IcoAtmosBenchmark
DYNAMICO kernels

SPPEXA/AIMES Benchmarking team

May 17, 2018

Contents

Chapter 1

Brief introduction of DYNAMICO

1.1 *DYNAMICO* is

DYNAMICO[*1)]is a new dynamical core for LMD-Z, the atromspheric GCM part of IPSL-CM Earth System Model. *DYNAMICO* is funded by the Indo-French Centre for the Promotion of Advanced Research, by IPSL and by the G8 Research Councils Initiative on Multilateral Research Funding, project ICOMEX.

The primary goal of *DYNAMICO* is to re-formulate in LMD-Z the horizontal advection and dynamics on a icosahedral grid, while preserving or improving their qualities with respect to accuracy, conservation laws and wave dispersion. A broader goal is to revisit all fundamental features of the dynamical core, especially the shallow-atmosphere/traditional approximation, the vertical coordinate and the coupling with physics. Also efficient implementation of present and future supercomputing architectures is a key issue.

This manual describes the overview of *DYNAMICO* and each kernel program briefly. For the details of *DYNAMICO*, see Dubos et al. (2015), etc.

Kernel programs for *DYNAMICO* are taken from *DYNAMICO* ver 1.0, r339. Main feature of *DYNAMICO*-1.0 are;

- hydrostatic, traditional shallow atmosphere,
- icosahedral-hexagonal C-grid in horizontal, mass-based Lorentz staggering in vertical,
- Mimetic finite difference + slope-limited finite volume transport, and
- explicit Runge-Kutta time stepping.

1.2 Governing equations

Basic scheme of *DYNAMICO* is the energy/voticity conserving schemes and the curl (vector-invariant) form. To deliver governing equations, *DYNAMICO* adopts the Hamiltonian formulation of the equations of motion. This Hamiltonian theory has been extended for compressible hydrostatic flows and for non-Eulerian vertical coordinates (Tort and Dubos, 2014; Dubos and Tort, 2014). Derivation of governing equations is complicated, we skip it here. See Dubos et al. (2015) etc.

1.3 Horizontal and vertical grid

DYNAMICO adopts the icosahedral-hexagonal C-grid in horizontal and mass-based Lorentz staggering grid in vertical. Figure 1.1 shows horizontal and vertical grids.

Scalar variables, such as entropy Θ, are defined on the center of hexagonal control volume (circle points in the figure), velocities and fluxes, are defined on the edge (square points), and tracer are defined on the vertex (triangle points). See Dubos et al. (2015) for details.

[*1)] This section is based on the *DYNAMICO* Wiki page (http://forge.ipsl.fr/dynamico/wiki)

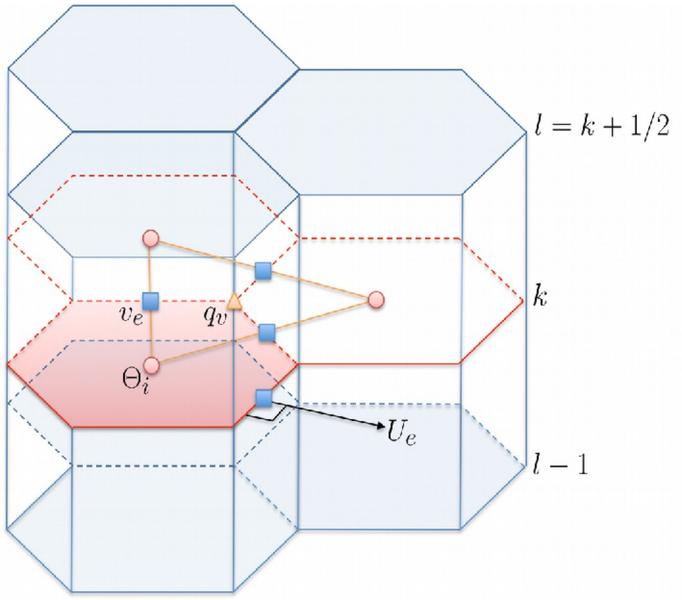


Figure 1.1: Icosahedral C-grid and Lorenz grid.

1.4 Parallelization

Like *NICAM* and *DYNAMICO*, icosahedral grids on entire globe can be separated by 10 “diamonds”, each are consist of neighboring two triangles of an icosahedron. Each diamonds can be divided in `nsplit_i`×`nsplit_j` areas, and one of divided area, called “patch” in *DYNAMICO*, is the basis of domain decomposition. `nsplit_i` and `nsplit_j` are control parameter and read from configuration file on execution. One MPI process can handle `ndomain` patches, which is decided by the number of total patchs and the number of MPI processes.

1.5 Data structure

Basic data structure in *DYNAMICO* is a `t_field`, as shown in List 1.1. One instance of `t_field` is to access one field within one patch. One MPI process may handle several patches, and one field may be usually an array of `t_field`. Allocation and halo-exchange routines are work on `t_field(:)` variables, and other high-level computational routines work on them, too. See the next section as an example.

List 1.1: `t_field` structure

```
TYPE t_field
  CHARACTER(30)      :: name
  REAL(rstd),POINTER :: rval2d(:)
  REAL(rstd),POINTER :: rval3d(:,:)
  REAL(rstd),POINTER :: rval4d(:,:,:)

  INTEGER,POINTER :: ival2d(:)
  INTEGER,POINTER :: ival3d(:,:)
  INTEGER,POINTER :: ival4d(:,:,:)

  LOGICAL,POINTER :: lval2d(:)
  LOGICAL,POINTER :: lval3d(:,:)
  LOGICAL,POINTER :: lval4d(:,:,:)

  INTEGER :: ndim
  INTEGER :: field_type
  INTEGER :: data_type
  INTEGER :: dim3
  INTEGER :: dim4
END TYPE t_field
```

One of members of `t_field` are pointer to the array of `REAL(rstd)`, `INTEGER` or `LOGICAL`, and whose dimension is one, two or three. If the field is horizontal, such as surface pressure or sea surface temperature, `rval2d` is used. Note that horizontal index I and J are merged to one dimension.

Figure 1.2 shows horizontal indexing in *DYNAMICO*. As shown in previous chapter, *DYNAMICO* adopts icosahedral grid, and control volume is hexagonal as usual. One “patch” is rhomboid, and can be indexed as two-dimensional, each size are `iim` and `jjm`, as shown in left figure of Figure 1.2. These can be re-written as usual orthogonal i-j plane, shown in the right figure of Figure 1.2. The `n` point in the figure is surrounded by six neighbouring cells, named `right`, `rup`, etc. So stencil calculation comes from finite difference in horizontal uses seven points, not five as in usual orthogonal grid.

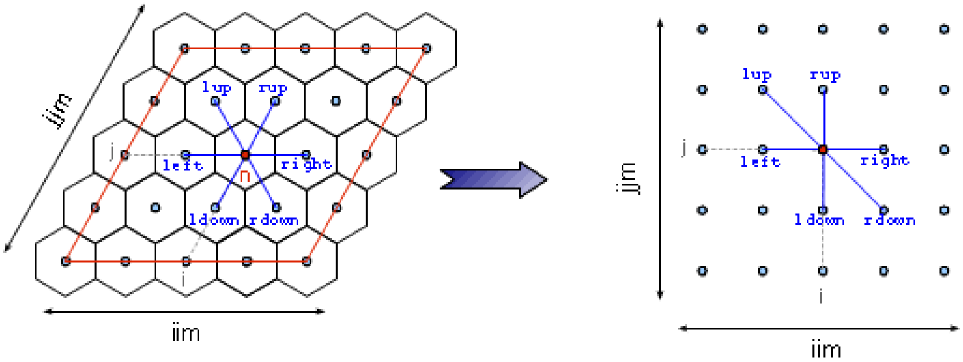


Figure 1.2: Horizontal indexing

The number of the edge point is three times larger than that of the center point. As shown in Figure 1.3, each center point manages three edge points. These points are named `u_right`, `u_lup`, and `u_ldown`.

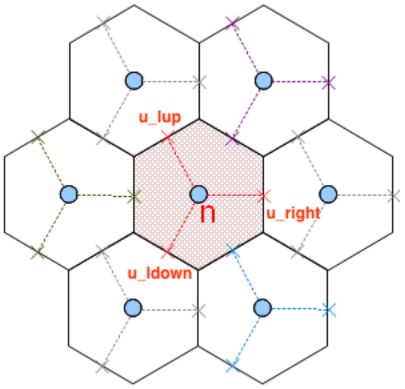


Figure 1.3: Relationship of center points and edge points

To allocate one `t_field` instance, subroutine `allocate_field` is called. Below is the example of allocating orography named “phis”.

```
 ! Time-independant orography
    CALL allocate_field(f_phis,field_t,type_real,name=’phis’)
```

1.6 Code structure

Global program structure of *DYNAMICO* is as follows.

In the main program, after the various initialization, time step loop is carried by a single subroutine `timeloop`. Figure 1.4 shows PAD (Problem Analysis Diagram)*2)of main processes in subroutine `timeloop`.

As seen in the top of this PAD,

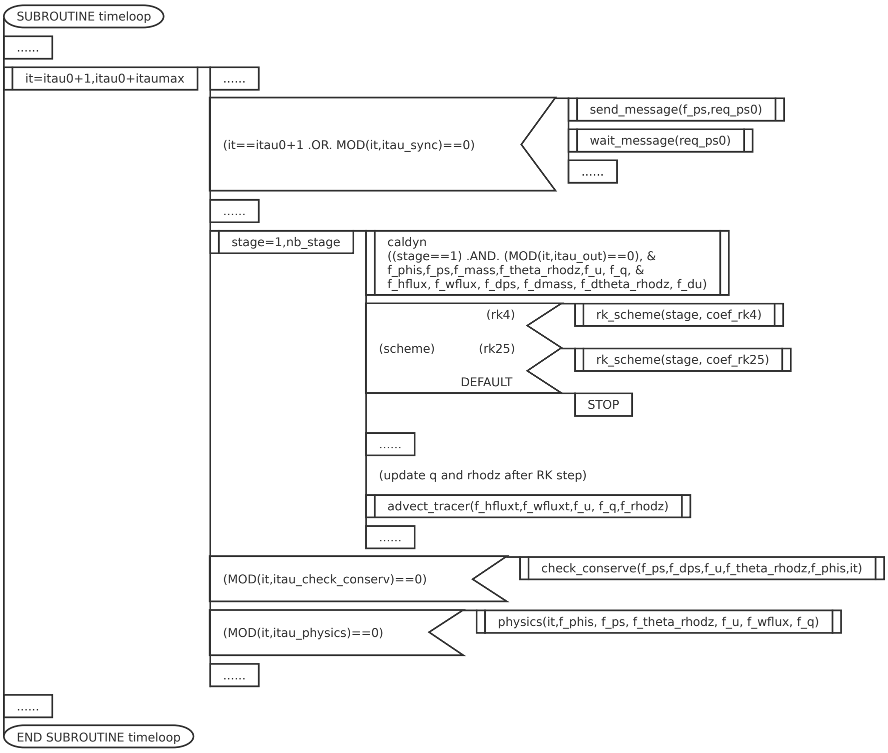

SUBROUTINE timeloop
......
it=itau0+1,itau0+itaumax
......
(it==itau0+1 .OR. MOD(it,itau_sync)==0)
send_message(f_ps,req_ps0)
wait_message(req_ps0)
......
......
stage=1,nb_stage
caldyn
((stage==1) .AND. (MOD(it,itau_out)==0), &
f_phis,f_ps,f_mass,f_theta_rhodz,f_u, f_q, &
f_hflux, f_wflux, f_dps, f_dmass, f_dtheta_rhodz, f_du)
(rk4)
rk_scheme(stage, coef_rk4)
(scheme)
(rk25)
rk_scheme(stage, coef_rk25)
DEFAULT
STOP
......
(update q and rhodz after RK step)
advect_tracer(f_hfluxt,f_wfluxt,f_u, f_q,f_rhodz)
......
(MOD(it,itau_check_conserv)==0)
check_conserve(f_ps,f_dps,f_u,f_theta_rhodz,f_phis,it)
(MOD(it,itau_physics)==0)
physics(it,f_phis, f_ps, f_theta_rhodz, f_u, f_wflux, f_q)
......
......
END SUBROUTINE timeloop

Figure 1.4: PAD of `timeloop`

Main time step loop is described in first `it` loop, from step `itau0` to `itau0+italmax`. First IF block in this loop is for halo exchange of several fields, using subroutine `send_message` and `wait_message`. In the next `stage` loop subroutine `caldyn`, one of `*_scheme` and `advect_tracer` are called sequentially. Subroutine `caldyn` calculate dynamical terms such as potential vorticity, etc. Subroutine `*_scheme` is for time-advancing. For example, `rk_scheme` uses Runge-Kutta scheme. This is a default scheme for this kernel package. Subroutine `advect_tracer` is to calculate advection of tracer quantities. Here variable `nb_stage` in the loop range is the number of iteration necessary for each time-advancing scheme. For example, `nb_stage=1` for Euler scheme, `nb_stage=4` for Runge-Kutta scheme. Finally, if time step `it` is at `itau_physics`'th step, subroutine `physics` is called to calculate physics part.

