General Estuarine Transport Model

Source Code and Test Case Documentation

Hans Burchard¹, Karsten Bolding²
and Lars Umlauf¹

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

Version pre_1.8 (based on v1.7.x code)

July 7, 2011
7.3.1 init\_variables\_2d - initialise 2D related stuff. ........................................... 54
7.3.2 clean\_variables\_2d - cleanup after 2D run. ................................................... 55
7.3.3 momentum - 2D-momentum for all interior points. ....................................... 56
7.3.4 umomentum - 2D-momentum for all interior points. .................................... 57
7.3.5 vmomentum - 2D-momentum for all interior points. .................................... 58
7.3.6 uv\_advect - 2D advection of momentum ......................................................... 59
7.4 uv\_diffusion - 2D diffusion of momentum (Source File: uv\_diffusion.F90) .. 62
7.4.1 bottom\_friction - calculates the 2D bottom friction. .................................. 64
7.4.2 sealevel - using the cont. eq. to get the sealevel. ..................................... 65
7.4.3 sealevel\_nancheck - Sweep the sealevel (z) for NaN values ............................. 66
7.4.4 sealevel\_nanum - Helper routine to spot NaNs ............................................. 67
7.4.5 depth\_update - adjust the depth to new elevations. ...................................... 67
7.4.6 uv\_depths - calculate depths in u and v points. ....................................... 68
7.4.7 update\_2d\_bdy - update 2D boundaries every time step. .................................. 69
7.5 cfl\_check - check for explicit barotropic time step. (Source File: cfl\_check.F90) . 71
7.5.1 have\_bdy - checks whether this node has boundaries. ................................... 72
7.6 do\_residual - barotropic residual currents. (Source File: residual.F90) .......... 73

8 Introduction to 3d module ............................................................... 75
8.1 Overview over 3D routines in GETM ............................................................... 75
8.2 Tracer equations ...................................................................................... 76
8.3 Equation of state ..................................................................................... 78
8.4 Module m3d - 3D model component (Source File: m3d.F90) ...................... 79
8.4.1 init\_3d - initialise 3D related stuff ......................................................... 80
8.4.2 integrate\_3d - calls to do 3D model integration ........................................... 81
8.4.3 clean\_3d - cleanup after 3D run ............................................................... 82
8.5 Module variables\_3d - global 3D related variables (Source File: variables\_3d.F90) 83
8.5.1 init\_variables\_3d - initialise 3D related stuff ......................................... 85
8.5.2 clean\_variables\_3d - cleanup after 3D run. ............................................. 85
8.6 coordinates - defines the vertical coordinate (Source File: coordinates.F90) .... 86
8.7 equidistant and zoomed sigma-coordinates (Source File: sigma\_coordinates.F90) 88
8.8 general vertical coordinates (Source File: general\_coordinates.F90) ............ 90
8.9 hybrid vertical coordinates - (z-sigma) (Source File: hybrid\_coordinates.F90) .... 92
8.10 hcc\_check - hydrostatic consistency criteria (Source File: hcc\_check.F90) ...... 93
8.11 uu\_momentum\_3d - x-momentum eq. (Source File: uu\_momentum\_3d.F90) ......... 94
8.12 vv\_momentum\_3d - y-momentum eq. (Source File: vv\_momentum\_3d.F90) ......... 96
8.13 ww\_momentum\_3d - continuity eq. (Source File: ww\_momentum\_3d.F90) ........... 98
8.14 structure\_friction\_3d - (Source File: structure\_friction\_3d.F90) ................. 99
8.15 uv\_advec\_3d - 3D momentum advection (Source File: uv\_advec\_3d.F90) ........ 100
8.16 uv\_diffusion\_3d - hor. momentum diffusion (Source File: uv\_diffusion\_3d.F90) .... 104
8.17 tke\_eps\_advec\_3d - 3D turbulence advection (Source File: tke\_eps\_advec\_3d.F90) 106
8.18 bottom\_friction\_3d - bottom friction (Source File: bottom\_friction\_3d.F90) ......... 107
8.19 Module internal\_pressure (Source File: internal\_pressure.F90) ..................... 109
8.19.1 init\_internal\_pressure - initialising internal pressure gradient .................. 110
8.19.2 do\_internal\_pressure - internal pressure gradient .................................... 110
8.20 ip\_blumberg\_mellor - (Source File: ip\_blumberg\_mellor.F90) ....................... 112
8.21 ip\_blumberg\_mellor\_lin (Source File: ip\_blumberg\_mellor\_lin.F90) ............... 113
8.22 ip\_interp - (Source File: ip\_interp.F90) ....................................................... 114
8.23 ip\_song\_wright (Source File: ip\_song\_wright.F90) ...................................... 115
8.24 ip\_wu\_fan (Source File: ip\_wu\_fan.F90) ..................................................... 116
| 8.25 | ip\_shchepetkin\_mcwilliams (Source File: ip\_shchepetkin\_mcwilliams.F90) | 117 |
| 8.26 | ip\_stelling\_vankester (Source File: ip\_stelling\_vankester.F90) | 118 |
| 8.27 | Module 3D advection (Source File: advection\_3d.F90) | 119 |
| 8.27.1 | init\_advection\_3d | 120 |
| 8.27.2 | do\_advection\_3d - 3D advection schemes | 120 |
| 8.27.3 | upstream\_adv - 3D upstream advection | 122 |
| 8.27.4 | upstream\_2dh\_adv - 2D upstream advection | 123 |
| 8.27.5 | u\_split\_adv - 1D x-advection | 125 |
| 8.27.6 | v\_split\_adv - 1D y-advection | 127 |
| 8.27.7 | w\_split\_adv - 1D z-advection | 128 |
| 8.27.8 | w\_split\_it\_adv - iterated 1D z-advection | 129 |
| 8.27.9 | fct\_2dh\_adv - 2D flux-corrected transport | 130 |
| 8.28 | Module temperature (Source File: temperature.F90) | 132 |
| 8.28.1 | init\_temperature - initialisation of temperature | 132 |
| 8.28.2 | do\_temperature - temperature equation | 133 |
| 8.29 | Module Salinity (Source File: salinity.F90) | 135 |
| 8.29.1 | init\_salinity - initialisation of salinity | 135 |
| 8.29.2 | do\_salinity - salinity equation | 136 |
| 8.30 | Module suspended\_matter (Source File: spm.F90) | 138 |
| 8.30.1 | init\_spm | 139 |
| 8.30.2 | do\_spm - suspended matter equation | 140 |
| 8.31 | Module eqstate (Source File: eqstate.F90) | 142 |
| 8.31.1 | init\_eqstate | 142 |
| 8.31.2 | do\_eqstate - equation of state | 143 |
| 8.32 | ss\_mm - calculates shear and buoyancy frequency (Source File: ss\_mm.F90) | 144 |
| 8.33 | stresses\_3d - bottom and surface stresses (Source File: stresses\_3d.F90) | 146 |
| 8.34 | gotm - a wrapper to call GOTM (Source File: gotm.F90) | 147 |
| 8.35 | Module rivers (Source File: rivers.F90) | 149 |
| 8.35.1 | init\_rivers | 150 |
| 8.35.2 | init\_rivers\_bio | 151 |
| 8.35.3 | do\_rivers - updating river points | 151 |
| 8.35.4 | clean\_rivers | 152 |
| 8.36 | Module bdy\_3d - 3D boundary conditions (Source File: bdy\_3d.F90) | 153 |
| 8.36.1 | init\_bdy\_3d - initialising 3D boundary conditions | 153 |
| 8.36.2 | do\_bdy\_3d - updating 3D boundary conditions | 154 |
| 8.37 | slow\_bottom\_friction - slow bed friction (Source File: slow\_bottom\_friction.F90) | 155 |
| 8.38 | slow\_advection - slow advection terms (Source File: slow\_advection.F90) | 156 |
| 8.39 | slow\_diffusion - slow diffusion terms (Source File: slow\_diffusion.F90) | 157 |
| 8.40 | slow\_terms - calculation of slow terms (Source File: slow\_terms.F90) | 158 |
| 8.41 | start\_macro - initialise the macro loop (Source File: start\_macro.F90) | 159 |
| 8.42 | stop\_macro - terminates the macro loop (Source File: stop\_macro.F90) | 160 |
| 8.42.1 | divergence | 160 |

9 NetCDF I/O modules

| 9.1 | Module ncd\_common - interfaces for NetCDF IO subroutines (Source File: ncd\_common.F90) | 164 |
| 9.2 | Module Encapsulate grid related quantities (Source File: grid\_ncdf.F90) | 165 |
| 9.3 | Module Encapsulate 2D netCDF quantities (Source File: ncdf\_2d.F90) | 166 |
| 9.4 | Module ncdf\_bdy - input in NetCDF format (Source File: ncdf\_bdy.F90) | 168 |
| 9.4.1 | init\_2d\_bdy\_ncdf | 169 |
| 9.4.2 | do\_2d\_bdy\_ncdf | 170 |
9.5 Module Encapsulate 3D netCDF quantities (Source File: ncd_f3d.F90) .......................... 172
9.6 Module ncd_f3d_bdy - input in NetCDF format (Source File: ncd_f3d_bdy.F90) ........... 174
9.7 init_f3d_bdy_ncdf - (Source File: ncd_f3d_bdy.F90) .................................................... 176
9.8 doum_f3d_bdy_ncdf - (Source File: ncd_f3d_bdy.F90) .................................................. 177
9.9 Module ncd_meteo - (Source File: ncd_meteo.F90) ....................................................... 179
  9.9.1 init_meteo_input_ncdf - ............................................................................................. 181
  9.9.2 get_meteo_data ncdf - ................................................................................................. 182
  9.9.3 open_meteo_file - ........................................................................................................ 183
  9.9.4 read_meteo_data - ......................................................................................................... 183
  9.9.5 copy_var - .................................................................................................................... 184
9.10 Module ncd_river - (Source File: ncd_rivers.F90) ......................................................... 185
  9.10.1 init_river_input_ncdf - ............................................................................................... 186
  9.10.2 get_river_data ncdf - ................................................................................................. 187
9.11 Module Encapsulate netCDF restart quantities (Source File: ncd_restart.F90) .......... 188
9.12 Module Encapsulate netCDF mean quantities (Source File: ncd_mean.F90) ............... 190
9.13 Module ncd_topo() - read bathymetry and grid info (NetCDF) (Source File: ncd_topo.F90) 191
  9.13.1 ncd_read_topo_file() - read required variables ........................................................ 192
  9.13.2 coords_and_grid_spacing - ........................................................................................ 193
  9.13.3 ncd_read_2d() - generic reading routine ............................................................... 194
9.14 Sets various attributes for a NetCDF variable. (Source File: set_attributes.F90) ........ 196
  9.14.1 Initialise grid related variables .................................................................................. 196
  9.14.2 Save grid related variables ........................................................................................ 198
  9.14.3 Initialise 2D netCDF variables .................................................................................. 199
9.15 save_2d_ncdf() - saves 2D-fields. (Source File: save_2d_ncdf.F90) ......................... 201
  9.15.1 Initialise 3D netCDF variables .................................................................................... 202
9.16 Save 3D netCDF variables (Source File: save_3d_ncdf.F90) ......................................... 204
9.17 get_field_3d_ncdf - (Source File: get_field_3d_ncdf.F90) ........................................... 206
9.18 read_field_3d_ncdf - (Source File: read_field_3d_ncdf.F90) ....................................... 207
9.19 ncd_close() - closes the specified NetCDF file. (Source File: ncd_close.F90) .......... 208
9.20 Module field_2d_ncdf - Interface with 2D field from file (Source File: get_2d_field_ncdf.F90) 209
  9.20.1 get_2d_field_3d_ncdf() ............................................................................................ 209
9.21 Create a GETM NetCDFNetCDF hotstart file (Source File: create_restart_ncdf.F90) 211
9.22 Writes variables to a GETM NetCDF hotstart file (Source File: write_restart_ncdf.F90) 213
  9.22.1 Initialise restart NetCDF variables ............................................................................ 214
9.23 Read variables from a GETM NetCDF hotstart file (Source File: read_restart ncdf.F90) 216
  9.23.1 Initialise mean netCDF variables .............................................................................. 217
9.24 Initialise mean netCDF variables (Source File: save_mean_ncdf.F90) ...................... 219
1 What’s new

- 2006-03-27: v1.5.0
  - Support for the Fortran XLF-compiler
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 $k$-$\varepsilon$ 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 $k$-$\varepsilon$ 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:

\[ \partial_t u + \partial_z (uw) - \partial_z ((\nu_t + \nu) \partial_z u) + \alpha \left( \partial_x (u^2) + \partial_y (uv) - \partial_x (2A_h^M \partial_x u) - \partial_y (A_h^M (\partial_y u + \partial_x v)) \right) \]

\[ -fv - \int_z^\zeta \partial_x b \, dz' = -g \partial_x \zeta, \]

\[ \partial_t v + \partial_z (vw) - \partial_z ((\nu_t + \nu) \partial_z v) + \alpha \left( \partial_x (vu) + \partial_y (v^2) - \partial_y (2A_h^M \partial_y v) - \partial_x (A_h^M (\partial_y u + \partial_x v)) \right) \]

\[ +fu - \int_z^\zeta \partial_y b \, dz' = -g \partial_y \zeta. \]

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

\[ \partial_x u + \partial_y v + \partial_z w = 0. \]

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. \( \nu_t \) is the vertical eddy viscosity, \( \nu \) 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

\[ b = -g \frac{\rho - \rho_0}{\rho_0} \]

with the density \( \rho \) and a reference density \( \rho_0 \). 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 \( \alpha \) which equals unity in regions where a critical
water depth $D_{\text{crit}}$ is exceeded and approaches zero when the water depth $D$ tends to a minimum value $D_{\text{min}}$:

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

Thus, $\alpha = 1$ for $D \geq D_{\text{crit}}$, 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, $D_{\text{crit}}$ is of the order of 0.1 m and $D_{\text{min}}$ 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:

$$
\begin{align*}
w &= \partial_t \zeta + u \partial_x \zeta + v \partial_y \zeta \quad \text{for } z = \zeta, \\
w &= -u \partial_x H - v \partial_y H \quad \text{for } z = -H.
\end{align*}
$$

3.1.3 Dynamic boundary conditions

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

$$
u = 0, \quad v = 0.
$$

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 8.18.

At the surface, the dynamic boundary conditions read:

$$
\begin{align*}
(\nu_t + \nu) \partial_x u &= \alpha \tau^x, \\
(\nu_t + \nu) \partial_x v &= \alpha \tau^y,
\end{align*}
$$

The surface stresses (normalised by the reference density) $\tau^x$ and $\tau^y$ are calculated as functions of wind speed, wind direction, surface roughness etc. Also here, the drying parameter $\alpha$ 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 $\vec{v} = (u, v)$ 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:

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

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:

$$\vec{n} \cdot \vec{\nabla} G^o = 0, \quad \vec{n} \cdot \vec{\nabla} G^o = 0,$$

(11)

with $\vec{n} = (-\partial_y, \partial_x)$ 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 $\zeta$ 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 choosing 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

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

(12)

define the local layer depths $h_k$ with

$$h_k = z_k - z_{k-1}.$$  

(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 $\sigma$ transformation (see Phillips (1957) for a first application in meteorology and Freeman et al. (1972) for a first application in hydrodynamics) with

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

(14)

and

$$z_k = D\sigma_k$$

(15)
for $0 \leq k \leq N$.

The $\sigma$-coordinates can also be refined towards the surface and the bed:

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

(16)

such that $z$-levels are obtained as follows:

$$
z_k = D \beta_k
$$

(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 $\sigma$-transformation (14) with $\beta_k = \sigma_k$ is retained. Figure 1 shows four examples for vertical layer distributions obtained with the $\sigma$-transformation.

Due to the fact that all layer thicknesses are proportional to the water depth, the equidistant and also the non-equidistant $\sigma$-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 $\sigma$-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 $\sigma$-transformations given by (14) and (16). The weight for the interpolation depends on the ratio of a critical water depth $D_\gamma$ (below which equidistant $\sigma$-coordinates are used) and the actual water depth:

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

(18)

with

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

(19)

and $\sigma_k$ from (14) and $\beta_k$ 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 $D_\gamma$ 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: $\sigma$-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 $\gamma$ 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 $D_{\gamma}$ is here set to 20 m, such that at all shallower depths the equidistant $\sigma$-transformation is used. The same underlying bathymetry as in figure 1 has been used.
Figure 3: Boundary layer transformation (or $\gamma$ transformation) with concentration of layers in both, the surface mixed layer and the bottom mixed layer. Four different realisations are shown. The critical depth $D_\gamma$ is here set to 20 m, such that at all shallower depths the equidistant $\sigma$-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 $D_\gamma$ is here set to 20 m, such that at all shallower depths the equidistant $\sigma$-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

\[
\int_{z_{k-1}}^{z_k} \partial_z f \, dz = \partial_z \int_{z_{k-1}}^{z_k} f \, dz - f(z_k)\partial_z z_k + f(z_{k-1})\partial_z z_{k-1}
\]

for any function \( f \). For the vertical staggering of the layer notation see figure 8. More details about the layer integration are given in Burchard and Petersen (1997). With the further definitions of layer integrated transport,

\[
p_k := \int_{z_{k-1}}^{z_k} u \, dz, \quad q_k := \int_{z_{k-1}}^{z_k} v \, dz,
\]

layer mean velocities,

\[
u_k := \frac{p_k}{h_k}, \quad v_k := \frac{q_k}{h_k},
\]

and layer averaged tracer concentrations and buoyancy,

\[
c_i^k := \frac{1}{h_k} \int_{z_{k-1}}^{z_k} c^i \, dz, \quad b_k := \frac{1}{h_k} \int_{z_{k-1}}^{z_k} b \, dz,
\]

and the grid related vertical velocity,

\[
\tilde{w}_k := (w - \partial_t z - \partial_x z - v \partial_y z)_{z=z_k},
\]

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

\[
\partial_t h_k + \partial_x p_k + \partial_y q_k + \tilde{w}_k - \tilde{w}_{k-1} = 0.
\]

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

\[
\partial_t p_k + \tilde{w}_k \tilde{u}_k - \tilde{w}_{k-1} \tilde{u}_{k-1} - \tau^x_k + \tau^x_{k-1} + \alpha \left\{ \partial_x (u_k p_k) + \partial_y (v_k p_k) \right\} - \partial_x (2A^M_k \partial_x u_k) - \partial_y (A^M_k \partial_y u_k + \partial_x v_k) - f q_k
\]

\[
- h_k \left( \frac{1}{2} h_N (\partial^2_x b)_N + \sum_{j=h}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial^2_x b)_j \right) \right\} = -gh_k \partial_x \zeta,
\]

18
\[ \partial_t q_k + \hat{w}_k \hat{v}_k - \hat{\omega}_{k-1} \hat{v}_{k-1} - \tau_k^w + \tau_k^\nu \]
\[ + \alpha \left\{ \partial_x(u_k q_k) + \partial_y(v_k q_k) \right\} \]
\[ - \partial_y \left( 2A_k^M \bar{h}_k \partial_y v_k \right) - \partial_x \left( A_k^M \bar{h}_k (\partial_y u_k + \partial_x v_k) \right) + fp_k \]
\[ - h_k \left( \frac{1}{2} \bar{h}_N (\partial_y^* b)_N + \sum_{j=k}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial_y^* b)_j \right) \right\} = -g h_k \partial_y \zeta \]

with suitably chosen advective horizontal velocities \( \hat{u}_k \) and \( \hat{v}_k \) (see section 8.15) on page 100, the shear stresses
\[ \tau_k^w = (\nu_i \partial_x u)_k, \] (27)
\[ \tau_k^\nu = (\nu_i \partial_x v)_k, \] (28)
and
\[ (\partial_x^* b)_k = \frac{1}{2} (\partial_x b_{k+1} + \partial_x b_k) - \partial_x z_k b_{k+1} - b_k \]
\[ = \frac{1}{2} \frac{1}{(h_{k+1} + h_k)} \] (29)
and
\[ (\partial_y^* b)_k = \frac{1}{2} (\partial_y b_{k+1} + \partial_y b_k) - \partial_y z_k b_{k+1} - b_k \]
\[ = \frac{1}{2} \frac{1}{(h_{k+1} + h_k)} \] (30)

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)):
\[ w_k = \frac{1}{h_k} \left( \partial_k (h_k z_{k-1/2}) + \partial_x (p_k z_{k-1/2}) + \partial_y (q_k z_{k-1/2}) + \hat{w}_k z_k - \hat{\omega}_{k-1} z_{k-1} \right). \] (31)

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:
\[ \partial_t (h_k c_k^t) + \partial_x (p_k c_k^t) + \partial_y (q_k c_k^t) + (\hat{w}_k + w_k) c_k^t - (\hat{\omega}_{k-1} + w_{k-1}) c_{k-1} \]
\[ - (\nu_t^x \partial_x c^t)_k - (\nu_t^y \partial_y c^t)_k = Q_k^t. \] (32)

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 orthogona! 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 \( X \) and \( Y \) is mapped to a physical domain with four corners in such a way that the local coordinates of the physical space, \((\xi_x, \xi_y)\) are orthogonal to each others everywhere:

\[
X \rightarrow \xi_x, \quad Y \rightarrow \xi_y.
\]

The infinitesimal increments in the physical space, \(d_\xi_x\) and \(d_\xi_y\) are related to the infinitesimal increments in the transformed space, \(dX\) and \(dY\) by so-called metric coefficients \(m(x, y)\) and \(n(x, y)\):

\[
d_\xi_x = \left(\frac{1}{m}\right) dX, \quad d_\xi_y = \left(\frac{1}{n}\right) dY.
\]

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

\[
m = \frac{1}{r_E \cos \phi}, \quad n = \frac{1}{r_E},
\]

spherical coordinates with \(X = \lambda\) and \(Y = \phi\) are retained (with the Earth’s radius \(r_E\), longitude \(\lambda\) and latitude \(\phi\)).

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

**Continuity equation:**

\[
\partial_t \left( \frac{h_k}{mn} \right) + \partial_X \left( \frac{p_k}{n} \right) + \partial_Y \left( \frac{q_k}{m} \right) + \bar{w}_k - \bar{w}_{k-1} - \frac{1}{mn} = 0.
\]

**Momentum in \(\xi_x\) direction:**

\[
\partial_t \left( \frac{p_k}{mn} \right) + \bar{w}_k \tilde{u}_k - \bar{w}_{k-1} \tilde{u}_{k-1} - \frac{\tau^X_{k-1}}{mn} + \alpha \left\{ \partial_X \left( \frac{u_k p_k}{n} \right) + \partial_Y \left( \frac{v_k p_k}{m} \right) - q_k \left( \frac{f}{mn} + v_k \partial_X \left( \frac{1}{n} \right) - u_k \partial_Y \left( \frac{1}{m} \right) \right) \right. \\
- \partial_X \left( \frac{2 A^M h_k}{n} m \partial_X u_k \right) - \partial_Y \left( \frac{A^M h_k}{m} (n \partial_Y u_k + m \partial_X v_k) \right) \\
- \frac{h_k}{n} \left\{ \frac{1}{2} h_N (\partial_X b)_N + \sum_{j=k}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial_X b)_j \right\} = -g h_k \tilde{w}_k \
\]

20
Momentum in $\xi_y$ direction:

$$
\frac{\partial}{\partial t} \left( \frac{q_k}{mn} \right) + \bar{w}_k \tilde{v}_k - \bar{w}_{k-1} \tilde{v}_{k-1} = \frac{\tau^y_k - \tau^y_{k-1}}{mn} \\
+ \alpha \left\{ \frac{\partial}{\partial x} \left( \frac{u_k q_k}{n} \right) + \frac{\partial}{\partial y} \left( \frac{v_k q_k}{m} \right) + p_k \left( \frac{l}{mn} + v_k \partial_X \left( \frac{1}{n} \right) - u_k \partial_Y \left( \frac{1}{m} \right) \right) \\
- \frac{\partial}{\partial y} \left( 2 A_T^T h_k \frac{1}{m} n \partial_Y v_k \right) - \frac{\partial}{\partial x} \left( \frac{A_T^T h_k}{n} \left( n \partial_Y u_k + m \partial_X v_k \right) \right) \right\} = -g \frac{h_k}{m} \partial_Y \zeta. 
$$

In (38) and (39), the velocity and momentum components $u_k$ and $p_k$ are now pointing into the $\xi_x$-direction and $v_k$ and $q_k$ are pointing into the $\xi_y$-direction. The stresses $\tau^x_k$ and $\tau^y_k$ 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:

$$
\frac{\partial}{\partial t} \left( \frac{h_k c^i_k}{mn} \right) + \frac{\partial}{\partial x} \left( \frac{p_k c^i_k}{m} \right) + \frac{\partial}{\partial y} \left( \frac{q_k c^i_k}{m} \right) + \bar{w}_k \tilde{c}_k - \bar{w}_{k-1} \tilde{c}_{k-1} \\
- \frac{(v'_i \partial_x c^i)_k - (v'_i \partial_x c^i)_{k-1}}{mn} \\
- \frac{\partial}{\partial x} \left( \frac{A^T h_k}{n} m \partial_X c_k \right) - \frac{\partial}{\partial y} \left( \frac{A^T h_k}{m} n \partial_Y c_k \right) = \frac{Q^i_k}{mn}. 
$$

## 5 Discretisation

### 5.1 Mode splitting

The external system consisting of the surface elevation equation (56) and the transport equations (60) and (61) underlies a strict time step constraint if the discretisation is carried out explicitly:

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

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

$$
\Delta t < \min \left\{ \frac{\Delta x}{u_{\text{max}}}, \frac{\Delta y}{v_{\text{max}}} \right\}. 
$$
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 (62) - (69) is thus not necessary in such models.

Another approach is to use different time steps for the internal (macro times steps $\Delta t$) and the external mode (micro time steps $\Delta t_m$). 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 $\Delta t_m$ has to be an integer fraction $M$ of the macro time step $\Delta t$. $\Delta t_m$ is limited by the speed of the surface waves (41), $\Delta t$ 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:

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

and

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

such that

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

### 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 8. 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 ($X$-) direction, with $j$-indices in northern ($Y$-) 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 8. 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; ×: U-points; ⋆: V-points; •: X-points. The inserted box denotes grid points with the same index \((i, j)\).
Figure 7: Grid layout and indexing of corner points for curvilinear grids.

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 $\nu_t$ and eddy diffusivity $\nu'_t$ are located on the W-points.

For curvilinear grids, several arrays for spatial increments $\Delta x$ and $\Delta y$ have to be defined:

\begin{align}
\text{T-points:} & \quad 1 \leq i \leq i_{\text{max}}, \quad 1 \leq j \leq j_{\text{max}}, \quad 1 \leq k \leq k_{\text{max}} \\
\text{U-points:} & \quad 0 \leq i \leq i_{\text{max}}, \quad 1 \leq j \leq j_{\text{max}}, \quad 1 \leq k \leq k_{\text{max}} \\
\text{V-points:} & \quad 1 \leq i \leq i_{\text{max}}, \quad 0 \leq j \leq j_{\text{max}}, \quad 1 \leq k \leq k_{\text{max}} \\
\text{W-points:} & \quad 1 \leq i \leq i_{\text{max}}, \quad 1 \leq j \leq j_{\text{max}}, \quad 0 \leq k \leq k_{\text{max}}
\end{align}
\begin{align*}
\Delta x_{i,j}^c &= \left| \frac{1}{2} (X_{i,j-1} + X_{i,j} - X_{i-1,j-1} - X_{i-1,j}) \right| \\
\Delta x_{i,j}^u &= \left| \frac{1}{4} (X_{i+1,j-1} + X_{i+1,j} - X_{i-1,j-1} - X_{i-1,j}) \right| \\
\Delta x_{i,j}^v &= |X_{i,j} - X_{i-1,j}| \\
\Delta x_{i,j}^+ &= \left| \frac{1}{2} (X_{i+1,j} - X_{i-1,j}) \right| \\
\Delta y_{i,j}^c &= \left| \frac{1}{2} (X_{i-1,j} + X_{i,j} - X_{i-1,j-1} - X_{i,j-1}) \right| \\
\Delta y_{i,j}^u &= |X_{i,j} - X_{i,j-1}| \\
\Delta y_{i,j}^v &= \left| \frac{1}{4} (X_{i-1,j+1} + X_{i,j+1} - X_{i-1,j-1} - X_{i,j-1}) \right| \\
\Delta y_{i,j}^+ &= \left| \frac{1}{2} (X_{i,j+1} - X_{i,j-1}) \right|
\end{align*}

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 \(\Delta x\) or \(\Delta y\) is centred 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 7.

