Piggybacking in WRF V3.7.1; Documentation

Written by Noémi Sarkadi, 2018.

Basic concept of piggybacking:

The basic concept in briefly is the following. We have two sets of variables:

(1) DRIVER: temp, pressure, u, v, w, moisture, and microphysical variables (number concentration and mixing ratio for different type of particles as cloud water, raindrops, snowflakes, graupel particle and pristine ice).
(2) PIGGYBACK: temp2, pressure, u, v, w, moisture2 and microphysical variables2.

NOTE: temperature, moisture (water vapor content) and microphysics are differed in the DRIVER and in PIGGYBACK. The pressure, u, v, w, etc. are the same for both sets.

1. step: the driver and piggyback variables are the same.
2. step:
   a. calculating diagnostic and prognostic variables for the driver scheme.
   b. calculating diagnostic and prognostic variables for the piggyback scheme.

Update temperature, moisture, microphysics and dynamics in the case of DRIVER. Update temperature, moisture and microphysics in the case of PIGGYBACK – in this case same dynamics will set as in the DRIVER.

3. step: repeat the above steps until the end of the simulation. In every timestep DRIVER updates the dynamics, and PIGGYBACK updates just the related temperature, moisture and microphysics but using the same dynamics (pressure, u, v, w, etc.) like the DRIVER used.

For more details see reference: Grabowski (2015) and Grabowski-Morrison (2016).

Added new variables to WRF packages:

New namelist variables list:

- mp_physics_pg
    - related to mp_physics_pg new physics packages are defined (all are the same, with the same namelist number as originally, just with scalar variables.)
      For example:

package thompson mp_physics==8 - moist:qv,qc,qr,qi,qs,qg;scalar:qni,qnr

package thompson_pg mp_physics_pg==8 - scalar:qv_pg,qc_pg,qr_pg,qi_pg,qs_pg,qg_pg,qni_pg,qnr_pg

- do_piggyback
- diag_piggyback_init
- refl_10cm_pg

New output/diagnostic variables list:

Thermodynamic variables:

- t_pg
- t_phy_pg
- th_phy_pg
- t_init_pg
- t_base_pg
- t_save_pg
- t_2save_pg
- ttend_pg
- ttendf_pg
- qv_base_pg
- dfi_rad_ttend_pg
- h_diabatic_pg

For initialization of different input_soundings:

- mub_pg: additional variable for input sounding calculations, do not used in other parts of the code, just in module_initialization_*casename*.F
- p_top_pg: additional variable for input sounding calculations, do not used in other parts of the code, just in module_initialization_*casename*.F
- phb_pg: additional variable for input sounding calculations, do not used in other parts of the code, just in module_initialization_*casename*.F
- pb_pg: additional variable for input sounding calculations, do not used in other parts of the code, just in module_initialization_*casename*.F
- alb_pg: additional variable for input sounding calculations, do not used in other parts of the code, just in module_initialization_*casename*.F

Moisture variables:

- qv_pg: water vapor content, set as scalar (to do advection, but do not interact with the dynamics)
- qc_pg: cloud water content, set as scalar (to do advection, but do not interact with the dynamics)
- qr_pg: rain water content, set as scalar (to do advection, but do not interact with the dynamics)
- qi_pg: pristine ice content, set as scalar (to do advection, but not interact with the dynamics)
- qs_pg: snow content, set as scalar (to do advection, but do not interact with the dynamics)
- qg_pg: graupel content, set as scalar (to do advection, but do not interact with the dynamics)
- qnc_pg: cloud water number concentration, set as scalar (to do advection, but do not interact with the dynamics)
- qnr_pg: rain water number concentration, set as scalar (to do advection, but do not interact with the dynamics)
- qni_pg: pristine ice number concentration, set as scalar (to do advection, but do not interact with the dynamics)
- qns_pg: snowflakes number concentration, set as scalar (to do advection, but do not interact with the dynamics)
- qng_pg: graupel number concentration, set as scalar (to do advection, but do not interact with the dynamics)
- all the related dfi_* variables was defined for the piggybacking.

Others:

- All the related HALO, PERIOD, etc. were updated including the piggybacking variables (temperature, moisture, microphysics if it is includes the original (DRIVER) variables).

Modified source code list (additional information) for piggybacking:

1. **<u>dyn_em/</u>**

   The following modules/subroutines were modified to be ready for the piggybacking method. All the coding was following the original philosophy of the WRF module system.