List 1.2 is the definition part of subroutine `caldyn`[3], and Figure 1.5 is the PAD of it. Note that all of current four kernel program in this package is taken from the subroutine called from this `caldyn` (See subsection 2.1.1). As mentioned in section 1.5, all of fields used in this subroutine is given as pointers of instance of `t_field`.

List 1.2: Definition part of `caldyn`

```
SUBROUTINE caldyn(write_out,f_phis, f_ps, f_mass, f_theta_rhodz, f_u, f_q, &
     f_hflux, f_wflux, f_dps, f_dmass, f_dtheta_rhodz, f_du)
  USE icosa
  USE disvert_mod, ONLY : caldyn_eta, eta_mass
  USE vorticity_mod
  USE kinetic_mod
  USE theta2theta_rhodz_mod
  USE wind_mod
  USE mpipara
  USE trace
```

[2] See Appendix A for reading PAD.

[3] Here is the version in `caldyn_gcm.f90`

```fortran
    USE omp_para
    USE output_field_mod
    USE checksum_mod
    IMPLICIT NONE
    LOGICAL,INTENT(IN)    :: write_out
    TYPE(t_field),POINTER :: f_phis(:)
    TYPE(t_field),POINTER :: f_ps(:)
    TYPE(t_field),POINTER :: f_mass(:)
    TYPE(t_field),POINTER :: f_theta_rhodz(:)
    TYPE(t_field),POINTER :: f_u(:)
    TYPE(t_field),POINTER :: f_q(:)
    TYPE(t_field),POINTER :: f_hflux(:), f_wflux(:)
    TYPE(t_field),POINTER :: f_dps(:)
    TYPE(t_field),POINTER :: f_dmass(:)
    TYPE(t_field),POINTER :: f_dtheta_rhodz(:)
    TYPE(t_field),POINTER :: f_du(:)

    REAL(rstd),POINTER :: ps(:), dps(:)
    REAL(rstd),POINTER :: mass(:,:), theta_rhodz(:,:), dtheta_rhodz(:,:)
    REAL(rstd),POINTER :: u(:,:), du(:,:), hflux(:,:), wflux(:,:)
    REAL(rstd),POINTER :: qu(:,:)
    REAL(rstd),POINTER :: qv(:,:)

! temporary shared variable
    REAL(rstd),POINTER  :: theta(:,:)
    REAL(rstd),POINTER  :: pk(:,:)
    REAL(rstd),POINTER  :: geopot(:,:)
    REAL(rstd),POINTER  :: convm(:,:)
    REAL(rstd),POINTER  :: wwuu(:,:)

    INTEGER :: ind
    LOGICAL,SAVE :: first=.TRUE.
```

In Figure 1.5 there are some assignment from `t_field` to real pointer, such as `ps = f_ps(ind)`. This is defined as module procedure and generic subroutine `get_val` and using interface assignment. All of them are defined in module `field_mod` in `field.f90`.

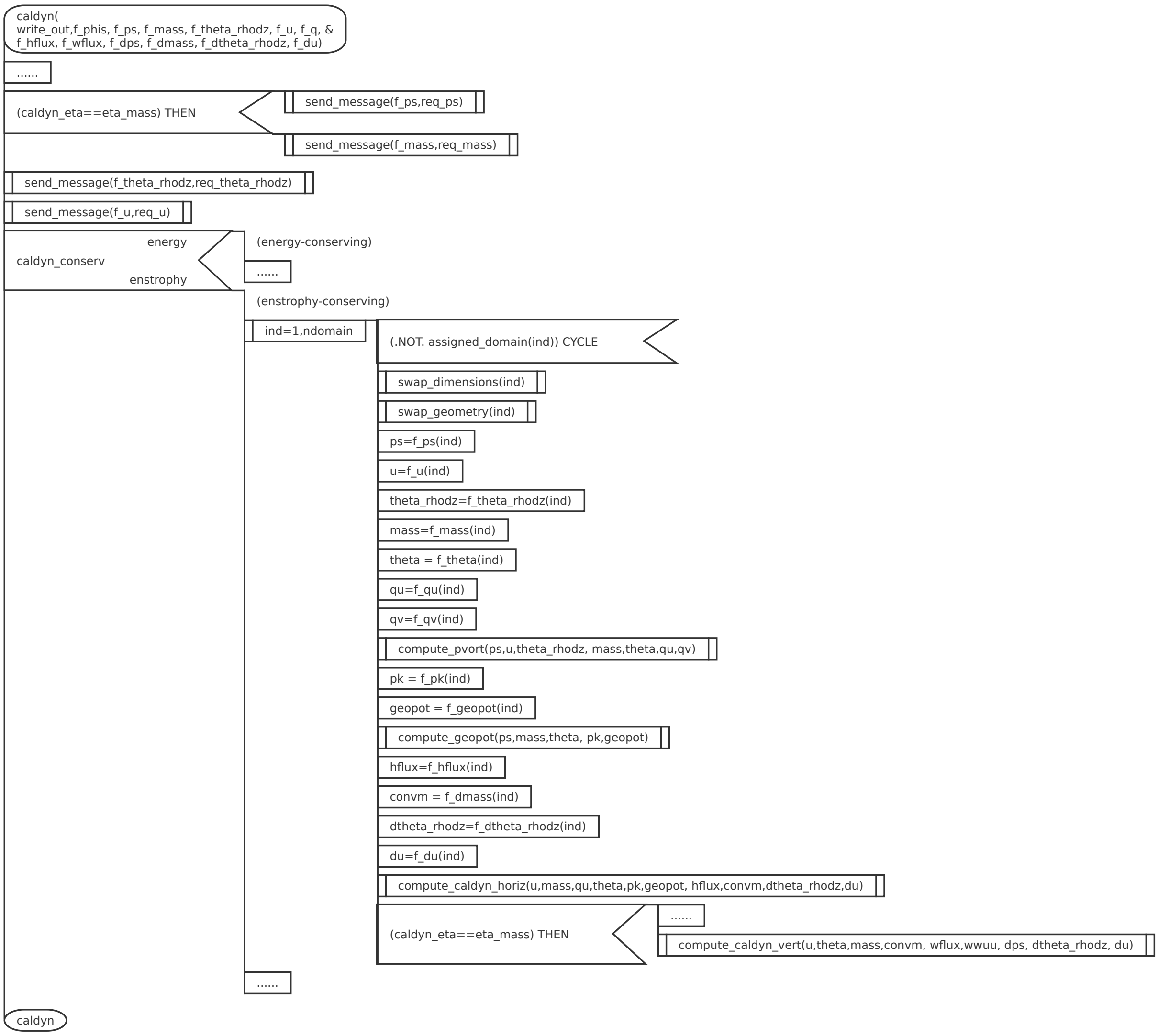

caldyn(
write_out,f_phis, f_ps, f_mass, f_theta_rhodz, f_u, f_q, &
f_hflux, f_wflux, f_dps, f_dmass, f_dtheta_rhodz, f_du)
......
(caldyn_eta==eta_mass) THEN
send_message(f_ps,req_ps)
send_message(f_mass,req_mass)
send_message(f_theta_rhodz,req_theta_rhodz)
send_message(f_u,req_u)
energy
caldyn_conserv
enstrophy
(energy-conserving)
......
(enstrophy-conserving)
ind=1,ndomain
(.NOT. assigned_domain(ind)) CYCLE
swap_dimensions(ind)
swap_geometry(ind)
ps=f_ps(ind)
u=f_u(ind)
theta_rhodz=f_theta_rhodz(ind)
mass=f_mass(ind)
theta = f_theta(ind)
qu=f_qu(ind)
qv=f_qv(ind)
compute_pvort(ps,u,theta_rhodz, mass,theta,qu,qv)
pk = f_pk(ind)
geopot = f_geopot(ind)
compute_geopot(ps,mass,theta, pk,geopot)
hflux=f_hflux(ind)
convm = f_dmass(ind)
dtheta_rhodz=f_dtheta_rhodz(ind)
du=f_du(ind)
compute_caldyn_horiz(u,mass,qu,theta,pk,geopot, hflux,convm,dtheta_rhodz,du)
(caldyn_eta==eta_mass) THEN
......
compute_caldyn_vert(u,theta,mass,convm, wflux,wwuu, dps, dtheta_rhodz, du)
......
caldyn

Figure 1.5: PAD of `caldyn`

Chapter 2

Description of each kernel

2.1 Overview and common stuff

2.1.1 Kernelize

Kernel programs in this package are as follows;

- `comp_pvort`
- `comp_geopot`
- `comp_caldyn_horiz`
- `comp_caldyn_vert`

All kernels are single subroutines in the original[*1)]*DYNAMICO*, and extracted and imposed to the wrapper for the kernel program Each subroutine has no modification, except using modules and some parameter settings.

All of input arrays and most of arrays/variables defined in various modules in the original model are read from input data file. Input data for each subroutine and reference (output) data are dumped from the execution of original *DYNAMICO*. Main routine of each kernel program reads these input and reference data, and call the subroutine with them as arguments for 1000 times, in current setting, then compare output values with reference data.

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

- min/max/sum of input data,
- min/max/sum of output data,
- min/max/sum of difference between output and validation data,
- computational time (elapsed time).

Elapsed time is measured using `omp_get_wtime()`.

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

2.1.2 MPI and OpenMP

While original *DYNAMICO* is parallelized by MPI and OpenMP, all kernel programs in this package are meant to be executed as one process with no threading.

Different from the *NICAM* kernel programss in this package, you don't need MPI library to compile/execute *DYNAMICO* kernel programs, but you need to make OpenMP enable in order to use `omp_get_wtime()`.

[*1)] Note that “original” here means “before kernelize”, since some bug fixes have made by AICS. Please contact the address shown on the back cover of this manual for details.

2.1.3 Mesuring environment

In the following sections, the example of performance result part of the log output file of each kernel program is shown. These were measured on the machine environment shown in Table 2.1, with setting export IAB_SYS=Ubuntu-gnu-ompi on compilation (See QuickStart.md).

Table 2.1: Measuring environment

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

2.2 comp_pvort

2.2.1 Description

Kernel comp_pvort is taken from the original subroutine compute_pvort in *DYNAMICO*. This subroutine is originally defined in module caldyn_gcm_mod. This module defines subroutine caldyn, which is the main subroutines for dynamics part of the model, and several sub-subroutines for various terms in the governing equation, such as potential vorticity, geopotential, etc. This subroutine calculates potential vorticity.

2.2.2 Discretization and code

List 2.1 shows the definition part of this subroutine and Figure 2.1 shows the PAD of this. Note that sources shown here are modified in the process of kernelization from the original distributed version.

