

FMS: the GFDL Flexible Modeling System

**Brian Gross
NOAA/GFDL**

Princeton NJ November 2005

GFDL Strategic Objectives

GFDL is a NOAA climate modeling centre. The primary focus is the use and development of coupled climate models for simulations of climate variability and climate change on short (seasonal-interannual) and long (decadal-centennial) time scales.

- Provide timely and reliable knowledge for the nation on natural climate variability and anthropogenic change.
- Develop Earth Systems Models (ESMs) for climate variability and change.
- Advance expert assessment of global and regional climate change through research, improved model and data products.

GFDL Computing

- Reliance on Cray vector architecture in previous decades.
- Transition to scalable computing begun in 1997 with the acquisition of Cray T3E.
- Current computing capability: 128 + 256 + 2 × 96p Origin 3900; 128 + 4 × 256 + 3 × 512p SGI Altix.

Technological trends

In climate research... increased emphasis on detailed representation of individual physical processes governing the climate; requires many teams of specialists to be able to contribute components to an overall coupled system;

In computing technology... increase in hardware and software complexity in high-performance computing, as we shift toward the use of scalable computing architectures.

Technological trends

In software design for broad communities... The open source community provided a viable approach to the construction of software to meet diverse requirements through “open standards”. The standards evolve through consultation and prototyping across the user community.

“Rough consensus and working code.” [IETF]

The GFDL response: modernization of modeling software

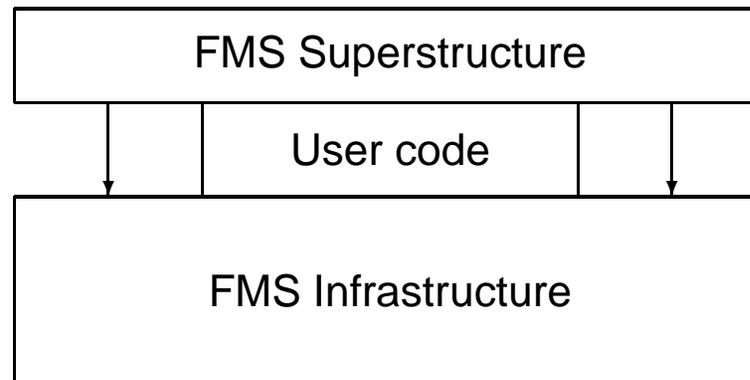
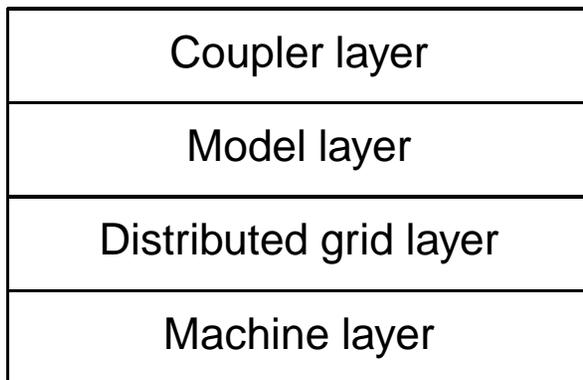
- Abstraction of underlying hardware to provide *uniform programming model* across vector, uniprocessor and scalable architectures;
- Distributed development model: many contributing authors. Use high-level abstract language features to facilitate development process;
- Modular design for interchangeable dynamical cores and physical parameterizations, development of *community-wide standards* for components.

FMS: the GFDL Flexible Modeling System

Jeff Anderson, V. Balaji, Will Cooke, Jeff Durachta, Matt Harrison, Isaac Held, Paul Kushner, Amy Langenhorst, Zhi Liang, Sergey Malyshev, Giang Nong, Ron Pacanowski, Pete Phillipps, Lori Thompson, Mike Winton, Bruce Wyman, ...

