

Global Modeling Initiative: Tutorial and User's Guide

NASA Center for Computational Science (NCCS)
NASA Goddard Space Center
Greenbelt, MD 20771

This document was written by:

Jules Kouatchou
Bigyani Das
Hamid Oloso

with the help of

Dan Bergmann
John Tannahill
Ghalib Bello

OCTOBER 15, 2004

Abstract

In this report, we provide a description of the GMI code. It is intended to help users in the GMI community who wish to obtain, install, compile, run, and modify the code. We present the code organization and data structures, procedures on how to manipulate the code, and its parallel performance.

Examples introduced here were carried out on the Compaq SC45, the SGI O3600 and a linux cluster.

Contents

1	Introduction	1
2	Structure of the Code	3
2.1	Components of GMI	3
2.2	Directory Structure of The Code	4
2.3	Coding Principles	6
2.4	General Flowchart of the Code	7
3	Installation and Testing	11
3.1	Getting the Code	11
3.2	Model Files and Directory Structure	12
3.3	Setting Environment Variables	12
3.4	Code Installation and Basic Test Run	14
3.4.1	Compiling the model	14
3.4.2	Testing the Executable	14
3.5	Summary of the Necessary Steps	16
4	GMI Files	18
4.1	Input Namelist Files	18
4.2	Input Datasets	20
4.2.1	netCDF Input Files	20
4.2.2	ASCII Input Files	21
4.2.3	Input Files Needed for "troposphere" Chemical Mechanism	21
4.3	Output Files	22
4.3.1	ASCII Diagnostic Output File	22
4.3.2	netCDF Output Files	23
5	Performing Specific Runs	24
5.1	Specific Runs	24
5.2	Restart Runs	26
6	Making Changes	29
6.1	Coding Consideration	29
6.2	Adding Chemical Mechanisms	29
6.3	The Make System	30

CONTENTS

6.4	Making Changes	30
6.5	Debugging the Code	32
7	Script Tools	33
8	How to Use CVS?	36
8.1	What is CVS?	36
8.2	How to Use CVS?	37
8.3	Use CVS to Keep Up to Date with GMI Source Code Changes	37
8.4	Use CVS to Track Both New Releases and Your Changes	41
8.5	Where To Obtain CVS?	42
9	Parallel Performance	43
9.1	Description of the Platforms	43
9.2	Parallel Performance	43
9.2.1	Description of the Test Cases	43
9.2.2	Model Performance	44
9.3	Profiling the Code	44
A	Include Files	50
B	Single/Multiple Processor Runs	52
C	NetCDF Files	53
C.1	Contents of netCDF Files	53
C.2	Output Frequency of NetCDF Files	53
D	New Features	56
D.1	Diagnostics	56
D.1.1	Choice of Species for Surface Emission Diagnostics	56
D.1.2	Choice of Species for Dry Deposition Diagnostics	57
D.1.3	Choice of Species for Wet Deposition Diagnostics	58
D.1.4	Choice of Species for Tendency Diagnostics	58
D.1.5	Choice of Vertical Levels	59
D.1.6	Noon Variables	59
D.2	Tracer Runs	60
D.3	Addition of FastJX	61
E	Input Namelist Variables	62

List of Tables

2.1	Chemical mechanisms	4
2.2	GMI code directories	6
9.1	Features of the different platforms.	44
9.2	Information on the test cases.	44
9.3	10-day simulation ($46 \times 72 \times 28$): wall clock time (in seconds) as function of the number of processors.	45
9.4	31-day simulation ($46 \times 72 \times 46$): wall clock time (in seconds) as function of the number of processors.	45
9.5	7-day simulation ($46 \times 72 \times 46$): wall clock time (in seconds) as function of the number of processors.	46
9.6	Number of cycles and computer operations.	46
9.7	One-day simulation of the combined strat/trop chemical mechanism: breakup of the resources consumed by the major routines.	47
9.8	31-day simulation of tropospheric chemical mechanism: timing breakup of the different operators.	48
C.1	General netCDF files.	54
C.2	Frequency of netCDF files.	55
C.3	Namelist variables to produce netCDF files.	55
E.1	Namelist variables	73

List of Figures

2.1	Flowchart of the main program	8
2.2	Flowchart of the time stepping routine.	9
2.3	Flowchart of the chemistry operator. The numbers in bold are the values of the namelist variable <i>chem_opt</i> that determines the chemistry option.. . . .	10

Chapter 1

Introduction

The Global Modeling Initiative (GMI) ¹ was initiated under the auspices of the Atmospheric Effects of Aircraft Program (AEAP) in 1995. The goal of GMI is to develop and maintain a state-of-the-art modular 3-D chemistry transport model (CTM) that can be used for assessment of the impact of various natural and anthropogenic perturbations on atmospheric composition and chemistry, including, but not exclusively, the effect of aircraft.

More recently, ACMAP has selected the approach of GMI to serve both as an assessment facility and a testbed for model improvements for future assessment in all areas of atmospheric chemistry. The goals in the design of GMI as assessment tool are [4]

1. The model should be well-characterized and thoroughly tested against observations.
2. The model should be able to test and compare a diversity of approaches to specific processes by being able to easily swap modules containing different formulations of chemical processes, within a common framework.
3. The model should be optimized for computational efficiency and be able to run on different platforms.
4. Model results should be examined by a large representation of the scientific community, thus facility consensus on the significance of assessment results.
5. Ultimately, the model integration could provide a unique assessment capability for other anthropogenic impacts of concern, by providing a testbed for other algorithms and intercomparisons used in assessment of those issues.

Many elements of the GMI model address these goals. The GMI model is a modular chemistry-transport model (CTM) with the ability to carry out multi-year assessment simulations as well as incorporate different modules, such as meteorological fields, chemical mechanisms, numerical methods, and other modules representing the different approaches of current models. This capability facilitates the understanding of the differences and uncertainties of model results.

The testing of GMI results against observations is a high priority of GMI activities. Science Team members contribute by either supplying a particular module and/or contributing

¹<http://gmi.gsfc.nasa.gov/gmi.html>

to the analysis of the results and comparison with atmospheric observations [3, 5]. Application of the model to the potential impacts of stratospheric aircraft emissions is presented in [4]. The model has been employed to investigate the effects of stratospheric aircraft emissions on the polar stratospheric clouds [2] and simulate ozone recovery over a 36-year time period [1].

Besides acting as a testbed for different modules, GMI will also act as a 3-D assessment facility. The GMI modular code is currently implemented at NASA/Goddard Space Flight Center (the core institution). The core institution is responsible for:

- Integrating and testing component of the GMI model,
- Maintaining coding standards which will make the model portable to different platforms
- Carrying out assessment calculations, and
- Providing first-order results and diagnostics for analysis by team members.

The current version code has been developed to run on a variety of computing platforms, both with single and multiple processors (SGI Origin series, HP Compaq SC45, Beowulf clusters, single processor workstations, etc.).

This report is intended to familiarize users with the GMI code. Users will be able to

- Have information on the code structure (Chapter 2).
- Obtain instructions on how to obtain the code, install it, compile it, run it on any platform (Chapter 3).
- Have a knowledge of all the input and output files involved in the code (Chapter 4, Appendix C and Appendix E).
- Carry out specific and restart runs (Chapter 5).
- Learn how to make changes in the code (Chapter 6).
- Execute useful script tools needed for instance to search for words, to produce restart input namelist files, etc. (Chapter 7).
- Learn basic CVS commands (Chapter 8).
- Analyze the parallel performance of the code (Chapter 9).
- Be familiar with include files used to select the desired architecture, to set up compilation options, etc. (Appendix A).
- Know how to carry out a single or multiple processor run (Appendix B).
- Use new features in the code (Appendix D).

Chapter 2

Structure of the Code

2.1 Components of GMI

The modules that make up the GMI assessment model are [5]:

1. Input meteorological data coming from three major Global Circulation Models (from NCAR, GISS and DAO). Data from all these input sets included horizontal U and V winds, temperature, and surface pressure.
2. Advection algorithm to transport trace species
3. Mass tendencies
4. Numerical schemes for chemistry solutions
5. Chemistry mechanism
6. Heterogeneous processes
7. Photolysis
8. Diagnostics
9. Tropospheric treatment
10. Initial conditions
11. Boundary conditions

All the above modules have multiple options. The GMI model incorporates four chemical mechanisms:

- Aerosol
- Stratosphere
- Troposphere
- Combined stratostphere/troposphere (`strat_trop`)

	aerosol	stratosphere	troposphere	strat_trop
# species	30	57	86	125
# thermal reactions	200	122	224	322
# photolytic reactions	61	44	50	82

Table 2.1: Chemical mechanisms

A summary of the mechanisms appears in Table 2.1.

2.2 Directory Structure of The Code

The top directory of the GMI code is *gem/* which contains the sub-directories

- *actm/*: for the atmospheric transport model
- *bin/*: location of the code executable
- *doc/*: general information about the code and how it is to be used
- *esm/*: Earth System Modeling package
- *esm_tools/*: tools for the ESM package
- *include/*: general-purpose header files for platform selection, compilation selection, message passing options, etc (see Appendix A).

In Table 2.2, we give more details on the structure of each of the above directories.

Directory Name	Synopsis
actm	Atmospheric Chemistry Transport Model
actm/gmimod	
actm/gmimod/Other	
actm/gmimod/Other/doc	README files
actm/gmimod/Other/misc	
actm/gmimod/Other/scripts	Script files performing various functions (see Chapter 7)
actm/gmimod/Other/test	
actm/gmimod/Other/test/nopar	
actm/gmimod/Other/test/nopar/Old	
actm/gmimod/Other/test/nopar/infiles	Sample input namelist files
actm/gmimod/Other/test/nopar/outfiles	ASCII output files for test cases in infiles/
actm/gmimod/Other/test/par	
actm/gmimod/Other/test/par/infiles	Sample input namelist files
actm/gmimod/Other/test/par/outfiles	ASCII output files for test cases in infiles/

2.2. Directory Structure of The Code

actm/gmimod/advec	Advection operator module
actm/gmimod/advec/dao2advec	DAO advection routines
actm/gmimod/advec/dao2utils	Advection utility routine computing, courant numbers, Divergence, etc.
actm/gmimod/advec/include	Advection include file
actm/gmimod/chem	Chemistry operator package
actm/gmimod/chem/aerosol	Aerosol chemistry
actm/gmimod/chem/aerosol/include_setkin	Include files for aerosol
actm/gmimod/chem/aerosol/setkin	Routines for rate constants and rates of kinetic
actm/gmimod/chem/include	Chemistry include file
actm/gmimod/chem/sad	Aerosol surface area density and condensed phase mixing ratio modules
actm/gmimod/chem/smv2chem	Chemistry solver routines
actm/gmimod/chem/strat_trop	Strat/Trop chemistry
actm/gmimod/chem/strat_trop/include_setkin	Include files for the combined strat/trop
actm/gmimod/chem/strat_trop/setkin	Routines for rate constants and rates of kinetic
actm/gmimod/chem/stratosphere	Stratospheric chemistry
actm/gmimod/chem/stratosphere/include_setkin	Include files for stratosphere
actm/gmimod/chem/stratosphere/setkin	Routines for rate constants and rates of kinetic
actm/gmimod/chem/sulfur	Routines for sulfur chemistry
actm/gmimod/chem/troposphere	Tropospheric chemistry
actm/gmimod/chem/troposphere/include_setkin	Include files for troposphere
actm/gmimod/chem/troposphere/setkin	Routines for rate constants and rates of kinetic
actm/gmimod/comm	MPI communication routines
actm/gmimod/control	Routines defining ACTM procedures (init, advance, final)
actm/gmimod/convec	Convection operator package
actm/gmimod/depos	Deposition operator package
actm/gmimod/depos/include	
actm/gmimod/diffu	Diffusion operator package
actm/gmimod/emiss	Emission operator package
actm/gmimod/emiss/Harvard	Harvard emission routines
actm/gmimod/emiss/include	
actm/gmimod/emiss/llnl	LLNL emission routines
actm/gmimod/in_out	Model input/output routines
actm/gmimod/include	
actm/gmimod/include_data	Include data files
actm/gmimod/mem_manage	Dynamic memory allocation routines
actm/gmimod/phot	Photolysis “operator” package
actm/gmimod/phot/fastj	Fast-J routines
actm/gmimod/phot/include	
actm/gmimod/phot/lookup	Routines implementing the photolysis lookup table

actm/gmimod/phot/utils	
actm/gmimod/step	GMI time stepping routine and control routines for operators (advection, chemistry, etc.)
actm/gmimod/trans	UCI transport package
actm/gmimod/trans/include	
actm/gmimod/trans/ucitrans	
actm/gmimod/trans/Uci_Data	
actm/lib	
bin	Location where the executable is placed
doc	General information on gem/ and how it is used
esm	All the esm-level source code and include file.
esm/comm	MPI communication routines for ESM
esm/control	Routine defining ESM procedures (generate, advance, terminate)
esm/in_out	Routine reading in the namelisted data for ESM
esm/include	
esm/lib	
esm/main	Main program of the code (esm_main.F)
esm/mem_manage	Routines for dynamic allocations of arrays for ESM
esm/utils	Utility routines such as timing, performance, statistics, flushing the buffer, etc.
esm_tools	All the esm_tools source code and include files
esm_tools/baseline	
esm_tools/fi	Routine for assigning unit numbers to input/output files.
esm_tools/include	
esm_tools/lib	
include	General-purpose header files that are used throughout the code (see Appendix A).

Table 2.2: GMI code directories

2.3 Coding Principles

A ".F" (Fortran code) or a ".c" (C code) suffix denotes source code files. The capital "F" suffix indicates that the Fortran source contains preprocessing directives. Files named with a ".h" suffix are header files that contain preprocessing directives, variable declarations, parameter definitions, and common block definitions. Contents of selected header files are included via *#include* statements at the beginning portion of each of the ".F" and ".c" files.

2.4. General Flowchart of the Code

To enable multiyear chemistry simulations, the GMI core model was parallelized to make use of the most powerful computational platforms available. The parallel strategy uses a two-dimensional longitude/latitude domain decomposition whereby each subdomain consists of a number of contiguous columns having a full vertical extent. Processors are assigned to subdomains, and variables local to a given package/subdomain are stored on the memory of the assigned processor. Data are transmitted between computational processes, when needed, in the form of messages. The number of meshpoints per subdomain may not be uniform, under the constraint that the decomposition be logically rectangular. The choice to decompose in only two dimensions is based on the fact that chemistry, photolysis, and cold sulfate algorithms make up the vast majority of the computational requirements and are all either local or column calculations. These computations require no communication with neighboring grid zones and hence maximize the parallel efficiency [5].

2.4 General Flowchart of the Code

In this section, we give through a flowchart the major routines called in the execution of the GMI code. We start from the main program and go down to the GMI time stepping routine.

The main program is `Esm_Main` (filename `./esm/main/esm_main.F`). This calls routines to initialize the MPI programming environment, to control the ESM package and to close the MPI programming environment (see Figure 2.1). The control routine (`Esm_control`) for the ESM package calls routines to set up ESM environment (`Esm_generate`), to advance physical process components of ESM (`Esm_advance`) and to terminate the ESM simulation (`Esm_terminate`). The routine `Esm_advance` itself points to `Gmimod` in order to move forward in time the model simulation: the met data are updated (as needed) and the time stepping routine is called.

The most important calculations done in the time stepping routine appear in Figure 2.2. The figure only shows the main components for the LLNLTRANS Transport case (`trans_opt=1` in the namelist file). We observe that the major operators (emission, diffusion, advection, convection, deposition and chemistry) are executed here.

A flow sequence for the chemistry operator is shown in Figure 2.3. It is important to note that

- The package Fast-J or the lookup table is employed in "Update photolysis rate constants" which computes the rate constants Q_j .
- The setkin routine *Kcalc* for calculating and returning rate constants (Q_k) is called in the box "Update the thermal rate constants".
- The chemistry solver SMVGEAR II is used in the box labeled "Solve the chemical ODEs".