   1.1.start_em.F:

   (i) added variables t_init_pg, t_pg_1, t_pg_2.

   (ii) In the case of same sounding using for the piggyback and driver simulation, these are the same as the driver scheme variables (as t_init, t_1, t_2). For the piggybacking variables subroutine *set_physical_bc3d* also called (t_pg_1, t_pg_2, t_init_pg)

   1.2.solve_em.F:

   (i) variables for piggybacking with the microphysics: theta_pre_advect_pg, p8w_old, p_phy_old, pi_phy_old, rho_old, w_2_old. These saved as the driver before calling the microphysics_driver for the driver case (just in the case when call twice the microphysics driver in solve_em.F. If we follow the method that calls the microphysics driver just one these variables are unnecessary.).

   (ii) Call *zero_bdytend* including piggybacking variables: grid%t_pg_btxs, grid%t_pg_btxe, grid%t_pg_btys, grid%t_pg_btye.

   (iii) Call *rk_step_prep* including piggybacking variables: grid%t_pg_2.

   (iv) Call *first_rk_step_part1* including piggybacking variables: t_tendf_pg, th_phy_pg and grid%t_phy_pg.

   (v) Call *first_rk_step_part2* including piggybacking variables: t_tendf_pg, th_phy_pg and grid%t_phy_pg.

   (vi) Call *rk_tendency* including piggybacking variables: t_tend_pg, t_tendf_pg, t_save_pg, rthftenpg, t_pg_2, t_pg_1, h_diabatic_pg, t_init_pg, t_base_pg, qv_base_pg.

   (vii) Call *relax_bdy_dry* including piggybacking variables: t_save_pg, t_pg_2, grid%t_pg_bxs, grid%t_pg_bxe, grid%t_pg_bys, grid%t_pg_bye, grid%t_pg_btxs, grid%t_pg_btxe, grid%t_pg_btys, grid%t_pg_btye.

   (viii) Call *rk_addtend_dry* including piggybacking variables.

   (ix) Call *spec_bdy_dry* including related piggybacking variables.

   (x) Call *small_step_prep* including related piggybacking variables.

   (xi) Call *set_physical_bc3d* with t_pg_1 and t_save_pg.

   (xii) Call *advance_mu_t* including related piggybacking variables.

   (xiii) Call *spec_bdyupdate* with t_pg_2 and t_tendf_pg.

   (xiv) Call *advance_w* including related piggybacking variables.

   (xv) Call *small_step_finish* including related piggybacking variables.

   (xvi) Call *rk_scalar_tend* including related piggybacking variables.

   (xvii) Call *rk_phys_bc_dry_2* including related piggybacking variables.

   (xviii) Call *moist_physics_prep_em* including related piggybacking variables.

(xix) Call *microphysics_driver* including both driver and piggybacker variables. Inside the driver two physics selection (1.driver, 2.piggybacker).
(xx) Call *moist_physics_finish_em* including related piggybacking variables.
(xxi) Call *set_phys_bc_dry_2* including relate piggybacking variables.

1.3.module_small_step.F:

(i) subroutine *small_step_prep* includes piggybacking variables: t_pg_1, t_pg_2, t_save_pg, and the following equations solved for piggybacking variables:
t_pg_1(i,k,j) = t_pg_2 (i,k,j);
t_save_pg(i,k,j) = t_pg_2(i,k,j)
t_pg_2(i,k,j) = muts(i,j)*t_pg_1(i,k,j)-mut(i,j)*t_pg_2(i,k,j)

(ii) subroutine *small_step_finish* includes piggybacking variables: t_pg_2, t_pg_1, t_save_pg and h_diabatic_pg and the following are calculated:
t_pg_2(i,k,j) = (t_pg_2(i,k,j) + t_save_pg(i,k,j)*mut(i,j))/muts(i,j);
t_pg_2(i,k,j) = (t_pg_2(i,k,j) + t_save_pg(i,k,j)*mut(i,j))/muts(i,j);
t_pg_2(i,k,j) = (t_pg_2(i,k,j) -dts*number_of_small_timesteps*mut(i,j)* h_diabatic_pg(i,k,j) & + t_save_pg(i,k,j)*mut(i,j))/muts(i,j)

