MM5 Version 2 Release Notes

(Updated April 2 1999)

1. Introduction

MM5 V2 (referred to as V2 hereafter) has been under development since last winter (1995). In addition to the normal upgrade of the source code (e.g. incorporating accumulated changes since the V1 release), effort has been put in to increase portability of the code. As a result of several months' work, V2 is available for users to test. In the following pages, you will find information about what is new in V2, how to run V2, and how to make changes to source code.
With the release of V2, we now formally support MM5 model code on both Cray and workstations. However, because there are many machines and compiler versions available, it is impossible to cover every one of them. Our current policy is to support the compiler options we provide (see configure.user). Like in the previous policy, we will only be able to support STANDARD use of the model.

Starting from release-2-5 (June 24 1997), f90 compiler options for Crays are added. The Cray batch job deck is updated to enable users to either use cf77 options or f90 options. For more information on using Cray f90 compiler, please visit the Web page:

http://www.mmm.ucar.edu/mm5/mm5sysv2-f90.html

Starting from release-2-8 (March 22 1998), the same source code may be run on shared-memory machines (such as Cray J90's, SGI Origin 200/2000), as well as distributed-memory machines (such as IBM SP2, Cray T3E). For more information on how to run the model on a distributed-memory machine, please read the README.MPP file inside the MM5 tar file and visit our Web page:

http://www.mmm.ucar.edu/mm5/mm5v2/mm5v2.html

Please send comments and bug fixes to *mesouser@ucar.edu.*

(last revision: April 2 1999)

2. What is New in V2?

2.1 More Physics Options:

The following physics options are added to the model.

- CCM2 radiation (IFRAD=3).
- Level 2.5 turbulence closure PBL (Burk-Thompson, IBLTYP=3).
- MRF PBL (Hong and Pan 1996; IBLTYP=5; since release-2-2 in Dec 1996).
- Multi-layer soil model (ISOIL=1; and works only with IBLTYP = 2, 5 options).
- Kain-Fritsch cumulus scheme (ICUPA=6).
- Fritsch-Chappell cumulus scheme (ICUPA=5).
- Betts-Miller cumulus scheme (ICUPA=7).
- Explicit moisture scheme that includes graupel (Reisner2 and Goddard schemes: IMPHYS=6 and 7).
- T' diffusion in non-hydrostatic model (ITPDIF=1).
- Improving existing physics: vertical diffusion, solar scattering, and Grell scheme.
- Eta Mellor-Yamada scheme (Janjic 1990; IBLTYP=4; since release-2-11 in Dec 1998).
- Schultz explicit moisture scheme (including graupel) (Schultz 1995; IMPHYS=8; since release-2-12 in April 1999).

Three new namelist variables are added to represent (1) different explicit schemes (IMPHYS=1 through 7); (2) perturbation temperature diffusion in nonhydrostatic model (ITPDIF=0 or 1); and (3) multi-layer soil model (ISOIL=0 or 1). Please see mm5.deck in mm5v2.tar file for complete explanation of the namelist variables.

2.2 Other Improvements and Changes:

1) Look-up table options for explicit schemes (IMPHYS=3,4, and 5).

2) Efficiency modifications, particularly replacing /PSA with *RPSA and replacing statement functions with arrays.

3) FDDA memory is separated for obs and grid nudging to save memory when doing just obs nudging. Also memory is separated from other model memory in the restart file saving memory for a restart after FDDA is switched off (e.g. dynamic initialization).

4) Add new option to save only the last save file: SVLAST = .TRUE.. This option is useful if disk usage is of a concern (available since release-2-3).

5) Add new switches (IVTADV and IVQADV) to allow the use of linear vertical interpolation of temperature and moisture fields (including microphysical variables) when calculating vertical advection. This option could save 5-8 % of total CPU time when used. Small difference in results if more than 20 sigma levels are used (available since release-2-4).

6) Add new switches (IDYNIN and DTRAMP) to allow for using ramp-down function to gradually turn off nudging in dynamic initialization (available since release-2-5).

7) Add new switch ITHADV to allow for using potential temperature in temperature advection and adiabatic term in the temperature equation. Test results indicate that this formulation gives more accurate results in high-resolution simulation near complex topography. For more details, please see Skamarock et al. (1997, 7th MM5 Users' Workshop preprint, p80, and equations on p82. Available since release-2-6).

8) The solar constant is no longer a constant since release-2-7 (December 1997). The solar constant is calculated using eccentricity factor (Paltridge and Platt 1976) and varies with the time of year. The value ranges from 1324 - 1418 Wm^{-2}, and it used to be 1395 Wm^{-2}.

9) Subroutine *sound.F* is reconstructed in release-2-7 (Dec. 1997) for efficiency purpose. Minor difference in the output fields is expected. This change allows a speed-up of 8 - 10 % of total CPU time in a typical run. Memory increase is minimal.