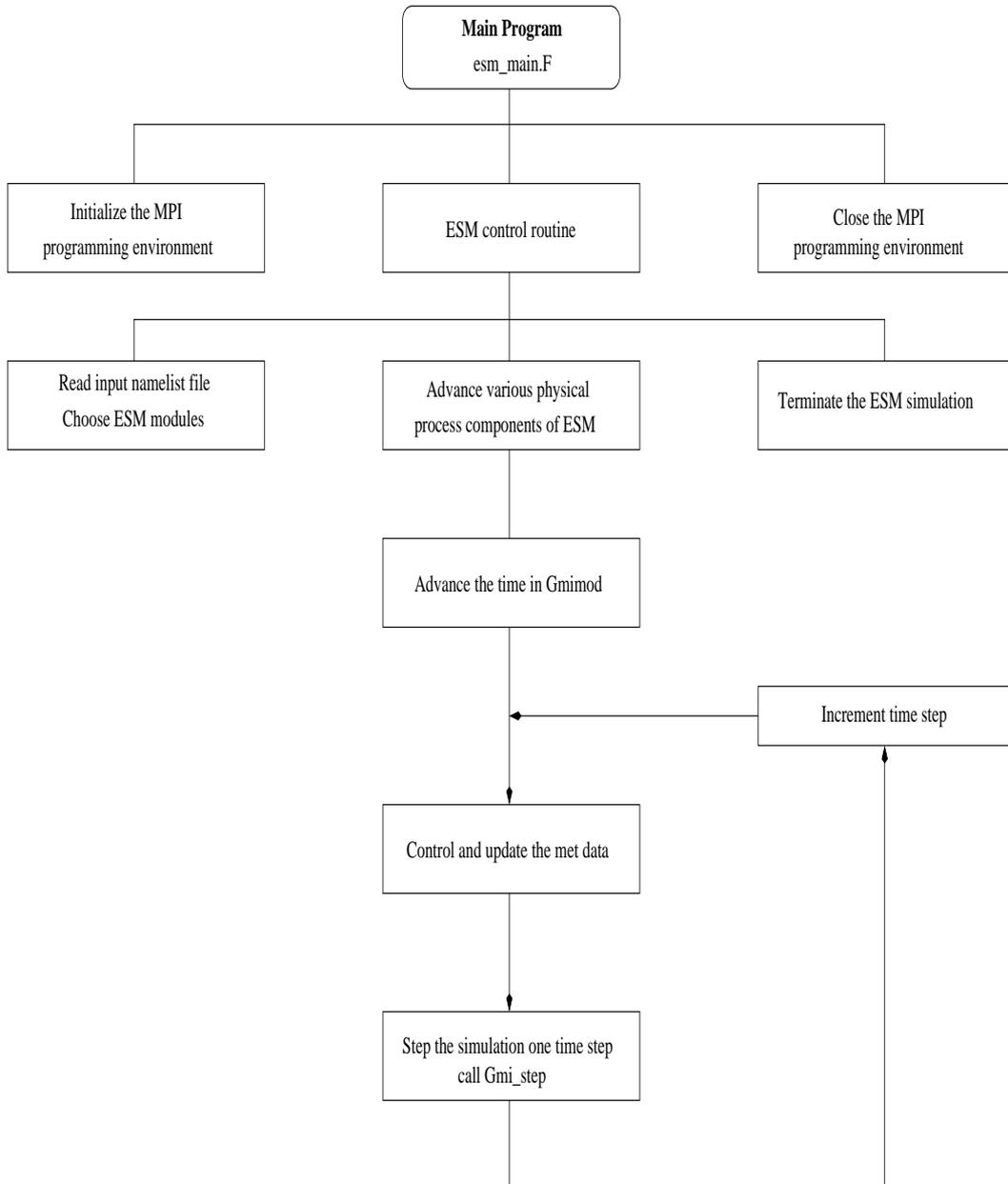


Figure 2.1: Flowchart of the main program

2.4. General Flowchart of the Code

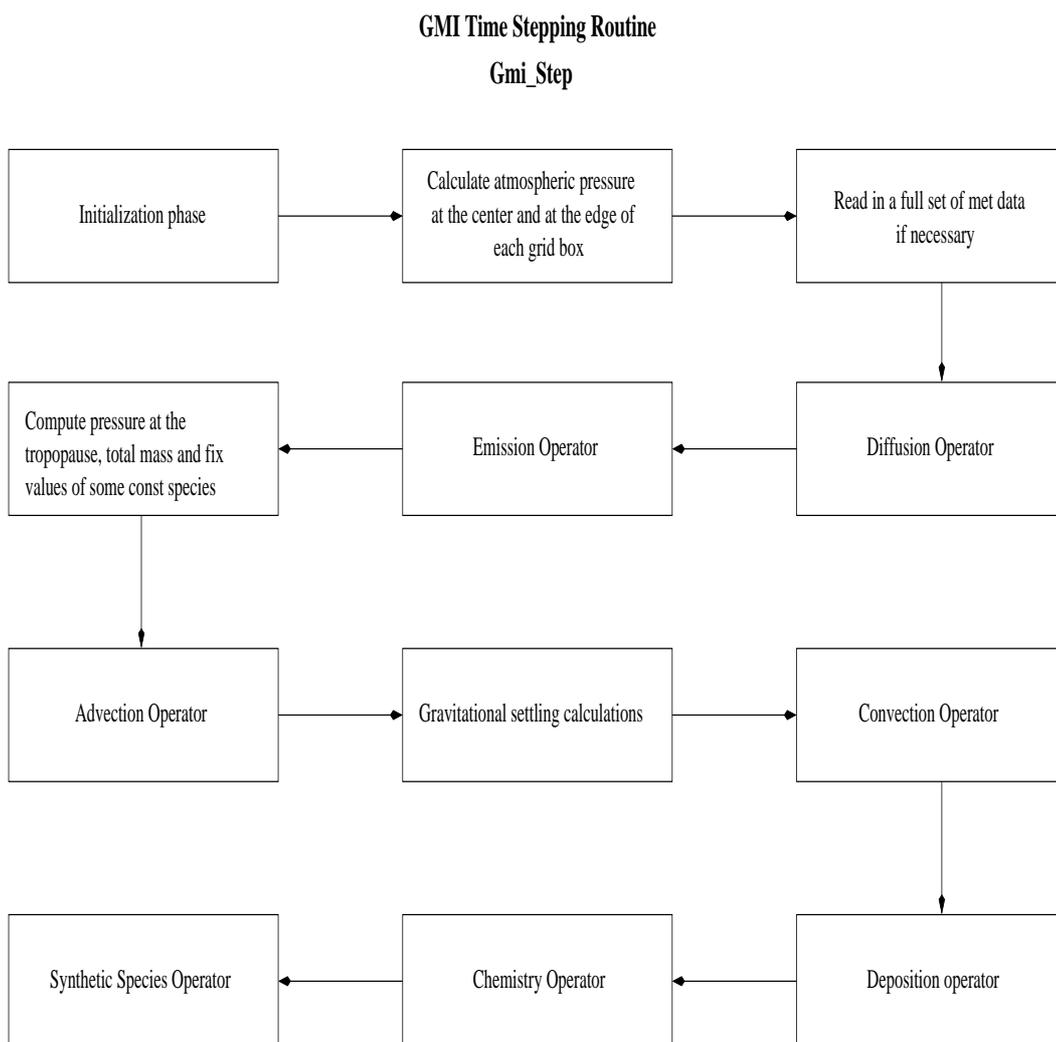


Figure 2.2: Flowchart of the time stepping routine.

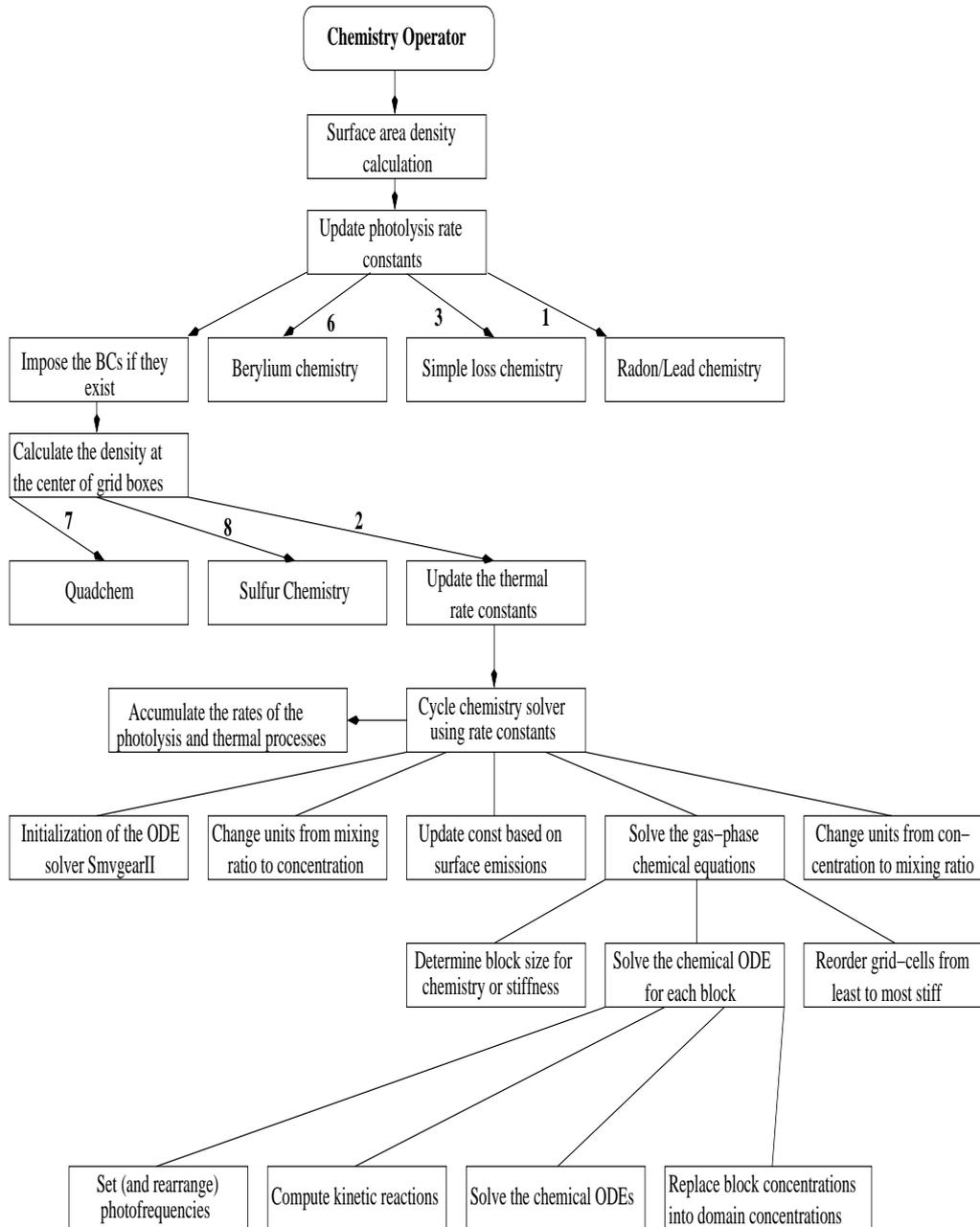


Figure 2.3: Flowchart of the chemistry operator. The numbers in bold are the values of the namelist variable *chem_opt* that determines the chemistry option..

Chapter 3

Installation and Testing

This chapter is written to help new users to install and test the GMI code. We provide specific instructions on how to obtain the code, to properly set environment variables, to select the model configuration, to choose a particular platform, to compile the code and to perform basic test runs. The focus of the document is for the installation and execution of the GMI code on `halem`, `daley` and `thunderhead`. The same procedures can easily be applied to any platform.

To get and install the GMI code, the following system software is needed:

- CVS (see Chapter 8 for instruction)
- F90/95 (ideally *ifc* for intel)
- C (ideally *icc* for intel)
- MPI needed only if running the message passing version of the code
- netCDF (version 3.4 or higher). The location of netCDF should be provided in the file `gem/include/gem_config.h` before compiling the code (see Section 3.3 for details).
- make
- makedepend (generally in `/usr/bin/X11`)
- perl
- a debugger (if possible)

During this process of installing and testing the code, it is assumed that *Cshell* is the default shell employed by the user. In fact, the GMI environment variables required for these procedures are set up using *Cshell*.

3.1 Getting the Code

To obtain the GMI code,

- Select the directory where you want to install the GMI model, say *MYGMI/*

- Get the latest version of the model from the cvs repository at *source motel* by typing the command lines:

```
%setenv CVS_RSH ssh
%cvs -d usrid@source motel.gsfc.nasa.gov:/cvsroot/gmi co gmi_gsfc
```

Here *usrid* is your login name on *source motel*. You will be asked to provide your password on *source motel*. The directory *gmi_gsfc/*, which is the main GMI directory, will then appear.

3.2 Model Files and Directory Structure

Move into *gmi_gsfc/*

```
%cd gmi_gsfc
%ls
```

You will find (in *gmi_gsfc/*) the files and directories:

```
CVS/          README.install  gem/          gmi_install
README.first  cshrc.ggmi      gmi_data/    login.ggmi
```

The top directory of the GMI code is *gem/* which contains sub-directories presented in Chapter 2.

3.3 Setting Environment Variables

In the directory *gmi_gsfc*, read all the README files by starting with *README.first* file that guides a new user to take the required steps for installing and running the GMI code. The top portions of the files *cshrc.ggmi* and *login.ggmi* include instructions for setting up the environment variables which are discussed in this section.

Edit the file *cshrc.ggmi*

- Select the chemistry mechanism you want to consider by setting the variable *CHEM-CASE*. Currently four mechanisms are available: *troposphere*, *aerosol*, *stratosphere*, and *strat_trop* (for the combined stratosphere/troposphere). If you want to have *stratosphere* for instance, uncomment the corresponding line to have

```
setenv CHEMCASE stratosphere
```

- For the platform you want the GMI model to run on, update the variables *GEMHOME* (location of the main model directory) and *GMI_DATA* (directory where the input data to test the installation are located):

```
setenv GEMHOME ~/MYGMI/gmi_gsfc/gem
setenv GMI_DATA ~/MYGMI/gmi_gsfc/gmi_data
```

3.3. Setting Environment Variables

Copy the files *cshrc.ggmi* and *login.ggmi* to your home directory

```
%cp cshrc.ggmi ~/.cshrc.ggmi
%cp login.ggmi ~/.login.ggmi
```

Go to the directory *gem/include/* and edit the file *gem_sys_options.h*. Modify the line

```
#define ARCH_OPTION ARCH_XXXX
```

to select the architecture you want to run the code on. *XXXX* is *COMPAQ* for *halem*, *SGLORIG* for *daley*, and *INTEL* for *thunderhead*.

Set the variable *MSG_OPTION* to determine if you want a single processor version of the code

```
#define MSG_OPTION MSG_NONE
```

or a multiple processor version (using MPI) of the code

```
#define MSG_OPTION MSG_MPI
```

You may also choose to edit the file *gem_options.h* to select debugging, optimization, or profiling options. If necessary, provide the paths to MPI and netCDF include files and libraries in the file *gem_config.h*. Some compilation options may have to be changed in the same file. Go to your home directory and edit the file *.cshrc*

```
%cd ~/
%vi .cshrc
```

Include the lines

```
setenv CVS_RSH ssh
setenv ARCHITECTURE ARCH_XXXX
if (-e ~/.cshrc.ggmi) then
    source ~/.cshrc.ggmi
endif
```

You can also edit the *.login* file and add the lines

```
if (-e ~/.login.ggmi) then
    source ~/.login.ggmi
endif
```

Update the changes made in the files *.cshrc* and *.login* by typing

```
%source .cshrc
%source .login
```

The setting of the environment variables ended with the previous two commands. The setting automatically creates aliases that allow the user to easily access the code directories and to execute scripts (see Chapter 7). For instance, by typing:

- `cd gem` or `cd $GEMHOME`, you will get to the code main directory.

- `cd phot`, you will move to the directory containing the photolysis operator package.
- `cd chem`, you will move to the directory containing the chemistry operator package.
- `seabf my_words`, you will search through all the Gem/Gmimod ".F" files for the string *my_words*.

3.4 Code Installation and Basic Test Run

Go back to the working directory *MYGMI/gmi_gsf/gem/*:

```
%cd gem
```

3.4.1 Compiling the model

To compile the code on **thunderhead** you first need to select the proper MPI environment variables. To view what is available, type

```
%mpi_env -p
```

Since we want to run with *intel*, type

```
%mpi_env -c intel
```

to change the MPI environment to *intel*.