- Develop high-performance kernels for the numerical algorithms underlying non-linear flow and physical processes in complex fluids;
- Maintain high-level code structure needed to harness component models and representations of climate subsystems developed by independent groups of researchers;
- Establish standards, and provide a shared software infrastructure implementing those standards, for the construction of climate models and model components portable across a variety of scalable architectures.
- Benchmarked on a wide variety of high-end computing systems;
- Run in production on very different architectures: parallel vector (PVP), distributed massively-parallel (MPP) and distributed shared-memory (NUMA).

Architecture of FMS



FMS shared infrastructure: machine and grid layers

MPP modules communication kernels, domain decomposition and update, parallel I/O.

Time and calendar manager tracking of model time, scheduling of events based on model time.

Diagnostics manager Runtime output of model fields.

Data override Runtime input of model fields.

Scientific libraries Uniform interface to proprietary and open scientific library routines.

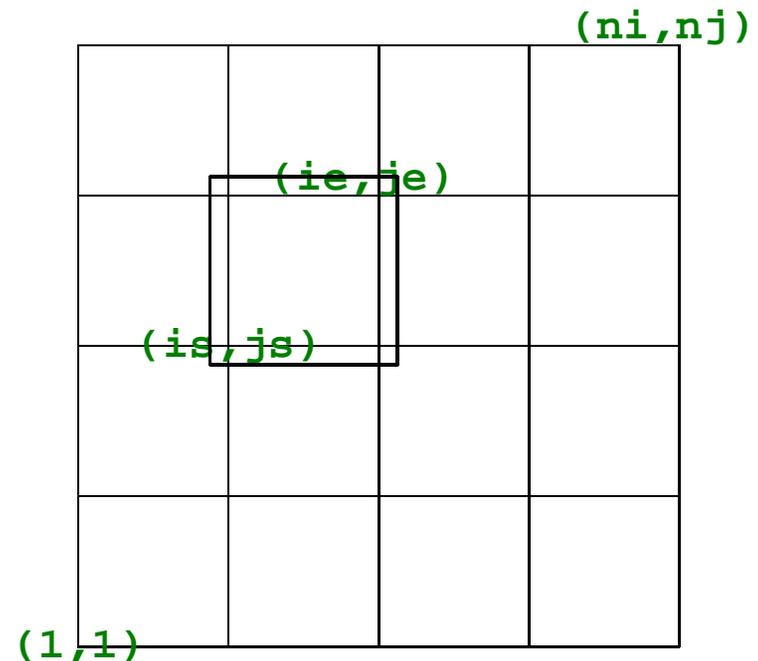
Communication kernels

provide uniform interface to:

- MPI message-passing across clusters.
- MPI or SHMEM on tightly-coupled distributed memory (T3E).
- Pointer-sharing and direct copy on shared-memory and distributed-shared memory (NUMA).

User interface to communication kernels

- `mpp_define_domains()`
- `mpp_update_domains()`



```
type(domain2D) :: domain
call mpp_define_domains( (/1,ni,1,nj/), domain, xhalo=2, yhalo=2 )
...
call mpp_update_domains( f, domain )
!perform computations on f
```

Parallel I/O interface

`mpp_io_mod` is a set of simple calls to simplify I/O from a parallel processing environment. It uses the domain decomposition and communication interfaces of `mpp_mod` and `mpp_domains_mod`. It is designed to deliver high-performance I/O from distributed data, in the form of self-describing files (verbose metadata).

`mpp_io_mod` supports three types of parallel I/O:

- Single-threaded I/O: a single PE acquires all the data and writes it out.
- Multi-threaded, single-fileset I/O: many PEs write to a single file.
- Multi-threaded, multi-fileset I/O: many PEs write to independent files (requires post-processing).

Features of `mpp_io_mod`

- Simple, minimal API, with freedom of access to native APIs.
- Strong focus on performance of parallel write.
- Accepts netCDF format, widely used in the climate/weather community. Extensible to other formats.
- Post-processing tool for distributed netCDF files provided with FMS.
- Compact dataset (comprehensively self-describing).
- Final dataset bears no trace of parallelism.

