Evolution of standards in modeling software

V. Balaji
SGI/GFDL Princeton University

Bruce Ross Symposium
GFDL Smagorinsky Room
Princeton, New Jersey
20 June 2002

GFDL Computing

- Reliance on vector architecture in previous decades.
- Transition to scalable computing begun in 1997 with the acquisition of Cray T3E.
- Current computing capability: $2 \times 256 + 6 \times 128 + 2 \times 64$p Origin 3000.

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, Matt Harrison, Isaac Held, Paul Kushner, Ron Pacanowski, Pete Phillipps, 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).

FMS design principles

Modularity data-hiding, encapsulation, self-sufficiency;

Portability adherence to official language standards, the use of community-standard software packages, compliance with internal standards;

Flexibility address a wide variety of climate issues by configuring particular experiments out of a wide choice of available components and modules.

Extensibility attempt to anticipate future needs: choices for the same physical function to present similar external interfaces;

Community users encouraged to become developers by contributing components: public release of infrastructure, components and complete model configurations.

Architecture of FMS

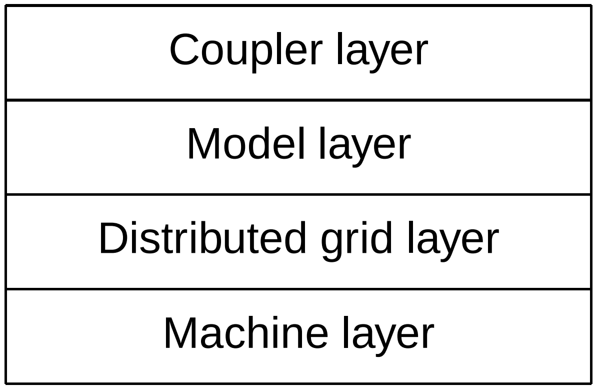
Coupler layer scheduling of component models, data exchange between component models.

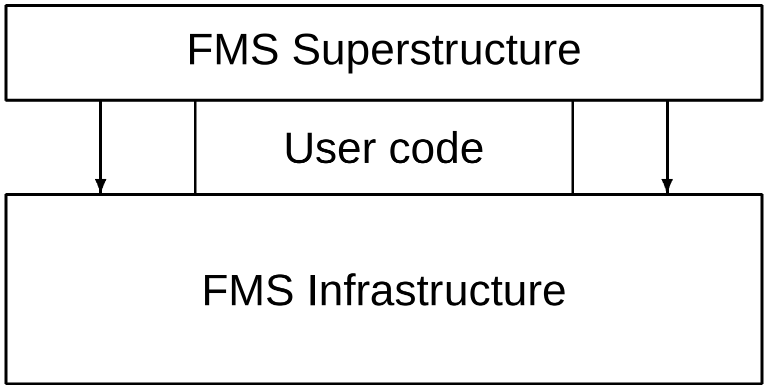
Model layer component models (atmosphere, ocean, etc.) compliant with the framework code standards.

Distributed grid layer standard for physical description of model fields on spatial grids distributed across parallel systems, and parallel operations on these fields.

Machine layer communication primitives (MPI, shmem, etc), I/O, other platform-specific operations.

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.

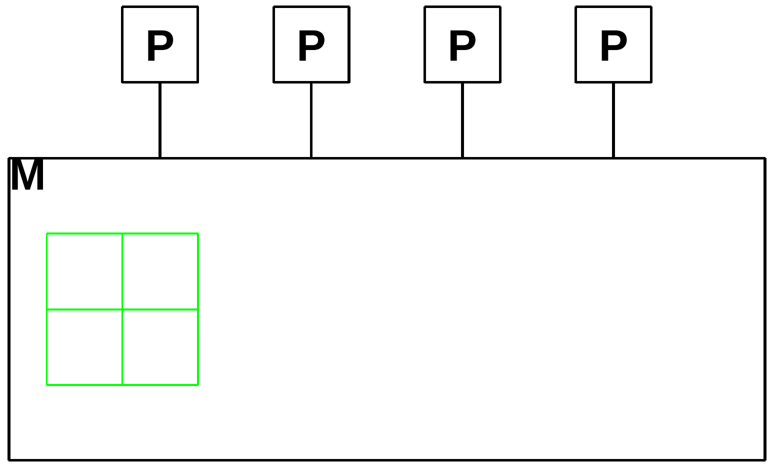
Diagnostics manager Runtime output of model fields.

