# Annual Report: WRF PHYSICS

Shu-hua Chen and Jimy Dudhia

## ABSTRACT

The Weather Research and Forecast (WRF) model is a next-generation mesoscale model, and it is developed by a group of scientists from different institutes and centers. This project is part of the WRF development efforts, and it mainly focuses on the implementation of physics and the design of the WRF physics interface. A basic set of physics components is incorporated into WRF, and a three-level physics structure is designed.

## 1. INTRODUCTION

The Weather Research and Forecast (WRF) model is a next-generation mesoscale model, and it is a joint effort of model development among several organizations, such as the National Center for Atmospheric Research (NCAR), National Centers for Environmental Prediction (NCEP), Forecast Systems Laboratory (FSL) / National Oceanic and Atmospheric Administration (NOAA), Center for the Analysis and Prediction of Storms (CAPS) at University of Oklahoma, National Aeronautics and Space Administration (NASA), Air Force Weather Agency (AFWA), and a number of collaborating institutes and universities. Besides developing a next-generation model, one of the major purposes of developing this model is to tie the research and operational groups together in order to combine the essences of model-developed efforts from theoretical and practical aspects, as can be visualized from the title of the model. Therefore, the WRF model can be applied to real weather simulations, as well as idealized studies. The model is designed to be platform independent, and it can be executed on both shared and distributed machines. The application of the WRF model is focused on simulations with a resolution of 1-10 km, though it could also be applied to lower resolution.

The WRF is a fully compressible nonhydrostatic model, and its governing equations are written in flux form for the purpose of conservation of mass, dry entropy, and scalars. Instead of using a B grid as in MM5, the WRF chooses an Arakawa C grid to gain a better accuracy in higher

resolution simulations. In dynamic framework, there are three candidate codes under development. They will be verified through a series of tests and one of them will be chosen and maintained inside the official WRF model. The first two candidates use a time split explicit scheme (Klemp and Wilhelmson 1978) to solve the dynamic equations, namely using two different time steps. Also, the model uses a implicit scheme to solve vertical high frequency waves and explicit schemes to solve the others. The major difference between these two dynamic codes is the use of coordinates, which are geometric height and mass (hydrostatic pressure) coordinates. The Runge-Kutta second- (Wicker and Skamarock 1998) and third-order time schemes, and higher order space schemes are implemented. The third candidate of dynamic code applies a semi-implicit semi-lagrangian scheme to solve the dynamic equations, and this approach allows the model to take a larger time step than those of the previous two candidates. Currently, only the one using the time split explicit scheme with geometric height coordinates is ready, and the other two are still under development.

To simulate real weather and to do simulations with coarse resolutions, a set of physics components are required, such as the radiation, boundary layer parameterization, convective parameterization, subgrid eddy diffusion, and microphysics. Since the model is developed for both research and operational groups, sophisticated physics schemes, as well as simple physics schemes, are needed in the model. The objectives of this project are to implement a basic set of physics into WRF and to further design a user friendly three-level physics interface. Several meetings have been called in the WRF physics group to discuss the detailed implementation of the physics, as well as to choose the first set of physics schemes. Since WRF is targeted for a resolution of 1-10 km, some of the physics schemes might not work properly in this range of resolution (*e.g.* cumulus parameterization). However, at this early stage of model development, only existing physics schemes are implemented, and most of them are taken from existing models or packages. In the future, new physics schemes for the resolution of 1-10 km should be developed and implemented.

Section 2 briefly introduces those physics schemes, which have been implemented into WRF. The design of a three-level physics structure (physics interface) is given in Section 3, and the procedure of implementing a new scheme is provided in Section 4. A squall line simulation is presented in Section 5, and a short summary is given in Section 6.

## 2. IMPLEMENTED PHYSICS SCHEMES

Currently, several physics components have been included in the WRF: microphysics, cumulus parameterization, long wave radiation, short wave radiation, boundary layer turbulence (PBL), surface layer, land-surface parameterization, and subgrid scale diffusion. As mentioned earlier, the physics schemes are simply taken from existing packages or other models at this early stage of model development. The available schemes, as well as development schemes, are summarized in Table 1. The rest of this section will briefly introduce those available schemes.

### 2.1 Microphysics

(a) Kessler (kesslerscheme*)

Kessler scheme (Kessler 1969), which was taken from COMMAS, is a simple warm cloud scheme, and it includes water vapor, cloud water, and rain. The microphysics processes included are: the production, fall and evaporation of rain, the accretion of cloud water, and the production of cloud water from condensation.

(b) Lin (linscheme)

The scheme is taken from Purdue cloud model (Chen 1994). Six classes of hydrometeors are included: water vapor, cloud water, rain, cloud ice, snow, and graupel. All parametrization production terms are based on Lin et al. (1983) and Rutledge and Hobbs (1984) with some modifications, and the saturation adjustment follows Tao (1989). This is a relatively sophisticated microphysics scheme in the current WRF, and it is more suitable for use in research mode.

(c) NCEP simple ice (ncepcloud3)

This scheme follows Hong et al. (1998) with some modification. Three categories of hydrometeors are included: vapor, cloud water/ice, and rain/snow. The cloud ice and cloud water are counted as the same category, and they are distinguished by temperature, namely that the cloud ice can only exist when the temperature is less than or equal to the freezing point or otherwise, cloud water can exist. The same condition is applied to rain and snow. Though the ice phase is included, it is considered simple enough for using in operational mode.

* Name in parenthesis denotes current WRF code desination.

Currently, the sedimentation process, which could be combined with the vertical velocity in the future, is counted inside the microphysics, and a smaller time step is allowed to calculate the vertical flux of precipitation to prevent instability. The saturation adjustment is also included inside the microphysics, and it might be separated as an individual subroutine in the future.

2.2 Convective schemes

(a) Kain-Fritsch (kfscheme)

The Kain-Fritsch scheme (Kain and Fritsch 1990) is taken from the MM5 model. A simple cloud model, which includes detrainment, entrainment, updraft, and downdraft, is incorporated into this scheme. The current version only includes deep convection, while a new version in the near future will also include shallow convection.

(b) Bett-Miller-Janjic (bmjscheme)

The scheme is based on the Bett-Miller scheme (Betts 1986; Betts and Miller 1986) with some modifications (Janjic 1994) and it is taken from the Eta model. This is a convective adjustment scheme that also includes shallow convection.

2.3 Long wave radiation

(a) RRTM (rrtmscheme)

The RRTM, which is taken from MM5, is based on Mlawer et al. (1997), and it is a spectral-band scheme using the correlated-k method. It uses pre-set tables to accurately represent longwave processes due to water vapor, ozone, CO2, and trace gases (if present) as well as accounting for cloud optical depth.

2.4 Short wave radiation

(a) Simple short wave (swscheme)

The scheme is base on Dudhia (1989) and taken from MM5. It is a simple downward integration of solar flux, accounting for clear-air scattering, water vapor absorption (Lacis and Hansen 1974), and cloud albedo and absorption. It uses look-up tables for clouds from Stephens (1978).

(b) Goddard short wave (gsfcswscheme)

It is a sophisticated spectral scheme developed by Chou and Suarez (1994) and Chou et al. (1998).

2.5 Surface layer

(a) Similarity theory (sfclayscheme)

The scheme uses stability functions from Paulson (1970), Dyer and Hicks (1970), and Webb (1970) to compute surface exchange coefficients for heat, moisture, and momentum.

2.6 Land-surface

(a) Thermal diffusion (slabscheme)

The scheme is based on the MM5 5-layer soil temperature model. Layers are 1, 2, 4, 8, and 16 cm thick. Below these the temperature is fixed at diurnal average. The energy budget includes radiation, sensible, and latent heat flux. Also it allows for a snow-cover flag and a split shorter timestep.

(b) Oregon State-Eta scheme (lsmscheme)

This is a 4-layer soil temperature and moisture model (Chen and Dudhia 2000) with canopy moisture and snow cover prediction. It includes root zone, and other vegetation effects, drainage, and runoff. The model provides sensible and latent heat fluxes to boundary-layer scheme.

2.7 Boundary layer parameterization

(a) MRF (mrfscheme)

The scheme is described by Hong and Pan (1996). This uses a so-called countergradient flux for heat and moisture in unstable conditions. It uses enhanced vertical flux coefficients in the PBL, and the PBL height is determined from a critical bulk Richardson number (Ri). It handles vertical diffusion with an implicit local scheme, and it is based on local Ri in the free atmosphere.

2.8 Subgrid eddy diffusion

(a) Simple diffusion (diff_opt=1)

A simple diffusion is built using K theory, where K is a constant. This option comes with the choice of diff_opt =1 in the namelist.input file. The momentum eddy coefficients are khdif for

the horizontal and kvdif for the vertical (khdif and kvdif also in the namelist.input file). The heat eddy coefficients are three times of momentum eddy coefficients. This option simply acts on the terrain-following surfaces.

(b) Stress/deformation form (diff_opt=2)

The subgrid diffusion equations with stress formulae are derived to match the WRF governing equations, and the detailed derivation is in Appendix A. Three eddy diffusion coefficients are available: constant, function of turbulence kinetic energy (TKE), and function of deformation. The constant coefficients (km_opt =1) take the values of khdif and kvdif as in (a), but use deformation rather than just second order derivatives for momentum. The variable eddy coefficients are in terms of predicted TKE (km_opt=2) or Smagorinsky's diagnostic deformation-dependent term (km_opt=3). The TKE equation mainly follows the ARPS model, Moeng (1984), and Moeng and Wyngaard (1989). It is noted that the vertical subgrid diffusion is only called if there is no boundary layer scheme, and the horizontal terms take into account the terrain-following coordinate.