10) New output fields are added since Release-2-7. These are: 3-D atmospheric radiative tendencies (if IFRAD=2,3), PBL height, PBL regime, surface sensible and latent heat fluxes, frictional velocity, downward short- and long-wave radiation, soil temperatures (currently in 5 layers plus a substrate). These fields will be in your output depending on physics options chosen.

11) The standard MM5 V2 code can be run on a few multiple processor machines: Crays and SGI since release-2-0, and HP SPP-UX since release-2-7 (Dec. 1997).

12) Starting from release-2-8 (March 22, 1998), the same source code may be run on a distributed-memory machine (commonly referred to as MPP machines), such as IBM SP2 and Cray T3E.

13) OpenMP parallel directives for shared-memory parallel machines are added in release-2-11 (Dec 23, 1998). A number of vendors support it already: Cray, SGI, DEC, pgf77 for Linux, for example.

14) Eta model version of Mellor-Yamada PBL scheme (Janjic 1990) is released in 2-11.

15) An option for inputting nest terrain during model integration is added in release-2-11 (IOVERW=2). This option allows users to initialize new nest during model integration with higher resolution nest terrain files.

16) An option for outputting history files at different time frequencies for different domains is added in release-2-11 (INCTAP).

17) A new explicit moisture scheme developed by Schultz (Schultz 1995), and implemented by Greg Thompson of RAP/NCAR is added in release-2-12 (April 1999). This is another graupel scheme. It is very efficient on scalar machines.

18) A way of allowing user to input different landuse categories is implemented in release-2-12. The corresponding land surface properties (such as albedo, roughness length) can be changed in a new table file located in the Run directory. Currently the model supports the old 13-category, the SiB 16-category, and USGS 24-category landuse.

2.3 Expected Differences from V1

Differences are expected in v2 results compared to v1 for the following reasons:

1) solar scattering reduced (SWRAD)

2) background vertical diffusion reduced (KZO) (HIRPBL, BLKPBL)

3) ice sedimentation included (EXMOISS, EXMOISR)

4) v2 Grell scheme - changed limits to allow more clouds (cup.F)

5) curvature term in w eqn (very small effect) (SOLVE3)

6) solar angle calculation centered in RADFRQ +-window (very small effect) (SWRAD, SFCRAD)

7) sfcdownlw.mods (from PSU) included for longwave (SFCRAD)

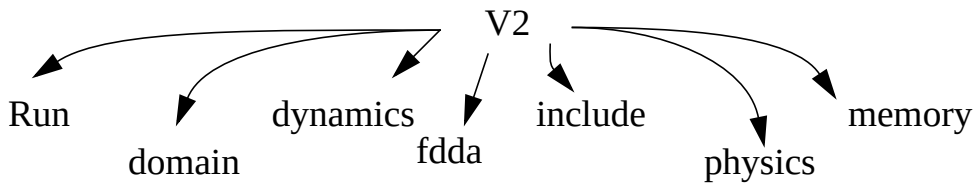
8) replacement of statement functions with arrays (very small effect) (SOLVE3, SOUND)

9) replaced many divides by multiplies (rounding differences only)