### 5.3 Lateral boundary conditions

Usually, a land mask is defined on the horizontal numerical grid. This mask is denoted by \(a^x\) for T-points, \(a^u\) for U-points and \(a^v\) for V-points with \(a^x, a^u, \) and \(a^v\) being integer fields. A T-point is either a land point \((a^x = 0)\) or a water point \((a^x > 0)\). 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 \(a^x > 1\) for T-points. Forced boundary points are marked by \(a^x = 2\) and passive boundary points by \(a^x = 3\). All other T-points are characterised by \(a^x = 1\). 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 \(a^x = 1\). 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 \(a^x = 1\). U-points which are adjacent to T-points with \(a^x = 2\) and which are not denoted by \(a^u = 2\) or \(a^v = 3\) 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 \(a^x = 2\) 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 \(\zeta\) is prescribed in all T-points with \(a^x = 2\). For passive boundary conditions \((a^x = 3)\), 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:

\[ \zeta_{i,j} = \zeta_{i+1,j} + (\zeta_{i+1,j} - \zeta_{i+2,j}) = 2\zeta_{i+1,j} - \zeta_{i+2,j}. \]
Figure 8: 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 9: Classification of grid points for a simple $5 \times 5$ configuration ($i_{\text{max}} = j_{\text{max}} = 5$). 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

$$\frac{\tau_x}{\rho_0} = u_{bx}^* u_b^*, \quad (49)$$

$$\frac{\tau_y}{\rho_0} = u_{by}^* u_b^* \quad (50)$$

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

$$\frac{u(z')}{u_*} = \frac{1}{\kappa} \ln \left( \frac{z' + z^b_0}{z^b_0} \right), \quad (51)$$

with the bottom roughness length $z^b_0$, 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:

$$u_b = \frac{u_{bx}^*}{\kappa} \ln \left( \frac{0.5h_1 + z^b_0}{z^b_0} \right), \quad (52)$$

$$v_b = \frac{u_{by}^*}{\kappa} \ln \left( \frac{0.5h_1 + z^b_0}{z^b_0} \right). \quad (53)$$

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 $\alpha$ which approaches zero when a minimum water depth is reached.

By using formulation (70) 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 calculate 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):

\[
H^u_{i,j} = \frac{1}{2} (H_{i,j} + H_{i+1,j}), \quad H^v_{i,j} = \frac{1}{2} (H_{i,j} + H_{i,j+1}).
\] (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:

\[
H^u_{i,j} = \min\{H_{i,j}, H_{i+1,j}\}, \quad H^v_{i,j} = \min\{H_{i,j}, H_{i,j+1}\}.
\] (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 \( D_{\text{min}} \) 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 \texttt{vel_depth_method}, see section 7.4.6 (subroutine \texttt{uv_depths}) documented on page 68.

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 \(-H_{i,j} + H_{\text{min}}\). 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 variable 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
4: curvilinear grid on a sphere - dx, dy are functions of (i,j) and are calculated under the assumption of a perfect sphere

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 information in the xy-plane with equidistant spacing. The name of the coordinate variables are xcord and ycord.
2: proper monotone coordinate information 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.
4: position on the sphere of the grid-vertices. The names of these variables are clled lonx and latx

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 late 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, late and convc
The longitude, latitude positions of the T-points are required when using forcing from a NWP-model. lonc and late 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 Module 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 straightforward and can be done without any changes to domain. Public variables defined in this module is used throughout 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, global index range :: ilg=-1, ihg=-1, jlg=-1, jhg=-1
integer, local index range :: 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
REALTYPE :: min_depth, crit_depth
REALTYPE :: longitude = _ZERO_
REALTYPE :: latitude = _ZERO_
logical :: f_plane = .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(:,,:)
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=0.001

DEFINED PARAMETERS:

integer, parameter :: INNER = 1
REALTYPE, private, parameter :: pi = 3.141592654
REALTYPE, private, parameter :: deg2rad = pi/180.
REALTYPE, private, parameter :: omega = 2.*pi/86400.

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: domain.F90,v $
Revision 1.41  2010-03-30 11:48:37  kb
removing adaptive_coordinates
Revision 1.39  2009-11-06 07:04:25  bjb
Consistent application of bathymetry.adjust with new bathy read
Revision 1.38  2009-11-05 14:36:10  bjb
Consistent application of mask.adjust with new bathy read
Revision 1.37  2009-09-30 05:32:47  kb
fixed calculation of dx, dy when dlon, dlat not present
Revision 1.36  2009-09-29 07:17:41  kb
fixed typos - for protex
Revision 1.35  2009-09-24 12:37:03  kb
comments and empty lines allowed in: bdyinfo.dat, minimum_depth.dat, bathymetry.adjust and mask...
Revision 1.34 2009-09-23 10:09:20 kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.33 2009-08-31 10:37:03 bjb
Consistent treatment of topo in halo zones
Revision 1.32 2009-05-15 06:59:10 bjb
typo fix
Revision 1.31 2009-05-07 15:50:46 kb
added global and local horizontal index range
Revision 1.30 2009-05-07 10:10:15 kb
fixed tag in wait_halo() - Buchmann
Revision 1.29 2008-12-09 00:31:57 kb
added new 2D open boundaries
Revision 1.28 2007-10-16 06:22:56 kbb
curvi-linear now runs in parallel
Revision 1.27 2007-03-30 13:10:59 hb
Use of adaptive and hybrid vertical coordinates technically enabled
Revision 1.26 2007-02-08 06:43:27 kbb
update HALOS for z0 - and changed loop boundaries for zub0, zvb0
Revision 1.25 2007-02-07 16:32:22 kbb
added spatial varying bottom roughness
Revision 1.24 2007-02-07 16:27:06 kbb
changed and fixed loop boundaries
Revision 1.23 2006-08-25 09:05:20 kbb
metric coefficients also calculated in HALO zones
Revision 1.22 2006-06-03 11:43:16 kbb
added namelist fallback longitude - heatfluxes
Revision 1.21 2005-06-27 07:18:13 frv-bjb
Changed STOP statements to call getm_error(…)
Revision 1.20 2005/06/17 07:40:19 frv-bjb
Added check/bailout for zero dlat and dlon
Revision 1.19 2005/05/25 10:43:42 kbb
fixed ax calculation
Revision 1.18 2005/05/25 10:32:12 kbb
merged from stabe branch v1_2_1
Revision 1.17 2005/04/25 09:32:34 kbb
added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.16 2004/11/04 11:07:00 kbb
fixed format statement in print_mask
Revision 1.15 2004/01/05 13:24:27 kbb
maxdepth from domain namelist - should be calculated later
Revision 1.14 2003/09/02 14:12:14 kbb
au and av also in HALO-zones
Revision 1.13 2003/08/28 10:36:30 kbb
also calculate ax in HALO-zones
Revision 1.12 2003/08/21 15:28:29 kbb
re-enabled update_2d_halo for lonc and latc + cleaning
Revision 1.11 2003/08/15 12:52:49 kbb
moved az mask calculation + removed print statements
Revision 1.10 2003/08/03 09:52:11 kbb
nicer print statements
Revision 1.9 2003/06/29 17:09:04 kbb
removed reference to barrier
Revision 1.8 2003/05/09 11:52:08 kbb
do not mirror coordinate info + use mask for inverse area calculation
Revision 1.7 2003/05/02 08:32:31  kbk
re-ordering mask calculation
Revision 1.6 2003/04/23 11:59:39  kbk
update_2d_halo on spherical variables + TABS to spaces
Revision 1.5 2003/04/07 14:34:42  kbk
parallel support, proper spherical grid init. support
Revision 1.1.1.1 2002/05/02 14:01:11  gotm
recovering after CVS crash
Revision 1.19 2001/10/23 14:15:55  bbh
Moved ga from coordinates.F90 to domain.F90
Revision 1.18 2001/10/22 12:10:26  bbh
Partly support for SPHERICAL grid is coded
Revision 1.17 2001/09/26 10:01:41  bbh
lat and lon maps now read in ncdf_topo.F90
Revision 1.16 2001/09/24 07:49:32  bbh
Include .h files for memory declaration/allocation
Revision 1.15 2001/09/21 11:52:47  bbh
Minimum depth for specific areas through - set_min_depth()
Revision 1.14 2001/09/14 12:04:15  bbh
Added xc,yc to hold coordinates + cleaning
Revision 1.13 2001/09/04 08:00:14  bbh
Fill coru and corv arrays
Revision 1.12 2001/09/04 07:36:32  bbh
We need ioff and joff in parallel runs
Revision 1.11 2001/09/03 15:14:22  bbh
Bug with Coriolis removed
Revision 1.10 2001/09/01 17:10:25  bbh
Vertical coordinate definition now specified via namelist
Revision 1.9 2001/08/27 11:55:02  bbh
TVD-advection for momentum added, some bugs removed
Revision 1.8 2001/08/01 08:19:57  bbh
Fields for CURVILINEAR - now done
Revision 1.7 2001/07/26 14:31:43  bbh
Manual merge
Revision 1.6 2001/07/26 14:20:02  bbh
Added grid_type, vert_cord, lonmap and latmap
Revision 1.5 2001/06/22 08:19:10  bbh
Compiler options such as USE_MASK and OLD_DRY deleted.
Open and passive boundary for z created.
Various inconsistencies removed.
wait_halo added.
Checked loop boundaries
Revision 1.4 2001/05/14 12:38:58  bbh
Set minimum depth to 10. meters if not COAST_TEST - to be fixed later.
Revision 1.3 2001/05/06 18:51:55  bbh
Towards proper implementation of specified 2D bdy.
Revision 1.2 2001/04/24 08:24:58  bbh
Use runtype instead of macro
Revision 1.1.1.1 2001/04/17 08:43:08  bbh
initial import into CVS

LOCAL VARIABLES:

REALTYPE, parameter :: rearth_default = 6378815.
6.1.1 init_domain() - initialise the computational domain

INTERFACE:

subroutine init_domain(input_dir)
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

REVISION HISTORY:

See log for module

LOCAL VARIABLES:

integer :: rc
integer :: np,sz
integer :: i,j,n
integer :: kdum
character(len=PATH_MAX) :: bathymetry = 'topo.nc'
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'
integer :: il=-1,ih=-1,jl=-1,jh=-1
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  read_par_setup() - reads domain partition

INTERFACE:

    subroutine read_par_setup(myid)
    IMPLICIT NONE

DESCRIPTION:
Reads the partitioning of the global domain in a parallel run

INPUT PARAMETERS:

    integer, intent(in) :: myid

REVISION HISTORY:

    22Apr99  Karsten Bolding & Hans Burchard  Initial code.

LOCAL VARIABLES:

    integer :: id

6.1.5  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 ! kkb
    character(len=255) :: line
    integer :: iostat
    integer :: i,j,k=0,n=-1
    integer :: il,jl,ilh,jh
    integer :: il,j1
    REALTYPE :: dmin
6.1.6 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

6.1.7 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
6.1.8 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
6.2 part_domain() - partition the domain (Source File: part_domain.F90)

INTERFACE:

subroutine part_domain()

DESCRIPTION:

Set various integers defining the calculation domain. The settings depend 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
6.3 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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard

$Log: bdy_spec.F90,v $
Revision 1.9 2009-09-29 08:07:28 kb
fixed reading western boundary information
Revision 1.8 2009-09-24 12:37:03 kb
comments and empty lines allowed in: bdyinfo.dat, minimum_depth.dat, bathymetry.adjust and mask.adjust
Revision 1.7 2009-09-23 10:11:47 kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.6 2008-12-09 00:31:57 kb
added new 2D open boundaries
Revision 1.5 2005-04-24 12:55:31 kbb
removing print statement
Revision 1.4 2003/08/03 09:52:11 kbb
nicer print statements
Revision 1.3 2003/04/23 11:59:39 kbb
update_2d_halo on spherical variables + TABS to spaces
Revision 1.2 2003/04/07 15:20:53 kbb
added bdy_index and bdy_map
Revision 1.1.1.1 2002/05/02 14:01:11 gotm
recovering after CVS crash
Revision 1.2 2001/09/01 17:12:13 bbb
Removed a STDERR
Revision 1.1.1.1 2001/04/17 08:43:08 bbb
initial import into CVS
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)
6.4  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 information 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 each
sub-domain will be printed.

USES:

use domain, only: NWB,NWB,NEB,NSB
use domain, only: wi,wfj,wi,j,ni,nil,ei,efj,elj,sj,sfj,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

OUTPUT/INPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: print_bdy.F90,v $
Revision 1.4 2009-09-23 10:11:47 kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.3 2008-12-09 00:31:57 kb
added new 2D open boundaries
Revision 1.2 2003-04-23 11:59:39 kbk
update_2d_halo on spherical variables + TABS to spaces
Revision 1.1.1.1 2002/05/02 14:01:12 gotm
recovering after CVS crash
Revision 1.1.1.1 2001/04/17 08:43:08 bbh
initial import into CVS

LOCAL VARIABLES:

    integer :: m,n
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
    $Log: mirror_bdy_2d.F90,v $
    Revision 1.6  2009-09-23 10:11:47  kb
    rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
    Revision 1.5  2009-08-31 10:37:03  bjb
    Consistent treatment of topo in halo zones
    Revision 1.4  2007-05-14 08:12:43  kbk
    fixed loops
    Revision 1.3  2006-08-25 09:00:19  kbk
    fixed sequence of boundary updates
    Revision 1.2  2003/04/23 11:59:39  kbk
    update_2d_halo on spherical variables + TABS to spaces
    Revision 1.1  2003/04/07 15:22:03  kbk
    parallel support

LOCAL VARIABLES:

    integer :: i,j,n
6.6  mirror_bdy_3d() - mirrors 3d variables (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 routine 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
$Log: mirror_bdy_3d.F90,v $
Revision 1.7  2009-09-23 10:11:47  kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.6  2009-08-31 10:37:03  bjb
Consistent treatment of topo in halo zones
Revision 1.5  2007-06-07 10:25:19  kbk
imin,imax,jmin,jmax -> imin,imax,jmin,jmax
Revision 1.4  2007-05-14 08:12:43  kbk
fixed loops
Revision 1.3  2006-08-25 09:00:19  kbk
fixed sequence of boundary updates
Revision 1.2  2003/04/23 11:59:39  kbk
update_2d_halo on spherical variables + TABS to spaces
Revision 1.1  2003/04/07 15:22:03  kbk
parallel support

LOCAL VARIABLES:

    integer :: i,j,n

46
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-dimensional 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:

\[ \partial_t \zeta = -\partial_x U - \partial_y V. \]  

with

\[ U = \int_{-H}^{\zeta} u \, dz, \quad V = \int_{-H}^{\zeta} v \, dz. \]  

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

\[ \partial_t U + \tau^x_b + \alpha \left( \int_{-H}^{\zeta} (\partial_x u^2 + \partial_y (uv)) \, dz \right) \\
- \tau^x_s - \int_{-H}^{\zeta} (\partial_x (2A^M_h \partial_x u) - \partial_y (A^M_h (\partial_y u + \partial_x v))) \, dz \]

\[ -fV - \int_{-H}^{\zeta} \int_{z'}^{\zeta} \partial_x b \, dz' \, dz \]  

= \quad -gD \partial_x \zeta \]  

and

\[ \partial_t V + \tau^y_b + \alpha \left( \int_{-H}^{\zeta} (\partial_y (uv) + \partial_y u^2)) \, dz \right) \\
- \tau^y_s - \int_{-H}^{\zeta} (\partial_y (2A^M_h \partial_y v) - \partial_x (A^M_h (\partial_y u + \partial_x v))) \, dz \]

\[ + fU - \int_{-H}^{\zeta} \int_{z'}^{\zeta} \partial_y b \, dz' \, dz \]  

= \quad -gD \partial_y \zeta. \]

Here, \( \tau^x_b \) and \( \tau^y_b \) are bottom stresses. Their calculation is discussed in section 8.18. As a first preparation for the mode splitting, these integrals of the momentum equations can be formally rewritten as
\[\begin{aligned}
\partial_t U + \frac{R}{D^2} U \sqrt{U^2 + V^2} + S_F^x + \alpha \left( \partial_x \left( \frac{U^2}{D} \right) + \partial_y \left( \frac{UV}{D} \right) \right) \\
- \tau_x^x - \partial_x \left( 2A_h^M D \partial_x \left( \frac{U}{D} \right) \right) - \partial_y \left( A_h^M D \left( \frac{U}{D} \right) + \partial_x \left( \frac{V}{D} \right) \right) \\
-fV + S_A^x - S_D^x + S_B^x \right) = -gD \partial_x \zeta
\end{aligned}\]

(60)

and

\[\begin{aligned}
\partial_t V + \frac{R}{D^2} V \sqrt{U^2 + V^2} + S_F^y + \alpha \left( \partial_x \frac{UV}{D} + \partial_y \frac{V^2}{D} \right) \\
- \tau_y^y - \partial_y \left( A_h^M D \left( \frac{U}{D} \right) + \partial_x \left( \frac{V}{D} \right) \right) - \partial_y \left( 2A_h^M D \partial_y \left( \frac{V}{D} \right) \right) \\
+fU + S_A^y - S_D^y + S_B^y \right) = -gD \partial_y \zeta
\end{aligned}\]

(61)

with the so-called slow terms for bottom friction

\[S_F^x = \tau_x^x - \frac{R}{D^2} U \sqrt{U^2 + V^2},\]

(62)

\[S_F^y = \tau_y^y - \frac{R}{D^2} V \sqrt{U^2 + V^2},\]

(63)

horizontal advection

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

(64)

\[S_A^y = \int_{-H}^{\zeta} \left( \partial_x (uv) + \partial_y v^2 \right) dz - \partial_x \left( \frac{UV}{D} \right) - \partial_y \left( \frac{V^2}{D} \right),\]

(65)

horizontal diffusion

\[S_D^x = \int_{-H}^{\zeta} \left( \partial_x \left( 2A_h^M D \partial_x \right) - \partial_y \left( A_h^M (\partial_y u + \partial_x v) \right) \right) dz\]

(66)

\[- \partial_x \left( 2A_h^M D \partial_x \left( \frac{U}{D} \right) \right) - \partial_y \left( A_h^M D \left( \frac{U}{D} \right) + \partial_x \left( \frac{V}{D} \right) \right),\]

\[S_D^y = \int_{-H}^{\zeta} \left( \partial_y \left( 2A_h^M D \partial_y \right) - \partial_x \left( A_h^M (\partial_y u + \partial_x v) \right) \right) dz\]

(67)

\[- \partial_y \left( 2A_h^M D \partial_y \left( \frac{V}{D} \right) \right) - \partial_x \left( A_h^M D \left( \frac{U}{D} \right) + \partial_x \left( \frac{V}{D} \right) \right),\]

and internal pressure gradients

\[S_B^x = - \int_{-H}^{\zeta} \int_{z}^{\zeta} \partial_x b \, dz' \, dz\]

(68)
and

\[ S_B^y = - \int_{-H}^{\zeta} \int_{-H}^{\zeta} \partial_y b \, d\zeta' \, dz. \]  

(69)

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)):

\[ R = \left( \frac{\kappa}{\ln \left( \frac{D^2}{z_b^0} + \frac{\kappa^2}{z_b^0} \right)} \right)^2. \]  

(70)

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 on page 62.
7.2 Module 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, HU, HV, min_depth
use domain, only: ilg, ihg, jlg, jhg
use domain, only: ill, ihl, jll, jhl
use domain, only: openbdy, z0_method, z0_const, z0
use domain, only: az, ax
use halo_zones, only: update_2d_halo, wait_halo
use halo_zones, only: U_TAG, V_TAG, H_TAG
use variables_2d

IMPLICIT NONE

Temporary interface (should be read from module):

interface

subroutine get_2d_field(fn, varname, il, ih, jl, jh, f)
character(len=*), intent(in) :: fn, varname
integer, intent(in) :: il, ih, jl, jh
REALTYPE, intent(out) :: f(:, :)
end subroutine get_2d_field

end interface

PUBLIC DATA MEMBERS:

logical :: have_boundaries
REALTYPE :: dtm, 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, bdyramp_2d=-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(:)
integer, parameter :: comm_method=-1

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 :: vel_depth_method=0
namelist /m2d/ &
    sealevel_check, bdy2d,bdyfmt_2d,bdyramp_2d,bdyfile_2d
7.2.2 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_adveect      calculate 2D advection terms
- call uv_diffusion    calculate 2D diffusion terms
- call momentum        iterate 2D momentum equations
- call sealevel        update sea surface elevation
- call depth_update    update water depths
- call do_residual     calculate intermediate 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:

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

INTERFACE:

module variables_2d

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 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,H,HU,HV,min_depth

IMPLICIT NONE

PUBLIC DATA MEMBERS:

#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

LOCAL VARIABLES:

integer :: rc

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 55.

INPUT PARAMETERS:

integer, intent(in) :: runtype

INPUT/OUTPUT PARAMETERS:
Table 1: Public 2D variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>sea surface elevation in T-point</td>
<td>[m]</td>
</tr>
<tr>
<td>zu</td>
<td>sea surface elevation in U-point</td>
<td>[m]</td>
</tr>
<tr>
<td>zv</td>
<td>sea surface elevation in V-point</td>
<td>[m]</td>
</tr>
<tr>
<td>U</td>
<td>x component of transport in U-point</td>
<td>[m$^2$s$^{-1}$]</td>
</tr>
<tr>
<td>DU</td>
<td>water depth in U-point</td>
<td>[m]</td>
</tr>
<tr>
<td>fU</td>
<td>Coriolis term for V-equation in V-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>S1Ux</td>
<td>slow term for U-equation in U-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>S1ru</td>
<td>slow bottom friction for U-equation in U-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>V</td>
<td>y component of transport in V-point</td>
<td>[m$^2$s$^{-1}$]</td>
</tr>
<tr>
<td>DV</td>
<td>water depth in V-point</td>
<td>[m]</td>
</tr>
<tr>
<td>fV</td>
<td>Coriolis term for U-equation in U-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>S1Vx</td>
<td>slow term for V-equation in V-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>S1rv</td>
<td>slow bottom friction for V-equation in V-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>Uint</td>
<td>x-component of mean transport in U-point</td>
<td>[m$^2$s$^{-1}$]</td>
</tr>
<tr>
<td>Vint</td>
<td>y-component of mean transport in V-point</td>
<td>[m$^2$s$^{-1}$]</td>
</tr>
<tr>
<td>UEx</td>
<td>sum of explicit terms for U-equation in U-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>VEx</td>
<td>sum of explicit terms for V-equation in V-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>ru</td>
<td>bottom friction for U-equation in U-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>rv</td>
<td>bottom friction for V-equation in V-point</td>
<td>[m$^2$s$^{-2}$]</td>
</tr>
<tr>
<td>res_du</td>
<td>residual depth in U-point</td>
<td>[m]</td>
</tr>
<tr>
<td>res_u</td>
<td>x-component of residual transport in U-point</td>
<td>[m$^2$s$^{-1}$]</td>
</tr>
<tr>
<td>res_dv</td>
<td>residual depth in V-point</td>
<td>[m]</td>
</tr>
<tr>
<td>res_v</td>
<td>y-component of residual transport in V-point</td>
<td>[m$^2$s$^{-1}$]</td>
</tr>
<tr>
<td>surfdiv</td>
<td>divergence of surface currents in T-point</td>
<td>[s$^{-1}$]</td>
</tr>
</tbody>
</table>

7.3.2 clean_variables_2d - cleanup after 2D run.

INTERFACE:

    subroutine clean_variables_2d()
    IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:
OUTPUT PARAMETERS:

DESCRIPTION:
This routine is currently empty.

LOCAL VARIABLES:

7.3.3 momentum - 2D-momentum for all interior points.

INTERFACE:

subroutine momentum(n,tausx,tausy,airp)

DESCRIPTION:
This small routine calls the U-equation and the V-equation in an alternating sequence (UVUVUVUVU), 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:

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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

logical :: ufirst=.false.
7.3.4 umomentum - 2D-momentum for all interior points.

INTERFACE:

    subroutine umomentum(tausx,airp)

DESCRIPTION:

Here, the vertically integrated $U$-momentum equation (60) given on page 48 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 $S_x^F$ is now also multiplied with the parameter $\alpha$ 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 29 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 explicitly:

\[
U^{n+1} = \frac{U^n - \Delta t_m (gD \partial_x \zeta + \alpha (-\frac{\tau_s}{\rho_0} - fV^n + U_{Ez} + S_A^x - S_D^x + S_B^x + S_F^x))}{1 + \Delta t_m \frac{R_D^2}{\sqrt{(U^n)^2 + (V^n)^2}^2}},
\]

with $U_{Ez}$ combining advection and diffusion of $U$, see routines `uv_advect` (section 7.3.6 on page 59) and `uv_diffusion` (section 7.4 on page 62). The slow terms are calculated in the routine `slow_terms` documented in section 8.40 on page 158. In (71), $U^{n+1}$ 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 13.

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
    #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)

INPUT/OUTPUT PARAMETERS:
OUTPUT PARAMETERS:

LOCAL VARIABLES:

- integer :: i,j
- REALTYPE :: zx(E2DFIELD)
- REALTYPE :: tausu(E2DFIELD)
- REALTYPE :: Slr(E2DFIELD)
- REALTYPE :: zp,zm,Uloc
- REALTYPE :: gamma=rho_0*g
- REALTYPE :: cord_curv=_ZERO_
- REALTYPE :: gammai

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

INTERFACE:

subroutine vmomentum(tausy,airp)

DESCRIPTION:

Here, the vertically integrated $V$-momentum equation (61) given on page 48 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 $\alpha$ 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 29 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 explicitly:

$$V^{n+1} = V^n - \Delta t_m \left( gD \partial_y \zeta + \alpha \left( -\frac{z^2}{\rho_0} + fU^n + V_{Ex} + S_A^y - S_D^y + S_B^y + S_F^y \right) \right) \frac{1 + \Delta t_m \frac{H}{2} \sqrt{(U^n)^2 + (V^n)^2}}{1 + \Delta t_m \frac{H}{2} \sqrt{(U^n)^2 + (V^n)^2}},$$

(72)

with $V_{Ex}$ combining advection and diffusion of $V$, see routines $uv_{advect}$ (section 7.3.6 on page 59) and $uv_{diffusion}$ (section 7.4 on page 62). The slow terms are calculated in the routine $slow_terms$ documented in section 8.40 on page 158. In (72), $V^{n+1}$ 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 13.