## 3. A THREE-LEVEL PHYSICS STRUCTURE

The WRF model does not allow to use common blocks and therefore, all variables have to pass into subroutines through argument lists. The use of the modules (a feature of FORTRAN 90) makes the design of the physics interface easier. A three-level structure (solver, driver, and individual scheme) of physics (Figure 1) has been constructed. The idea of three-level structure is hoping that users can easily implement their schemes (individual scheme) into the WRF with less involvement of other parts of WRF code.

### 3.1 Level 1 - Solver

The first level, solver (solve_rk.F), is the main subroutine of calling dynamic and physics as the schematic diagram shown in Figure 2. The solver is a bridge, which connects dynamics and physics. In addition, the parallel processes regarding the model part, such as multi threading and message passing, are also dealt with in this subroutine.

### 3.2 Level 2 - Driver

Instead of calling different physics schemes directly, the solver calls the physics drivers (Figure 2), which are interfaces between the solver and individual physics scheme. Except sub-grid eddy diffusion, every physics component has its own driver or shared driver, such as microphysics_driver, cumulus_driver, pbl_driver, and radiation_driver. The pbl_driver includes the surface layer, sub-surface layers, and boundary layer, and the radiation_driver includes long wave and short wave radiation. In each driver, one or few CASE SELECT structures are used to determine the choices of different schemes (Figure 1), and the flags for choosing different physics schemes are listed in the namelist.input file. The design of the driver interface is to provide a friendly environment for users to plug in their own physics packages.

### 3.3 Level 3 - Individual physics scheme.

The prohibition of using common blocks separates users' codes from the WRF code quite well, and it allows users to easily implement their packages into WRF, though some minor changes of the WRF code might be required. One module comes with each physics scheme, and all the subroutines related to the scheme should be included into this module. The details are given in Section 4.

## 4. IMPLEMENTING A NEW PHYSICS SCHEME

This section provides some information about the coding rules of WRF physics, and the procedure for implementing a new physics scheme in detail.

### 4.1 Naming rules

Except for subgrid scale diffusion, each physics scheme has its own module, and the module is named as *module_yy_xxx.F. yy* represents different physics components and it is defined as follows.

*ra* is for radiation parameterization,
*bl* is for PBL parameterization,
*cu* is for cumulus parameterization, and
*mp* is for microphysics.

*xxx* is named to match each individual scheme. For example, *module_cu_kf.F* is a cumulus parameterization module, and it includes all subroutines related to the Kain-Fritsch scheme.

Some WRF variables are also named corresponding to the different physics components, such as physics tendencies and the frequencies of calling different physics components. As seen in the WRF code, physics tendencies are named as zzzyyTEN (*e.g.* RTHRATEN, RUBLTEN, RQCCUTEN) where zzz is RU, RV, RTH, RQV, RQC, RQR, etc. STEPyy (*e.g.* STEPCU, STEPRA, STEPCL) is the frequency, in terms of model time steps, of calling physics.

(Note: The physics tendency arrays will remain constant, and add to the other tendencies, at each model time step until the physics is called again after STEPxx time steps. Microphysics and subgrid eddy diffusion have no corresponding STEPyy since they are called every time step. To change the frequencies of calling physics such as radiation, cumulus, and PBL, go to file namelist.input and modify the values of RADT (for radiation), BLDT (for PBL), and CUDT (for cumulus). They are in the unit of minutes.)

## 4.2 Coding rules

(a) As mentioned earlier, no common block is allowed in the WRF model if the scheme is considered for the official version of the WRF. All variables used in subroutines have to pass through argument lists.

(b) FORTRAN 90 is the official language used in the model part and it has some C++ features, such as pointers, user defined data type, and modules.

(c) Before calculating physics, all variables or decoupled variables (e.g. u and v) have to be horizontally interpolated to the column of mass fields (A grid) if it is necessary, but the vertical ones (*e.g.* w) can be still staggered. This also means that some calculated physics tendencies (*e.g.* RUBLTEN and RVBLTEN) need to be interpolated back to horizontal staggered grid later on. Attention has also to be paid to the vertical index order between user's code and the WRF model. In the WRF, kms (smallest number) is the bottom level and kme (largest num-

ber) is the top level. In a user's scheme, if 1 is at the top level, then you have to reverse the order in the k direction when interfacing it. The following is the vertical structure of WRF. Please also see 4.4.a for more information of indices.

```
kme       -    H (no data at this level)
kme     -----  F
kme-1     -    H
kme-1   -----  F
             .
             .
             .
kms+2     -    H
kms+2   -----  F
kms+1     -    H
kms+1   -----  F
kms       -    H (half level, u, v, w, p, T, q, ect.)
kms     -----  F (full level,w), kms is taken as 1.
```

4.3 Implementing a new scheme

Assume that we are interested in implementing a new cumulus parameterization scheme such as the Arakawa scheme, into the WRF model. The procedure is as follows:

(Note: In the following procedure, the names of modules and files and the pieces of WRF code are italic. In addition, new code regarding the new scheme is bold.)

(a) Create a new cumulus module *module_cu_arakawa.F.* (The pseudo codes of different physics components are available in Appendix B). All subroutines related to the new scheme should be included inside this module such as the scheme itself and its initialization. The scheme-related local constants, which will not be used in other schemes, have to place at the top of the module.

(b) Declare a new package in the file *Registry*, which is in the directory Registry. For example,

*package kfscheme cu_physics==1 - -*

***package arakawascheme cu_physics==2 - -***

Package is a new defined data type. The first line is the declaration of an existing Kain-Fritsch scheme package, and the second bold line is the declaration of the new scheme package. In the namelist, ***arakawascheme*** corresponds to the number 2, and the string ***arakawascheme*** should be used to represent the scheme itself (examples are given later) instead of using the number 2. Currently, there are seven different categories of physics packages defined in the model:

| | |
|---|---|
| cu_physics | is for cumulus parameterization. |
| mp_physics | is for microphysics. |
| ra_lw_physics | is for long wave radiation. |
| ra_sw_physics | is for short wave radiation. |
| bl_sfclay_physics | is for surface layer. |
| bl_surface_physics | is for land-surface layer. |
| bl_pbl_physics | is for boundary layer. |

The declaration of the mp_physics is slightly different from the others. For example, the Kessler scheme package in the Registry is defined:

*package kesslerscheme mp_physics==1 - moist:qv,qc,qr*

The fifth column (moist:qv,qc,qr) contains the moisture fields required in this scheme. This also implies that only these moisture fields are active (memory is allocated) in the model and also available for other physics parameterization schemes, such as cumulus parameterization, radiation, and pbl parameterization.

(Note: A *README* file in the directory WRFMODEL/REGISTRY introduces the detail of the Registry file.)

(c) Reassign *cu_physics = 2* in the file *namelist.input* in order to choose the new scheme. After compiling the model, the schemes corresponding to the different numbers can also be found in the file *module_state_description.F.* The choices of different schemes for different physics components are also in the *namelist.input* such as, mp_physics, ra_lw_physics, ra_sw_physics, bl_sfclay_physics, bl_surface_physics, bl_pbl_physics, cu_physics. Their

definitions are the same as those in the file *Registry* (The third column of the package declaration).

(d) Initialize initial values for the new scheme called from *phy_init* in *module_start.F* if it is necessary. Otherwise, skip this step.

(i) Go to subroutine *cu_init* in *module_start.F* (*ra_init* for radiation initialization, *bl_init* for PBL initialization, and *mp_init* for microphysics initialization).

(ii) Add a new case statement in SELECT CASE, such as:

*cps_loop: SELECT CASE(config_flags%cu_physics)*
*CASE (KFSCHEME)*
*CALL kfinit(...)*
***CASE (ARAKAWASCHEME)***
***CALL arakawainit(...)*** <-- include this subroutine inside the new module *(module_cu_arakawa.F)*
*CASE DEFAULT*
*ENDSELECT cps_loop*

The subroutine ***arakawainit*** should be included inside the new module of the new scheme (*module_cu_arakawa.F*). As mentioned earlier, the string "***ARAKAWASCHEME***" has to be used (with case free) instead of the number, and it spells exactly the same as the one in the Registry (the second column of the package declaration).

(e) Make some changes in the driver *cumulus_driver.F.* (*microphysics_driver.F* for microphysics, *pbl_driver.F* for PBL, and *radiation_driver.F* for radiation.)

(i) Include the new module into the driver such as:

*USE module_cu_kf*
***USE module_cu_arakawa*** <--- new scheme

(ii) Check the available variables passed through argument list. The meanings and units of variables are explained inside each driver. If there is any variable, which is not available in this driver, the user needs to go one level up (*solve_rk.F)* and pass the variable through the driver argument list. Meanwhile, make the same change to the interface file *physics.int* . If the new required variable is not available in the model, the user has to declare a new

variable in the *Registry* (it will magically show up in the file *solve_rk.F*) or just use any inactive working space/variable and then, calculate it in the subroutine *phy_prep*, which is called by *solve_rk.F*, and pass it to driver.

(iii) Check all global constants defined in the *module_model_constants.F.* If there is any global constant, which is not defined yet, please add the new one into the file *module_model_constants.F* and it will show up in the driver (*cumulus_driver.F*) subroutines since the module *module_model_constants* is included inside the drivers. Only use this for constants that might be needed in more than one physics package. Otherwise, define constants at the top of the new physics module.

(iv) Create a new SELECT CASE in *cumulus_driver.F*

```
cps_loop: SELECT CASE(config_flags%cu_physics)
          CASE (KFSCHEME)
               CALL KFCPS(...)
          CASE (ARAKAWASCHEME)    <--- new scheme
               CALL arakawa(...) <--- include this subroutine inside the new
                                      module  (module_cu_arakawa.F)
          CASE DEFAULT
END SELECT cps_loop
```

Again, the string "***ARAKAWASCHEME***" should match the one in the *Registry* (second column) you have defined.

(f) Check physics tendencies that currently exist in the file *Registry.* Create new tendency variables in the *Registry* if it is necessary. They will show up in the file *solve_rk.F.*

(g) Update physics tendencies to non-advective tendencies.

