only: idpdx,idpdy
#endif
#ifdef STRUCTURE_FRICTION
   use variables_3d, only: sf
#endif
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: vertsum
```

8.13.12 stop_macro - terminates the macro loop (Source File: stop_macro.F90)

INTERFACE:

```
subroutine stop_macro
```

DESCRIPTION:

This routine should be called from `m3d` at the end of each macro time step in order to copy the vertically interated and temporally averaged transports to old values `Uinto` and `Vinto`, and to reinitialise the transports `Uint` and `Vint` to zero. **USES:**

```
use variables_2d, only: Uint,Uinto,Vint,Vinto
use getm_timers, only: tic, toc, TIM_STOPMCR
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

8.13.13 ss_nn - calculates shear and buoyancy frequency (Source File: ss_nn.F90)

INTERFACE:

```
subroutine ss_nn()
```

DESCRIPTION:

Here, the shear frequency squared, $M^2 = (\partial_z u)^2 + (\partial_z v)^2$, and the buoyancy frequency squared, $N^2 = \partial_z b$, with buoyancy b from (4) are calculated. For both calculations, two alternative methods are coded. The two straight-forward methods which are explained first, do both have the disadvantage of generating numerical instabilities. The straight-forward way for calculating M^2 is as follows:

$$\begin{aligned}(M^2)_{i,j,k} \approx \frac{1}{2}\Bigg(&\left(\frac{u_{i,j,k+1}-u_{i,j,k}}{\frac{1}{2}(h^u_{i,j,k+1}+h^u_{i,j,k})}\right)^2 + \left(\frac{u_{i-1,j,k+1}-u_{i-1,j,k}}{\frac{1}{2}(h^u_{i-1,j,k+1}+h^u_{i-1,j,k})}\right)^2 \\ &+ \left(\frac{v_{i,j,k+1}-v_{i,j,k}}{\frac{1}{2}(h^v_{i,j,k+1}+h^v_{i,j,k})}\right)^2 + \left(\frac{v_{i,j-1,k+1}-v_{i,j-1,k}}{\frac{1}{2}(h^v_{i,j-1,k+1}+h^v_{i,j-1,k})}\right)^2\Bigg)\end{aligned} \tag{128}$$

Burchard (2002a) developed a new scheme, which guarantees that the mean kinetic energy which is dissipated from the mean flow equals the shear production of turbulent kinetic energy. Therefore, this scheme should be numerically more stable than (128):

$$\begin{aligned}(M^2)_{i,j,k} \approx \frac{1}{2}\Bigg(&\frac{\frac{1}{2}(\nu_{i,j,k}+\nu_{i+1,j,k})(u_{i,j,k+1}-u_{i,j,k})^2}{\frac{1}{2}(h^u_{i,j,k+1}+h^u_{i,j,k})} \\ &+ \frac{\frac{1}{2}(\nu_{i-1,j,k}+\nu_{i,j,k})(u_{i-1,j,k+1}-u_{i-1,j,k})^2}{\frac{1}{2}(h^u_{i-1,j,k+1}+h^u_{i-1,j,k})} \\ &+ \frac{\frac{1}{2}(\nu_{i,j,k}+\nu_{i,j+1,k})(v_{i,j,k+1}-v_{i,j,k})^2}{\frac{1}{2}(h^v_{i,j,k+1}+h^v_{i,j,k})} \\ &+ \frac{\frac{1}{2}(\nu_{i,j-1,k}+\nu_{i,j,k})(v_{i,j-1,k+1}-v_{i,j-1,k})^2}{\frac{1}{2}(h^v_{i,j-1,k+1}+h^v_{i,j-1,k})}\Bigg) \\ &\cdot \left(\frac{1}{2}\left(h^c_{i,j,k}+h^c_{i,j,k+1}\right)\nu_{i,j,k}\right)^{-1}\end{aligned} \tag{129}$$

The straight-forward discretisation of N^2 is given by

$$(N^2)_{i,j,k} \approx \frac{b_{i,j,k+1}-b_{i,j,k}}{\frac{1}{2}(h^t_{i,j,k+1}+h^t_{i,j,k})}. \tag{130}$$

In some cases, together with the straight-forward discretisation of the shear squared, (128), this did not produce stable numerical results. The reason for this might be that the velocities involved in the calculation for the shear squared do depend on the buoyancies in the two neighbouring T-points such that the straight-forward method (130) leads to an inconsistency. However, other experiments with the energy-conserving discretisation of the shear stress squared, (129) and the straight-forward discretisation of N^2, (130), produced numerically stable results.
Most stable results have been obtained with a weighted average for the N^2 calculation:

$$(N^2)_{i,j,k} \approx \frac{1}{6}\left(2\frac{b_{i,j,k+1}-b_{i,j,k}}{\frac{1}{2}(h^t_{i,j,k+1}+h^t_{i,j,k})}\right.$$
$$+\frac{b_{i+1,j,k+1}-b_{i+1,j,k}}{\frac{1}{2}(h^t_{i+1,j,k+1}+h^t_{i+1,j,k})}+\frac{b_{i-1,j,k+1}-b_{i-1,j,k}}{\frac{1}{2}(h^t_{i-1,j,k+1}+h^t_{i-1,j,k})} \tag{131}$$
$$\left.+\frac{b_{i,j+1,k+1}-b_{i,j+1,k}}{\frac{1}{2}(h^t_{i,j+1,k+1}+h^t_{i,j+1,k})}+\frac{b_{i,j-1,k+1}-b_{i,j-1,k}}{\frac{1}{2}(h^t_{i,j-1,k+1}+h^t_{i,j-1,k})}\right).$$

These stability issues need to be further investigated in the future. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax,au,av,az
   use variables_3d, only: kmin,kumin,hn,uu,hun,kvmin,vv,hvn,SS,num
   use parameters, only: g,rho_0
#ifndef NO_BAROCLINIC
   use variables_3d, only: NN,buoy,T,S
#ifndef _OLD_BVF_
   use variables_3d, only: alpha,beta
#endif
#endif
   use getm_timers, only: tic, toc, TIM_SSNN
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer                   :: i,j,k,nb
   REALTYPE                  :: dz,NNc,ttt
   REALTYPE                  :: NNe,NNw,NNn,NNs
   REALTYPE, parameter       :: small_bvf = 1.d-10
#ifdef _SMOOTH_BVF_VERT_
   REALTYPE                  :: below,center,above
#endif
```

8.13.14 stresses_3d - bottom and surface stresses (Source File: stresses_3d.F90)

INTERFACE:

```
subroutine stresses_3d
```

DESCRIPTION:

As preparation of the call to `do_turbulence` in the routine `gotm`, see section 8.13.15, the normalised surface and bottom stresses, τ_s/ρ_0 (variable `taus`) and τ_b/ρ_0 (variable `taub`), respectively, are calculated and interpolated to the T-points. Input parameters to this routine are `rru` and tt rrv, which contain $r\sqrt{u^2+v^2}$ for the U- and V-points, respectively. The modules of the surface and bottom stress vectors are calculated then by means of taking the square root of the sum of the squares of the stess components. In a similar way also the x- and y-components of the bottom stress are computed for output. **USES:**

```
use parameters, only: rho_0
use domain, only: az,au,av,imin,imax,jmin,jmax
use variables_3d, only: kumin,kvmin,uu,vv,hun,hvn,rru,rrv
use variables_3d, only: taus,taubx,tauby,taub
use meteo, only: tausx,tausy
use halo_zones, only : update_2d_halo,wait_halo,z_TAG
use getm_timers, only: tic, toc, TIM_STRESSES3D, TIM_STRESSES3DH
$ use omp_lib
IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: i,j,k,ku1,ku2,kv1,kv2
REALTYPE                  :: rho_0i
```

8.13.15 gotm - a wrapper to call GOTM (Source File: gotm.F90)

INTERFACE:

```
subroutine gotm()
```

DESCRIPTION:

Here, the turbulence module of the General Ocean Turbulence Model (GOTM, see `www.gotm.net` and *Umlauf et al.* (2005)) is called. First, all necessary parameters are transformed to suit with a 1D water column model, i.e., 3D fields are transformed to a vertical vector, 2D horizontal fields are converted to a scalar. The transformed 3D fields are the layer heights `hn` $\rightarrow$ `h`, the shear squared `SS` $\rightarrow$ `SS1d`, the buoyancy frequency squared `NN` $\rightarrow$ `NN1d`, the turbulent kinetic energy `tke` $\rightarrow$ `tke1d`, the dissipation rate `eps` $\rightarrow$ `eps1d` (from which the integral length scale `L1d` is calculated), the eddy viscosity `num` $\rightarrow$ `num1d`, and the eddy diffusivity `nuh` $\rightarrow$ `nuh1d`. The scalars are the surface and bottom friction velocities, `u_taus` and `u_taub`, respectively, the surface roughness parameter `z0s` (which is currently hard-coded), and the bottom roughess parameter `z0b`. Then, the GOTM turbulence module `do_turbulence` is called with all the transformed parameters discussed above. Finally, the vertical vectors `tke1d`, `eps1d`, `num1d` and `nuh1d` are transformed back to 3D fields.
In case that the compiler option `STRUCTURE_FRICTION` is switched on, the additional turbulence production by structures in the water column is calculated by calculating the total production as

$$P_{tot} = P + C\left(u^2 + v^2\right)^{3/2}, \tag{132}$$

with the shear production P, and the structure friction coefficient C. The latter is calculated in the routine `structure_friction_3d.F90`.
There are furthermore a number of compiler options provided, e.g. for an older GOTM version, for barotropic calcuations, and for simple parabolic viscosity profiles circumventing the GOTM turbulence module. **USES:**

```
   use halo_zones, only: update_3d_halo,wait_halo,H_TAG
   use domain, only: imin,imax,jmin,jmax,kmax,az,min_depth,crit_depth
   use variables_2d, only: D,zub,zvb,z
   use variables_3d, only: dt,kmin,ho,hn,tke,eps,SS,num,taus,taub
#ifndef NO_BAROCLINIC
   use variables_3d, only: NN,nuh
#endif
   use variables_3d, only: avmback,avhback
#ifdef STRUCTURE_FRICTION
   use variables_3d, only: uu,vv,hun,hvn,sf
#endif
   use turbulence, only: do_turbulence,cde
   use turbulence, only: tke1d => tke, eps1d => eps, L1d => L
   use turbulence, only: num1d => num, nuh1d => nuh
   use getm_timers, only: tic, toc, TIM_GOTM, TIM_GOTMTURB, TIM_GOTMH
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
REALTYPE                  :: u_taus,u_taub,z0s,z0b
REALTYPE                  :: h(0:kmax),dry,zz
REALTYPE                  :: NN1d(0:kmax),SS1d(0:kmax)
REALTYPE                  :: xP(0:kmax)
```

8.13.16 tke_eps_advect_3d - 3D turbulence advection (Source File: tke_eps_advect_3d.F90)

INTERFACE:

```
subroutine tke_eps_advect_3d()
```

DESCRIPTION:

This routine carries out advection of the prognostic turbulence quantities `tke` (turbuent kinetic energy, k) and `eps` (lenght scale related turbulence quantity, e.g. dissipation rate of k, ε, or turbulent frequency, $\omega = \varepsilon/k$. Here, the TVD advection schemes are used which are also used for the momentum advection. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax,az,ax
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxv,dyu
#else
   use domain, only: dx,dy
#endif
   use m3d, only: turb_adv_split,turb_adv_hor,turb_adv_ver
   use variables_3d, only: tke,eps,dt,uu,vv,ww,hun,hvn,ho,hn
   use advection, only: J7
   use advection_3d, only: do_advection_3d,W_TAG
   use halo_zones, only: update_3d_halo,wait_halo,H_TAG
   use turbulence, only: k_min,eps_min
 $ use omp_lib
   IMPLICIT NONE
```

REVISION HISTORY:

```
Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
integer                      :: i,j,k
REALTYPE,dimension(I3DFIELD) :: uuadv,vvadv,wwadv,hoadv,hnadv,huadv,hvadv
```

8.13.17 numerical_mixing() (Source File: numerical_mixing.F90)

INTERFACE:

```
subroutine numerical_mixing(F_2,F,nm3D,nm2d)
```

DESCRIPTION:

Here, the numerical tracer variance decay is calculated as proposed in *Burchard and Rennau* (2008).

USES:

```
use domain,       only: imin,imax,jmin,jmax,kmax
use variables_3d, only: dt,hn
IMPLICIT NONE
```

INPUT PARAMETERS:

```
REALTYPE, intent(in)  :: F_2(I3DFIELD)
REALTYPE, intent(in)  :: F(I3DFIELD)
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out) :: nm3d(I3DFIELD)
REALTYPE, intent(out) :: nm2d(I2DFIELD)
```

REVISION HISTORY:

```
Original author(s): Hannes Rennau
```

LOCAL VARIABLES:

```
integer                   :: i,j,k
```

8.13.18 physical_mixing() (Source File: physical_mixing.F90)

INTERFACE:

```
subroutine physical_mixing(F,AH,diffusivity,pm3d,pm2d)
```

DESCRIPTION:

Here, the physical tracer variance decay for the tracer F, $D^{\mathrm{phys}}\left(\langle F\rangle^2\right)$, due to horizontal and vertical mixing is calculated as proposed in *Burchard and Rennau* (2008):

$$D^{\mathrm{phys}}\left(F^2\right) = 2K_h\left(\partial_x F\right)^2 + 2K_h\left(\partial_y F\right)^2 + 2K_v\left(\partial_z F\right)^2 . \tag{133}$$

USES:

```
   use domain,       only: imin,imax,jmin,jmax,kmax,H,au,av
#if defined(SPHERICAL) || defined(CURVILINEAR)
   use domain, only: dxu,dyv
#else
   use domain, only: dx,dy
#endif
   use variables_3d, only: dt,nuh,hn,ssen

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)  :: F(I3DFIELD)
   REALTYPE, intent(in)  :: AH
   REALTYPE, intent(in)  :: diffusivity
  !INPUT PARAMETERS
   REALTYPE, intent(out) :: pm3d(I3DFIELD)
   REALTYPE, intent(out) :: pm2d(I2DFIELD)
```

REVISION HISTORY:

```
   Original author(s): Hannes Rennau
```

LOCAL VARIABLES:

```
   REALTYPE                  :: dupper,dlower
   integer                   :: i,j,k
   REALTYPE                  :: aux(I3DFIELD)
```

8.13.19 structure_friction_3d - (Source File: structure_friction_3d.F90)

INTERFACE:

```
subroutine structure_friction_3d()
```

DESCRIPTION:

Here, the quadratic friction term resulting from a structure in the water column is calculated. This term will be added as additional forcing to the three-dimensional momentum equations, where it is treated numerically implicitly. Therefore here, only the following terms is calculated:

$$\texttt{sf} = C(z)\sqrt{u(z)^2 + v(z)^2}, \tag{134}$$

with the friction coefficient C bearing the physical unit [1/m]. **USES:**

```
   use domain, only: imin,imax,jmin,jmax,kmax
   use variables_3d, only: uu,vv,sf,huo,hvo
   #define CALC_HALO_WW
#ifndef CALC_HALO_WW
   use domain, only: az
   use halo_zones, only: update_3d_halo,wait_halo,z_TAG
#endif
   use getm_timers, only: tic, toc, TIM_STRCTFRICT
   IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Hans Burchard & Karsten Bolding
```

LOCAL VARIABLES:

```
   REALTYPE                  :: dtm1
   integer                   :: i,j,k
#ifdef STRUCTURE_FRICTION
   REALTYPE                  :: cds(I2DFIELD)
#endif
```

9 NetCDF I/O modules

The use of external files - both input and output - is done via generic wrapper routines in GETM. For specific formats the I/O routines must be coded. In this section the specific NetCDF related I/O routines are given.

9.1 Fortran: Module Interface ncdf_common - interfaces for NetCDF IO subroutines (Source File: ncdf_common.F90)

INTERFACE:

```
module ncdf_common
```

DESCRIPTION:

!USE: IMPLICIT NONE **REVISION HISTORY:**

```
Original author(s): Karsten Bolding & Hans Burchard
```

9.2 Fortran: Module Interface Encapsulate grid related quantities (Source File: grid_ncdf.F90)

INTERFACE:

```
module grid_ncdf
```

DESCRIPTION:

This module is a container for grid related variables and parameters which are used jointly by different parts of the netCDF storage system. **USES:**

```
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
integer                              :: xlen=-1,ylen=-1,zlen=-1
integer                              :: xc_dim=-1,yc_dim=-1
integer                              :: xx_dim=-1,yx_dim=-1

!DEFINED PARAMETERS
REALTYPE, parameter                  :: h_missing      =-10.0
REALTYPE, parameter                  :: xy_missing     =-999.0
REALTYPE, parameter                  :: latlon_missing =-999.0
REALTYPE, parameter                  :: conv_missing   =-999.0
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

9.3 Fortran: Module Interface Encapsulate 2D netCDF quantities (Source File: ncdf_2d.F90)

INTERFACE:

```
module ncdf_2d
```

DESCRIPTION:

USES:

```
use output
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
integer                          :: ncid=-1

integer                          :: x_dim,y_dim
integer                          :: time_dim
integer                          :: time_id

integer                          :: elev_id,u_id,v_id
#if defined(CURVILINEAR)
integer                          :: urot_id,vrot_id
#endif
integer                          :: res_u_id=-1,res_v_id=-1
integer                          :: u10_id,v10_id
integer                          :: airp_id,t2_id,hum_id,tcc_id
integer                          :: tausx_id,tausy_id
integer                          :: zenith_angle_id
integer                          :: swr_id,albedo_id,shf_id
integer                          :: evap_id=-1,precip_id=-1
integer                          :: break_stat_id=-1

!DEFINED PARAMETERS
REALTYPE, parameter              :: elev_missing      =-9999.0
REALTYPE, parameter              :: vel_missing       =-9999.0
REALTYPE, parameter              :: airp_missing      =-9999.0
REALTYPE, parameter              :: t2_missing        =-9999.0
REALTYPE, parameter              :: hum_missing       =-9999.0
REALTYPE, parameter              :: tcc_missing       =-9999.0
REALTYPE, parameter              :: stress_missing    =-9999.0
REALTYPE, parameter              :: angle_missing     =-9999.0
REALTYPE, parameter              :: swr_missing       =-9999.0
REALTYPE, parameter              :: albedo_missing    =-9999.0
REALTYPE, parameter              :: shf_missing       =-9999.0
REALTYPE, parameter              :: evap_missing      =-9999.0
REALTYPE, parameter              :: precip_missing    =-9999.0
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

9.4 Fortran: Module Interface ncdf_2d_bdy - input in NetCDF format (Source File: ncdf_2d_bdy.F90)

INTERFACE:

```
module ncdf_2d_bdy
```

DESCRIPTION:

USES:

```
use netcdf
use m2d, only: dtm,bdy_times,bdy_data,bdy_data_u,bdy_data_v
use time, only: string_to_julsecs,time_diff,add_secs
use time, only: julianday,secondsofday,juln,secsn
use time, only: write_time_string,timestr
use domain,  only: need_2d_bdy_elev,need_2d_bdy_u,need_2d_bdy_v
IMPLICIT NONE
private
public                                :: init_2d_bdy_ncdf,do_2d_bdy_ncdf
!PRIVATE DATA MEMBERS:
integer                               :: ncid
integer                               :: time_id,elev_id=-1,nsets,bdy_len
integer                               :: u_id=-1, v_id=-1
integer                               :: start(2),edges(2)
REALTYPE                              :: offset

REAL_4B                              :: bdy_old(1500)
REAL_4B                              :: bdy_new(1500)
REAL_4B                              :: bdy_old_u(1500)
REAL_4B                              :: bdy_new_u(1500)
REAL_4B                              :: bdy_old_v(1500)
REAL_4B                              :: bdy_new_v(1500)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

9.4.1 init_2d_bdy_ncdf -

INTERFACE:

```
subroutine init_2d_bdy_ncdf(fname)
```

DESCRIPTION:

kurt,kurt **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fname
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See log for module
```

LOCAL VARIABLES:

```
integer                   :: err,rec_id,bdy_id
character(len=256)        :: units
character(len=19)         :: tbuf
integer                   :: j1,s1,j2,s2
```

9.4.2 do_2d_bdy_ncdf -

INTERFACE:

```
subroutine do_2d_bdy_ncdf(loop)
```

DESCRIPTION:

kurt,kurt **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: loop
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer,save              :: i,n
integer                   :: err
logical                   :: first=.true.
REALTYPE                  :: t
REALTYPE, save            :: t1,t2= -_ONE_,loop0
```

9.5 Fortran: Module Interface Encapsulate 3D netCDF quantities (Source File: ncdf_3d.F90)

INTERFACE:

```
module ncdf_3d
```

DESCRIPTION:

USES:

```
use output
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   integer                           :: ncid=-1

   integer                           :: x_dim,y_dim,z_dim
   integer                           :: time_dim
   integer                           :: time_id

   integer                           :: hcc_id,h_id
   integer                           :: elev_id,u_id,v_id
   integer                           :: taubx_id,tauby_id
   integer                           :: uu_id,vv_id,w_id
#ifdef _MOMENTUM_TERMS_
   integer                           :: tdv_u_id
   integer                           :: adv_u_id
   integer                           :: vsd_u_id
   integer                           :: hsd_u_id
   integer                           :: cor_u_id
   integer                           :: epg_u_id
   integer                           :: ipg_u_id

   integer                           :: tdv_v_id
   integer                           :: adv_v_id
   integer                           :: vsd_v_id
   integer                           :: hsd_v_id
   integer                           :: cor_v_id
   integer                           :: epg_v_id
   integer                           :: ipg_v_id
#endif
#if defined(CURVILINEAR)
   integer                           :: uurot_id,vvrot_id
#endif
   integer                           :: salt_id=-1
   integer                           :: temp_id=-1
   integer                           :: sigma_t_id=-1
   integer                           :: rad_id=-1
   integer                           :: tke_id,num_id,nuh_id,eps_id
   integer                           :: SS_id,NN_id
#ifdef SPM
   integer                           :: spmpool_id,spm_id
#endif
#ifdef GETM_BIO
```

```
   integer, allocatable                :: bio_ids(:)
#endif
#ifdef _FABM_
   integer, allocatable, dimension(:)  :: fabm_ids,fabm_ids_diag,fabm_ids_ben,fabm_ids_diag_hz
#endif
   integer                             :: nm3dS_id,nm3dT_id,nm2dS_id,nm2dT_id
   integer                             :: pm3dS_id,pm3dT_id,pm2dS_id,pm2dT_id
   integer                             :: nm3d_id,nm2d_id

  !DEFINED PARAMETERS
   REALTYPE, parameter                 :: hh_missing     =-9999.0
   REALTYPE, parameter                 :: elev_missing   =-9999.0
   REALTYPE, parameter                 :: vel_missing    =-9999.0
   REALTYPE, parameter                 :: tau_missing    =-9999.0
   REALTYPE, parameter                 :: salt_missing   =-9999.0
   REALTYPE, parameter                 :: temp_missing   =-9999.0
   REALTYPE, parameter                 :: rho_missing    =-9999.0
   REALTYPE, parameter                 :: rad_missing    =-9999.0
   REALTYPE, parameter                 :: tke_missing    =-9999.0
   REALTYPE, parameter                 :: nuh_missing    =-9999.0
   REALTYPE, parameter                 :: num_missing    =-9999.0
   REALTYPE, parameter                 :: eps_missing    =-9999.0
   REALTYPE, parameter                 :: SS_missing     =-9999.0
   REALTYPE, parameter                 :: NN_missing     =-9999.0
#ifdef SPM
   REALTYPE, parameter                 :: spmpool_missing=-9999.0
   REALTYPE, parameter                 :: spm_missing    =-9999.0
#endif
#if (defined(GETM_BIO) || defined(_FABM_))
   REALTYPE, parameter                 :: bio_missing=-9999.0
#endif
   REALTYPE, parameter                 :: nummix_missing=-9999.0
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

9.6 Fortran: Module Interface ncdf_3d_bdy - input in NetCDF format (Source File: ncdf_3d_bdy.F90)

INTERFACE:

```
module ncdf_3d_bdy
```

DESCRIPTION:

USES:

```
use netcdf
use domain, only: imin,imax,jmin,jmax,kmax,ioff,joff
use domain, only: nsbv,NWB,NNB,NEB,NSB,bdy_index
use domain, only: wi,wfj,wlj,nj,nfi,nli,ei,efj,elj,sj,sfi,sli
use domain, only: H
use m2d, only: dtm
use variables_3d, only: hn
use bdy_3d, only: T_bdy,S_bdy
use time, only: string_to_julsecs,time_diff,add_secs
use time, only: julianday,secondsofday,juln,secsn
use time, only: write_time_string,timestr
IMPLICIT NONE
private
public                                :: init_3d_bdy_ncdf,do_3d_bdy_ncdf
!PRIVATE DATA MEMBERS:
integer                               :: ncid
integer                               :: time_id,temp_id,salt_id
integer                               :: start(4),edges(4)
integer                               :: zax_dim,zax_len,zax_pos
integer                               :: time_dim,time_len,time_pos
logical                               :: climatology=.false.
logical                               :: from_3d_fields
REALTYPE                              :: offset
REAL_4B, allocatable                  :: bdy_times(:),wrk(:)
REAL_4B,  allocatable, dimension(:)     :: zlev
REALTYPE, allocatable, dimension(:,:)   :: T_old, T_new
REAL_4B,  allocatable, dimension(:,:)   :: T_wrk
REALTYPE, allocatable, dimension(:,:)   :: S_old, S_new
REAL_4B,  allocatable, dimension(:,:)   :: S_wrk
REALTYPE, allocatable, dimension(:,:,:) :: T_bdy_clim,S_bdy_clim
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

9.6.1 init_3d_bdy_ncdf - (Source File: ncdf_3d_bdy.F90)

INTERFACE:

```
subroutine init_3d_bdy_ncdf(fname)
```

DESCRIPTION:

kurt,kurt **USES:**

```
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fname
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
See log for module
```

LOCAL VARIABLES:

```
character(len=256)        :: units
character(len=19)         :: tbuf
integer                   :: j1,s1,j2,s2
integer                   :: ndims, nvardims
integer                   :: vardim_ids(4)
integer, allocatable, dimension(:):: dim_ids,dim_len
character(len=16), allocatable :: dim_name(:)
integer                   :: rc,err
integer                   :: i,j,k,l,m,n,id
```

9.6.2 do_3d_bdy_ncdf - (Source File: ncdf_3d_bdy.F90)

INTERFACE:

```
subroutine do_3d_bdy_ncdf(loop)
```

DESCRIPTION:

kurt,kurt **USES:**

```
use time, only: day,month,secondsofday,days_in_mon,leapyear,secsprday
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)                :: loop
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
integer         :: err
REALTYPE        :: rat
integer         :: monthsecs,prev,this,next
logical, save   :: first=.true.
integer, save   :: loop0
REALTYPE        :: t
REALTYPE, save  :: t1=_ZERO_,t2=-_ONE_
integer         :: i,j,k,l,n
```

9.7 Fortran: Module Interface ncdf_meteo - (Source File: ncdf_meteo.F90)

INTERFACE:

```
module ncdf_meteo
```

DESCRIPTION:

USES:

```
use netcdf
use time, only: string_to_julsecs,time_diff,add_secs,in_interval
use time, only: jul0,secs0,julianday,secondsofday,timestep,simtime
use time, only: write_time_string,timestr
use domain, only: imin,imax,jmin,jmax,az,lonc,latc,convc
use grid_interpol, only: init_grid_interpol,do_grid_interpol
use grid_interpol, only: to_rotated_lat_lon
use meteo, only: meteo_file,on_grid,calc_met,met_method,hum_method
use meteo, only: RELATIVE_HUM,WET_BULB,DEW_POINT,SPECIFIC_HUM
use meteo, only: airp,u10,v10,t2,hum,tcc
use meteo, only: fwf_method,evap,precip
use meteo, only: tausx,tausy,swr,shf
use meteo, only: new_meteo,t_1,t_2
use meteo, only: evap_factor,precip_factor
use exceptions
IMPLICIT NONE
private
```

PUBLIC MEMBER FUNCTIONS:

```
public init_meteo_input_ncdf,get_meteo_data_ncdf
!PRIVATE DATA MEMBERS:
REALTYPE        :: offset
integer         :: ncid,ndims,dims(3)
integer         :: start(3),edges(3)
integer         :: u10_id,v10_id,airp_id,t2_id
integer         :: hum_id,convp_id,largep_id,tcc_id
integer         :: evap_id=-1,precip_id=-1
integer         :: tausx_id,tausy_id,swr_id,shf_id
integer         :: iextr,jextr,textr,tmax=-1
integer         :: grid_scan=1
logical         :: point_source=.false.
logical         :: rotated_meteo_grid=.false.

REALTYPE, allocatable     :: met_lon(:),met_lat(:)
REALTYPE, allocatable     :: met_times(:)
REAL_4B, allocatable      :: wrk(:,:)
REALTYPE, allocatable     :: wrk_dp(:,:)

For gridinterpolation
REALTYPE, allocatable     :: beta(:,:)
REALTYPE, allocatable     :: ti(:,:),ui(:,:)
integer, allocatable      :: gridmap(:,:,:)
REALTYPE, parameter       :: pi=3.14159265358979323846264338327950 29
REALTYPE, parameter       :: deg2rad=pi/180.,rad2deg=180./pi
REALTYPE                  :: southpole(3) = (/0.0,-90.0,0.0/)
```

```
character(len=10)         :: name_lon="lon"
character(len=10)         :: name_lat="lat"
character(len=10)         :: name_time="time"
character(len=10)         :: name_u10="u10"
character(len=10)         :: name_v10="v10"
character(len=10)         :: name_airp="slp"
character(len=10)         :: name_t2="t2"
character(len=10)         :: name_hum1="sh"
character(len=10)         :: name_hum2="rh"
character(len=10)         :: name_hum3="dev2"
character(len=10)         :: name_hum4="twet"
character(len=10)         :: name_tcc="tcc"
character(len=10)         :: name_evap="evap"
character(len=10)         :: name_precip="precip"

character(len=10)         :: name_tausx="tausx"
character(len=10)         :: name_tausy="tausy"
character(len=10)         :: name_swr="swr"
character(len=10)         :: name_shf="shf"
character(len=128)        :: model_time
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding & Hans Burchard
```

9.7.1 init_meteo_input_ncdf -

INTERFACE:

```
subroutine init_meteo_input_ncdf(fn,nstart)
IMPLICIT NONE
```

DESCRIPTION:

Prepares reading meteorological forcing from a NetCDF formatted file. Based on names of various variables the corresponding variable ids are obtained from the NetCDF file. The dimensions of the meteological grid is read (x,y,t). If the southpole is not (0,-90,0) a rotated grid is assumed and coefficients for interpolation between the meteorological grid and the model grid are calculated. The arry *met_times* are filled with the times where forcing is available. Finally, meteorological fields are initialised by a call to *get_meteo_data_ncdf*. **INPUT PARAMETERS:**