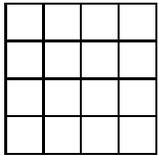
mpp_io_mod API

- `mpp_io_init()`
- `mpp_open()`
- `mpp_close()`
- `mpp_read()`
- `mpp_read_meta()`
- `mpp_write()`
- `mpp_write_meta()`

User interface to parallel I/O

```
type(domain2D) :: domain
type(axistype) :: x, y, z, t
type(fieldtype) :: field
integer :: unit
character*(*) :: file
real, allocatable :: f(:, :, :)
call mpp_define_domains( (/1,ni,1,nj/), domain )
call mpp_open( unit, file, action=MPP_WRONLY, format=MPP_IEEE32, &
  access=MPP_SEQUENTIAL, threading=MPP_SINGLE )
call mpp_write_meta( unit, x, 'X', 'km', ... )
...
call mpp_write_meta( unit, field, (/x,y,z,t/), 'Temperature', 'kelvin', ...
...
call mpp_write( unit, field, domain, f, tstamp )
```

MPP developments: Mosaics



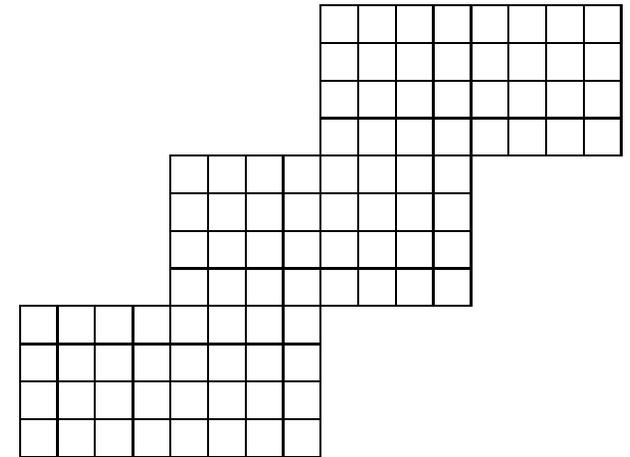
On the left is a basic 4×4 **tile**; on the right are examples of grids composed of a mosaic of such tiles. The first is a **continuous grid**, below is a **refined grid**.

Most current software only supports what we call **tiles** here. The **mosaic** extension will allow the development of more complex grids for next-generation models.

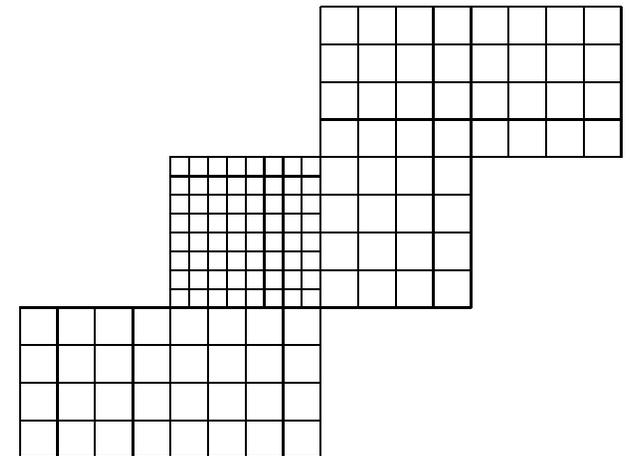
First in our (GFDL's) sights is the **cubic sphere**, primarily targeted at a next-generation finite-volume atmospheric dynamical core, but potentially others as well.

Further developments will include support for irregular tiling (e.g. of the ocean surface following coastlines), and for refined, nested and adaptive grids.

Also, regular grids where an irregular decomposition is needed (e.g. for a polar filter) can use mosaics to define different decompositions in different regions.



Regular grid mosaic.



Refined grid mosaic.

Coupler

Used for data exchange between models. Key features include:

Conservation: required for long runs.

Resolution: no constraints on component model timesteps and spatial grid. Supports both explicit and implicit timestepping.

Exchange grid: union of component model grids, where detailed flux computations are performed (Monin-Obukhov, tridiagonal solver for implicit diffusion, ...)