2.4 Logical Subdivision of Code

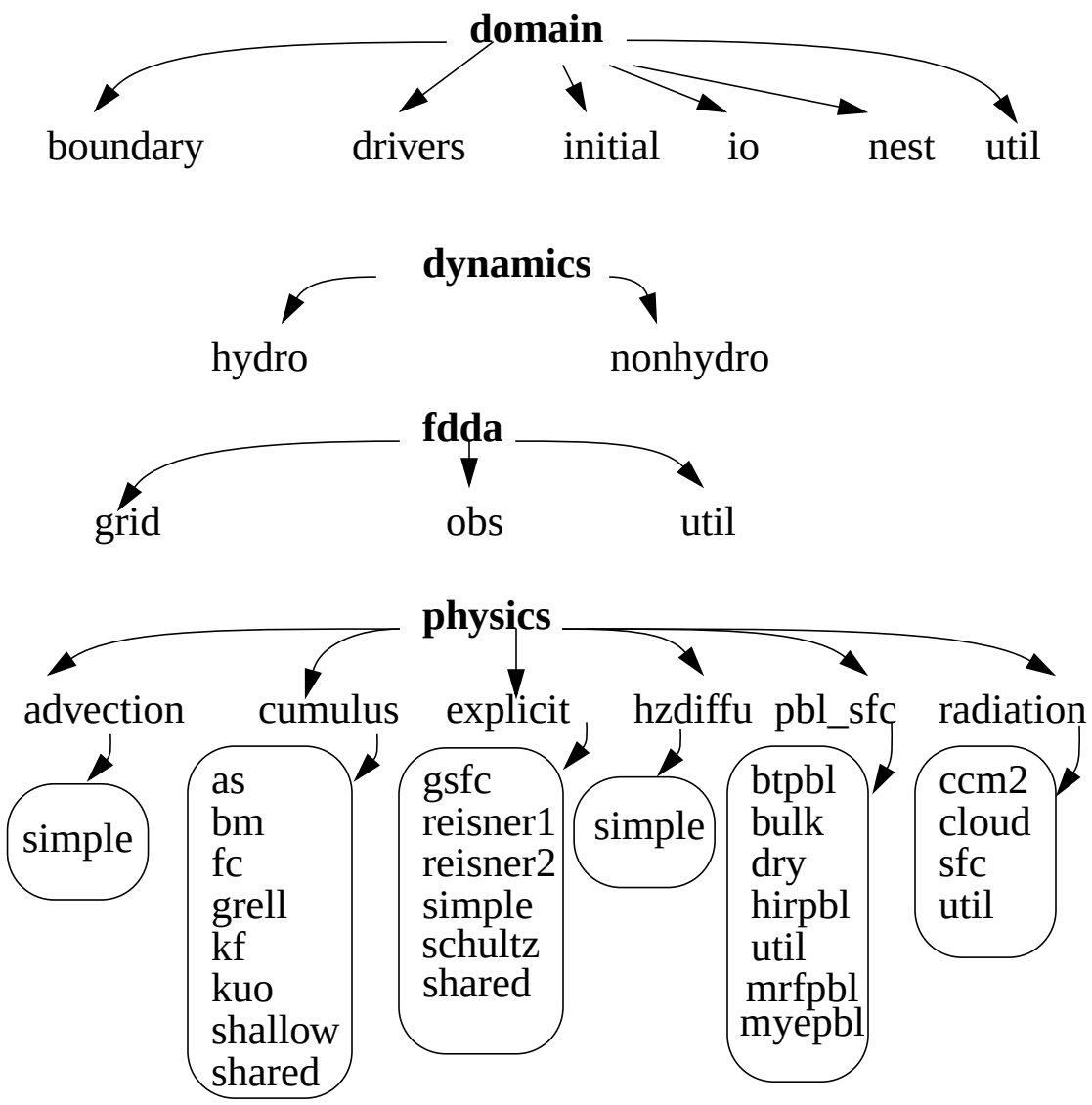
The goal was to provide a logical structure to code development and encourage modular development of new options and subroutines. In addition, it was desired to supply the user/ developer with some “pointers” as to the location of routines of particular interest. So the hope was to create something that appeared simple to the casual user but allowed more convenient access for the power user.

This structure was implemented implicitly by taking advantage of the Unix file system structure. Since directories are arranged as trees, the subroutines were subdivided into conceptual groups. These groupings were used to create the following structure.



The include directory contains the include files for the various subroutines. The domain, dynamics, fdda, physics, and util directories contain the subroutines divided by function. The Run directory hold the main program source code.

It is this view that the casual user sees. All the files are hidden from view. However, the power user will find the underlying structure arranged as shown in the next figure.



The casual user doesn't need to see all these directories because we take advantage of the recursive behavior of make to keep the details of the compilation hidden at each level. A complete listing of the file structure is in Appendix A.

2.5 Minimize Portability Issues

Writing portable code involves not only following language standards, but creating a development structure that is equally standard. Every time code moves to a new machine you not only need to

worry about your code but compilers, the operating system and the system environment, available libraries, and options for all of the above.

The answer to this problem is two-fold, namely, use only standard tools and minimize use of esoteric options. The following Unix development tools were chosen.

- make
- cpp
- cvs

Make and cpp are used mainly for conditional compilation.

2.6 Conditional Compilation (make and cpp)

One of the stated goals was conditional compilation. This is done in two different ways. **Make** keys off the user's options to skip compilation of those directories not required. When a source file is compiled, **cpp** is used to avoid including code that is not required. So Make skips unnecessary compilation while cpp is used to modify compilation.

Make

"All makes are equal, but some makes are more equal than others."
Every decent Unix box will have make and cpp. However, they may throw in non-standard options. The quotation above reminds us to keep to standard constructions whenever possible. Make is a tool that executes "makefiles". Makefiles contain "targets" and "dependencies". A target is what you want to compile. A dependency is what needs to be done to compile the target.

We use a 3-tiered makefile structure following the directory structure.

- Top-Level
- Middle (branching) Level
- Lowest (compilation) Level

Examples of each makefile follow.

- Top Level hides everything. The casual user edits the parameters in configure.user and then just types "make". We take care of the rest.
- Middle Level is where branching occurs. These would be modified for something like the addition of a new moist physics scheme.
- Lowest Level is where object files are made. Change this when adding files. In addition, the power user will make in these lower directories to avoid remaking the whole structure.

CPP

The cpp pre-processor is about as old as Unix itself. A pre-processor scans a file and make modifications according to user-supplied definitions. Typically this facility is used for global substitutions, conditional code inclusion, including files, and function templating. We only use the cpp "conditional code inclusion", "including files", and "function templating" features.