(iii) subroutine *advance_mu_*t includes piggybacking variables: t_pg, t_pg_1, t_ave_pg, ft_pg and the following calculations are made:
t_ave_pg(i,k,j) = t_pg(i,k,j);
t_pg (i,k,j) = t_pg(i,k,j) + msfty(i,j)*dts*ft_pg(i,k,j);
wdtnpg(i,k)= ww(i,k,j)*(fnm(k)*t_pg_1(i,k ,j)+fnp(k)*t_pg_1(i,k-1,j));
Multiplication by msfty uncouples result for Theta with piggybacking temperatures: t_pg(i,k,j) = …

(iv) subroutine *advance_w* includes piggybacking variables: t_2ave_pg, t_pg_2 and t_pg_1 and the following calculations are made:
t_2ave_pg(i,k,j)=.5*((1.+epssm)*t_pg_2(i,k,j) +(1.-epssm)*t_2ave_pg(i,k,j))
t_2ave_pg(i,k,j)=(t_2ave_pg(i,k,j) + muave(i,j)*t0) /(muts(i,j)*(t0+t_pg_1(i,k,j)))

1.4.module_stoch.F:

(i) subroutine *CALCULATE_STOCH_TEN* includes piggybacking variables: t_tendf_pg, GPTFORCPG and rt_pg_real and the following calculation was made:
GPTFORCPG(i,k,j)= rt_pg_real(i,k,j)

(ii) subroutine *UPDATE_STOCH_TEN* includes piggybacking variables: t_tendf_pg and GPTFORCPG and the following calculation was made:
t_tendf_pg(i,k,j) = t_tendf_pg(i,k,j) + GPTFORCPG(i,k,j) * (mu(i,j)+mub(i,j))

(iii) subroutine *SP2GP_prep* includes piggybacking variables: RT_PG_REAL and RT_PG_IMAG.

(iv) subroutine *do_fftback_along_x* includes piggybacking variables: fieldc_T_PG_xxx,fields_T_PG_xxx and calculations made as same in the case of the driver.

(v) Subroutine *do_fftback_along_y* includes piggybacking variables: fieldc_T_PG_yyy,fields_T_PG_yyy and calculations made the same ways as in the case of the driver.

1.5.module_first_rk_step_part1.F:

(i) subroutine *first_rk_step_part1* includes: t_tendf_pg, th_phy_pg, t_phy_pg.
(ii) CALL *init_zero_tendency* includes t_tendf_pg variable.
(iii) Call *phy_prep* includes: t_pg_2, th_phy_pg, t_phy_pg right after the original variables.

1.6.module_first_rk_step_part2.F:

(i) subroutine *first_rk_step_part2* includes: t_tendf_pg, th_phy_pg and t_phy_pg.
(ii) call *sp2gp_prep* includes: RT_PG_REAL, RT_PG_IMAG variables (all grid%).
(iii) Call *do_fftback_along_x* includes: RT_PG_REAL_xxx and RT_PG_IMAG_xxx (all grid%)
(iv) Call *do_fftback_along_y* includes: grid%RT_PG_REAL_yyy,grid%RT_PG_IMAG_yyy varibales.
(v) Calling the *calculate_stoch_ten* subroutine, it includes: t_tendf_pg, grid%rt_tendf_pg_stoch and grid%RT_PG_REAL variables.
(vi) Calling *update_stoch_ten* subroutine, including the following piggybacking variables: t_tendf_pg, grid%rt_tendf_pg_stoch.
(vii) Calling *damptop* twice: first time with the original variables, and second time with the following piggybacking variables: t_pg_2, t_tendf_pg.
(viii) Calling *vertical_diffusion_2* including piggybacking variables: t_tendf_pg, t_pg_2, t_base_pg, and qv_base_pg.
(ix) Calling *horizontal_diffusion_2* including piggybacking variables: t_tendf_pg, t_pg_2 and th_phy_pg.

1.7.couple_or_uncouple.F:

(i) An extra line added for t_pg_2 (grid%t_pg_2(i,k,j) = grid%t_pg_2(i,k,j)*mut_2(i,j)), where mut_2 related to the driver mut_2.
Subroutine *set_physical_bc3d* called for t_pg_1 and t_pg_2 as well.

