An Overview of the Flexible Model System (FMS) and How to Contribute

Tom Robinson

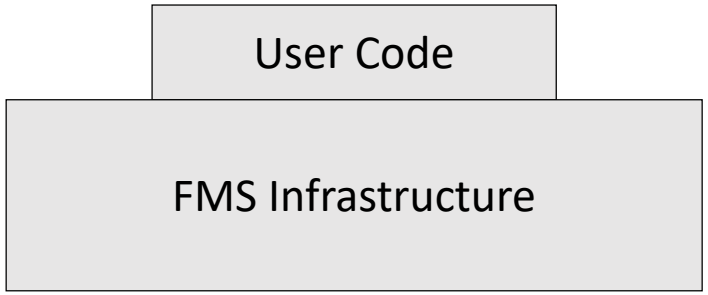
thomas.robinson@noaa.gov

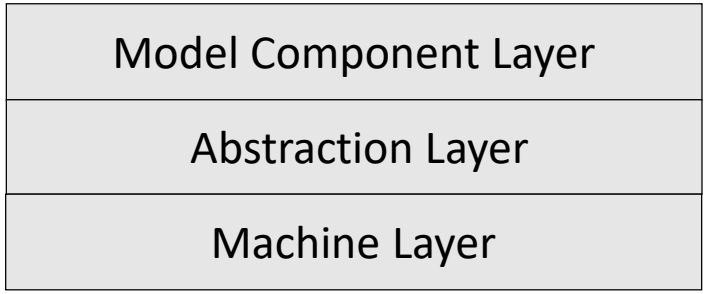
Outline

- FMS background
- Code management
- How to contribute
- Notable modules

Flexible Model System

- Began in the prehistoric time of 1998
- Supports multiple dynamical cores, components (ocean, ice, land, etc) and atmospheric physics packages
  - Includes null and data models
- Infrastructure comprised of common utilities needed by model components
- Mostly written in Fortran





FMS Features

- Provides MPI communication and halo updates
- Input and output handling
- Error handling
- Supports multiple grids and exchanges between them
  - Tripolar, unstructured, cubed-sphere
- Manages diagnostic variables
- Ability to override data
- Centralized tracer management

Code Management

- FMS code is hosted on GitHub at https://GitHub.com/NOAA-GFDL/FMS
- Code releases are named by the year and number of release
  - 2019.01 in December of 2019, 2020.01 in February of 2020, etc
  - Patches can be released: 2019.01.03 supports updates to 2019.01 for slurm, diag_manager interfaces, and nesting
- The code is managed by the Modeling Systems Division at GFDL
  - FMS group - code development and updates as well as model testing
  - Each team member is an “expert” in some subset of the code (Code owners)
  - Subset of group to handle management of pull requests
  - Colin Gladue is the code manager responsible for performing PRs
- Any inquiries or issues should be addressed on the issues tab in the GitHub repo
- The “main” branch is the latest code available, but it not guaranteed to work as it is a development stream

Contributing

- The guidelines for contributing are available in the GitHub repository
  - A copy of the Gnu Lesser General Public License must be included at the top of each file.
  - Comments should be in doxygen style
  - No common blocks, goto statements, or variables with Fortran key words
  - No trailing whitespace or tabs
  - Efforts are being made to clean up areas where the style guide is violated by existing code
- To make code updates
  - Fork the FMS repository
  - Create a branch in your fork
  - Push updates to your fork/branch
  - Submit a pull request (PR) to the NOAA-GFDL/FMS repository
- You must add an issue to link to your PR the describes the issue
- Pull requests should be made to the “main” branch of FMS

Contributing (cont.)

- You must fill in the templates for the issue and pull request
- All check boxes must be checked
  - Even if there are no new checks, you must still click the “New check tests, if applicable, are included” box
- All pull requests must pass the checks of the CI
- Your updates will be reviewed by at least one person on the MSD team
- Updates that are not code related (ie. cmake) can be merged in right away to the “main” branch
- Code updates will be tested in the GFDL supported models and released when all tests pass and release goals are met

MPP

- ”Machine layer”
- Communication Layer - *provides fundamental point-to-point and asynchronous, non-blocking communication*
- Domains Layer – *handles decompositions and tracks which ranks are “connected” for halo updates, I/O, and nesting*
- I/O Layer - *handles all file types including direct interface to NetCDF/HDF*
- Clocks - *allow high-resolution timing of code segments*
- Miscellaneous - *pelists, error handling, reductions, unit tests (functionality and performance), etc*