```
character(len=*), intent(in)        :: fn
integer, intent(in)                 :: nstart
```

REVISION HISTORY:

```
See module for log.
```

LOCAL VARIABLES:

```
integer         :: i,j,n
integer         :: err
logical         :: ok=.true.
REALTYPE        :: olon,olat,rlon,rlat,x
character(len=10) :: name_thisvar
```

9.7.2 get_meteo_data_ncdf - .

INTERFACE:

```
subroutine get_meteo_data_ncdf(loop)
IMPLICIT NONE
```

DESCRIPTION:

Do book keeping about when new fields are to be read. Set variables used by *do_meteo* and finally calls *read_data* if necessary. **INPUT PARAMETERS:**

```
integer, intent(in)                 :: loop
```

REVISION HISTORY:

```
See module for log.
```

LOCAL VARIABLES:

```
integer         :: i,indx
REALTYPE        :: t
logical, save   :: first=.true.
integer, save   :: save_n=1
integer         :: j,s
character(len=19) :: met_str
```

9.7.3 open_meteo_file - .

INTERFACE:

```
subroutine open_meteo_file(meteo_file)
IMPLICIT NONE
```

DESCRIPTION:

Instead of specifying the name of the meteorological file directly - a list of names can be specified in *meteo_file*. The rationale for this approach is that output from operational meteorological models are of typically 2-5 days length. Collecting a number of these files allows for longer model integrations without have to reformat the data. It is assumed that the different files contains the same variables and that they are of the same shape. **INPUT PARAMETERS:**

```
character(len=*), intent(in)        :: meteo_file
```

REVISION HISTORY:

```
See module for log.
```

LOCAL VARIABLES:

```
integer, parameter        :: iunit=55
character(len=256)        :: fn,time_units
integer         :: junit,sunit,j1,s1,j2,s2
integer         :: n,err,idum
logical         :: first=.true.
logical         :: found=.false.,first_open=.true.
integer, save   :: lon_id=-1,lat_id=-1,time_id=-1,id=-1
integer, save   :: time_var_id=-1
character(len=256) :: dimname
logical         :: have_southpole
character(len=19) :: str1,str2
```

9.7.4 read_data -

INTERFACE:

```
subroutine read_data()
IMPLICIT NONE
```

DESCRIPTION:

Reads the relevant variables from the NetCDF file. Interpolates to the model grid if necessary. After a call to this routine updated versions of either variables used for calculating stresses and fluxes or directly the stresses/fluxes directly are available to *do_meteo*. **REVISION HISTORY:**

```
See module for log.
```

LOCAL VARIABLES:

```
integer         :: i1,i2,istr,j1,j2,jstr
integer         :: i,j,err
REALTYPE        :: angle,uu,vv,sinconv,cosconv
```

9.7.5 copy_var -

INTERFACE:

```
 subroutine copy_var(grid_scan,var)
subroutine copy_var(grid_scan,inf,outf)
IMPLICIT NONE
```

DESCRIPTION:

Reads the relevant variables from the NetCDF file. Interpolates to the model grid if necessary. After a call to this routine updated versions of either variables used for calculating stresses and fluxes or directly the stresses/fluxes directly are available to *do_meteo*. **INPUT PARAMETERS:**

```
integer, intent(in)                 :: grid_scan
REAL_4B, intent(in)                 :: inf(:,:)
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)               :: outf(:,:)
```

REVISION HISTORY:

```
See module for log.
```

LOCAL VARIABLES:

```
integer         :: i1,i2,istr,j1,j2,jstr
integer         :: i,j,err
```

9.8 Fortran: Module Interface ncdf_river - (Source File: ncdf_rivers.F90)

INTERFACE:

```
   module ncdf_river
```

DESCRIPTION:

USES:

```
   use netcdf
   use time, only: string_to_julsecs,time_diff,add_secs
   use time, only: julianday,secondsofday,juln,secsn,timestep
   use time, only: write_time_string,timestr
   use rivers, only: nriver,river_data,river_name,river_flow,river_factor
   use rivers, only: ok,rriver,real_river_name,river_split
   use rivers, only: temp_missing,salt_missing
   use rivers, only: use_river_temp,use_river_salt,river_temp,river_salt
#ifdef GETM_BIO
   use bio, only: bio_calc
   use bio_var, only: numc,var_names
   use rivers, only: river_bio
#endif
#ifdef _FABM_
   use getm_fabm, only: model,fabm_calc
   use rivers, only: river_fabm
#endif
   IMPLICIT NONE
   private
```

PUBLIC MEMBER FUNCTIONS:

```
   public init_river_input_ncdf,get_river_data_ncdf
  !PRIVATE DATA MEMBERS:
   REALTYPE                            :: offset
   integer                             :: ncid,ndims,dims(2),unlimdimid,textr
   integer                             :: start(1),edges(1)
   integer                             :: timedim,time_id
   integer, allocatable                :: r_ids(:)
   integer, allocatable                :: salt_id(:)
   integer, allocatable                :: temp_id(:)
   integer, allocatable                :: r_salt(:)
   integer, allocatable                :: r_temp(:)
   REAL_4B, allocatable                :: river_times(:)
#ifdef GETM_BIO
   integer, allocatable                :: bio_id(:,:)
#endif
#ifdef _FABM_
   integer, allocatable                :: fabm_id(:,:)
#endif
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

9.8.1 init_river_input_ncdf -

INTERFACE:

```
subroutine init_river_input_ncdf(fn,nstart)
IMPLICIT NONE
```

DESCRIPTION:

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fn
integer, intent(in)                 :: nstart
```

REVISION HISTORY:

```
See module for log.
```

LOCAL VARIABLES:

```
integer                   :: i,j,m,n
integer                   :: err
character(len=19)         :: tbuf
integer                   :: j1,s1,j2,s2
character(len=256)        :: time_units
character(len=256)        :: bio_name
#ifdef _FABM_
character(len=256)        :: fabm_name
#endif
```

9.8.2 get_river_data_ncdf - .

INTERFACE:

```
subroutine get_river_data_ncdf(loop)
IMPLICIT NONE
```

DESCRIPTION:

INPUT PARAMETERS:

```
integer, intent(in)                :: loop
```

REVISION HISTORY:

```
See module for log.
```

LOCAL VARIABLES:

```
integer                    :: i,j,n,nn,ni,m,indx,err
REALTYPE                   :: t
REAL_4B                    :: x(1)
logical, save              :: first=.true.
integer, save              :: save_n=1,last_indx=-1
REALTYPE, save             :: t_1,t_2,loop0
```

9.9 Fortran: Module Interface Encapsulate netCDF restart quantities (Source File: ncdf_restart.F90)

INTERFACE:

```
module ncdf_restart
```

DESCRIPTION:

This module and the related *_restart_ncdf() subroutines provide a drop-in replacement for the binary file hotstart facility in GETM. The main reason for using NetCDF formatted hotstart files instead of binary format is the abillity to use standard tools (nco, ncmerge) is a much easier way to to introduce a new subdomain decomposition for an already running set-up - without having to start all over again. See *read_restart_ncdf*() for further explanation.
This modules just contains variables shared accros the *_restart_ncdf() routines. **USES:**

```
use output
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   integer                           :: ncid=-1
   integer                           :: xdim_id=-1
   integer                           :: ydim_id=-1
   integer                           :: zdim_id=-1
   integer                           :: xax_id
   integer                           :: yax_id
   integer                           :: zax_id
   integer                           :: loop_id
   integer                           :: julianday_id
   integer                           :: secondsofday_id
   integer                           :: timestep_id
   integer                           :: z_id,zo_id
   integer                           :: U_id
   integer                           :: SlUx_id,Slru_id
   integer                           :: V_id
   integer                           :: SlVx_id,Slrv_id
#ifndef NO_3D
   integer                           :: ssen_id,ssun_id,ssvn_id
   integer                           :: sseo_id,ssuo_id,ssvo_id
   integer                           :: Uinto_id,Vinto_id
   integer                           :: uu_id,vv_id,ww_id
   integer                           :: uuEx_id,vvEx_id
   integer                           :: tke_id,eps_id
   integer                           :: num_id,nuh_id
   integer                           :: hn_id
#ifndef NO_BAROCLINIC
   integer                           :: T_id,S_id
#endif
#ifdef SPM
   integer                           :: spm_id,spmpool_id
#endif
#ifdef GETM_BIO
   integer                           :: biodim_id
   integer                           :: bio_id
#endif
```

```
#ifdef _FABM_
   integer                             :: fabmpeldim_id
   integer                             :: fabmbendim_id
   integer                             :: fabm_pel_id
   integer                             :: fabm_ben_id
#endif
#endif

   integer                             :: xlen,ylen,zlen
   integer                             :: status
   integer                             :: start(5),edges(5)
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

9.10 Fortran: Module Interface Encapsulate netCDF mean quantities (Source File: ncdf_mean.F90)

INTERFACE:

```
module ncdf_mean
```

DESCRIPTION:

USES:

```
use output
IMPLICIT NONE
```

PUBLIC DATA MEMBERS:

```
   integer                              :: ncid=-1

   integer                              :: x_dim,y_dim,z_dim
   integer                              :: time_dim
   integer                              :: time_id

   integer                              :: swrmean_id,ustarmean_id,ustar2mean_id
   integer                              :: elevmean_id
   integer                              :: uumean_id,vvmean_id,wmean_id
   integer                              :: hmean_id
   integer                              :: saltmean_id=-1
   integer                              :: tempmean_id=-1
   integer                              :: sigma_tmean_id=-1
   integer                              :: nm3dS_id,nm3dT_id,nm2dS_id,nm2dT_id
   integer                              :: pm3dS_id,pm3dT_id,pm2dS_id,pm2dT_id
   integer                              :: nm3d_id,nm2d_id
#ifdef GETM_BIO
   integer, allocatable                 :: biomean_id(:)
#endif
#ifdef _FABM_
   integer, allocatable                 :: fabmmean_ids(:)
   integer, allocatable                 :: fabmmean_ids_ben(:)
   integer, allocatable                 :: fabmmean_ids_diag(:)
   integer, allocatable                 :: fabmmean_ids_diag_hz(:)
#endif

   REALTYPE, parameter                  :: elev_missing=-9999.0
   REALTYPE, parameter                  :: hh_missing=-9999.0
   REALTYPE, parameter                  :: swr_missing=-9999.0
   REALTYPE, parameter                  :: vel_missing=-9999.0
   REALTYPE, parameter                  :: salt_missing=-9999.0
   REALTYPE, parameter                  :: temp_missing=-9999.0
   REALTYPE, parameter                  :: rho_missing=-9999.0
   REALTYPE, parameter                  :: tke_missing=-9999.0
   REALTYPE, parameter                  :: eps_missing=-9999.0
   REALTYPE, parameter                  :: nummix_missing=-9999.0
#if (defined(GETM_BIO) || defined(_FABM_))
   REALTYPE, parameter                  :: bio_missing=-9999.0
#endif

   Original author(s): Adolf Stips & Karsten Bolding
```