If there is any new created or new calculated tendencies, those tendencies have to be passed into subroutine *update_phy_ten* (called by *solve_rk*) and passed down to the bottom of the calling tree of subroutines such as *phy_cu_ten* for cumulus parameterization (*phy_ra_ten* for radiation and *phy_bl_ten* for PBL). Then, create a new case statement in SELECT CASE, such as:

*SELECT CASE (config_flags%cu_physics)*

*CASE (KFSCHEME)*

*CALL add_a2a(rt_tendf,RTHCUTEN, ...)*

*CALL add_a2a(moist_tendf(ims,kms,jms,P_QV),RQVCUTEN, ...)*

*.*

***CASE(ARAKAWASCHEME)***

***CALL add_a2a(rt_tendf,RTHCUTEN, ...)***

***CALL add_a2a(moist_tendf(ims,kms,jms,P_QV),RQVCUTEN, ...)***

*CASE DEFAULT*

*END SELECT*

Every physics tendency in the new scheme has to accumulate into the non-advective tendencies, such as *rt_tendf, moist_tendf,* and so on, by means of a series of calls (*CALL add_a2a, CAll add_a2c_u, or CALL add_a2c_v*). The subroutine *add_a2a* is used to add the physics tendencies into the non-advective tendencies on the same grid points such as moisture, temperature, and vertical velocity fields. The subroutine *add_a2c_u* (*add_a2c_v*) is used to add the rho_u (rho_v) physics tendencies, which is on A grid, to rho_u (rho_v) non-advective tendencies, which are on the C grid. Please also read 4.4d and 4.4g for some other details.

(Note: Currently microphysics is treated differently from other physics. The other physics drivers are called before the sound-wave short steps to allow diabatic terms to be used for theta on the short steps. The microphysics driver is called at the end of the time step after all the variables have been updated. This means that the microphysics directly updates the variables and has no associated tendencies. This is done at the end to allow an accurate saturation adjustment, and the diabatic term is saved for the next time-step's short steps as an approximation. [In the future we could separate the saturation adjustment from the rest of the microphysics so that we can have the microphysics treated like other physics.])

(h) Add the new module into the *Makefile* which is in the directory src and modify the dependencies.

4.4 General notes

(a) The example codes show three sets of indices such as

```
...
ids,ide, jds,jde, kds,kde,          &
ims,ime, jms,jme, kms,kme,          &
its,ite, jts,jte, kts,kte           )
```

The first set (*ids*, ...) refer to domain start and end, and should only be used for boundary checks (not often needed in the physics). The second set (*ims*, ...) refer to memory dimensions and are distinct from the first set to enable distributed memory. Arrays passed in from the solver are dimensioned by these. The third set (*its*, ...) refer to tile dimensions, and differ from the domain dimensions when there is multiple threading. Local arrays inside the physics generally need only to be dimensioned by the tile size, *e.g.* (*its:ite, kts:kte*), if the physics routine is called inside a j loop (*do j=jts,jte*). All loops should use these limits for *i, j,* and *k,* and operations should not affect array elements outside these limits.

(b) The index order for WRF is IKJ. Physics code can conform to this by calling the lower level routines inside a J loop and having local variables dimensioned (*i,k*), and then having an inner i-loop allowing vectorization (see XXX2D example for pbl scheme). Column physics is also allowed but discouraged because of its usually poor vector performance and non-compliance with the WRF storage and loop-order standards. It requires local (k) arrays in the physics routines (XXX1D examples). Array syntax is used in only a few limited situations.

(c) Note that *module_model_constants* contains many of the basic physical constants. The physics driver uses this module and passes selected constants to the physics routines. It is recommended to use these constants as much as possible, and not to define alternative values inside the physics module. This also applies to the saturation vapor pressure calculation.

(d) The Registry can also be used to add variables required at the solver level. For instance, if you have a cumulus scheme with momentum transport, you will have to add *RUCUTEN* and

*RVCUTEN* to the Registry, and make additions to the routine *phy_cu_ten*. Recall also that momentum tendencies in the physics are calculated on the A-grid and interpolated back to the C-grid (*e.g.* see *phy_bl_ten*).

(e) The Registry can also be used to add species for moisture and chemistry. Moisture variables are carried in WRF as a 4d array (*i,k,j,im*) where *im* denotes species number. This is general enough to add any number of species. Note that any new additions in moist will be advected and diffused and will contribute to total density. The 4d array exists down to the physics driver level and is separated into 3d moisture arrays such as moist(*ims,kms,jms,P_QV*) by use of the P_Qx constants in the CALL from the driver. P_QV, P_QC, P_QR, P_QI, P_QS, P_QG indicate water vapor, cloud water, rain, cloud ice, snow, and graupel, respectively.

(f) The 4d moisture array is not suitable for number concentration or "moment" variables, but these can possibly use the chemical scalar array, or we may add a general scalar array. Turbulent kinetic energy and tracers would also be examples of scalars that do not contribute to mass. More information on the *Registry* is in the WRFMODEL/*README* file.

(g) Horizontally dependent physics such as diffusion has some special considerations. All operations should be on variables within the tile. They can use values outside the tile, but it should be ensured that those values are accounted for in the distributed-memory exchanges of "halo" zones around each patch. This will need expert consultation for specific cases that go beyond the halo zones WRF currently has.

## 5. SQUALL-LINE SIMULATION

A few cases have been tested to ensure the model behavior properly with physics, such as two-dimensional squall line, three-dimensional supercell, and baroclinic waves. Here, only the two-dimensional squall-line case is presented. The horizontally homogeneous sounding is applied to the whole domain as shown in Figure 3. A resolution of 250 m is applied to both horizontal and vertical directions and a time step of 3 seconds is utilized. A thermal bubble with the radius of 4 km and the maximum perturbation of 3 K is imposed in the lower troposphere. The horizontal domain is 50 km, and the vertical is 20 km. Four runs are tested with different combi-

nation of diffusion schemes (simple diffusion and TKE turbulence scheme) and microphysics schemes (Lin and Kessler). The model integrates for 1 hour for each test.

Figure 4 show the vertical velocities of four different runs after 1-hour integration. The cloud patterns are quite different when different diffusion schemes are applied (Figures 4a vs. 4b, and Figures 4c vs. 4d). The TKE turbulence scheme maintains the squall-line structure better than the simple diffusion scheme does since the TKE scheme tries to mix air locally, while the other one diffuses globally. The figures also indicate that the squall line develops more strongly with the Lin scheme, which includes the ice phase, than that with Kessler scheme, which excludes the ice phase. In general, these simulation results are quite reasonable.

## 6. SUMMARY

The Weather Research and Forecast (WRF) model is a next-generation mesoscale model and it is developed by a group of scientists in a number of centers and institutes. The objectives of this project are to carry out the implementation of a basic suite of physics and to design a user friendly interface for implementing a new physics scheme into the WRF. In this early stage of model development, only existing physics schemes are implemented, and most of them are taken from existing models or packages. Those pre-existing codes have been modified properly in order to fit the model dynamics code and architecture. Currently, the implemented physics schemes include:

| | |
|---|---|
| Microphysics | : Kessler, Lin, and NCEP simple ice schemes |
| Convective parameterization | : Kain-Fritsch, and Betts-Miller-Janjic schemes |
| Long wave radiation | : RRTM (Rapid Radiative Transfer Model) |
| Short wave radiation | : Simple short wave and Goddard short wave |
| Surface layer | : Similarity theory |
| Land-surface layer | : Thermal diffusion, and Oregon State-Eta |
| Boundary layer | : MRF |
| Subgrid eddy diffusion | : Simple diffusion, and Stress/deformation form |

The model portion of the WRF is coded in FORTRAN90, and no common block is allowed. Some naming rules are designed to make the use of the model physics more friendly and clear. A

three-level physics structure (solver, driver, and individual physics scheme) is constructed to make the implementation of physics schemes easier, and it is introduced in Section 4 in great detail.

A few cases have been tested to ensure the model perform properly with physics. The squall-line is the only one presented here, and the others are available in WRF web site (http:// wrf-model.org).

ACKNOWLEDGEMENT

The authors would like to thank Drs. John Brown, Tom Black, Song-You Hong, Wei-Kuo Tao, Ming Xue, Bill Skamarock, John Michalakes, Dave Gill, Joe Klemp, and Chris Davis for their discussion and suggestions. Thanks are also extended to Mike Farrar and Jerry Wegiel at AFWA for their strong suppots of this project.

REFERENCES


Betts, A. K., 1986: A new convective adjustment scheme. Part I: Observational and theoretical basis. *Quart. J. Roy. Meteor. Soc.*, **112**, 677-691.

Betts, A. K., and M. J. Miller, 1986: A new convective adjustment scheme. Part II: Single column tests using GATE wave, BOMEX, and arctic air-mass data sets. *Quart. J. Roy. Meteor. Soc.*, **112**, 693-709.

Chen, F., and J. Dudia, 2000: Coupling an advanced land-surface/hydrology model with the Penn State/NCAR MM5 modeling system. Part I: Model description and implementation. *Mon. Wea. Rev.*, in press.

Chen, S.-H., 1994: One-dimensional time dependent cloud model. M.S. thesis, Purdue University 105pp.

Chou M.-D., and M. J. Suarez, 1994: An efficient thermal infrared radiation parameterization for use in general circulation models. NASA Tech. Memo. 104606, **3**, 85pp.

Dudia, J., 1989: Numerical study of convection observed during the winter monsoon experiment using a mesoscale two-dimensional model. *J. Atmos. Sci*., **46**, 3077-3107.

Dyer, A. J., and B. B. Hicks, 1970: Flux-gradient relationships in the constant flux layer. *Quart. J. Roy. Meteor. Soc.*, **96**, 715-721.

Hong, S.-Y., and H.-L. Pan, 1996: Nonlocal boundary layer vertical diffusion in a medium-range forecast model. *Mon. Wea. Rev.*, **124**, 2322-2339.