To compile the code on all the platforms, use the commands:

```
%mkmf  
%make
```

The *make* command compiles and links the code. ".f", ".o" and ".a" files are created and executable named *gem* is placed in the directory *bin/*.

Remark 1 *On daley and thunderhead, you may get an error message caused by the fact that the f90 compiler complains that lines in some Fortran files use more than 72 columns. This is strange because the compilation options are properly set. To alleviate the problem, edit the file where the compilation failed and break the line(s) (into several ones) causing the problem.*

3.4.2 Testing the Executable

To test the executable, we will use a sample namelist file coming with the code. For each platform, we show examples of job script files (named *gmitest.job*) to test the executable. On **halem** and **daley** you need to have your sponsor code account (type the command *getsponsor* to obtain it).

It is assumed that the user wants to test the model from the directory */scratch/usrid* on **daley**, */scr/usrid* on **halem** and */mnt/pvfs/pvfs1/usrid* on **thunderhead**¹.

¹It is important to note that the model will only run on **thunderhead** from *pvfs*. In addition, the MPI commands work only with bash shell. Please refer to <http://newton.gsf.nasa.gov/thunderhead/> for details.

3.4. Code Installation and Basic Test Run

```
gmitest.job on daley
#!/bin/csh -fx
#PBS -N gmi_CO2run
#PBS -l ncpus=16
#PBS -l walltime=00:05:00
##PBS -l mem=2gb
#PBS -A a930b
#PBS -S /bin/csh
#PBS -V
#PBS -e error.file
#PBS -o output.file
#
cd /scratch/usrid
cp $GEMHOME/actm/gmimod/Other/test/par/infiles/dao2rn_dao46_06.in .
ln -s $GMI_DATA gmi_data
#
mpirun -np 16 $GEMHOME/bin/gem -d dao2rn_dao46_06.in

gmitest.job on halem
!/bin/csh
#BSUB -P a930b
#BSUB -J gmitest
#BSUB -n 16
#BSUB -W 00:20
#BSUB -q general_lng
#BSUB -o gmiout.%J
#BSUB -e gmierr.%J
#
cd /scr/usrid
cp $GEMHOME/actm/gmimod/Other/test/par/infiles/dao2rn_dao46_06.in .
ln -s $GMI_DATA gmi_data
prun -n 16 $GEMHOME/bin/gem -d dao2rn_dao46_06.in
```

Remark 2 *Replace a930b in the above script files with your sponsor code account. Here usrid is the user's login name.*

```
gmitest.job on thunderhead
### Number of nodes, processes and minutes requested
Nno=8
Npr=16
Nmi=60
ltmbegin -n $Nno -m $Nmi
#
WORK_DIR=/home/usrid/MYGMI/gmi_gsfc/gem
RUN_DIR=/mnt/pvfs/pvfs1/usrid
EXE_DIR=/home/usrid/rhome/MYGMI/gmi_gsfc/gem/bin
```

```

cd $RUN_DIR
cp $WORK_DIR/bin/gem $EXE_DIR
cp $WORK_DIR/actm/gmimod/Other/test/par/infiles/dao2rn_dao46_06.in .
### Submit the job
time ltmpr -n $Nno -t $Npr=16 $WORK_DIR/bin/gem -d dao2rn_dao46_06.in
### End the session
ltmend

```

To submit the job script, do the following

```

On daley
  %qsub gmitest.job
On halem
  %bsub < gmitest.job
On thunderhead
  %mkdir /home/usrid/rhome/MYGMI /home/usrid/rhome/MYGMI/gmi_gsfc
  %mkdir /home/usrid/rhome/MYGMI/gmi_gsfc/gem
  %mkdir /home/usrid/rhome/MYGMI/gmi_gsfc/gem/bin
  %bash
  %ltmsuper
  %./gmitest.job

```

Remark 3 *Though the thunderhead job script can only be submitted from the PVFS directory, the executable should not reside there. It is suggested to create the same directory structure (location of the executable) on the development node and the execute nodes. This explains the presence of the multiple mkdir commands.*

When the job is complete, compare the new output result with the previous result:

```

diff dao2rn_dao46_06.asc \
    $GEMHOME/actm/gmimod/Other/test/par/outfiles/dao2rn_dao46_06.asc

```

The matching results ensure the installation and compilation of GMI model is complete.

Remark 4 *Due to the changes made within the code, it may turn out that the verification process fails at the end of the run. That does not mean the installation of the code was unsuccessful. A failure may be due to the fact that the latest version of the code contains several updates (as a result of the bugs that were found). The comparison is done between the output from the new code and the output of the original one (containing bugs).*

3.5 Summary of the Necessary Steps

In this section, we give the list of steps needed to obtain, install and run the GMI code on any platform.

1. Obtain the code (*gmi_gsfc* release) from the cvs repository.
2. Move to the GMI working directory (*gmi_gsfc/*).

3.5. Summary of the Necessary Steps

3. Edit the file *cshrc.ggmi* to update the variables `GEMHOME`, `GMI_DATA` and `CHEMCASE`.
4. Copy the files *cshrc.ggmi* and *login.ggmi* to *.cshrc.ggmi* and *.login.ggmi* in your home directory.
5. Go to the directory *include/* to edit the files *gem_sys_options.h* and *gem_config.h* to select the architecture and to update the compilation options respectively.
6. Go to your home directory to edit and source the files *.cshrc* and *.login*.
7. Type `cd gem` and compile the code by typing `mkmf` then `make`.
8. Select a test case and in your input namelist file, update the variable `gmi_data_dir` providing the location of the input data.
9. Write a job script file and submit the job to run the executable.

Chapter 4

GMI Files

4.1 Input Namelist Files

Input namelist files are generally named <problem_name>.in. They allow many variables to be changed without having to recompile or relink the code. Each namelist file is broken into the categories

1. ESM SECTION
2. ACTM_CONTROL SECTION
3. ACTM_INPUT SECTION
4. ACTM_OUTPUT SECTION
5. ACTM_RESTART SECTION
6. ACTM_ADVEC SECTION
7. ACTM_CONVEC SECTION
8. ACTM_DEPOS SECTION
9. ACTM_DIFFU SECTION
10. ACTM_EMISS SECTION
11. ACTM_CHEM SECTION
12. ACTM_PHOT SECTION
13. ACTM_TRAC SECTION

Here are some basic requirements for editing namelist files:

- All namelist sections must be present and in proper order, even if no variable are listed in them.
- Variable names must be exact and placed in the proper section.

4.1. Input Namelist Files

- Real numbers need a “d” exponent, even if it is “d0”.
- Put a comma after each variable entry except the last one.
- End each section with a “/”.
- Some variable settings are incompatible with each other. The code does checking operations to catch these.

Remark 5 *All REAL*8 namelist variables need to be input with a "D" exponent (even if it is D0). The defaults below do not contain the "D" exponent, but be sure to add one to all the real variables referenced in your namelist input file.*

Appendix E gives a list of all the namelist variables, their types, and their description. Additional information can be obtained from the routine *Get_Namelist* located in the file *actm/gmimod/in_out/gmi_namelist.F*.

In the *gem/actm/gmimod/Other/test/par/infiles/* directory, sample input namelist files for different types of runs are available. They are:

- *dao2be_koch_ncar52_10.in*: the input file to make runs for beryllium. This uses Koch table for Be-7 and Be-10.
- *dao2be_nagai_ncar52_10.in*: used to make runs for Be-7 and Be-10 using Nagai table.
- *dao2co2_dao29_10.in*: for CO2 runs
- *dao2coldiag_ncar52_10.in*: for Beryllium runs
- *dao2isop_dao46_02.in*: used to make runs for Isoprene, Acetone, Propene and NO
- *dao2n2o_dao29_10.in*: for N2O runs
- *dao2n2o_dao44_10.in*: for N2O runs
- *dao2no_dao46_02.in*: for NO runs
- *dao2rn_dao46_06.in*: used to make runs for radionuclides Pb and Rn
- *dao2sf6_dao29_10.in*: for sulfur runs
- *gmit4_giss23_06.in*: for full chemistry runs
- *pfixllnl_dao29_04.in*: used to make runs with LLNL pressure fixer algorithm
- *pfiruci_dao29_04.in*: used to make runs with GMI pressure fixer algorithm (Prather's algorithm)

To run similar cases with GISS met fields and CCM3 met fields one can use the appropriate naming convention and then modify the input namelist files to reflect the corresponding cases. For instance

- If you want to make a radionuclide run for Pb and Rn, then the file `dao2rn_dao46_06.in` to `gissrn_giss23_06.in` and change the number of levels from 46 to 23 (`k2_gl = 23`), use GISS `met_infile_names` instead of DAO `met_infile_names`, etc.
- If you want to run for a year instead of six hours, then change `tfinal_days` to 365.d0.
- If you wish to print out monthly averages, then set `do_mean=T` and `pr_nc_period_days = -1.0d0`.

Remark 6 *In the input namelist file, the user must provide the exact path for `gmi_data_dir` and for other input files such as `met_infile_names`, `fixed_const_infile_name`, etc.*

For restart runs, there is a utility tool available to automatically create input namelist files. The procedure is described in Section 5.2.

4.2 Input Datasets

4.2.1 netCDF Input Files

Depending on the type of run you want to carry out, several netCDF files may be needed. The may include

- Met data
- An initial species concentration file
- A photolysis rate ("qj") file.
- An emission rate file
- Surface area density files.
- A restart file.

Met Fields Met fields from 3 different global models are used for GMI simulations. They are: DAO, GISS and CCM3. The files (located in the directory `MYGMI/gmi_gsfc/gmi_data/`) that come with the code are

```
(DAO Met Fields)
  DAO_GS_4x5x29_951101.nc
  DAO_GS_4x5x46_971023.nc
(GISS Met Fields)
  GISS_2prime_4x5x23_770701.nc
(CCM3 Met Fields)
  NCAR_CCM2_4x5x44_961115.nc
  NCAR_MATCH_4x5x52_970101.nc
```

Additional data files are available from the anonymous ftp site `dirac.gsfc.nasa.gov` in the directories:

```
pub/gmidata/input_data/met_field/GISS
pub/gmidata/input_data/met_field/DAO
pub/gmidata/input_data/met_field/MACCM3
```

4.2. Input Datasets

4.2.2 ASCII Input Files

A variety of ASCII input files are often used to run the model. We can mention for instance files containing data on soil type, leaf area index, vegetation type, fertilizer scale, etc. Some of the ASCII files are grid resolution independent while others are not.

ASCII data files and other miscellaneous files are available from the anonymous ftp site `dirac.gsfc.nasa.gov` in the directory:

```
pub/gmidata/input_data/misc
```

4.2.3 Input Files Needed for "troposphere" Chemical Mechanism

In this section, we list the files available to run a "troposphere" test case. They are not all required at the same time to run the case. The use of each of them depends on the logical variable selections made in the input namelist file.

1. Meteorological data files. Five different sets can be found on `dirac.gsfc.nasa.gov` and they are DAO, GISS, and MACCM3 (used for tropospheric simulations) and FVDAS and FVGCM (used for stratospheric simulations). To start this type of run, begin with the GISS fields. There are 24 files for an entire year (2 per month) and the first file name is *GISS_2prime_4x5x23_770101.nc*.
2. *GISS_2prime_4x5x23.in* - this is a file containing a list of meteorological data file names.
3. *GISS_gmit3_sad.nc* - This NetCDF file contains aerosol surface area densities which are used in the chemistry rate calculations (on the GISS vertical grid).
4. *fertscale_4x5_dao.asc* - Fertilizer scale data file needed by the emission calculation subroutine which was provided by Harvard.
5. *fixed_acetone_gmi_giss.nc* - This NetCDF file has monthly average values of acetone (on the GISS vertical grid)
6. *gmi_giss4x5_030520.nc* - This NetCDF file has monthly average emissions (on the GISS vertical grid).
7. *gmit_giss_jul_spin.in* - Namelist input deck to run the first 3 months of spinup
8. *gmit_giss_oct_spin.in* - Namelist input deck to run the second 3 months of spinup
9. *gmit_giss_apr_rst.nc* - This NetCDF file is a restart file from a previous simulation which is used to initialize the present simulation.
10. *isopconvtable.asc* - Isoprene conversion file needed by the emission calculation subroutine which was provided by Harvard.
11. *ju_atms.dat* - Data file needed by the FastJ photolysis package.
12. *ju_spec.dat* - Data file needed by the FastJ photolysis package.

13. *lai_4x5_dao.asc* - Leaf area index data file.
14. *lighttable.asc* - Light data file needed by the emission calculation subroutine which was provided by Harvard.
15. *monotconvtable.asc* - Monoterpene conversion file needed by the emission calculation subroutine which was provided by Harvard.
16. *precip_4x5_dao.asc* - Precipitation file needed by the emission calculation subroutine which was provided by Harvard.
17. *ratj.d* - Data file needed by the FastJ photolysis package.
18. *run_gmit4_giss_jul_spin* - short script which, when submitted to the batch system on halem, will run the initial 3 months of spinup.
19. *run_gmit4_giss_oct_spin* - short script which, when submitted to the batch system on halem, will run the second 3 months of spinup.
20. *soiltype.asc* - Soil type data file needed by the emission calculation subroutine which was provided by Harvard.
21. *uvalbedo.geos.4x5.asc* - Uv-albedo data file needed by the FastJ photolysis package.
22. *vegtype_4x5_dao.asc* - vegetation types.

4.3 Output Files

ASCII output and binary output files in netCDF data format are produced from GMI runs. The contents and the number of different output files can be controlled by using appropriate namelist parameters. To obtain information on how these parameters are set, we refer to Appendix E.

4.3.1 ASCII Diagnostic Output File

The following rules apply to the ASCII diagnostic output file:

- Is named <problem_name.asc>.
- Contains up to five sections, each of which can be turned on or off.
- For the first three sections, only information on a single specified species is output.
- Can specify a particular longitude index to use in the second section.
- Can specify the output frequency (in number of time steps).

4.3. Output Files

4.3.2 netCDF Output Files

The netCDF output files that can be produced by the GMI model are:

1. <problem_name>.const.nc
 - species concentration
 - [+mass]
 - [+ pressure and/or temperature]
 - [+dry depos. and/or wet depos.]
2. <problem_name>.col.nc
3. <problem_name>.flux.nc
4. <problem_name>.qj.nc
5. <problem_name>.qk.nc
6. <problem_name>.qqi.nc
7. <problem_name>.qqk.nc
8. <problem_name>.sad.nc

Each of the above netCDF output files

- Can be turned on or off.
- Can specify snapshots or mean values for most.
- Can specify frequency of output (number of days, monthly, and/or the 1st and 15th of each month).

In addition to the previous files, the code can also produce a netCDF restart file named <problem_name>.rst.nc. The file contains everything needed to do restart and can be written out with the frequency (namelist variable *pr_rst_period_days*): number of days, monthly, and/or the 1st and 15th of each month. There is a namelist variable (*do_overwrt_rst*) specifying whether to over-write or append to the file.

In Appendix C, we provide tables listing the contents of the netCDF output files and the frequency variables in them can be written out.

Chapter 5

Performing Specific Runs

5.1 Specific Runs

As for testing the model, we assume that the model will be run from the directory */scratch/usrid* on *daley*, */scr/usrid* on *halem* and */mnt/pvfs/usrid* on *thunderhead*. Copy the appropriate input namelist into the directory.

Suppose that you want to carry out a CO₂ run. The namelist file needed for such a run is *dao2co2_dao29_10.in*.

```
%cp $GEMHOME/gem/actm/gmimod/Other/test/par/infiles/dao2co2_dao29_10.in .
```

Edit the file *dao2co2_dao29_10.in* and provide the location of the *gmi_data/* directory (where the Met Fields reside) by setting

For *daley* and *thunderhead*

```
gmi_data_dir = '/home/usrid/MYGMI/gmi_gsfc/gmi_data'
```

For *halem*

```
gmi_data_dir = '/u1/usrid/MYGMI/gmi_gsfc/gmi_data'
```

The met input file needed for this run is *DAO_GS_4x5x29_951101.nc* that already resides in the *gmi_data/* directory.