9.11 Fortran: Module Interface ncdf_topo() - read bathymetry and grid info (NetCDF) (Source File: ncdf_topo.F90)

INTERFACE:

```
module ncdf_topo
```

DESCRIPTION:

This module reads the bathymetry and grid information required by the module *domain*. The file format is NetCDF and data are read from the file specified as an paramater *ncdf_read_topo_file*(). For a full description of the required variables see the documention for domain. The specific readings are guided by *grid_type*. **USES:**

```
use netcdf
use exceptions
use domain, only                    : have_lonlat,have_xy
use domain, only                    : iextr,jextr,ioff,joff
use domain, only                    : imin,imax,jmin,jmax
use domain, only                    : il,ih,jl,jh
use domain, only                    : ilg,ihg,jlg,jhg
use domain, only                    : ill,ihl,jll,jhl
use domain, only                    : H, Hland
use domain, only                    : grid_type
use domain, only                    : xcord,ycord
use domain, only                    : xxcord,yxcord
use domain, only                    : dx,dy
use domain, only                    : xc,yc
use domain, only                    : xx,yx
use domain, only                    : dlon,dlat
use domain, only                    : latc,lonc
use domain, only                    : latx,lonx
use domain, only                    : convx,convc
use domain, only                    : z0_method,z0
IMPLICIT NONE
```

PUBLIC MEMBER FUNCTIONS:

```
public                                ncdf_read_topo_file
```

DEFINED PARAMETERS:

```
REALTYPE, parameter                 :: missing_double =-999.
REALTYPE, parameter                 :: rearth_default = 6378815
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf (adapted from an earlier version of
                    Karsten Bolding and Hans Burchard)
```

LOCAL VARIABLES:

```
private                               ncdf_read_2d
```

9.11.1 ncdf_read_topo_file() - read required variables

INTERFACE:

```
subroutine ncdf_read_topo_file(filename)
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

This routine checks for and opens a NetCDF file with GETM bathymetry and grid information. The first variable read and checked is *grid_type*. Subsequent operations depends on the value of *grid_type*.
The following steps are done in $ncdf\_read\_topo\_file()$:

1: check and open NetCDF file specified by 'filename'

2: read *grid_type*

3: inquire *bathymetry_id*

4: some test related to *bathymetry_id*

5: set local and global index ranges for reading

6: read bathymetry into H

7: depending on *grid_type* read axes and grid information - also check for optional variables

8: finally - check for and read spatially z_0

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: filename
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
integer                             :: ncid
integer                             :: status
integer                             :: ndims
integer                             :: dimlen
integer                             :: id
integer                             :: bathymetry_id
integer                             :: xaxis_id=-1
integer                             :: yaxis_id=-1
integer, dimension(2)               :: dimidsT(2)
character*(NF90_MAX_NAME)           :: xaxis_name,yaxis_name
integer                             :: i,j,n
integer                             :: iskipl,jskipl
integer, dimension(1)               :: start
integer, dimension(1)               :: count
logical                             :: have_dx=.true.,have_dy=.true.
logical                             :: have_dlon=.true.,have_dlat=.true.
logical                             :: have_lon=.false.
```

```
logical                         :: have_lat=.false.
logical                         :: have_xc=.false.
logical                         :: have_yc=.false.
REALTYPE                        :: a(2)
integer                         :: rc
```

9.11.2 coords_and_grid_spacing

INTERFACE:

```
subroutine coords_and_grid_spacing(ncid,varid,iextr,cordname,x0,dx)
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

Computes x and dx given that the netcdf file contains the axis (T-point) information. It is assumed that the coordinate values are equidistantly spaced. The equidistance is tested and warnings given if non-equidistant values are noted.
The routine also works for y, lon, and lat. **INPUT PARAMETERS:**

```
integer,      intent(in)             :: ncid
character(len=*), intent(in)         :: spacing_name
character(len=*), intent(in)         :: cord_name
integer,      intent(in)             ::
character(len=*), intent(in)         :: cordname
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)               :: x0, dx
```

REVISION HISTORY:

```
Original author(s): Bjarne Buchmann
```

LOCAL VARIABLES:

```
integer                   :: status
integer                   :: indx(1)
integer                   :: i
REALTYPE                  :: startval,endval
REALTYPE                  :: expectval,readval,dval
```

9.11.3 ncdf_read_2d() - generic reading routine

INTERFACE:

```
subroutine ncdf_read_2d(ncid,varid,field,il,ih,jl,jh)
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

A two-dimensional netCDF variable with specified global range `il < i < ih` and `jl < j < jh` is read into `field`. It is checked if the sizes of the fields correspond exactly. When calling this funtions, remember that FORTRAN netCDF variables start with index 1. **INPUT PARAMETERS:**

```
integer,          intent(in)        :: ncid
integer,          intent(in)        :: varid
integer,          intent(in)        :: il,ih,jl,jh
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(inout)             :: field(:,:)
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
integer                             :: status
integer, dimension(2)               :: start
integer, dimension(2)               :: count
integer, dimension(2)               :: ubounds
character(len=20)                   :: varname
```

9.12 Fortran: Module Interface ncdf_get_field() (Source File: ncdf_get_field.F90)

INTERFACE:

```
module ncdf_get_field
```

DESCRIPTION:

Provides 2 subroutines for reading 2D and 3D fields from NetCDF files. Vertical interpolation to the model grid is done for 3D fields. **USES:**

```
use netcdf
use exceptions
IMPLICIT NONE
```

PUBLIC MEMBER FUNCTIONS:

```
public inquire_file_ncdf, get_2d_field_ncdf, get_3d_field_ncdf
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

9.12.1 inquire_file_ncdf()

INTERFACE:

```
subroutine inquire_file_ncdf(fn,ncid,varids,varnames)
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

INPUT PARAMETERS:

```
   character(len=*), intent(in)        :: fn
KB   integer,          intent(in)        :: il,ih,jl,jh
KB   logical, intent(in)                 :: break_on_missing
```

OUTPUT PARAMETERS:

```
integer, intent(inout)              :: ncid
integer, allocatable, intent(inout) :: varids(:)
character(len=50), allocatable, intent(out) :: varnames(:)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
integer                             :: status,n
integer                             :: ndims,nvars
character(len=50)                   :: kurt
```

9.12.2 get_2d_field_ncdf_by_name()

INTERFACE:

```
subroutine get_2d_field_ncdf_by_name(fn,varname,il,ih,jl,jh,break_on_missing,field)
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

A two-dimensional netCDF variable with specified global range `il < i < ih` and `jl < j < jh` is read into `field`. It is checked if the sizes of the fields correspond exactly. When calling this funtions, remember that FORTRAN netCDF variables start with index 1. **INPUT PARAMETERS:**

```
character(len=*), intent(in)        :: fn,varname
integer,          intent(in)        :: il,ih,jl,jh
logical, intent(in)                 :: break_on_missing
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)               :: field(:,:)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Lars Umlauf
```

LOCAL VARIABLES:

```
integer, dimension(2)               :: start
integer, dimension(2)               :: edges
integer, dimension(2)               :: ubounds
integer                             :: status,ncid,varid
```

9.12.3 get_2d_field_ncdf()

INTERFACE:

```
subroutine get_2d_field_ncdf_by_id(ncid,varid,il,ih,jl,jh,break_on_missing,field)
```

USES:

```
IMPLICIT NONE
```

DESCRIPTION:

A two-dimensional netCDF variable with specified global range `il < i < ih` and `jl < j < jh` is read into `field`. It is checked if the sizes of the fields correspond exactly. When calling this funtions, remember that FORTRAN netCDF variables start with index 1. **INPUT PARAMETERS:**

```
integer, intent(in)                 :: ncid,varid
integer, intent(in)                 :: il,ih,jl,jh
logical, intent(in)                 :: break_on_missing
```

OUTPUT PARAMETERS:

```
REALTYPE, intent(out)               :: field(:,:)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding, Lars Umlauf
```

LOCAL VARIABLES:

```
integer, dimension(2)               :: start
integer, dimension(2)               :: edges
integer, dimension(2)               :: ubounds
integer                             :: status
```

9.12.4 get_3d_field_ncdf -

INTERFACE:

```
subroutine get_3d_field_ncdf(fname,var,nf,break_on_missing,f)
```

DESCRIPTION:

From a NetCDF files - fname - read the variable - var - into the field - f. **USES:**

```
 use netcdf
use domain, only: imin,jmin,imax,jmax,kmax,iextr,jextr,ioff,joff
use domain, only: il_domain=>il,ih_domain=>ih,jl_domain=>jl,jh_domain=>jh
use domain, only: H,az
#ifndef NO_3D
use variables_3d, only: hn
#endif
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fname,var
integer, intent(in)                 :: nf
logical, intent(in)                 :: break_on_missing
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

```
REALTYPE, intent(inout)             :: f(I3DFIELD)
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
integer                   :: il,jl,iloc,jloc,indx
integer                   :: ih,jh,kh,nh
integer                   :: rc,err,ncid,var_id,i,j,k,n
integer                   :: start(4),edges(4)
integer                   :: ndims
integer                   :: xax_id=-1,yax_id=-1,zax_id=-1,time_id=-1
character(len=256)        :: dimname
REAL_4B, allocatable      :: zax(:), tax(:), wrk(:)
REALTYPE, allocatable     :: zax_2d(:), wrk_2d(:,:,:)
```

9.12.5 Sets various attributes for a NetCDF variable. (Source File: set_attributes.F90)

INTERFACE:

```
   subroutine set_attributes(ncid,id,                              &
                             units,long_name,                      &
                             netcdf_real,                          &
                             valid_min,valid_max,valid_range,      &
                             scale_factor,add_offset,              &
                             FillValue,missing_value,              &
                             C_format,FORTRAN_format)
```

DESCRIPTION:

This routine is used to set a number of attributes for the various variables. The routine make heavy use of the *optional* keyword. The list of recognized keywords is very easy expandable. We have included a sub-set of the COARDS conventions. **USES:**

```
   use netcdf
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)                 :: ncid,id
   integer, optional                   :: netcdf_real
   character(len=*), optional          :: units,long_name
#if 1
   REALTYPE, optional                  :: valid_min,valid_max,valid_range(2)
   REALTYPE, optional                  :: scale_factor,add_offset
   REALTYPE, optional                  :: FillValue,missing_value
#else
   REAL_4B, optional                  :: valid_min,valid_max,valid_range(2)
   REAL_4B, optional                  :: scale_factor,add_offset
   REAL_4B, optional                  :: FillValue,missing_value
#endif
   character(len=*), optional          :: C_format,FORTRAN_format
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
   See ncdfout module
```

LOCAL VARIABLES:

```
   integer, parameter :: kind_real_single = SELECTED_REAL_KIND(p=5)
   integer, parameter :: kind_real_double = SELECTED_REAL_KIND(p=14)
   integer                    :: iret
   integer                    :: ft
```

9.12.6 Initialse grid related variables

INTERFACE:

```
subroutine init_grid_ncdf(ncid,init3d,x_dim,y_dim,z_dim)
```

DESCRIPTION:

This routine creates netCDF variables in an already existing netCDF file in define mode with netCDF file-id "`ncid`". All variables are related the numerical grid and the bathymetery. If the logical flag "`init3d`" evaluates false, no information about the vertical grid is initalised (e.g. if results from a horizontally integrated run are stored). Output arguments are the dimension id's for the netCDF dimensions, which may be needed for creating other, not grid related, netCDF variables. **USES:**

```
use exceptions
use netcdf
use ncdf_common, only: set_attributes
use grid_ncdf
use domain, only: imin,imax,jmin,jmax,kmax
use domain, only: grid_type,vert_cord
use domain, only: have_lonlat,have_xy
use output, only: save_metrics,save_masks
IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)            :: ncid
logical, intent(in)            :: init3d
```

INPUT PARAMETERS:

```
integer, intent(out)           :: x_dim
integer, intent(out)           :: y_dim
integer, intent(out), optional :: z_dim
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
integer                   :: status
integer                   :: id
integer                   :: axisdim(1)
integer                   :: f2_dims(2)
REALTYPE                  :: fv,mv,vr(2)
character(32)             :: xname,yname,zname
character(32)             :: xxname,yxname
character(32)             :: xunits,yunits,zunits
```

9.12.7 Save grid related variables

INTERFACE:

```
subroutine save_grid_ncdf(ncid,save3d)
```

DESCRIPTION:

This routine saves netCDF variables in an already existing netCDF file in save mode with netCDF file-id "ncid". The variables saved correspond to those GETM variables not changing in time, i.e. grid related variables and bathymetry. If the logical flag "save3d" evaluates false, no information about the vertical grid is saved (e.g. if results from a horizontally integrated run are stored). **USES:**

```
use exceptions
use netcdf
use grid_ncdf
use domain, only: imin,imax,jmin,jmax
use domain, only: grid_type,vert_cord
use domain, only: have_lonlat,have_xy
use domain, only: ioff,joff
use domain, only: dx,dy
use domain, only: dlon,dlat
use domain, only: xcord,ycord
use domain, only: xxcord,yxcord
use domain, only: xc,yc
use domain, only: xx,yx
use domain, only: latc,lonc,convc
use domain, only: latx,lonx,convx
use domain, only: latu,latv
use domain, only: dxc,dyc,dxu,dyu,dxv,dyv,dxx,dyx
KB   use domain, only: rearth
use domain, only: H,ga
use domain, only: az,au,av
use output, only: save_metrics,save_masks

IMPLICIT NONE
```

INPUT PARAMETERS:

```
integer, intent(in)        :: ncid
logical, intent(in)        :: save3d
```

REVISION HISTORY:

```
Original author(s): Lars Umlauf
```

LOCAL VARIABLES:

```
integer                    :: i,j
integer                    :: status
integer                    :: start(2),edges(2)
integer                    :: id
character(32)              :: zname
REALTYPE                   :: ws(E2DFIELD)
```

9.12.8 Initialise 2D netCDf variables

INTERFACE:

```
subroutine init_2d_ncdf(fn,title,starttime)
```

DESCRIPTION:

USES:

```
use netcdf
use exceptions
use ncdf_common
use ncdf_2d
use domain, only: imin,imax,jmin,jmax
use domain, only: ioff,joff
use meteo,  only: metforcing,calc_met
use meteo,  only: fwf_method
use m2d,    only: residual
use getm_version
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fn,title,starttime
```

DEFINED PARAMETERS:

```
logical,    parameter               :: init3d=.false.
```

REVISION HISTORY:

LOCAL VARIABLES:

```
integer                   :: err
integer                   :: scalar(1),f2_dims(2),f3_dims(3)
REALTYPE                  :: fv,mv,vr(2)
character(len=80)         :: history,ts
```

9.12.9 save_2d_ncdf() - saves 2D-fields. (Source File: save_2d_ncdf.F90)

INTERFACE:

```
   subroutine save_2d_ncdf(secs)
```

DESCRIPTION:

USES:

```
   use netcdf
   use exceptions
   use ncdf_2d
   use grid_ncdf,    only: xlen,ylen
   use domain,       only: ioff,joff,imin,imax,jmin,jmax
   use domain,       only: H,az,au,av,crit_depth
   use domain,       only: convc
   use variables_2d, only: z,D,U,DU,V,DV,res_u,res_v
#ifdef USE_BREAKS
   use variables_2d, only: break_stat
#endif
   use meteo,        only: metforcing,calc_met
   use meteo,        only: airp,u10,v10,t2,hum,tcc
   use meteo,        only: evap,precip
   use meteo,        only: tausx,tausy,zenith_angle,swr,albedo,shf

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in)               :: secs
  !DEFINED PARAMTERS:
   logical, parameter                 :: save3d=.false.
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   integer                   :: err
   integer                   :: start(3),edges(3)
   integer, save             :: n2d=0
   REALTYPE                  :: dum(1)
   integer                   :: i,j
   REALTYPE                  :: Utmp(E2DFIELD),Vtmp(E2DFIELD)
#if defined(CURVILINEAR)
   REALTYPE                  :: Urot(E2DFIELD),Vrot(E2DFIELD)
   REALTYPE                  :: deg2rad = 3.141592654/180.
   REALTYPE                  :: cosconv,sinconv
#endif
   REALTYPE,dimension(E2DFIELD) :: ws
```

9.12.10 Initialise 3D netCDF variables

INTERFACE:

```
   subroutine init_3d_ncdf(fn,title,starttime)
```

DESCRIPTION:

USES:

```
   use netcdf
   use exceptions
   use ncdf_common
   use ncdf_3d
   use domain, only: ioff,joff
   use domain, only: imin,imax,jmin,jmax,kmax
   use domain, only: vert_cord
   use m3d, only: calc_temp,calc_salt
#ifdef SPM
   use suspended_matter, only: spm_save
#endif
#ifdef GETM_BIO
   use bio_var, only: numc,var_names,var_units,var_long
#endif
#ifdef _FABM_
   use getm_fabm, only: model,fabm_calc,output_none
#endif
   use getm_version
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   character(len=*), intent(in)        :: fn,title,starttime
```

DEFINED PARAMETERS:

```
   logical,    parameter               :: init3d=.true.
```

REVISION HISTORY:

LOCAL VARIABLES:

```
   integer                   :: err
   integer                   :: n,rc
   integer                   :: scalar(1),f3_dims(3),f4_dims(4)
   REALTYPE                  :: fv,mv,vr(2)
   character(len=80)         :: history,ts
```

9.12.11 Save 3D netCDF variables (Source File: save_3d_ncdf.F90)

INTERFACE:

```
   subroutine save_3d_ncdf(secs)
```

DESCRIPTION:

USES:

```
   use netcdf
   use exceptions
   use ncdf_3d
   use grid_ncdf,    only: xlen,ylen,zlen
   use domain,       only: ioff,joff,imin,imax,jmin,jmax,kmax
   use domain,       only: H,HU,HV,az,au,av,min_depth
   use domain,       only: convc
#if defined CURVILINEAR || defined SPHERICAL
   use domain,       only: dxv,dyu,arcd1
#else
   use domain,       only: dx,dy,ard1
#endif
   use variables_2d, only: z,D
   use variables_3d, only: Uavg, Vavg, Dun, Dvn
   use variables_3d, only: dt,kmin,ho,hn,uu,hun,vv,hvn,ww,hcc,SS
   use variables_3d, only: taubx,tauby
#ifdef _MOMENTUM_TERMS_
   use variables_3d, only: tdv_u,adv_u,vsd_u,hsd_u,cor_u,epg_u,ipg_u
   use variables_3d, only: tdv_v,adv_v,vsd_v,hsd_v,cor_v,epg_v,ipg_v
#endif
#ifndef NO_BAROCLINIC
   use variables_3d, only: S,T,rho,rad,NN
#endif
   use variables_3d, only: nummix3d_S,nummix3d_T,phymix3d_S,phymix3d_T
   use variables_3d, only: numdis3d
   use variables_3d, only: tke,num,nuh,eps
#ifdef SPM
   use variables_3d, only: spm_pool,spm
#endif
#ifdef SPM
   use suspended_matter, only: spm_save
#endif
#ifdef GETM_BIO
   use bio_var, only: numc
   use variables_3d, only: cc3d
#endif
#ifdef _FABM_
   use getm_fabm,only: model,fabm_pel,fabm_ben,fabm_diag,fabm_diag_hz
#endif
   use parameters,   only: g,rho_0
   use m3d, only: calc_temp,calc_salt
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in) :: secs
```

```
   !DEFINED PARAMTERS:
    logical, parameter   :: save3d=.true.
```

REVISION HISTORY:

```
    Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
    integer                   :: err,n
    integer                   :: start(4),edges(4)
    integer, save             :: n3d=0
    REALTYPE                  :: DONE(E2DFIELD)
    REALTYPE                  :: dum(1)
    integer                   :: i,j
    REALTYPE                  :: uutmp(I3DFIELD),vvtmp(I3DFIELD)
 #if defined(CURVILINEAR)
    REALTYPE                  :: uurot(I3DFIELD),vvrot(I3DFIELD)
    REALTYPE                  :: deg2rad = 3.141592654/180.
    REALTYPE                  :: cosconv,sinconv
 #endif
    REALTYPE,dimension(E2DFIELD) :: ws2d
    REALTYPE,dimension(I3DFIELD) :: ws
```