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,dyv
- use m2d, only: U

58
INPUT PARAMETERS:
REALTYPE, intent(in) :: tausy(E2DFIELD),airp(E2DFIELD)

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

7.3.6  uv_advect - 2D advection of momentum

INTERFACE:

subroutine uv_advect

DESCRIPTION:

The advective terms in the vertically integrated momentum equation are discretised in a momentum-
conservative form. This is carried out here for the advective terms in the $U$-equation (60) and the
$V$-equation (61) (after applying the curvilinear coordinate transformation and multiplying these
equations with $mn$).

First advection term in (60):

$$
\left( mn \partial_X \left( \frac{U^2}{D_n} \right) \right)_{i,j} \approx \frac{1}{2} \left( U_{i+1,j} + U_{i,j} \right) \frac{\Delta y_{i+1,j}^c}{\Delta x_{i,j}} \Delta y_{i,j}^c - \frac{1}{2} \left( U_{i,j} + U_{i-1,j} \right) \frac{\Delta y_{i,j}^c}{\Delta x_{i,j}} \Delta y_{i,j}^c
$$

(73)

For the upwind scheme used here, the inter-facial velocities which are defined on T-points are here
calculated as:
\[ \tilde{u}_{i,j} = \begin{cases} \frac{U_{i-1,j}}{D_{i-1,j}^u} & \text{for } \frac{1}{2}(U_{i,j} + U_{i-1,j}) > 0 \\ \frac{U_{i,j}}{D_{i,j}^u} & \text{else.} \end{cases} \] (74)

Second advection term in (60):
\[ \left( mn \frac{\partial y}{\partial y} \left( \frac{UV}{Dm} \right) \right)_{i,j,k} \approx \ \frac{1}{2}(V_{i+1,j} + V_{i,j})\tilde{u}_{i,j} \Delta x_{i,j}^+ - \frac{1}{2}(V_{i+1,j-1} + V_{i,j-1})\tilde{u}_{i,j-1} \Delta x_{i,j-1}^+ \Delta y_{i,j}^+ \] (75)

For the upwind scheme used here, the interfacial velocities which are defined on X-points are here calculated as:
\[ \tilde{u}_{i,j} = \begin{cases} \frac{U_{i,j}}{D_{i,j}^u} & \text{for } \frac{1}{2}(V_{i+1,j,k} + V_{i,j,k}) > 0 \\ \frac{U_{i+1,j}}{D_{i+1,j}^u} & \text{else.} \end{cases} \] (76)

First advection term in (61):
\[ \left( mn \frac{\partial \chi}{\partial x} \left( \frac{UV}{Dn} \right) \right)_{i,j,k} \approx \ \frac{1}{2}(U_{i,j+1} + U_{i,j})\tilde{v}_{i,j} \Delta y_{i,j}^+ - \frac{1}{2}(U_{i-1,j+1} + U_{i-1,j})\tilde{v}_{i-1,j} \Delta y_{i-1,j}^+ \Delta x_{i,j}^+ \] (77)

For the upwind scheme used here, the interfacial velocities which are defined on X-points are here calculated as:
\[ \tilde{v}_{i,j} = \begin{cases} \frac{V_{i,j}}{D_{i,j}^v} & \text{for } \frac{1}{2}(U_{i+1,j} + U_{i,j}) > 0 \\ \frac{V_{i+1,j}}{D_{i+1,j}^v} & \text{else.} \end{cases} \] (78)

Second advection term in (61):
\[ \left( mn \frac{\partial y}{\partial y} \left( \frac{V^2}{Dm} \right) \right)_{i,j,k} \approx \ \frac{1}{2}(V_{i,j+1} + V_{i,j})\tilde{e}_{i,j+1} \Delta x_{i,j+1}^+ - \frac{1}{2}(V_{i,j} + V_{i,j-1})\tilde{e}_{i,j} \Delta x_{i,j}^+ \Delta y_{i,j}^+ \] (79)

For the upwind scheme used here, the interfacial velocities which are defined on T-points are here calculated as:
\[
\bar{v}_{i,j} = \begin{cases} 
\frac{V_{i,j-1}}{Dv_{i,j-1}} & \text{for } \frac{1}{2}(V_{i,j} + V_{i,j-1}) > 0 \\
\frac{V_{i,j}}{Dv_{i,j}} & \text{else.}
\end{cases}
\]

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`
- use domain, only: `ioff,joff`
  
  #if defined(SPHERICAL) || defined(CURVILINEAR)
  - use domain, only: `dyc,arudi,dxx,dyx,arvdi,dxc`
  #else
  - use domain, only: `dx,dy,ard1`
  #endif
- use variables_2d, only: `U,DU,UEx,V,DV,VEx,PP`
- use getm_timers, only: `tic, toc, TIM_UVADVECT`

$ use omp_lib

IMPLICIT NONE

**INPUT PARAMETERS:**

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

Original author(s): Hans Burchard & Karsten Bolding

**LOCAL VARIABLES:**

- integer :: `i,j,ii,jj`

---

61
7.4 \texttt{uv_diffusion} - 2D diffusion of momentum (Source File: \texttt{uv_diffusion.F90})

INTERFACE:

    subroutine \texttt{uv_diffusion}(Am, An_method, An, AnX)

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 $U_{\text{Ex}}$ and $V_{\text{Ex}}$ for the $U$- and the $V$-equation, respectively. The physical diffusion with the given eddy viscosity coefficient $A^M$ 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 $A^N$, see the example given as equations (89) and (90).

First diffusion term in (60):

$$
\left( \frac{m n}{2} \partial_X \left( 2 A^M h D \frac{U}{D} + A^N h \partial_X U \right) \right)_{i,j} \approx \frac{F_{Dxx}^{i+1,j} - F_{Dxx}^{i,j}}{\Delta x_{i,j} \Delta y_{i,j}}
$$

with diffusive fluxes

$$
F_{Dxx}^{i,j} = \left( 2 A^M h \frac{D_{i,j}}{D_{i,j} + D_{i+1,j}} \right) \Delta x^x_{i,j} \left( \frac{U_{i,j+1} - U_{i,j}}{D_{i,j+1} - D_{i,j}} \right) \frac{1}{\Delta y_{i,j}} + \left( A^N h \left( U_{i+1,j} - U_{i,j} \right) \right) \frac{1}{\Delta x^x_{i,j}}
$$

Second diffusion term in (60):

$$
\left( \frac{m n}{2} \partial_Y \left( A^M h D \left( \partial_Y \left( \frac{U}{D} \right) + \partial_X \left( \frac{V}{D} \right) \right) + A^N h \partial_Y U \right) \right)_{i,j} \approx \frac{F_{Dxy}^{i,j} - F_{Dxy}^{i-1,j}}{\Delta x_{i,j} \Delta y_{i,j}}
$$

with diffusive fluxes

$$
F_{Dxy}^{i,j} = \left( A^M h \frac{1}{2} \left( D_{i,j} + D_{i+1,j} \right) \Delta x^x_{i,j} \left( \frac{U_{i,j+1} - U_{i,j}}{D_{i,j+1} - D_{i,j}} \right) \frac{1}{\Delta y_{i,j}} + \left( A^N h \left( V_{i,j+1} - V_{i+1,j} \right) \right) \frac{1}{\Delta x^x_{i,j}}
$$

First diffusion term in (61):

$$
\left( \frac{m n}{2} \partial_X \left( \frac{U}{D} \right) + \partial_X \left( \frac{V}{D} \right) \right) \approx \frac{F_{Dyx}^{i,j} - F_{Dyx}^{i-1,j}}{\Delta x_{i,j} \Delta y_{i,j}}
$$

with diffusive fluxes

$$
F_{Dyx}^{i,j} = \left( A^M h \frac{1}{2} \left( D_{i,j} + D_{i+1,j} \right) \Delta y^x_{i,j} \left( \frac{U_{i,j+1} - U_{i,j}}{D_{i,j+1} - D_{i,j}} \right) \frac{1}{\Delta x^x_{i,j}} + \left( A^N h \left( V_{i+1,j} - V_{i,j} \right) \right) \frac{1}{\Delta y^x_{i,j}}
$$

62
Second diffusion term in (61):

\[
\left( m n \partial_y \left( 2 A_h^M D \partial_y \left( \frac{V}{D} \right) + A_h^N \partial_y V \right) \right)_{i,j} \approx \frac{F_{Dyy}^{i,i+1} - F_{Dyy}^{i,j}}{\Delta x_{i,j}^{v} \Delta y_{i,j}^{v}}
\]  

(87)

with diffusive fluxes

\[
F_{Dyy}^{i,j} = \left( 2 A_h^M D_{V,i,j} \left( \frac{V_{i,j}}{D_{i,j}} - \frac{V_{i,j-1}}{D_{i,j-1}} \right) + A_h^N (V_{i,j} - V_{i,j-1}) \right) \frac{\Delta x_{i,j}^{v} \Delta y_{i,j}^{v}}{c_{i,j}}
\]  

(88)

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:

\[
\begin{align*}
\partial_t \eta &= -\partial_x U - \partial_y V \\
\partial_t U &= -gD \partial_x \eta + A_h^N \left( \partial_{xx} U + \partial_{yy} U \right) \\
\partial_t V &= -gD \partial_y \eta + A_h^N \left( \partial_{xx} V + \partial_{yy} V \right),
\end{align*}
\]  

(89)

which can be transformed into an equation for \( \partial_t \eta \) by derivation of the \( \eta \)-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 \( V \):

\[
\partial_t (\partial_t \eta) = gD \left( \partial_{xx} \eta + \partial_{yy} \eta \right) + A_h^N \left( \partial_{xx} (\partial_t \eta) + \partial_{yy} (\partial_t \eta) \right),
\]  

(90)

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 variables_2d, only: D,U,DU,UEx,V,DY,VEx,PP
- use getm_timers, only: tic,toc,TIM_UVDIFFUS
- $ use omp_lib

IMPLICIT NONE

INPUT PARAMETERS:

- REALTYPE, intent(in) :: Am
- integer, intent(in) :: An_method
- REALTYPE, intent(in) :: An(E2DFIELD),AnX(E2DFIELD)

INPUT/OUTPUT PARAMETERS:
**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

Original author(s): Hans Burchard

**LOCAL VARIABLES:**

integer :: i,j

### 7.4.1 bottom_friction - calculates the 2D bottom friction.

**INTERFACE:**

```plaintext
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. (70) on page 49. For $\text{runtype}=1$ (only vertically integrated calculations), the bottom roughness length is depending on the bed friction velocity $u_b^*$ and the molecular viscosity $\nu$:

$$
  z_b^0 = 0.1 \frac{\nu}{u_b^*} + (z_b^0)_{\text{min}},
$$

(91)

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 calculated 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

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

64
7.4.2 sealevel - using the cont. eq. to get the sealevel.

INTERFACE:

    subroutine sealevel

DESCRIPTION:

Here, the sea surface elevation is iterated according to the vertically integrated continuity equation
given in (56) on page 47.
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
    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:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:
7.4.3 sealevel_nan_check - Sweep the sealevel (z) for NaN values

INTERFACE:

subroutine sealevel_nan_check

DESCRIPTION:

The sea surface elevation (2d) variable is swept 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
use m2d, only: sealevel_check
use variables_2d, only: z
use exceptions, only: getm_error
IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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
7.4.4 sealevel_nandum. Helper routine to spot NaNs

INTERFACE:

    subroutine sealevel_nandum(a,b,idum)

USES:

INPUT PARAMETERS:

    REALTYPE, intent(in) :: a,b

INPUT/OUTPUT PARAMETERS:

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.

7.4.5 depth_update - adjust the depth to new elevations.

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, the sea surface elevation in the U- and V-points, zu and zv, respectively, and the drying value α defined in equation (5) on page 12 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,zu,DV,zv
    use getm_timers, only: tic, toc, TIM_DPTHUPDATE
    $ use omp_lib

    IMPLICIT NONE

INPUT PARAMETERS:
INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

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

7.4.6 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:

\[
H_{i,j}^u = \begin{cases} 
\frac{1}{2} (H_{i,j} + H_{i+1,j}), & \text{for } \text{vel_depth_method} = 0, \\
\min \{H_{i,j} + H_{i+1,j}\}, & \text{for } \text{vel_depth_method} = 1, \\
\min \{H_{i,j} + H_{i+1,j}\}, & \text{for } \text{vel_depth_method} = 2 \text{ and } \min \{H_{i,j}, H_{i+1,j}\} < D_{\text{crit}} \\
\frac{1}{2} (H_{i,j} + H_{i+1,j}), & \text{for } \text{vel_depth_method} = 2 \text{ and } \min \{H_{i,j}, H_{i+1,j}\} \geq D_{\text{crit}} 
\end{cases}
\]

The calculation of \(H_{i,j}^v\) 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 variables_2d, only: DU,DV  
use getm_timers, only: tic,toc,TIM_UVDENPSH  
IMPLICIT NONE

INPUT PARAMETERS:

integer, intent(in) :: vel_depth_method
INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:
Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

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

7.4.7  update_2d_bdy - update 2D boundaries every time step.

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,jmin,jmax,az
use domain, only: wi,wfj,wlj,nfj,nli,ei,efj,elj,sfj,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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

69
LOCAL VARIABLES:

logical, save : first=.true.
REALTYPE, save : time_array(1000),zbo(1000),zbn(1000)
REALTYPE, save : t,t1,t2
REALTYPE : a,amp,ratio,fac
integer : i,j,k,l,n
REALTYPE, parameter : FOUR=4.*_ONE_
7.5  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 shallow water criterium by Beckers and Deleersnijder (1993):

\[
\Delta t_{\text{max}} = \min_{i,j} \left\{ \frac{\Delta x_{i,j} \Delta y_{i,j}}{\sqrt{2 c_{i,j} \sqrt{\Delta x_{i,j}^2 + \Delta y_{i,j}^2}}} \right\}
\]  

(93)

with the local Courant number

\[
c_{i,j} = \sqrt{g H_{i,j}},
\]

(94)

where \( g \) is the gravitational acceleration and \( H_{i,j} \) is the local bathymetry value. In case that the chosen micro time step \( \Delta t_m \) is larger than \( \Delta t_{\text{max}} \), the program will be aborted. In any 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
    IMPPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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
7.5.1 have_bdy - checks whether this node has boundaries.

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
    use m2d, only: have_boundaries
    IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:

    integer :: i,j,k,m,n
    integer :: nbdy
    integer :: f,l
7.6  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:

\[
\begin{align*}
    u_{res} &= \frac{\int_{t_0}^{t_1} U \, d\tau}{\int_{t_0}^{t_1} D_u \, d\tau}, \\
    v_{res} &= \frac{\int_{t_0}^{t_1} V \, d\tau}{\int_{t_0}^{t_1} D_v \, d\tau},
\end{align*}
\]

(95)

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_u, res_v, res_du, res_dv, du, dv

IMPLICIT NONE

INPUT PARAMETERS:

    integer, intent(in) :: finish

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:
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

<table>
<thead>
<tr>
<th>quantity</th>
<th>description</th>
<th>unit</th>
<th>variable</th>
<th>routine name</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_k$</td>
<td>layer-int. $u$-transport</td>
<td>m$^2$s$^{-1}$</td>
<td>$uu$</td>
<td>uu_momentum</td>
<td>94</td>
</tr>
<tr>
<td>$q_k$</td>
<td>layer-int. $v$-transport</td>
<td>m$^2$s$^{-1}$</td>
<td>$vv$</td>
<td>vv_momentum</td>
<td>96</td>
</tr>
<tr>
<td>$\theta$</td>
<td>potential temperature</td>
<td>°C</td>
<td>$T$</td>
<td>do_temperature</td>
<td>133</td>
</tr>
<tr>
<td>$S$</td>
<td>salinity</td>
<td>psu</td>
<td>$S$</td>
<td>do_salinity</td>
<td>136</td>
</tr>
<tr>
<td>$C$</td>
<td>suspended matter</td>
<td>kg m$^{-3}$</td>
<td>$spm$</td>
<td>do_spm</td>
<td>140</td>
</tr>
</tbody>
</table>

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 8.6 on page 8.6.

The grid-related vertical velocity $\bar{w}_k$ is calculated directly from the layer-integrated continuity equation (25) which here done in the routine uu_momentum described on page 98.

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 147. 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 $\nu_t$ and eddy diffusivity $\nu'_t$ 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, documentation 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 147. Major input to the turbulence model are the shear squared $M^2 = (\partial_z u)^2 + (\partial_z v)^2$ 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 144.

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 146.

The module rivers (see section 8.35 on page 149) 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 8.36 described on page 153.

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 in the routines slow_bottom_friction, slow_advection and slow_diffusion, and they are added together in slow_terms, see the descriptions in sections 8.37, 8.38, 8.39 and 8.40 on pages 155, 156, 157 and 158, 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 8.41 on page 159) 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 8.42 on page 160) 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:

$$
\partial_t c^i + \partial_x (uc^i) + \partial_y (vc^i) + \partial_z (\alpha w_{c}^i c^i) - \partial_z (\nu'_z \partial_z c^i) - \partial_x (A^z_{hi} \partial_x c^i) - \partial_y (A^z_{hi} \partial_y c^i) = Q^i.
$$

(96)

Here, $\nu'_z$ denotes the vertical eddy diffusivity and $A^z_{hi}$ the horizontal diffusivity. Vertical migration of concentration with migration velocity $w_{c}^i$ (positive for upward motion) is considered as well. This could be e.g. 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 $\alpha$, (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:

$$
-\alpha w_{c}^i c^i + \nu'_z \partial_z c^i = F^i_s \quad \text{for } z = \zeta
$$

(97)

and

$$
-\alpha w_{c}^i c^i + \nu'_z \partial_z c^i = -F^i_b \quad \text{for } z = -H,
$$

(98)

with surface and bottom fluxes $F^i_s$ and $F^i_b$ directed into the domain, respectively.

At open lateral boundaries, the tracers $c^i_b$ 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 $\theta$ in °C and salinity $S$ in psu (practical salinity units):
\[ \partial_t \theta + \partial_z(u \theta) + \partial_y(v \theta) + \partial_z(w \theta) - \partial_z(\nu' \partial_z \theta) \]

\[ -\partial_z(A^0_s \partial_z \theta) - \partial_y(A^0_s \partial_y \theta) = \frac{\partial_z I}{c'_p \rho_0}, \]

\[ \partial_z S + \partial_z(u S) + \partial_y(v S) + \partial_z(w S) - \partial_z(\nu' \partial_z S) \]

\[ -\partial_z(A^0_s \partial_z S) - \partial_y(A^0_s \partial_y S) = 0. \] (99)

On the right hand side of the temperature equation (99) 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

\[ I(z) = I_0 \left( a e^{-\eta_2 z} + (1 - a) e^{-\eta_1 z} \right). \] (101)

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, \( \eta_1 \) and \( \eta_2 \), 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 \( \eta_1 \) and \( \eta_2 \).

At the surface, flux boundary conditions for \( T \) and \( S \) have to be prescribed. For the potential temperature, it is of the following form:

\[ \nu'_z E = \frac{Q_s + Q_l + Q_b}{c'_p \rho_0}, \quad \text{for } z = \zeta, \] (102)

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 (2000a). 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

\[ T_f = -0.0575 S_s + 1.710523 \cdot 10^{-3} S_s^{1.5} - 2.154996 \cdot 10^{-4} S_s^2 \approx -0.0575 S_s \] (103)

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

\[ Q_{surf} = \begin{cases} 
Q_s + Q_l + Q_b, & \text{for } T_s > T_f, \\
\max\{0, Q_s + Q_l + Q_b\}, & \text{else}.
\end{cases} \] (104)

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:

\[ \nu'_z E = \frac{S(Q_E - Q_P)}{\rho_0(0)}, \quad \text{for } z = \zeta, \] (105)

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 formulations 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:

$$\rho = \rho(\theta, S, p_0)$$ (106)

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 8.31 on page 142.

For convenient use in other subroutines the buoyancy $b$ as defined in (4 is calculated and stored in the GETM variable tt buoy.
8.4 Module 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 80. The actual calculation routines are called in integrate_3d (see section 8.4.2 on page 81) and are linked in from the library lib3d.a. After the simulation, the module is closed in clean_3d, see section 8.4.3 on page 82.

USES:

use exceptions
use parameters, only: avmmol
use domain, only: openbdy,maxdepth,vert_cord,az
use m2d, only: Am
use variables_2d, only: D,z,UEx,VEx
ifndef NO_BAROCLINIC
use temperature,only: init_temperature, do_temperature
use salinity, only: init_salinity, do_salinity
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_3d, only: init_advection_3d
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_
logical :: calc_temp=.true.
logical :: calc_salt=.true.
logical :: bdy3d=.false.
integer :: bdyfmt_3d,bdyramp_3d
character(len=PATH_MAX) :: bdyfile_3d
REALTYPE :: ip_fac=_ONE_
integer :: vel_check=0
REALTYPE :: min_vel=-4.,max_vel=4.

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

DESCRIPTION:

Here, the m3d namelist is read from getm.inp, and the initialisation of variables is called (see routine init_variables described on page 85). 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 $\Delta t$ is calculated from the micro time step $\Delta t_m$ and the split factor $M$. Then, in order to have the vertical coordinate system present already here, coordinates (see page 86) needs to be called, in order to enable proper interpolation of initial values for potential temperature $\theta$ and salinity $S$ for cold starts. Those initial values are afterwards read in via the routines init_temperature (page 132) and init_salinity (page 135). 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, &
  bdy3d,bdyfmt_3d,bdyramp_3d,bdyfile_3d, &
  bdy3d_tmrlx, bdy3d_tmrlx_ucut, &
  bdy3d_tmrlx_max,bdy3d_tmrlx_min, &
  vel_hor_adv,vel_ver_adv,vel_adv_split, &
  calc_temp,calc_salt, &
  avmbakc,avbback,ip_method,ip_ramp, &
  vel_check,min_vel,max_vel

80
8.4.2 integrate_3d - calls to do 3D model integration

INTERFACE:

subroutine integrate_3d(runtype,n)
use getm_timers, only: tic, toc, TIM_INTEGR3D
IMPLICIT NONE

INPUT PARAMETERS:

integer, intent(in) :: runtype,n

INPUT/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 159)
- do_bdy_3d: boundary conditions for $\theta$ and $S$ (see page 154)
- coordinates: layer heights ($\text{MUDFLAT}$ defined) (see page 86)
- bottom_friction_3d: bottom friction (see page 107)
- do_internal_pressure: internal pressure gradient (see page 110)
- uu_momentum_3d: layer-integrated $u$-velocity (see page 94)
- vv_momentum_3d: layer-integrated $v$-velocity (see page 96)
- coordinates: layer heights ($\text{MUDFLAT}$ not defined) (see page 86)
- ww_momentum_3d: grid-related vertical velocity (see page 98)
- uv_advect_3d: momentum advection (see page 100)
- uv_diffusion_3d: momentum diffusion (see page 104)
- stresses_3d: stresses (for GOTM) (see page 146)
- ss_nn: shear and stratification (for GOTM) (see page 144)
- gotm: interface and call to GOTM (see page 147)
- do_temperature: potential temperature equation (see page 133)
- do_salinity: salinity equation (see page 136)
- do_eqstate: equation of state (see page 143)
- do_spm: suspended matter equation (see page 140)
- do_getm_bio: call to GOTM-BIO (not yet released) (see page 147)
- slow_bottom_friction: slow bottom friction (see page 155)
- slow_advection: slow advection terms (see page 156)
- slow_diffusion: slow diffusion terms (see page 157)
- slow_terms: sum of slow terms (see page 158)
- stop_macro: finishing a 3d step (see page 160)
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.
```

### 8.4.3 clean_3d - cleanup after 3D run

**INTERFACE:**

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

**INPUT PARAMETERS:**

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**DESCRIPTION:**

Here, a call to the routine `clean_variables_3d` which however does not do anything yet.

**LOCAL VARIABLES:**
8.5 Module variables\_3d - global 3D related variables (Source File: variables\_3d.F90)

**INTERFACE:**

```fortran
module variables_3d
```

**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 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmin</td>
<td>2D [-]</td>
<td>lowest index in T-point</td>
</tr>
<tr>
<td>kumin</td>
<td>2D [-]</td>
<td>lowest index in U-point</td>
</tr>
<tr>
<td>kvmin</td>
<td>2D [-]</td>
<td>lowest index in V-point</td>
</tr>
<tr>
<td>kmin_pmz</td>
<td>2D [-]</td>
<td>lowest index in T-point (poor man’s z-coordinate)</td>
</tr>
<tr>
<td>kumin_pmz</td>
<td>2D [-]</td>
<td>lowest index in U-point (poor man’s z-coordinate)</td>
</tr>
<tr>
<td>kvmin_pmz</td>
<td>2D [-]</td>
<td>lowest index in V-point (poor man’s z-coordinate)</td>
</tr>
<tr>
<td>uu</td>
<td>3D [m(^2)s(^{-1})]</td>
<td>layer integrated u transport (p_k)</td>
</tr>
<tr>
<td>vv</td>
<td>3D [m(^2)s(^{-1})]</td>
<td>layer integrated v transport (q_k)</td>
</tr>
<tr>
<td>ww</td>
<td>3D [m(^{-1})]</td>
<td>grid-related vertical velocity (\bar{w}_k)</td>
</tr>
<tr>
<td>ho</td>
<td>3D [m]</td>
<td>old layer height in T-point</td>
</tr>
<tr>
<td>hn</td>
<td>3D [m]</td>
<td>new layer height in T-point</td>
</tr>
<tr>
<td>huo</td>
<td>3D [m]</td>
<td>old layer height in U-point</td>
</tr>
<tr>
<td>hun</td>
<td>3D [m]</td>
<td>new layer height in U-point</td>
</tr>
<tr>
<td>hvo</td>
<td>3D [m]</td>
<td>old layer height in V-point</td>
</tr>
<tr>
<td>hvn</td>
<td>3D [m]</td>
<td>new layer height in V-point</td>
</tr>
<tr>
<td>hcc</td>
<td>3D [-]</td>
<td>hydrostatic consistency index in T-points</td>
</tr>
<tr>
<td>uuEx</td>
<td>3D [m(^2)s(^{-2})]</td>
<td>sum of advection and diffusion for u-equation</td>
</tr>
<tr>
<td>vvEx</td>
<td>3D [m(^2)s(^{-2})]</td>
<td>sum of advection and diffusion for v-equation</td>
</tr>
<tr>
<td>num</td>
<td>3D [m(^2)s(^{-1})]</td>
<td>eddy viscosity on w-points (\nu)</td>
</tr>
<tr>
<td>nuh</td>
<td>3D [m(^2)s(^{-1})]</td>
<td>eddy diffusivity on w-points (\nu'_l)</td>
</tr>
<tr>
<td>tke</td>
<td>3D [m(^2)s(^{-2})]</td>
<td>turbulent kinetic energy (k)</td>
</tr>
<tr>
<td>eps</td>
<td>3D [m(^2)s(^{-3})]</td>
<td>turbulent dissipation rate (\epsilon)</td>
</tr>
<tr>
<td>SS</td>
<td>3D [s(^{-2})]</td>
<td>shear-frequency squared (M^2)</td>
</tr>
<tr>
<td>NN</td>
<td>3D [s(^{-2})]</td>
<td>Brunt-Väisälä frequency squared (N^2)</td>
</tr>
<tr>
<td>S</td>
<td>3D [psu]</td>
<td>salinity (S)</td>
</tr>
<tr>
<td>T</td>
<td>3D [(^\circ)C]</td>
<td>potential temperature (\theta)</td>
</tr>
<tr>
<td>rad</td>
<td>3D [Wm(^{-2})]</td>
<td>Short wave penetration</td>
</tr>
<tr>
<td>rho</td>
<td>3D [kg m(^{-3})]</td>
<td>density (\rho)</td>
</tr>
<tr>
<td>buoy</td>
<td>3D [m(^{-2})]</td>
<td>buoyancy (b)</td>
</tr>
<tr>
<td>idpdx</td>
<td>3D [m(^2)s(^{-2})]</td>
<td>x-component of internal pressure gradient</td>
</tr>
<tr>
<td>idpdy</td>
<td>3D [m(^2)s(^{-2})]</td>
<td>y-component of internal pressure gradient</td>
</tr>
<tr>
<td>spm</td>
<td>3D [kg m(^{-3})]</td>
<td>suspended matter concentration</td>
</tr>
<tr>
<td>spm_ws</td>
<td>3D [m(^{-1})]</td>
<td>settling velocity of suspended matter</td>
</tr>
<tr>
<td>spm_pool</td>
<td>2D [kg m(^{-2})]</td>
<td>bottom pool of suspended matter</td>
</tr>
<tr>
<td>uadv</td>
<td>3D [m(^{-1})]</td>
<td>interpolated x-component of momentum advection velocity</td>
</tr>
<tr>
<td>vadv</td>
<td>3D [m(^{-1})]</td>
<td>interpolated y-component of momentum advection velocity</td>
</tr>
</tbody>
</table>
wadv  3D  [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  [m s$^{-1}$] drag coefficient times current speed in U-point
rrv  2D  [m s$^{-1}$] drag coefficient times current speed in V-point
taus 2D  [m$^2$ s$^{-2}$] normalised surface stress (T-point)
taub 2D  [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
IMPLICIT NONE
PUBLIC DATA MEMBERS:
```

```
REALTYPE  :: dt,cnpar=0.9
REALTYPE  :: avmback=_ZERO_,avhback=_ZERO_
character(len=64) :: adv_schemes(7)
#endif
#include "static_3d.h"
#else
#include "dynamic_declarations_3d.h"
#endif

REALTYPE, allocatable :: cc3d(:,,:,:,:)
REALTYPE, allocatable :: ws3d(:,,:,:,:)

integer  :: size3d_field
integer  :: mem3d

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:
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 \texttt{dynamic_allocations_3d.h} (unless the compiler option \texttt{STATIC} is set). Furthermore, most variables are initialised here.