Fully parallel: Calls are entirely processor-local: exchange software will perform all inter-processor communication.

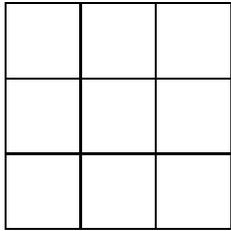
Modular design: uniform interface to main calling program.

No brokering: each experiment must explicitly set up field pairs.

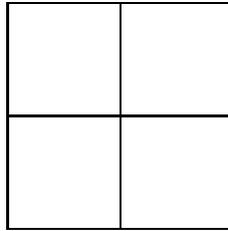
Single executable:

Highly efficient: currently able to couple atmos/ocean at 3h intervals, atmos/land/ice implicitly at each timestep at current dec/cen resolutions.

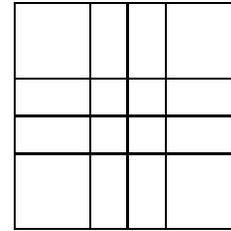
Exchange grid



Atmosphere



Ocean



Exchange

- Each cell on exchange grid “belongs” to one cell on each parent grid;
- Conservative interpolation up to second order;
- All calls exchange local data; data-sharing among processors is internal to the exchange software, and non-blocking.
- Physically identical grids (e.g ocean and sea ice) exchange data without interpolation.

Features of the FMS coupler

- Encapsulated boundary state and boundary fluxes.
- Single location for initialization and linking of boundary fields.
- Use of field manager to organize operations on individual fields and field bundles.
- Support for serial and concurrent coupling within single executable.
- Implicit coupling between land-ocean surface and atmosphere on atmospheric timestep; explicit coupling between ocean surface and ocean on coupling timestep.

coupler_main slow loop

```
do nc = 1, num_cpld_calls
  call generate_sfc_xgrid( Land, Ice )
  call flux_ocean_to_ice( Ocean, Ice, Ocean_ice_flux )
  call update_ice_model_slow_up( Ocean_ice_flux, Ice )
!fast loop
  call update_land_model_slow(Land)
  call flux_land_to_ice( Land, Ice, Land_ice_flux )
  call update_ice_model_slow_dn( Atmos_ice_flux, Land_ice_flux, Ice )
  call flux_ice_to_ocean( Ice, Ice_ocean_flux )
  call update_ocean_model( Ice_ocean_flux, Ocean )
enddo
```

coupler_main fast loop

```
do na = 1, num_atmos_calls
  Time = Time + Time_step_atmos
  call sfc_boundary_layer( Atm, Land, Ice, &
                          Land_ice_atmos_flux )
  call update_atmos_model_down( Land_ice_atmos_flux, Atm )
  call flux_down_from_atmos( Time, Atm, Land, Ice, &
                             Land_ice_atmos_flux, &
                             Atmos_land_flux, Atmos_ice_flux )
  call update_land_model_fast( Atmos_land_flux, Land )
  call update_ice_model_fast( Atmos_ice_flux, Ice )
  call flux_up_to_atmos( Time, Land, Ice, Land_ice_atmos_flux )
  call update_atmos_model_up( Land_ice_atmos_flux, Atm )
enddo
```

Example: ocean boundary

```
type ocean_boundary_data_type
  type(domain2D) :: Domain
  real, pointer, dimension(:, :) :: t_surf, s_surf, sea_lev, &
    frazil, u_surf, v_surf
  logical, pointer, dimension(:, :) :: mask
  type (time_type) :: Time, Time_step
end type ocean_boundary_data_type
```

```
type, public :: ice_ocean_boundary_type
  real, dimension(:, :), pointer :: u_flux, v_flux, t_flux, q_flux
  real, dimension(:, :), pointer :: salt_flux, lw_flux, sw_flux, lprec, fpr
  real, dimension(:, :), pointer :: runoff, calving
  real, dimension(:, :), pointer :: p
  real, dimension(:, :, :), pointer :: data
  integer :: xtype !REGRID, REDIST or DIRECT
end type ice_ocean_boundary_type
```

Flux exchange

Three types of flux exchange are permitted: **REGRID**, **REDIST** and **DIRECT**.