For output files, the following setting

```
pr_diag      = F,  
pr_ascii     = T,  
pr_ascii3    = F,  
pr_netcdf    = T
```

will produce

```
dao2co2_dao29_10.asc  
dao2co2_dao29_10.const.nc
```

Remark 7 *Three namelist variables are important to choose the number of processors. They are NP_{actm} , NPI_{actm} and NPJ_{actm} , and they satisfy the relation $NP_{actm} = NPI_{actm} \times NPJ_{actm}$. To execute the code, the number of processors N_{cpus} is given by $N_{cpus} = NP_{actm} + 1$.*

5.1. Specific Runs

Here are examples of batch script files (named *gmiCO2run.job*) for *daley*, *halem* and *thunderhead* respectively. The run requires 16 processors.

```
gmiCO2run.job on daley
#!/bin/csh -fx
#PBS -N gmi_CO2run
#PBS -l ncpus=16
#PBS -l walltime=00:05:00
##PBS -l mem=2gb
#PBS -A k3002
#PBS -S /bin/csh
#PBS -V
#PBS -e error.file
#PBS -o output.file
#
cd /scratch/usrid
mpirun -np 16 $GEMHOME/bin/gem -d dao2co2_dao29_10.in
```

```
gmiCO2run.job on halem
#!/bin/csh
#BSUB -P a930b
#BSUB -J gmiCO2run
#BSUB -n 16
#BSUB -W 00:20
#BSUB -q general_lng
#BSUB -o /scr/usrid/gmiout.%J
#BSUB -e /scr/usrid/gmierr.%J
#
cd /scr/usrid
prun -n 16 $GEMHOME/bin/gem -d dao2co2_dao29_10.in
exit
```

```
gmiCO2run.job on thunderhead
### Number of nodes, processes and minutes requested
Nno=8
Npr=16
Nmi=60
ltmbegin -n $Nno -m $Nmi
#
WORK_DIR=/home/usrid/MYGMI/gmi_gsfc/gem
RUN_DIR=/mnt/pvfs/pvfs1/usrid
cd $RUN_DIR
cp $WORK_DIR/bin/gem /home/usrid/rhome/MYGMI/gmi_gsfc/gem/bin
### Submit the job
time ltmpi -n $Nno -t $Npr=16 $WORK_DIR/bin/gem -d dao2co2_dao29_10.in
```

```
### End the session
ltmend
```

5.2 Restart Runs

This section contains information on how to use Gem/Gmimod's restart capability.

To output restart data periodically during a run:

1. In the ACTM.RESTART section of the input namelist file, set

```
pr_restart          = T,
pr_rst_period_days = #.#d0,
```

Replace "#.#" with whatever you want the period to be.

Note that `pr_rst_period_days` when converted to seconds, must be a multiple of `mdt`, the time increment for reading in new met data.

2. Restart netCDF output will be written to a file named

```
<problem_name>.rst.nc
```

Set the namelist variable, `do_overwrt_rst` as appropriate (i.e., do you want to keep just a single set of restart data or multiple sets?).

To create a new namelist input file to use for running from a restart point:

1. Many people choose to create the new namelist input file manually, performing the same basic steps described in the more automated procedure below.
2. Alternatively, you can execute the `mk_rstnl` ("make restart namelist") script as follows

```
$ggmi/Other/scripts/mk_rstnl \
  -onl <old_nlfiler>  -nnl <new_nlfiler> \
  -rst <rst_ncfile>  -tfi <tfinal_days> \
  [-npr <problem_name>]
```

The first four arguments/value pairs are required

```
old_nlfiler  : name of the namelist input file that was used to get to
               the restart point
new_nlfiler  : name to call the new namelist input file that will be
               used to continue from the restart point
rst_ncfile   : name of the netCDF restart file that was written out at
```

5.2. Restart Runs

the restart point
tfinal_days : amount of time to run from the restart point; note that
tbegin_days will be set by the script to what tfinal_days
was when the netCDF restart file was written out

The last argument/value pair is optional

problem_name : new problem name to use in the new namelist input file

For example

```
$ggmi/Other/scripts/mk_rstnl -onl nfile.in.old -nml nfile.in.new \  
-rst nfile.rst.nc -tfi 10.0d0
```

This script automatically creates a new namelist file with various namelist variables modified or added to accommodate the restart. The script uses information from the netCDF restart file to accomplish this. Values that the script will use are output to the screen; these should be reviewed to verify that they are what you expected them to be.

The namelist variables that the script will potentially modify or add are:

```
ESM =>  
  problem_name      : set to value provided by -nnp script argument;  
                    otherwise unchanged from value in old_nfile  
  tbegin_days       : set to tfinal_days at restart point  
  tfinal_days       : set to value provided by -tfi script argument  
  
ACTM_CONTROL =>  
  start_ymd         : set to ending value at restart point  
  start_hms         : set to ending value at restart point  
  gmi_sec          : set to ending value at restart point  
  
ACTM_INPUT =>  
  met_infile_num    : set to ending value at restart point  
  mrnum_in          : set to ending value at restart point - 1  
  tmet1             : set to tmet2 at restart point  
  
ACTM_OUTPUT =>  
  pr_qqjk           : set to ending value at restart point  
  
ACTM_RESTART =>  
  rd_restart        : set to T  
  restart_infile_name : set to value provided by -rst script argument
```

Note that restart dumps can only occur at the end of a met data cycle.

You should edit the new namelist input file directly if you want to change any other namelist variables not listed above.

If there is more than one record in the restart file, the default is to use the last one. This can be changed by setting the namelist variable, *restart_inrec*, to a different record number.

To resume running from the restart point:

1. Start the run just as you normally would, but this time use the new restart namelist input file you created above.

Chapter 6

Making Changes to the Code

This chapter briefly describes coding considerations, recompiling and linking the code, and adding chemical mechanisms.

6.1 Coding Consideration

Some coding conventions are required before any change is made:

- All real variables must be declared "real*8".
- All real numbers must have a "d" exponent even if "d0".
- Must use "# include" for all include statements.
- Use only generic intrinsic function calls (for instance use MOD instead of AMOD).
- Do not use the real or float intrinsic functions, just assign integer variable to a real*8 variable if need be and then proceed.
- Do not use machine specific calls.
- Use of appropriate F90 language features is encouraged (dynamic allocation, array syntax, etc.).
- Use the physical constants and conversion factors defined in `actm/gmimod/include/gmi_phys_constants.h`.
- Use the time constants and conversion factors defined in `actm/gmimod/include/gmi_phys_constants.h`.

6.2 Adding Chemical Mechanisms

Currently, the GMI code has four chemical mechanisms:

1. *aerosol*

2. *stratosphere*
3. *strat_trop* (combined stratosphere/troposphere)
4. *troposphere*

The code portion for each mechanism is contained in its own subdirectory (located at *actm/gmimod/chem/*) having the name of the corresponding mechanism. It is organized into two subdirectories (see Chapter 2): *include_setkin/* and *setkin/*.

If you want to add another chemical mechanism, just create a new subdirectory from *actm/gmimod/chem/* and move the setkin files there. To compile the code, follow the procedures described in Section 6.4.

The selection of a particular chemical mechanism is done through the environment variable *CHEMCASE* in the file *cshrc.ggmi* (see Chapter 3).

6.3 The Make System

Makefile.cpp : a template for Makefile.

mkmf : a command line that uses Makefile.cpp and the system files to produce the Makefile.

make : to compile the code (use regular make, not GNU make).

make link : for linking.

make clean : to remove all the object files created at compilation.

6.4 Making Changes

After modifying any source file, do one of the following:

1. If you want to be absolutely certain that everything gets taken care of, type:

```
%cd $gem
%mkmf
%make
```

2. For changes to ".F" files, in the directory or directories where you made the changes, type:

```
%mkmf
%make
```

3. For changes to ".h" files, go to the highest level directory that contains any ".F" files that depend on the modified ".h" file, then type:

6.4. Making Changes

```
%mkmf
%make
```

Note that it is often easy to miss re-compiling something using this method. If you are uncertain exactly what directory to start from, use one of the other two options.

To re-link the code, we assume that the user's has made changes in a portion of the code and compiled that portion in the sub-directory where it resides. The re-linking is then done at the top directory by typing:

```
%cd gem
%mkmf
%make link
```

Sometimes it is a good idea to start with a fresh slate after repeated re-compilations or a large number of changes. To re-compile and re-link all of the files in the Gem/Gmimod directory tree, type:

```
%cd gem
%make clean
%mkmf
%make
```

Remark 8 *It is important to note that if you make changes specific to a particular chemical mechanism (i.e., outside setkin files), use the variable "chem_mecha" to delimit your changes. You need to include "gmi_chemcase.h" at the appropriate place. As example, consider the routine Remove_Trop_Water (in gmi_step_subs.F) that was modified to accomodate calculations specific for the strat_trop mechanism. The new code looks like*

```
subroutine Remove_Trop_Water
& (ih2o_num, const, pchem_water, strat_water)

implicit none

# include "gmi_dims.h"
# include "gmi_chemcase.h"

integer :: ih2o_num
real*8  :: const(i1:i2, ju1:j2, k1:k2, num_species)
real*8  :: pchem_water(i1:i2, ju1:j2, k1:k2)
real*8  :: strat_water(i1:i2, ju1:j2, k1:k2)

if (chem_mecha == 'strat_trop') then
  const(:, :, :, ih2o_num) = strat_water(:, :, :)
else
  const(:, :, :, ih2o_num) =
```

```
&      strat_water(:, :, :) *
&      (1.0d0 +
&      ((const(:, :, :, ih2o_num) - pchem_water(:, :, :)) /
&      pchem_water(:, :, :)))
endif

return

end
```

6.5 Debugging the Code

If you want to run the code in a debug mode,

- Edit the file *include/gem_option.h* and set the parameter *Debug_Option* to 1 and the other two options (optimization and profiling) to 0.
- Go to the main directory
- Type: `mkmf`
- Type: `make`
- Submit the executable using a debugger such as TotalView

Remark 9 *It is also possible to run only part of the code in a debug mode. You need to set `Debug_Option` (as above) and move to your desired directory where you need to execute the commands `mkmf` and `make`. Then move to the main directory and type `make link`.*

Chapter 7

Script Tools

The GMI code comes with several scripts that allow users to perform commands such as searching for words within the code, counting the number of lines in the source code, constructing a restart input namelist file, etc. All the scripts are located in the directory *gem/actm/gmimod/Other/scripts* and can be executed from anywhere in the code. They are:

check_ver searches for all file names in the current directory containing "search_string" and replaces the first instance of "search_string" with "replace_string".

Usage: `chname search_string replace_string`

clgem deletes a number of the files created when gem is run.

doflint runs the flint Fortran source code analyzer on the gem code.

It can be run on any machine where flint is available (e.g., tckk), and where the gem code has been installed and compiled.

Usage: `doflint`

gmi_fcheck can be used to run the Flint source code analyzer on Gem/Gmimod.

gmi_fcheckrm can be used after "gmi_fcheck" is run on the Gem/Gmimod code to strip out Flint messages that are of no consequence. Gmi_fcheckrm uses gmi_fcheck.out as its input file, and produces a new file called gmi_fcheckrm.out.

grabf grabs all of the Gem ".f" files and puts them in \$gem/CODE. These files then can be ftped to a machine with access to the FORTRAN Lint (flint) source code analyzer tool.

lastmod lists all Gem/Gmimod files in reverse order of when they were last modified. It is useful for determining which routines a user has modified since they were last installed in the code.

Usage: `lastmod [tail_num]`

lastmod_all lists all Gem/Gmimod files in reverse order of when they were last modified. It is useful for determining which routines a user has modified since they were last installed in the code.

Usage: `lastmod_all [tail_num]`

line_count_gem does a variety of line counts on the Gem/Gmimod source code.

line_count_gmi does a variety of line counts on the Gmimod source code.

list_species lists all the species labels for the selected chemical mechanism (environment variable CHEMCASE).

Usage: list_species

lnsdatt (lns (symbolic link) data) symbolically links the Gmimod input file directory to "gmi_data" in the current directory. The Gmimod namelist file can then point to a generic gmi_data directory.

mk_rstnl (make restart namelist) constructs a restart input namelist file (see Section 5.2 for its use).

savit creates a clean copy of a Gem/Gmimod code tree (considered only files with extension [.F—.c—.h]) in a tmp directory, tars it up into a tarfile named ggmsav.tar, and puts this file in the directory where your gem directory resides. The tmp files are then deleted.

savset creates a clean copy of the Gem/Gmimod setkin files (considered only files with extension [.F—.c—.h]) in a tmp directory, tars it up into a tarfile named setsav.tar, and puts this file in the directory where your gem directory resides. The tmp files are then deleted.

seabf searches through the Gem/Gmimod ".F" source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: seabf search_string [file_name]

seabf_word searches through the Gem/Gmimod ".F" source files for a particular word (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: seabf_word search_string [file_name]

seac searches through the Gem/Gmimod .c source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: seac search_string [file_name]

seah searches through the Gem/Gmimod ".h" include files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: seah search_string [file_name]

seah_word searches through the Gem/Gmimod ".h" include files for a particular word (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: seah_word search_string [file_name]

sealf searches through the Gem/Gmimod ".f" source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: sealf search_string [file_name]

seamf searches through the Gem/Gmimod "Makefile.cpp" source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.

Usage: seamf search_string [file_name]

Chapter 8

How to Use CVS?

8.1 What is CVS?

CVS is an acronym for the "Concurrent Versions System". It is a "Source Control" or "Revision Control" tool having the following features:

- Non-proprietary and can be downloaded from the internet;
- Allows users to work simultaneously on the same file, keep track of changes by revision, tag and date;
- Can obtain an earlier version of the software easily;
- Allows the user to track the supplier's software releases while making code changes locally.
- Enables the user to merge code changes between his version and supplier's automatically and identify problems if merge presents contradictions;
- A user of CVS needs only to know a few basic commands to use the tool.

Here are some important terms used with CVS:

Repository: The directory storing the master copies of the files. The main or master repository is a tree of directories.

Module: A specific directory (or mini-tree of directories) in the main repository. Modules are defined in the CVS modules file.

RCS: Revision Control System. A lower-level set of utilities on which CVS is layered.

Check out: To make a copy of a file from its repository that can be worked on or examined.

Revision: A numerical or alpha-numerical tag identifying the version of a file.

8.2. How to Use CVS?

8.2 How to Use CVS?

There are two ways you can use CVS:

1. Use CVS to keep up to date with the GMI code changes. This will require a *source-motel* account.
2. Use CVS to track both GMI code releases and your own changes. Again you can do this either on *source-motel* or on your local machine (with your own CVS installation).

8.3 Use CVS to Keep Up to Date with GMI Source Code Changes

CVS is used to keep track of collections of files in a shared directory called "The Repository". Each collection of files can be given a "module" name, which is used to "checkout" that collection. After checkout, files can be modified (using your favorite editor), "committed" back into the Repository and compared against earlier revisions. Collections of files can be "tagged" with a symbolic name for later retrieval. You can add new files, remove files you no longer want, ask for information about sets of files in three different ways, produce patch "diffs" from a base revision and merge the committed changes of other developers into your working files. In this section, we explain how these operations are done with the GMI code. It is assumed that the user has an account on *source-motel* and that CVS is installed on his local computer.

We assume that you have already obtained a copy of the code (say the most recent release) from *source-motel* by using the command:

```
%cvs -d usrid@source-motel.gsfc.nasa.gov:/cvsroot/gmi co gmi_gsfc
```

A directory labeled *gmi_gsfc* (containing the code) will be created at the location where the command was executed.