PE Lists

- Describes how MPI ranks are connected for updating
- Stored as an array
- Set up communicators for different groups (atm_pes, ocean_pes, etc)
- The “root_pe” is the lowest rank in the pe list
- An MPI rank can be in multiple PE lists.
  - A root_pe can be the root of multiple lists

Communication

- Rich set of higher-level routines to abstract MPI from the users
  - Model components using FMS don’t have to call MPI routines/functions
- Uses isend and irecv asynchronous communication routines
- PE lists are used to handle the communication
- Collectives also exist
- Support all types (real, complex, integer, character)

Domains

- Supports decomposition of grids amongst PEs
  - tiled grids
  - unstructured grids
  - rectilinear grids
  - tri-polar grids
- Creates an internal data structure containing:
  - halo update communication mapping
  - directional data orientation and rotations
  - tile information
- Domain data structure is unique to a halo width
- Manages nested grid <-> coarse grid boundary updates
  - Telescoping and multiple nests available
- Uses message aggregation to reduce communication costs

Example

- C96 run (96x96) with layout of 4,4
  - The global domain is 96x96
  - The compute domain is 16x16
  - The data domain is 17x17 (halo)
  - PE list (0,1,…,14,15)

4 x 4 decomposition

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Other mpp features

- Clocks
  - Different granularies (routine, module, subcomponent)
  - End of run summary (total and per rank)
  - clock_sync to isolate load imbalance
- Errors/notes
  - FATAL, WARNING, NOTE
  - Calls MPI_ABORT on a fatal
- Sums
  - Checksums for files
    - Debugging, IO corruption, etc
  - Fast sums using allreduce (does not bitwise reproduce)
  - Bitwise exact summing

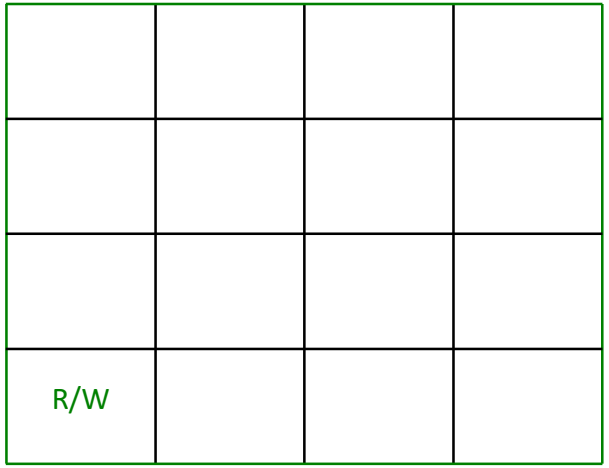
FMS2_IO

- Netcdf IO is handled by the new FMS2_IO module
- Replaces mpp_io
- Handles reading and writing
  - Special routines for restart files
- Modeled after the netcdf
- Set up to work with mpp_domains
  - Open file
  - Register axis
  - Register variable
  - Add metadata
  - Write output
  - Reads restarts after registering variables
- IO domains are defined to reduce the number of cores doing the actual IO

4 x 4 decomposition

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

1 x 1 io_subset



0 is pe_root and io_root

4 x 4 decomposition

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Root_pe is 0

1 x 2 io_subset

Two io_pe lists (0:7) and (8:15)

R/W			
R/W			

0 and 8 and io_root_pe for respective lists

Managers/Overrides modules

- Time manager
  - Supports Julian, Gregorian, 30-day months, no-leap, no-calendar
- Block manager
  - Stores the extent of each MPI rank to be used with openmp
  - Support for horizontal and collapsed indexing
- Field/tracer manager
  - Field descriptions given in an ascii file *field_table*
  - Tracers added by code and matched by name to description in ascii file
- Diagnostic manager
  - Build on top of parallel IO system
  - Supports scalar-3D fields and multiple types of averaging and time intervals
  - Input read from ascii *diag_table*
- Data override
  - Uses information in the ascii file *data_table*
  - Performs spatial and temporal interpolation to replace a prognostic field
  - Works at any point in the simulation

Summary

- FMS has a long history of serving as the infrastructure for many models
- FMS is used so scientists can focus on science and not have to worry about the parallelism and commpunication of the code
- Contributions to FMS can be made on the FMS GitHub page