REGRID physically distinct grids, requires exchange grid.

REDIST identical global grid, different domain decomposition.

DIRECT identical grid and decomposition.

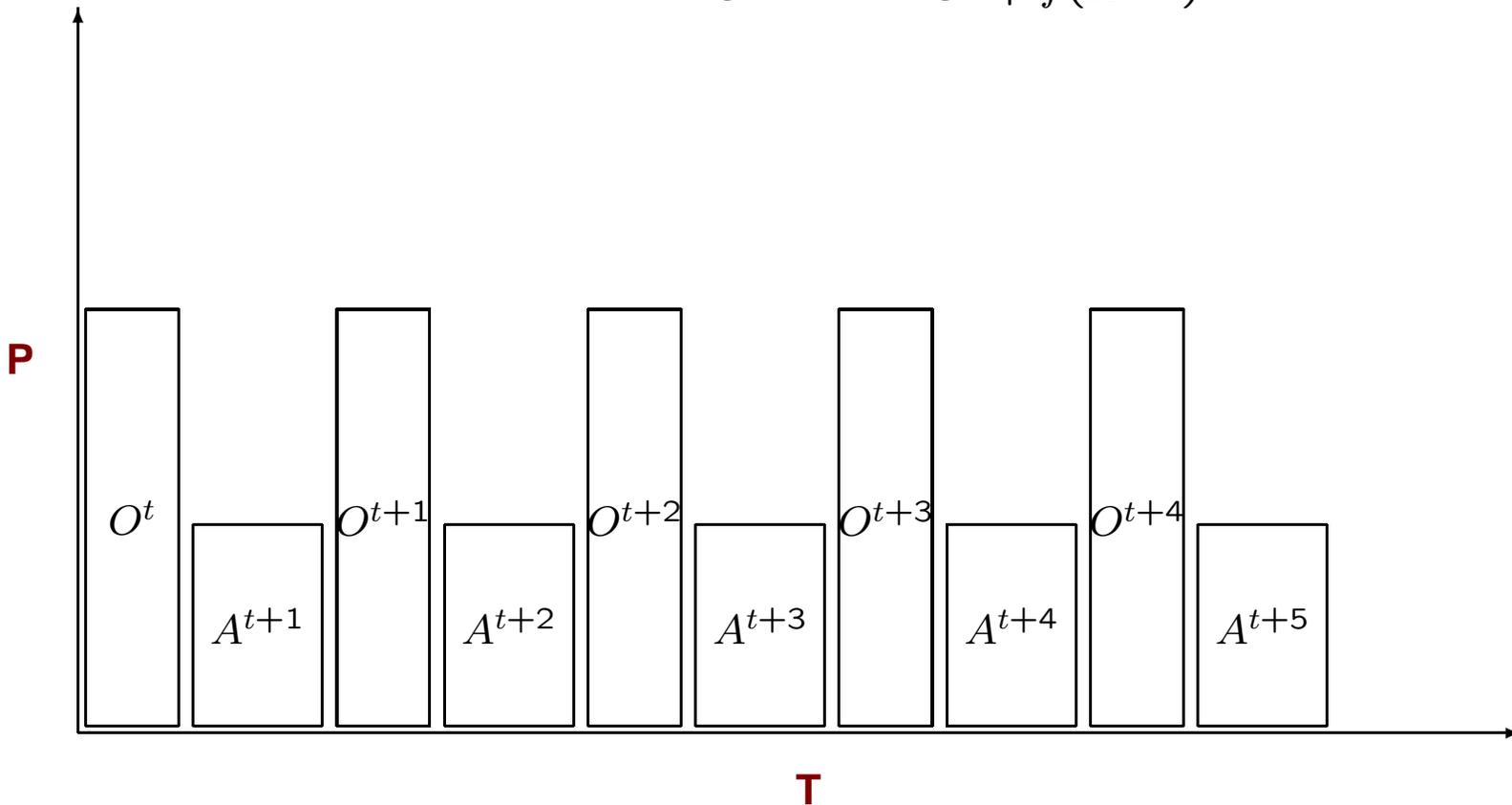
Current use: **REGRID** between atmos \longleftrightarrow ice, atmos \longleftrightarrow land, land \longleftrightarrow ice, **REDIST** between ocean \longleftrightarrow ice.