List 2.1: Definition part of compute_pvort

```fortran
SUBROUTINE compute_pvort(ps,u,theta_rhodz, rhodz,theta,qu,qv)
  USE icosa
  USE disvert_mod, ONLY : mass_dak, mass_dbk, caldyn_eta, eta_mass
  USE exner_mod
  USE trace
  USE omp_para
  IMPLICIT NONE
  REAL(rstd),INTENT(IN)  :: u(iim*3*jjm,llm)
  REAL(rstd),INTENT(IN)  :: ps(iim*jjm)
  REAL(rstd),INTENT(IN)  :: theta_rhodz(iim*jjm,llm)
  REAL(rstd),INTENT(INOUT) :: rhodz(iim*jjm,llm)
  REAL(rstd),INTENT(INOUT) :: theta(iim*jjm,llm)
  REAL(rstd),INTENT(INOUT) :: qu(iim*3*jjm,llm)
  REAL(rstd),INTENT(INOUT) :: qv(iim*2*jjm,llm)

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

Here u, ps, theta_rhodz are velocity on the edge, surface pressure, and mass-weighted potential temperature, respectively. Output arrays rhodz, theta, qu, qv are mass, potential temperature, potential vorticity on the edge, and potential vorticity on the vertex, respectively. These arrays except ps have two dimensions, first one is for horizontal index and second one is for vertical index. Note that *DYNAMICO* adopts C-grid in horizontal, number of horizontal grid point for scalar, or number of control volume, in one domain is iim*jjm, but u and qu are defined on the edge of control volume, the size of the first dimension of them are iim*3*jjm. Also qv is defined on the vertex of control volume, the size of the first dimension of qv is iim*2*jjm.

The first section of this subroutine calculates theta in the double loop of ij and l. Note that in this kernel package, caldyn_eta is set as eta_mass, that means that vertical coordinate η uses Lagrangian vertical coordinate, rather than mass coordinate. In the second section, qv at two points, qv(ij+z_up,l)

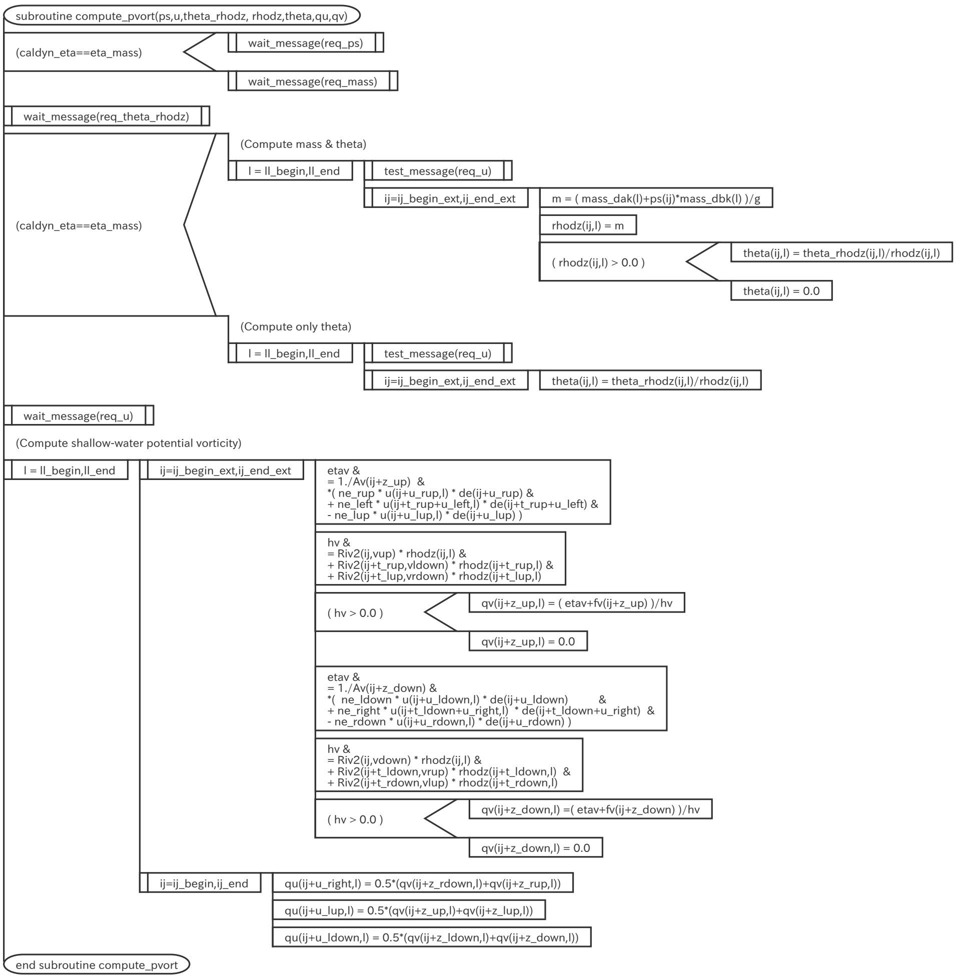

subroutine compute_pvort(ps,u,theta_rhodz, rhodz,theta,qu,qv)
(caldyn_eta==eta_mass)
wait_message(req_ps)
wait_message(req_mass)
wait_message(req_theta_rhodz)
(caldyn_eta==eta_mass)
(Compute mass & theta)
l = ll_begin,ll_end
test_message(req_u)
ij=ij_begin_ext,ij_end_ext
m = (mass_dak(l)+ps(ij)*mass_dbk(l))/g
rhodz(ij,l) = m
(rhodz(ij,l) > 0.0)
theta(ij,l) = theta_rhodz(ij,l)/rhodz(ij,l)
theta(ij,l) = 0.0
(Compute only theta)
l = ll_begin,ll_end
test_message(req_u)
ij=ij_begin_ext,ij_end_ext
theta(ij,l) = theta_rhodz(ij,l)/rhodz(ij,l)
wait_message(req_u)
(Compute shallow-water potential vorticity)
l = ll_begin,ll_end
ij=ij_begin_ext,ij_end_ext
etav &
= 1./Av(ij+z_up) &
*(ne_rup * u(ij+u_rup,l) * de(ij+u_rup) &
+ ne_left * u(ij+t_rup+u_left,l) * de(ij+t_rup+u_left) &
- ne_lup * u(ij+u_lup,l) * de(ij+u_lup))
hv &
= Riv2(ij,vup) * rhodz(ij,l) &
+ Riv2(ij+t_rup,vldown) * rhodz(ij+t_rup,l) &
+ Riv2(ij+t_lup,vrdown) * rhodz(ij+t_lup,l)
(hv > 0.0)
qv(ij+z_up,l) = (etav+fv(ij+z_up))/hv
qv(ij+z_up,l) = 0.0
etav &
= 1./Av(ij+z_down) &
*(ne_ldown * u(ij+u_ldown,l) * de(ij+u_ldown) &
+ ne_right * u(ij+t_ldown+u_right,l) * de(ij+t_ldown+u_right) &
- ne_rdown * u(ij+u_rdown,l) * de(ij+u_rdown))
hv &
= Riv2(ij,vdown) * rhodz(ij,l) &
+ Riv2(ij+t_ldown,vrup) * rhodz(ij+t_ldown,l) &
+ Riv2(ij+t_rdown,vlup) * rhodz(ij+t_rdown,l)
(hv > 0.0)
qv(ij+z_down,l) =(etav+fv(ij+z_down))/hv
qv(ij+z_down,l) = 0.0
ij=ij_begin,ij_end
qu(ij+u_right,l) = 0.5*(qv(ij+z_rdown,l)+qv(ij+z_rup,l))
qu(ij+u_lup,l) = 0.5*(qv(ij+z_up,l)+qv(ij+z_lup,l))
qu(ij+u_ldown,l) = 0.5*(qv(ij+z_ldown,l)+qv(ij+z_down,l))
end subroutine compute_pvort

Figure 2.1: PAD of `compute_pvort`

and qv(ij+z_down,l), are calculated in the first double loop, then qu at three pounts, qu(ij+u_right,l), qu(ij+u_lup,l) and qu(ij+l_down,l) are calculated.

2.2.3 Input data and result

Input data file is prepared and you can download from official server using `data/download.sh` script. This data file is created by original *DYNAMICO*[*2)]with Held-Suarez case parameter set included in the original source archive. Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of `$IAB_SYS=Ubuntu-gnu-ompi` case.

```
[KERNEL] comp_pvort
*** Start  initialize
               iim, jjm, llm:    23    25    19
            ij_begin, ij_end:    48   528
    ij_begin_ext, ij_end_ext:    24   552
            ll_begin, ll_end:     1    19
       t_right, t_rup, t_lup:     1    23    22
    t_left, t_ldown, t_rdown:    -1   -23   -22
       u_right, u_rup, u_lup:     0  1173   575
    u_left, u_ldown, u_rdown:    -1  1150   553
          z_rup, z_up, z_lup:   598     0   597
    z_ldown, z_down, z_rdown:   -23   575   -22
                  caldyn_eta:     1
                           g:     9.80000000
+check[Av              ] max=  4.1228713627140027E+11,min=  0.0000000000000000E+00,sum=  3.2428753277257527E+13
+check[de              ] max=  4.5171816240714993E+06,min=  0.0000000000000000E+00,sum=  4.7785815753077912E+08
+check[Riv2            ] max=  3.8193271158709069E-01,min=  0.0000000000000000E+00,sum=  9.6499999999999977E+02
+check[fv              ] max=  8.2383275804860789E-05,min= -6.6879410680009186E-05,sum=  4.1115175112148659E-03
+check[mass_dak        ] max=  3.9864758943842335E+03,min= -4.2200064724847380E+03,sum=  1.1368683772161603E-13
+check[mass_dbk        ] max=  1.6280745944237918E-01,min=  0.0000000000000000E+00,sum=  1.0000000000000000E+00
*** Finish initialize
*** Start kernel
### check point iteration:        1000
### Input ###
+check[u               ] max=  4.1278968179782127E-01,min= -4.1278968179782127E-01,sum=  1.6791131703073393E+01
+check[ps              ] max=  1.0000000000000000E+05,min=  1.0000000000000000E+05,sum=  5.7500000000000000E+07
+check[theta_rhodz     ] max=  3.9393370687019045E+05,min=  0.0000000000000000E+00,sum=  1.8099621340626464E+09
+check[theta_prev      ] max=  8.0139914420291746E+02,min=  0.0000000000000000E+00,sum=  3.8582633571973117E+06
+check[rhodz_prev      ] max=  1.2306877011993038E+03,min=  0.0000000000000000E+00,sum=  5.3979591836733194E+06
+check[qu_prev         ] max=  1.0339537867296609E-06,min= -8.4408169682701225E-07,sum=  3.9419811615778674E-04
+check[qv_prev         ] max=  1.0397552030841796E-06,min= -8.4408169685381862E-07,sum=  2.6984926372303133E-04
### Output ###
+check[theta           ] max=  8.0139914420291746E+02,min=  0.0000000000000000E+00,sum=  3.8582633571973117E+06
+check[rhodz           ] max=  1.2306877011993038E+03,min=  0.0000000000000000E+00,sum=  5.3979591836733194E+06
+check[qu              ] max=  1.0626772908333491E-06,min= -8.5290650439776975E-07,sum=  3.9864842567446446E-04
+check[qv              ] max=  1.0855993800007362E-06,min= -8.9078811385791910E-07,sum=  2.7461310165802981E-04
### final iteration:        1000
### Validation : grid-by-grid diff ###
+check[theta           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[rhodz           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[qu              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[qv              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel
```

Check the lines below ‘‘`Validation :  grid-by-grid diff`’’ line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable. There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.2.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in subsection 2.1.3. Note that in this program kernel part is iterated 1000 times.