LOCAL VARIABLES:

    integer :: rc

8.5.2 clean_variables_3d - cleanup after 3D run.

INTERFACE:

    subroutine clean_variables_3d()
    IMPLICIT NONE

DESCRIPTION:

This routine cleans up after a 3D integration by doing nothing so far.
8.6 coordinates - defines the vertical coordinate (Source File: coordinates.F90)

INTERFACE:

subroutine coordinates(cord_type,cord_relax,maxdepth)

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 independently 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 \texttt{uv_depths} described on page 68.

Here, three different methods for the vertical layer distribution are coded:

1. Classical $\sigma$ coordinates where layer interfaces for each layer index have a fixed relative position $\sigma_k$ 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 \texttt{domain} namelist in \texttt{getm.inp} as $d_{du}$ and $d_{dl}$. In the first call to coordinates, the relative interface positions $d_{ga}$ are calculated as a one-dimensional vector (in case of non-equidistant $\sigma$ coordinates), and those are then multiplied with the water depths in all T-, U- and V-points to get the layer thicknesses.

2. Also $z$- (i.e. geopotential) coordinates are enabled in GETM in principle. However, they may not yet work and need further development. First of all, fixed $z$-levels are defined by means of zooming factors and the maximum water depth $H_{\text{max}}$:

$$z_k = H_{\text{max}} \left( \frac{\tanh((d_l + d_u)(1 + \sigma_k) - d_l) + \tanh(d_l)}{\tanh(d_l) + \tanh(d_u)} - 1 \right), \quad k = 0, \ldots, N \quad (107)$$

Then, layers are from the surface down filled into the T-point water column locally. When the last layer is shallower than $h_{\text{min}}$ (hard coded as local variable), the two last layers are combined. The index of the lowest layer is then stored in the integer field $k_{\text{min pmz}}$. Layer thicknesses in U- and V-points are then taken as the minimum values of adjacent thicknesses in T-points, and bottom indices $k_{\text{vmin pmz}}$ and $k_{\text{vmin pmz}}$ are taken as the maximum of adjacent $k_{\text{min pmz}}$ indices.

3. The third and so far most powerful method are the general vertical coordinates, discussed in section 4.1, see equations (14) - (19), which is basically an interpolation between equidistant and non-equidistant $\sigma$ coordinates. During the first call, a three-dimensional field $g_{\text{ga}}$ 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.

A fourth option will soon be the adaptive grids which have been conceptionally developed by Burchard and Beckers (2004).

USES:

\begin{verbatim}
#ifdef SLICE_MODEL
   use domain, only: imin,imax,jmin,jmax,kmax
   use variables_3d, only: kvmin,hvo,hvn
#endif
   use getm_timers, only: tic, toc,TIM_COORDS
ENDIMPLICIT NONE
\end{verbatim}

86
INPUT PARAMETERS:

- integer, intent(in) :: cord_type
- REALTYPE, intent(in) :: cord_relax
- REALTYPE, intent(in) :: maxdepth

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

- logical, save :: first=.true.
- #ifdef SLICE_MODEL
  - integer :: i,j,k
- #endif
8.7 equidistant and zoomed sigma-coordinates (Source File: sigma_coordinates.F90)

INTERFACE:

    subroutine sigma_coordinates(first)

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 independently 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 68.

Here, three different methods for the vertical layer distribution are coded:

1. Classical σ coordinates where layer interfaces for each layer index have a fixed relative position \( \sigma_k \) 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 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.

2. Also \( z \)- (i.e. geopotential) coordinates are enabled in GETM in principle. However, they may not yet work and need further development. First of all, fixed \( z \)-levels are defined by means of zooming factors and the maximum water depth \( H_{\text{max}} \):

\[
z_k = H_{\text{max}} \left( \frac{\tanh((d_l + d_u)(1 + \sigma_k) - d_l) + \tanh(d_l)}{\tanh(d_l) + \tanh(d_u)} - 1 \right), \quad k = 0, \ldots, N
\]

Then, layers are from the surface down filled into the T-point water column locally. When the last layer is shallower than \( h_{\text{min}} \) (hard coded as local variable), the two last layers are combined. The index of the lowest layer is then stored in the integer field \( k_{\text{min,pzm}} \). Layer thicknesses in U- and V-points are then taken as the minimum values of adjacent thicknesses in T-points, and bottom indices \( k_{\text{umin,pzm}} \) and \( k_{\text{vmin,pzm}} \) are taken as the maximum of adjacent \( k_{\text{min,pzm}} \) indices.

3. The third and so far most powerful method are the general vertical coordinates, discussed in section 4.1, see equations (14) - (19), which is basically an interpolation between equidistant and non-equidistant 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.

A fourth option will soon be the adaptive grids which have been conceptionally developed by Burchard and Beckers (2004).

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,hm,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
8.8 general vertical coordinates (Source File: general_coordinates.F90)

INTERFACE:

    subroutine general_coordinates(first,cord_relax,maxdepth)

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 independently 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 68.

Here, three different methods for the vertical layer distribution are coded:

1. Classical $\sigma$ coordinates where layer interfaces for each layer index have a fixed relative position $\sigma_k$ 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 $d_{du}$ and $d_{dl}$. In the first call to coordinates, the relative interface positions $d_{ga}$ are calculated as a one-dimensional vector (in case of non-equidistant $\sigma$ coordinates), and those are then multiplied with the water depths in all T-, U- and V-points to get the layer thicknesses.

2. Also $z$- (i.e. geopotential) coordinates are enabled in GETM in principle. However, they may not yet work and need further development. First of all, fixed $z$-levels are defined by means of zooming factors and the maximum water depth $H_{\text{max}}$:

\[
z_k = H_{\text{max}} \left( \frac{\tanh((d_l + d_u)(1 + \sigma_k) - d_l) + \tanh(d_l)}{\tanh(d_l) + \tanh(d_u)} - 1 \right), \quad k = 0, \ldots, N \quad (109)
\]

Then, layers are from the surface down filled into the T-point water column locally. When the last layer is shallower than $h_{\text{min}}$ (hard coded as local variable), the two last layers are combined. The index of the lowest layer is then stored in the integer field `kmin_pmz` layer thicknesses in U- and V-points are then taken as the minimum values of adjacent thicknesses in T-points, and bottom indices `kumin_pmz` and `kvmin_pmz` are taken as the maximum of adjacent `kmin_pmz` indices.

3. The third and so far most powerful method are the general vertical coordinates, discussed in section 4.1, see equations (14) - (19), which is basically an interpolation between equidistant and non-equidistant $\sigma$ 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.

A fourth option will soon be the adaptive grids which have been conceptionally developed by Burchard and Beckers (2004).

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

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**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
8.9 hybrid vertical coordinates - (z-sigma) (Source File: hybrid_coordinates.F90)

INTERFACE:

    subroutine hybrid_coordinates(first)

DESCRIPTION:

For Hans to do

USES:

#if 0
    use domain, only: imin,imax,jmin,jmax,kmax,H,HU,HV,az,au,av,min_depth
    use domain, only: ga,ddu,ddl,d_gamma,gamma_surf
    use variables_3d, only: dt,kmin,kumin,kvmin,ho,hn,huo,hun,hvo,hvn
    use variables_3d, only: sseo,ssen,ssuo,ssun,ssvo,ssvn
#endif

IMPLICIT NONE

INPUT PARAMETERS:

    logical, intent(in) :: first

REVISION HISTORY:

    Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

#if 0
    integer :: i,j,k,rc,kk
    REALTYPE, save, dimension(:), allocatable :: dga,be,sig,zlev
    REALTYPE, save, dimension(:,,:), allocatable :: gga
#endif
8.10  \texttt{hcc\_check} - hydrostatic consistency criteria (Source File: \texttt{hcc\_check.F90})

**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:

$$h^c_{i,j,k} = \max \left\{ \left| \frac{\partial_z z_k}{\Delta x} \left( h_{i,j,k} + h_{i+1,j,k} \right) \right|, \left| \frac{\partial_y z_k}{\Delta y} \left( h_{i,j,k} + h_{i,j+1,k} \right) \right| \right\}.$$  \hspace{1cm} (110)

For the numerical calculation it is used here that $\Delta x$ and $\Delta y$ 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
8.11 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, $f q_k$ 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 $e_x(k)$), 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 $C u \sqrt{u^2 + v^2}$ (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, $C u^2 + v^2$, 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,ax
    #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 STRUCTURE_FRICTION
    use variables_3d, only: sf
    #endif
    #ifndef NO_BAROCLINIC
    use variables_3d, only: idpdx
    #endif
    #ifdef UV_TVD
    use variables_3d, only: uadv,vadv,wadv,huadv,hvadv,hoadv,hnadv
    #endif
#endif
use halo_zones, only: update_3d_halo, wait_halo, U_TAG
use meteo, only: tauxx, 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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i,j,k,rc
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 :: gamma_i, rho_0i
integer :: status
8.12 vv_momentum_3d - $y$-momentum eq. (Source File: vv_momentum_3d.F90)

INTERFACE:

```fortran
subroutine vv_momentum_3d(n,bdy3d)
```

DESCRIPTION:

Here, the budget equation for layer-averaged momentum in eastern 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 $C_v\sqrt{u^2 + v^2}$ (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 $j = 3$. If the compiler option XZ_PLUME_TEST is set, a slope of $y_{slope}$ for bottom and isopycnals into the $y$-direction is prescribed, which has to be hard-coded as local variable.

USES:

```fortran
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,ax
#ifdef 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,vhu,vho,hv,vvEx,ww,hvu
use variables_3d, only: num,nuh,sseo,ssvn,rrv
use variables_3d, only: ssvo
#endif XZ_PLUME_TEST
use variables_3d, only: buoy
#endif STRUCTURE_FRICTION
use variables_3d, only: sf
#endif
```
#ifndef NO_BAROCLINIC
  use variables_3d, only: idpdy
#endif
#ifdef UV_TVD
  use variables_3d, only: uadv,vadv,wadv,huadv,hvadv,hoadv,hnadv
#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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

  integer :: i,j,k,rc
  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_
  #ifdef XZ_PLUME_TEST
    REALTYPE :: yslope=0.001
  #endif
  integer :: status
8.13  **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 must be zero, i.e. \( \bar{w}_{\text{max}} = 0 \). This is a good consistency 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
    #ifdef CALC_HALO_WW
    #ifndef CALC_HALO_WW
        use domain, only: az
    #endif
    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

**INPUT PARAMETERS:**

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

    Original author(s): Hans Burchard & Karsten Bolding

**LOCAL VARIABLES:**

    REALTYPE :: dtm1
    integer :: i,j,k
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:

\[ sf = C(z)\sqrt{u(z)^2 + v(z)^2}, \]  

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

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

REALTYPE :: dtm1
integer :: i,j,k
#ifdef STRUCTURE_FRICTION
REALTYPE :: cds(I2DFIELD)
#endif
8.15  uv_advect_3d - 3D momentum advection (Source File: uv_advect_3d.F90)

INTERFACE:

    subroutine uv_advect_3d(hor_adv, ver_adv, adv_split)

DESCRIPTION:

For the discretisation of the momentum advection terms, two conceptionally different methods have been implemented in GETM. The first is the straight-forward multidimensional advection scheme, which is here realised as the first-order upwind scheme, see paragraph Multidimensional approach on page 100.

In order to make use of the higher-order directional-split methods for tracers (see section 8.27.2), an alternative method is implemented, in which the complete advection step is first made, and then the resulting advection terms, which are needed for the calculation of the slow terms, see equations (64) and (65)) are calculated from this (see paragraph Directional-split approach on page 102).

The choice which of the two methods to be used is made by means of the compiler option UV_TVD which has to be set in the Makefile of the application in order to activate the more accurate but computationally more demanding high-order directional-split method. The effect of high-order advection can be impressively studied by means of the freshwater lens test case described in detail by Burchard and Bolding (2002).

When working with the option SLICE_MODEL, the calculation of all gradients in y-direction is suppressed.

Multidimensional approach  The advective terms in the momentum equation are discretised in a momentum-conservative form. This is carried out here for the advective terms in the u-equation (38) and the v-equation (39) (after multiplying these equations with mn).

First advection term in (38):

\[
\left(\frac{mn \partial x}{u_k p_k} \right)_{i,j,k} \approx \frac{1}{2} \left( \frac{p_{i+1,j,k} + p_{i-1,j,k}}{2} \right) \bar{u}_{i+1,j,k} \Delta y_{i+1,j} - \frac{1}{2} \left( \frac{p_{i,j,k} + p_{i-1,j,k}}{2} \right) \bar{u}_{i,j,k} \Delta y_{i,j}
\]

(112)

For an upwind scheme, the inter-facial velocities which are defined on T-points are here calculated as:

\[
\bar{u}_{i,j,k} = \begin{cases} 
  u_{i-1,j,k} & \text{for } \frac{1}{2} \left( p_{i,j,k} + p_{i-1,j,k} \right) > 0 \\
  u_{i,j,k} & \text{else.}
\end{cases}
\]

(113)

Second advection term in (38):

\[
\left(\frac{mn \partial y}{v_k p_k} \right)_{i,j,k} \approx \frac{1}{2} \left( \frac{q_{i+1,j,k} + q_{i,j,k}}{2} \right) \bar{u}_{i,j,k} \Delta x_{i,j}^+ - \frac{1}{2} \left( \frac{q_{i+1,j-1,k} + q_{i,j-1,k}}{2} \right) \bar{u}_{i,j-1,k} \Delta x_{i,j-1}^+
\]

(114)

For an upwind scheme, the inter-facial velocities which are defined on X-points are here calculated as:
\[ \tilde{u}_{i,j,k} = \begin{cases} u_{i,j,k} & \text{for } \frac{1}{2}(q_{i+1,j,k} + q_{i,j,k}) > 0 \\ u_{i,j+1,k} & \text{else.} \end{cases} \] (115)

First advection term in (39):

\[
\left( mn \partial_x \left( \frac{v_k q_k}{n} \right) \right)_{i,j,k} \approx \frac{1}{2} \left( (q_{i+1,j,k} + p_{i,j,k}) \tilde{v}_{i,j,k} \Delta y_{i,j} - \frac{1}{2} (q_{i,j+1,k} + q_{i,j,k}) \tilde{v}_{i-1,j,k} \Delta y_{i-1,j} \right) \Delta x_{i,j} \Delta y_{i,j} \] (116)

For an upwind scheme, the interfacial velocities which are defined on X-points are here calculated as:

\[ \tilde{v}_{i,j,k} = \begin{cases} v_{i,j,k} & \text{for } \frac{1}{2} (p_{i+1,j,k} + p_{i,j,k}) > 0 \\ v_{i+1,j,k} & \text{else.} \end{cases} \] (117)

Second advection term in (39):

\[
\left( mn \partial_y \left( \frac{v_k q_k}{m} \right) \right)_{i,j,k} \approx \frac{1}{2} \left( (q_{i,j+1,k} + q_{i,j,k}) \tilde{v}_{i,j+1,k} \Delta x_{i,j} - \frac{1}{2} (q_{i,j,k} + q_{i,j-1,k}) \tilde{v}_{i,j,k} \Delta x_{i,j} \right) \Delta x_{i,j} \Delta y_{i,j} \] (118)

For an upwind scheme, the interfacial velocities which are defined on T-points are here calculated as:

\[ \tilde{v}_{i,j,k} = \begin{cases} v_{i,j-1,k} & \text{for } \frac{1}{2} (q_{i,j,k} + q_{i,j-1,k}) > 0 \\ v_{i,j,k} & \text{else.} \end{cases} \] (119)

The vertical advection terms in equations (38) and (39) can be discretised in an upstream scheme as well.

Vertical advective flux in (38):

\[
(\tilde{w}_k \tilde{u}_k)_{i,j} \approx \frac{1}{2} (w_{i+1,j,k} + w_{i,j,k}) \tilde{u}_{i,j,k} \] (120)

with

\[ \tilde{u}_{i,j,k} = \begin{cases} u_{i,j,k} & \text{for } \frac{1}{2} (w_{i+1,j,k} + w_{i,j,k}) > 0, \\ u_{i,j,k+1} & \text{else.} \end{cases} \] (121)

Vertical advective flux in (39):

\[
(\tilde{w}_k \tilde{v}_k)_{i,j} \approx \frac{1}{2} (w_{i,j+1,k} + w_{i,j,k}) \tilde{v}_{i,j,k} \] (122)

with

\[ \tilde{v}_{i,j,k} = \begin{cases} v_{i,j,k} & \text{for } \frac{1}{2} (w_{i,j+1,k} + w_{i,j,k}) > 0, \\ v_{i,j,k+1} & \text{else.} \end{cases} \] (123)
Directional-split approach  Multidimensional treatment of advective terms in three-dimensional models is often quite unhandy, especially when higher-order advection schemes are needed. On the other hand, directional-split methods (which update the advected fields in each directional step and then "forget" the advection terms) as discussed in section 8.27.2 on page 120, cannot directly be used for momentum advection when the models are based on mode splitting as e.g. GETM. The reason for this is that the three-dimensional advection terms are also needed for calculating the slow terms of the barotropic (external) mode, see equations (64) and (65).

The procedure suggested here is as follows. First, the pure momentum advection equations are formally solved with the directional-split method described in section 8.27.2:

\[
\begin{align*}
\partial_t p_k + \partial_x(u_k p_k) + \partial_y(v_k p_k) + \bar{w}_k \bar{u}_k - \bar{w}_{k-1} \bar{u}_{k-1} &= 0, \\
\partial_t q_k + \partial_x(u_k q_k) + \partial_y(v_k q_k) + \bar{w}_k \bar{v}_k - \bar{w}_{k-1} \bar{v}_{k-1} &= 0.
\end{align*}
\]

The new solutions \( \hat{p}_{i,j,k} \) and \( \hat{q}_{i,j,k} \) are however not further used, but instead the resulting advective terms \( -(\hat{p}_{i,j,k} - p_{i,j,k})/\Delta t \) and \( -(\hat{q}_{i,j,k} - q_{i,j,k})/\Delta t \) are later applied to the momentum equations (together with other processes such as horizontal diffusion, pressure gradients, etc.) and also used for the calculation of the slow terms in (64) and (65).

With this method, all higher-order directional-split advection schemes are now available for the momentum advection. The advective fluxes needed for this have to be averaged from the conservative advective fluxes resulting from the continuity equation Continuity will still be retained due to the linearity of the continuity equation.

! USES:

use domain, only: imin,imax,jmin,jmax,kmax,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 variables_3d, only: dt,kumin,kvmin,uu,vv,ww,hun,hvn,huo,hvo,uuEx,vvEx
#ifdef UV_TVD
use variables_3d, only: uadv,vadv,wadv,huadv,hvadv,hoadv,hnadv
#endif
use advection_3d, only: do_advection_3d
use halo_zones, only: update_3d_halo,wait_halo,U_TAG,V_TAG
use getm_timers, only: tic, toc, TIM_UVADV3D, TIM_UVADV3DH
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:
integer, intent(in) :: hor_adv,ver_adv,adv_split

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:
Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:
integer :: i,j,k,rc
REALTYPE :: PP(imin-1:imax+1,jmin-1:jmax+1,1:kmax)
REALTYPE, POINTER :: www(:)
#endif
integer :: azadv(I2DFIELD)
integer :: auadv(I2DFIELD)
integer :: avadv(I2DFIELD)
REALTYPE :: dxuadv(I2DFIELD)
REALTYPE :: dxvadv(I2DFIELD)
REALTYPE :: dyuadv(I2DFIELD)
REALTYPE :: dyvadv(I2DFIELD)
REALTYPE :: area_inv(I2DFIELD)
REALTYPE :: AH=_ZERO_
REALTYPE :: dti,dxdyi
#endif
8.16  uv_diffusion_3d - hor. momentum diffusion (Source File: uv_diffusion_3d.F90)

INTERFACE:

subroutine uv_diffusion_3d(Am)

DESCRIPTION:

Here, the horizontal diffusion terms are discretised in momentum-conserving form as well. For simplicity, this shown here for these terms as they appear in equations (38) and (39), i.e. without multiplying them by \(mn\).

First horizontal diffusion term in (38):

\[
\partial_X \left( \frac{2A_M h_k}{n} m \partial_X u_k \right)_{i,j,k} \approx 2A^M_{i+1,j,k} \Delta y_{i,j} h_{i+1,j,k} \frac{u_{i+1,j,k} - u_{i,j,k}}{\Delta x_{i+1,j}} - 2A^M_{i,j,k} \Delta y_{i,j} h_{i,j,k} \frac{u_{i,j,k} - u_{i-1,j,k}}{\Delta x_{i,j}}
\]  

(126)

Second horizontal diffusion term in (38):

\[
\partial_Y \left( \frac{A_M h_k}{m} (n \partial_Y u_k + m \partial_X v_k) \right)_{i,j,k} \approx \frac{1}{4} \left( A^M_{i+1,j+1,k} + A^M_{i,j+1,k} + A^M_{i+1,j,k} + A^M_{i+1,j+1,k} \right) h^+_i,j,k \Delta x^+_i,j \times \left( \frac{u_{i,j+1,k} - u_{i,j,k}}{\Delta y^+_i,j} - \frac{v_{i+1,j,k} - v_{i,j,k}}{\Delta x^+_i,j} \right)
\]

- \[
\frac{1}{4} \left( A^M_{i+1,j,k} + A^M_{i,j,k} + A^M_{i+1,j-1,k} + A^M_{i,j-1,k} \right) h^+_i,j-1 \Delta x^+_i,j-1 \times \left( \frac{u_{i,j,k} - u_{i,j-1,k}}{\Delta y^+_i,j-1} - \frac{v_{i+1,j,k} - v_{i,j,k}}{\Delta x^+_i,j-1} \right)
\]  

(127)

First horizontal diffusion term in (39):

\[
\partial_Y \left( \frac{2A_M h_k}{m} n \partial_Y v_k \right)_{i,j,k} \approx 2A^M_{i,j+1,k} \Delta x_{i,j+1} h_{i,j+1,k} \frac{v_{i,j+1,k} - v_{i,j,k}}{\Delta y_{i,j+1}} - 2A^M_{i,j,k} \Delta x_{i,j} h_{i,j,k} \frac{v_{i,j,k} - v_{i,j-1,k}}{\Delta y_{i,j}}
\]  

(128)
Second horizontal diffusion term in (39):

\[
\partial_x \left( \frac{A^M h_k}{n} (n \partial_y u_k + m \partial_x v_k) \right)_{i,j,k} \approx \frac{1}{4} \left( A^M_{i,j+1,k} + A^M_{i,j+1,k} + A^M_{i,j,k} + A^M_{i,j+1,k} \right) h^+_{i,j,k} \Delta x^+_{i,j} \\
\times \left( \frac{u_{i,j+1,k} - u_{i,j,k}}{\Delta y^+_{i,j}} - \frac{v_{i+1,j,k} - v_{i,j,k}}{\Delta x^+_{i,j}} \right)
\]

\[
- \frac{1}{4} \left( A^M_{i,j+1,k} + A^M_{i-1,j+1,k} + A^M_{i-1,j,k} + A^M_{i,j,k} \right) h^+_{i-1,j,k} \Delta x^+_{i-1,j} \\
\times \left( \frac{u_{i-1,j+1,k} - u_{i-1,j,k}}{\Delta y^+_{i-1,j}} - \frac{v_{i,j-1,k} - v_{i,j,k}}{\Delta x^+_{i-1,j}} \right)
\]

(129)

It is assumed here that the horizontal momentum diffusivities \( A^M_{i,j,k} \) are located on the T-points. For the case of a slice model simulation (compiler option \texttt{SLICE\_MODEL} activated) the diffusive fluxes in \( y \)-direction are set to zero.

**USES:**

use domain, only: imin,imax,jmin,jmax,kmax,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 variables_3d, only: kumin,kvmin,uu,vv,ww,hn,hvn,uuEx,vvEx

use getm_timers, only: tic, toc, TIM_UVDIFF3D

$ use omp_lib

IMPLICIT NONE

**INPUT PARAMETERS:**

REALTYPE, intent(in) :: Am

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

Original author(s): Hans Burchard & Karsten Bolding

**LOCAL VARIABLES:**

integer :: i,j,k,ii,jj,kk

REALTYPE :: PP(imin-1:imax+1,jmin-1:jmax+1,kmax+1:1:kmax)
8.17  tke_eps_advect_3d - 3D turbulence advection (Source File: tke_eps_advect_3d.F90)

INTERFACE:

subroutine tke_eps_advect_3d(hor_adv, ver_adv, adv_split)

DESCRIPTION:

This routine carries out advection of the prognostic turbulence quantities \textit{tke} (turbulent kinetic energy, \(k\)) and \textit{eps} (length scale related turbulence quantity, e.g. dissipation rate of \(k\), \(\varepsilon\), or turbulent frequency, \(\omega = \varepsilon/k\). Here, the TVD advection schemes are used which are also used for the momentum advection.