Serial coupling

Uses a forward-backward timestep for coupling.

$$A^{t+1} = A^t + f(O^t) \quad (1)$$

$$O^{t+1} = O^t + f(A^{t+1}) \quad (2)$$

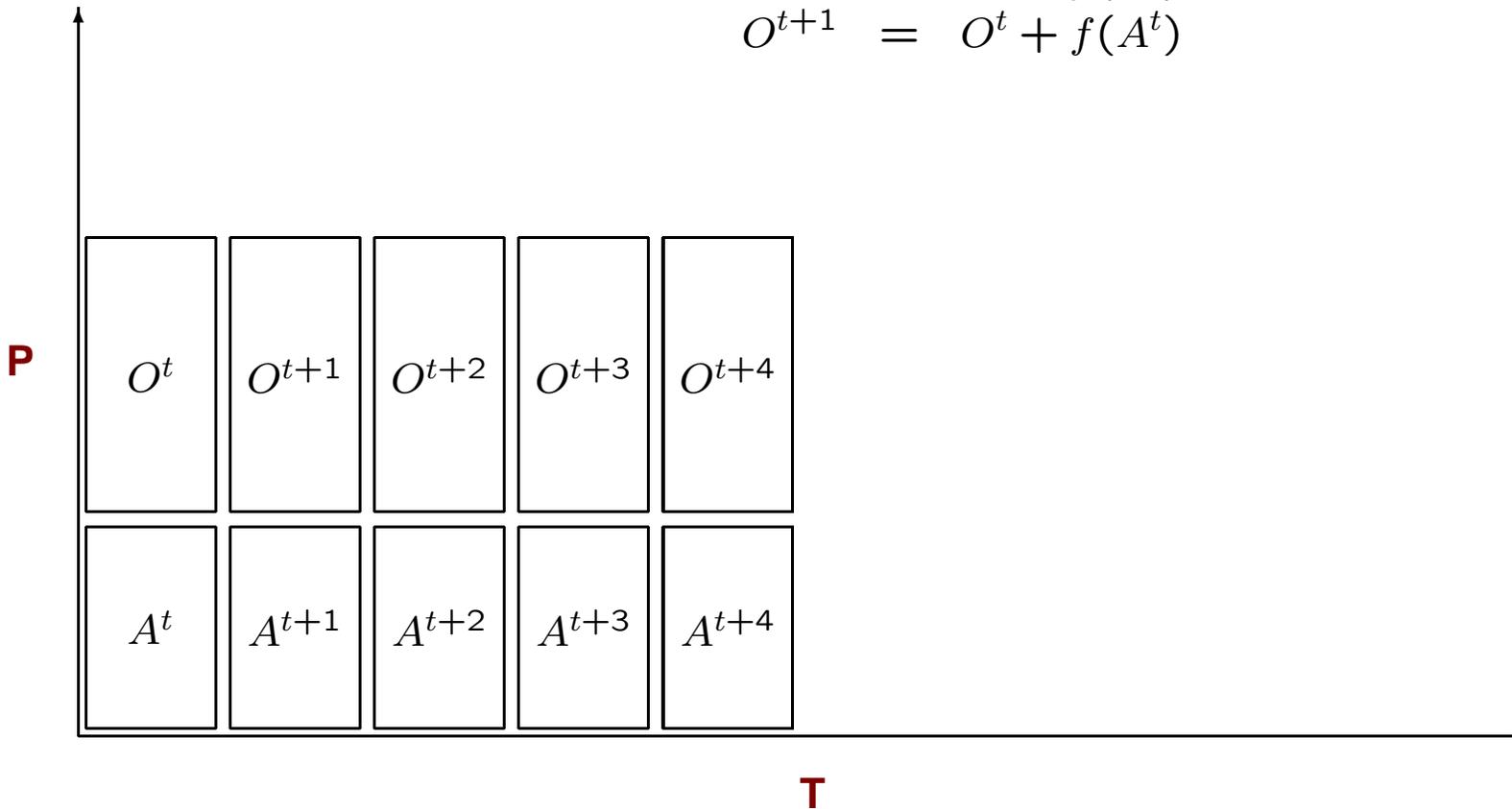


Concurrent coupling

This uses a forward-only timestep for coupling. While formally this is unconditionally unstable, the system is strongly damped. Answers vary with respect to serial coupling, as the ocean is now forced by atmospheric state from Δt ago.

$$A^{t+1} = A^t + f(O^t) \quad (3)$$

$$O^{t+1} = O^t + f(A^t) \quad (4)$$



In terms of model code...

```
do nc = 1, num_cpld_calls
  call generate_sfc_xgrid()
  call flux_ocean_to_ice()
  if( concurrent_coupling )call flux_ice_to_ocean()
  if( atmos_pe )then
    call update_ice_model_slow_up()
    call update_atmos... !fast loop
    call update_land_model_slow()
    call flux_land_to_ice()
    call update_ice_model_slow_dn()
  endif
  if( serial_coupling )call flux_ice_to_ocean()
  if( ocean_pe )call update_ocean_model()
enddo
```

- The pelists are set up in the coupler layer, and subsequently all the **mpp** calls automatically operate within their pelists, with no changes to the model code.
- Within the atmos pelist, we can further declare land and ice as concurrent if needed. Not currently implemented, since $T_{ice} \gg T_{land}$.

FMS component models

- Atmosphere:
 - BGRID: hydrostatic finite difference model on a staggered Arakawa B grid and hybrid σ/P vertical coordinate (Wyman);
 - SPECTRAL: hydrostatic spectral transform model also with the hybrid σ/P vertical coordinate (Held, Phillipps);
 - FV: hydrostatic primitive equations using finite-volume dynamics (S.J Lin);
 - ZETAC: non-hydrostatic C-grid regional/global model on terrain-following coordinates (Garner, Orlanski, Kerr);