Assume that you want to know all the different available releases (with the associated tags) of the GMI code. From the *gmi_gsfc* directory, type

```
%cvs status -v cshrc.ggmi
```

to obtain the status of the file *cshrc.ggmi*. The results give

```
File: cshrc.ggmi          Status: Up-to-date

Working revision:      1.2
Repository revision:  1.2    /cvsroot/gmi/gmi_gsfc/cshrc.ggmi,v
Sticky Tag:           (none)
Sticky Date:         (none)
Sticky Options:      (none)

Existing Tags:
    COMBINED_STRAT-TROP_ADDED      (revision: 1.2)
```

```
READY_FOR_BOTH_SEQUENTIAL_PARALLEL      (revision: 1.2)
GMI_GSFC_BASELINE                        (revision: 1.1.1.1)
GMI_GSFC_STANDARD                        (revision: 1.1.1.1)
GMI_GSFC                                  (revision: 1.1.1.1)
GSFC_STANDARD                            (revision: 1.1.1.1)
LLNL_ALLCASE_HARVARDMOD_3                (revision: 1.1.1.1)
LLNL_TROPO_OLDWETDEPOS_3                 (revision: 1.1.1.1)
LLNL_AEROTROPO_MODIFIED_2                (revision: 1.1.1.1)
llnl_initial_3CHEMCASEs                  (revision: 1.1.1.1)
start                                     (revision: 1.1.1.1)
llnl                                       (branch: 1.1.1)
```

One can observe that the code has three releases (1.1.1, 1.1.1.1 and 1.2) and twelve associated tags.

```
%cvs export [-D today][-r tag] gmi_gsfc
```

gives exported version of the *gmi_gsfc* directory. The expressions in [] are options. ‘-D today’ gives the latest version of the code. The user can also specify “-D ‘June 12, 2004’” (note that the date is in single quote) for version of that day, or use ‘-r release-1-1-1-1’ for release 1.1.1.1 (release-1-1-1-1 is a CVS tag), or ‘-r LLNL_ALLCASE_HARVARDMOD_3’ for the *gem* directory with tag LLNL_ALLCASE_HARVARDMOD_3.

```
%cvs checkout gmi_gsfc
```

provides in addition to exported version, CVS information. With such information, users will be able to keep up-to-date with our release automatically with simple `cvs update` command (instead of having to manually insert the changes we broadcast). Once you check out a version of the code, you form a ‘working directory’.

```
%cvs update gmi_gsfc
```

only works if a user has cvs-checked-out version. This brings the changes made in the master repository to the user’s working directory. An example of the print out from this command:

Example 1 *You want to checkout a copy of the GMI code from sourcemotel. Then type*

```
%cvs checkout gmi_gsfc
```

It creates a copy of the code in your own directory. Assume you have made some changes in the code and the next code release arrives. You can simply do a `cvs update` to bring the new changes in the new release into your copy:

```
%cd gmi_gsfc
%cvs update
```

A list is printed on your screen to let you know which files were updated (a ‘U’ in front of the file) from the new release, and which files were modified (a ‘M’ in front of the file) and any conflict that may result from this update.

8.3. Use CVS to Keep Up to Date with GMI Source Code Changes

Remark 10 Note that doing `cv`s `update` under the `gmi_gsfc` directory will automatically update the entire code. You can update individual directory or file by going into the directory and do `cv`s `update`- which updates that directory and any sub-directories, or `cv`s `update filename`- which updates only that file.

```
%cv
```

s diff filename

This does the differencing between the file in your working repository with the one you checkout from the `sourcemotel` repository.

Example 2 Assume that you want to compare the file `gmi_step.F` from your working repository with the one on `sourcemotel` in the release with TAG `COMBINED_STRAT-TROP_ADDED`:

```
%cd gem/actm/gmimod/step
%diff -r COMBINED_STRAT-TROP_ADDED gmi_step_stub.F

Index: gmi_step_subs.F
=====
RCS file: /cvsroot/gmi/gmi_gsfc/gem/actm/gmimod/step/gmi_step_subs.F,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 gmi_step_subs.F
200a201,202
> CHARACTER*15 E_NAME
> EXTERNAL GETENV
210a213
> CALL GETENV('CHEMCASE',E_NAME)
212c215,220
< strat_water(:,:,:) = const(:,:,:,ih2o_num)
---
> IF (E_NAME == 'strat_trop') THEN
> strat_water(:,:,:) = MIN_STRAT_WATER
> const(:,:,:,ih2o_num) = strat_water(:,:,:)
> ELSE
> strat_water(:,:,:) = const(:,:,:,ih2o_num)
> ENDIF
229d236
<
275a283,284
> CHARACTER*15 E_NAME
> EXTERNAL GETENV
284a294,304
> CALL GETENV('CHEMCASE',E_NAME)
>
> IF (E_NAME == 'strat_trop') THEN
> const(:,:,:,ih2o_num) = strat_water(:,:,:)

```

```
> ELSE
>     const(:,:,:,ih2o_num) =
>     &     strat_water(:,:,:) *
>     &     (1.0d0 +
>     &     ((const(:,:,:,ih2o_num) - pchem_water(:,:,:)) /
>     &     pchem_water(:,:,:)))
>     ENDIF
286,290d305
<     const(:,:,:,ih2o_num) =
<     &     strat_water(:,:,:) *
<     &     (1.0d0 +
<     &     ((const(:,:,:,ih2o_num) - pchem_water(:,:,:)) /
<     &     pchem_water(:,:,:)))

%cvs log filename
```

This lists the log messages and status of the master repository.

Example 3 *Assume that you want to check the log messages and status of the file cshrc.ggmi.*

```
%cvs log cshrc.ggmi

RCS file: /cvsroot/gmi/gmi_gsfc/cshrc.ggmi,v
Working file: cshrc.ggmi
head: 1.2
branch:
locks: strict
access list:
symbolic names:
    COMBINED_STRAT-TROP_ADDED: 1.2
    READY_FOR_BOTH_SEQUENTIAL_PARALLEL: 1.2
    GMI_GSFC_BASELINE: 1.1.1.1
    GMI_GSFC_STANDARD: 1.1.1.1
    GMI_GSFC: 1.1.1.1
    GSFC_STANDARD: 1.1.1.1
    LLNL_ALLCASE_HARVARDMOD_3: 1.1.1.1
    LLNL_TROPO_OLDWETDEPOS_3: 1.1.1.1
    LLNL_AEROTROPO_MODIFIED_2: 1.1.1.1
    llnl_initial_3CHEMCASEs: 1.1.1.1
    start: 1.1.1.1
    llnl: 1.1.1
keyword substitution: kv
total revisions: 3;      selected revisions: 3
description:
-----
revision 1.2
```

8.4. Use CVS to Track Both New Releases and Your Changes

```
date: 2004/07/01 19:47:52; author: kouatch; state: Exp; lines: +12 -3
Modifications were made so that the code can be compiled and run on
a single processor or in parallel.
The user just needs to set the pre-processing option
MSG_OPTION (gem/include/gmi_sys_options.h) to MSG_NONE for single
processor case or to MSG_MPI for the parallel run.
In addition, a new pre-processing option MPI_2_OPTION was
introduced to add/remove MPI-2 calls in the code. Setting MPI_2_OPTION to
NO_MPI_2 will disable MPI-2 calls while setting it to WITH_MPI_2
will enable MPI-2 calls.
```

```
-----
revision 1.1
date: 2003/11/10 21:39:18; author: clune; state: Exp;
branches: 1.1.1;
Initial revision
```

```
-----
revision 1.1.1.1
date: 2003/11/10 21:39:18; author: clune; state: Exp; lines: +0 -0
imported from coremodel 030602 from LLNL
=====
```

8.4 Use CVS to Track Both New Releases and Your Changes

If you want to maintain your own code and keep track of the changes from *source motel*, what you should do is create your own repository and use the ‘vendor branch’ concept in CVS. If you do it from your local machine, set

```
setenv CVSROOT some-home-directory-on-your-local-machine
```

(e.g. `setenv CVSROOT /home/userid/gmi_repository`)
in your `.cshrc` file.

To initialize the repository, type

```
%cd /home/userid/gmi_repository
%cvs init
```

Now you can checkout any release of the GMI code. Assume that you want to obtain the release *READY_FOR_BOTH_SEQUENTIAL_PARALLEL*

```
%cvs -d usrid@source motel.gsfc.nasa.gov:/cvsroot/gmi co -r \
READY_FOR_BOTH_SEQUENTIAL_PARALLEL gmi_gsfc
```

If you only want the *gem/* directory, type

```
%cvs -d usrid@source motel.gsfc.nasa.gov:/cvsroot/gmi co -d gem -r \
READY_FOR_BOTH_SEQUENTIAL_PARALLEL gmi_gsfc/gem
```

You can now work with the code. If you make some changes and want to bring them to your repository for keep, do the following from the directory *gmi_repository/gmi_gsfc/gem/*:

```
%cvs diff > output.diff
%cvs update
%cvs commit -m 'message for the commit'
```

If you want to create a new file that does not exist in the repository and you want to add it in the repository, type (from the directory where the new file resides)

```
%cvs add new_file
```

Example 4 Assume that you want to add a new chemical mechanism, *new_chem*. In the directory *actm/gmimod/chem*, you have created the directory *new_chem/* that contains the subdirectories *include_setkin/* and *setkin/*. To add the directory structure of the new chemical mechanism into the repository, do the following:

```
%cd actm/gmimod/chem
%cvs add new_chem/
%cvs commit new_chem/
%cd new_chem
%cvs add include_setkin/
%cvs commit include_setkin/
%cd include_setkin
%cvs add *
%cd ../
%cvs add setkin/
%cvs commit setkin/
%cd setkin
%cvs add *
```

8.5 Where To Obtain CVS?

<https://ccvs.cvshome.org/servlets/ProjectDocumentList>

Chapter 9

Parallel Performance

One of the objectives of the Earth Space Data Computing Division (ESDCD) as part of the Earth Sciences Enterprise at NASA Goddard Space Flight Center (GSFC), is to make the computational capabilities of and parallel computers available to scientists involved in Earth Sciences research. ESDCD supports initiatives to report benchmark results of various atmospheric forecast models in order to indicate how well its NASA Center for Computational Science (NCCS) computers perform in comparison with other machines. In addition, ESDCD aims to provide these results to the scientific community and to create a user-friendly environment so that users of NCCS systems can easily access, run, and examine performance of climate and weather simulation codes. To this end, we propose to benchmark the Global Modeling Initiative (GMI) code.

In this report we present the parallel performance of the GMI code on the SGI Origin 3800, the Compaq SC45, and a linux cluster. Using test problems of different sizes, we analyze how efficient the code is on these computers.

9.1 Description of the Platforms

The NASA Center for Computational Sciences (NCCS) is the high-performance scientific computing facility operated, maintained, and managed by ESDCD. The NCCS mission is to provide computing resources and support services to enable space and Earth scientists who are currently funded by NASA Headquarters to accomplish their research goals. Among the computers available at NCCS, we can mention the SGI Origin 3800 (*daley*), the Compaq SC45 (*halem*), and a linux cluster (*thunderhead*). The characteristics of each of the platforms are shown in Table 9.1.

9.2 Parallel Performance

9.2.1 Description of the Test Cases

For our experiments, we use three test cases that differ with the chemical mechanism employed. They are summarized in Table 9.2.

	halem	daley	thunderhead
Processor	Dec Alpha	MIPS R12000	Pentium 4 Xeons
Proc/Node	4	4	2
Proc Speed	1.25Ghz	0.4Ghz	2.4Ghz
Memory/Node	2Gb	2Gb	1Gb
Cache/Proc		8Mb	
disk/Node		38Gb	80Gb
OS	OSF1 5.1	Irix 6.5	RedHat Linux 7.3
Interconnect			Myrinet 2000
Module/Compiler	fortran/55II	MIPSpro 7.2.1.2m	Fortran Intel 7.0

Table 9.1: Features of the different platforms.

	STRATOSPHERE	AEROSOL	TROPOSPHERE
Number of grid points	$72 \times 46 \times 28$	$72 \times 46 \times 46$	$72 \times 46 \times 46$
Model time step	3600s	3600s	3600s
Number of species	57	30	86
Beginning date	Jan. 1, 2000	Jan. 1, 1999	Jan. 1, 2002
Simulation length (days)	10	31	7
Chemistry option	SmygearII	Sulfur	SmygearII

Table 9.2: Information on the test cases.

9.2.2 Model Performance

For each of the test cases described in the previous section, we provide the wall clock time as function of the number of processes and processors. The results are presented in Table 9.3, Table 9.4, and Table 9.5

In each table, the first column lists the number of processors requested and the second column lists the number of processes (the decomposition gives the number of processes in the longitude and latitude directions and the master process) used. For some of the runs on *thunderhead*, we employed one process per node (each node has 2 CPUs). This gave us access to more computing resource.

In all the test cases, we note that in general, the computing time decreases as the number of processor increases. *halem* displays the smallest wall clock time.

9.3 Profiling the Code

Profiling a code can be defined as the used of software tools to measure a program's run-time characteristics and resource utilization. It is important to identify where the bottlenecks are and why these areas might be causing problems. By utilizing profiling tools and techniques, we want to to learn which areas of the code offer the greatest potential performance increase. We want to target the most time consuming and frequently executed portions of the program

9.3. Profiling the Code

STRATOSPHERIC TEST CASE				
CPUs requested	Processes used	halem	daley	thunderhead
16	$5 \times 3 + 1$	1226	3725	2786
	$3 \times 5 + 1$	1144	3719	2833
32	$5 \times 3 + 1$			2051
	$3 \times 5 + 1$			2011
	$6 \times 5 + 1$	747	2024	1612
	$5 \times 6 + 1$	802	2093	1627
62	$6 \times 5 + 1$			1158
	$5 \times 6 + 1$			1198
64	$9 \times 7 + 1$	527	1192	1011
	$7 \times 9 + 1$	504	1139	962
128	$9 \times 7 + 1$			741
	$7 \times 9 + 1$			731
	$14 \times 9 + 1$	413	782	712
254	$14 \times 9 + 1$			538

Table 9.3: 10-day simulation ($46 \times 72 \times 28$): wall clock time (in seconds) as function of the number of processors.

AEROSOL TEST CASE				
CPUs requested	Processes used	halem	daley	thunderhead
16	$5 \times 3 + 1$	1170	2552	2091
	$3 \times 5 + 1$	1084	2514	2045
32	$5 \times 3 + 1$			1560
	$3 \times 5 + 1$			1461
	$6 \times 5 + 1$	926	1610	1401
	$5 \times 6 + 1$	970	1655	1466
62	$6 \times 5 + 1$			1027
	$5 \times 6 + 1$			1077
64	$9 \times 7 + 1$	826	1218	1295
	$7 \times 9 + 1$	806	1181	1246
128	$9 \times 7 + 1$			948
	$7 \times 9 + 1$			896
	$14 \times 9 + 1$	790	1066	1332
254	$14 \times 9 + 1$			967

Table 9.4: 31-day simulation ($46 \times 72 \times 46$): wall clock time (in seconds) as function of the number of processors.

for optimization with the objective of reducing the overall wall clock execution time.

In this section, we give a quick profile of the code obtained by carrying out a 6-day

TROPOSPHERIC TEST CASE				
CPUs requested	Processes used	halem	daley	thunderhead
16	$5 \times 3 + 1$	1665	4304	3274
	$3 \times 5 + 1$	1501	4096	3068
32	$5 \times 3 + 1$			2283
	$3 \times 5 + 1$			2331
	$6 \times 5 + 1$	975	2294	1893
62	$5 \times 6 + 1$	1196	2402	1818
	$6 \times 5 + 1$			1377
64	$5 \times 6 + 1$			1551
	$9 \times 7 + 1$	822	1383	1355
128	$7 \times 9 + 1$	938	1292	1254
	$9 \times 7 + 1$			1050
254	$7 \times 9 + 1$			943
	$14 \times 9 + 1$	601		1088
	$14 \times 9 + 1$			830

Table 9.5: 7-day simulation ($46 \times 72 \times 46$): wall clock time (in seconds) as function of the number of processors.

simulation of the combined strat/trop chemical mechanism. The computations were done on *halem* using one processor. The profiling tool used was PIXIE (Instruction-counting profiler for optimization and coverage- analysis).

We count the number of instructions, cycles, flops, loads and loads followed by a load. The results appear in Table 9.6. We observe that (1) the code executes more than one instruction per cycle, (2) 27% of instructions are floating point operations and 28.6% of them are for memory access.

	Number performed	Ratio
Cycles	14577956172319	0.915
Instructions	15926111272717	1.000
Flops	4296235981869	0.270
Loads	4558051358935	0.286
Load followed by load	2064339526026	0.130

Table 9.6: Number of cycles and computer operations.

Table 9.7 reports the profiling of the code by providing for each routine the number of cycles used, the number of instructions, and the number of times it is called.