```
*** Computational Time Report
*** ID=001 : MAIN_comp_pvort                  T=     0.248 N=   1000
```

[*2)] with slight modification by AICS.

2.3 comp_geopot

2.3.1 Description

Kernel comp_geopot is taken from the original subroutine compute_geopot in *DYNAMICO*. This subroutine is originally defined in module caldyn_gcm_mod. This module defines subroutine caldyn, which is the main subroutines for dynamics part of the model, and several sub-subroutines for various terms in the governing equation, such as potential vorticity, geopotential, etc. This subroutine calculates geopotential.

2.3.2 Discretization and code

List 2.2 shows the definition part of this subroutine, and Figure 2.2 shows the PAD of this.

List 2.2: Definition part of compute_geopot

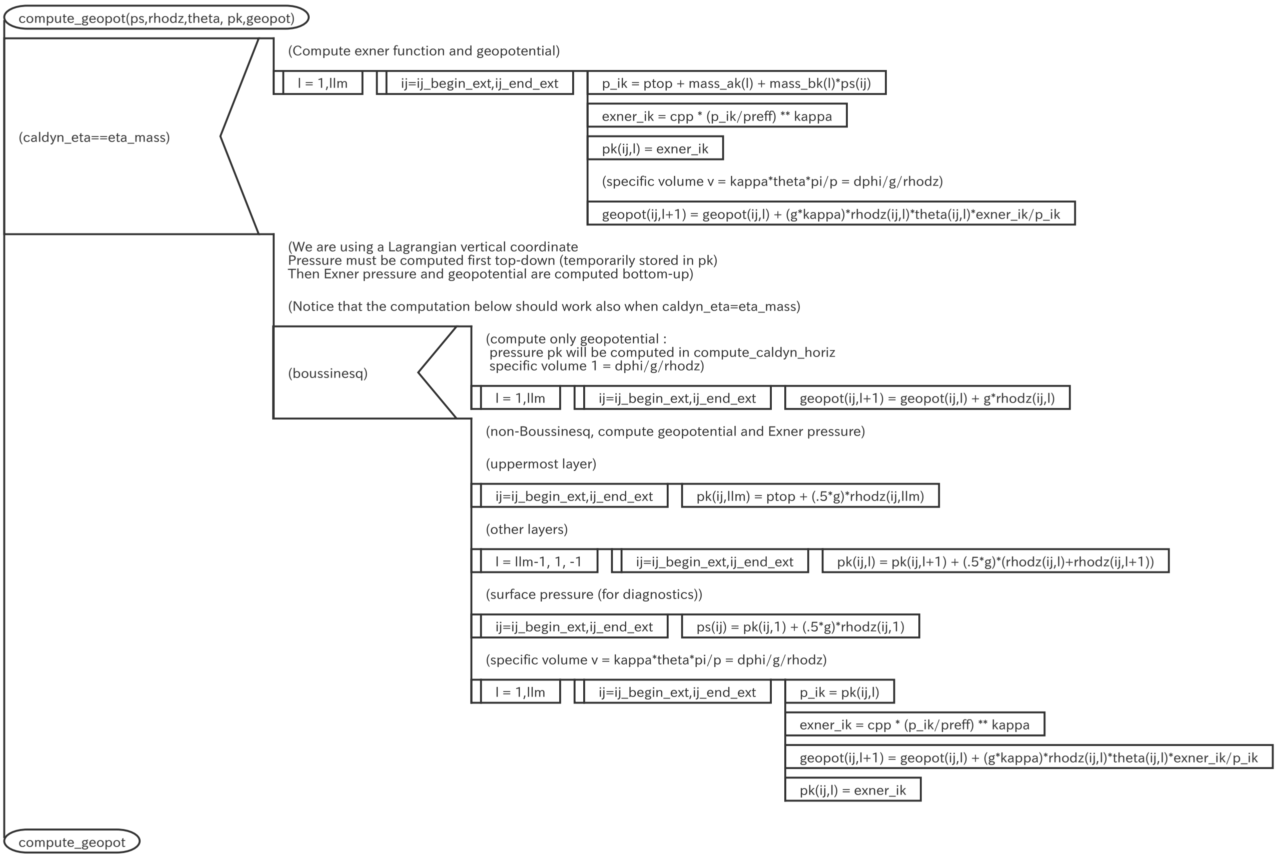
```
SUBROUTINE compute_geopot(ps,rhodz,theta, pk,geopot)
USE icosa
USE disvert_mod
USE exner_mod
USE trace
USE omp_para
IMPLICIT NONE
  REAL(rstd),INTENT(INOUT) :: ps(iim*jjm)
  REAL(rstd),INTENT(IN)    :: rhodz(iim*jjm,llm)
  REAL(rstd),INTENT(IN)    :: theta(iim*jjm,llm)    ! potential temperature
  REAL(rstd),INTENT(INOUT) :: pk(iim*jjm,llm)       ! Exner function
  REAL(rstd),INTENT(INOUT) :: geopot(iim*jjm,llm+1) ! geopotential

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

Where ps, rhodz, theta, pk, and geopot are surface pressure, mass, potential temperature, Exner function, and geopotential, respectively. These arrays except ps have two dimensions, first one is for horizontal index and second one is for vertical index. All of these are defined in the center of control volume in horizontal, the size of first dimension is iim*jjm. Also these except ps and geopot are defined in the full level in vertical, the size of second dimension of these are llm, while geopot has the size of llm+1.

Note that in this kernel package caldyn_eta is set as eta_mass, and boussinesq is set as .true., so in this subroutine only geopot is calculated as

```
geopot(ij,l+1) = geopot(ij,l) + g*rhodz(ij,l)
```

and Exner pressure are calculated in subroutine compulte_caldyn_horiz, which is also included in this package as kernel comp_caldyn_horiz.



```
compute_geopot(ps,rhodz,theta, pk,geopot)
  (caldyn_eta==eta_mass)
    (Compute exner function and geopotential)
    l = 1,llm
      ij=ij_begin_ext,ij_end_ext
        p_ik = ptop + mass_ak(l) + mass_bk(l)*ps(ij)
        exner_ik = cpp * (p_ik/preff) ** kappa
        pk(ij,l) = exner_ik
        (specific volume v = kappa*theta*pi/p = dphi/g/rhodz)
        geopot(ij,l+1) = geopot(ij,l) + (g*kappa)*rhodz(ij,l)*theta(ij,l)*exner_ik/p_ik
  else
    (We are using a Lagrangian vertical coordinate
    Pressure must be computed first top-down (temporarily stored in pk)
    Then Exner pressure and geopotential are computed bottom-up)
    (Notice that the computation below should work also when caldyn_eta=eta_mass)
    (boussinesq)
      (compute only geopotential :
       pressure pk will be computed in compute_caldyn_horiz
       specific volume 1 = dphi/g/rhodz)
      l = 1,llm
        ij=ij_begin_ext,ij_end_ext
          geopot(ij,l+1) = geopot(ij,l) + g*rhodz(ij,l)
    else
      (non-Boussinesq, compute geopotential and Exner pressure)
      (uppermost layer)
      ij=ij_begin_ext,ij_end_ext
        pk(ij,llm) = ptop + (.5*g)*rhodz(ij,llm)
      (other layers)
      l = llm-1, 1, -1
        ij=ij_begin_ext,ij_end_ext
          pk(ij,l) = pk(ij,l+1) + (.5*g)*(rhodz(ij,l)+rhodz(ij,l+1))
      (surface pressure (for diagnostics))
      ij=ij_begin_ext,ij_end_ext
        ps(ij) = pk(ij,1) + (.5*g)*rhodz(ij,1)
      (specific volume v = kappa*theta*pi/p = dphi/g/rhodz)
      l = 1,llm
        ij=ij_begin_ext,ij_end_ext
          p_ik = pk(ij,l)
          exner_ik = cpp * (p_ik/preff) ** kappa
          geopot(ij,l+1) = geopot(ij,l) + (g*kappa)*rhodz(ij,l)*theta(ij,l)*exner_ik/p_ik
          pk(ij,l) = exner_ik
compute_geopot
```

Figure 2.2: PAD of `compute_geopot`

2.3.3 Input data and result

Input data file is prepared and you can download from official server using `data/download.sh` script. This data file is created by original *DYNAMICO*[*3)]with Held-Suarez case parameter set included in the original source archive. Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of `$IAB_SYS=Ubuntu-gnu-ompi` case.

```
[KERNEL] comp_geopot
*** Start  initialize
               iim, jjm, llm:    23    25    19
            ij_begin, ij_end:    48   528
    ij_begin_ext, ij_end_ext:    24   552
            ll_begin, ll_end:     1    19
       t_right, t_rup, t_lup:     1    23    22
    t_left, t_ldown, t_rdown:    -1   -23   -22
       u_right, u_rup, u_lup:     0  1173   575
    u_left, u_ldown, u_rdown:    -1  1150   553
          z_rup, z_up, z_lup:   598     0   597
    z_ldown, z_down, z_rdown:   -23   575   -22
                  caldyn_eta:     1
                  boussinesq:     F
                           g:     9.80000000
+check[mass_ak         ] max=  2.2205608555404415E+04,min=  2.9655441593806341E+02,sum=  1.7315769223740909E+05
+check[mass_bk         ] max=  9.8820601234384886E-01,min=  0.0000000000000000E+00,sum=  6.4254510678414123E+00
*** Finish initialize
*** Start kernel
### check point iteration:        1000
### Input ###
+check[ps_prev         ] max=  1.0000000000000000E+05,min=  1.0000000000000000E+05,sum=  5.7500000000000000E+07
+check[rhodz           ] max=  1.2306877011993038E+03,min=  0.0000000000000000E+00,sum=  5.3979591836733194E+06
+check[theta           ] max=  8.0139914420291746E+02,min=  0.0000000000000000E+00,sum=  3.8582633571973117E+06
+check[pk_prev         ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[geopot_prev     ] max=  3.8250620498369227E+05,min=  0.0000000000000000E+00,sum=  1.1718001851963627E+09
### Output ###
+check[ps              ] max=  1.0000000000000000E+05,min=  1.0000000000000000E+05,sum=  5.7500000000000000E+07
+check[pk              ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[geopot          ] max=  3.8250620498369227E+05,min=  0.0000000000000000E+00,sum=  1.1718001851963627E+09
### final iteration:        1000
### Validation : grid-by-grid diff ###
+check[ps              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[pk              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[geopot          ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel
```

Check the lines below “`Validation :  grid-by-grid diff`” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable. There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.3.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in subsection 2.1.3. Note that in this program kernel part is iterated 1000 times.

```
*** Computational Time Report
*** ID=001 : MAIN_comp_geopot                 T=     0.824 N=   1000
```

2.4 `comp_caldyn_horiz`

2.4.1 Description

Kernel `comp_caldyn_horiz` is taken from the original subroutine `compute_caldyn_horiz` in *DYNAMICO*. This subroutine is originally defined in module `caldyn_gcm_mod`. This module defines subroutine `caldyn`,

[*3)] with slight modification by AICS.

which is the main subroutines for dynamics part of the model, and several sub-subroutines for various terms in the governing equation, such as potential vorticity, geopotential, etc. This subroutine calculates several horizontal terms, including mass flux, Bernouilli term, etc.

2.4.2 Discretization and code

List 2.3 shows the definition part of this subroutne, and Figure 2.3, 2.4, 2.5 show the PAD of this.