The ".F" suffix for all the fortran files is used to indicate that cpp is being used as part of the compilation process.

CPP "inclusion"

One cpp directive is "#include <filename>". This directive indicates that filename should be included in the source prior to compilation.
Example:

```
      SUBROUTINE solve1(IEXEC,INEST,NN)
# include <parame.incl>
```

turns into

```
      SUBROUTINE solve1(IEXEC,INEST,NN)
C PARAME
C
C--- ADDITIONAL MEMORY REQUIREMENTS FOR FDDA RUNS (IFDDA=1),
C--- NONHYDROSTATIC RUNS (INHYD=1), HIGHER ORDER PBL RUNS (INAV=1),
C--- EXPLICIT MOISTURE SCHEME (IEXMS=1), ARAKAWA-SCHUBERT
C--- CONVECTIVE PARAMETERIZATION (IARASC=1), ATMOSPHERIC
C--- RADIATION (IRDDIM=1), MIXED-PHASE ICE SCHEME (IICE=1).
C--- GRAUPEL SCHEME (IICEG=1).
C
C
     PARAMETER (IFDDA=0,INHYD=1,INAV=0,IICE=0,IICEG=0,
    1          IEXMS=1,IARASC=0,IRDDIM=1)
```

CPP "conditionals"

cpp also recognizes conditional directives. You define a macro in your source code using the "define" directive - you can then use the "ifdef" test on this macro to selectively include code.
Example:

```
#define IMPHYS4 1
#define IMPHYS1 1
#define ICUPA3 1
#define IBLT2 1
```

```
C-  WHAT CUMULUS PARAMETERIZATION? 1. NONE
      IF(ICUPA(INEST).EQ.1)THEN
      CALL VADV(KZZ,QVTEN,QVA,QDOT,MSFX,PSA,SCR3,J,2,INEST)
C
C--- 2. ANTHES KUO
#ifdef ICUPA2
      ELSE IF(ICUPA(INEST).EQ.2)THEN
      CALL CUPARA2(J,INEST,QVTEN,TTEN)
#endif
```

..... *and so on.*

In this example ICUPA2 is not defined, so the CUPARA2 call is not included in the final source. Another example:

```
#define FDDAGD1 1
#define FDDAOBS0 1
```

In solve3.F, one can use

```
#if defined(FDDAGD1) || defined(FDDAOBS1)
       IF(IFDDA.EQ.1)CALL SETFD(INEST,FDTIM,INFR,IDARST)
#endif
```

to check if *FDDAGD* or *FDDAOBS* is defined. In this case, *FDDAGD* is defined, so subroutine *SETFD* will be called in the source code.

2.7 Version Control (CVS)

In order to move away from Cray's 'nupdate' function, and make code maintenance the same for both Cray and workstations, we have chosen to use Unix tool, *CVS*, for source code maintenance. This should be transparent to users. Users will not need to know CVS to use the model.

CVS

In the non-propriety portion of the Unix world, CVS is the standard for version control (CVS stands for Concurrent Version System). In fact, many vendors offer proprietary versions of CVS that simply add a GUI or a few extra features.
The main features of CVS are described in the following.

CVS features

- Highly portable.

- Allows multiple developers to work on the same code without overwriting each others work. Will merge code together and complain if merge presents contradictions. In particular this means that if a user is also using CVS they can merge in changes using **one** command.
- Allows recursive operations on your directory structure.
- Can checkout an earlier version.
- Independent from the other aspects of the development system.
- Has a track record - first developed in the 80's - so well tested.
- More advanced features allow you to checkout and update code over networks, tag particular versions and create subbranches.

Information on where to obtain CVS on the net can be found from our Web page:
http://www.mmm.ucar.edu/mm5/cvs.html

2.8 Passing Changes to Users

We will pass changes to the source code to users in the following way:

- Send email to users as before, announcing the changes (if you are not on our mailing list, please see http://www.mmm.ucar.ncar/mm5/news.html on the Web for on-line subscription);
- Will have a new source code tar file available on both NCAR's MSS and anonymous ftp;
- Users should find a description of the changes in the file CHANGES in the top directory, and a difference file between the current and previous releases in files named diff.ddmmmyy located in ReleaseNotes directory, where dd, mmm, and yy denote day, month, and year respectively. The line identifier in the original code as well as in the modified code will help users to find the places where changes occur.
- For users who makes changes to V2 source code, it is most convenient if CVS is utilized in maintaining the code and keeping up-to-date with the mesouser releases.

3. How to Run MM5 V2

3.1 General Remarks

The following sections are designed to provide users with information on how to **compile** ("making") and **run** the V2 code on a UNIX workstation and Cray. Every effort has been made to make the system portable and easy to use. Nonetheless, the user is presumed to have a basic familiarity with the UNIX compilation environment - in particular, it is presumed that the user knows what basic utilities and compilers are or are not available on his/her specific architecture.