\textit{USES:}

use domain, only: imin, imax, jmin, jmax, kmax, az, au, av

#ifdef SPHERICAL || defined(CURVILINEAR)
use domain, only: dxu, dxv, dyu, dyv, arcd1
#else
use domain, only: dx, dy, ardi
#endif
use variables_3d, only: dt, kumin, kvmin, uu, vv, ww, hun, hvn, ho, hn, uuEx, vvEx
#ifdef UV_TVD
use variables_3d, only: uadv, vadv, wadv, huadv, hvadv, hoadv, hnadv
#endif
use variables_3d, only: tke, eps
use advection_3d, only: do_advection_3d
use halo_zones, only: update_3d_halo, wait_halo, U_TAG, V_TAG

IMPLICIT NONE

INPUT PARAMETERS:

integer, intent(in) :: hor_adv, ver_adv, adv_split

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i, j, k
#ifdef UV_TVD
REALTYPE :: dxuadv(I2DFIELD)
REALTYPE :: dxvadv(I2DFIELD)
REALTYPE :: dyuadv(I2DFIELD)
REALTYPE :: dyvadv(I2DFIELD)
REALTYPE :: area_inv(I2DFIELD)
REALTYPE :: AH=_ZERO_
REALTYPE :: dxxyi
#endif
8.18  **bottom_friction_3d - bottom friction** (Source File: bottom_friction_3d.F90)

**INTERFACE:**

```fortran
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,

$$r \sqrt{u_b^2 + v_b^2}$$

with the velocity components of the bottom layer, $u_b$ and $v_b$, and the drag coefficient

$$r = \left( \frac{\kappa}{\ln \left( \frac{0.5h_b + z_b^0}{z_b^0} \right)} \right)^2,$$

is calculated and provided as output parameters $rru$ (for U-points) and $rrv$ (for V-points). The layer height $h_1$ in (131) 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 $z_b^0$ according to (91), see page 64. For a derivation of (131), see section 5.4 on page 28.

**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, kvmn, uu, vv, huo, hun, hv0, hvo, rru, rrv
- use getm_timers, only: tic, toc, TIM_BOTTFRICT3D
- use omp_lib

**INPUT PARAMETERS:**

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**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)
8.19 Module 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)):

$$h_k \left( \frac{1}{2} h_N (\partial_x^* b)_N + \sum_{j=k}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial_x^* b)_j \right)$$  \hspace{1cm} (132)

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

$$h_k \left( \frac{1}{2} h_N (\partial_y^* b)_N + \sum_{j=k}^{N-1} \frac{1}{2} (h_j + h_{j+1}) (\partial_y^* b)_j \right)$$  \hspace{1cm} (133)

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 ?, 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,hn,hun,hvn,idpdx,idpdy,buoy,ssun,ssvn,ssen
IMPLICIT NONE

109
PUBLIC DATA MEMBERS:

```fortran
  public init_internal_pressure, do_internal_pressure
  integer, public :: ip_method=1
#ifdef STATIC
  REALTYPE :: zz(I3DFIELD)
#endif SUBSTR_INI_PRESS
  REALTYPE :: idpdx0(I3DFIELD),idpdy0(I3DFIELD)
#else
  REALTYPE, allocatable :: zz(:,,:,:)
#endif SUBSTR_INI_PRESS
  REALTYPE, allocatable :: idpdx0(:,,:,:),idpdy0(:,,:,:)
#endif
```

PRIVATE DATA MEMBERS:

```fortran
  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

8.19.1 init_internal_pressure - initialising internal pressure gradient

INTERFACE:

```fortran
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

8.19.2 do_internal_pressure - internal pressure gradient

INTERFACE:

```fortran
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:**

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

**INPUT PARAMETERS:**

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**LOCAL VARIABLES:**

```fortran
integer :: i, j, k
logical, save :: first=.true.
```
8.20 ip_blumberg_mellor - (Source File: ip_blumberg_mellor.F90)

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 \((\partial^*_y b)_k\) (in the case of the \(u\)-equation), is discretised between two vertically adjacent velocity points:

\[
\frac{1}{2}(h_{i,j,k} + h_{i,j,k+1}) (m \partial^*_y b)_{i,j,k} \\
\approx \frac{1}{2}(h_{i,j,k}^u + h_{i,j,k+1}^u) \frac{1}{\Delta x_{i,j}}(b_{i+1,j,k+1} + b_{i+1,j,k}) - \frac{1}{2}(b_{i,j,k+1} + b_{i,j,k}) \\
- \frac{z_{i+1,j,k}^i - z_{i,j,k}^i}{\Delta x_{i,j}^u} \left( \frac{1}{2}(b_{i+1,j,k+1} + b_{i,j,k+1}) - \frac{1}{2}(b_{i+1,j,k} + b_{i,j,k}) \right),
\]

(134)

where \(z_{i,j,k}^i\) 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 (134) 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

112
8.21 ip_blumberg_mellor_lin (Source File: ip_blumberg_mellor_lin.F90)

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.

\[
\frac{1}{2}(h_{i,j,k} + h_{i,j,k+1}) \left( m \partial^y b \right)_{i,j,k} 
\]

\[
\approx \frac{1}{2}(\frac{1}{2}(b_{i+1,j,k+1} + b_{i,j,k}) - \frac{1}{2}(b_{i,j,k+1} + b_{i-1,j,k})) 
\]

\[
\frac{1}{2}(\frac{1}{2}(z_{c,i+1,j,k+1} + z_{c,i+1,j,k}) - \frac{1}{2}(z_{c,i,j,k+1} + z_{c,i,j,k})) 
\]

\[
- \frac{1}{2} \left( \frac{1}{2}(z_{c,i+1,j,k+1} - z_{c,i-1,j,k}) + \frac{b_{i,j,k+1} - b_{i,j,k}}{2} \right) \right),
\]

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,dxz1,dyz1,dxz1,dyz1
REALTYPE :: dzn2,dzr1,dxr1,dyr1,dy1,aa,bb,cc

113
INTERFACE:

    subroutine ip_z_interpol()

DESCRIPTION:

Here, the horizontal gradients of buoyancy, \( \frac{\partial^*}{\partial x} b \) and \( \frac{\partial^*}{\partial y} b \), 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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>i,j,k, rc</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>dxm1, dym1</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>grdl, grdu, buoy1, prgr, dxz, dyz</td>
</tr>
<tr>
<td>integer</td>
<td>kplus, kminus</td>
</tr>
<tr>
<td>REALTYPE, POINTER</td>
<td>zx(:)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>buoyplus, buoyminus</td>
</tr>
</tbody>
</table>
8.23  ip_song_wright (Source File: ip_song_wright.F90)

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:

\[
\frac{1}{2} (h_{i,j,k} + h_{i,j,k+1}) (m \frac{\partial^* y b}{\Delta x_{i,j}})_{i,j,k} \\
\approx \frac{1}{3} (b_{i+1,j,k+1} + b_{i,j,k+1}) (h^{c}_{i+1,j,k+1} + h^{c}_{i,j,k}) - \frac{1}{4} (b_{i,j,k+1} + b_{i,j,k}) (h^{c}_{i,j,k+1} + h^{c}_{i,j,k}) \\
- \left[ \frac{1}{2} (b_{i+1,j,k+1} + b_{i,j,k+1}) \frac{z^{c}_{i+1,j,k+1} - z^{c}_{i,j,k+1}}{\Delta x_{i,j}} \right] \\
- \frac{1}{2} (b_{i+1,j,k} + b_{i,j,k}) \frac{z^{c}_{i+1,j,k} - z^{c}_{i,j,k}}{\Delta x_{i,j}} \\
\] (136)

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
8.24  ip_chu_fan (Source File: ip_chu_fan.F90)

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
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 monotized by using harmonic mean instead of linear averages to interpolate slopes. Exact anti-symmetry of the density Jacobian

\[ J(\rho, zz) = -J(zz, \rho) \]  \hspace{1cm} (137)

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

\[
-\frac{1}{\rho_0} \partial_x p = -\frac{1}{\rho_0} \partial_x p_0 - g \partial_x \eta + \text{buoy}(\eta) \partial_x \eta + \int_{\eta}^{\sigma} J(\text{buoy}, zz) d\sigma \]  \hspace{1cm} (138)

Details about the calculation of the integral over the Jacobian in (138) can be found in Shchepetkin and McWilliams, 2003.

If parameter \text{OneFifth} (below) is set to zero, the scheme should become identical to standard Jacobian.

USES:

    use internal_pressure
    use variables_3d, only: hn,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)
8.26  ip_stelling_vankester (Source File: ip_stelling_vankester.F90)

INTERFACE:

    subroutine ip_stelling_vankester()

DESCRIPTION:

Here, the horizontal gradients of buoyancy, \((\partial x^k b)_k\) and \((\partial y^k b)_k\), are calculated as suggested in Stelling and van Kester (1994). The horizontal gradient of buoyancy is calculated with defining \(k_{\text{max}}\) 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 inconsistently 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 :: lvl(:)
    integer, POINTER :: m(:)
    integer, POINTER :: n(:)
8.27 Module 3D advection (Source File: advection_3d.F90)

INTERFACE:

module advection_3d

DESCRIPTION:

This module do 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 halo_zones, only: update_3d_halo,wait_halo,D_TAG
IMPLICIT NONE
private

PUBLIC DATA MEMBERS:

public init_advection_3d, do_advection_3d

# ifdef STATIC
REALTYPE, public : cu(I3DFIELD)
REALTYPE, public : hi(I3DFIELD)
REALTYPE, public : hio(I3DFIELD)
# else
REALTYPE, public, dimension(:,:,:), allocatable : hi,hio,cu
# endif

integer, public, parameter : UPSTREAM=1,UPSTREAM_SPLIT=2,P2=3
integer, public, parameter : Superbee=4,MUSCL=5,P2_PDM=6,FCT=7
REALTYPE, public, parameter : one6th=1./6.
REALTYPE, public, parameter : ONE=_ONE_,TWO=2.*_ONE_

!PRIVATE DATA MEMBERS:

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:

integer :: advection_method
8.27.1 init_advection_3d

INTERFACE:

```fortran
subroutine init_advection_3d(method)
```

DESCRIPTION:

Here, memory for some variables is allocated, which are then initialised to zero.

```fortran
INPUT PARAMETERS:
integer, intent(in) :: method
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

```fortran
integer :: rc
```

8.27.2 do_advection_3d - 3D advection schemes

INTERFACE:

```fortran
subroutine do_advection_3d(dt,f,uu,vv,ww,hun,hvn,ho,hn, &
  delxu,delyu,delxv,delyv,area_inv, &
  az,au,av,hor_adv,ver_adv,adv_split,AH)
```

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, but is not coded yet. Inside this advection routine, it does not matter where the advected variable is located on the grid. All finite volume fluxes and geometric coefficients need to be calculated before `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):

\[
\left( mn \frac{\partial y}{\partial x} \left( \frac{p_r c_k}{n} \right) \right)_{i,j} \approx \frac{p_{i,j,k} c_{i,j,k}^u \Delta y_{i,j}^u - p_{i-1,j,k} c_{i-1,j,k}^u \Delta y_{i-1,j}^u}{\Delta x_{i,j} \Delta y_{i,j}^u} \tag{139}
\]

Second advection term in (40):

\[
\left( mn \frac{\partial y}{\partial y} \left( \frac{q_r c_k}{m} \right) \right)_{i,j} \approx \frac{q_{i,j,k} c_{i,j,k}^v \Delta y_{i,j}^v - q_{i,j-1,k} c_{i,j-1,k}^v \Delta y_{i,j-1}^v}{\Delta x_{i,j} \Delta y_{i,j}^v} \tag{140}
\]

120
Vertical advective fluxes in (40):

\[
(\tilde{w}_{k} \tilde{c}_{k})_{i,j} \approx w_{i,j,k} \tilde{c}_{i,j,k}'.
\] (141)

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 8.27.3 - 8.27.9. However, as said above, in the same way these routines may be applied to quantities on U-, V-, and W-points, if the transports and geometric coefficients are properly calculated. There are various combinations of advection schemes possible. The first selection is whether a one-step 3D first-order upstream method is chosen, or a fractional step method. The next selection is (if a fractional step method is selected) how to do the fractional steps (selection on \texttt{adv_split}). There are different options,

1. directional split with subsequent full steps in \(x-\), \(y-\) and \(z-\)direction,
2. split with subsequent half steps in \(x-\) and \(y-\)direction, a full step in \(z-\)direction, and half steps in \(y-\) and \(x-\)direction.
3. directional split into a 2D horizontal step and a 1D vertical step.

For the 1D directional-split schemes, first-order upstream, ULTIMATE QUICKEST, and the Total Variation Diminishing (TVD) schemes Superbee, MUSCL, and \(P_2\)PDM are available. For the 2D horizontal step, an upstream scheme and a Flux-Corrected Transport (FCT) scheme have been coded.

If the compiler option \texttt{ITERATE_VERT_ADV} is chosen, the vertical advection is iterated as many times with reduced time step that the CFL criterium for vertical advection is fulfilled, see the routine \texttt{w_split_it_adv}.

With the compiler option \texttt{SLICE_MODEL}, the advection in \(y-\)direction is not executed.

\texttt{USES:}

\begin{verbatim}
use getm_timers, only: tic, toc, TIM_ADVECT3DTOT
implicit none
\end{verbatim}

\texttt{INPUT PARAMETERS:}

\begin{verbatim}
REALTYPE, intent(in) :: uu(I3DFIELD)  ! layer-integrated x-transport
REALTYPE, intent(in) :: vv(I3DFIELD)  ! layer-integrated y-transport
REALTYPE, intent(in) :: ww(I3DFIELD)  ! grid-related vertical velocity
REALTYPE, intent(in) :: ho(I3DFIELD)  ! old height of finite volume box
REALTYPE, intent(in) :: hn(I3DFIELD)  ! new height of finite volume box
REALTYPE, intent(in) :: hun(I3DFIELD) ! height of x-interfaces
REALTYPE, intent(in) :: hvn(I3DFIELD) ! height of y-interfaces
REALTYPE, intent(in) :: delxu(I2DFIELD) ! length of y-interface
REALTYPE, intent(in) :: delvu(I2DFIELD) ! length of u-interface
REALTYPE, intent(in) :: delxy(I2DFIELD) ! length of y-interface
REALTYPE, intent(in) :: delyu(I2DFIELD) ! length of u-interface
REALTYPE, intent(in) :: area_inv(I2DFIELD) ! inverse of horizontal box area
REALTYPE, intent(in) :: dt ! advection time step
REALTYPE, intent(in) :: AH ! constant horizontal diffusivity
integer, intent(in) :: az(E2DFIELD) ! mask for box centre (1: water)
integer, intent(in) :: au(E2DFIELD) ! mask for u-transport (1: water)
integer, intent(in) :: av(E2DFIELD) ! mask for v-transport (1: water)
integer, intent(in) :: hor_adv ! selection for horizontal scheme
integer, intent(in) :: ver_adv ! selection for vertical scheme
integer, intent(in) :: adv_split ! selection for split mode
\end{verbatim}

\texttt{INPUT/OUTPUT PARAMETERS:}
REALTYPE, intent(inout) :: f(I3DFIELD)

OUTPUT PARAMETERS:

LOCAL VARIABLES:

REALTYPE, parameter :: a1=0.5*ONE,a2=ONE
integer :: k

8.27.3 upstream_adv - 3D upstream advection

INTERFACE:

subroutine upstream_adv(dt,f,uu,vv,ww,ho,hn, &
delxv,delyu,delxu,delyv,area_inv,az,AH)

DESCRIPTION:

Here the advection terms (139), (140) and (141) are calculated by a upstream scheme and the
advection is done in one single time step.

This means that the variable to advect is approximated at the interfaces as:

\[ c_{i,j,k}^u = \begin{cases} 
  c_{i,j,k} & \text{for } p_{i,j,k} \geq 0, \\
  c_{i+1,j,k} & \text{else},
\end{cases} \tag{142} \]

\[ c_{i,j,k}^v = \begin{cases} 
  c_{i,j,k} & \text{for } q_{i,j,k} \geq 0, \\
  c_{i,j+1,k} & \text{else},
\end{cases} \tag{143} \]

and

\[ c_{i,j,k}^w = \begin{cases} 
  c_{i,j,k} & \text{for } w_{i,j,k} \geq 0, \\
  c_{i,j,k+1} & \text{else}. \tag{144} \end{cases} \]

It should be noted that the quantities \( p_{i,j,k}, q_{i,j,k}, w_{i,j,k} \) are defined relative to the finite volume
box, and may (if \( c_{i,j,k} \) defines velocities) be calculated by means of interpolation.

Then, the one-step scheme is executed as follows:

\[
\begin{align*}
  h_{i,j,k}^n c_{i,j,k}^n &= h_{i,j,k}^o c_{i,j,k}^o \\
  -\Delta t \left( p_{i,j,k} e_{i,j,k}^u \Delta y_{i,j}^u - p_{i-1,j,k} e_{i-1,j,k}^u \Delta y_{i-1,j}^u \right) h_{i,j,k}^n c_{i,j,k}^n &+ q_{i,j,k} e_{i,j,k}^v \Delta y_{i,j}^v - q_{i,j-1,k} e_{i,j-1,k}^v \Delta y_{i,j-1}^v & + w_{i,j,k} e_{i,j,k}^w - w_{i,j,k-1} e_{i,j,k-1}^w \\
  \Delta x_{i,j}^u &\Delta y_{i,j}^u & \Delta x_{i,j}^v &\Delta y_{i,j}^v & \Delta x_{i,j}^w &\Delta y_{i,j}^w 
\right),
\end{align*}
\]

with the suffices \( n \) and \( o \) denoting new and old values, respectively.

USES:

122
use domain, only: imin,imax,jmin,jmax,kmax
use advection_3d, only: cu
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

REALTYPE, intent(in) :: uu(I3DFIELD), vv(I3DFIELD)
REALTYPE, intent(in) :: ww(I3DFIELD)
REALTYPE, intent(in) :: ho(I3DFIELD), hn(I3DFIELD)
REALTYPE, intent(in) :: delxv(I2DFIELD), delyv(I2DFIELD)
REALTYPE, intent(in) :: delxu(I2DFIELD), delyu(I2DFIELD)
REALTYPE, intent(in) :: area_inv(I2DFIELD), dt, AH
integer, intent(in) :: az(E2DFIELD)

INPUT/OUTPUT PARAMETERS:

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

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: rc, i, ii, j, jj, k, kk
#ifdef USE_ALLOCATED.Arrays
REALTYPE, dimension(:, :, :), allocatable :: adv
#else
REALTYPE :: adv(I3DFIELD)
#endif

8.27.4 upstream_2dh_adv - 2D upstream advection

INTERFACE:

subroutine upstream_2dh_adv(dt, f, uu, vv, ho, hn, hun, hvn, &
  delxv, delyv, delxu, delyu, area_inv, az, AH)

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.

Before or after this 2D-horizontal advection is executed, a 1D vertical advection step, possibly with another scheme, needs to be carried out.

The advection terms are calculated according to (139) and (140) and the interface fluxes are again calculated according to (142) and (143).
However, here in contrast to the one-step advection scheme with (145) implemented in `upstream_adv`, first the complete horizontal and then the complete vertical advection needs to be executed. For consistency and conservation reasons, a partial step for the continuity equation needs to be executed as well. This is done as follows:

\[
h_{n_i,j,k}^{c_i,j,k} = h_{o_i,j,k}^{c_i,j,k} - \Delta t \left( \frac{p_{i,j,k} u_{c_i,j,k} \Delta y_{i,j}^u - p_{i-1,j,k} u_{c_i,j-1,k} \Delta y_{i-1,j}^u}{\Delta x_{i,j}^u \Delta y_{i,j}^u} + \frac{q_{i-1,j,k} v_{c_i,j-1,k} \Delta y_{i-1,j}^v - q_{i,j,k} v_{c_i,j,k} \Delta y_{i,j}^v}{\Delta x_{i,j}^v \Delta y_{i,j}^v} \right),
\]

(146)

with the layer height changes

\[
h_{n_i,j,k}^{o} = h_{o_i,j,k}^{o} - \Delta t \left( \frac{p_{i,j,k} \Delta y_{i,j}^u - p_{i-1,j,k} \Delta y_{i-1,j}^u}{\Delta x_{i,j}^u \Delta y_{i,j}^u} + \frac{q_{i-1,j,k} \Delta y_{i-1,j}^v - q_{i,j,k} \Delta y_{i,j}^v}{\Delta x_{i,j}^v \Delta y_{i,j}^v} \right).
\]

(147)

Here, \( n \) and \( o \) denote values before and after this operation, respectively, \( n \) denote intermediate values when the 1D advection step comes after this and \( o \) denotes intermediate values when the 1D advection step came before this.

USES:

use domain, only: imin,imax,jmin,jmax,kmax
use advection_3d, only: hi,hio
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

REALTYPE, intent(in) :: uu(I3DFIELD),vv(I3DFIELD)
REALTYPE, intent(in) :: ho(I3DFIELD),hn(I3DFIELD)
REALTYPE, intent(in) :: hun(I3DFIELD),hvn(I3DFIELD)
REALTYPE, intent(in) :: delxv(I2DFIELD),delyu(I2DFIELD)
REALTYPE, intent(in) :: delxu(I2DFIELD),delyv(I2DFIELD)
REALTYPE, intent(in) :: area_inv(I2DFIELD),dt,AH
integer, intent(in) :: az(E2DFIELD)

INPUT/OUTPUT PARAMETERS:

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

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: rc,i,j,k,ii,jj
#ifdef USE_ALLOCATED_ARRAYS
REALTYPE, dimension(:,::,:), allocatable :: flx,fly
REALTYPE, dimension(:,::,:), allocatable :: cmin,cmax
#else
REALTYPE :: flx(I3DFIELD),fly(I3DFIELD)
REALTYPE :: cmin(I3DFIELD),cmax(I3DFIELD)
#endif
8.27.5 \texttt{u_split_adv} - 1D x-advection

\textbf{INTERFACE:}

\begin{verbatim}
subroutine u_split_adv(dt,f,uu,hun, &
delxu,delyu,area_inv,au,splitfac,method,az,AH)
\end{verbatim}

\textbf{DESCRIPTION:}

Here, the $x$-directional split 1D advection step is executed with a number of options for the numerical scheme. The basic advection equation is accompanied by a fractional step for the continuity equation and both equations look as follows:

$$h_{i,j,k}^n c_{i,j,k}^n = h_{i,j,k}^o c_{i,j,k}^o - \Delta t \left[ p_{i,j,k} \epsilon_{i,j,k}^u \Delta y_{i,j}^u - p_{i-1,j,k} \epsilon_{i-1,j,k}^w \Delta y_{i-1,j}^u \right],$$

\hspace{1cm} (148)

with the 1D continuity equation

$$h_{i,j,k}^n = h_{i,j,k}^o - \Delta t \left[ p_{i,j,k} \Delta y_{i,j}^u - p_{i-1,j,k} \Delta y_{i-1,j}^u \right].$$

\hspace{1cm} (149)

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 $x$-directional split step is repeated during the total time step (Strang splitting), the time step $\Delta t$ denotes a fraction of the full time step.

The interfacial fluxes $\tilde{c}_{i,j,k}$ are calculated by means of monotone Total Variation Diminishing (TVD), the first-order monotone upstream and the (non-monotone) unlimited third-order polynomial scheme according to:

$$\tilde{c}_{i,j,k} = \begin{cases} 
  c_{i,j,k} + \frac{1}{2} \tilde{c}_{i,j,k}^+ (1 - |c_{i,j,k}|)(c_{i+1,j,k} - c_{i,j,k}) & \text{for } p_{i,j,k} \geq 0, \\
  c_{i+1,j,k} + \frac{1}{2} \tilde{c}_{i,j,k}^- (1 - |c_{i,j,k}|)(c_{i,j,k} - c_{i+1,j,k}) & \text{else,}
\end{cases}$$

\hspace{1cm} (150)

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

$$\tilde{c}_{i,j,k}^+ = \alpha_{i,j,k} + \beta_{i,j,k} r_{i,j,k}^+, \quad \tilde{c}_{i,j,k}^- = \alpha_{i,j,k} + \beta_{i,j,k} r_{i,j,k}^-,$$

\hspace{1cm} (151)

where

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

\hspace{1cm} (152)

and

$$r_{i,j,k}^+ = \frac{c_{i+1,j,k} - c_{i-1,j,k}}{c_{i+1,j,k} - c_{i,j,k}}, \quad r_{i,j,k}^- = \frac{c_{i+2,j,k} - c_{i+1,j,k}}{c_{i+1,j,k} - c_{i,j,k}}.$$  

\hspace{1cm} (153)

It should be noted that by formulation (150) this so-called $P_2$ scheme is cast into the so-called Lax-Wendroff form, which would be recovered for $\tilde{c}_{i,j,k}^+ = \tilde{c}_{i,j,k}^- = 1$.

In order to obtain a monotonic and positive scheme, the factors $\tilde{c}_{i,j,k}^\pm$ are limited in the following way:

$$\tilde{c}_{i,j,k}^\pm \rightarrow \max \left[ 0, \min \left( \tilde{c}_{i,j,k}^\pm, \frac{2}{1 - |c_{i,j,k}|}, \frac{2r_{i,j,k}^+}{|c_{i,j,k}|} \right) \right],$$

\hspace{1cm} (154)
and, equivalently, for $\tilde{c}_{i,j,k}^-$. 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 (150) have been listed by Zalesak (1987). Among those are the MUSCL scheme by van Leer (1979),

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

(155)

and the Superbee scheme by Roe (1985),

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

(156)

The selector for the schemes is method:

- method = UPSTREAM_SPLIT: first-order upstream (monotone)
- method = P2: third-order polynomial (non-monotone)
- method = P2_PDM: third-order ULTIMATE-QUICKEST (monotone)
- method = MUSCL: second-order TVD (monotone)
- method = Superbee: second-order TVD (monotone)

Furthermore, the horizontal diffusion in y-direction with the constant diffusion coefficient $\Delta H$ is carried out here by means of a central difference second-order scheme.

USES:

- use domain, only: imin,imax,jmin,jmax,kmax
- use advection_3d, only: hi,hio,cu
- use advection_3d, only: UPSTREAM_SPLIT,P2,SUPERBEE,MUSCL,P2_PDM
- use advection_3d, only: one6th

$\$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

- REALTYPE, intent(in) :: uu(I3DFIELD), hun(I3DFIELD)
- REALTYPE, intent(in) :: delxu(I2DFIELD), delyu(I2DFIELD)
- REALTYPE, intent(in) :: area_inv(I2DFIELD), dt
- integer, intent(in) :: au(E2DFIELD), az(E2DFIELD)
- REALTYPE, intent(in) :: splitfac
- integer, intent(in) :: method
- REALTYPE, intent(in) :: AH

INPUT/OUTPUT PARAMETERS:

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

OUTPUT PARAMETERS:

REVISION HISTORY:

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

LOCAL VARIABLES:
8.27.6 \texttt{v_split_adv} - 1D y-advection

**INTERFACE:**