List 2.3: Definition part of `compute_caldyn_horiz`

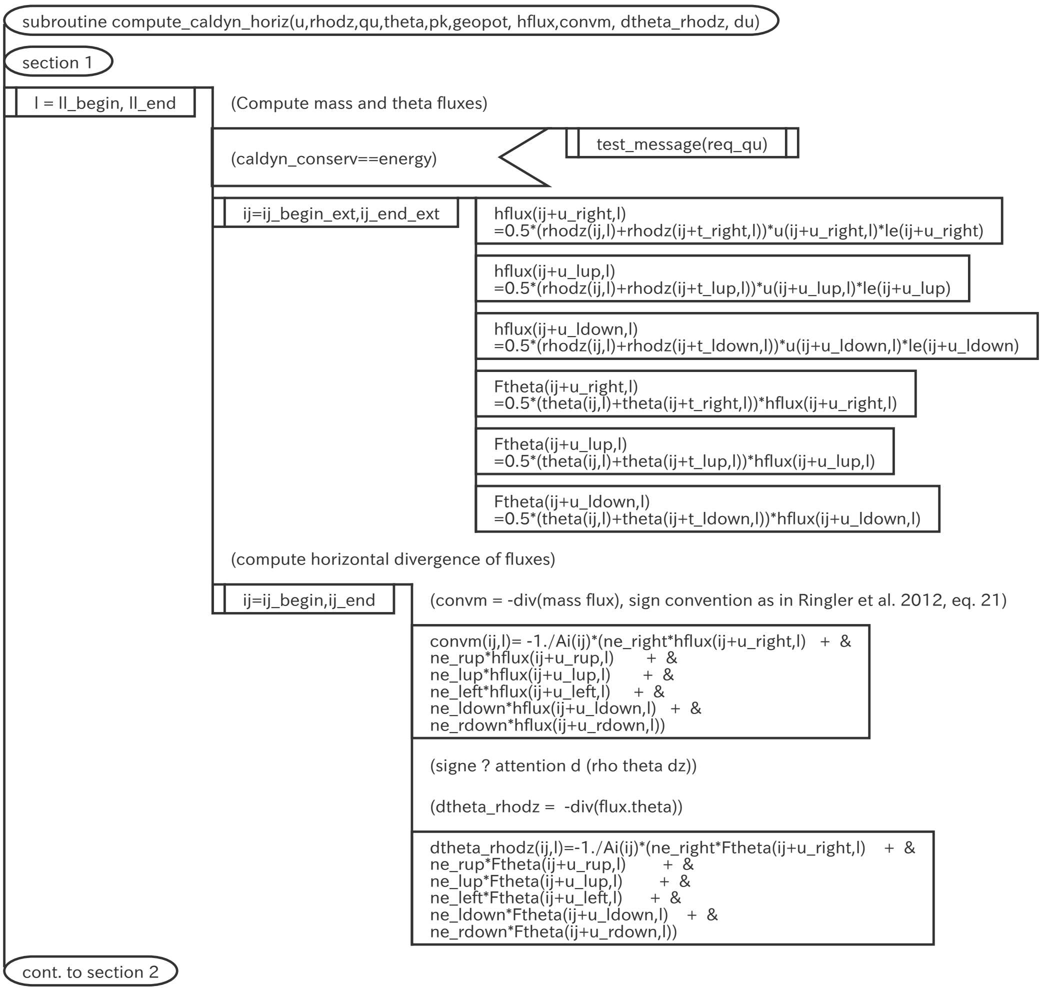
```fortran
SUBROUTINE compute_caldyn_horiz(u,rhodz,qu,theta,pk,geopot, hflux,convm, dtheta_rhodz, du)
USE icosa
USE disvert_mod
USE exner_mod
USE trace
USE omp_para
IMPLICIT NONE
  REAL(rstd),INTENT(IN)  :: u(iim*3*jjm,llm)    ! prognostic "velocity"
  REAL(rstd),INTENT(IN)  :: rhodz(iim*jjm,llm)
  REAL(rstd),INTENT(IN)  :: qu(iim*3*jjm,llm)
  REAL(rstd),INTENT(IN)  :: theta(iim*jjm,llm)  ! potential temperature
  REAL(rstd),INTENT(INOUT) :: pk(iim*jjm,llm) ! Exner function
  REAL(rstd),INTENT(IN)  :: geopot(iim*jjm,llm+1)    ! geopotential

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

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

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

Where `u`, `rhodz`, `qu`, `geopot` are wind velocity on the edge, mass, potential vorticity on the edge, and geopotential, respectively. `pk`, `hflux`, `convm`, `dtheta_rhodz`, `du` are Exner function, horizontal mass flux on the edge, mass flux convergence, time derivative of the mass-weighted potential temperature, and time derivative of wind velocity on the edge. Local arrays `cor_NT`, `urel`, `Ftheta`, `berni` are Coriolis's force, relative velocity, potential temperature flux and Bernoulli function, respectively. All of these arrays are two dimensional. First dimension is for horizontal index, and the size depends on the point where the variable is defined, since *DYNAMICO* adopts C-grid. Second dimension is for vertical index, and the size is `llm`, except `llm+1` for `geopot` that is defined on the half level in vertical, while others are defined on the full level.

This subroutine is relatively long, and is able to be split by three sections. In the first section (Figure 2.3), there is one l-loop and two ij-loop in it. The first one calculates mass flux `hflux` and potential temperature flux `Ftheta` at the edge of each control volume. The second loop calculates convergence of mass flux `convm` and convergence of potential temperature flux `dtheta_rhodz`.

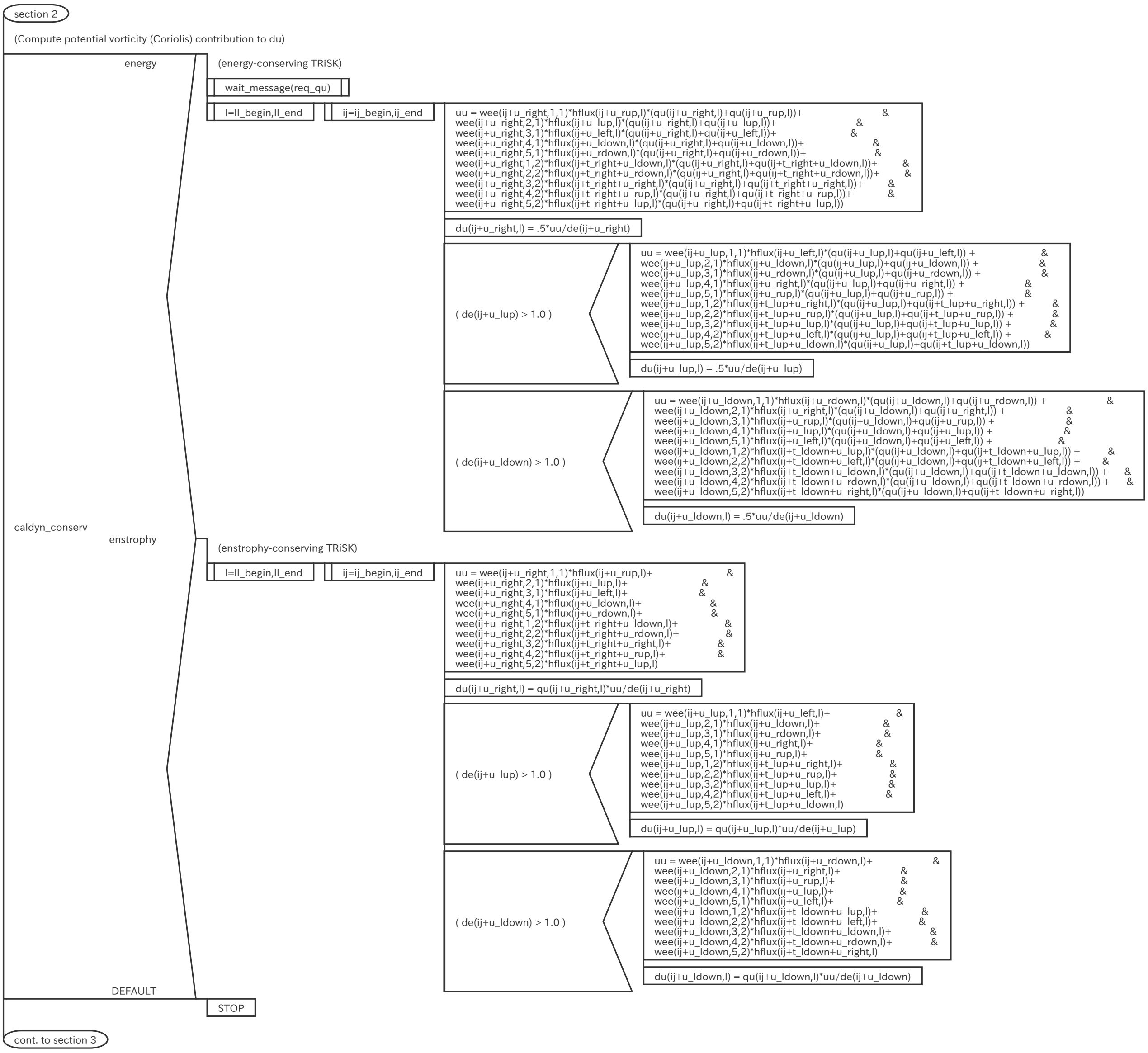
The second section (Figure 2.4) calculates potential vorticity contribution to `du` based on the TRiSK scheme (Ringler et al., 2010). Here `wee` is interpolating weight, prepared in module `geometry` in original *DYNAMICO*. Note that since `caldyn_conserv` is set as `energy` in this kernel program, second choice of CASE is selected.

The last section (Figure 2.5) calculates Bernoulli term first, then adds gradients of it and Extern functions to `du`. Here Bernoulli term is sum of kinetic energy and geopotential. Note that `boussinesq` is set as `.true.` in this kernel package, Exner function `pk` is calculated in advance to calculate Bernoulli term `berni`.



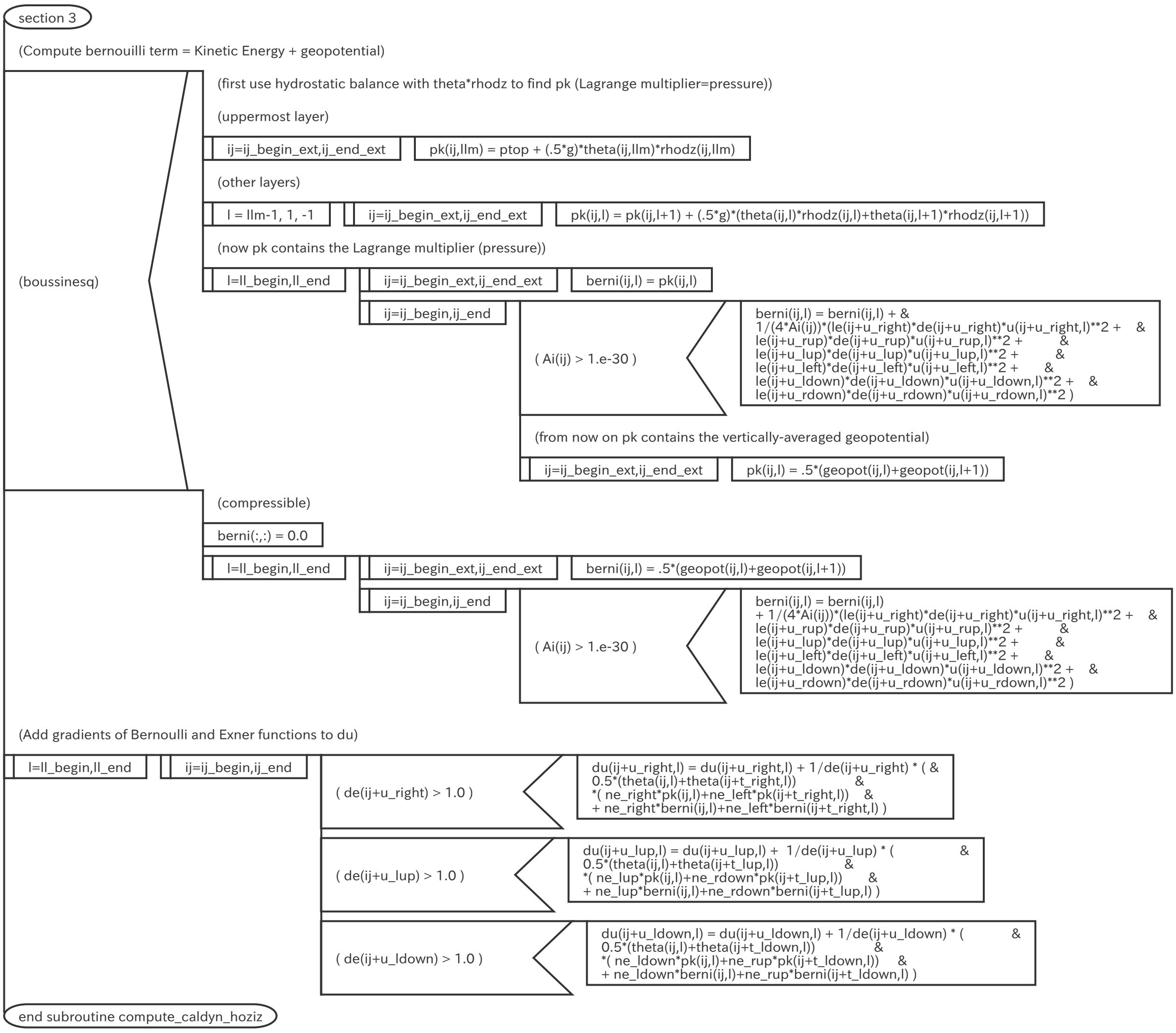
```
subroutine compute_caldyn_horiz(u,rhodz,qu,theta,pk,geopot, hflux,convm, dtheta_rhodz, du)
section 1
l = ll_begin, ll_end
    (Compute mass and theta fluxes)
    (caldyn_conserv==energy)
        test_message(req_qu)
    ij=ij_begin_ext,ij_end_ext
        hflux(ij+u_right,l)
        =0.5*(rhodz(ij,l)+rhodz(ij+t_right,l))*u(ij+u_right,l)*le(ij+u_right)
        hflux(ij+u_lup,l)
        =0.5*(rhodz(ij,l)+rhodz(ij+t_lup,l))*u(ij+u_lup,l)*le(ij+u_lup)
        hflux(ij+u_ldown,l)
        =0.5*(rhodz(ij,l)+rhodz(ij+t_ldown,l))*u(ij+u_ldown,l)*le(ij+u_ldown)
        Ftheta(ij+u_right,l)
        =0.5*(theta(ij,l)+theta(ij+t_right,l))*hflux(ij+u_right,l)
        Ftheta(ij+u_lup,l)
        =0.5*(theta(ij,l)+theta(ij+t_lup,l))*hflux(ij+u_lup,l)
        Ftheta(ij+u_ldown,l)
        =0.5*(theta(ij,l)+theta(ij+t_ldown,l))*hflux(ij+u_ldown,l)
    (compute horizontal divergence of fluxes)
    ij=ij_begin,ij_end
        (convm = -div(mass flux), sign convention as in Ringler et al. 2012, eq. 21)
        convm(ij,l)= -1./Ai(ij)*(ne_right*hflux(ij+u_right,l)   +  &
        ne_rup*hflux(ij+u_rup,l)       +  &
        ne_lup*hflux(ij+u_lup,l)       +  &
        ne_left*hflux(ij+u_left,l)     +  &
        ne_ldown*hflux(ij+u_ldown,l)   +  &
        ne_rdown*hflux(ij+u_rdown,l))
        (signe ? attention d (rho theta dz))
        (dtheta_rhodz =  -div(flux.theta))
        dtheta_rhodz(ij,l)=-1./Ai(ij)*(ne_right*Ftheta(ij+u_right,l)    +  &
        ne_rup*Ftheta(ij+u_rup,l)        +  &
        ne_lup*Ftheta(ij+u_lup,l)        +  &
        ne_left*Ftheta(ij+u_left,l)      +  &
        ne_ldown*Ftheta(ij+u_ldown,l)    +  &
        ne_rdown*Ftheta(ij+u_rdown,l))
cont. to section 2
```

Figure 2.3: PAD of `compute_caldyn_horiz(1)`



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

du(ij+u_right,l) = .5*uu/de(ij+u_right)

uu = wee(ij+u_lup,1,1)*hflux(ij+u_left,l)*(qu(ij+u_lup,l)+qu(ij+u_left,l)) +                &
wee(ij+u_lup,2,1)*hflux(ij+u_ldown,l)*(qu(ij+u_lup,l)+qu(ij+u_ldown,l)) +                  &
wee(ij+u_lup,3,1)*hflux(ij+u_rdown,l)*(qu(ij+u_lup,l)+qu(ij+u_rdown,l)) +                  &
wee(ij+u_lup,4,1)*hflux(ij+u_right,l)*(qu(ij+u_lup,l)+qu(ij+u_right,l)) +                &
wee(ij+u_lup,5,1)*hflux(ij+u_rup,l)*(qu(ij+u_lup,l)+qu(ij+u_rup,l)) +                    &
wee(ij+u_lup,1,2)*hflux(ij+t_lup+u_right,l)*(qu(ij+u_lup,l)+qu(ij+t_lup+u_right,l)) +      &
wee(ij+u_lup,2,2)*hflux(ij+t_lup+u_rup,l)*(qu(ij+u_lup,l)+qu(ij+t_lup+u_rup,l)) +          &
wee(ij+u_lup,3,2)*hflux(ij+t_lup+u_lup,l)*(qu(ij+u_lup,l)+qu(ij+t_lup+u_lup,l)) +          &
wee(ij+u_lup,4,2)*hflux(ij+t_lup+u_left,l)*(qu(ij+u_lup,l)+qu(ij+t_lup+u_left,l)) +      &
wee(ij+u_lup,5,2)*hflux(ij+t_lup+u_ldown,l)*(qu(ij+u_lup,l)+qu(ij+t_lup+u_ldown,l))

du(ij+u_lup,l) = .5*uu/de(ij+u_lup)

uu = wee(ij+u_ldown,1,1)*hflux(ij+u_rdown,l)*(qu(ij+u_ldown,l)+qu(ij+u_rdown,l)) +            &
wee(ij+u_ldown,2,1)*hflux(ij+u_right,l)*(qu(ij+u_ldown,l)+qu(ij+u_right,l)) +               &
wee(ij+u_ldown,3,1)*hflux(ij+u_rup,l)*(qu(ij+u_ldown,l)+qu(ij+u_rup,l)) +                   &
wee(ij+u_ldown,4,1)*hflux(ij+u_lup,l)*(qu(ij+u_ldown,l)+qu(ij+u_lup,l)) +                   &
wee(ij+u_ldown,5,1)*hflux(ij+u_left,l)*(qu(ij+u_ldown,l)+qu(ij+u_left,l)) +                &
wee(ij+u_ldown,1,2)*hflux(ij+t_ldown+u_lup,l)*(qu(ij+u_ldown,l)+qu(ij+t_ldown+u_lup,l)) +      &
wee(ij+u_ldown,2,2)*hflux(ij+t_ldown+u_left,l)*(qu(ij+u_ldown,l)+qu(ij+t_ldown+u_left,l)) +    &
wee(ij+u_ldown,3,2)*hflux(ij+t_ldown+u_ldown,l)*(qu(ij+u_ldown,l)+qu(ij+t_ldown+u_ldown,l)) +   &
wee(ij+u_ldown,4,2)*hflux(ij+t_ldown+u_rdown,l)*(qu(ij+u_ldown,l)+qu(ij+t_ldown+u_rdown,l)) +   &
wee(ij+u_ldown,5,2)*hflux(ij+t_ldown+u_right,l)*(qu(ij+u_ldown,l)+qu(ij+t_ldown+u_right,l))

du(ij+u_ldown,l) = .5*uu/de(ij+u_ldown)

uu = wee(ij+u_right,1,1)*hflux(ij+u_rup,l)+                 &
wee(ij+u_right,2,1)*hflux(ij+u_lup,l)+              &
wee(ij+u_right,3,1)*hflux(ij+u_left,l)+             &
wee(ij+u_right,4,1)*hflux(ij+u_ldown,l)+              &
wee(ij+u_right,5,1)*hflux(ij+u_rdown,l)+              &
wee(ij+u_right,1,2)*hflux(ij+t_right+u_ldown,l)+        &
wee(ij+u_right,2,2)*hflux(ij+t_right+u_rdown,l)+        &
wee(ij+u_right,3,2)*hflux(ij+t_right+u_right,l)+        &
wee(ij+u_right,4,2)*hflux(ij+t_right+u_rup,l)+          &
wee(ij+u_right,5,2)*hflux(ij+t_right+u_lup,l)

du(ij+u_right,l) = qu(ij+u_right,l)*uu/de(ij+u_right)

uu = wee(ij+u_lup,1,1)*hflux(ij+u_left,l)+                 &
wee(ij+u_lup,2,1)*hflux(ij+u_ldown,l)+              &
wee(ij+u_lup,3,1)*hflux(ij+u_rdown,l)+              &
wee(ij+u_lup,4,1)*hflux(ij+u_right,l)+            &
wee(ij+u_lup,5,1)*hflux(ij+u_rup,l)+              &
wee(ij+u_lup,1,2)*hflux(ij+t_lup+u_right,l)+        &
wee(ij+u_lup,2,2)*hflux(ij+t_lup+u_rup,l)+          &
wee(ij+u_lup,3,2)*hflux(ij+t_lup+u_lup,l)+          &
wee(ij+u_lup,4,2)*hflux(ij+t_lup+u_left,l)+        &
wee(ij+u_lup,5,2)*hflux(ij+t_lup+u_ldown,l)

du(ij+u_lup,l) = qu(ij+u_lup,l)*uu/de(ij+u_lup)

uu = wee(ij+u_ldown,1,1)*hflux(ij+u_rdown,l)+                &
wee(ij+u_ldown,2,1)*hflux(ij+u_right,l)+             &
wee(ij+u_ldown,3,1)*hflux(ij+u_rup,l)+               &
wee(ij+u_ldown,4,1)*hflux(ij+u_lup,l)+               &
wee(ij+u_ldown,5,1)*hflux(ij+u_left,l)+             &
wee(ij+u_ldown,1,2)*hflux(ij+t_ldown+u_lup,l)+        &
wee(ij+u_ldown,2,2)*hflux(ij+t_ldown+u_left,l)+      &
wee(ij+u_ldown,3,2)*hflux(ij+t_ldown+u_ldown,l)+      &
wee(ij+u_ldown,4,2)*hflux(ij+t_ldown+u_rdown,l)+      &
wee(ij+u_ldown,5,2)*hflux(ij+t_ldown+u_right,l)

du(ij+u_ldown,l) = qu(ij+u_ldown,l)*uu/de(ij+u_ldown)
```