 - Spectral shallow water, 2D energy balance, data model, etc.
- Land:
 - Land Dynamics model (LaD) 5 temperature layers, 11 soil/vegetation types, stomatal resistance, bucket hydrology, river routing (Milly);
 - SHE: soil, hydrology and ecology model, interactive biospheric carbon, ecosystem dynamics with dynamic tiling (Malyshev, Shevliakova);

FMS component models

- Ocean:
 - MOM: primitive equation ocean climate model with generalized horizontal coordinates and vertical z -coordinate, full suite of physics options, compatible with state-of-art adjoint compiler (Pacanowski, Griffies, Rosati, Harrison);
 - HIM: isopycnal coordinate ocean model (Hallberg, Cooke);
- Ice: Sea Ice Simulator (SIS) full sea ice dynamics with elastic-plastic-viscous rheology, N-category ice thickness, 3-layer vertical thermodynamics (Winton);

Fitting into FMS

To incorporate your own ocean model (say) into FMS, you have to provide a few key routines (`ocean_model_init`, `update_ocean_model`) and encapsulate your ocean boundary state into `ocean_boundary_type`.

It helps to use the FMS infrastructure but not essential.

CM2.1 was created from CM2 by importing the new FV core into FMS, and then switching atmospheric cores in the coupled system.

The FMS user interface: FRE

Comprehensive website for all information and documentation:

<http://www.gfdl.noaa.gov/~fms>

- Source code maintenance under CVS; browse over the net using webCVS.
- Model configuration, launching and regression testing encapsulated in XML;
- Relational database for archived model results;
- Standard and custom diagnostic suites;

The FMS Runtime Environment (FRE) describes all the steps for configuring and running a model jobstream; archiving, postprocessing and analysis of model results.