Hong, S.-Y., H.-M. H. Juang, and Q. Zhao, 1998: Implementation of prognostic cloud scheme for a regional spectral model, *Mon. Wea. Rev.*, **126**, 2621-2639.

Janjic, Z. I., 1994: The step-mountain eta coordinate model: Further developments of the convection, viscous sublayer, and turbulence closure schemes. *Mon. Wea. Rev.*, **121**, 927-945.

Kain, J. S., and J. M. Fritsch, 1990: A one-dimensional entraining/detraining plume model and its application in convective parameterization. *J. Atmos. Sci*., **47**, 2784-2802.

Kessler, E., 1969: *On the distribution and continuity of water substance in atmospheric circulation. Meteor. Monogr.,* No. 32, Amer. Meteor. Soc., 84pp.

Klemp, J. B., and R. B. Wilhelmson, 1978: The simulation of three-dimensional convective storm dynamics, J. Atmos. Sci., **35**, 1070-1096.

Lacis, A. A., and J. E. Hansen, 1974: A parameterization for the absorption of solar radiation in the earth's atmosphere. *J. Atmos. Sci*., **31**, 118-133.

Lin, Y.-L., R. D. Farley, and H. D. Orville, 1983: Bulk parameterization of the snow field in a cloud model. *J. Climate Appl. Meteor.,* **22**, 1065-1092.

Moeng, C.-H., 1984: A large-eddy-simulation model for the study of planetary boundary-layer turbulence. *J. Atmos. Sci*., **41**, 2052-2062.

Moeng, C.-H., and J. C. Wyngaard, 1989: Evaluation of turbulence and dissipation closures in second-order modeling. *J. Atmos. Sci*., **46**, 23112330.

Mlawer, E. J., S. J. Taubman, P. D. Brown, M. J. Iacono, and S. A.Clough, 1997: Radiative transfer for inhomogeneous atmosphere: RRTM, a validated correlated-k model for the longwave. *J. Geophys. Res.*, **102**(D14), 16663-16682.

Paulson, C. A., 1970: The mathematical representation of wind speed and temperature profiles in the unstable atmospheric surface layer. *J. Appl. Meteor.*, **9**, 857-861.

Rutledge, S. A., and P. V. Hobbs, 1984: The mesoscale and microscale structure and organization of clouds and precipitation in midlatitude cyclones. XII: A diagnostic modeling study of precipitation development in narrow cloud-frontal rainbands. *J. Atmos. Sci.*, **20**, 2949-2972.

Stephens, G. L., 1978: Radiation profiles in extended water clouds. Part II: Parameterization schemes. *J. Atmos. Sci*., **35**, 2123-2132.

Tao, W.-K., 1989: An ice-water saturation adjustment. *Mon. Wea. Rev.*, **117**, 231-235.

Webb, E. K., 1970: Profile relationships: The log-linear range, and extension to strong stability. *Quart. J. Roy. Meteor. Soc.*, **96**, 67-90.

Wicker, L. J., and W. Skamarock, 1998: A time-splitting scheme for the elastic equations incorporating second-order Runge-Kutta time differencing, *Mon. Wea. Rev.*, 126, 1992-1999.

# APPENDIX A

## The derivation of the eddy diffusion is as follows:

Define $U = \rho u$, and the diffusion equation of $U$ is expressed as:

$$\frac{\partial U}{\partial t} = -\frac{\partial \rho \overline{u'u'}}{\partial x} - \frac{\partial \rho \overline{v'u'}}{\partial y} - \frac{\partial \rho \overline{w'u'}}{\partial z}.$$

Add Map-Scale Factor.

$$\frac{\partial U}{\partial t} = -m^2 \frac{\partial \tau_{11}/m}{\partial x} - m^2 \frac{\partial \tau_{12}/m}{\partial y} - \frac{\partial \tau_{13}}{\partial z}.$$

where

$$\tau_{11} = \rho \overline{u'u'},$$

$$\tau_{12} = \rho \overline{v'u'},$$

$$\tau_{13} = \rho \overline{w'u'}$$

Define $\hat{\tau} = \tau/m$, and redefine $U = \rho u/m$.

$$\frac{\partial U}{\partial t} = -m \frac{\partial \hat{\tau}_{11}}{\partial x} - m \frac{\partial \hat{\tau}_{12}}{\partial y} - \frac{\partial \hat{\tau}_{13}}{\partial z}.$$

Let

$$\zeta = \frac{z - h(x)}{(H - h(x))},$$

where H is the model top, and $h(x)$ is the terrain elevation.

$$\zeta_z = \frac{1}{(H - h(x))}.$$

Define $\tilde{\tau} = \tau/m/\zeta_z$ and redefine $U = \rho u/m/\zeta_z$.

$$\frac{\partial U}{\partial t} = -\frac{m}{\zeta_z} \frac{\partial \zeta_z \tilde{\tau}_{11}}{\partial x} - \frac{m}{\zeta_z} \frac{\partial \zeta_z \tilde{\tau}_{12}}{\partial y} - \frac{\partial \tilde{\tau}_{13}}{\partial z}.$$

After coordinates transformation from *(x, y, z)* to *(X, Y, ζ)*, we can get:

$$\frac{\partial U}{\partial t} = -m \frac{\partial \tilde{\tau}_{11}}{\partial X} - m \frac{\partial \zeta_x \tilde{\tau}_{11}}{\partial \zeta} - m \frac{\partial \tilde{\tau}_{12}}{\partial Y} - m \frac{\partial \zeta_y \tilde{\tau}_{12}}{\partial \zeta} - \frac{\partial \zeta_z \tilde{\tau}_{13}}{\partial \zeta}.$$

$$\frac{\partial U}{\partial t} = -m \frac{\partial \tilde{\tau}_{11}}{\partial X} - m \frac{\partial \tilde{\tau}_{12}}{\partial Y} - \frac{\partial}{\partial \zeta} \Big[ \big( m\zeta_x \tilde{\tau}_{11} + m\zeta_y \tilde{\tau}_{12} + \zeta_z \tilde{\tau}_{13} \big) \Big].$$

Same processes applied to $V$, and $W$.

$$\frac{\partial V}{\partial t} = -m\frac{\partial \tilde{\tau}_{21}}{\partial X} - m\frac{\partial \tilde{\tau}_{22}}{\partial Y} - \frac{\partial}{\partial \zeta}\Big[\big(m\zeta_x\tilde{\tau}_{21} + m\zeta_y\tilde{\tau}_{22} + \zeta_z\tilde{\tau}_{23}\big)\Big],$$

$$\frac{\partial W}{\partial t} = -m\frac{\partial \tilde{\tau}_{31}}{\partial X} - m\frac{\partial \tilde{\tau}_{32}}{\partial Y} - \frac{\partial}{\partial \zeta}\Big[\big(m\zeta_x\tilde{\tau}_{31} + m\zeta_y\tilde{\tau}_{32} + \zeta_z\tilde{\tau}_{33}\big)\Big],$$

where

$$\begin{aligned}
\tilde{\tau}_{11} &= \tilde{\rho}\Big[2/3\ e - K_{mh}\big(D_{11} - \frac{2}{3}\ Div\big)\Big],\\
\tilde{\tau}_{12} &= -\tilde{\rho}K_{mh}D_{12},\\
\tilde{\tau}_{13} &= -\tilde{\rho}K_{mv}D_{13},\\
\tilde{\tau}_{21} &= -\tilde{\rho}K_{mh}D_{12},\\
\tilde{\tau}_{22} &= \tilde{\rho}\Big[2/3\ e - K_{mh}\big(D_{22} - \frac{2}{3}\ Div\big)\Big],\\
\tilde{\tau}_{23} &= -\tilde{\rho}K_{mv}D_{23},\\
\tilde{\tau}_{31} &= -\tilde{\rho}K_{mh}D_{13},\\
\tilde{\tau}_{32} &= -\tilde{\rho}K_{mh}D_{23},\\
\tilde{\tau}_{33} &= \tilde{\rho}\Big[2/3\ e - K_{mv}\big(D_{33} - \frac{2}{3}\ Div\big)\Big].
\end{aligned}$$

$\tilde{\rho} = \rho/m/\zeta_z$ and $e$ is the TKE and defined as $(\overline{u'u'} + \overline{v'v'} + \overline{w'w'})/2$.

$$D_{11} = 2m^2\frac{\partial \frac{u}{m}}{\partial x} = 2m^2\frac{\partial \hat{u}}{\partial x} = 2m^2\Big(\frac{\partial \hat{u}}{\partial X} + \zeta_x\frac{\partial \hat{u}}{\partial \zeta}\Big),$$

$$D_{22} = 2m^2\frac{\partial \frac{v}{m}}{\partial y} = 2m^2\Big(\frac{\partial \hat{v}}{\partial Y} + \zeta_y\frac{\partial \hat{v}}{\partial \zeta}\Big),$$

$$D_{33} = 2\frac{\partial w}{\partial z} = 2\zeta_z\big(\frac{\partial w}{\partial \zeta}\big),$$

$$D_{12} = m^2\Big(\frac{\partial \frac{v}{m}}{\partial x} + \frac{\partial \frac{u}{m}}{\partial y}\Big) = m^2\Big(\frac{\partial \hat{v}}{\partial X} + \frac{\partial \hat{u}}{\partial Y} + \zeta_x\frac{\partial \hat{v}}{\partial \zeta} + \zeta_y\frac{\partial \hat{u}}{\partial \zeta}\Big)$$

$$D_{13} = m^2\frac{\partial \frac{w}{m}}{\partial x} + \frac{\partial u}{\partial z} = m^2\Big(\frac{\partial \hat{w}}{\partial X} + \zeta_x\frac{\partial \hat{w}}{\partial \zeta}\Big) + \zeta_z\frac{\partial u}{\partial \zeta}$$

$$D_{23} = m^2\frac{\partial \frac{w}{m}}{\partial y} + \frac{\partial v}{\partial z} = m^2\Big(\frac{\partial \hat{w}}{\partial Y} + \zeta_y\frac{\partial \hat{w}}{\partial \zeta}\Big) + \zeta_z\frac{\partial v}{\partial \zeta}$$

where $\hat{u} = u/m$, $\hat{v} = v/m$, and $\hat{w} = w/m$

$$\begin{aligned} Div &= m^2 \frac{\partial \frac{u}{m}}{\partial x} + m^2 \frac{\partial \frac{v}{m}}{\partial y} + \frac{\partial w}{\partial z} \\ &= m^2 \frac{\partial \hat{u}}{\partial X} + m^2 \frac{\partial \hat{v}}{\partial Y} + m^2 \zeta_x \frac{\partial \hat{u}}{\partial \zeta_z} + m^2 \zeta_y \frac{\partial \hat{v}}{\partial \zeta_z} + \zeta_z \frac{\partial w}{\partial \zeta} \end{aligned}$$