Figure 2.4: PAD of `compute_caldyn_horiz(2)`



```
section 3
(Compute bernouilli term = Kinetic Energy + geopotential)
(boussinesq)
    (first use hydrostatic balance with theta*rhodz to find pk (Lagrange multiplier=pressure))
    (uppermost layer)
    ij=ij_begin_ext,ij_end_ext
        pk(ij,llm) = ptop + (.5*g)*theta(ij,llm)*rhodz(ij,llm)
    (other layers)
    l = llm-1, 1, -1
        ij=ij_begin_ext,ij_end_ext
            pk(ij,l) = pk(ij,l+1) + (.5*g)*(theta(ij,l)*rhodz(ij,l)+theta(ij,l+1)*rhodz(ij,l+1))
    (now pk contains the Lagrange multiplier (pressure))
    l=ll_begin,ll_end
        ij=ij_begin_ext,ij_end_ext
            berni(ij,l) = pk(ij,l)
        ij=ij_begin,ij_end
            ( Ai(ij) > 1.e-30 )
                berni(ij,l) = berni(ij,l) + &
                1/(4*Ai(ij))*(le(ij+u_right)*de(ij+u_right)*u(ij+u_right,l)**2 +    &
                le(ij+u_rup)*de(ij+u_rup)*u(ij+u_rup,l)**2 +          &
                le(ij+u_lup)*de(ij+u_lup)*u(ij+u_lup,l)**2 +          &
                le(ij+u_left)*de(ij+u_left)*u(ij+u_left,l)**2 +       &
                le(ij+u_ldown)*de(ij+u_ldown)*u(ij+u_ldown,l)**2 +    &
                le(ij+u_rdown)*de(ij+u_rdown)*u(ij+u_rdown,l)**2 )
            (from now on pk contains the vertically-averaged geopotential)
            ij=ij_begin_ext,ij_end_ext
                pk(ij,l) = .5*(geopot(ij,l)+geopot(ij,l+1))
(compressible)
    berni(:,:) = 0.0
    l=ll_begin,ll_end
        ij=ij_begin_ext,ij_end_ext
            berni(ij,l) = .5*(geopot(ij,l)+geopot(ij,l+1))
        ij=ij_begin,ij_end
            ( Ai(ij) > 1.e-30 )
                berni(ij,l) = berni(ij,l)
                + 1/(4*Ai(ij))*(le(ij+u_right)*de(ij+u_right)*u(ij+u_right,l)**2 +    &
                le(ij+u_rup)*de(ij+u_rup)*u(ij+u_rup,l)**2 +          &
                le(ij+u_lup)*de(ij+u_lup)*u(ij+u_lup,l)**2 +          &
                le(ij+u_left)*de(ij+u_left)*u(ij+u_left,l)**2 +       &
                le(ij+u_ldown)*de(ij+u_ldown)*u(ij+u_ldown,l)**2 +    &
                le(ij+u_rdown)*de(ij+u_rdown)*u(ij+u_rdown,l)**2 )
(Add gradients of Bernoulli and Exner functions to du)
l=ll_begin,ll_end
    ij=ij_begin,ij_end
        ( de(ij+u_right) > 1.0 )
            du(ij+u_right,l) = du(ij+u_right,l) + 1/de(ij+u_right) * ( &
            0.5*(theta(ij,l)+theta(ij+t_right,l))              &
            *( ne_right*pk(ij,l)+ne_left*pk(ij+t_right,l))    &
            + ne_right*berni(ij,l)+ne_left*berni(ij+t_right,l) )
        ( de(ij+u_lup) > 1.0 )
            du(ij+u_lup,l) = du(ij+u_lup,l) +  1/de(ij+u_lup) * (                &
            0.5*(theta(ij,l)+theta(ij+t_lup,l))               &
            *( ne_lup*pk(ij,l)+ne_rdown*pk(ij+t_lup,l))      &
            + ne_lup*berni(ij,l)+ne_rdown*berni(ij+t_lup,l) )
        ( de(ij+u_ldown) > 1.0 )
            du(ij+u_ldown,l) = du(ij+u_ldown,l) + 1/de(ij+u_ldown) * (           &
            0.5*(theta(ij,l)+theta(ij+t_ldown,l))              &
            *( ne_ldown*pk(ij,l)+ne_rup*pk(ij+t_ldown,l))    &
            + ne_ldown*berni(ij,l)+ne_rup*berni(ij+t_ldown,l) )
end subroutine compute_caldyn_hoziz
```