Where to obtain source code:

On NCAR's MSS, the file name for the most current release is

/MESOUSER/MM5V2/MM5/MM5V2.TAR

Previous releases are named similarly with the release number attached, e.g.

/MESOUSER/MM5V2/MM5/MM5V2.TAR.release-2-1

On anonymous ftp, ftp to ftp.ucar.edu, login as anonymous, and use your full email address as password, then cd to /mesouser/MM5V2/MM5 directory, and type

binary
get mm5v2.tar.Z

Or if you are using CVS, and have access to NCAR's computer *meeker.ncar.ucar.edu* (Sun machine), you can obtain the V2 MM5 code by checkout or export directly from our repository by typing

cvs checkout (or export) -r release-2-x MM5

You need to set the following in the .cshrc file:
setenv CVSROOT /fs/othrorgs/home0/mesouser/repository

Overview:

There are 2 steps to compiling and running the MM5 (V2) system.

- Choosing compilation options and compiling the code.
- Modifying the run-time options and executing the program.

3.2 Compiling MM5

- Edit the file “configure.user”
- make

The user chooses those compilation options appropriate to his/her system by editing the “configure.user” file. This file is included in every Makefile used in compiling the model so it contains many rules, but the user need only concern him/herself with 4 things.

1. set the RUNTIME_SYSTEM variable. This should indicate what kind of machine you are running on (e.g., CRAY)
2. Pick the Fortran compiler options appropriate for your system. We have provided default options for every major vendor to which we have access - if these are sufficient, simply uncomment the ones for your system. If your system has its own compilers you may have to modify these parameters.
3. Make sure that the general utilities required in a UNIX environment for compilation are available and appropriate. For example, there are many versions of the program “make” - if yours has special quirks and/or options, this would be the place to indicate them.
4. Set model options in sections 5 and 6 of configure.user. These are used to set up domain sizes, dynamics and physics option for (selective) compiling purposes.

Once you have chosen the parameters you need, save the changes in the “configure.user” file. Type “make” at the shell prompt and hit return. If the parameters you have set are correct for your system and the utilities you have specified exist, the system will compile. That’s it!

> *Trouble Shooting*: If your make fails, there are probably clues in the diagnostics that will help locate the problem. Check your options for correctness. Some versions of make will fail if a system call returns any non-zero value - even the top level make! If this is the case with your system, use “make -i” rather than “make” and the problem will be solved. Some compilers do not support the usage of the cpp preprocessor (e.g., the xlf FORTRAN compiler on the IBM). If this is true of your system, we have supplied an alternate Makefile target called “little_f”. This target will first use cpp to explicitly create .f files from .F files and use the .f files to compile. If this is true of your system use “make little_f” instead of “make”.
> It is also a good rule to do a ‘make clean’ and then ‘make’ to recompile.

3.3 Running MM5

- make the “mm5.deck” script.
- edit the mm5.deck script
- run the “mm5.deck” script.

Make the “mm5.deck” script by entering “make mm5.deck”. The system will use the RUNTIME_SYSTEM variables to choose a template appropriate for running the MM5 (V2.0) system on your machine. Within this script there are additional physics options as well as I/O file names required to run MM5. Make such modifications to the options as required.
Once you have chosen your physics options, run the script by entering “mm5.deck” as you would with any shell script.

Trouble Shooting: If you receive a message indicating that you do not have execute permission to run mm5.deck, remedy the problem by “chmod 777 mm5.deck”. This command will add execute permissions to the script.

Note:

1) All the input files, mm5.exe, etc. are in the directory Run;
2) If you have changed anything in configure.user, just type ‘make’ again.
3) If errors occur during compilation, the first thing to do is to type ‘make clean’ to remove all .o files and archive, and redo ‘make’.
4) Even though the V2 system is designed for selective compiling, it is possible to *compile* the code for more **physics** options than you would use in a run. To do so, fill the arrays of physics parameters in configure.user file (e.g. IMPHYS, ICUPA) with the options you want, and compile the code. Remember the order of the parameter does not matter during the compilation, but it is important when executing: the first colume always corresponds to the options used the the coarse domain. An exception is the FRAD array, only the number in the first colume is used during execution and it is for all domains. The rest of columes are only used for compilation purpose.
5) If you want to compile and run the model in background mode, say on a workstation, use the following command:

   nohup make >&! make.out && mm5.deck >&! mm5.out &

   where

nohup	ignore hangups
make >&! make.out	compile the code and redirect stdout and stderr to file make.out
&&	execute the next command if the previous command is successful (return 0 status)
mm5.deck >&!	run the model and redirect stdout and stderr to file mm5.out
&	put the job in background

A special note for Cray users:

1. If you want to work in batch mode, whether to compile and/or execute, get a copy of mm5.deck from mesouser directory: ~mesouser/MM5V2/MM5 on ouray/paiute, or anonymous ftp under /mesouser/MM5V2/MM5. This deck has an appearance of V1 deck, and has the *configure.user* file inside the deck. This deck is designed to be used for both interactive and batch mode.