For any scalar $\phi$,

$$\tilde{\rho} \frac{\partial \phi}{\partial t} = -m \frac{\partial \tilde{\rho} H_1(\phi)}{\partial X} - m \frac{\partial \tilde{\rho} H_2(\phi)}{\partial Y} - \frac{\partial}{\partial \zeta} \Big[ \tilde{\rho} \big( m \zeta_x H_1(\phi) + m \zeta_y H_2(\phi) + \zeta_z H_3(\phi) \big) \Big],$$

where

$$\begin{aligned} H_1(\phi) &= -m K_{Hh} \frac{\partial \phi}{\partial x} = -m K_{Hh} \Big( \frac{\partial \phi}{\partial X} + \zeta_x \frac{\partial \phi}{\partial \zeta} \Big), \\ H_2(\phi) &= -m K_{Hh} \frac{\partial \phi}{\partial y} = -m K_{Hh} \Big( \frac{\partial \phi}{\partial Y} + \zeta_y \frac{\partial \phi}{\partial \zeta} \Big), \\ H_3(\phi) &= -K_{Hv} \frac{\partial \phi}{\partial z} = -K_{Hv} \zeta_z \frac{\partial \phi}{\partial \zeta}. \end{aligned}$$

$\Theta_m (= \tilde{\rho}_d \theta (1 + \frac{\rho_d}{\rho_M} q_v))$ equation,

$$\frac{\partial \Theta_m}{\partial t} = (1 + \frac{R_v}{R_d} q_v) \tilde{\rho} \frac{\partial \theta}{\partial t} + \theta \frac{R_v}{R_d} \tilde{\rho} \frac{\partial q_v}{\partial t}$$

TKE equation (e)

$$\frac{\partial e}{\partial t} = -u_i \frac{\partial e}{\partial x_i} - \frac{\partial \overline{u_i' \big( e' + \frac{p'}{\rho_0} \big)}}{\partial x_i} - \overline{u_i' u_j'} \frac{\partial u_i}{\partial x_j} + \frac{g}{\theta_0} \overline{w' \theta'} - Dissp$$

Define $E = \rho e / m / \zeta_z$. The flux form of the TKE is:

$$\begin{aligned} \frac{\partial E}{\partial t} =& - m \frac{\partial U^t e^t}{\partial X} - m \frac{\partial V^t e^t}{\partial Y} - \frac{\partial \Omega^t e^t}{\partial \zeta} \\ & - 2m \frac{\partial \tilde{\rho} H_1(e)}{\partial X} - 2m \frac{\partial \tilde{\rho} H_2(e)}{\partial Y} - 2 \frac{\partial}{\partial \zeta} \Big[ \tilde{\rho} \big( m \zeta_x H_1(e) + m \zeta_y H_2(e) + \zeta_z H_3(e) \big) \Big] \\ & - \Big[ m \tilde{\tau}_{11} \Big( \frac{\partial u}{\partial X} + \zeta_x \frac{\partial u}{\partial \zeta} \Big) + m \tilde{\tau}_{12} \Big( \frac{\partial u}{\partial Y} + \zeta_y \frac{\partial u}{\partial \zeta} \Big) + \tilde{\tau}_{13} \zeta_z \frac{\partial u}{\partial \zeta} + \end{aligned}$$

$$m\tilde{\tau}_{21}\Big(\frac{\partial v}{\partial X}+\zeta_x\frac{\partial v}{\partial \zeta}\Big)+m\tilde{\tau}_{22}\Big(\frac{\partial v}{\partial Y}+\zeta_y\frac{\partial v}{\partial \zeta}\Big)+\tilde{\tau}_{23}\zeta_z\frac{\partial v}{\partial \zeta}+$$
$$m\tilde{\tau}_{31}\Big(\frac{\partial w}{\partial X}+\zeta_x\frac{\partial w}{\partial \zeta}\Big)+m\tilde{\tau}_{32}\Big(\frac{\partial w}{\partial Y}+\zeta_y\frac{\partial w}{\partial \zeta}\Big)+\tilde{\tau}_{33}\zeta_z\frac{\partial w}{\partial \zeta}\Big]$$
$$+\tilde{\rho}B-\tilde{\rho}Dissp$$

where

$$B=\begin{cases}-gK_{Hv}\zeta_z\left[A\frac{\partial\theta_e}{\partial\zeta}-\frac{\partial(q_v+q_c+q_r+q_i+q_s+q_g)}{\partial\zeta}\right] & \text{for } q_v \geq q_s \text{ or } q_c \geq 0.01\text{g/Kg}\\ -gK_{Hv}\zeta_z\left[\frac{1}{\theta}\frac{\partial\theta}{\partial\zeta}+0.61\frac{\partial q_v}{\partial\zeta}-\frac{\partial(q_c+q_r+q_i+q_s+q_g)}{\partial\zeta}\right] & \text{for } q_v < q_s \text{ or } q_c < 0.01\text{g/Kg}\end{cases}$$

A is defined as

$$A=\frac{1}{\theta}\frac{\frac{1+1.61\varepsilon L q_v}{RT}}{\frac{1+\varepsilon L^2 q_v}{C_p R T^2}}$$

$$\varepsilon=0.622$$

$$K_{mh}=0.1e^{1/2}l_h$$

$$K_{mv}=0.1e^{1/2}l_v$$

If $Ri$ $(=N^2/|Def|^2)$ is less than a small number, then

$$K_{mh}=max\big(0.1e^{1/2}l_h,\ \alpha\Delta s_h^2\big)$$
$$K_{mv}=max\big(0.1e^{1/2}l_v,\ \alpha\Delta s_v^2\big)$$

where $\alpha=10^{-6}$

When $\Delta x$, $\Delta y$, and $\Delta z$ << dominant turbulence size

$$Dissp=\frac{Ce^{3/2}}{l}$$

$$C=0.19+\Big(0.74\ \frac{l}{\Delta s}\Big)$$

$$l=l_h=l_v=\begin{cases}=\Delta s=(\Delta x\Delta y\Delta z)^{1/3} & \text{neutral or unstable}\\ =min\big(\Delta s,\ l_s\big) & \text{stable}\end{cases}$$

$$l_s=0.76\ e^{1/2}\Big|\frac{g}{\theta}\frac{\partial\theta}{\partial z}\Big|^{-1/2}$$

When $\Delta x$ and $\Delta y >$ dominant turbulence size

$$Dissp = \frac{2\sqrt{2}}{15}\frac{e^{3/2}}{l}$$

$$l = \frac{kz}{1 + kz/l_0}$$

$$l_0 = min\Big(\frac{\alpha_b \int_0^{z_i} \sqrt{e} z dz}{\int_0^{z_i} \sqrt{e} dz},\ 80\Big)$$

where $\alpha_b = 0.2$

$$l_h = \Delta s_h = (\Delta x \Delta y)^{1/2}, \quad \text{and}$$

$$l_v = \begin{cases} = \Delta s_v = \Delta z & \text{neutral or unstable} \\ = min\big(\Delta s_v,\ l_s\big) & \text{stable} \end{cases}$$

$$K_H = \Big(1 + \frac{2l_v}{\Delta s_v}\Big) K_m$$

# APPENDIX B

To take a look of the exist physics modules is definitely a good starting point for preparing a new code and to plug the new scheme into the WRF. Four pseudo codes - module_ra_xxx.F, module_bl_xxx.F, module_cu_xxx.F, and module_mp_xxx.F - are provided for different physics as follows:

### module_ra_xxx.F (radiation parameterization)

```
MODULE module_ra_xxx

!!! put scheme related parameters or constants here

CONTAINS
!-------------------------------------------------------------------------------------------------------------
  SUBROUTINE XXX(RTHRATEN,....QV,pi,rr,.....,rvovrd,...                          &
                           ids,ide, jds,jde, kds,kde,                           &
                           ims,ime, jms,jme, kms,kme,                           &
                           its,ite, jts,jte, kts,kte                            )
!-------------------------------------------------------------------------------------------------------------
  IMPLICIT NONE
!-------------------------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN )      ::  ids,ide, jds,jde,kds,kde,ims,ime, jms,jme,kms,kme, &
                                 its,ite, jts,jte, kts,kte
  REAL,      INTENT(IN   )    ::  rvovrd
  REAL,      DIMENSION( ims:ime , kms:kme , jms:jme ), INTENT(IN   ) ::          &
             QV, pi, rr, . .
  REAL,      DIMENSION( ims:ime, kms:kme, jms:jme ),  INTENT(INOUT)  ::       &
             RTHRATEN

!  LOCAL VARIABLES (use tile size)
  REAL,    DIMENSION( its:ite, kts:kte ) :: TTEN2D
  INTEGER:: i,j,K
!-------------------------------------------------------------------------------------------------------------
  DO J=jts,jte

      ! either use ik slice or just use 3D array
      ! be careful about vertical order

      CALL XXX2D(TTEN2D,QV(ims,kms,J),.....                                    &
                           ids,ide, jds,jde, kds,kde,                          &
                           ims,ime, jms,jme, kms,kme,                          &
                           its,ite, jts,jte, kts,kte                           )

  ENDDO

  DO j=jts,jte
     DO k=kts,kte
```

```
          DO i=its,ite
              RTHRATEN(I,K,J)=RTHRATEN(I,K,J)+rr(I,K,J)*   &
                                (1.+rvovrd*QV(I,K,J))* TTEN2D(I,K)/pi(I,K,J)
          ENDDO
       ENDDO
  ENDDO

  END SUBROUTINE XXX

!=================================================================
  SUBROUTINE XXX2D(TTEN2D,QV2D,.....                                  &
                              ids,ide, jds,jde, kds,kde,              &
                              ims,ime, jms,jme, kms,kme,              &
                              its,ite, jts,jte, kts,kte,              )
!-----------------------------------------------------------------------------------------------
    IMPLICIT NONE
!-----------------------------------------------------------------------------------------------
  INTEGER,  INTENT(IN   )   :: ids,ide,jds,jde,kds,kde,ims,ime,jms,jme, kms,kme,  &
                               its,ite, jts,jte, kts,kte
  REAL,    DIMENSION( ims:ime,kms:kme ), INTENT(IN   ) :: QV2D
  REAL,    DIMENSION( its:ite,kts:kte ), INTENT(INOUT) :: TTEN2D

!  LOCAL VARIABLES (use tile size)
  REAL,     DIMENSION( its:ite, kts:kte ) ::          .....
!-----------------------------------------------------------------------------------------------
         .
!!! Main body of the new scheme
         .
  END SUBROUTINE XXX2D

!=================================================================
  SUBROUTINE xxxinit(RTHRATEN,                                        &
                              ids, ide, jds, jde, kds, kde,           &
                              ims, ime, jms, jme, kms, kme,           &
                              its, ite, jts, jte, kts, kte            )
!-----------------------------------------------------------------------------------------------
  IMPLICIT NONE
!-----------------------------------------------------------------------------------------------
  INTEGER,  INTENT(IN   )   :: ids,ide,jds,jde,kds,kde,ims,ime,jms,jme, kms,kme,   &
                               its,ite, jts,jte, kts,kte
   REAL , DIMENSION( ims: , kms: , jms: ) , INTENT(INOUT) :: RTHRATEN

  INTEGER :: i, j, k, itf, jtf, ktf

  jtf=min0(jte,jde-1)
  ktf=min0(kte,kde-1)
  itf=min0(ite,ide-1)

  DO j=jts,jtf
       DO k=kts,ktf
            DO i=its,itf
                RTHRATEN(i,k,j)=0.
            ENDDO
       ENDDO
```

```
  ENDDO

  END SUBROUTINE xxxinit
END MODULE module_ra_xxx
```

## module_bl_xxx.F (boundary layer parameterization)

```
MODULE module_bl_xxx
!!! put scheme related parameters or constants here

CONTAINS
!==================================================================
  SUBROUTINE XXX(QV,pi,TH,rr,....., RUBLTEN,RVBLTEN,RTHBLTEN,         &
                           RQVBLTEN,RQCBLTEN,RQIBLTEN, rvovrd,P_QI,.....,     &
                            ids,ide, jds,jde, kds,kde,                        &
                            ims,ime, jms,jme, kms,kme,                        &
                            its,ite, jts,jte, kts,kte                          )
!------------------------------------------------------------------------------------------------------
  IMPLICIT NONE
!------------------------------------------------------------------------------------------------------
  INTEGER,  INTENT(IN   )   :: ids,ide,jds,jde,kds,kde,ims,ime,jms,jme, kms,kme,  &
                               its,ite, jts,jte, kts,kte
  INTEGER,  INTENT(IN   )   ::  P_QI
  REAL,       INTENT(IN   )   ::  rvovrd
  REAL,       DIMENSION( ims:ime , kms:kme , jms:jme ),  INTENT(IN   ) ::         &
              QV, pi, TH, rr, ....
  REAL,       DIMENSION( ims:ime, kms:kme, jms:jme ), INTENT(INOUT)   ::     &
              RUBLTEN, RVBLLEN, RTHBLTEN, RQVBLTEN, RQCBLTEN,      &
              RQIBLTEN

! LOCAL VARIABLES (use tile size)
  REAL,          DIMENSION( its:ite, kts:kte ) ::           .....
  INTEGER   ::  I,J,K
!------------------------------------------------------------------------------------------------------
  DO J=jts,jte
    ! either use ik slice or just use 3D array
    ! be careful about vertical order
    CALL XXX2D(......,P_QI,                                                 &
                    RUBLTEN(ims,kms,j),RVBLTEN(ims,kms,j),              &
                    RTHBLTEN(ims,kms,j),RQVBLTEN(ims,kms,j),         &
                    RQCBLTEN(ims,kms,j),RQIBLTEN(ims,kms,j),         &
                    ids,ide, jds,jde, kds,kde,                                &
                    ims,ime, jms,jme, kms,kme,                                &
                    its,ite, jts,jte, kts,kte                                  )
  ENDDO
!
  DO j=jts,jte
      DO k=kts,kte
          DO i=its,ite
              RUBLTEN(I,K,J)=rr(I,K,J)*RUBLTEN(I,K,J)
              RVBLTEN(I,K,J)=rr(I,K,J)*RVBLTEN(I,K,J)
              RTHBLTEN(I,K,J)=rr(I,K,J)* ((1.+rvovrd*QV(I,K,J))*RTHBLTEN(I,K,J)/pi(I,K,J)  &
                              +rvovrd*TH(I,K,J)*RQVBLTEN(I,K,J))
```

```
            RQVBLTEN(I,K,J)=rr(I,K,J)*RQVBLTEN(I,K,J)
            RQCBLTEN(I,K,J)=rr(I,K,J)*RQCBLTEN(I,K,J)
         ENDDO
      ENDDO
  ENDDO

  IF (P_QI .gt. 0) THEN
     DO j=jts,jte
        DO k=kts,kte
           DO i=its,ite
              RQIBLTEN(I,K,J)=rr(I,K,J)*RQIBLTEN(I,K,J)
           ENDDO
        ENDDO
     ENDDO
  ENDIF

  END SUBROUTINE XXX
!=================================================================
  SUBROUTINE XXX2D(.....,P_QI,                                    &
             U2DTEN,V2DTEN,T2DTEN,QV2DTEN,QC2DTEN,QI2DTEN,        &
             ids,ide, jds,jde, kds,kde,                           &
             ims,ime, jms,jme, kms,kme,                           &
             its,ite, jts,jte, kts,kte,                           )
!-----------------------------------------------------------------------------------------------
     IMPLICIT NONE
!-----------------------------------------------------------------------------------------------
     INTEGER,  INTENT(IN   ) :: ids,ide,jds,jde,kds,kde,ims,ime,jms,jme,kms,kme, &
                                its,ite, jts,jte, kts,kte,P_QI
     REAL,       DIMENSION( ims:ime, kms:kme ), INTENT(INOUT)   ::              , &
                 U2DTEN, V2DTEN, T2DTEN, QV2DTEN, QC2DTEN, QI2DTEN

! LOCAL VARIABLES (use tile size)

     REAL,       DIMENSION( its:ite, kts:kte ) ::          .....
!-----------------------------------------------------------------------------------------------
          .

!!! Main body of the new scheme
          .
     IF (P_QI .gt. 0) THEN
        DO k=kts,kte
           DO i=its,ite
              QI2DTEN(I,K)=....
           ENDDO
        ENDDO
     ENDIF
          .
  END SUBROUTINE XXX2D

!=================================================================
  SUBROUTINE xxxinit(RUBLTEN,RVBLTEN,RTHBLTEN,RQVBLTEN,           &
                     RQCBLTEN,RQIBLTEN,P_QI,                      &
                     ids, ide, jds, jde, kds, kde,                &
                     ims, ime, jms, jme, kms, kme,                &
```

```
                                its, ite, jts, jte, kts, kte                                    )
!-------------------------------------------------------------------------------------------------------------
  IMPLICIT NONE
!-------------------------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                                its, ite, jts, jte, kts, kte, P_QI
  REAL ,      DIMENSION( ims: , kms: , jms: ) , INTENT(OUT) ::                    &
              RUBLTEN, RVBLTEN, RTHBLTEN, RQVBLTEN, RQCBLTEN,        &
              RQIBLTEN
  INTEGER :: i, j, k, itf, jtf, ktf

  jtf=min0(jte,jde-1)
  ktf=min0(kte,kde-1)
  itf=min0(ite,ide-1)

  DO j=jts,jtf
      DO k=kts,ktf
          DO i=its,itf
              RUBLTEN(i,k,j)=0.
              RVBLTEN(i,k,j)=0.
              RTHBLTEN(i,k,j)=0.
              RQVBLTEN(i,k,j)=0.
              RQCBLTEN(i,k,j)=0.
          ENDDO
      ENDDO
  ENDDO

  IF (P_QI .gt. 0) THEN
    DO j=jts,jtf
        DO k=kts,ktf
            DO i=its,itf
                RQIBLTEN(i,k,j)=0.
            ENDDO
        ENDDO
    ENDDO
  ENDIF

  END SUBROUTINE xxxinit
!-------------------------------------------------------------------------------------------------------------

END MODULE module_bl_xxx
```