9.12.12 ncdf_close() - closes the specified NetCDF file. (Source File: ncdf_close.F90)

INTERFACE:

```
   subroutine ncdf_close()
```

DESCRIPTION:

USES:

```
   use netcdf
   use ncdf_2d, only: nc2d => ncid
#ifndef NO_3D
   use ncdf_3d, only: nc3d => ncid
#endif
   IMPLICIT NONE
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding & Hans Burchard
```

LOCAL VARIABLES:

```
   integer                   :: err
   REALTYPE                  :: dummy=-_ONE_
```

9.12.13 Create a GETM NetCDFNetCDF hotstart file (Source File: create_restart_ncdf.F90)

INTERFACE:

```
subroutine create_restart_ncdf(fname,loop,runtype)
```

DESCRIPTION:

Creates a new NetCDF formatted file for storing variables necessary to make a correct GETM hotstart. The created file contains dimensions (xax, yax, zax) as well as the (empty) variables. Variables are named corresponding to the names used in the Fortran files. Only the actual domain is stored (i.e. not the halo-zones). This allows easy use of 'ncmerge' to stitch a number of hotstart files together to cover the entire computational domain. See read_restart_ncdf() for use. **USES:**

```
use netcdf
use ncdf_restart
use getm_version
use getm_config
use domain, only: ioff,joff
use domain, only: imin,imax,jmin,jmax,kmax
use domain, only: vert_cord
#ifdef GETM_BIO
use bio, only: bio_calc
use bio_var, only: numc
#endif
#ifdef _FABM_
use getm_fabm, only: fabm_calc,model
#endif
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fname
integer, intent(in)                 :: loop
integer, intent(in)                 :: runtype
```

REVISION HISTORY:

```
Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
character(len=80)         :: history,tts
character(len=80)         :: str_error
```

9.12.14 Writes variables to a GETM NetCDF hotstart file (Source File: write_restart_ncdf.F90)

INTERFACE:

```
subroutine write_restart_ncdf(runtype,secs,loop,julianday,secondsofday)
```

DESCRIPTION:

Writes to a NetCDF file previously created using the create_restart_ncdf() subroutine all variables necessary to make a correct GETM hotstart. The Fortran variables are written directly into the corresponding NetCDF variable. **USES:**

```
   use netcdf
   use ncdf_restart
   use domain, only: xcord,ycord
   use domain, only: imin,imax,jmin,jmax,kmax
   use variables_2d
#ifndef NO_3D
   use variables_3d
#ifdef GETM_BIO
   use bio, only: bio_calc
   use bio_var, only: numc
#endif
#ifdef _FABM_
   use getm_fabm, only: fabm_pel,fabm_ben
#endif
#endif
#ifdef SPM
   use suspended_matter
#endif
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)       :: runtype
   REALTYPE, intent(in)      :: secs ! not used now
   integer, intent(in)       :: loop,julianday,secondsofday
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
   integer         :: k,n, rc
   REALTYPE, allocatable :: zax(:)
```

9.12.15 Initialise restart netCDf variables

INTERFACE:

```
subroutine open_restart_ncdf(fname,runtype)
```

DESCRIPTION:

Opens a NetCDF formatted GETM hotstart file. All NetCDF variable id's necessary for making a correct GETM hotstart are read. The id's are shared with the reading routine using the ncdf_restart module. **USES:**

```
   use netcdf
   use ncdf_restart
#ifndef NO_3D
   use domain, only: vert_cord
#ifdef GETM_BIO
   use bio, only: bio_calc
   use getm_bio, only: bio_init_method
#endif
#ifdef _FABM_
   use getm_fabm, only: fabm_calc,fabm_init_method
#endif
#endif
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   character(len=*), intent(in)        :: fname
   integer, intent(in)                 :: runtype
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
   integer         :: dimids(3)
   character(len=20) :: varnam
```

9.12.16 Read variables from a GETM NetCDF hotstart file (Source File: read_restart_ncdf.F90)

INTERFACE:

```
subroutine read_restart_ncdf(runtype,loop,julianday,secondsofday,tstep)
```

DESCRIPTION:

Reads from a NetCDF files (with handler ncid) opened with open_restart_ncdf(). All variable id's are initialised. The variables can be read from hotstart files with the same dimensions as given by imin:imax,jmin:jmax - or - from a hotstart file with the same dimensions as topo.nc (and on the same grid). This allows to use 'ncmerge' to combine a number of hotstart files in to one - make a new sub-domain decomposition and use the newly created hotstart file. It might be necessary to use 'ncks' to cut the file to be have the same dimensions as topo.nc. Allowing for the file naming scheme in GETM links for each sub-domain should be made - e.g. ln -s restart.in restart.000.in; ln -s restart.in restart.001.in etc.
Halo-zones are updated using calls to update_2d_halo() and update_3d_halo(). **USES:**

```
   use netcdf
   use ncdf_restart
   use domain, only: iextr,jextr,ioff,joff
   use domain, only: az,au,av
   use halo_zones, only: update_2d_halo,update_3d_halo,wait_halo
   use halo_zones, only: H_TAG,U_TAG,V_TAG
   use variables_2d
   use exceptions, only: getm_error
#ifndef NO_3D
   use variables_3d
#ifdef GETM_BIO
   use bio, only: bio_calc
   use bio_var, only: numc
   use getm_bio, only: bio_init_method
#endif
#ifdef _FABM_
   use getm_fabm, only: fabm_init_method
   use getm_fabm, only: fabm_pel,fabm_ben
#endif
#endif
#ifdef SPM
   use suspended_matter
#endif
   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   integer, intent(in)      :: runtype
```

OUTPUT PARAMETERS:

```
   integer, intent(out)      :: loop,julianday,secondsofday
   REALTYPE, intent(out)     :: tstep
  !DEFINED PARAMTERS:
```

REVISION HISTORY:

```
   Original author(s): Karsten Bolding
```

LOCAL VARIABLES:

```
integer         :: il,ih,iloc,ilen,i,istart,istop
integer         :: jl,jh,jloc,jlen,j,jstart,jstop
integer         :: n
```

9.12.17 Initialise mean netCDf variables

INTERFACE:

```
subroutine init_mean_ncdf(fn,title,starttime)
```

DESCRIPTION:

USES:

```
use netcdf
use exceptions
use ncdf_common
use ncdf_mean
use domain, only: ioff,joff
use domain, only: imin,imax,jmin,jmax,kmax
use domain, only: vert_cord
use m3d, only: calc_temp,calc_salt
#ifdef GETM_BIO
use bio_var, only: numc,var_names,var_units,var_long
#endif
#ifdef _FABM_
use getm_fabm, only: model,fabm_pel,output_none
#endif
use getm_version
IMPLICIT NONE
```

INPUT PARAMETERS:

```
character(len=*), intent(in)        :: fn,title,starttime
```

DEFINED PARAMETERS:

```
logical,    parameter               :: init3d=.true.
```

REVISION HISTORY:

```
Original author(s): Adolf Stips & Karsten Bolding
Revision 1.1  2004/03/29 15:38:10  kbk
possible to store calculated mean fields
```

LOCAL VARIABLES:

```
integer                   :: n
integer                   :: err
integer                   :: scalar(1),f3_dims(3),f4_dims(4)
REALTYPE                  :: fv,mv,vr(2)
character(len=80)         :: history,tts
```

9.12.18 Initialise mean netCDF variables (Source File: save_mean_ncdf.F90)

INTERFACE:

```
   subroutine save_mean_ncdf(secs)
```

DESCRIPTION:

USES:

```
   use netcdf
   use exceptions
   use grid_ncdf,    only: xlen,ylen,zlen
   use ncdf_mean
   use diagnostic_variables
   use domain,       only: ioff,joff,imin,imax,jmin,jmax,kmax
   use domain,       only: H,az
   use domain,       only: min_depth
   use variables_3d, only: kmin
   use variables_3d, only: Dn
   use m3d, only: calc_temp,calc_salt
#ifdef GETM_BIO
   use bio_var, only: numc
#endif
#ifdef _FABM_
   use getm_fabm, only: model
#endif

   IMPLICIT NONE
```

INPUT PARAMETERS:

```
   REALTYPE, intent(in) :: secs
  !DEFINED PARAMTERS:
   logical, parameter   :: save3d=.true.
```

REVISION HISTORY:

```
   Original author(s): Adolf Stips & Karsten Bolding
```

LOCAL VARIABLES:

```
   integer                   :: n
   integer                   :: err
   integer                   :: start(4),edges(4)
   integer, save             :: n3d=0
   REALTYPE                  :: dum(1)
   REALTYPE,dimension(E2DFIELD) :: ws2d
   REALTYPE,dimension(I3DFIELD) :: ws3d
```

References

Arakawa, A., and V. R. Lamb, Computational design of the basic dynamical processes of the UCLA General Circulation Model, *Meth. Comput. Phys.*, pp. 173–263, 1977.

Backhaus, J. O., A three-dimensional model for the simulation of shelf sea dynamics, *Dt. Hydrogr. Z.*, *38*, 165–187, 1985.

Baretta, J. W., W. Ebenhöh, and P. Ruardij, The European Regional Seas Ecosystem Model, a complex marine ecosystem model, *Neth. J. Sea Res.*, *33*, 233–246, 1995.

Baumert, H., and G. Radach, Hysteresis of turbulent kinetic energy in nonrotational tidal flows: A model study, *J. Geophys. Res.*, *97*, 3669–3677, 1992.

Beckers, J.-M., and E. Deleersnijder, Stability of a FBTCS scheme applied to the propagation of shallow-water inertia-gravity waves on various space grids, *J. Computat. Phys.*, *108*, 95–104, 1993.

Beckers, J.-M., H. Burchard, J.-M. Campin, E. Deleersnijder, and P.-P. Mathieu, Another reason why simple discretizations of rotated diffusion operators cause problems in ocean models. Comments on the paper *isoneutral diffusion in a z-coordinate ocean model* by Griffies et al., *J. Phys. Oceanogr.*, *28*, 1552–1559, 1998.

Beckers, J.-M., H. Burchard, E. Deleersnijder, and P.-P. Mathieu, On the numerical discretisation of rotated diffusion operators in ocean models, *Monthly Weather Review*, *128*, 2711–2733, 2000.

Blumberg, A. F., and G. L. Mellor, A description of a coastal ocean circulation model, in *Three dimensional ocean models*, edited by N. S. Heaps, pp. 1–16, American Geophysical Union, Washington, D.C., 1987.