1.8.module_bc_em.F:

(i) Subroutine *relax_bdy_dry* additional variables for piggybacking (t_tendf_pg, t_pg, t_pg_bxs, t_pg_bxe, t_pg_bys, t_pg_bye, t_pg_btxs, t_pg_btxe, t_pg_btys, t_pg_btye). All of the above variables defined in the same way as driver (original) temperature variables.

In this subroutine rtfield set for piggybacking after original temperature field: rfield(i,k,j) = t_pg(i,k,j)*mut(i,j), and then subroutine *relax_bdytend_tile* called with (rfield, and t_tendf_pg)

(ii) Subroutine *spec_bdytend* called with piggyback temperature variables as well.

(iii) Followed the abovementioned philosophy subroutine *spec_bdy_dry* also includes new variables for piggybacking temperature. And subroutine *spec_bdytend* called with piggybacking temperatures.

(iv) Subroutine *set_phys_bc_dry_2* has additional variables for piggybacking (t_pg_1 and t_pg_2, just right after the original (driver) t_1 and t_2 variables). They are defined as the same way of the driver variables. Subroutine *set_physical_bc3d* called with the piggybacking temperature.

(v) Subroutine *rk_phys_bc_dry_2* includes t_pg_2 piggybacking temperature, and inside this subroutine *set_physical_bc3d* called with t_pg_2 as well.

(vi) Subroutine *zero_bdytend* includes piggybacking variables (t_pg_btxs, t_pg_btxe, t_pg_btys, t_pg_btye) and all set to be zero like the driver t_btxs, etc.

1.9.module_big_step_utilities_em.F:

(i) USE *module_state_description*: additional variable: qv_pg

(ii) subroutine *phy_prep* includes additional variables for piggybacking, these are the following: t_pg, th_phy_pg, t_phy_pg. With these variables:
th_phy_pg(i,k,j) = t_pg(i,k,j) + t0 and
t_phy_pg(i,k,j) = th_phy_pg(i,k,j)*pi_phy(i,k,j) are calculated.

(iii) subroutine *moist_physics_prep_em* includes piggybacking variables: t_pg_new, t_pg_old, th_phy_pg, h_diabatic_pg and the following equations repeated by using the piggybacking variables:

    t_pg_new(i,k,j) = t_pg_new(i,k,j)-h_diabatic_pg(i,k,j)*dt
    th_phy_pg(i,k,j) = t_pg_new(i,k,j) + t0
    h_diabatic_pg(i,k,j) = th_phy_pg(i,k,j)

(iv) subroutine *moist_physics_finish_em* also includes the abovementioned piggybacking variables: t_pg_new, t_pg_old, th_phy_pg, h_diabatic_pg and dfi_ttend_rad_pg
Also defined: REAL :: dfi_tten_max_pg, old_max_pg.
The following repeated for piggybacking too:
#if (WRF_DFI_RADAR == 1)
#endif
Also ! add microphysics theta diff to perturbation theta, set h_diabatic – repeated for piggybacking theta, perturbation theta and h_diabatic_pg…

(v) subroutine *rk_rayleigh_damp* includes piggybacking variables, as: t_tendf_pg, t_pg, t_init_pg and t_base_pg.
Adjust potential temperature to base state piggyback. (everything is same as in the case of the driver, just using piggybacking temperature variables)

(vi) Subroutine *theta_relaxation* also have additional piggybacking variables: t_tendf_pg, t_pg, t_init_pg and t_base_pg.
Adjust potential temperature to base state piggyback. (everything is same as in the case of the driver, just using piggybacking temperature variables)

1.10. module_diffusion_em.F

(i) subroutine *horizontal_diffusion_2* includes piggybacking variables: rt_tendf_pg, thp_pg, theta_pg. In this subroutine *horizontal_diffusion_s* with rt_tendf_pg and thp_pg was called.

(ii) Subroutine *vertical_diffusion_2* includes piggybacking variables: rt_tendf_pg, thp_pg, t_base_pg, qv_base_pg. In this subroutine *vertical_diffusion_s* with piggybacking rt_tendf_pg called. Also in related to config_flags%isfflx calculations the original equations repeated with piggybacking variables (P_QV_PG, and rt_tendf_pg).

1.11. module_em.F:

(i) subroutine *rk_step_prep* includes t_pg variable (it is not necessary, but did not change any results).