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

Parallel programming models

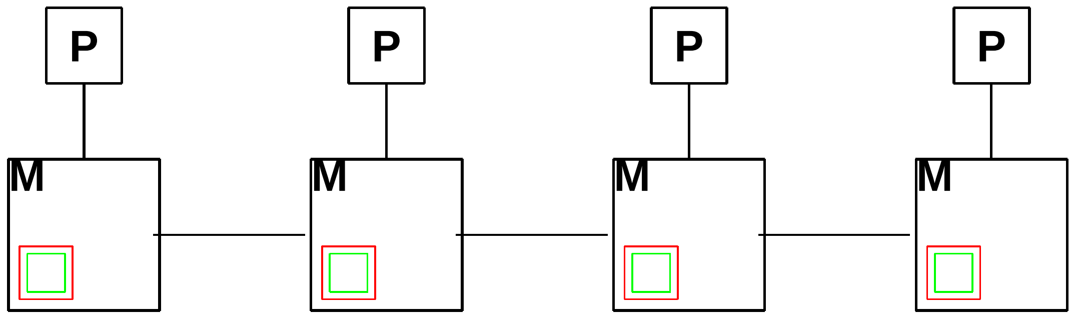
- Shared memory parallelism.
- Distributed memory parallelism.
- Hybrid parallelism.

Shared memory parallelism



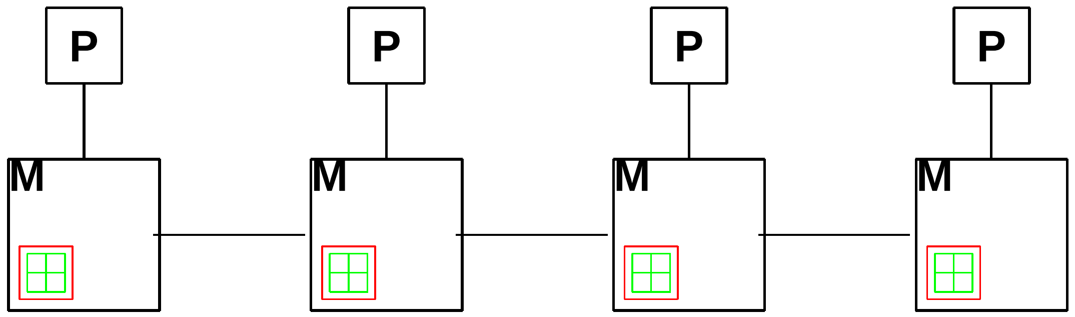
- Canonical architecture: shared memory (UMA), limited scalability.
- Private and shared variables.
- Critical regions.

Distributed memory parallelism



- Canonical architecture: distributed memory (NUMA).
- Decompose global domain `(1:I,1:J)` into `npes` subdomains. `(is:ie,js:je)` defines subdomain start and end.
- Copy data between PEs (message-passing or remote memory access).

Hybrid parallelism



- Canonical architecture: cluster of SMPs.
- Divide global domain `(1:I,1:J)` into `nthreads*npes` threads on `npes` processors. Each processor receives `nthreads` threads.
- Each processor could also be a node on an SMP.

Computer architecture and programming models

- Memory speed will always lag processor speed.
- Shared memory will scale only so far.

Exotic new architectures (HTMT, MTA, etc) attempt various means of latency hiding. PIM attempts to reduce physical distance to memory. But physically distributed memory is a fact of life for the foreseeable future.

To deal with physically distributed memory, one must either have explicit communication (message-passing or remote memory access) or rely on compilers to do the dirty work (ccNUMA).

The MPP modules define a clean interface to various hardware models of physically distributed memory.

The MPP modules

GFDL has a homegrown parallelism API written as a set of 3 F90 modules:

- `mpp_mod` is a low-level interface to message-passing APIs (currently SHMEM and MPI; MPI-2 and Co-Array Fortran to come);
- `mpp_domains_mod` is a set of higher-level routines for domain decomposition and domain updates;
- `mpp_io_mod` is a set of routines for parallel I/O.

`http://www.gfdl.gov/˜vb`

Implementation of mpp_transmit

```
call mpp_transmit( send_buf, n, to_pe, recv_buf, m, from_pe )
```

- MPI: `MPI_Isend()` and `MPI_Recv()`.
- SHMEM: `shmem_get`.
- on shared memory: direct copy.
- on ccNUMA: send address, then direct copy.