   Or, you may get the deck once you obtain the mm5v2.tar file on your local machine. To do so, first un-tar the tar file, edit the *configure.user* file to define RUNTIME_SYSTEM=‘‘CRAY”; then type ‘make mm5.deck’.

2. If you would like to compile interactively on a Cray, you can either use the above deck, or use the Cray interactive deck, by setting the RUNTIME_SYSTEM=‘‘CRAY_IA’’, and followed by typing ‘make mm5.deck’. The mm5.deck generated this way has an appearance of other workstations decks.

   When you use this deck to compile, you will still need to use the deck described in (1) to submit a batch job for executing. Before you submit the batch job, remember to tar up your entire directory structure, and save it to some place (whether it is NCAR’s MSS, or your local archive). Your batch job needs to access this tar file (default name mm5exe.tar) for executing.

Note: The *mmlif* (namelist file) for running MM5 is now generated from both your *configure.user* file (pre-compilation options) and mm5.deck.

3.4 Make Source Listing

To make a source listing of the fortran code, type ‘make code’ at the top directory, and cd to ./pick directory. Read the README file in the ./pick directory for instructions on making a single listing of all *.f files.

4.Miscellaneous Information

4.1 Debugging

If you modify the code, you may introduce “bugs”. Some “bugs” the compiler will be able to catch, particularly problems with syntax. But logical defects may be more difficult. If you intend to debug, we recommend that you reset the compilation options (in configure.user) for debugging and then recompile all the code to make sure that the new compilation options are in effect for every subroutine (make sure you do ‘make clean’ first to remove existing .o files). Typically, you will need to remove all the optimization options and replace them with the debugging option “-g”. Some compilers allow you to mix and match optimization options and debugging options and if your compiler supports this then feel free to do so. However, we do not recommend this as a general practice. In any event, make sure you read the man pages for your particular compiler before doing anything to make sure that you have set the options correctly.

Recompiling the entire system:

- make clean
- make

4.2 Changing subroutines

Changing several subroutines:

Once you’ve compiled all the subroutines, there is rarely a need to recompile them all again. If you’ve changed several routines then

- cd (change directory) to the top of the directory structure.
- make

The “make” program is smart enough to only compile those subroutines you have changed.

Changing a single subroutine:

If waiting for the library to be re-archived becomes tedious to you, you can recompile subroutines as you change them. Let’s use an example. In the file *“domain/initial/param.F*”, say you want to add a line after line PARAM.440:

*PRINT *, ‘I HAVE MODIFIED THIS VERSION OF PARAM’*

How do you do this?

- cd to the subdirectory domain/initial

- edit the file param.F and make the change;
- type ‘make -i -r’ to compile the subroutine and archive it to the library;
- cd to ../../Run
- type ‘make -i -r’ to regenerate mm5.exe;

Either,

- cd .. to top directory, and edit mm5.deck, if needed
- type ‘mm5.deck’ to run the model.

Or,

- stay in the Run directory, type mm5.exe >& mm5.print.out & to execute.

4.3 Test datasets

Test dataset attributes:

Non-hydrostatic input data:
Coarse domain - 90 km - 25x28x23
Fine domain - 30 km - 34x37x23

For Cray runs: (on NCAR’s MSS only)

/MESOUSER/DATASETS/SESAME/NH/BDYOUT_DOMAIN1
/MESOUSER/DATASETS/SESAME/NH/MMINPUT_DOMAIN1
/MESOUSER/DATASETS/SESAME/NH/MMINPUT_DOMAIN2

For workstation runs:

/MESOUSER/DATASETS/SESAME/NH/IEEE32/BDYOUT_DOMAIN1
/MESOUSER/DATASETS/SESAME/NH/IEEE32/MMINPUT_DOMAIN1
/MESOUSER/DATASETS/SESAME/NH/IEEE32/MMINPUT_DOMAIN2

Or from anonymous ftp:
cd /mesouser/Data/SESAME
get mminput_nh_data.tar.Z (use binary mode)

4.4 Running graph job

Grab version 2 graph tar file, graph.tar.Z, and/or graph.exe (graph.exe.Z) from ouray/paiute/chipeta *~mesouser/MM5V2/Graph* directory if you are running on NCAR's Crays. This version of Graph code may be used on a workstation. The tar file is also available from anonymous ftp:

ftp://ftp.ucar.edu/mesouser/MM5V2/Graph

4.5 How to maintain your copy of the source code

It is important to maintain a copy of your source code. When there are changes to the source code, we will broadcast the changes. You will then find a new copy of the mm5v2.tar file on both NCAR's MSS and anonymous ftp. You will find a description of the changes in the current release in the file CHANGES in the top directory of the tar file, and the actual difference in ReleaseNotes/diff.ddmmmyy, where dd, mmm, and yy denote day, month, and year of the release. You should be able to find where the code has been changed using the information provided. Then you can make a decision whether you want to include the changes.