(ii) Subroutine *rk_tendency* includes: t_tend_pg, t_tendf_pg, t_save_pg, RTHFTENPG, t_pg, t_old_pg, h_diabatic_pg, t_init_pg, t_base_pg and qv_base_pg.
In this subroutine *zero_tend* called with t_tend_pg, t_save_pg.
Subroutine *advect_scalar* called with t_pg, t_tend_pg variables.
Subroutine *set_tend* called with RTHFTENPG, t_tend_pg variables.
Subroutine *horizontal_diffusion_3dmp* called with t_pg, t_tendf_pg, t_init_pg variables.
Subroutine *vertical_diffusion_3dmp* called with t_pg, t_tendf_pg, t_init_pg variables.
Subroutine *sixth_order_diffusion* called with t_pg and t_tendf_pg variables.
Subroutine *rk_rayleigh_damp* and *theta_relaxation* includes the piggybacking variables when they called (*see modifications related to 1.11*)

(iii) Subroutine *rk_addtend_dry* includes piggybacking variables: t_tend_pg, t_tendf_pg, t_save_pg and h_diabatic_pg. In this subroutine t_tend_pg calculated follows t_tend calculation except using piggybacking temperature tendency and h_diabatic variables (all the other variables are the same).

(iv) Subroutine *rk_scalar_tend* includes piggybacking variables: RQVFTENPG, base_pg. And call subroutine *set_tend* with RQVFTENPG.

(v) Subroutine *init_zero_tendency* includes: t_tendf_pg and inside the subroutine *zero_tend* called with t_tendf_pg.

1.12. module_initialize_quarter_ss.F:

(i) modified get_souding subroutine with new variables for piggybacking input_sounding reading, in the case if different initial conditions would like to set with same microphysics.
piggyback = .TRUE.
CALL get_sounding(zk_pg, p_in_pg, pd_in_pg, theta_pg, rho, u, v, qv_pg, dry_sounding, nl_max, nl_in_pg, piggyback)
piggyback = .FALSE.
CALL get_sounding(zk, p_in, pd_in, theta, rho, u, v, qv, dry_sounding, nl_max, nl_in, piggyback)

(ii) modified read_sounding subroutine:
call read_sounding(p_surf, th_surf, qv_surf, h_input, th_input, qv_input, u_input, v_input,n, nl, debug, piggyback).
Including if statement in the case of piggyback = .true., then using a different input_sounding named input_sounding_pg.

1.13. module_initialize_squall2d_x.F:

(i) Same modifications made like in module_initialize_quarter_ss.F

2. <u>phys/</u>

2.1.module_microphysics_driver.F: including piggybacking variables as well. Two physics selection.

2.2.module_physics_init.F:

(i) In subroutine mp_init second selection of microphysics options based on the namelist variables mp_physics_pg. The same microphysics options used as originally.

Piggyback with WRF:

Different options are available for users who would like to test with piggyback code in idealized cases (including 2D and 3D cases).

Option 1: Using same microphysics as a driver and as the piggybacker, but with different initial conditions (temperature and moisture).

What need to do to simulate such cases?

1. Set namelist variable: mp_physics_pg same as mp_physics
2. Set diag_piggyback_init: 1 (default is 0, which means that no piggyback microphysics initialization)
3. Set do_piggyback: 1 (default is 0, which means that no piggyback)
4. Copy to work directory input_sounding – as for the driver, and copy input_sounding_pg – input sounding for the piggybacker code.
5. Run WRF.

NOTE: The above option are really poorly tested. It is not entirely clear what conclusions can be drawn from the results.

Option 2: Using the same initial conditions, but with two different sets of microphysics

What need to do to simulate such cases?

1. Namelist variable mp_physics_pg describe which microphysics would like to use for piggybacking. Mp_physics option still related to the driver scheme.
2. diag_piggyback_init : 1
3. do_piggyback: 1
4. Select the case: quarter_ss, les
5. Run WRF

References:

W. W. Grabowski, 2015: **Untangling Microphysical Impacts on Deep Convection Applying a Novel Modeling Methodology**. *Journal of the Atmospheric Sciences*, Vol. 72, pp. 2446-2464.

W.W. Grabowski and H. Morrison, 2016: **Untangling Microphysical Impacts on Deep Convection Applying a Novel Modeling Methodology. Part 2: Double-Moment Microphysics**. *Journal of the Atmospheric Sciences*, Vol. 73, pp. 3749-3770.