`mpp_domains_mod` : domain class library

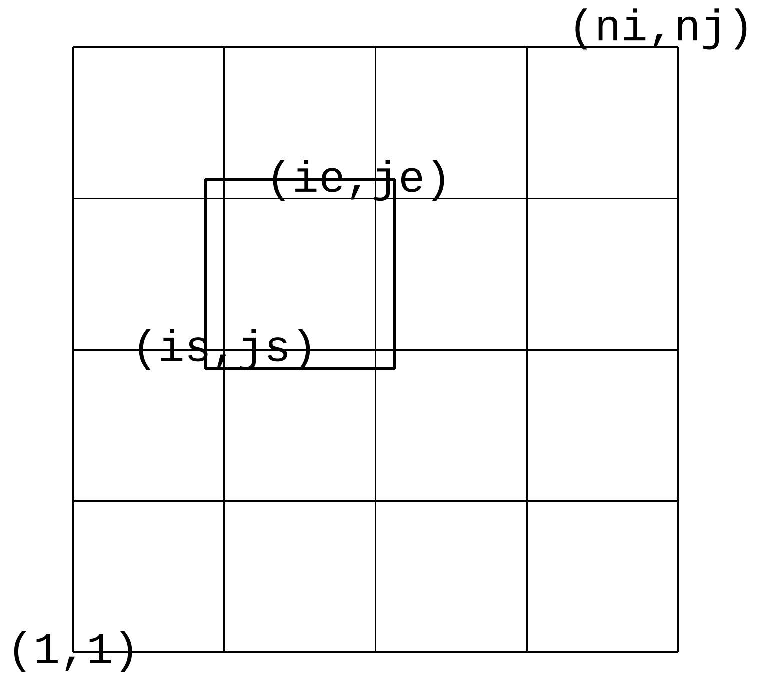
Definition of *domain*:

- *Global domain*: the entire model grid.
- *Compute domain*: set of points calculated by a PE.
- *Data domain*: set of points required by the computation (i.e including halo).

All the information required for domain-related operations are maintained in compact form in the *domain* types supplied by `mpp_domains_mod`. Complicated grids, such as the bipolar grid and the cubed sphere can be represented in this class, so long as they are logically rectilinear.

mpp_domains_mod calls:

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



mpp_io_mod: a 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).

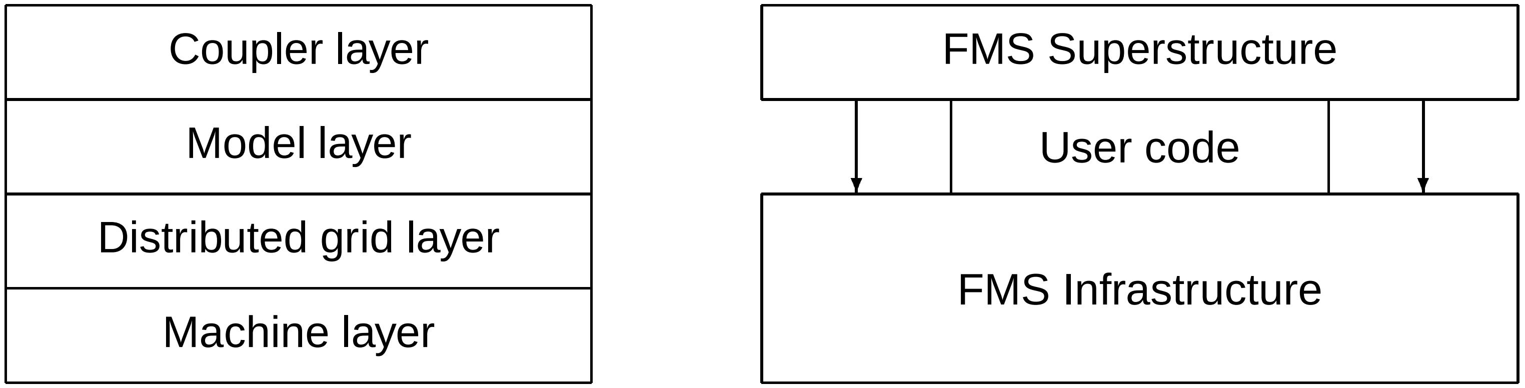
`mpp_io_mod` supports three types of parallel I/O:

- 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).

mpp_io_mod API

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

Upward evolution of standards



Standards currently sit in the *machine layer* (e.g MPI, netCDF), and in FMS, the distributed grid layer is supplied by GFDL.

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.

Summary

- 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 HPC industry and the standards bodies must be actively involved if this is to be a success.