9.3. Profiling the Code

cycles	%cycles	cum%	instructions	c/i	calls	c/call	name
2106782961361	14.5	14.5	1890532428486	1.1	112135	18787916	decomp
2085230800969	14.3	28.8	2003809294042	1.0	26316	79238137	smvgear
1793047403937	12.3	41.1	1425798316323	1.3	112135	15990078	pderiv
1617179002710	11.1	52.1	1578159817171	1.0	393157	4113316	backsub
1385973782189	9.5	61.7	2087114432498	0.7	419473	3304083	subfun
1158256718609	7.9	69.6	711311082188	1.6	13248	87428798	conv_tran
874604074507	6.0	75.6	1152066322724	0.8	144	6073639406	update_wetdep
727473931580	5.0	80.6	768224797719	0.9	18837	38619416	fzppm
213659684666	1.5	82.1	561220762529	0.4	144	1483747810	lookup_qj
183904862421	1.3	83.3	211607494209	0.9	23766912	7738	calc_wet_loss_rate
174011101173	1.2	84.5	315338980788	0.6	18837	9237729	fyppm
168316421742	1.2	85.7	138527529965	1.2	37982880	4431	lmtppm
157083154722	1.1	86.7	201103904430	0.8	18837	8339075	calc_advect_cross_terms
155913150013	1.1	87.8	466932212882	0.3	2880	54136510	save_diag_llnl1
149069595190	1.0	88.8	263828405875	0.6	33688044	4425	fxppm
132060167424	0.9	89.7	148331834640	0.9	144	91708496	do_vert_diffu
124404183351	0.9	90.6	135120220190	0.9	18837	6604246	do_tpcore_dao2
107139490560	0.7	91.3	91457620992	1.2	119708928	895	kcalc_SKARR
106411371311	0.7	92.1	142186079913	0.7	144	738967856	do_convect_ncar
102225460428	0.7	92.8	134601252786	0.8	18837	5426844	yadv_dao2
86274439524	0.6	93.4	89565150402	1.0	18837	4580052	ytp
70097694849	0.5	93.8	96981861249	0.7	18837	3721277	xmist
65388652056	0.4	94.3	119830953879	0.5	18837	3471288	ymist
64036138023	0.4	94.7	93981912453	0.7	18837	3399487	xadv_dao2
62359030539	0.4	95.1	110808299628	0.6	18837	3310454	xtp
58010206457	0.4	95.5	129754132126	0.4	144	402848656	update_advect
56564025393	0.4	95.9	68765066730	0.8	18837	3002815	qckxyz
55948997828	0.4	96.3	59611502319	0.9	144	388534707	update_smv2chem
54474873836	0.4	96.7	97202628604	0.6	26316	2070029	calcrate
51092538590	0.4	97.0	104830067913	0.5	144	354809296	gmi_step
40506220080	0.3	97.3	71836767648	0.6	144	281293195	update_qk
29719020528	0.2	97.5	27905746264	1.1	953856	31157	kcalc_SKSTS_HOCL_HCL
28062443520	0.2	97.7	22315461120	1.3	4769280	5884	kcalc_FYRNO3
25958714112	0.2	97.9	47602660608	0.5	476928	54429	kcalc
20941907796	0.1	98.0	17827437900	1.2	953856	21955	kcalc_SKSTS_CLONO2_HCL
20180496816	0.1	98.2	16620294150	1.2	953856	21157	kcalc_SKSTS_HOBR_HCL
19722880512	0.1	98.3	22180013568	0.9	21938688	899	kcalc_SKLP
19092381480	0.1	98.4	16665641364	1.1	953856	20016	kcalc_SKSTS_CLONO2
15237849600	0.1	98.5	7268382720	2.1	14307840	1065	kcalc_SKFO
14776279534	0.1	98.7	35181139658	0.4	576	25653263	solve_block

Table 9.7: One-day simulation of the combined strat/trop chemical mechanism: breakup of the resources consumed by the major routines.

The GMI code uses a namelist variable, *do_ftiming*, if set to *.true.*, allows the profiling of the code, in particular how much time is spent on each operator. We did a 31-day simulation of the tropospheric chemical mechanism with initial conditions taken from January 1, 2002 at hour 00. The experiment were carried out on halem with 30 processors (the I/O processor not included).

Block	Min Time	Max Time	Avg Time
whole_actm	4782.5593	4782.5596	4782.5594
gmi_step	4544.8621	4559.7873	4552.7762
mgroupsync_beg	1.2370	16.1455	9.1320
gmi_step_first	0.0007	0.0104	0.0058
gmi_step_misc	16.7744	19.9291	17.8405
emiss	5.0691	6.1272	5.6206
diffu	21.4622	25.3274	22.3803
mgroupsync_advect	1.4002	6.2127	4.6978
advect	2386.5001	2390.1191	2388.4267
convect	90.8310	102.3972	94.2011
drydep	4.3659	7.7016	5.3814

wetdep	114.9082	131.7440	123.2534
addwat	2.2521	2.8839	2.5315
chem	531.3664	1403.0029	1104.1971
remwat	2.2020	2.6483	2.3287
synspc	2.5696	3.0228	2.7519
mgroupsync_ret	469.5580	1356.7404	769.9750

Table 9.8: 31-day simulation of tropospheric chemical mechanism: timing breakup of the different operators.

The results of Table 9.8 show that more than 50% of the time was spent on the advection operator. The chemistry operator took about 23% of the time. The fact that there is a ratio of about 2.6 between the maximum time and the minimum time spent by processes on the chemistry operator, shows that this operator has some load imbalance.

Bibliography

- [1] D.B. Considine, P.S. Connell, D.J. Bergmann, D.A. Rotman, and S.E. Strahan. Sensitivity of global modeling initiative ctm predictions of anartatic ozone recovery to gcm and das generated meteorological fields. preprint.
- [2] D.B. Considine, A.R. Douglass, P.S. Connell, D.E. Kinnison, and D.A. Rotman. A polar stratospheric cloud parameterization for the global modeling initiative three dimensional model and its response to stratospheric aircraft. *J. Geo. Res.*, 105(D3):3955–3973, 2000.
- [3] A.R. Douglass, M.J. Prather, T.M. Hall, S.E. Strahan, P.J. Rasch, L.C. Sparling, L. Coy, and J.M. Rodriguez. Choosing meteorological input for the Global Modeling Initiative assessment of high-speed aircraft. *J. Geo. Res.*, 104(D22):27545–27564, 1999.
- [4] D.E. Kinnison, P.S. Connell, J.M. Rodriguez, D.A. Rotman, D.B. Considine, J. Tannahill, R. Ramarosan, P.J. Rasch, A.R. Douglass, S.L. Baughcum, L. Coy, D.W. Waugh, S.R. Kawa, and M.J. Prather. The Global Modeling Initiative assessment model: application to high speed civil transport perturbation. *J. Geo. Res.*, 106(D2):1693–1711, 2001.
- [5] D.A. Rotman, J.R. Tannahill, D.E. Kinnison, P.S. Connell, D. Bergmann, D. Proctor, J.M. Rodriguez, S.J. Lin, R.B. Rood, M.J. Prather, P.J. Rasch, D.B. Considine, R. Ramarosan, and S.R. Kawa. The Global Modeling Initiative assessment model: model description, integration, and testing of the transport. *J. Geo. Res.*, 106(D2):1669–1691, 2001.

Appendix A

Include Files

The files presented here are located in *include/* directory. They are required for the preprocessing and compilation of the code. Any modification of these files leads to the execution of the commands `mkmf` for the creation of the Makefile file and `make` in order to obtain a new executable.

gem_config.h

Sets configuration parameters for gem Makefiles. The following information must be provided:

- Location of the netCDF include files (variable *INCLUDES_NETCDF*) and library (*NETCDFLibs*)
- Location of the MPI include files (variable *INCLUDES_MSG*) and library (*MSGLIBDIR*)
- Commands to call the C (variable *CC*) and Fortran (*FC*) compilers.
- Compilations options.

gem_options.h

To select the package, the chemical solver, and the desire compilation mode (debugging, optimization, profiling).

gem_sys_options.h

To select the architecture and the message passing options. The user must set the variables:

- *ARCH_OPTION*: to determine the platform used.
- *MSG_OPTION*: for the message passing option. Choose *MSG_NONE* if you want the single processor version of the code, or *MSG_MPI* if you prefer the multiple processor one with MPI as message passing.
- *MPI2_OPTION*: to determine if you want to include or not MPI-2 calls. Consider *NO_MPI2* if the platform you will be running the code on does not support MPI-2, otherwise choose *WITH_MPI2*.

gem_msg_numbers.h

Contains the message numbers used to identify specific messages in the message passing calls. Should not be edited.

gem_rules.h

Contains suffix dependencies and compilation rules for all Makefile.cpp files in directories that contain source files. Should not be edited.

Appendix B

Single/Multiple Processor Runs

We describe how to carry out single (MPI is not used) or multiple processor runs. Before you compile the code, you need to edit the file *include/gem_sys_options.h* and set

```
#define MSG_OPTION MSG_NONE
```

if you want to produce the executable for a single processor run, or

```
#define MSG_OPTION MSG_MPI
```

for the multiple processor run.

After you obtained the executable, you need to edit your input namelist file.

Namelist Variables for Single CPU

```
NP_actm      = 1
NPI_actm     = 1
NPJ_actm     = 1
```

Namelist Variables for Multiple CPUs

```
NP_actm      = 15
oneprcsr     = 0
NPI_actm     = 5
NPJ_actm     = 3
```

The following relationships should hold in order to run the code:

$$\begin{aligned} NP_actm &= NPI_actm \times NPJ_actm \\ \frac{i2_gl - i1_gl + 1}{NPI_actm} &\geq gmi_nborder \\ \frac{j2_gl - ju1_gl + 1}{NPJ_actm} &\geq gmi_nborder \end{aligned}$$

and the number of processors to be requested to submit the executable is equal to $NP_actm + 1$.

Here $i2_gl$, $i1_gl$, $j2_gl$, $ju1_gl$, and $gmi_nborder$ are namelist variables (see Appendix E).

Appendix C

GMI NetCDF Files

C.1 Contents of netCDF Files

Table C.1 gives a summary of the information contained in the netCDF files introduced in Section 4.3.2.

C.2 Output Frequency of NetCDF Files

For each of the files in Section 4.3.2, we provide in Table C.2 the frequency in which data can be written in the file.

File	Suffix	Variables	Mean	Snapshot	Buffered	Writer
Col	.col.nc	psf ^a const ^a		X X		Slaves
Const	.const.nc	const psf ^b kel ^b drydep ^b wetdep ^b semisss_out ^b mass ^b metwater ^b	X X X X X	X X X accum. accum. accum. X X	X	Master
Noon	.noon.nc	const_noon ^b metwater ^b qj qqj qqk	X X X X X	X	X X X X	Master
Mass Flux	.flux.nc	mf gmf amf	X X X	X X X	X X	Master
Qj	.qj.nc	qj opt_depth ^b O3+hv- >O1D ^b	X X ^c X ^c	X X X	X	Master
Qk	.qk.nc	qk	X	X	X	Master
Qqjk	.qqjk.nc	qqj qqk [yda]	X X	X X X	X X X	Master
Restart	.rst.nc	const h2ocond pctm2		X X X	X	Master
Sad	.sad.nc	sad hno3cond hno3gas h2oback h2ocond reffsts reffice vfall	X X X X X X X X	X X X X X X X	X	Master
Tend	.tend.nc	ncumt		X	X	Master

^a Very small subset, i.e., column info at a specific site; one site per file. Note also that this data is on its own/ different pressure grid.

^b Sub-selectable.

^c Option can be selected for local noon time output.

Table C.1: General netCDF files.

C.2. Output Frequency of NetCDF Files

File	Same/Nc Freq	Indepen Freq	Monthly	1st & 15th	Last Step
Col		X			
Const	X		X	X	X
Mass Flux	X		X	X	X
Qj	X		X	X	X
Qk	X		X	X	X
Qqjk	X		X	X	X
Noon		X	X	X	X
Restart		X	X	X	X ^a
Sad	X		X	X	X
Tend	X		X	X	X

^a Only if at the end of a met record.

Table C.2: Frequency of netCDF files.

File	Namelist Variables
Col	
Const	pr_const
Noon	pr_noon ^a
Mass Flux	pr_flux
Qj	pr_qj
Qk	pr_qk
Qqjk	pr_qqjk
Restart	pr_restart
Sad	pr_sad
Tend	pr_tend

^a May be automatically set in the code if some conditions are met (see Section D.1.6).

Table C.3: Namelist variables to produce netCDF files.

Appendix D

New Features

D.1 Diagnostics

During the model integration, there are several information that are written out but are not necessarily of interest to the user. For instance, the model saves out surface emission information for all the species while the user may only need data for emitted species only.

In the latest version of the code, we give the user the flexibility to select (at run time through namelist variables) the following:

- The set of species for surface emission diagnostics
- The set of species for dry deposition diagnostics
- The set of species for wet deposition diagnostics
- The set of species for tendency diagnostics
- The range of vertical levels for model outputs.

In addition, the noon species concentrations and variables saved around noon time are now written out in a separate file. All these features are important for at least two reasons:

1. Users will be able to choose species that produce non zero values in the diagnostics, and they will consider only vertical levels relevant for their experiments.
2. The sizes of output files will be significantly reduced.

It is important to note that tendency diagnostics are now written out in three dimensions.

D.1.1 Choice of Species for Surface Emission Diagnostics

To allow the selection of species, two new namelist variables were added:

pr_emiss_all : if set to T, then information on all the species is written out, otherwise only information for user specified species is provided (see next variable). This variable works only if **pr_surf_emiss=T**.

D.1. Diagnostics

pr_emiss_rec_flag(1:n) : used only if **pr_emiss_all=F**.

= 1, species selected

= 0, species not selected.

A sample namelist setting looks like (in the ACTM_OUTPUT section):

```
pr_surf_emiss = T,
.
.
.
pr_emiss_all = F,
pr_emiss_rec_flag(1) = 0,
pr_emiss_rec_flag(4) = 1,
pr_emiss_rec_flag(7) = 1,
pr_emiss_rec_flag(8) = 1,
pr_emiss_rec_flag(9) = 1,
pr_emiss_rec_flag(10) = 1,
pr_emiss_rec_flag(33) = 1,
pr_emiss_rec_flag(41) = 1,
pr_emiss_rec_flag(52) = 1,
pr_emiss_rec_flag(63) = 1,
```

Remark 11 *The script tool list_species (see Chapter 7) was written to allow users to list all the species available.*

D.1.2 Choice of Species for Dry Deposition Diagnostics

To allow the selection of species, two new namelist variables were added:

pr_drydep_all : if set to T, then information on all the species are written out, otherwise consider only the species provided by the user (see next variable). This variable works only if **pr_dry_depos=T**.

pr_drydep_rec_flag(1:n) : used only if **pr_drydep_all=F**.

= 1, species selected

= 0, species not selected.

A sample namelist setting looks like (in the ACTM_OUTPUT section):

```
pr_dry_depos = T,
.
.
.
pr_drydep_all = F,
pr_drydep_rec_flag(1 ) = 0,
pr_drydep_rec_flag(9 ) = 1,
pr_drydep_rec_flag(46) = 1,
pr_drydep_rec_flag(53) = 1,
pr_drydep_rec_flag(55) = 1,
pr_drydep_rec_flag(57) = 1,
```

D.1.3 Choice of Species for Wet Deposition Diagnostics

To allow the selection of species, two new namelist variables were added:

pr_wetdep_all : if set to T, then information on all the species are written out, otherwise consider only the species provided by the user (see next variable). This variable works only if `pr_wet_depos=T`.

pr_wetdep_rec_flag(1:n) : used only if `pr_wetdep_all=F`.
= 1, species selected
= 0, species not selected.

A sample namelist setting looks like (in the ACTM_OUTPUT section):

```
pr_wet_depos = T,  
.  
.  
.  
pr_wetdep_all = F,  
pr_wetdep_rec_flag(1 ) = 0,  
pr_wetdep_rec_flag(9 ) = 1,  
pr_wetdep_rec_flag(19) = 1,  
pr_wetdep_rec_flag(23) = 1,  
pr_wetdep_rec_flag(46) = 1,
```

D.1.4 Choice of Species for Tendency Diagnostics

Two new namelist variables are introduced to perform this function:

pr_tend_all : if set to T, then information on all the species is written out, otherwise only information specific to the species provided by the user is written out (see next variable). This variable works only if `pr_tend=T`.

pr_tend_rec_flag(1:n) : used only if `pr_tend_all=F`.
= 1, species selected
= 0, species not selected.