## module_cu_xxx.F  (cumulus parameterization)

```
!!! put scheme related parameters or constants here

CONTAINS
!-------------------------------------------------------------------------------------------------------------
  SUBROUTINE XXX(QV,pi,TH,rr,.....,RTHCUTEN,RQVCUTEN,RQCCUTEN,     &
                              RQRCUTEN,RQICUTEN,RQSCUTEN,RAINCV,              &
                              rvovrd,P_QI,P_QS,.....,                                  &
                              ids,ide, jds,jde, kds,kde,                               &
```

```
                          ims,ime, jms,jme, kms,kme,                                    &
                          its,ite, jts,jte, kts,kte                                      )
!-------------------------------------------------------------------------------------------------------------
  IMPLICIT NONE
!-------------------------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                          its, ite, jts, jte, kts, kte, P_QI, P_QS
  REAL,      INTENT(IN ) :: rvovrd
  REAL,      DIMENSION( ims:ime , kms:kme , jms:jme ), INTENT(IN   ) ::          &
             QV,  pi, TH, rr, ...
  REAL,      DIMENSION( ims:ime , kms:kme , jms:jme ), INTENT(INOUT) ::      &
             RTHCUTEN, RQVCUTEN, RQCCUTEN, RQRCUTEN, RQICUTEN,  &
             RQSCUTEN
  REAL,      DIMENSION( ims:ime , jms:jme ), INTENT(INOUT) ::  RAINCV

! LOCAL VARIABLES (use tile size)
  REAL,      DIMENSION( its:ite,kts:kte ) ::                  ...
  INTEGER :: i,j,k
!-------------------------------------------------------------------------------------------------------------
  DO j = jts,jte

     ! either use ik slice or just use 3D array
     ! be careful about vertical order

!    RAINCV:  mm in one time step

    CALL XXX2D(QV(ims,kms,J),....,P_QI,P_QS,                                 &
                     RQVCUTEN(ims,kms,J),RQCCUTEN(ims,kms,J),        &
                     RQRCUTEN(ims,kms,J),RQICUTEN(ims,kms,J),        &
                     RQSCUTEN(ims,kms,J),RTHCUTEN(ims,kms,J),        &
                     RAINCV(ims,J),...,                                        &
                     ids,ide, jds,jde, kds,kde,                              &
                     ims,ime, jms,jme, kms,kme,                              &
                     its,ite, jts,jte, kts,kte                                )
  ENDDO

  DO j=jts,jte
     DO k=kts,kte
        DO i=its,ite
           RTHCUTEN(I,K,J)=rr(I,K,J)*((1+rvovrd*QV(I,K,J))*RTHCUTEN(I,K,J)/pi(I,K,J) &
                          + rvovrd*TH(I,K,J)*RQVCUTEN(I,K,J))
           RQVCUTEN(I,K,J)=rr(I,K,J)*RQVCUTEN(I,K,J)
           RQCCUTEN(I,K,J)=rr(I,K,J)*RQCCUTEN(I,K,J)
           RQRCUTEN(I,K,J)=rr(I,K,J)*RQRCUTEN(I,K,J)
        ENDDO
     ENDDO
  ENDDO

  IF (P_QI .gt. 0)THEN
    DO j=jts,jte
       DO k=kts,kte
          DO i=its,ite
             RQICUTEN(I,K,J)=rr(I,K,J)*RQICUTEN(I,K,J)
          ENDDO
```

```
          ENDDO
       ENDDO
    ENDIF

    IF (P_QS .gt. 0)THEN
       DO j=jts,jte
          DO k=kts,kte
             DO i=its,ite
                RQSCUTEN(I,K,J)=rr(I,K,J)*RQSCUTEN(I,K,J)
             ENDDO
          ENDDO
       ENDDO
    ENDIF
!
    END SUBROUTINE XXX
!=================================================================
    SUBROUTINE XXX2D(QV2D,....,P_QI,P_QS, DQDT2D,DQCDT2D,DQRDT2D, &
                                DQIDT2D,DQSDT2D,DTDT2D,rain,.....          &
                                ids,ide, jds,jde, kds,kde,                 &
                                ims,ime, jms,jme, kms,kme,                 &
                                its,ite, jts,jte, kts,kte,                  )
!-----------------------------------------------------------------------------------------------------
    IMPLICIT NONE
!-----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                                  its, ite, jts, jte, kts, kte, P_QI, P_QS
    REAL,      DIMENSION( ims:ime, kms:kme ), INTENT(IN ) :: QV2D
    REAL,      DIMENSION( ims:ime, kms:kme ), INTENT(INOUT):: DQDT2D,      &
               DQIDT2D, DQCDT2D, DQRDT2D, DQSDT2D, DTDT2D
    REAL,      DIMENSION( ims:ime ), INTENT(INOUT) :: rain

!  LOCAL VARIABLES (use tile size)
    REAL,    DIMENSION( its:ite, kts:kte ) ::             .....
!-----------------------------------------------------------------------------------------------------
          .

!!! Main body of the new scheme
          .
       IF (P_QI .gt. 0) THEN
          DO k=kts,kte
             DO i=kts,kte
                DQIDT(I,K)=....
             ENDDO
          ENDDO
       ENDIF

       IF (P_QS .gt. 0) THEN
          DO k=kts,kte
             DO i=kts,kte
                DQSDT(I,K)=....
             ENDDO
          ENDDO
       ENDIF
          .
```

```
  END SUBROUTINE XXX2D

!=================================================================
  SUBROUTINE xxxinit(...                                          &
                  ids, ide, jds, jde, kds, kde,                   &
                  ims, ime, jms, jme, kms, kme,                   &
                  its, ite, jts, jte, kts, kte                     )
!-----------------------------------------------------------------------------------------------
  IMPLICIT NONE
!-----------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                          its, ite, jts, jte, kts, kte
        .
!!!  add initialization here if it is necessary
        .
  END SUBROUTINE xxxinit

!-----------------------------------------------------------------------------------------------

END MODULE module_cu_xxx
```

## module_mp_xxx.F (microphysics)

```
MODULE module_mp_xxx

!!! put scheme related parameters or constants here

CONTAINS
!-----------------------------------------------------------------------------------------------
  SUBROUTINE XXX(qv,th,.....,RAINNC,.....,                        &
                          ids,ide, jds,jde, kds,kde,              &
                          ims,ime, jms,jme, kms,kme,              &
                          its,ite, jts,jte, kts,kte                )
!-----------------------------------------------------------------------------------------------
 IMPLICIT NONE
!-----------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                          its, ite, jts, jte, kts, kte
  REAL,      DIMENSION( ims:ime , kms:kme , jms:jme ), INTENT(INOUT) :: th,    &
             qv, ...
  REAL,      DIMENSION( ims:ime , jms:jme ), INTENT(INOUT) :: RAINNC

!  LOCAL VARIABLES (use tile size)
  REAL, DIMENSION( its:ite )             :: rain
  REAL, DIMENSION( its:ite, kts:kte )   :: ...
  INTEGER :: i,j,k
!-----------------------------------------------------------------------------------------------
  DO j = jts,jte

    ! either use ik slice or just use 3D array
    ! be careful about vertical order
!
    CALL XXX2D(qv(ims,kms,J),rain,....                            &
```

```
                      ids,ide, jds,jde, kds,kde,                    &
                      ims,ime, jms,jme, kms,kme,                    &
                      its,ite, jts,jte, kts,kte                      )

! rain -- mm in one time step
! RAINNC  -- mm (accumulated precipitation)

    DO i = its,ite
        RAINNC(i,j)=RAINNC(i,j) + rain(i)
    ENDDO

  ENDDO

  END SUBROUTINE XXX

!=================================================================
  SUBROUTINE XXX2D(qv2d,rain,....                                  &
                              ids,ide, jds,jde, kds,kde,           &
                              ims,ime, jms,jme, kms,kme,           &
                              its,ite, jts,jte, kts,kte,            )
!-----------------------------------------------------------------------------------------------------
    IMPLICIT NONE
!-----------------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                              its, ite, jts, jte, kts, kte
  REAL,     DIMENSION( ims:ime, kms:kme ), INTENT(IN ) ::  QV2D
  REAL,     DIMENSION( its,ite ), INTENT(INOUT)        ::  rain

!  LOCAL VARIABLES (use tile size)
  REAL,    DIMENSION( its:ite, kts:kte ) ::  .....
!-----------------------------------------------------------------------------------------------------
        .
!!! Main body of the new scheme
        .
  END SUBROUTINE XXX2D

!=================================================================
  SUBROUTINE xxxinit(...                                           &
                            ids, ide, jds, jde, kds, kde,          &
                            ims, ime, jms, jme, kms, kme,          &
                            its, ite, jts, jte, kts, kte            )
!-----------------------------------------------------------------------------------------------------
  IMPLICIT NONE
!-----------------------------------------------------------------------------------------------------
  INTEGER, INTENT(IN)  :: ids, ide,jds, jde,kds, kde,ims, ime, jms, jme, kms, kme, &
                              its, ite, jts, jte, kts, kte
        .
!!!  add initialization here if it is necessary
        .
  END SUBROUTINE xxxinit

!-----------------------------------------------------------------------------------------------------

END MODULE module_mp_xxx
```

Table 1: The schemes in the column “In” have already implemented into the WRF, and in the column “Working” are currently under working.

| Physics | In | Working |
|---|---|---|
| Microphysic | Kessler,<br>Lin,<br>NCEP simple ice | NCEP 5-species,<br>Goddard microphysics |
| Convective parameterization | Kain-Fritsch,<br>Bett-Miller-Janjic | Grell,<br>New Kain-Fritsch |
| Long-wave radiation | RRTM | Goddard long wave |
| Short-wave radiation | Simple short wave,<br>Goddard short wave | |
| Surface layer | Similarity theory | |
| Land-surface layer | Thermal diffusion | Oregon State-Eta |
| Boundary layer | MRF | |
| Subgrid eddy diffusion | Simple diffusion,<br>Stress/deformation form | |

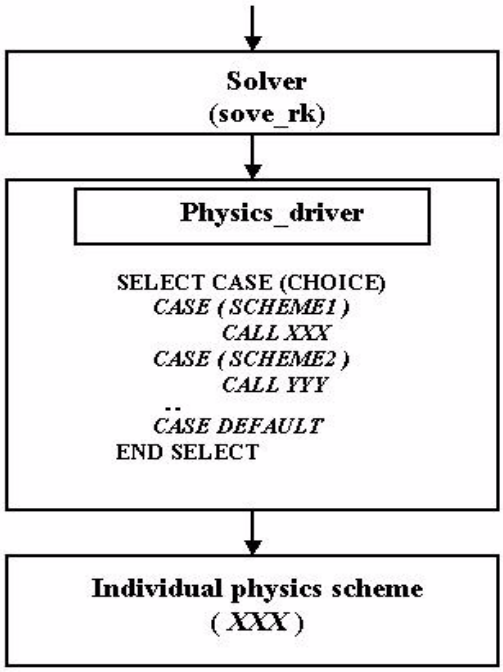


Figure 1: A three-level physics structure.