Once again, if you are going to make changes to the MM5 V2 code, it is most convenient if you use CVS to maintain your code. This way, you can merge the changes we make into your code automatically.

5. On-going Plan

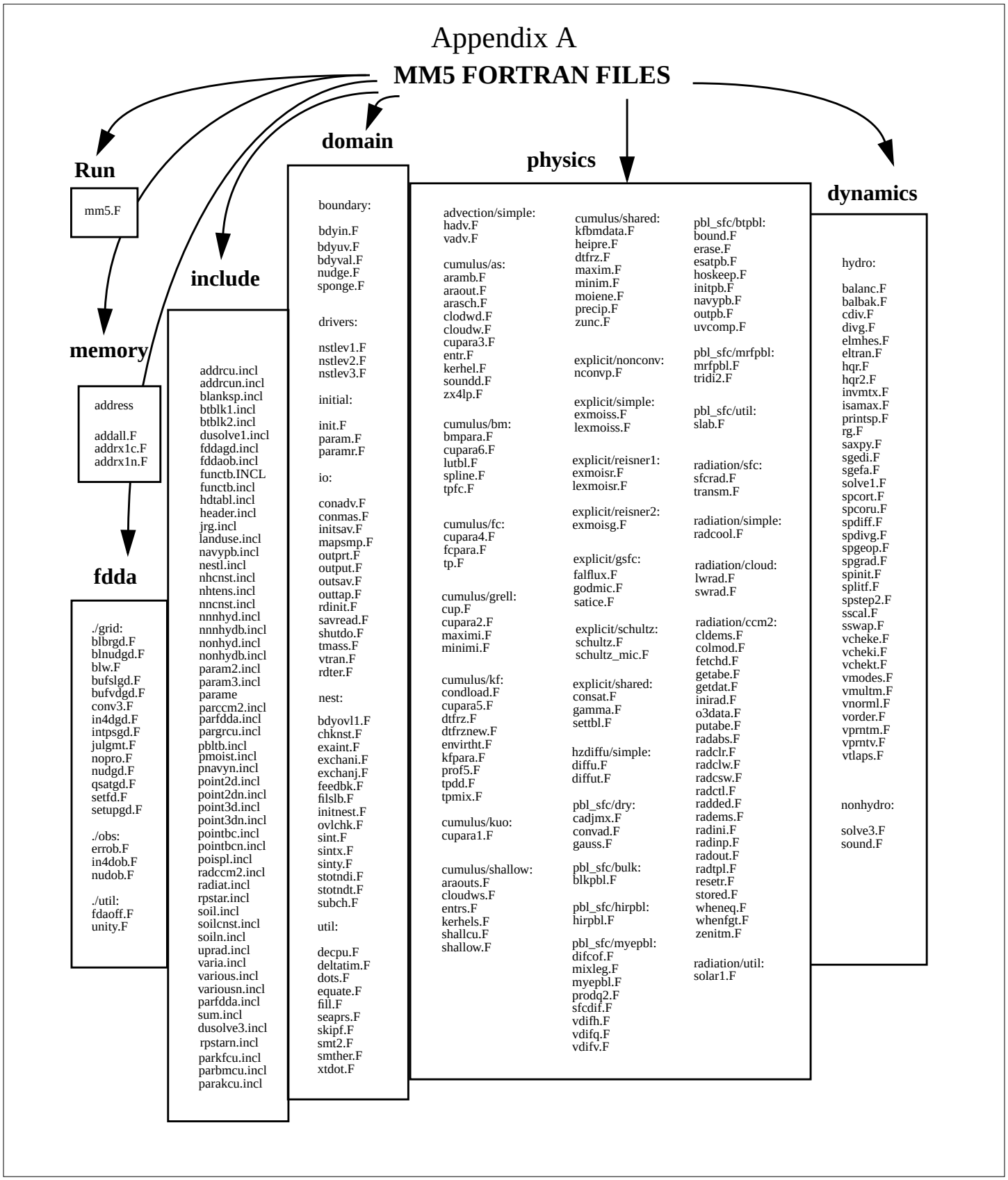
Since most of the MM5 code was developed on a Cray, when it is ported to a workstation, some portion of the code does not work well. The problem arises partially from errors in compiler, especially when any level of optimization is used, and partially from numerical accuracy that is available on Cray and workstations (e.g. 64 bit vs 32 bit machines). Therefore not all the dynamic and physical options that are available on a Cray are working on all workstations. The following is a list of options that we have not tested on certain workstations.