A sample namelist setting looks like (in the ACTM_OUTPUT section):

```
pr_tend = T,  
.  
.  
.  
pr_tend_all = F,  
pr_tend_all = F,  
pr_tend_rec_flag(1 ) = 0,  
pr_tend_rec_flag(9 ) = 1,  
pr_tend_rec_flag(10) = 1,  
pr_tend_rec_flag(19) = 1,
```

D.1. Diagnostics

```
pr_tend_rec_flag(23) = 1,  
pr_tend_rec_flag(46) = 1,  
pr_tend_rec_flag(51) = 1,  
pr_tend_rec_flag(52) = 1,  
pr_tend_rec_flag(53) = 1,  
pr_tend_rec_flag(55) = 1,  
pr_tend_rec_flag(57) = 1,  
pr_tend_rec_flag(82) = 1,  
pr_tend_rec_flag(86) = 1,
```

D.1.5 Choice of Vertical Levels

All 3D variables are currently saved out on all vertical levels when in fact the user only needs to analyze the output results on specific range levels. At run time, the user can set the following namelist variables:

pr_level_all : if set to T, all the vertical levels are considered, otherwise a range (selected in the next two variables) is used.

k1r_gl : First global altitude index for output. Should be at least equal to **k1_gl**.

k2r_gl : Last global altitude index for output. Should be at most equal to **k2_gl**.

A sample namelist setting looks like (in the ACTM_CONTROL section):

```
pr_level_all = F,  
k1r_gl      = 3,  
k2r_gl      = 20,
```

D.1.6 Noon Variables

Information on many variables at around noon time are saved out in netCDF output file. We want to provide to users a range of time they wish to save out noon variables and to write them in a unique file, <problem_name>.noon.nc. The variables are:

1. *const_noon* (written out if *noon_species* is set)
2. *metwater* (written out if *do_mean*=T, *noon_metwater*=T, and *pr_metwater*=T)
3. *qj* (written out if *do_mean*=T, *do_noon_rate*=T, and *pr_qj*=T)
4. *qqj* (written out if *do_mean*=T, *do_noon_rate*=T, and *pr_qqjk*=T)
5. *qjk* (written out if *do_mean*=T, *do_noon_rate*=T, and *pr_qqjk*=T)

Four new namelist variables were added

pr_noon : if set to true, the noon output file is created. the file will still be created if the any of the above conditions is satisfied.

noon_species_beg_time : begin time.

noon_species_end_time : end time.

pr_noon_period_days : noon variable output period.

A sample namelist setting looks like (in the ACTM_OUTPUT section):

```
do_mean = T,  
do_noon_rate = T,  
pr_metwater = T,  
noon_metwater =T,  
pr_qj = T,  
pr_qqjk = T,  
pr_noon = T,  
noon_species(25) = 1,  
noon_species(44) = 1,  
noon_species(52) = 1,  
noon_species(53) = 1,  
noon_species(56) = 1,  
noon_species_beg_time = 7.0d0,  
noon_species_end_time = 16.0d0  
pr_noon_period_days = -1.0d0
```

In the above namelist setting, *pr_noon* = T. Therefore the file <problem_name>.noon.nc will be created. It will contain information (between 7am and 4pm) of the variables *const_noon_mean*, *metwater_mean*, *qj_mean*, *qqj_mean* and *qqk_mean*. It is important to note the following:

1. If *do_mean*=F, the file <problem_name>.noon.nc will only have information on *const_noon*.
2. If *pr_noon* is not set at all in namelist file, the code will will automatically set it to true if one of the following four conditions is true:
 - *noon_species* is non zero for at least one specy
 - *pr_metwater*=T and *noon_metwater*=T
 - *pr_qj*=T and *do_mean*=T and *do_noon_rate*=T
 - *pr_qqjk*=T and *do_mean*=T and *do_noon_rate*=T
3. If *pr_noon* is set to false in the namelist file, then the file <problem_name>.noon.nc will not be created at all regardless of the setting of the other variables.

D.2 Tracer Runs

At run time users can determine if they want to carry out tracer runs. A new namelist section (*ACTM_TRAC*) was included with the following variables:

tracer_opt : integer variable taking two values: 0 for no tracer run and 1 for tracer runs.

efold_time : e-folding time to be adjusted for different tracers.

D.3. Addition of FastJX

```
&ACTM_TRAC
  tracer_opt = 1,
  fold_time  = 1800.d0 /
```

D.3 Addition of FastJX

FastJX was incorporated in the code. To run it, three files are needed. Their names are passed through namelist variables:

cross_section_file : X-Section quantum yield

rate_file : Master rate file

T_O3_climatology_file : T & O3 climatology.

A sample namelist setting reads

```
&ACTM_PHOT
  phot_opt = 8,
  do_clear_sky = T,
  uvalbedo_opt = 1,
  uvalbedo_init_val = 0.3d0,
  cross_section_file = 'jx_spec.dat',
  rate_file = 'rat_JX_combo.d',
  T_O3_climatology_file = 'jx_atms.dat' /
```

Note that fastJX is chosen when *phot_opt*=8.

Appendix E

Input Namelist Variables

A model run is controlled using a namelist input file named $\langle \textit{problem_name} \rangle .in$. The focus of this section is to describe the variables that constitute the namelist input. We provide the name of each variable, the default, and a brief description ¹.

Variable Name	Type	Default Value	Description
ESM SECTION			
problem_name	C*128	'gmi_test'	The name of the problem to be run.
timer_esm	I	0	Enables the timing routines for the esm; a value of 1 enables monitoring of the entire esm simulation.
NP_actm	I	0	A positive value enables the esm package "actm" and sets the number of processors for the actm package to NP_actm; to run Gem/Gmimod, NP_actm must always be set to ≥ 1 .
oneprcsr	I	0	A value of 1 turns the esm into a single processor code; the default is to run with master and slave processors. To run Gem/Gmimod, normally set to 0 for parallel machines, and to 1 for non-parallel machines.
day0	I	0	The integer day of the year at time 0.
tbegin_days	R	0.0	The starting time of the esm simulation (days).
tfinal_days	R	0.0	The ending time of the esm simulation (days).
ACTM_CONTROL SECTION			
do_ftiming	L	F	Do fine timing?
Processor distribution:			
NPI_actm	I	1	Number of processors in the i direction (longitude).
NPJ_actm	I	1	Number of processors in the j direction (latitude).
Global dimension info:			
gmi_nborder	I	4	Number of longitude and latitude ghost zones.
i1_gl	I	1	Index of first global longitude (no ghost zones).
i2_gl	I	72	Index of last global longitude (no ghost zones).
jul_gl	I	0	Index of first global "u" latitude (no ghost zones).

¹This was taken from *gem/actm/gmimod/Other/doc/README.namelist*

jv1_gl	I	1	Index of first global "v" latitude (no ghost zones).
j2_gl	I	46	Index of last global "u&v" latitude (no ghost zones).
k1_gl	I	1	Index of first global altitude (no ghost zones).
k2_gl	I	29	Index of last global altitude (no ghost zones).
num_species	I	1	Number of species.
pr_level_all	L	T	Should output be done on all the vertical levels? If pr_level_all=F, then set k1r_gl and k2r_gl
k1r_gl	I	1	First altitude index for output ($k1r_gl \geq k1_gl$).
k2r_gl	I	29	Last altitude index for output ($k2r_gl \leq k2_gl$).
Time:			
leap_year_flag	I	0	Leap year flag: < 0: no year is a leap year = 0: leap years are determined normally > 0: every year is a leap year
start_hms	I	000000	Starting hour/min/sec (HHMMSS).
start_ymd	I	890101	Starting year/month/day (YYMMDD).
gmi_sec	R	0.0	Total Gmimod seconds (s).
tdt	R	180.0	Model time step (s).
Main transport option:			
trans_opt	I	1	Transport option: 1: do LLNLTRANS transport 2: do UCITRANS transport (non-parallel mode)
loss_opt	I	0	range 0-1 The loop that accounts startospheric loss for full chemistry runs is activated when loss_opt = 1
ACTM_INPUT SECTION			
General input data:			
gmi_data_dir	C*80	' '	Directory where input files are located.
hdr_var_name	C*32	'hdr'	NetCDF header variable name.
hdf_dim_name	C*32	'hdf_dim'	NetCDF header dimension name.
lat_dim_name	C*32	'latitude_dim'	NetCDF latitude dimension name.
lon_dim_name	C*32	'longitude_dim'	NetCDF longitude dimension name.
prs_dim_name	C*32	'pressure_dim'	NetCDF pressure dimension name.
spc_dim_name	C*32	'species_dim'	NetCDF species dimension name.
rec_dim_name	C*32	'rec_dim'	NetCDF record dimension name.
tim_dim_name	C*32	'time_dim'	NetCDF time dimension name.
Met data:			
met_opt	I	3	Met input option: 1: use values fixed in code for u, v, ps, kel; no other data set 2: read in a minimal set of met data: u, v, ps, & kel; no other data set 3: read in a full set of met data.
met_grid_type	C	'A'	Met grid type: 'A': use A grid (DAO, NCAR(CCM)) 'C': use C grid (GISS)
mdt	R	21600.0	Time increment for reading new met data;

Chapter E. Input Namelist Variables

			must be a multiple of tdt (s).
do_cycle_met	L	F	When the last met input file has been read, should the code cycle back and continue with the first file again?
do_timinterp_met	L	T	Should the met fields, except the winds, be time interpolated?
do_timinterp_winds	L	T	Should the wind fields be time interpolated? Note that pressure fields are always interpolated.
do_wind_pole	L	F	When met_opt = 1, should the transport be over the poles or around the equator?
met_infile_num	I	1	Index of NetCDF file to start reading met input data from.
mrnum_in	I	1	NetCDF file record to start reading met data from.
tmet1	R	0.0	Time tag of the mrnum_in (s).
do_read_met_list	L	F	Should the met file names be read in from met_filnam_list?
met_filnam_list	C*80	'met_filnam_list.in'	Name of file to get names of met input files from. Note that currently this file must reside in the same directory that you are running the gem executable from.
met_infile_names()	C*128	' '	An array of met input file names (list may be used instead).
Species (i.e., "const") input data:			
Base const units = mixing ratio			
const_opt	I	2	Const input option: 1: set const values to const_init_val 2: read in const values 3: solid body rotation 4: dummy test pattern with linear slope in each dimension 5: exponential in vertical (decays with height) 6: sin in latitude (largest at equator) 7: linear vertical gradient 8: sin in latitude (largest at equator) + vertical gradient
mw()	R	0.0	Array of species' molecular weights (g/mol).
const_init_val()	R	1.0d-30	When const_opt = 1, this array of values will be used to initialize each const species (note that if a negative const_init_val() marker is set in the namelist input file, all of the const_init_val's from the negative value on will be set to the value preceding the negative value).
const_infile_name	C*128	' '	Constituent input file name.
const_var_name	C*32	'const'	NetCDF constituent variable name.
const_labels()	C*24	' '	Constituent string labels.
fixed_const_timpyr	I	12	Fixed const times per year: 1: one set of emissions per year (yearly) 12: twelve sets of emissions per year (monthly)
fixed_const_map()	I	0	Mapping of fixed const number to const species number.

fixed_const_infile_name	C*128	' '	Fixed const input file name.
io3_num	I	0	Index of ozone constituent.
ACTM_OUTPUT SECTION			
ASCII output:			
Terminal screen output:			
pr_diag	L	F	Print some diagnostic output to screen?
pr_time	L	T	Should the time be printed to the terminal screen each time step (if false, will still get time output to the screen at the end of each day)?
Namelist file output:			
pr_nl	L	F	Should all the namelist variables be written to a file (problem_name.nl)?
Species/Mass ASCII file output:			
pr_ascii	L	T	Should the ASCII output file be written at all?
pr_ascii1	L	T	Should the first section of the ASCII output file be written (the mass data)?
pr_ascii2	L	F	Should the second section of the ASCII output file be written (the species concentration data)?
pr_ascii3	L	T	Should the third section of the ASCII output file be written (the species concentration min/maxs)?
pr_ascii4	L	F	Should the fourth section of the ASCII output file be written (total mass of each species)?
pr_ascii5	L	F	Should the fifth section of the ASCII output file be written (total production and loss of each species)?
ascii_out_n	I	1	Single species index to use.
ascii_out_i	I	1	Longitude index to use in the second section.
pr_ascii_step_interval	I	1	Interval for ASCII output: > 0: ASCII output at specified step interval = -1: ASCII output at monthly intervals
SmygearII file output:			
pr_smv2	L	F	Should the SmygearII output file be written (non-parallel mode only)?
General NetCDF output:			
pr_netcdf	L	T	Should any of the periodic output files be written at all?
pr_const	L	T	Should the periodic species concentrations output file be written?
pr_psf	L	F	Should the surface pressures also be written to the periodic const output file?
pr_kel	L	F	Should the temperatures also be written to the periodic const output file?
pr_mass	L	F	Should the mass also be written to the periodic const output file?
pr_metwater	L	F	Should the meteorological water also be written to the periodic const output file?
pr_dry_depos	L	F	Should the dry depositions also be written to the periodic const output file?
pr_wet_depos	L	F	Should the wet depositions also be written to

Chapter E. Input Namelist Variables

			the periodic const output file?
pr_surf_emiss	L	F	Should the surface emissions also be written to the periodic const output file?
pr_flux	L	F	Should the periodic flux diagnostics output file be written?
pr_qj	L	F	Should the periodic qj output file be written?
pr_qj_o3_o1d	L	F	Should the special reaction O3→O1D be saved with the qj's?
pr_qj_opt_depth	L	F	Should the optical depth be saved with the qj's?
pr_qk	L	F	Should the periodic qk output file be written?
pr_qqjk	L	F	Should the periodic qqjk output file be written?
pr_sad	L	F	Should the periodic sad output file be written?
pr_tend	L	F	Should the periodic tendency diagnostics output file be written?
pr_const_all	L	T	Should all of the species concentrations be written to the periodic const output file?
do_mean	L	F	Should means or current values be put in the periodic output files?
do_noon_rate	L	F	Should photolysis rate constants be calculated at local Noon?
do_qqjk_inchem	L	F	If pr_qqjk is on, should qqj's & qqk's be determined inside the chemistry solver, or outside?
noon_metwater	L	F	Should the metwater be calculated at local noon?
flux_var_name	C*8	'mf'	NetCDF flux variable name.
noon_species(1:n)	I	0	If non-zero and do_mean is on, then only accumulate information on this species at around local Noon by default (11:00 - 13:00) or between noon_species_beg_time and noon_species_end_time.
noon_species_beg_time	R*8	11.d0	Beginning time for noon_species constituent concentration.
noon_species_end_time	R*8	13.d0	Ending time for noon_species constituent concentration.
pr_const_rec_flag(1)	I	1	Should species #1 be written to the const output file (0=no, 1=yes)?
pr_const_rec_flag(2:n)	I	0	Should species #n be written to the const output file (0=no, 1=yes)?
			Note that pr_const_rec_flag only used if pr_const_all is false. Note that pr_const_rec_flag is also used to determine the species written to dry_depos and wet_depos and the flux file
pr_nc_period_days	R	1.0	NetCDF output period: > 0.0: periodic output at specified interval (days) -1.0: periodic output at monthly intervals -2.0: periodic output on 1st & 15th of each month
pr_emiss_all	L	T	Should all the surface emissions be written to the periodic const output file? If set to F and pr_surf_emiss=T, then specify pr_emiss_rec_flag.
pr_emiss_rec_flag(1:n)	I	0	Should species #n surf_emiss diagnostic be written the const output file (0=no, 1=yes)?