`fremake, frerun, frepp, frecheck, ...`

The Regression Test Suite (RTS) is a set of tests that are run continuously on a set of FMS models to maintain and verify code integrity.

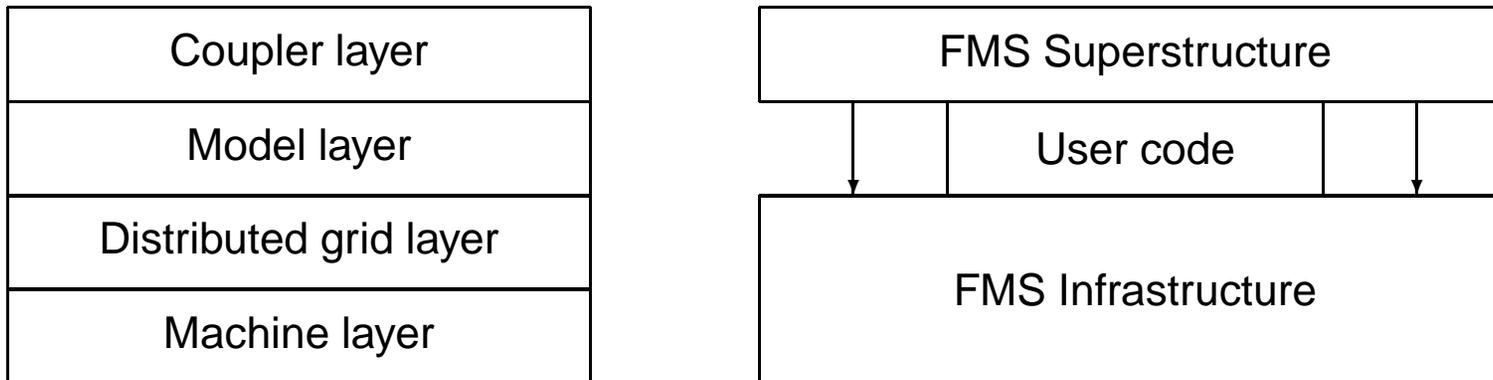
Current GFDL activities using FMS

- Coupled model simulations of climate and climate change targeting the current IPCC cycle;
- Development of seasonal-interannual forecasting capabilities;
- Incorporation of global biogeochemical models into coupled model for carbon cycle modeling;
- Development of next-generation models: AM3 physics, hybrid vertical coordinate ocean models; novel atmospheric grids (cubed sphere); grid nesting and mosaics.

Future developments: FMS and community standards

- FMS authors are now active participants in the design of the Earth Systems Modeling Framework (ESMF) community-wide modeling standard and framework, for which FMS is a design prototype. ESMF prototype v2.0 is currently released to the public. (<http://www.esmf.org>)
- But this is not enough...

Upward evolution of standards



Standards currently sit in the **machine layer** (e.g MPI, netCDF).

In FMS and ESMF, the distributed grid layer is part of an open architecture.

By developing an **open standard** for the **distributed grid layer**, we permit much greater freedom of innovation in software and hardware architectures for scalable systems.

The “standard benchmarks” (LINPACK, SPEC, etc) do not yet reflect this trend.

The FMS infrastructure demonstrates that...

- It is possible to write a data-sharing layer spanning flat shared memory, distributed memory, ccNUMA, cluster-of-SMPs. The API is not as extensive as, say, MPI, but has been designed to serve the climate/weather modeling community.
- It is possible to write another layer that expresses these operations in a manner natural to our algorithms (“halo update”, “data transpose” instead of “buffered send”, “thread nesting”).
- The current standardization efforts (ESMF, PRISM) departs from BLAS, MPI, etc in that they are explicitly formulated in high-level language constructs (classes, modules, types).
- The “standard benchmarks” do not yet stress the high-level language abstractions used by this community. The HPC industry and the standards bodies must be actively involved in this effort.

The jaws of code complexity

```
error = atmos()
```

```
common a(ni,nj,nk); do
```

load a to vreg

multithreaded speculative prefetch