```fortran
subroutine v_split_adv(dt,f,vv,hvn, &
delxv,delyv,area_inv,av,splitfac,method,az,AH)
```

**DESCRIPTION:**

Here, the \(y\)-directional split 1D advection step is executed with a number of options for the numerical scheme. The basic advection equation is accompanied by a fractional step for the continuity equation and both equations look as follows:

\[
h_n^{i,j,k} c_n^{i,j,k} = h_o^{i,j,k} c_o^{i,j,k} - \Delta t \frac{q_{i,j,k} c_{i,j,k}^v \Delta y_{i,j}^v - q_{i,j-1,k} c_{i,j-1,k}^v \Delta y_{i,j-1}^v}{\Delta x_{i,j}^v \Delta y_{i,j}^v},
\]

with the 1D continuity equation

\[
h_n^{i,j,k} = h_o^{i,j,k} - \Delta t \frac{q_{i,j,k} \Delta y_{i,j}^v - q_{i,j-1,k} \Delta y_{i,j-1}^v}{\Delta x_{i,j}^v \Delta y_{i,j}^v}.
\]

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 \(y\)-directional split step is repeated during the total time step (Strang splitting), the time step \(\Delta t\) denotes a fraction of the full time step.

The interfacial fluxes \(c_{i,j,k}^v\) are calculated by means of monotone and non-monotone schemes which are described in detail in \texttt{u_split_adv}, see section 8.27.5 on page 125.

Furthermore, the horizontal diffusion in \(y\)-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,kmax
- use advection_3d, only: hi,hio,cu
- use advection_3d, only: UPSTREAM_SPLIT,P2,SUPERBEE,MUSCL,P2_PDM
- use advection_3d, only: one6th
- $ use omp_lib
- IMPLICIT NONE

**INPUT PARAMETERS:**

- REALTYPE, intent(in) :: vv(I3DFIELD),hvn(I3DFIELD)
- REALTYPE, intent(in) :: delxv(I2DFIELD),delyv(I2DFIELD)
- REALTYPE, intent(in) :: area_inv(I2DFIELD),dt
- integer, intent(in) :: av(E2DFIELD),az(E2DFIELD)
- REALTYPE, intent(in) :: splitfac
- integer, intent(in) :: method
- REALTYPE, intent(in) :: AH

**INPUT/OUTPUT PARAMETERS:**
REALTYPE, intent(inout) :: f(I3DFIELD)

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i,j,k
REALTYPE :: c,x,r,Phi,limit,fu,fc,fd

8.27.7 w_split_adv - 1D z-advection

INTERFACE:

subroutine w_split_adv(dt,f,ww,az,splitfac,method)

DESCRIPTION:

Here, the z-directional split 1D advection step is executed with a number of options for the numerical scheme. The basic advection equation is accompanied by an fractional step for the continuity equation and both equations look as follows:

\[
\begin{align*}
 h_{i,j,k}^n & = h_{i,j,k}^o - \Delta t \left( w_{i,j,k} \tilde{c}_{i,j,k} - w_{i,j,k-1} \tilde{c}_{i,j,k-1} \right), \\
 h_{i,j,k}^o & = h_{i,j,k} - \Delta t \left( w_{i,j,k} \tilde{c}_{i,j,k} - w_{i,j,k-1} \right).
\end{align*}
\]

with the 1D continuity equation

\[
 h_{i,j,k}^o = h_{i,j,k} - \Delta t \left( w_{i,j,k-1} - w_{i,j,k} \right).
\]

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 \( \tilde{c}_{i,j,k} \) are calculated by means of monotone and non-monotone schemes which are described in detail in u_split_adv, see section 8.27.5 on page 125.

USES:

use domain, only: imin,imax,jmin,jmax,kmax
use advection_3d, only: hi,hio,cu
use advection_3d, only: UPSTREAM_SPLIT,P2,SUPERBEE,MUSCL,P2_PDM
use advection_3d, only: ONE6TH
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

REALTYPE , intent(in) :: ww(I3DFIELD),dt
integer , intent(in) :: az(E2DFIELD)
REALTYPE, intent(in) :: splitfac
integer, intent(in) :: method

INPUT/OUTPUT PARAMETERS:

128
REALTYPE, intent(inout) :: f(I3DFIELD)

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i,j,k
REALTYPE :: c,x,r,Phi,limit,fu,fc,fd

8.27.8  w_split_it_adv - iterated 1D z-advection

INTERFACE:

subroutine w_split_it_adv(dt,f,ww,az,splitfac,method)

DESCRIPTION:

Here, the same one-dimensional advection step as in w_split_adv (see section 8.27.7 on page 128) is applied, but with an iteration in time in case that the vertical Courant number exceeds unity at any interface of the water column under calculation.

The number of time steps is calculated as

\[ N_{i,j}^{\Delta t} = \max_k \{ \text{int}(C_{i,j,k} + 1) \}, \]

with the Courant number

\[ C_{i,j,k} = \left| w_{i,j,k} \right| \frac{\Delta t}{\frac{1}{2} \left( h_{i,j,k}^{n} + h_{i,j,k+1}^{n} \right)} \]

and the truncation function \( \text{int} \).

After the number of iterations \( N_{i,j}^{\Delta t} \) is calculated, the vertical advection step is calculated \( N_{i,j}^{\Delta t} \) with a time step of \( \Delta t / N_{i,j}^{\Delta t} \). By doing so, it is avoided that the model blows up due to violation of the CFL criterium, which could happen fast in case of high vertical resolution together with processes such as upwelling or fast sinking material. The good thing about this procedure is that only the water column is punished by higher numerical load in which the potential violation of the CFL criterium occurs.

USES:

use domain, only: imin,imax,jmin,jmax,kmax
use advection_3d, only: hi,hio,cu
use advection_3d, only: UPSTREAM_SPLIT,P2,SUPERBEE,MUSCL,P2_PDM
use advection_3d, only: ONE6TH
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:
REALTYPE, intent(in) :: ww(I3DFIELD), dt
integer, intent(in) :: az(E2DFIELD)
REALTYPE, intent(in) :: splitfac
integer, intent(in) :: method

INPUT/OUTPUT PARAMETERS:

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

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i, ii, j, jj, k, kk, it
REALTYPE :: c, alpha, beta, x, r, Phi, limit, fu, fc, fd, cmax
logical :: READY

8.27.9 fct_2dh_adv - 2D flux-corrected transport

INTERFACE:

subroutine fct_2dh_adv(dt, f, uu, vv, ho, hn, hun, hvn, &
delxv, delyu, delxu, delyv, area_inv, az, AH)

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 the routine upstream_2dh_adv (see section 8.27.4 on page 123).

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 direction-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, kmax
use advection_3d, only: hi, hio
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

130
REALTYPE, intent(in) :: uu(I3DFIELD), vv(I3DFIELD)
REALTYPE, intent(in) :: ho(I3DFIELD), hn(I3DFIELD)
REALTYPE, intent(in) :: hun(I3DFIELD), hvn(I3DFIELD)
REALTYPE, intent(in) :: delxv(I2DFIELD), delyu(I2DFIELD)
REALTYPE, intent(in) :: delxu(I2DFIELD), delyv(I2DFIELD)
REALTYPE, intent(in) :: area_inv(I2DFIELD), dt, AH
integer, intent(in) :: az(E2DFIELD)

INPUT/OUTPUT PARAMETERS:

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

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: rc, i, ii, j, jj, k, kk
#ifdef USE_ALLOCATED_ARRAYS
REALTYPE, dimension(:, :, :), allocatable :: flx, fly
REALTYPE, dimension(:, :, :), allocatable :: fhx, fhy
REALTYPE, dimension(:, :, :), allocatable :: fi
REALTYPE, dimension(:, :, :), allocatable :: rp, rm
REALTYPE, dimension(:, :, :), allocatable :: cmin, cmax
#else
REALTYPE :: flx(I3DFIELD), fly(I3DFIELD)
REALTYPE :: fhx(I3DFIELD), fhy(I3DFIELD)
REALTYPE :: fi(I3DFIELD)
REALTYPE :: rp(I3DFIELD), rm(I3DFIELD)
REALTYPE :: cmin(I3DFIELD), cmax(I3DFIELD)
#endif
REALTYPE :: CNW, CW, CSW, CSSW, CWW, CSWW, CC, CS
REALTYPE :: uuu, vvv, x, CExx, C1, Cu, fac
8.28 Module 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
    use variables_3d, only: T,rad,hn,adv_schemes,kmin,A,g1,g2
    use halo_zones, only: update_3d_halo,wait_halo,D_TAG
    IMPLICIT NONE
    private

PUBLIC DATA MEMBERS:

    public init_temperature, do_temperature

PRIVATE DATA MEMBERS:

    integer :: temp_method=1,temp_format=2
    character(len=PATH_MAX) :: temp_file="t_and_s.nc"
    character(len=32) :: temp_name='temp'
    REALTYPE :: temp_const=20.
    integer :: temp_hor_adv=1,temp_ver_adv=1
    integer :: temp_adv_split=0
    REALTYPE :: temp_AH=-1.
    integer :: attenuation_method=0,jerlov=1
    character(len=PATH_MAX) :: attenuation_file="attenuation.nc"
    REALTYPE :: A_const=0.58,g1_const=0.35,g2_const=23.0
    integer :: temp_check=0
    REALTYPE :: min_temp=-2.,max_temp=35.

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:

8.28.1 init_temperature - initialisation of temperature

INTERFACE:

    subroutine init_temperature(adv_method)
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:

IMPLICIT NONE

INPUT PARAMETERS:

integer, intent(in) :: adv_method

OUTPUT PARAMETERS:

LOCAL VARIABLES:

integer :: k,i,j,n
integer, parameter :: nmax=10000
REALTYPE :: zlev(nmax),prof(nmax)
integer :: temp_field_no=1
integer :: status

namelist /temp/ &
temp_method,temp_const,temp_file, &
temp_format,temp_name,temp_field_no, &
temp_hor_adv,temp_ver_adv,temp_adv_split,temp_AH, &
attenuation_method,attenuation_file,jerlov, &
A_const,g1_const,g2_const, &
temp_check,min_temp,max_temp

8.28.2 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 (101) on page 77). 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 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,au,av
    use meteo, only: swr,shf
    use parameters, only: rho_0,cp

#if defined(SPHERICAL) || defined(CURVILINEAR)
    use domain, only: dxu,dxv,dyu,dyv,ard1
#else
    use domain, only: dx,dy,ard1
#endif

    use parameters, only: avmolt
    use getm_timers, only: tic, toc, TIM_TEMP

$ use omp_lib

IMPLICIT NONE

INPUT PARAMETERS:

    integer, intent(in) :: n

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

    integer :: i,j,k,rc
    REALTYPE :: delxu(I2DFIELD),delxv(I2DFIELD)
    REALTYPE :: delyu(I2DFIELD),delyv(I2DFIELD)
    REALTYPE :: area_inv(I2DFIELD)

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 :: rho_0_cpi
    integer :: status
8.29 Module 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
    #ifdef HAIDVOGEL_TEST
    use domain, only: iextr,jextr
    #endif
    use domain, only: H,az
    use variables_2d, only: fwf_int
    use variables_3d, only: S,hn,adv_schemes,kmin
    use halo_zones, only: update_3d_halo,wait_halo,D_TAG
    IMPLICIT NONE
    private

PUBLIC DATA MEMBERS:

    public init_salinity, do_salinity

!PRIVATE DATA MEMBERS:

    integer :: salt_method=1,salt_format=2
    character(len=PATH_MAX) :: salt_file="t_and_s.nc"
    character(len=32) :: salt_name='salt'
    REALTYPE :: salt_const=35.
    integer :: salt_hor_adv=1,salt_ver_adv=1
    integer :: salt_adv_split=0
    REALTYPE :: salt_AH=-_ONE_
    integer :: salt_check=0
    REALTYPE :: min_salt=0.,max_salt=40.

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:

8.29.1 init_salinity - initialisation of salinity

INTERFACE:

    subroutine init_salinity(adv_method)
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:

IMPLICIT NONE

INPUT PARAMETERS:

```fortran
integer, intent(in) :: adv_method
```

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

```fortran
integer :: i,j,k,n
#ifdef PECS_TEST
integer :: cc(1:30)
#endif
integer, parameter :: nmax=100
REALTYPE :: zlev(nmax),prof(nmax)
integer :: salt_field_no=1
integer :: status
NAMELIST /salt/ &
  salt_method,salt_const,salt_file, &
  salt_format,salt_name,salt_field_no, &
  salt_hor_adv,salt_ver_adv,salt_adv_split,salt_AH, &
  salt_check,min_salt,max_salt
```

8.29.2 do_salinity - salinity equation

INTERFACE:

```fortran
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:

```fortran
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,au,av
#endif
use domain, only: dxu,dxv,dyu,dyv,arcd1
#endif
use domain, only: dx,dy,ard1
#endif
use parameters, only: avmols
use getm_timers, only: tic, toc, TIM_SALT
$ use omp_lib
IMPLICIT NONE
```

INPUT PARAMETERS:

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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

```fortran
integer :: i,j,k,rc
REALTYPE, POINTER :: Res(:)
REALTYPE, POINTER :: auxn(:,),auxo(:)
REALTYPE, POINTER :: a1(:,),a2(:,),a3(:,),a4(:)
#endif
REALTYPE :: SRelax,kk
#endif
REALTYPE :: delxu(I2DFIELD),delxv(I2DFIELD)
REALTYPE :: delyu(I2DFIELD),delyv(I2DFIELD)
REALTYPE :: area_inv(I2DFIELD)
integer :: status
```
8.30 Module 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:

$$-w_s C - \partial_z (\nu/\partial_z C) = F_e - F_s,$$

where erosion and sedimentation fluxes are modelled following Krone (1962) as functions of the bottom shear stress $\tau_b$. In (163), $w_s$ 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 $\tau_{ce}$:

$$F_e = \begin{cases} 
    \max \left\{ \frac{c_e}{\rho_0} (|\tau_b| - \tau_{ce}), 0 \right\}, & \text{for } B > 0 \text{ and } |\tau_b| > \tau_{ce} \\
    0, & \text{else} 
\end{cases}$$

(164)

with $c_e$, erosion constant with units 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 $\tau_{cs}$. This flux is limited by the near bottom concentration $C_b$:

$$F_s = \max \left\{ \frac{w_s C_b}{\tau_{cs}} (\tau_{cs} - |\tau_b|), 0 \right\}.$$

(165)

Critical shear stresses for erosion and sedimentation ($\tau_{ce}$ and $\tau_{cs}$ have as units 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 $B = B_0$ kg m$^{-2}$ is initially assumed. Sedimentation and erosion fill and empty this pool, respectively:

$$\partial_t (B) = F_s - F_e$$

(166)

and the erosion flux is constricted by the availability of SPM from the pool (see eq. (164)). 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 $\alpha$, (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:

$$\rho = \rho_{T,S,p} + \left( 1 - \frac{\rho_{T,S,p}}{\rho_{spm}} \right) C.$$

(167)

USES:
use exceptions
use domain, only: i min, j min, i max, j max, k max, i o ff, j o ff
#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, adv_schemes, spm, spm_ws, spm_pool
use halo_zones, only: update_3d_halo, wait_halo, D_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_hor_adv=1, spm_ver_adv=1, spm_adv_split=0
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

8.30.1 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 prescribed or calculated by means of the Zanke (1977) formula.

USES:

For initialization of spm in intertidal flats
use domain, only: min_depth

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_hor_adv, spm_ver_adv, spm_adv_split, &
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

8.30.2 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. (167).

Finally, some special settings for single test cases are made via compiler options.