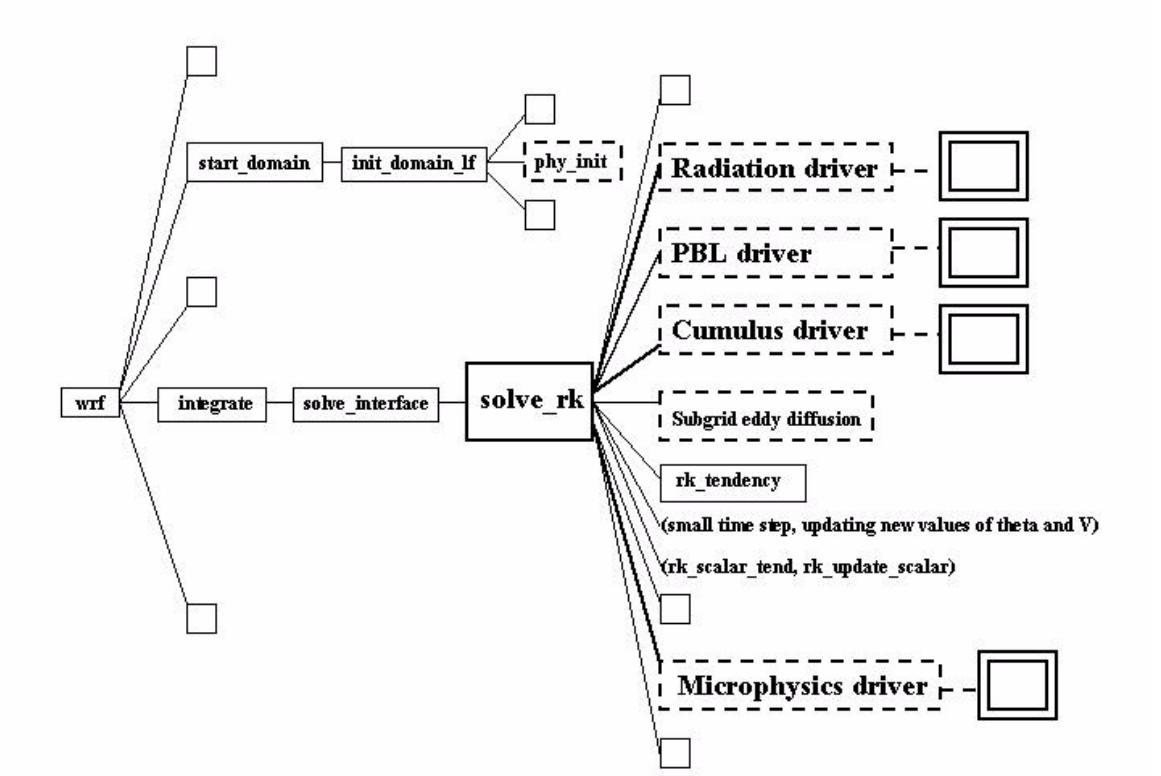


Figure 2: The schematic diagram of the WRF model. wrf is the main program and solve_rk is the major subrouine, which calls dynamics and physics. Dashed-line and double-line (individual scheme) boxes are codes involved with the model physics.

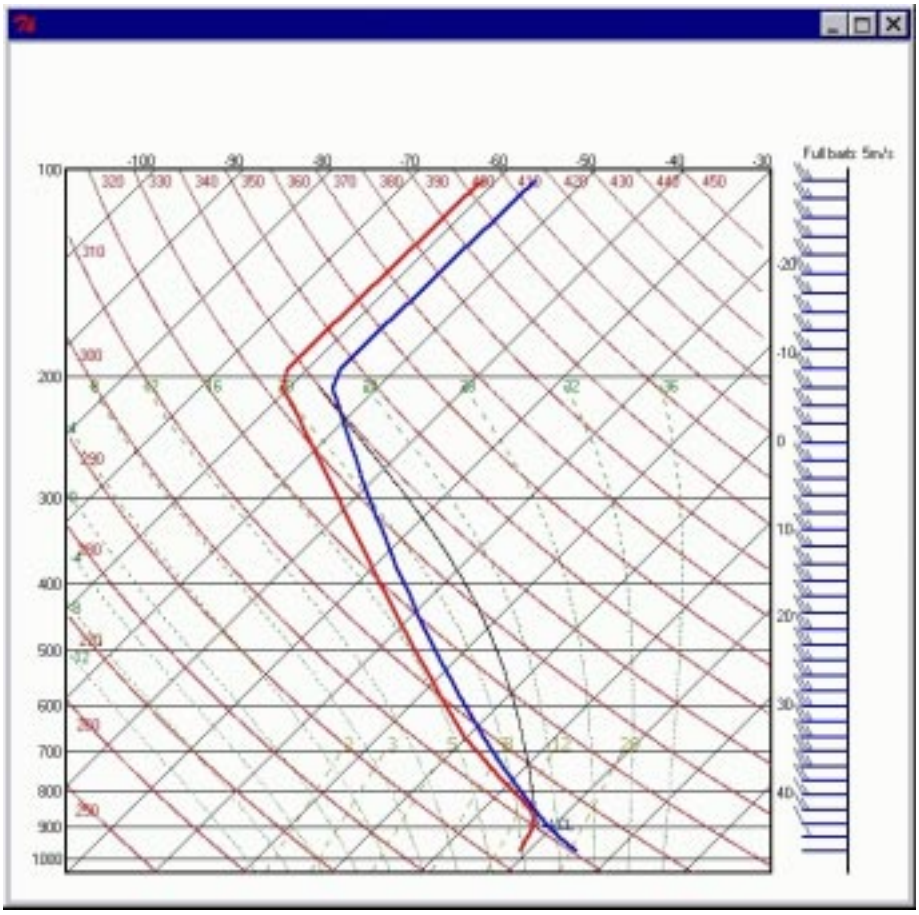


Figure 3: The sounding used in the squall line test.

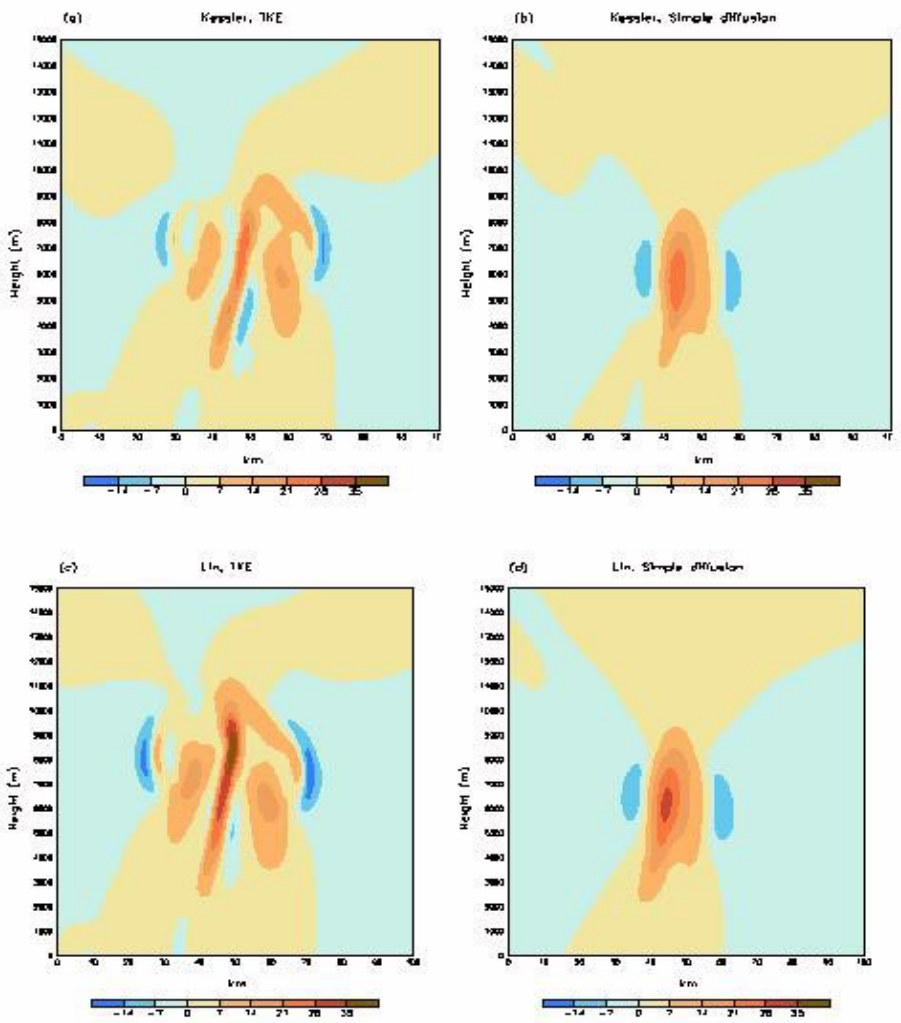

Figure 4: The vertical cross-sections of vertical velocities with (a) Kessler scheme and TKE turbulence, (b) Kessler scheme and simple diffusion, (c) Lin microphysics and TKE turbulence, and (d) Lin microphysics and simple diffusion after one-hour simulation.