Figure 2.5: PAD of `compute_caldyn_horiz(3)`

2.4.3 Input data and result

Input data file is prepared and you can download from official server using `data/download.sh` script. This data file is created by original *DYNAMICO*[*4)]with Held-Suarez case parameter set included in the original source archive. Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of `$IAB_SYS=Ubuntu-gnu-ompi` case.

```
[KERNEL] comp_caldyn_horiz
*** Start  initialize
               iim, jjm, llm:    23    25    19
            ij_begin, ij_end:    48   528
    ij_begin_ext, ij_end_ext:    24   552
            ll_begin, ll_end:     1    19
       t_right, t_rup, t_lup:     1    23    22
    t_left, t_ldown, t_rdown:    -1   -23   -22
       u_right, u_rup, u_lup:     0  1173   575
    u_left, u_ldown, u_rdown:    -1  1150   553
          z_rup, z_up, z_lup:   598     0   597
    z_ldown, z_down, z_rdown:   -23   575   -22
              caldyn_conserv:     1
                  boussinesq:     F
                           g:     9.80000000
+check[pk_prev         ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[hflux_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[convm_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz_pre] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du_prev         ] max=  3.4399140149818440E-03,min= -3.0658810374294527E-03,sum=  5.5109794763703335E-01
+check[le              ] max=  1.3457165724385556E+05,min=  0.0000000000000000E+00,sum=  1.6031419146648201E+08
+check[Ai              ] max=  3.4618288017294556E+10,min=  0.0000000000000000E+00,sum=  1.7746401564746273E+13
+check[de              ] max=  4.5171816240714993E+06,min=  0.0000000000000000E+00,sum=  4.7785815753077912E+08
+check[Av              ] max=  4.1228713627140027E+11,min=  0.0000000000000000E+00,sum=  3.2428753277257527E+13
+check[Wee             ] max=  5.9893054722291683E-01,min= -5.8540209553487599E-01,sum=  2.4695023837951297E+01
*** Finish initialize
*** Start kernel
### check point iteration:           1
### Input ###
+check[u               ] max=  4.1278968179782127E-01,min= -4.1278968179782127E-01,sum=  1.6791131703073393E+01
+check[rhodz           ] max=  1.2306877011993038E+03,min=  0.0000000000000000E+00,sum=  5.3979591836733194E+06
+check[qu              ] max=  1.0339537867296609E-06,min= -8.4408169682701225E-07,sum=  3.9419811615778674E-04
+check[theta           ] max=  8.0139914420291746E+02,min=  0.0000000000000000E+00,sum=  3.8582633571973117E+06
+check[geopot          ] max=  3.8250620498369227E+05,min=  0.0000000000000000E+00,sum=  1.1718001851963627E+09
+check[pk_prev         ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[hflux_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[convm_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz_pre] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du_prev         ] max=  3.4399140149818440E-03,min= -3.0658810374294527E-03,sum=  5.5109794763703335E-01
### Output ###
+check[pk              ] max=  1.0014594722514462E+03,min=  0.0000000000000000E+00,sum=  6.9872296819747351E+06
+check[hflux           ] max=  3.1805763161244854E+07,min= -2.8604204589892026E+07,sum=  2.4131331333014986E+08
+check[convm           ] max=  1.0361970643226587E-03,min= -1.0359249303947807E-04,sum= -1.5233533963107249E-01
+check[dtheta_rhodz    ] max=  3.2251351666935379E-01,min= -3.3676276308628725E-02,sum= -5.3720539414185993E+01
+check[du              ] max=  3.4404317002518906E-03,min= -3.0804630348046005E-03,sum=  5.5048589972605033E-01
### final iteration:        1000
### Validation : grid-by-grid diff ###
+check[pk              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[hflux           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[convm           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz    ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel
```

Check the lines below ''`Validation : grid-by-grid diff`'' line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable. There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.4.4 Sample of performance result

Here's an example of the performance result part of the log output. Below is an example executed with the machine environment described in subsection 2.1.3. Note that in this program kernel part is iterated 1000 times.

[*4)]with slight modification by AICS.

```
*** Computational Time Report
*** ID=001 : MAIN_comp_caldyn_horiz           T=     0.876 N=    1000
```

2.5 `comp_caldyn_vert`

2.5.1 Description

Kernel `comp_caldyn_vert` is taken from the original subroutine `compute_caldyn_vert` in *DYNAMICO*. This subroutine is originally defined in module `caldyn_gcm_mod`. This module defines subroutine `caldyn`, which is the main subroutines for dynamics part of the model, and several sub-subroutines for various terms in the governing equation, such as potential vorticity, geopotential, etc. This subroutine calculates vertical mass flux and vertical transport.

2.5.2 Discretization and code

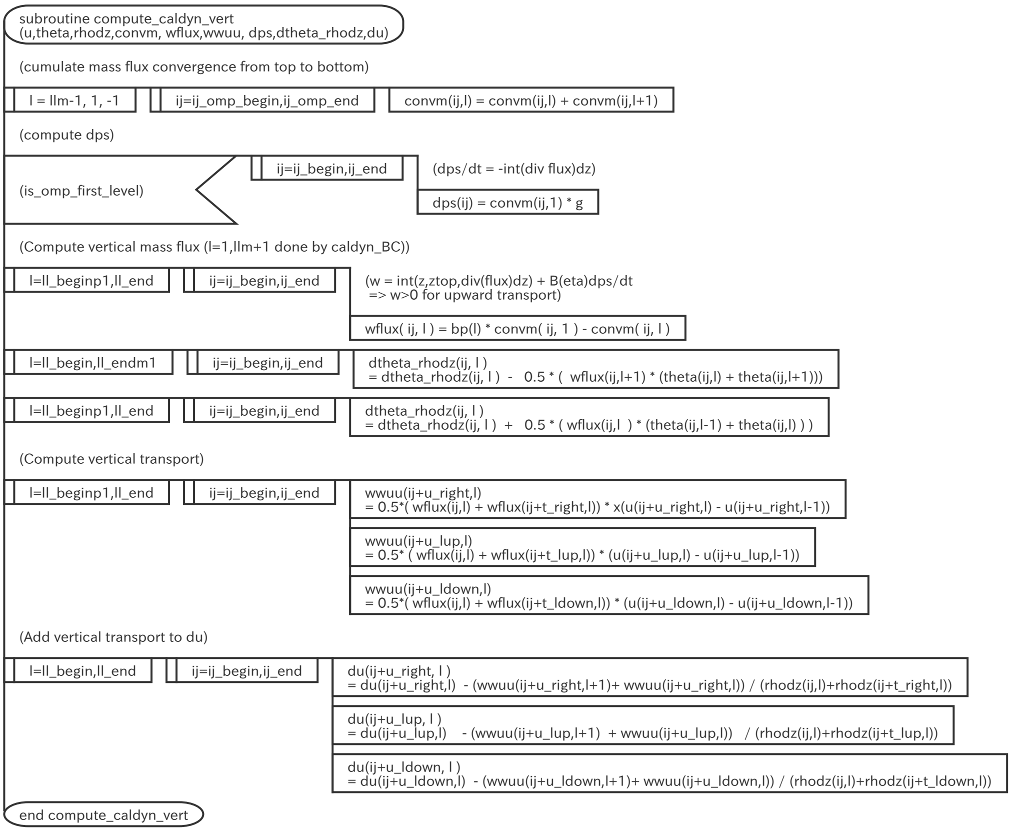
List 2.4 shows the definition part of this subroutine, and Figure 2.6 shows the PAD of this.

List 2.4: Definition part of `compute_caldyn_vert`

```fortran
SUBROUTINE compute_caldyn_vert(u,theta,rhodz,convm, wflux,wwuu, dps,dtheta_rhodz,du)
  USE icosa
  USE disvert_mod
  USE exner_mod
  USE trace
  USE omp_para
  IMPLICIT NONE
    REAL(rstd),INTENT(IN)  :: u(iim*3*jjm,llm)
    REAL(rstd),INTENT(IN)  :: theta(iim*jjm,llm)
    REAL(rstd),INTENT(IN)  :: rhodz(iim*jjm,llm)
    REAL(rstd),INTENT(INOUT)  :: convm(iim*jjm,llm)  ! mass flux convergence

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

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

Where `u`, `theta`, `rhodz` are wind velocity on the edge, potential temperature, and mass, respectively. `convm`, `wflux`, `wwuu` are mass flux convergence, vertical mass flux, and `wflux*u` on the edge, respectively. Last three variables are time derivatives. `du`, `dtheta_rhodz`, `dps` are for wind velocity on the edge, mass-weighted potential temperature, and surface pressure, respectively. All of these except `dps` are two dimensional. First dimension is for horizontal index, and the size depends on the point where the variable is defined, since *DYNAMICO* adopts C-grid. Second dimension is for vertical index, and the size is `llm`, except `llm+1` for `wflux` and `wwuu`, these are defined on the half level in vertical, while others are defined on the full level.

Main part of this subroutine is consist of several l- and ij- double loop. The first double loop is to accumulate mass flux convergence from top to bottom, then convert `convm` at the lowest level to `dps`. The second double loop is to compute vertical mass flux `wflux`. Note that the range of l-loop, because `wflux` is defined on half vertical level and at the top and the bottom are already set by subroutine `caldyn_BC` as a boundary condition. Next two double loop is to calculate convergence of potential temperature `dtheta_rhodz`. Again note that the range of two l-loop, since `dtheta_rhodz` is defined on full vertical level and needs to sum up both upper and lower face of the level. Next two double loop is to compute vertical transport `wwuu`, and to add it to `du`. Note the horizontal index here. `wwuu` and `du` are defined on the edge of control volume, there are three statement in each double loop.