USES:

```fortran
use advection_3d, only: do_advection_3d
use variables_3d, only: dt,cnpar,hun,hvn,ho,nuh,uu,vv,ww
#endif NO_BAROCLINIC
use variables_3d, only: rho
#endif
use domain, only: au,av
#ifdef SPHERICAL || defined(CURVILINEAR)
use domain, only: dxu,dxv,duy,dvy,arcd1
#endif
use domain, only :dxc
#else
use domain, only: dx,dy,ard1
#endif
use domain, only: dry_z
IMPLICIT NONE

LOCAL VARIABLES:

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>i, j, k, rc</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>spmtot</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>Res(0:kmax)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>auxn(1:kmax-1), auxo(1:kmax-1)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>a1(0:kmax), a2(0:kmax)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>a3(0:kmax), a4(0:kmax)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>delxu(I2DFIELD), delxv(I2DFIELD)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>delyu(I2DFIELD), delvy(I2DFIELD)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>area_inv(I2DFIELD)</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>bed_flux</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>c</td>
</tr>
<tr>
<td>REALTYPE</td>
<td>volCmud, volCpart</td>
</tr>
<tr>
<td>integer</td>
<td>k2</td>
</tr>
<tr>
<td>logical</td>
<td>patankar=.true.</td>
</tr>
</tbody>
</table>
#ifdef TRACER_POSITIVE
| logical   | kk |
#endif

REALTYPE, allocatable, dimension (:,:,::) :: ww_aux
8.31 Module 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 parameters, only: g,rho_0
   IMPLICIT NONE

PUBLIC DATA MEMBERS:
   public init_eqstate, do_eqstate
   !PRIVATE DATA MEMBERS:
   integer :: eqstate_method=1
   REALTYPE :: T0 = 10., S0 = 33.75, p0 = 0.
   REALTYPE :: dtr0 = -0.17, dsr0 = 0.78

REVISION HISTORY:
   Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:

8.31.1 init_eqstate

INTERFACE:
   subroutine init_eqstate()
   IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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

142
8.31.2  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: T,S,rho, buoy
    use getm_timers, only: tic, toc, TIM_EQSTATE

    $ use omp_lib
      implicit none

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

    integer :: i,j,k
    realtype :: x
    realtype :: KK
    realtype :: T1,T2,T3,T4,T5,S1,S15,S2,S3,p2
8.32  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 straightforward methods which are explained first, do both have the disadvantage of generating numerical instabilities. The straightforward way for calculating \( M^2 \) is as follows:

\[
(M^2)_{i,j,k} \approx \frac{1}{2} \left( \frac{u_{i,j,k+1} - u_{i,j,k}}{h_{i,j,k+1}^u + h_{i,j,k}^u} \right)^2 + \frac{1}{2} \left( \frac{u_{i-1,j,k+1} - u_{i-1,j,k}}{h_{i-1,j,k+1}^u + h_{i-1,j,k}^u} \right)^2 + \frac{1}{2} \left( \frac{v_{i,j,k+1} - v_{i,j,k}}{h_{i,j,k+1}^v + h_{i,j,k}^v} \right)^2 + \frac{1}{2} \left( \frac{v_{i,j-1,k+1} - v_{i,j-1,k}}{h_{i,j-1,k+1}^v + h_{i,j-1,k}^v} \right)^2 \]

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 (168):

\[
(M^2)_{i,j,k} \approx \frac{1}{2} \left( \frac{\nu_{i,j,k+1} + \nu_{i+1,j,k}}{h_{i,j,k+1}^\nu + h_{i,j,k}^\nu} \right)^2 \left( \frac{u_{i,j,k+1} - u_{i,j,k}}{h_{i,j,k+1}^u + h_{i,j,k}^u} \right)^2 + \frac{1}{2} \left( \frac{\nu_{i-1,j,k+1} + \nu_{i-1,j,k}}{h_{i-1,j,k+1}^\nu + h_{i-1,j,k}^\nu} \right)^2 \left( \frac{u_{i-1,j,k+1} - u_{i-1,j,k}}{h_{i-1,j,k+1}^u + h_{i-1,j,k}^u} \right)^2 + \frac{1}{2} \left( \frac{\nu_{i,j,k+1} + \nu_{i,j+1,k}}{h_{i,j,k+1}^\nu + h_{i,j,k}^\nu} \right)^2 \left( \frac{v_{i,j,k+1} - v_{i,j,k}}{h_{i,j,k+1}^v + h_{i,j,k}^v} \right)^2 + \frac{1}{2} \left( \frac{\nu_{i,j-1,k+1} + \nu_{i,j-1,k}}{h_{i,j-1,k+1}^\nu + h_{i,j-1,k}^\nu} \right)^2 \left( \frac{v_{i,j-1,k+1} - v_{i,j-1,k}}{h_{i,j-1,k+1}^v + h_{i,j-1,k}^v} \right)^2 \right) \cdot \left( \frac{1}{2} \left( \frac{h_{i,j,k}^\nu + h_{i,j,k+1}^\nu}{\nu_{i,j,k}} \right)^{-1} \right)
\]

The straightforward discretisation of \( N^2 \) is given by

\[
(N^2)_{i,j,k} \approx \frac{b_{i,j,k+1} - b_{i,j,k}}{2(h_{i,j,k+1}^\nu + h_{i,j,k}^\nu)}.
\]

In some cases, together with the straightforward discretisation of the shear squared, (168), 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 straightforward method (170) leads to an inconsistency. However, other experiments with the energy-conserving discretisation of the shear stress squared, (169) and the straightforward discretisation of \( N^2 \), (170), produced numerically stable results. Most stable results have been obtained with a weighted average for the \( N^2 \) calculation:
\[(N^2)_{i,j,k} \approx \frac{1}{6} \left( 2 \frac{b_{i,j,k+1} - b_{i,j,k}}{h_{i,j,k+1}^t + h_{i,j,k}^t} + \frac{b_{i+1,j,k+1} - b_{i+1,j,k}}{h_{i+1,j,k+1}^t + h_{i+1,j,k}^t} + \frac{b_{i-1,j,k+1} - b_{i-1,j,k}}{h_{i-1,j,k+1}^t + h_{i-1,j,k}^t} \right) \]
\[
+ \left( 2 \frac{b_{i,j+1,k+1} - b_{i,j+1,k}}{h_{i,j+1,k+1}^t + h_{i,j+1,k}^t} + \frac{b_{i,j-1,k+1} - b_{i,j-1,k}}{h_{i,j-1,k+1}^t + h_{i,j-1,k}^t} \right) \]
\[
+ \left( 2 \frac{b_{i+1,j,k+1} - b_{i+1,j,k}}{h_{i+1,j,k+1}^t + h_{i+1,j,k}^t} + \frac{b_{i+1,j,k+1} - b_{i+1,j,k}}{h_{i+1,j,k+1}^t + h_{i+1,j,k}^t} \right) \]
\[
+ \left( 2 \frac{b_{i,j+1,k+1} - b_{i,j+1,k}}{h_{i,j+1,k+1}^t + h_{i,j+1,k}^t} + \frac{b_{i,j+1,k+1} - b_{i,j+1,k}}{h_{i,j+1,k+1}^t + h_{i,j+1,k}^t} \right). \]

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
ifndef NO_BAROCLINIC
use variables_3d, only: NN, buoy
endif
use getm_timers, only: tic, toc, TIM_SSNN
$ use omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i,j,k,nb
REALTYPE :: dz,NNc,NNe,NNw,NNn,NNs
8.33 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 8.34, the normalised
surface and bottom stresses, $\tau_s/\rho_0$ (variable taus) and $\tau_b/\rho_0$ (variable taub), respectively, are
calculated and interpolated to the T-points. Input parameters to this routine are rru and ttrv, 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 stress 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, taub, 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
    IMPPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

    integer :: i, j, k, ku1, ku2, kv1, kv2
    REALTYPE :: rho_0i
8.34 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 $h_n \rightarrow h$, the shear squared $SS \rightarrow SS_{1d}$, the buoyancy frequency squared $NN \rightarrow NN_{1d}$, the turbulent kinetic energy $tke \rightarrow tke_{1d}$, the dissipation rate $\varepsilon \rightarrow \varepsilon_{1d}$ (from which the integral length scale $L_{1d}$ is calculated), the eddy viscosity $num \rightarrow num_{1d}$, and the eddy diffusivity $nuh \rightarrow nuh_{1d}$. The scalars are the surface and bottom friction velocities, $u_{taus}$ and $u_{taub}$, respectively, the surface roughness parameter $z_0s$ (which is currently hard-coded), and the bottom roughness parameter $z_0b$. Then, the GOTM turbulence module $do_turbulence$ is called with all the transformed parameters discussed above. Finally, the vertical vectors $tke_{1d}$, $\varepsilon_{1d}$, $num_{1d}$ and $nuh_{1d}$ 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

$$P_{tot} = P + C \left( u^2 + v^2 \right)^{3/2},$$

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 calculations, 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

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:
OUTPUT PARAMETERS:

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)
8.35 Module 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,arc1
    #else
       use domain, only: H,az,kmax,ard1
    #endif
    use m2d, only: dtm
    #ifdef NO_BAROCLINIC
       use m3d, only: calc_salt,calc_temp
    #else
       use variables_3d, only: hn,ssen,T,S
    #endif
    #ifdef GETM_BIO
       use bio, only: bio_calc
       use bio_var, only: numc
    #endif
    PRIVATE

PUBLIC DATA MEMBERS:

    public init_rivers, do_rivers, clean_rivers
    #ifdef GETM_BIO
       public init_rivers_bio
    #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
!PRIVATE DATA MEMBERS:
integer :: river_format = 2
character(len=64) :: river_info = "riverinfo.dat"
integer, allocatable :: ir(:), jr(:)
REALTYPE, allocatable :: irr(:)
REALTYPE, allocatable :: macro_height(:)
REALTYPE, allocatable :: flow_fraction(:)

REVISION HISTORY:
Original author(s): Karsten Bolding & Hans Burchard

LOCAL VARIABLES:

8.35.1 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 nriver 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

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:
8.35.2  

7.2.1 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

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

   integer :: rc

8.35.3  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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

  integer :: i,j,k,m,n
  integer, save :: nn=0
  REALTYPE :: ramp=_ONE_
  REALTYPE :: rvol,height
  REALTYPE :: svol,tvol,vol

8.35.4 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

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

  integer :: i,j,n
  REALTYPE :: tot=_ZERO_
8.36 Module 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
IMPLICIT NONE

PUBLIC DATA MEMBERS:

public init_bdy_3d, do_bdy_3d
REALTYPE, public, allocatable : S_bdy(:,,:), T_bdy(:,:)
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

LOCAL VARIABLES:

8.36.1 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

INPUT PARAMETERS:
INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

\[
\text{integer} \quad :: \text{rc, i, j, k, n}
\]

8.36.2 do_bdy_3d - updating 3D boundary conditions

INTERFACE:

\[
\text{subroutine do_bdy_3d(tag, field)}
\]

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 \textit{Martinsen and Engedahl} (1987).

As an extension 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:

\[
\text{IMPLICIT NONE}
\]

INPUT PARAMETERS:

\[
\text{integer, intent(in)} \quad :: \text{tag}
\]

INPUT/OUTPUT PARAMETERS:

\[
\text{REALTYPE, intent(inout)} \quad :: \text{field}(\text{I3DFIELD})
\]

OUTPUT PARAMETERS:

LOCAL VARIABLES:

\[
\begin{align*}
\text{integer} & \quad :: \text{i, j, k, l, n, ii, jj, kk} \\
\text{REALTYPE} & \quad :: \text{sp(1:4), rat} \\
\text{REALTYPE} & \quad :: \text{bdy3d_tmrlx_umin} \\
\text{REALTYPE} & \quad :: \text{wsum}
\end{align*}
\]

154
8.37 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 \( U_i \) and \( V_i \), respectively. The drag coefficient \( R \) is given by eq. (70) on page 49. The results for the variables ruu and rvv will then be used in the routine slow_terms described on page 158 for the calculation of the slow terms \( S_x^F \) and \( S_y^F \), 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 omp_lib
IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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)
8.38 slow_advection - slow advection terms (Source File: slow_advection.F90)

INTERFACE:

    subroutine slow_advection

DESCRIPTION:

Here, the calculation of the advective slow terms \( S_A^x \) and \( S_A^y \) (see eqs. (64) and (65)) is prepared. This routine basically repeats the calculations made in the routine `uv_advect`, see section 7.3.6, but this time based on the macro time step averaged and vertically integrated transports \( \text{Uint} \) and \( \text{Vint} \). The calculations of \( S_A^x \) and \( S_A^y \) are then completed in the routine `slow_terms`, see section 8.40 on page 158.

USES:

    use domain, only: imin,imax,jmin,jmax,HU,HV,az,au,av,ax
    use domain, only: H,min_depth
    #if defined(SPHERICAL) || defined(CURVILINEAR)
    use domain, only: dyc,arud1,dxx,dyx,arvd1,dxc
    #else
    use domain, only: dx,dy,ard1
    #endif
    use variables_2d, only: UEx,VEx,Uint,Vint,PP
    use variables_3d, only: ssun,ssvn
    use getm_timers, only: tic, toc, TIM_SLOWADV
    $ use omp_lib
    IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

    integer :: i,j,ii,jj
    REALTYPE :: DU1(I2DFIELD)
    REALTYPE :: DV1(I2DFIELD)
8.39  slow_diffusion - slow diffusion terms (Source File: slow_diffusion.F90)

INTERFACE:

   subroutine slow_diffusion(AM)

DESCRIPTION:

Here, the calculation of the diffusive slow terms $S^x_D$ and $S^y_D$ (see eqs. (66) and (66)) is prepared. This routine basically repeats the calculations made in the routine uv_iondvect, see section 7.3.6, but this time based on the macro time step averaged and vertically integrated transports $U_{int}$ and $V_{int}$. However, the damping of the external mode, as described in (89) and (90) is not considered here. The calculations of $S^x_D$ and $S^y_D$ are then completed in the routine slow_terms, see section 8.40 on page 158.

USES:

   use domain, only: imin,imax,jmin,jmax,az,au,av,ax,H,HU,HV
   #if defined(SPHERICAL) || defined(CURVILINEAR)
     use domain, only: dyc,arud1,dxx,dyx,arvd1,dxc
   #else
     use domain, only: dx,dy,ard1
   #endif
   use variables_2d, only: D,U,V,UEx,VEx,Uint,Vint,PP
   use variables_3d, only: ssen,ssun,ssvn
   use getm_timers, only: tic, toc, TIM_SLOWDIFF
   IMPLICIT NONE

INPUT PARAMETERS:

   REALTYPE, intent(in) :: AM

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

LOCAL VARIABLES:

   integer :: i,j,ii,jj
   REALTYPE :: D(iimin-1:imax+1,jimin-1:jmax+1)
   REALTYPE :: DU(iimin-1:imax+1,jimin-1:jmax+1)
   REALTYPE :: DV(iimin-1:imax+1,jimin-1:jmax+1)
8.40  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 (62) - (69), see section 7.1. These calculations have been prepared in the
routines slow_bottom_friction, slow_advection and slow_diffusion.

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,ruu,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

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

    integer         :: i,j,k
    REALTYPE        :: vertsum
8.41 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 $\text{sse}_0$, $\text{ss}_e$, $\text{ss}_u$, $\text{ss}_n$, $\text{ss}_v$ and $\text{ss}_n$, 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 83. Furthermore, the vertically integrated transports $\text{Uint}$ and $\text{Vint}$ 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
use getm_timers, only: tic, toc, TIM_STARTMCR
IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

integer :: i,j
REALTYPE :: split
8.42 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 iterated and temporally averaged transports to old values Uint and Vinto, and to reinitialize the transports Uint and Vint to zero. This is also the place where the surface velocity divergence is calculated, i.e. $\text{div} = \partial_x u(z = \eta) + \partial_y v(z = \eta)$, to be output in the 2d netcdf file.

USES:

    use domain, only: imin,imax,jmin,jmax,kmax
    use variables_2d, only: Uint,Uinto,Vint,Vinto,surfdiv
    use variables_3d, only: hun,hvn,uu,vv
    use getm_timers, only: tic, toc, TIM_STOPMCR
    #if defined(SPHERICAL) || defined(CURVILINEAR)
        use domain, only: arcd1,dxv,dyu
    #else
        use domain, only: dx,dy,ard1
    #endif

    IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Hans Burchard & Karsten Bolding

LOCAL VARIABLES:

8.42.1 divergence

INTERFACE:

    subroutine divergence()

DESCRIPTION:

USES:
use domain, only: imin,imax,jmin,jmax,kmax
#if defined(SPHHERICAL) || defined(CURVILINEAR)
   use domain, only: arcd1,dxv,dyu
#else
   use domain, only: ard1,dx,dy
#endif
use variables_2d, only: surfdiv
use variables_3d, only: hun,hvn,uu,vv
IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Hans Burchard & Karsten Bolding
$Log: divergence.F90,v $
Revision 1.1  2010-03-24 14:58:15  kb
cleaned divergence calculations

LOCAL VARIABLES:

   integer :: i,j
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  Module 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
   $Log: ncdf_common.F90,v $
   Revision 1.4  2009-03-13 14:44:14  kb
   grid information in NF_DOUBLE
   Revision 1.3  2005-04-25 09:32:34  kbb
   added NetCDF IO rewrite + de-stag of velocities - Umlauf
   Revision 1.2  2003/04/23 11:54:03  kbb
   cleaned code + TABS to spaces
   Revision 1.1.1  2002/05/02 14:01:49  gotm
   recovering after CVS crash
   Revision 1.1  2001/09/13 14:50:02  bbh
   Cleaner and smaller NetCDF implementation + better axis support
9.2 Module 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
$Log: grid_ncdf.F90,v $
Revision 1.3 2009-09-23 10:11:48 kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.2 2006-09-26 07:06:06 kbk
to compile on Macs with Intel compiler
Revision 1.1 2005-04-25 09:32:34 kbk
added NetCDF IO rewrite + de-stag of velocities - Umlauf
9.3 Module 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
integer :: res_u_id,res_v_id,surfdiv_id
integer :: u10_id,v10_id
integer :: airp_id,t2_id,hum_id,tcc_id
integer :: tauxx_id,tausy_id,swr_id,shf_id
integer :: evap_id=-1,precip_id=-1
integer :: break_stat_id=-1

REAL_4B, dimension(:,), allocatable :: ws

!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 :: swr_missing =-9999.0
REALTYPE, parameter :: shf_missing =-9999.0
REALTYPE, parameter :: divergence_missing =-9999.0
REALTYPE, parameter :: evap_missing =-9999.0
REALTYPE, parameter :: precip_missing =-9999.0

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: ncdf_2d.F90,v $
Revision 1.7 2008-09-16 11:21:50 kb
if -DUSE_BREAKS save break statistics
Revision 1.6 2007-06-27 08:39:37 kbk
support for fresh water fluxes at the sea surface - Adolf Stips
Revision 1.5 2005-04-25 09:32:34 kbb
added NetCDF IO rewrite + de-stag of velocities - Umlauf

166
Revision 1.4  2003/06/17 14:53:29  kkb
default meteo variables names comply with Adolf Stips suggestion + southpole(3)
Revision 1.3  2003/05/09 11:38:26  kkb
added proper undef support - based on Adolf Stips patch
Revision 1.2  2003/04/23 11:53:24  kkb
save lat/lon info for spherical grid
Revision 1.1.1.1  2002/05/02 14:01:49  gotm
recovering after CVS crash
Revision 1.3  2001/10/26 12:18:06  bbh
No actual storing of data in init_2d_ncdf.F90 -> save_2d_ncdf.F90
Revision 1.2  2001/09/27 08:35:10  bbh
Saving meteo again - in .2d.nc file
Revision 1.1  2001/09/13 14:50:02  bbh
Cleaner and smaller NetCDF implementation + better axis support
9.4 Module ncdf_2d_bdy - input in NetCDF format (Source File: ncdf_2d_bdy.F90)

INTERFACE:

    module ncdf_2d_bdy

DESCRIPTION:

USES:

    KB  use m2d, only: dtm,bdy_times,bdy_old,bdy_new,bdy_data
        use m2d, only: dtm,bdy_times,bdy_data,bdy_data_u,bdy_data_v
        use time, only: string_to_julsecs,time_diff,julianday,secondsofday
        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
    $Log: ncdf_2d_bdy.F90,v $
    Revision 1.8 2008-12-12 06:06:50 kb
    fixed serious error in specified u boundary velocity
    Revision 1.7 2008-12-09 00:31:58 kb
    added new 2D open boundaries
    Revision 1.6 2007-09-30 13:00:43 kbb
    prints real time as part of progress output
    Revision 1.5 2005-05-04 11:45:29 kbb
    adding model time stamp on IO
    Revision 1.4 2004/04/06 16:32:29 kbb
    TimeDiff --> time_diff
    Revision 1.3 2003/04/23 11:54:03 kbb
    cleaned code + TABS to spaces
    Revision 1.2 2003/04/07 12:49:47 kbb
    dont need variables_3d
    Revision 1.1.1.1 2002/05/02 14:01:46 gotm
    recovering after CVS crash
    Revision 1.8 2001/10/22 11:43:12 bbb
    Proper check of offset time
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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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
   integer  :: j1,s1
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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: ncdf_2d_bdy.F90,v $
Revision 1.8 2008-12-12 06:06:50 kb
fixed serious error in specified u boundary velocity
Revision 1.7 2008-12-09 00:31:58 kb
added new 2D open boundaries
Revision 1.6 2007-09-30 13:00:43 kbb
prints real time as part of progress output
Revision 1.5 2005-05-04 11:45:29 kbb
adding model time stamp on IO
Revision 1.4 2004/04/06 16:32:29 kbb
TimeDiff --> time_diff
Revision 1.3 2003/04/23 11:54:03 kbb
cleaned code + TABS to spaces
Revision 1.2 2003/04/07 12:49:47 kbb
dont need variables_3d
Revision 1.1.1.1 2002/05/02 14:01:46 gotm
recovering after CVS crash
Revision 1.8 2001/10/22 11:43:12 bbh
Proper check of offset time
Revision 1.7 2001/10/17 14:57:38 bbh
Force offset to _ZERO_ - needs fix
Revision 1.6 2001/09/19 14:21:13 bbh
Cleaning
Revision 1.5 2001/07/27 06:41:35 bbh
Added ncdf_lon_lat.F90
Revision 1.4 2001/07/26 14:07:18 bbh
Typos
Revision 1.3 2001/06/22 08:19:10 bbh
Compiler options such as USE_MASK and OLD_DRY deleted.
Open and passive boundary for z created.
Various inconsistencies removed.
wait_halo added.
Checked loop boundaries
Revision 1.2 2001/05/18 13:04:39 bbh
Cosmetics
Revision 1.1 2001/05/14 12:45:56 bbh
Introduced module ncdf_2d_bdy

LOCAL VARIABLES:

integer,save :: i,n
integer :: err
logical :: first=.true.
REALTYPE :: t
REALTYPE, save :: t1,t2= -_ONE_,loop0
9.5 Module Encapsulate 3D netCDF quantities (Source File: ncd_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
integer :: salt_id,temp_id,sigma_t_id
integer :: rad_id
integer :: tke_id,num_id,nuh_id,eps_id
integer :: SS_id,NN_id
#else define SPM
integer :: spmpool_id,spm_id
#endif
#endif define GETM_BIO
integer, allocatable :: bio_ids(:)
#endif define SPM

REAL_4B, dimension(:), allocatable :: ws

!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
#endif define SPM
REALTYPE, parameter :: spmpool_missing=-9999.0
REALTYPE, parameter :: spm_missing =-9999.0

#ifdef GETM_BIO
  REALTYPE, parameter :: bio_missing=-9999.0
#endif

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: ncdf_3d.F90,v $
Revision 1.10 2009-04-22 10:09:36 lars
  support for bottom stress output
Revision 1.9  2009-01-05 09:57:06 kb
  option for storing SS and NN
Revision 1.8  2007-02-20 13:52:15 kbk
  solar radiation -> 3d field - possible to save
Revision 1.7  2005/04/25 09:32:34 kbk
  added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.6  2004/06/15 08:25:57 kbk
  added support for spm - Ruiz
Revision 1.5  2004/05/04 09:23:51 kbk
  hydrostatic consistency criteria stored in .3d.nc file
Revision 1.4  2003/12/16 12:51:04 kbk
  preparing for proper support for SPM (manuel)
Revision 1.3  2003/05/09 11:38:26 kbk
  added proper undef support - based on Adolf Stips patch
Revision 1.2  2003/04/23 11:53:24 kbk
  save lat/lon info for spherical grid
Revision 1.1.1.1 2002/05/02 14:01:49 gotm
  recovering after CVS crash
Revision 1.3  2001/10/23 14:19:20 bbh
  Stores h if general vertical coordinates
Revision 1.2  2001/10/23 07:37:17 bbh
  Saving spm - if calc_spm and save_spm are both true
Revision 1.1  2001/09/13 14:50:02 bbh
  Cleaner and smaller NetCDF implementation + better axis support
9.6 Module ncdf_3d_bdy - input in NetCDF format (Source File: ncdf_3d_bdy.F90)

INTERFACE:

module ncdf_3d_bdy

DESCRIPTION:

USES:

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,julianday,secondsofday
use time, only: write_time_string,timestr

IMPLICIT NONE

private

public :: init_3d_bdy_ncdf,do_3d_bdy_ncdf

!PRIVATE DATA MEMBERS:

type(ncdf_type)

integer :: ncid
integer :: time_id,temp_id,salt_id
integer :: start(4),edges(4)
integer :: zax_dim,zax_len
integer :: time_dim,time_len
logical :: climatology=.false.
logical :: from_3d_fields=.false.
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

REVISED HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: ncdf_3d_bdy.F90,v $
Revision 1.18 2009-09-30 11:28:48 bjb
OpenMP threading initial implementation
Revision 1.17 2009-08-13 08:24:51 kb
added a few checks on z-xais in the 3D boundary file - suggested by Buchmann
Revision 1.16 2009-07-30 15:30:07 kb
fixed j-index for +1 eastern boundaries - Hofmeister
Revision 1.15 2007-10-10 10:25:20 kbb
oops
Revision 1.14 2007-10-10 10:01:19 kbb
fixed interpolation when model depth > boundary data depth
Revision 1.13 2007-09-30 13:00:43 kbb

174
prints real time as part of progress output
Revision 1.12 2007-05-11 07:51:27 frv-bjb
Free dimid numbering for 3d bdy files (order in var still fixed)
Revision 1.11 2007-05-08 09:08:18 kkb
rudimentary check for valid lower boundary data
Revision 1.10 2005-05-04 11:50:57 kkb
support for non-climatological 3D boundaries (S,T)
Revision 1.9 2004/04/06 16:32:29 kkb
TimeDiff --> time_diff
Revision 1.8 2003/12/16 16:50:41 kkb
added support for Intel/IFORT compiler - expanded TABS, same types in subroutine calls
Revision 1.7 2003/10/07 15:10:42 kkb
use zax_dim as argument to dim_len
Revision 1.6 2003/08/03 09:19:41 kkb
optimised reading of climatological boundary data
Revision 1.5 2003/05/05 15:44:20 kkb
reads boundary values from 3D fields as individual columns
Revision 1.4 2003/04/23 11:54:03 kkb
cleaned code + TABS to spaces
Revision 1.3 2003/04/07 16:19:52 kkb
parallel support
Revision 1.1.1.1 2002/05/02 14:01:49 gotm
recovering after CVS crash
Revision 1.1 2001/10/17 13:28:27 bbh
Initial import
9.7  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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard
    See log for module

LOCAL VARIABLES:

    character(len=256) :: units
    integer :: j1, s1
    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
9.8 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,secspday
    IMPLICIT NONE

INPUT PARAMETERS:

    integer, intent(in) :: loop

OUTPUT/INPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard
$Log: ncdf_3d_bdy.F90,v $
    Revision 1.18  2009-09-30 11:28:48  bjb
    OpenMP threading initial implementation
Revision 1.17  2009-08-13 08:24:51  kb
    added a few checks on z-xais in the 3D boundary file - suggested by Buchmann
Revision 1.16  2009-07-30 15:30:07  kb
    fixed j-index for +1 eastern boundaries - Hofmeister
Revision 1.15  2007-10-10 10:25:20  kbb
    oops
Revision 1.14  2007-10-10 10:01:19  kbb
    fixed interpolation when model depth > boundary data depth
Revision 1.13  2007-09-30 13:00:43  kbb
    prints real time as part of progressoutput
Revision 1.12  2007-05-11 07:51:27  frv-bjb
    Free dimid numbering for 3d bdy files (order in var still fixed)
Revision 1.11  2007-05-08 09:08:18  kbb
    rudimentary check for valid lower boundary data
Revision 1.10  2005-05-04 11:50:57  kbb
    support for non-climatological 3D boundaries (S,T)
Revision 1.9  2004/04/06 16:32:29  kbb
    TimeDiff --> time_diff
Revision 1.8  2003/12/16 16:50:41  kbb
    added support for Intel/FORT compiler - expanded TABS, same types in subroutine calls
Revision 1.7  2003/10/07 15:10:42  kbb
    use zax_dim as argument to dim_len
Revision 1.6  2003/08/03 09:19:41  kbb
    optimised reading of climatological boundary data

177
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
9.9 Module ncdf_meteo - (Source File: ncdf_meteo.F90)

INTERFACE:

module ncdf_meteo

DESCRIPTION:

USES:

use time, only: string_to_julseconds, time_diff, add_secs, in_interval
use time, only: jul0, seco0, julian_day, secondsofday, timestep, simtime
use time, only: write_time_string, time_str
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: airp, u10, v10, t2, hum, tcc
use meteo, only: fwf_method, evap, precip
use meteo, only: taux, 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, precip_id
integer :: taux_id, tausy_id, swr_id, shf_id
integer :: iextr, jextr, tmax, tmax=-1
integer :: grid_scan=1
logical :: point_source=.false.
logical :: rotated_meteo_grid=.false.

REALTYPE, allocatable :: met_lon(:,), met_lat(:,)
REAL_4B, allocatable :: met_times(:,)
REAL_4B, allocatable :: wrk(:,)
REALTYPE, allocatable :: wrk_dp(:,)

For grid interpolation
REALTYPE, allocatable :: beta(:,)
REALTYPE, allocatable :: ti(:,), ui(:,)
integer, allocatable :: gridmap(:, :,)
REALTYPE, parameter :: pi=3.1415926535897932384626433832795029
REALTYPE, parameter :: deg2rad=pi/180., rad2deg=180./pi
REALTYPE :: southpole(3) = (/0.0,-90.0,0.0/)

179
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"
integer, parameter :: SPECIFIC_HUM=1
integer, parameter :: RELATIVE_HUM=2
integer, parameter :: DEW_POINT=3
integer, parameter :: WET_BULB=4
character(len=10) :: name_tausx="taux"
character(len=10) :: name_tausy="tauys"
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
$Log: ncdf_meteo.F90,v $
Revision 1.26 2009-09-30 11:28:48 bjb
OpenMP threading initial implementation
Revision 1.25 2008-08-29 13:59:12 kb
fixed parallel run for constant metforing with convc<>0
Revision 1.24 2007-09-30 13:00:43 kkb
prints real time as part of progress output
Revision 1.23 2007-08-24 10:43:44 frv-bjb
Allow negative seconds in meteo nc-files input
Revision 1.22 2007-06-28 16:39:41 kkb
initialise evap_id and precip_id to -1
Revision 1.21 2007-06-27 08:39:37 kkb
support for fresh water fluxes at the sea surface - Adolf Stips
Revision 1.20 2006-03-01 13:52:22 kkb
renamed method to met_method
Revision 1.19 2005/05/04 11:45:29 kkb
adding model time stamp on IO
Revision 1.18 2005/04/25 09:25:33 kkb
conv --> convc
Revision 1.17 2005/04/25 07:55:50 kkb
use more general frame for error handling - Umlauf
Revision 1.16 2005/03/31 10:14:20 kkb
flux calc. for point source + combined rot. met. and grid convergence
Revision 1.15 2005/01/12 19:26:16 kkb
fixed printing of south pole
Revision 1.14 2005/01/12 19:17:47 kkb
setting grid_scan depending on lat-axis - Stips
Revision 1.13  2004/08/09 10:43:59  kbb
  correct length of met_times - Buchmann
Revision 1.12  2004/08/09 08:39:36  kbb
  if SPHERICAL and rotated meteo grid fixed turning of wind - Carsten Hansen (FRV)
Revision 1.11  2004/04/06 16:32:29  kbb
  TimeDiff --> time_diff
Revision 1.10  2004/01/15 11:45:01  kbb
  meteo point source forcing - taus, swr and shf - implemented
Revision 1.9   2003/12/16 16:50:41  kbb
  added support for Intel/IFORT compiler - expanded TABS, same types in subroutine calls
Revision 1.8   2003/11/03 14:34:54  kbb
  use time_var_id in addition to time_id
Revision 1.7   2003/10/30 16:31:36  kbb
  check validity of meteo interpolation coefficients
Revision 1.6   2003/10/07 15:16:50  kbb
  now works properly with varying length (time) files
Revision 1.5   2003/07/01 16:38:33  kbb
  cleaned code - new methods
Revision 1.4   2003/06/17 14:53:29  kbb
  default meteo variables names comply with Adolf Stips suggestion + southpole(3)
Revision 1.3   2003/04/07 15:34:15  kbb
  updated to lonc, latc
Revision 1.2.1.1 2002/05/02 14:01:47 gotm
  recovering after CVS crash
Revision 1.4   2001/10/17 14:27:39  bbh
  Met-data can now be read from a series of .nc files
Revision 1.3   2001/07/26 13:57:14  bbh
  Meteo working - needs some polishing
Revision 1.2   2001/06/04 13:15:12  bbh
  Further steps towards full implementation of meteorological forcing
Revision 1.1   2001/05/25 19:26:22  bbh
  ncdf_meteo.F90

**TO DO:**

Unified method of obtaining time info - needs some namelist variables.
Loop over met-files listed in meteo_file.
Make code independent of HIRLAM/ECMWF etc.

9.9.1  _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 meteorological 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 array `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:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>intent(in)</td>
<td>fn</td>
</tr>
<tr>
<td>integer</td>
<td>intent(in)</td>
<td>nstart</td>
</tr>
</tbody>
</table>

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

See module for log.

**LOCAL VARIABLES:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td></td>
<td>i, j, n</td>
</tr>
<tr>
<td>integer</td>
<td></td>
<td>err</td>
</tr>
<tr>
<td>logical</td>
<td></td>
<td>ok=.true.</td>
</tr>
<tr>
<td>REALTYPE</td>
<td></td>
<td>olon, olat, rlon, rlat, x</td>
</tr>
<tr>
<td>character</td>
<td></td>
<td>name_thisvar</td>
</tr>
</tbody>
</table>

9.9.2  

**INTERFACE:**