pr_drydep_all	L	T	Should all the dry depositions be written to the periodic const output file? If set to F and pr_dry_depos=T, then specify pr_drydep_rec_flag.
pr_drydep_rec_flag(1:n)	I	0	Should species #n dry_dep diagnostic be written to the const output file (0=no, 1=yes)?
pr_wetdep_all	L	T	Should all the wet depositions be written to the periodic const output file? If set to F and pr_wet_depos=T, then specify pr_wetdep_rec_flag.
pr_wetdep_rec_flag(1:n)	I	0	Should species #n wet_dep diagnostic be written to the const output file (0=no, 1=yes)?
pr_tend_all	L	T	Should periodic tendency diagnostics output file be written for all the species? If set to F and pr_tend=T, then specify pr_tend_rec_flag.
pr_tend_rec_flag(1:n)	I	0	Should species #n tendency diagnostic be written to the tendency output file (0=no, 1=yes)?
Column diagnostic NetCDF output:			
col_diag_num	I	0	Number of column diagnostic sites.
col_diag_period	R	3600.0	Column diagnostics output period (s).
col_diag_site()	C*24	' '	Names of site locations for column diag.
col_diag_species(1)	I	1	Should species be included in column diag.?
col_diag_species(2:n)	I	0	
col_diag_pres(1:10)	R	1000.0, 900.0, ... , 100.0	Pressure levels for column diag. (mb).
col_diag_lat_lon(2,n)	I	0.0	Lat/Lon location of each column diag. site.
ACTM_RESTART SECTION			
pr_restart	L	F	Should a restart file be written?
do_overwrt_rst	L	T	Should the restart file be over-written?
pr_rst_period_days	R	7.0	Restart output period: > 0.0: restart output at specified interval (days) -1.0: restart output at monthly intervals -2.0: restart output on 1st & 15th of each month
rd_restart	L	F	Should a restart file be read?
restart_infile_name	C*128	'gmi.rst.nc'	Name of restart input file; note that currently this file must reside in the same directory that you are running the gem executable from.
restart_inrec	I	last record # in rst file	Record number in restart (rst) input file to read from.
ACTM_ADVEC SECTION (advection)			
advec_opt	I	1	Advection option: 0: no advection 1: do DAO2 advection
press_fix_opt	I	1	Pressure fixer option: 0: no pressure fixer used 1: LLNL pressure fixer used (Cameron-Smith) 2: UCI pressure fixer used (Prather)
pmet2_opt	I	1	pmet2 option: 0: use pmet2 1: use (pmet2 - "global mean change in surface

Chapter E. Input Namelist Variables

			pressure”)
advec_consrv_opt	I	2	Advection conserve option: 0: conserve tracer conc.; use pmet2 1: conserve tracer mass; use pmet2 2: conserve both tracer conc. & mass; use pctm2 Note that if press_fix_opt = 0 & advec_consrv_opt = 2, the code will generate an error and exit.
advec_flag_default	I	1	Set all species to do advection or not to do advection as the default; can then use advec_flag turn individual species either off, if the default is on; or on, if the default is off: 0: do not advect any species as default 1: advect all species as default
advec_flag()	I	advec_flag_default	An array of flags that indicate whether or not to advect a given species; if not explicitly set for a particular species, advec_flag_default is used: 0: no transport 1: transport
j1p	I	3	Determines size of the Polar cap; $j2p = j2_gl - j1p + 1$
do_grav_set	L	F	Should gravitational settling of aerosols be done?
do_var_adv_tstp	L	F	Should variable advection time steps be taken as determined by the Courant condition?
ACTM_CONVEC SECTION (convection)			
Base convection units = kg/m²*s			
convec_opt	I	0	Convection option: 0: no convection 1: do DAO2 convection 2: do NCAR convection
ACTM_DEPOS SECTION (deposition)			
Base deposition units = m/s			
do_drydep	L	F	Should dry deposition be done?
do_wetdep	L	F	Should wet deposition be done?
do_simpledep	L	F	Should simple deposition be done?
num_ks_sdep	I	1	Number of vertical layers to apply 2 day loss factor to in simple deposition.
wetdep_eff()	R	0.0	Wet deposition (scavenging) efficiencies; should be set to values between 0.0 and 1.0 for each species.
ACTM_DIFFU SECTION (diffusion)			
diffu_opt	I	0	Diffusion option: 0: no diffusion 1: do DAO2 vertical diffusion
vert_diffu_coef	R	0.0	Scalar vertical diffusion coefficient (m ² /s).
pbl_mixing_tau	R	3600.0	Sets PBL mixing time which is needed for GISS runs. At present we use 18000.0. It influences the range of diffu_opt

			which is now 0-2.
ACTM_EMISS SECTION (emissions)			
Base emissions units = kg/s			
emiss_opt	I	0	Emissions option: 0: no emissions 1: do LLNL emissions only 2: do LLNL + Harvard emissions
emiss_in_opt	I	0	Emissions input option: 0: no emissions data 1: set all emiss values to emiss_init_val 2: read in emiss values
emiss_conv_flag	I	0	Emissions conversion flag: 0: no conversion performed 1: use scalar emiss_conv_fac (scalar * kg/s => kg/s) 2: use calculated emissions conversion factor (kg/km ² *hr => kg/s)
semis_inchem_flag	I	-1	Surface emissions inside chemistry flag: < 0: If emissions are on, surface emissions will be done in Smvgear chemistry if it is on; outside of chemistry if Smvgear chemistry is off. = 0: If emissions are on, surface emissions will be done outside of chemistry. > 0: If emissions are on, surface emissions will be done in Smvgear chemistry.
emiss_timpyr	I	1	Emission times per year: 1: one set of emissions per year (yearly) 12: twelve sets of emissions per year (monthly)
emiss_map()	I	0	Mapping of emission number to const species number; note that it is permissible to map a species in the emissions file to 0, and then the code will not read in or use that chunk of data. Since the default is 0 for all species, it is not necessary to specify the species you want to skip, just the ones you want to use.
emiss_conv_fac	R	1.0	Emission conversion factor when emiss_conv_flag = 1.
emiss_init_val	R	1.0	When emiss_opt = 1, this value will be used to initialize all emissions values.
emiss_infile_name	C*128	' '	Emissions input file name.
emiss_var_name	C*32	'emiss'	NetCDF emission variable name.
Harvard biogenic & soil emissions:			
isop_scale()	R	1.0d0	Isoprene scaling factors for each month.
Note that if ((emiss_opt == 2) && do_full_chem), the indices below will be automatically set by the setkin files.			
iacetone_num	I	0	Const array index for acetone (C ₃ H ₆ O) (ACET).
ico_num	I	0	Const array index for CO.
iisoprene_num	I	0	Const array index for isoprene (C ₅ H ₈) (ISOP).
ipropene_num	I	0	Const array index for propene (C ₃ H ₆) (PRPE).

Chapter E. Input Namelist Variables

ino_num	I	0	Const array index for NO.
fertscale_infile_name	C*128	'fertscale_4x5_dao.asc'	Fertilizer scale infile name.
lai_infile_name	C*32	'lai_4x5_dao.asc'	Leaf area index infile name.
light_infile_name	C*128	'lighttable.asc'	Light infile name.
precip_infile_name	C*128	'precip_4x5_dao.asc'	Precipitation infile name.
soil_infile_name	C*128	'soiltype.asc'	Soil type infile name.
veg_infile_name	C*128	'vegtype_4x5_dao.asc'	Vegetation type infile name.
isopconv_infile_name	C*128	'isopconvtable.asc'	Isoprene conversion infile name.
monotconv_infile_name	C*128	'monotconvtable.asc'	Monoterpene conversion infile name.
Michigan aerosol and dust emissions:			
emiss_aero_opt	I	0	0, 1; 0 for no michigan aerosol emissions.
naero	I	0	number of aerosol emissions.
emiss_map_aero()	I	0	map aerosol emissions to species #.
emiss_aero_infile_name	C*128	' '	Name of file containing michigan aerosol emissions.
emiss_dust_opt	I	0	0,1; 0 fo no michigan dust emissions.
ndust	I	0	number of dust emissions.
nst_dust	I	1	number of starting point in time for michigan dust emissions.
nt_dust	I	1	number of times of dust emissions per michigan dust emissions file.
emiss_map_dust()	I	0	map dust emissions to species #.
emiss_dust_infile_name	C*128	' '	Name of file containing michigan dust emissions.
ACTM_CHEM SECTION (chemistry)			
chem_opt	I	0	Chemistry option: 0: no chemistry (age of air, etc.) 1: call Radon/Lead chemistry 2: call SmvgearII 3: call simple loss (N2O, etc.) 4: call forcing boundary condition for a tracer (CO2, etc.) 5: call Synoz tracer (if num_species=1 then just Synoz, if num_species=2 then Nodoz tracer is species number 2) 6: call Beryllium chemistry 7: call Quadchem 8: call Sulfur chemistry
chem_cycle	R	1.0	Number of time steps to cycle chemistry calls on: < 1.0: chemistry will subcycle = 1.0: chemistry called each time step
chem_mask_klo	I	k1_gl	Lowest grid level at which chemistry is calculated.
chem_mask_khi	I	k2_gl	Highest grid level at which chemistry is calculated.
synoz_threshold	R	Huge	Chemistry turned off where synoz > this threshold (mixing ratio).
t_cloud_ice	R	263.0	Temperature for cloud ice formation.
do_chem_grp	L	F	Should chemical groups be used?
do_smv_reord	L	F	Should the grid boxes be reordered in order of stiffness?

do_wetchem	L	F	Should wet chemistry be done?
Be-7/Be-10 chemistry:			
be_opt	I	1	Beryllium star table option: 1: use Koch table for Be-7 and Be-10 2: use Nagai tables for Be-7 and Be-10
t_half_be7	R	53.3d0	Half life of Beryllium-7, or other cosmogenic radionuclide (days).
t_half_be10	R	5.84d8	Half life of Beryllium-10, or other cosmogenic radionuclide (days).
yield_be7	R	4.5d-7	Yield factor for Beryllium-7, or other cosmogenic radionuclide (unitless).
yield_be10	R	2.5d-7	Yield factor for Beryllium-10, or other cosmogenic radionuclide (unitless).
Base forcing boundary condition units = mixing ratio			
forc_bc_opt	I	1	Forcing boundary condition option: 1: set all forc_bc values to forc_bc_init_val 2: read in forc_bc 3: calculate forc_bc
forc_j1	I	ju1_gl	Forcing boundary condition j1 (low latitude).
forc_j2	I	j2_gl	Forcing boundary condition j2 (high latitude).
forc_bc_years	I	1	Number of years of forcing data.
forc_bc_start_num	I	1	Forcing boundary condition start number; index for year to use.
forc_bc_kmin	I	1	Minimum k level for forcing boundary condition.
forc_bc_kmax	I	1	Maximum k level for forcing boundary condition.
forc_bc_map()	I	0	Mapping of forcing boundary condition number to const species number.
forc_bc_init_val	R	0.0	When forc_bc_opt = 1, this value will be used to initialize all forc_bc values (ppmv).
forc_bc_incrpyr	R	0.3	Forcing boundary condition emission increase per year.
forc_bc_lz_val	R	0.0	Value to which lower zones are forced.
forc_bc_infile_name	C*128	'forc_bc_co2.asc'	Forcing boundary condition input file name.
Base simple loss units = s⁻¹			
loss_freq_opt	I	1	Loss frequency option: 1: set all loss_freq values to loss_init_val 2: read in loss data 3: use NCAR loss
kmin_loss	I	k1_gl	Minimum vertical index at which loss will occur; currently, below this altitude a constant boundary condition is enforced using const_init_val for all species.
kmax_loss	I	k2_gl	Maximum vertical index at which loss will occur.
loss_init_val	R	0.0	When loss_freq_opt = 1, this value will be used to initialize all loss_freq values.
loss_data_infile_name	C*128	'loss_n2o.asc'	Loss data input file name.
Surface Area Density (SAD):			
sad_opt	I	0	Surface area density (SAD) option: 0: do not allocate or process SAD array

Chapter E. Input Namelist Variables

			1: allocate, but zero out SAD array 2: call Considine code (i.e., Condense) 3: read SAD array from a file of monthly averages
sad_var_name	C*32	'sad'	NetCDF sad variable name.
sad_dim_name	C*32	'sad_dim'	NetCDF sad dimension name.
h2oclim_opt	I	2	Water climatology input option: 1: set all h2oclim values to h2oclim_init_val 2: read in h2oclim
h2oclim_timpyr	I	12	Water climatology times per year 1: yearly 12: monthly
ch4clim_init_val	R	0.0	When h2oclim_opt = 1, this value will be used to initialize all ch4clim
h2oclim_init_val	R	0.0	When h2oclim_opt = 1, this value will be used to initialize all h2oclim values.
h2oclim_infile_name	C*128	' '	Water climatology input file name.
lbssad_opt	I	2	Liquid binary sulfate input option: 1: set all lbssad values to lbssad_init_val 2: read in lbssad
lbssad_timpyr	I	12	Liquid binary sulfate times per year: 1: yearly 12: monthly
lbssad_init_val	R	0.0	When lbssad_opt = 1, this value will be used to initialize all lbssad values.
lbssad_infile_name	C*128	' '	Liquid binary sulfate input file name.
qk / qqk:			
qk_var_name	C*32	'qk'	NetCDF qk variable name.
qqk_var_name	C*32	'qqk'	NetCDF qqk variable name.
qk_dim_name	C*32	'qk_dim'	NetCDF qk dimension name.
qqk_dim_name	C*32	'qqk_dim'	NetCDF qqk dimension name.
Reaction rate adjustment:			
do_rxnr_adjust	L	F	Adjust reaction rates?
rxnr_adjust_infile_name	C*128	' '	Reaction rate adjustment input file name.
rxnr_adjust_var_name	C*32	'reac_rate_adj'	NetCDF reaction rate adjustment variable name.
ACTM_PHOT SECTION (photolysis)			
Base photolysis/qj units = s⁻¹			
phot_opt	I	1	Photolysis option: 0: no photolysis 1: set all qj values to qj_init_val 2: read in qj values 3: use fastj routine (troposphere only for now) 4: lookup table for qj (Kawa style) 5: lookup table for qj (Kawa style) + use ozone climatology for column ozone calc. 6: calculate from table and Gmimod data (Quadchem) 7: read in qj values (2-D, 12 months) 8: use fastjx
do_clear_sky	L	T	Should clear sky photolysis be done?

fastj_offset_sec	R	0.0d0	Offset from model time at which to do fastj (s).
qj_init_val	R	1.0d-30	When phot_opt = 1, this value will be used to initialize all qj values.
qj_infile_name	C*128	' '	qj input file name.
qj_var_name	C*32	'qj'	NetCDF qj variable name.
qqj_var_name	C*32	'qqj'	NetCDF qqj variable name.
qj_dim_name	C*32	'qj_dim'	NetCDF qj dimension name.
qqj_dim_name	C*32	'qqj_dim'	NetCDF qqj dimension name.
Surface albedo:			
sfalbedo_opt	I	0	Surface albedo option: 0: no sfalbedo 1: set each type of sfalbedo to an initial value 2: read in monthly sfalbedo values from a NetCDF file 3: read in values of four types of surface albedo from the met data
saldif_init_val	R	0.1	Surface albedo for diffuse light (near IR); when sfalbedo_opt = 1, this value will be used to initialize all saldif values.
saldir_init_val	R	0.1	Surface albedo for direct light (near IR); when sfalbedo_opt = 1, this value will be used to initialize all saldir values.
sasdif_init_val	R	0.1	Surface albedo for diffuse light (uv/vis); when sfalbedo_opt = 1, this value will be used to initialize all sasdif values.
sasdir_init_val	R	0.1	Surface albedo for direct light (uv/vis); when sfalbedo_opt = 1, this value will be used to initialize all sasdir values.
sfalbedo_infile_name	C*128	' '	Surface albedo input file name.
UV albedo:			
uvalbedo_opt	I	0	UV albedo option: 0: no uvalbedo 1: set all uvalbedo values to uvalbedo_init_val 2: read in monthly uvalbedo values from an ASCII file 3: read in bulk surface albedo values from the met data
uvalbedo_init_val	R	0.1	When uvalbedo_opt = 1, this value will be used to initialize all uvalbedo values.
uvalbedo_infile_name	C*128	' '	Uvalbedo input file name.
cross_section_file	C*128	' '	X-Section quantum yield
rate_file	C*128	' '	Master rate file
T_O3_climatology_file	C*128	' '	T & O3 climatology
ACTM_TRAC SECTION (tracer)			
tracer_opt	I	0	range 0-1
efold_time	R	0.0	should be adjusted for different tracers in tracer runs

Table E.1: Namelist variables