- Hydrostatic option has not been tested on IBM.

Also, the Arakawa-Schubert cumulus parameterization scheme does not run without the imsl library.

Appendix A

MM5 FORTRAN FILES



Appendix B

Timing Comparison for MM5 V2

The following results are based on the 12 hours forecast using the MM5 tutorial case.
All measures in cpu time (hours:minutes:second). To find more complete and up-to-date timing results, please visit our Web page: http://www.mmm.ucar.edu/mm5/mm5v2-timing.html.

1. General Information

Benchmark case:	1979 SESAME squall line simulation.
Domain size:	coarse (90 km) 25x28x23, 270 second time step
	fine (30 km) 34x37x23, 90 second time step
Forecast time:	720 minutes (160 large domain time steps, 480 nest domain time steps)
Physics:	
	Grell cumulus (ICUPA=3)
	Simple ice with warm rain microphysics (IMPHYS=4)
	Cloud radiation (FRAD=2)
	Blackadar PBL (IBLTYP=2)
	Nonhydrostatic (NHYDRO=1)
Machines:	
	Cray YMP (NCAR shavano)
	Cray J90 (NCAR paiute)
	SGI R8000
	SGI R5000
	SGI R4400
	SGI R4000
	DEC ALPHA 2100
	DEC ALPHA 500/333
	SUN SOLARIS SPARC 64
	SUN SOLARIS SPARC 20
	IBM AIX (NCAR chief)
	HP 735

2. Compilation timings

2.1 Compiler options:

Cray YMP and J90:

```
cf77 -Zu
```

SGI R8000:

```
f77 -O3 -SWP:=ON -n32 -mips4 -mp \
-OPT:roundoff=3:IEEE_arithmetic=3 \
OPT:fold_arith_limit=2001:fprop_limit=1000 \
-SWP:body_ins_count_max=0
```

SGI R5000, R4400 and R4000:

```
f77 -mips2 -32 -O0 -Nn30000 -Olimit 1500
```

DEC ALPHA:

```
f77 -cpp -c -O4 -Olimit 2000 -i8 \
-fpe0 -align dcommons -align records \
-warn nounreachable -convert big_endian
```

SUN SPARC 64 and SPARC 20:

```
f77 -fast -O2
```

IBM AIX:

```
xlf -O -qmaxmem=-1
```

HP735 (UX):

```
f77 -o
```

2.2 Compilation timing comparison:

The optimization levels are all different on the benchmark machines. Higher level optimizations tend to require longer compilation time. The numbers shown in the following table are for reference purpose. It doesn't mean any particular machine is faster than the other.

Table 1: CPU Time Required to Compile MM5 V2 (single processor)

Machine	CPU Time (h:m:s)
Cray YMP	0:06:41.56

Table 1: CPU Time Required to Compile MM5 V2 (single processor)

Machine	CPU Time (h:m:s)
Cray J90	0:19:19.10
SGI R8000	0:49:42.00
SGI R5000	0:01:02:00
SGI R4400	0:01:18.74
SGI R4000	0:02:06.00
DEC ALPHA 2100	0:02:50.65
DEC ALPHA 500	0:01:32.82
SUN SPARC 64	0:02:07.00
SUN SPARC 20	0:06:08.00
IBM AIX	0:16:08.93
HP 735	0:07:00.42

2.3 Comparison with V1 MM5

The compilation timing comparison are performed on Cray YMP, DEC ALPHA, and IBM AIX.

Table 2: CPU Time Required to Compile MM5 V1 (single processor)

Machine	CPU Time (V1)
Cray YMP	0:09:11.00
DEC ALPHA 2100	0:04:28.70
IBM AIX	0:02:09.15

3. Run time timings

3.1 Timing comparison

Table 3: CPU Time Required to Run 720 Minutes MM5 V2 (single processor)

Machines	CPU Time (h:m:s)	Megaflops (floating ops/CPU second)
Cray YMP	0:17:56.07	86.25
Cray J90	0:34:17.50	45.11
SGI R8000	0:29:04.26	51.44
SGI R5000	2:27:36.53	10.47
SGI R4400	3:05:40.76	08.33
SGI R4000	4:14:31.96	07.95
DEC ALPHA 2100	0:54:33.40	30.32
DEC ALPHA 500	0:33:16.5	48.01
SUN SPARC 64	1:30:07.09	17.17
SUN SPARC 20	2:29:37.30	10.34
IBM AIX	2:58:59.87	08.64
HP 735	1:13:41.95	20.99

3.2 Comparison with V1 MM5

The run time timing comparison are performed on Cray YMP, DEC ALPHA, and IBM AIX.

Table 4: CPU Time Required to Run 720 Minutes MM5 V1 (single processor)

Machine	CPU Time (V1)
Cray YMP	0:18:56.45
DEC ALPHA 2100	0:56:53.47
IBM AIX	5:41:0.74