Bryan, K., A numerical model for the study of the world ocean, *J. Computat. Phys.*, *4*, 347–376, 1969.

Burchard, H., Turbulenzmodellierung mit Anwendungen auf thermische Deckschichten im Meer und Strömungen in Wattengebieten, Ph.D. thesis, Institut für Meereskunde, Universität Hamburg, published as: Report 95/E/30, GKSS Research Centre, 1995.

Burchard, H., Presentation of a new numerical model for turbulent flow in estuaries, in *Hydroinformatics '98*, edited by V. Babovic and L. C. Larsen, pp. 41–48, Balkema, Rotterdam, Proceedings of the third International Conference on Hydroinformatics, Copenhagen, Denmark, 24-26 August 1998, 1998.

Burchard, H., Energy-conserving discretisation of turbulent shear and buoyancy production, *Ocean Modelling*, *4*, 347–361, 2002a.

Burchard, H., *Applied turbulence modelling in marine waters*, *Lecture Notes in Earth Sciences*, vol. 100, 215 pp. pp., Springer, Berlin, Heidelberg, New York, 2002b.

Burchard, H., Quantification of numerically induced mixing and dissipation in discretisations of shallow water equations, *International Journal on Geomathematics*, submitted, 2012.

Burchard, H., and H. Baumert, On the performance of a mixed-layer model based on the k-ε turbulence closure, *J. Geophys. Res.*, *100*, 8523–8540, 1995.

Burchard, H., and J.-M. Beckers, Non-uniform adaptive vertical grids in one-dimensional numerical ocean models, *Ocean Modelling*, *6*, 51–81, 2004.

Burchard, H., and K. Bolding, GETM – a general estuarine transport model. Scientific documentation, *Tech. Rep. EUR 20253 EN*, European Commission, 2002.

Burchard, H., and O. Petersen, Hybridisation between σ and z coordinates for improving the internal pressure gradient calculation in marine models with steep bottom slopes, *Int. J. Numer. Meth. Fluids*, *25*, 1003–1023, 1997.

Burchard, H., and H. Rennau, Comparative quantification of physically and numerically induced mixing in ocean models, *Ocean Modelling*, *20*, 293–311, 2008.

Burchard, H., K. Bolding, and M. R. Villarreal, Three-dimensional modelling of estuarine turbidity maxima in a tidal estuary, *Ocean Dynamics*, *54*, 250–265, 2004.

Burchard, H., K. Bolding, W. Kühn, A. Meister, T. Neumann, and L. Umlauf, Description of a flexible and extendable physical-biogeochemical model system for the water column, *J. Mar. Sys.*, *61*, 180–211, 2006.

Casulli, V., and E. Cattani, Stability, accuracy and efficiency of a semi-implicit method for three-dimensional shallow water flow, *Computers Math. Appl.*, *27*, 99–112, 1994.

Chu, P. C., and C. Fan, Hydrostatic correction for reducing horizontal pressure gradient errors in sigma coordinate models, *J. Geophys. Res.*, *108*, 3206, doi: 10.1029/2002JC001,668, 2003.

Cox, M. D., A primitive equation, 3-dimensional model for the ocean, *Tech. Rep. 1*, Geophysical Fluid Dynamics Laboratory, University of Princeton, Princeton, N. J., 75 pp., 1984.

de Kok, J. M., A 3D finite difference model for the computation of near- and far-field transport of suspended matter near a river mouth, *Cont. Shelf Res.*, *12*, 625–642, 1992.

Deleersnijder, E., and K. G. Ruddick, A generalized vertical coordinate for 3D marine problems, *Bulletin de la Société Royale des Sciences de Liège*, *61*, 489–502, 1992.

Duwe, K., Modellierung der Brackwasserdynamik eines Tideästuars am Beispiel der Unterelbe, Ph.D. thesis, Universität Hamburg, published in: Hydromod Publ. No. 1, Wedel, Hamburg, 1988.

Espelid, T. O., J. Berntsen, and K. Barthel, Conservation of energy for schemes applied to the propagation of shallow-water inertia-gravity waves in regions with varying depth, *Int. J. Numer. Meth. Engng*, *49*, 1521–1545, 2000.

Fairall, C. W., E. F. Bradley, D. P. Rogers, J. B. Edson, and G. S. Young, Bulk parameterization of air-sea fluxes for Tropical Ocean-Global Atmosphere Coupled-Ocean Atmosphere Response Experiment, *J. Geophys. Res.*, *101*, 3747–3764, 1996.

Fofonoff, N. P., and R. C. Millard, Algorithms for the computation of fundamental properties of seawater, *Unesco technical papers in marine sciences*, *44*, 1–53, 1983.

Freeman, N. G., A. M. Hale, and M. B. Danard, A modified sigma equations' approach to the numerical modeling of Great Lakes hydrodynamics, *J. Geophys. Res.*, *77*, 1050–1060, 1972.

Gerdes, R., A primitive equation ocean circulation model using a general vertical coordinate transformation. 1. Description and testing of the model, *J. Geophys. Res.*, *98*, 14,683–14,701, 1993.

Haidvogel, D. B., and A. Beckmann, *Numerical Ocean Circulation Modelling*, *Series on Environmental Science and Management*, vol. 2, 318 pp. pp., Imperial College Press, London, 1999.

Jackett, D. R., T. J. McDougall, R. Feistel, D. G. Wright, and S. M. Griffies, Updated algorithms for density, potential temperature, conservative temperature and freezing temperature of seawater, *Journal of Atmospheric and Oceanic Technology*, submitted, 2005.

Jerlov, N. G., *Optical oceanography*, Elsevier, 1968.

Kagan, B. A., *Ocean-atmosphere interaction and climate modelling*, Cambridge University Press, Cambridge, 1995.

Kantha, L. H., and C. A. Clayson, *Small-scale processes in geophysical fluid flows*, *International Geophysics Series*, vol. 67, Academic Press, 2000a.

Kantha, L. H., and C. A. Clayson, *Numerical models of oceans and oceanic processes*, *International Geophysics Series*, vol. 66, Academic Press, 2000b.

Kondo, J., Air-sea bulk transfer coefficients in diabatic conditions, *Bound. Layer Meteor.*, *9*, 91–112, 1975.

Krone, R. B., Flume studies of the transport of sediment in estuarial shoaling processes, *Tech. rep.*, Hydraulic Eng. Lab. US Army Corps of Eng., 1962.

Lander, J. W. M., P. A. Blokland, and J. M. de Kok, The three-dimensional shallow water model TRIWAQ with a flexible vertical grid definition, *Tech. Rep. RIKZ/OS-96.104x, SIMONA report 96-01*, National Institute for Coastal and Marine Management / RIKZ, The Hague, The Netherlands, 1994.

Large, W. G., J. C. McWilliams, and S. C. Doney, Oceanic vertical mixing : a review and a model with nonlocal boundary layer parameterisation, *Rev. Geophys.*, *32*, 363–403, 1994.

Leonard, B. P., The ULTIMATE conservative difference scheme applied to unsteady one-dimensional advection, *Comput. Meth. Appl. Mech. Eng.*, *88*, 17–74, 1991.

Leonard, B. P., M. K. MacVean, and A. P. Lock, The flux integral method for multidimensional convection and diffusion, *App. Math. Modelling*, *19*, 333–342, 1995.

Madala, R. V., and S. A. Piacsek, A semi-implicit numerical model for baroclinic oceans, *J. Computat. Phys.*, *23*, 167–178, 1977.

Martinsen, E. A., and H. Engedahl, Implementation and testing of a lateral boundary scheme as an open boundary condition in a barotropic ocean model, *Coastal Engineering*, *11*, 603–627, 1987.

Mathieu, P.-P., E. Deleersnijder, B. Cushman-Roisin, J.-M. Beckers, and K. Bolding, The role of topography in small well-mixed bays, with application to the lagoon of Mururoa, *Cont. Shelf Res.*, *22*, 1379–1395, 2002.

Mellor, G. L., T. Ezer, and L.-Y. Oey, The pressure gradient conundrum of sigma coordinate ocean models, *Journal of Atmospheric and Oceanic Technology*, *11*, 1126–1134, 1994.

Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, McGraw-Hill, New York, 1980.

Paulson, C. A., and J. J. Simpson, Irradiance measurements in the upper ocean, *J. Phys. Oceanogr.*, *7*, 952–956, 1977.

Pedlosky, J., *Geophysical fluid mechanics*, 2. ed., Springer, New York, 1987.

Phillips, N. A., A coordinate system having some special advantages for numerical forecasting, *J. Meteorol.*, *14*, 184–185, 1957.

Roe, P. L., Some contributions to the modeling of discontinuous flows, *Lect. Notes Appl. Math.*, *22*, 163–193, 1985.

Shchepetkin, A. F., and J. C. McWilliams, A method for computing horizontal pressure-gradient force in an oceanic model with a nonaligned vertical coordinate, *J. Geophys. Res.*, *108*, 10.1029/2001JC001,047, 2003.

Song, Y., A general pressure gradient formulation for ocean models. Part I: Scheme design and diagnostic analysis, *Monthly Weather Review*, *126*, 3213–3230, 1998.

Song, Y., and D. B. Haidvogel, A semi-implicit ocean circulation model using a generalised topography-following coordinate, *J. Computat. Phys.*, *115*, 228–244, 1994.

Stelling, G. S., and J. A. T. M. van Kester, On the approximation of horizontal gradients in sigma co-ordinates for bathymetry with steep bottom slopes, *Int. J. Numer. Meth. Fluids*, *18*, 915–935, 1994.

Umlauf, L., H. Burchard, and K. Bolding, General Ocean Turbulence Model. Source code documentation, *Tech. Rep. 63*, Baltic Sea Research Institute Warnemünde, Warnemünde, Germany, 2005.

van Leer, B., Toward the ultimate conservative difference scheme. V: A second order sequel to Godunov's method, *J. Computat. Phys.*, *32*, 101–136, 1979.

Zalezak, S. T., Fully multidimensional flux-corrected transport algorithms for fluids, *J. Computat. Phys.*, *31*, 335–362, 1979.

Zalezak, S. T., A preliminary comparison of modern shock-capturing schemes: linear advection, in *Advances in computer methods for partial differential equations*, edited by R. V. aand R. S. Stepleman, pp. 15–22, Publ. IMACS, 1987.

Zanke, U., Berechnung der Sinkgeschwindigkeiten von Sedimenten, *Mitteilungen des Franzius-Institutes*, *46*, 231–245, 1977.