```fortran
subroutine get_meteo_data_ncdf(loop)
IMPLICIT NONE
```

**DESCRIPTION:**

Do book keeping about when new fields are to be read. Set variables used by `de_meteo` and finally calls `read_data` if necessary.

**INPUT PARAMETERS:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>intent(in)</td>
<td>loop</td>
</tr>
</tbody>
</table>

**INPUT/OUTPUT PARAMETERS:**

**OUTPUT PARAMETERS:**

**REVISION HISTORY:**

See module for log.

**LOCAL VARIABLES:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td></td>
<td>i, indx</td>
</tr>
<tr>
<td>REALTYPE</td>
<td></td>
<td>t</td>
</tr>
<tr>
<td>logical, save</td>
<td></td>
<td>first=.true.</td>
</tr>
<tr>
<td>integer, save</td>
<td></td>
<td>save_n=1</td>
</tr>
</tbody>
</table>

182
9.9.3 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

OUTPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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

TO DO:

    Need a variable to indicate how much to read from each file.

9.9.4 read_data -

INTERFACE:

    subroutine read_data()
    IMPLICIT NONE

183
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:

OUTPUT PARAMETERS:

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

9.9.5 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(:,:)

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
9.10 Module ncdf_river - (Source File: ncdf_rivers.F90)

INTERFACE:

    module ncdf_river

DESCRIPTION:

USES:

    use time, only: string_to_julsec, time_diff, add_secs, in_interval
    use time, only: jul0, secs0, julianaday, secondsofday, timestep
    use time, only: write_time_string, timestep, 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

IMPLICIT NONE

PRIVATE 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(:,:)
    integer, allocatable :: r_bio(:, :)
    #endif

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard
    $Log: ncdf_rivers.F90,v $
    Revision 1.9  2010-03-30 10:06:51  kb
    removed temporarily diagnostic output
    Revision 1.8  2007-09-30 13:00:43  kbk
    prints real time as part of progresoutput
    Revision 1.7  2005-09-23 11:27:43  kbk
    support fo nutrient loading in rivers
    Revision 1.6  2005/05/04 11:45:29  kbk

185
adding model time stamp on IO
Revision 1.5 2005/01/13 09:20:47 kbk
support for T and S specifications in rivers - Stips
Revision 1.4 2004/04/06 16:32:29 kbk
TimeDiff --> time_diff
Revision 1.3 2003/10/14 10:05:54 kbk
checks if indices are in subdomain + cleaning
Revision 1.2 2003/04/23 11:54:03 kbk
cleaned code + TABS to spaces
Revision 1.1.1.1 2002/05/02 14:01:48 gotm
recovering after CVS crash
Revision 1.1 2001/10/07 14:50:22 bbh
Reading river data implemented - NetCFD

TO DO:

9.10.1  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

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    See module for log.

LOCAL VARIABLES:

    integer :: i,j,m,n
    integer :: err
    integer :: j1,s1,j2,s2
    character(len=256) :: time_units
    character(len=256) :: bio_name

186
9.10.2 get_river_data_ncdf -.

INTERFACE:

    subroutine get_river_data_ncdf(loop)
    IMPLICIT NONE

DESCRIPTION:

INPUT PARAMETERS:

    integer, intent(in) :: loop

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

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
9.11 Module 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 ability 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,zu_id
    integer :: SlVx_id,Slru_id
    integer :: V_id,zv_id
    integer :: SlVx_id,Slrv_id
    #ifndef NO_3D
    integer :: ssen_id,ssun_id,ssvn_id
    integer :: ssso_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
    #ifdef 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
#endif
#endif

integer :: xlen,ylen,zlen
integer :: status
integer :: start(5),edges(5)

REVISION HISTORY:

Original author(s): Karsten Bolding
$Log: ncdf_restart.F90,v $
Revision 1.5  2010-03-30 11:52:32  kb
removing adaptive_coordinates
Revision 1.3  2009-09-23 09:54:52  kb
fixed typos in DESCRIPTION
Revision 1.2  2007-10-19 07:52:36  kbk
zub and zvb not in hotstart files anymore
Revision 1.1  2007-09-21 13:03:42  kbk
added drop-in NetCDF replacement for binary hotstart file (default is binary)
9.12 Module Encapsulate netCDF mean quantities (Source File: ncdf_mean.F90)

INTERFACE:

module ncdf_mean

DESCRIPTION:

USES:

use output
INPLICIT NONE

PUBLIC DATA MEMBERS:

integer :: ncid=-1
integer :: x_dim,y_dim,z_dim
time_dim :: time_dim
integer :: time_id
swrmean_id,ustarmean_id,ustar2mean_id
integer :: uumean_id,vvmean_id,wmean_id
integer :: saltmean_id,tempmean_id,hmean_id

REALTYPE, parameter :: hh_missing=-10.0
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 :: tke_missing=-9999.0
REALTYPE, parameter :: eps_missing=-9999.0

REAL_4B, dimension(:), allocatable :: ws
Original author(s): Adolf Stips & Karsten Bolding
$Log: ncdf_mean.F90,v $
Revision 1.2 2005-04-25 09:32:34 kbk
added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.1 2004/03/29 15:38:10 kbk
possible to store calculated mean fields
9.13 Module 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 parameter ncdf_read_topo_file().
For a full description of the required variables see the documentation 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 : 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, yxxcord
use domain, only : dx, dy
use domain, only : xc, yc
use domain, only : xx, yy
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)
$Log: ncdf_topo.F90,v $
Revision 1.25 2010-03-26 19:12:49 kb
aborting if grid_type=4
Revision 1.24 2009-12-22 08:44:38 kb
added conditional compilation checks - Klingbeil
Revision 1.23 2009-12-10 14:22:52 kb
fixed typos - Hofmeister
Revision 1.22 2009-10-13 13:15:11 kb
added pseudo-coordinates when grid-type 3 or 4
Revision 1.21 2009-10-08 16:08:00 kb
axes defined in entire domain - cartesian, spherical
Revision 1.20 2009-10-05 11:40:03 kb
fixed behaviour for grid_type=3 and lonx, latx, convx not in topo.nc
Revision 1.19 2009-09-30 05:32:48 kb
fixed calculation of dx, dy when dlon, dlat not present
Revision 1.18 2009-09-23 10:09:20 kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.17 2009-09-23 10:04:40 kb
reverted to v1.15 - to allow for major update
Revision 1.15 2007-05-26 15:20:37 kkb
print NetCDF version info
Revision 1.14 2007-02-07 16:32:22 kkb
added spatial varying bottom roughness
Revision 1.13 2006-11-24 09:10:56 frv-bjb
 Higher accuracy in x0,dx computations
Revision 1.12 2006-01-29 20:32:34 hb
Small LaTeX corrections to source code documentation
Revision 1.11 2005-11-17 13:50:22 kkb
fixes to compile with gfortran
Revision 1.10 2005/06/17 07:57:46 frv-bjb
 Bug fix: fail on dlat/lat0 versions
Revision 1.9  2005/06/14 13:36:01 frv-bjb
temporary KBK stop statement deleted
Revision 1.8 2005/06/10 16:16:41 kkb
documentation updated
Revision 1.7 2005/06/10 16:01:22 kkb
test and use real axis before using axis offset+increment method
Revision 1.6 2005/04/25 09:32:34 kkb
added NetCDF IO rewrite + de-stag of velocities - Umlauf

LOCAL VARIABLES:

private ncdf_read_2d

9.13.1 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():

192
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)

9.13.2 coords_and_grid_spacing

**INTERFACE:**

subroutine coords_and_grid_spacing(ncid,varid,iextr,cordname,x0,dx)

**USES:**
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) :: cordname

OUTPUT PARAMETERS:

- REALTYPE, intent(out) :: x0, dx

REVISION HISTORY:

Original author(s): Bjarne Buchmann

9.13.3 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 function, 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

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:

    IMPLICIT NONE

INPUT PARAMETERS:

    integer, intent(in)        :: ncid,id
    integer, optional         :: netcdf_real
    character(len=*), optional :: units,long_name
    REALTYPE, optional        :: valid_min,valid_max,valid_range(2)
    REALTYPE, optional        :: scale_factor,add_offset
    REALTYPE, optional        :: FillValue,missing_value
    character(len=*), optional :: C_format,FORTRAN_format

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard
    See ncdfout module

LOCAL VARIABLES:

    integer        :: len,iret
    integer        :: ft
    REAL_4B        :: vals(2)

9.14.1  Initialise 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 bathymetry. If the logical flag "init3d" evaluates false, no information about the vertical grid is initialised (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
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
$Log: init_grid_ncdf.F90,v $
Revision 1.9 2009-10-13 13:15:11 kb
added pseudo-coordinates when grid-type 3 or 4
Revision 1.8 2009-09-28 14:39:06 kb
INCLUDE_HALOS -> SAVE_HALOS, renamed dimension names
Revision 1.7 2009-09-23 10:11:48 kb
rewrite of grid-initialisation, optional grid info saved to file, -DSAVE_HALO, updated documentation
Revision 1.6 2009-03-13 14:44:14 kb
grid information in NF_DOUBLE
Revision 1.5 2007-10-16 07:14:35 kbk
pseudo coordinate variables for curvi-linear grids
Revision 1.4 2007-03-30 13:11:00 bb
Use of adaptive and hybrid vertical coordinates technically enabled
Revision 1.3 2007-02-22 08:48:12 kbk
possible to save masks (az, au, av)
Revision 1.2 2005/04/29 12:45:41 kbk
stricter COARDS conforming
Revision 1.1 2005/04/25 09:32:34 kbk
added NetCDF IO rewrite + de-stag of velocities - Umlauf

LOCAL V ARIABLES:
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

9.14.2 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:

198
9.14.3 Initialise 2D netCDF variables

INTERFACE:

    subroutine init_2d_ncdf(fn,title,starttime)

DESCRIPTION:

USES:

    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: fwm_method

IMPLICIT NONE
INPUT PARAMETERS:

    character(len=*) , intent(in) :: fn,title,starttime

DEFINED PARAMETERS:

    logical, parameter :: init3d=.false.

REVISION HISTORY:

$Log: init_2d_ncdf.F90,v $
Revision 1.9  2008-09-16 11:21:51  kb
if -DUSE_BREAKS save break statistics
Revision 1.8  2007-06-27 08:39:37  kbb
support for fresh water fluxes at the sea surface - Adolf Stips
Revision 1.7  2005-04-25 09:32:34  kbb
added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.6  2003/10/07 09:11:38  kbb
fixed typo: longname U10 -> V10 for v10 variable
Revision 1.5  2003/06/17 14:53:29  kbb
default meteo variables names comply with Adolf Stips suggestion + southpole(3)
Revision 1.4  2003/05/09 11:38:26  kbb
added proper undef support - based on Adolf Stips patch
Revision 1.3  2003/04/23 11:53:24  kbb
save lat/lon info for spherical grid
Revision 1.2  2003/04/07 12:48:11  kbb
uses metfacing variable
Revision 1.1.1.1  2002/05/02 14:01:48  gotm
recovering after CVS crash
Revision 1.5  2001/10/26 12:18:06  bbb
No actual storing of data in init_2d_ncdf.F90 -> save_2d_ncdf.F90
Revision 1.4  2001/09/27 08:35:10  bbb
Saving meteo again - in .2d.nc file
Revision 1.3  2001/09/24 14:13:25  bbb
xc and yc have changing shape depending on grid_type
Revision 1.2  2001/09/19 11:20:32  bbb
Explicit de-allocates memory when -DFORTRAN90
Revision 1.1  2001/09/13 14:50:02  bbb
Cleaner and smaller NetCDF implementation + better axis support

LOCAL VARIABLES:

    integer :: err
    integer :: xlen,ylen
    integer :: scalar(1),axisdim(1),f2_dims(2),f3_dims(3)
    REALTYPE :: fv,mv,vr(2)
    character(len=80) :: history,ts
9.15  save_2d_ncdf() - saves 2D-fields. (Source File: save_2d_ncdf.F90)

INTERFACE:

    subroutine save_2d_ncdf(secs)

DESCRIPTION:

USES:

    use exceptions
    use ncdf_2d
    use grid_ncdf
    use domain,     only: ioff,joff,imin,imax,jmin,jmax
    use domain,     only: H,az,au,sv,crit_depth
    use variables_2d, only: z,D,U,DU,V,DV,res_u,res_v,surfdiv
    #if 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,swr,shf

IMPLICIT NONE

INPUT PARAMETERS:

    REALTYPE, intent(in) :: secs
    !DEFINED PARAMETERS:
    logical, parameter   :: save3d=.false.

REVISION HISTORY:

  Original author(s): Karsten Bolding & Hans Burchard
  $Log: save_2d_ncdf.F90,v $
  Revision 1.8  2008-09-16 11:21:51  kb
  if -DUSE_BREAKS save break statistics
  Revision 1.7  2007-06-27 09:32:34  kbb
  support for fresh water fluxes at the sea surface - Adolf Stips
  Revision 1.6  2005-04-25 09:32:34  kbb
  added NetCDF IO rewrite + de-stag of velocities - Umlauf
  Revision 1.5  2003/06/17 14:53:29  kbb
  default meteo variables names comply with Adolf Stips suggestion + southpole(3)
  Revision 1.4  2003/05/09 11:38:26  kbb
  added proper undef support - based on Adolf Stips patch
  Revision 1.3  2003/04/23 11:53:24  kbb
  save lat/lon info for spherical grid
  Revision 1.2  2003/04/07 12:43:12  kbb
  SPHERICAL and NO_BAROCLINIC
  Revision 1.1.1.1  2002/05/02 14:01:48  gotm
  recovering after CVS crash
  Revision 1.3  2001/10/26 12:18:06  bbh
  No actual storing of data in init_2d_ncdf.F90 -> save_2d_ncdf.F90
  Revision 1.2  2001/09/27 08:35:10  bbh

201
9.15.1 Initialise 3D netCDF variables

INTERFACE:

subroutine init_3d_ncdf(fn,title,starttime)

DESCRIPTION:

USES:

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
#ifdef SPM
use suspended_matter, only: spm_save
#endif
#ifdef GETM_BIO
use bio_var, only: numc,var_names,var_units,var_long
#endif

IMPLICIT NONE

INPUT PARAMETERS:

character(len=*) , intent(in) :: fn,title,starttime

DEFINED PARAMETERS:

logical, parameter :: init3d=.true.

REVISION HISTORY:

$Log: init_3d_ncdf.F90,v $
Revision 1.16  2009-04-23 14:30:37  lars
corrected wrong units/name for NN and SS
Revision 1.15  2009-04-22 10:09:36  lars
support for bottom stress output
Revision 1.14  2009-01-05 09:57:06  kb
option for storing SS and NN
Revision 1.13  2007-03-30 13:11:00  hb
Use of adaptive and hybrid vertical coordinates technically enabled
Revision 1.12 2007-02-20 13:52:15 kkb
solar radiation -> 3d field - possible to save
Revision 1.11 2006-03-17 11:06:33 kkb
cleaner inclusion of SPM module
Revision 1.10 2005/09/23 11:27:10 kkb
support for biology via GOTMs biology modules
Revision 1.9 2005/04/25 09:32:34 kkb
added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.8 2004/10/07 15:46:56 kkb
removed wrongly placed comments for save_nuh
Revision 1.7 2004/06/15 08:25:57 kkb
added supoort for spm - Ruiz
Revision 1.6 2004/05/04 09:23:51 kkb
hydrostatic consistency criteria stored in .3d.nc file
Revision 1.5 2003/12/16 12:51:04 kkb
preparing for proper support for SPM (manuel)
Revision 1.4 2003/09/05 11:38:26 kkb
added proper undef support - based on Adolf Stips patch
Revision 1.3 2003/04/23 11:53:24 kkb
save lat/lon info for spherical grid
Revision 1.2 2003/04/07 12:51:26 kkb
CURVILINEAR --> defined(SPHERICAL) || defined(CURVILINEAR)
Revision 1.1.1.1 2002/05/02 14:01:46 gotm
recovering after CVS crash
Revision 1.7 2001/10/25 16:16:20 bbh
No actual storing of data in init_3d_ncdf.F90 -> save_3d_ncdf.F90
Revision 1.6 2001/10/23 14:19:20 bbh
Stores h if general vertical coordinates
Revision 1.5 2001/10/23 07:37:17 bbh
Saving spm - if calc_spm and save_spm are both true
Revision 1.4 2001/10/22 08:03:13 bbh
Misplaced #else
Revision 1.3 2001/09/24 14:13:25 bbh
xc and yc have changing shape depending on grid_type
Revision 1.2 2001/09/19 11:20:32 bbh
Explicit de-allocates memory when -DFORTRAN90
Revision 1.1 2001/09/13 14:50:02 bbh
Cleaner and smaller NetCDF implementation + better axis support

LOCAL VARIABLES:

integer :: err
integer :: n,rc
integer :: xlen,ylen,zlen
integer :: scalar(1),axisdim(1),f3_dims(3),f4_dims(4)
REALTYPE :: fv,mv,vr(2)
character(len=80) :: history,ts
INTERFACE:

subroutine save_3d_ncdf(secs)

DESCRIPTION:

USES:

use exceptions
use ncdf_3d
use grid_ncdf
use domain, only: ioff, joff, imin, imax, jmin, jmax, kmax
use domain, only: H, HU, HV, az, au, av, min_depth
#if defined CURVILINEAR || defined SPHERICAL
use domain, only: dx, dy
#else
use domain, only: dxc, dyc
#endif
use variables_2d, only: z, D
use variables_2d, only: U, V, DU, DV
use variables_3d, only: dt, kmin, ho, hn, uu, hun, vv, hvn, ww, hcc, SS
use variables_3d, only: taux, tauy
#ifndef NO_BAROCLINIC
use variables_3d, only: S, T, rho, rad, NN
#endif
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, ws3d
#endif
use parameters, only: g, rho_0

IMPLICIT NONE

INPUT PARAMETERS:

REALTYPE, intent(in) :: secs

!DEFINED PARAMETERS:
logical, parameter :: save3d=.true.

REVISION HISTORY:

Original author(s): Karsten Bolding & Hans Burchard
$Log: save_3d_ncdf.F90,v $
Revision 1.19  2009-04-22 10:09:36  lars
support for bottom stress output
Revision 1.18  2009-01-05 09:57:06  kb
option for storing SS and NN

204
ii

min,iimax,jjmin,jjmax -> imin,imax,jmin,jmax

Revision 1.17 2007-06-07 10:25:19 kkb

destag of vertical velocities
Revision 1.16 2007-05-26 12:19:31 kkb

saving physical vertical velocities
Revision 1.15 2007-05-22 09:37:20 kkb

introduce buoy() and rename rho() to buoy()i where appropriate
Revision 1.13 2007-02-20 13:52:15 kkb

solar radiation -> 3d field - possible to save
Revision 1.12 2006-03-17 11:06:33 kkb

cleaner inclusion of SPM module
Revision 1.11 2005/09/23 11:27:10 kkb

support for biology via GUTMs biology modules
Revision 1.10 2005/04/25 09:32:34 kkb

added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.9 2004/06/15 08:25:57 kkb

added suppoort for spm - Ruiz
Revision 1.8 2004/05/04 09:23:51 kkb

hydrostatic consistency criteria stored in .3d.nc file
Revision 1.7 2003/12/16 12:47:11 kkb

rho_0 and g from parameters (manuel)
Revision 1.6 2003/12/08 07:21:53 hb

use proper layer heights for saving velocities
Revision 1.5 2003/09/09 11:53:13 kkb

forgot to delete some debug lines
Revision 1.4 2003/05/09 11:38:26 kkb

added proper undef support - based on Adolf Stips patch
Revision 1.3 2003/04/23 11:53:24 kkb

save lat/lon info for spherical grid
Revision 1.2 2003/04/07 12:43:12 kkb

SPHERICAL and NO_BAROCLINIC
Revision 1.1.1.1 2002/05/02 14:01:48 gotm

recovering after CVS crash
Revision 1.4 2001/10/25 16:16:21 bbh

No actual storing of data in init_3d_ncdf.F90 -> save_3d_ncdf.F90
Revision 1.3 2001/10/23 14:19:20 bbh

Stores h if general vertical coordinates
Revision 1.2 2001/10/23 07:37:17 bbh

Saving spm - if calc_spm and save_spm are both true
Revision 1.1 2001/09/13 14:50:02 bbh

Cleaner and smaller NetCDF implementation + better axis support

LOCAL VARIABLES:

integer :: err,n
integer, save :: start(4),edges(4)
REALTYPE :: DONE(E2DFIELD)

205
9.17  get_field_ncdf - (Source File: get_field_ncdf.F90)

INTERFACE:

    subroutine get_field_ncdf(fname, var, f)

DESCRIPTION:

From a NetCDF file - fname - read the variable - var - into the field - f.

USES:

    use domain, only: imin, imax, jmin, jmax, kmax

IMPLICIT NONE

INPUT PARAMETERS:

    character(len=*) , intent(in) :: fname, var

INPUT/OUTPUT PARAMETERS:

    REALTYPE , intent(out) :: f(I3DFIELD)

OUTPUT PARAMETERS:


REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard
    $Log: get_field_ncdf.F90,v $
    Revision 1.3 2007-06-07 10:25:19  kbk
    iimin, iimax, jjmin, jjmax -> imin, imax, jmin, jmax
    Revision 1.2 2003-04-23 11:54:03  kbk
    cleaned code + TABS to spaces
    Revision 1.1.1.1 2002/05/02 14:01:48  gotm
    recovering after CVS crash
    Revision 1.3 2001/10/22 08:10:43  bbh
    De-allocate wrk #ifdef FORTRAN90
    Revision 1.1 2001/05/10 11:38:29  bbh
    Added get_field_ncdf() + various small bug fixes

LOCAL VARIABLES:

    integer :: rc, err, ncid, var_id, i, j, k, size, index
    integer :: start(3), edges(3)
    REAL_4B, allocatable :: wrk(:)
9.18 read_field_ncdf - (Source File: read_field_ncdf.F90)

INTERFACE:
   subroutine read_field_ncdf(fname,var,nf,f)

DESCRIPTION:
From a NetCDF files - fname - read the variable - var - into the field - f.

USES:
   use domain, only: imin,jmin,imax,jmax,kmax,iextr,jextr,ioff,joff
   use domain, only: H,az
   use variables_3d, only: hn
   IMPLICIT NONE

INPUT PARAMETERS:
   character(len=*), intent(in) :: fname,var
   integer, intent(in) :: nf

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:
   REALTYPE, intent(inout) :: f(I3DFIELD)

REVISION HISTORY:
   Original author(s): Karsten Bolding & Hans Burchard
   $Log: read_field_ncdf.F90,v $
   Revision 1.7 2007-06-07 10:25:19 kbk
   Revision 1.6 2004-09-16 14:32:25 kbk
   Revision 1.5 2003/08/03 08:29:28 kbk
   Revision 1.4 2003/05/06 16:25:16 kbk
   Revision 1.3 2003/04/23 11:54:03 kbk
   Revision 1.2 2003/04/07 12:39:59 kbk
   Revision 1.1.1.1 2002/05/02 14:01:47 gotm
   Recovering after CVS crash

LOCAL VARIABLES:
   integer :: il,jl,iloc,jloc,index
   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(:,,:)
9.19  ncdf_close() - closes the specified NetCDF file.  (Source File: ncdf_close.F90)

INTERFACE:

    subroutine ncdf_close()

DESCRIPTION:

USES:

    use ncdf_2d, only: nc2d => ncid
    #ifndef NO_3D
    use ncdf_3d, only: nc3d => ncid
    #endif
    IMPLICIT NONE

INPUT PARAMETERS:

INPUT/OUTPUT PARAMETERS:

OUTPUT PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding & Hans Burchard
    $Log: ncdf_close.F90,v $
    Revision 1.5  2006-02-03 15:03:53  kbk
    removed call to save_2d_nCDF() - Stips
    Revision 1.4  2006-02-02 17:51:37  kbk
    do not try and save to un-opened NetCDF files
    Revision 1.3  2003/04/23 11:54:03  kbk
    cleaned code + TABS to spaces
    Revision 1.2  2003/04/07 12:46:06  kbk
    NO_3D
    Revision 1.1.1.1  2002/05/02 14:01:47  gotm
    recovering after CVS crash
    Revision 1.2  2001/09/13 14:56:58  bbh
    Also updated
    Revision 1.1.1.1  2001/04/17 08:43:07  bbh
    initial import into CVS

LOCAL VARIABLES:

    integer :: err
    REALTYPE :: dummy=-_ONE_
9.20 Module field_2d_ncdf - Interface with 2D field from file (Source File: get_2d_field_ncdf.F90)

INTERFACE:

module field_2d_ncdf

DESCRIPTION:

This module is responsible for reading 2D field quantities contained in a netCDF file. USES:

use netcdf
use exceptions
IMPLICIT NONE

PUBLIC MEMBER FUNCTIONS:

public get_2d_field_ncdf

PUBLIC DATA MEMBERS:

DEFINED PARAMETERS:

REVISION HISTORY:

Original author(s): Karsten Bolding, Lars Umlauf

9.20.1 get_2d_field_ncdf()

INTERFACE:

subroutine get_2d_field_ncdf(fn,varname,il,ih,jl,jh,field)

USES:

IMPLICIT NONE

DESCRIPTION:

A two-dimensional netCDF variable with specified global range $i_l < i < i_h$ and $j_l < j < j_h$ is read into field. It is checked if the sizes of the fields correspond exactly. When calling this functions, remember that FORTRAN netCDF variables start with index 1.

INPUT PARAMETERS:

character(len=*), intent(in) :: fn,varname
integer, intent(in) :: il,ih,jl,jh

OUTPUT PARAMETERS:

209
REALTYPE, intent(out) :: field(:, :)

LOCAL VARIABLES:

integer, dimension(2) :: start
integer, dimension(2) :: edges
integer, dimension(2) :: ubounds
integer :: status, ncid, varid
9.21 Create a GETM NetCDF 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 domain, only: ioff,joff
use domain, only: imin,imax,jmin,jmax,kmax
use domain, only: vert_cord
#endif
use bio, only: bio_calc
use bio_var, only: numc
#endif

IMPLICIT NONE

INPUT PARAMETERS:

character(len=*) , intent(in) :: fname
integer, intent(in) :: loop
integer, intent(in) :: runtype

DEFINED PARAMETERS:

REVISION HISTORY:

Original author(s): Karsten Bolding
$Log: create_restart_ncdf.F90,v $
Revision 1.10 2010-03-30 11:48:37 kb
removing adaptive_coordinates
Revision 1.8 2010-01-21 15:46:23 kb
fixed BIO-restart
Revision 1.7 2009-09-25 12:14:56 kb
INCLUDE_HALOS --> SAVE_HALOS
Revision 1.6 2009-09-23 09:54:52 kb
fixed typos in DESCRIPTION
Revision 1.5 2009-08-21 10:39:00 kb
-DINCLUDE_HALOS will include halo-zones when writing/reading NetCDF hotstart files
Revision 1.4 2009-07-18 12:36:01 kb
fixed SPM hot-start bug - Hofmeister
Revision 1.3 2009-04-27 08:03:02 kb
getm initialise.F90
zu b and zvb not in hotstart files anymore

Revision 1.1  2007-09-21 13:03:42  kbk
added drop-in NetCDF replacement for binary hotstart file (default is binary)

LOCAL VARIABLES:

character(len=80) :: history,tts
character(len=80) :: title
character(len=80) :: str_error
9.22 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 ncf Restart
    use domain, only: grid_type
    use domain, only: xc, yc, lonc, latc
    use domain, only: imin, imax, jmin, jmax, kmax
    use variables_2d
    ifndef NO_3D
        use variables_3d
    endif
    ifdef GETM_BIO
        use bio, only: bio_calc
        use bio_var, only: numc
    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

!DEFINED PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding
    $Log: write_restart_ncdf.F90,v $
    Revision 1.14  2010-03-30 12:03:24  kb
    fixed BIO restart in NetCDF
    Revision 1.13  2010-03-30 11:48:38  kb
    removing adaptive_coordinates
    Revision 1.11  2010-01-21 15:46:24  kb
    fixed BIO-restart
    Revision 1.10  2009-09-25 12:17:26  kb
    removed undef [IJ]RANGE
    Revision 1.9  2009-09-23 09:54:53  kb
    fixed typos in DESCRIPTION
    Revision 1.8  2009-08-21 10:39:01  kb
-DINCLUDE_HALOS will include halo-zones when writing/reading NetCDF hotstart files

Revision 1.7  2009-07-18 12:36:01 kb
fixed SPM hot-start bug - Hofmeister
Revision 1.6  2009-04-27 09:22:55 kb
mean calculation de-activated with -DN0_3D
Revision 1.5  2007-11-12 13:50:17 kb
also need bio_calc
Revision 1.4  2007-10-19 07:52:36 kbb
zub and zvb not in hotstart files anymore
Revision 1.3  2007-10-04 13:55:37 kbb
fixed variable type of runtype
Revision 1.2  2007-10-03 06:59:23 kbb
NetCDF restart uses runtype properly
Revision 1.1  2007-09-21 13:03:42 kbb
added drop-in NetCDF replacement for binary hotstart file (default is binary)

LOCAL VARIABLES:

integer :: k,n
REALTYPE, allocatable :: zax(:)

9.22.1 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
#ifdef GETM_BIO
use bio, only: bio_calc
use getm_bio, only: bio_init_method
#endif

IMPLICIT NONE

INPUT PARAMETERS:

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

DEFINED PARAMETERS:

REVISION HISTORY:
Original author(s): Karsten Bolding
Log: open_restart_ncdf.F90,v $ 
Revision 1.7 2010-03-30 11:48:37 kb
removing adaptive_coordinates
Revision 1.5 2010-01-21 15:46:23 kb
fixed BIO-restart
Revision 1.4 2009-09-23 09:54:53 kb
fixed typos in DESCRIPTION
Revision 1.3 2009-04-27 08:03:02 kb
getm/initialise.F90
Revision 1.2 2007-10-19 07:52:36 kbb
zub and zvb not in hotstart files anymore
Revision 1.1 2007-09-21 13:03:42 kbb
added drop-in NetCDF replacement for binary hotstart file (default is binary)

LOCAL VARIABLES:

    integer :: dimids(3)
9.23  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 ’ncs’ 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
#ifdef NO_3D
use variables_3d
#endif
#ifdef GETM_BIO
use bio, only: bio_calc
use bio_var, only: numc
use getm_bio, only: bio_init_method
#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 PARAMETERS:

REVISION HISTORY:

    Original author(s): Karsten Bolding
$Log: read_restart_ncdf.F90,v $
Revision 1.14  2010-03-30 11:48:38  kb
removing adaptive coordinates
Revision 1.12 2010-01-21 15:46:24 kb
fixed BIO-restart
Revision 1.11 2009-09-25 12:14:56 kb
INCLUDE_HALOS --> SAVE_HALOS
Revision 1.10 2009-09-23 09:54:53 kb
fixed typos in DESCRIPTION
Revision 1.9 2009-08-21 10:39:00 kb
-DINCLUDE_HALOS will include halo-zones when writing/reading NetCDF hotstart files
Revision 1.8 2009-07-18 12:36:01 kb
fixed SPM hot-start bug - Hofmeister
Revision 1.7 2009-04-27 09:22:55 kb
mean calculation de-activated with -DNO_3D
Revision 1.6 2007-11-12 13:50:17 kb
also need bio_calc
Revision 1.5 2007-10-19 07:52:36 kkb
zub and zvb not in hotstart files anymore
Revision 1.4 2007-10-16 13:34:05 kkb
differentiate between mandatory and non-mandatory fields
Revision 1.3 2007-10-04 13:55:04 kkb
only read S and T if runtype .ge. 3
Revision 1.2 2007-10-03 06:59:23 kkb
NetCDF restart uses runtype properly
Revision 1.1 2007-09-21 13:03:42 kkb
added drop-in NetCDF replacement for binary hotstart file (default is binary)

LOCAL VARIABLES:

integer :: il,ih,iloc,len
integer :: jl,jh,jloc,jlen

9.23.1 Initialise mean netCDF variables

INTERFACE:

subroutine init_mean_ncdf(fn,title,starttime)

DESCRIPTION:

USES:

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

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
$Log: init_mean_ncdf.F90,v $
Revision 1.2 2005-04-25 09:32:34 kbk
added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.1 2004/03/29 15:38:10 kbk
possible to store calculated mean fields

LOCAL VARIABLES:

integer :: err
integer :: xlen,ylen,zlen
integer :: scalar(1),axisdim(1),f3_dims(3),f4_dims(4)
REALTYPE :: fv,mv,vr(2)
character(len=80) :: history,tts
9.24 Initialise mean netCDF variables (Source File: save_mean_ncdf.F90)

INTERFACE:

subroutine save_mean_ncdf(secs)

DESCRIPTION:

USES:

use exceptions
use ncdf_mean
use grid_ncdf
use diagnostic_variables
use domain, only: ioff,joff,imin,imax,jmin,jmax,kmax
use domain, only: H,az
use variables_3d, only: kmin
IMPLICIT NONE

INPUT PARAMETERS:

REALTYPE, intent(in) :: secs
!DEFINED PARAMETERS:
logical, parameter :: save3d=.true.

REVISION HISTORY:

Original author(s): Adolf Stips & Karsten Bolding
$Log: save_mean_ncdf.F90,v $
Revision 1.3 2007-06-07 10:25:19 kbk
iimin,iimax,jjmin,jjmax -> imin,imax,jmin,jmax
Revision 1.2 2005-04-25 09:32:34 kbk
added NetCDF IO rewrite + de-stag of velocities - Umlauf
Revision 1.1 2004/03/29 15:38:10 kbk
possible to store calculated mean fields

LOCAL VARIABLES:

integer :: err
integer :: start(4),edges(4)
integer, save :: n3d=0
References