```
subroutine compute_caldyn_vert
(u,theta,rhodz,convm, wflux,wwuu, dps,dtheta_rhodz,du)
(cumulate mass flux convergence from top to bottom)
l = llm-1, 1, -1
    ij=ij_omp_begin,ij_omp_end
        convm(ij,l) = convm(ij,l) + convm(ij,l+1)
(compute dps)
(is_omp_first_level)
    ij=ij_begin,ij_end
        (dps/dt = -int(div flux)dz)
        dps(ij) = convm(ij,1) * g
(Compute vertical mass flux (l=1,llm+1 done by caldyn_BC))
l=ll_beginp1,ll_end
    ij=ij_begin,ij_end
        (w = int(z,ztop,div(flux)dz) + B(eta)dps/dt
         => w>0 for upward transport)
        wflux( ij, l ) = bp(l) * convm( ij, 1 ) - convm( ij, l )
l=ll_begin,ll_endm1
    ij=ij_begin,ij_end
        dtheta_rhodz(ij, l )
        = dtheta_rhodz(ij, l )  -   0.5 * (  wflux(ij,l+1) * (theta(ij,l) + theta(ij,l+1)))
l=ll_beginp1,ll_end
    ij=ij_begin,ij_end
        dtheta_rhodz(ij, l )
        = dtheta_rhodz(ij, l )  +   0.5 * ( wflux(ij,l  ) * (theta(ij,l-1) + theta(ij,l) ) )
(Compute vertical transport)
l=ll_beginp1,ll_end
    ij=ij_begin,ij_end
        wwuu(ij+u_right,l)
        = 0.5*( wflux(ij,l) + wflux(ij+t_right,l)) * x(u(ij+u_right,l) - u(ij+u_right,l-1))
        wwuu(ij+u_lup,l)
        = 0.5* ( wflux(ij,l) + wflux(ij+t_lup,l)) * (u(ij+u_lup,l) - u(ij+u_lup,l-1))
        wwuu(ij+u_ldown,l)
        = 0.5*( wflux(ij,l) + wflux(ij+t_ldown,l)) * (u(ij+u_ldown,l) - u(ij+u_ldown,l-1))
(Add vertical transport to du)
l=ll_begin,ll_end
    ij=ij_begin,ij_end
        du(ij+u_right, l )
        = du(ij+u_right,l)  - (wwuu(ij+u_right,l+1)+ wwuu(ij+u_right,l)) / (rhodz(ij,l)+rhodz(ij+t_right,l))
        du(ij+u_lup, l )
        = du(ij+u_lup,l)    - (wwuu(ij+u_lup,l+1)  + wwuu(ij+u_lup,l))   / (rhodz(ij,l)+rhodz(ij+t_lup,l))
        du(ij+u_ldown, l )
        = du(ij+u_ldown,l)  - (wwuu(ij+u_ldown,l+1)+ wwuu(ij+u_ldown,l)) / (rhodz(ij,l)+rhodz(ij+t_ldown,l))
end compute_caldyn_vert
```

Figure 2.6: PAD of `compute_caldyn_vert`

2.5.3 Inputdata and result

Input data file is prepared and you can download from official server using `data/download.sh` script. This data file is created by original *DYNAMICO*[*5)]with Held-Suarez case parameter set included in the original source archive. Max/min/sum of input/output data of the kernel subroutine are output as a log. Below is an example of `$IAB_SYS=Ubuntu-gnu-ompi` case.

```
[KERNEL] comp_caldyn_vert
*** Start  initialize
               iim, jjm, llm:    23    25    19
            ij_begin, ij_end:    48   528
    ij_begin_ext, ij_end_ext:    24   552
            ll_begin, ll_end:     1    19
        ll_beginp1, ll_endm1:     2    18
       t_right, t_rup, t_lup:     1    23    22
    t_left, t_ldown, t_rdown:    -1   -23   -22
       u_right, u_rup, u_lup:     0  1173   575
    u_left, u_ldown, u_rdown:    -1  1150   553
          z_rup, z_up, z_lup:   598     0   597
    z_ldown, z_down, z_rdown:   -23   575   -22
                      dbg: g:     9.80000000
+check[bp              ] max=  1.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  6.9254510678414132E+00
*** Finish initialize
*** Start kernel
### check point iteration:        1000
### Input ###
+check[u               ] max=  4.1278968179782127E-01,min= -4.1278968179782127E-01,sum=  1.6791131703073393E+01
+check[theta           ] max=  8.0139914420291746E+02,min=  0.0000000000000000E+00,sum=  3.8582633571973117E+06
+check[rhodz           ] max=  1.2306877011993038E+03,min=  0.0000000000000000E+00,sum=  5.3979591836733194E+06
+check[convm_prev      ] max=  1.0361970643226587E-03,min= -1.0359249303947807E-04,sum= -1.5233533963107249E-01
+check[wflux_prev      ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[wwuu_prev       ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du_prev         ] max=  3.4404317002518906E-03,min= -3.0804630348046005E-03,sum=  5.5048589972605033E-01
+check[dtheta_rhodz_pre] max=  3.2251351666935379E-01,min= -3.3676276308628725E-02,sum= -5.3720539414185993E+01
+check[dps_prev        ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
### Output ###
+check[convm           ] max=  6.9593389571287302E-03,min= -7.9269107622801825E-04,sum= -1.5227927267389074E+00
+check[wflux           ] max=  4.1171035230973149E-04,min= -3.6145630748324665E-03,sum=  5.8901163077820706E-01
+check[wwuu            ] max=  1.7149300599654128E-04,min= -1.8768192515764522E-04,sum= -5.0377733672629036E-04
+check[du              ] max=  3.4404317002518906E-03,min= -3.0804630348046005E-03,sum=  5.5048632410110032E-01
+check[dtheta_rhodz    ] max=  3.5427038431326496E-01,min= -4.2032604085394595E-02,sum= -5.3720539414186263E+01
+check[dps             ] max=  6.8201521779861565E-02,min= -7.7683725470345790E-03,sum= -1.3213658793877174E+00
### final iteration:        1000
### Validation : grid-by-grid diff ###
+check[convm           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[wflux           ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[wwuu            ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[du              ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dtheta_rhodz    ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
+check[dps             ] max=  0.0000000000000000E+00,min=  0.0000000000000000E+00,sum=  0.0000000000000000E+00
*** Finish kernel
```

Check the lines below “`Validation : grid-by-grid diff`” line, that shows difference between calculated output array and pre-calculated reference array. These should be zero or enough small to be acceptable. There are sample output log files in `reference/` in each kernel program directory, for reference purpose.

2.5.4 Sample of performance result

Here’s an example of the performance result part of the log output. Below is an example executed with the machine environment described in subsection 2.1.3. Note that in this program kernel part is iterated 1000 times.

```
*** Computational Time Report
*** ID=001 : MAIN_comp_caldyn_vert            T=     0.156 N=   1000
```

[*5)] with slight modification by AICS.

Bibliography

T. Dubos, S. Dubey, M. Tort, R. Mittal, Y. Meurdesoif, and F. Hourdin. Dynamico-1.0, an icosahedral hydrostatic dynamical core designed for consistency and versatility. *Geoscientific Model Development*, 8(10):3131–3150, 2015. doi: 10.5194/gmd-8-3131-2015. URL https://www.geosci-model-dev.net/8/3131/2015/.

Thomas Dubos and Marine Tort. Equations of atmospheric motion in non-eulerian vertical coordinates: Vector-invariant form and quasi-hamiltonian formulation. *Monthly Weather Review*, 142(10):3860–3880, 2014. doi: 10.1175/MWR-D-14-00069.1. URL https://doi.org/10.1175/MWR-D-14-00069.1.

T. D. Ringler, J. Thuburn, J. B. Klemp, and W. C. Skamarock. A unified approach to energy conservation and potential vorticity dynamics for arbitrarily-structured c-grids. *J. Comput. Phys.*, 229(9):3065–3090, May 2010. ISSN 0021-9991. doi: 10.1016/j.jcp.2009.12.007. URL http://dx.doi.org/10.1016/j.jcp.2009.12.007.

Marine Tort and Thomas Dubos. Usual approximations to the equations of atmospheric motion: A variational perspective. *Journal of the Atmospheric Sciences*, 71(7):2452–2466, 2014. doi: 10.1175/JAS-D-13-0339.1. URL https://doi.org/10.1175/JAS-D-13-0339.1.

Appendix A

Brief description of PAD

A.1 PAD is

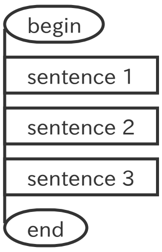
Problem Analysis Diagram (PAD) is to describe the logical structure of the program by the two dimensional tree. PAD is suitable for structured programing, like Fortran.

A.2 Elements in PAD

In PAD, one box shows one process or one sentence in Fortran, and only three basic structure is allowed; sequence, conditional branch, and iteration.

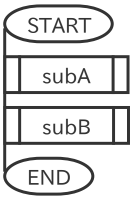
A.2.1 Sequence

Simple sequence of sentence is shown as listed box in same vertical line from top to bottom (Figure A.1). Subroutine call is shown as Figure A.2.



```
sentence 1
sentence 2
sentence 3
```

Figure A.1: elements of PAD: sequence

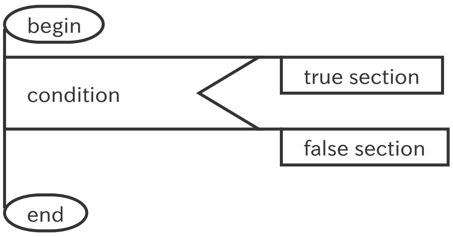


```
call subA()
call subB()
```

Figure A.2: elements of PAD: subroutine call

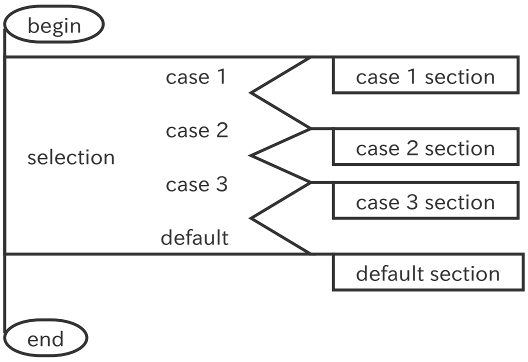
A.2.2 Conditional branch

Conditional branch and selection are shown in the same manner. Figure A.3 shows the IF-THEN-ELSE type branch, and Figure A.4 shows the CASE type branch.



```
if ( condition ) then
  true section
else
  false section
end if
```

Figure A.3: elements of PAD: IF-THEN-ELSE

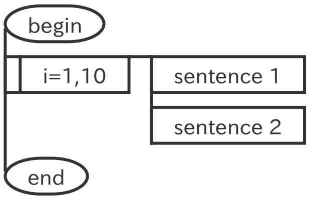


```
select case ( selection )
case (case1)
  case1 section
case (case2)
  case2 section
case (case3)
  case3 section
case default
  default section
end case
```

Figure A.4: elements of PAD: CASE Selection

A.2.3 Iteration

Iteration is shown as Figure A.5.

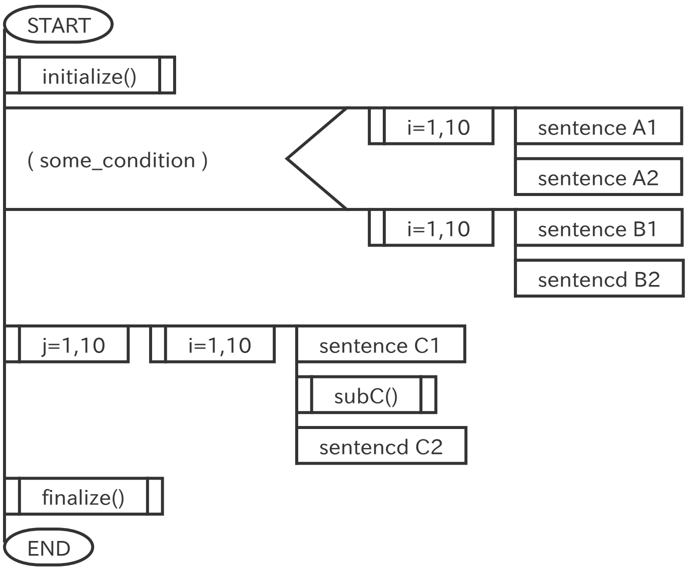


```
do i=1, 10
  sentence 1
  sentence 2
end do
```

Figure A.5: elements of PAD: Iteration

A.2.4 Hierarchy

Hierarchy in program is expressed as connection of boxes in right direction. So width of the PAD means complexity of the program and height means the size of the program. Figure A.6 shows the example of hierarcy in PAD. The first half shows combination of IF-THEN-ELSE clause and DO-loop, the latter half shows usual double Do-loop.



```
call initialize()

if ( some_condition ) then
  do i=1,10
    sentence A1
    sentence A2
  end do
else
  do i=1,10
    sentence B1
    sentence B2
  end do
end if

do j=1,10
  do i=1,10
    sentence C1
    call subC()
    sentence C2
  end do
end do

call finalize()
```

Figure A.6: Example of hierarcy in PAD

IcoAtmosBenchmark DYNAMICO kernels

Author & Editor

SPPEXA/AIMES Benchmarking team

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