MPI Collectives

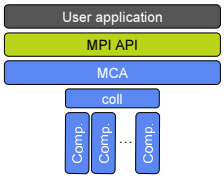
Jeff Squyres

Collective Algorithms

- Hot research topic
- Problem for 3rd party researchers:
  - How to implement new collective algorithms?
- Before components:
  - MPI Profiling Layer
  - Edit existing MPI implementation
  - Create new MPI implementation
  - Use alternate function names

The "coll" Framework (v1)

- Components as the back-end
  - Each contains all 15 MPI collectives



Framework Goals

- Allow multiple components in application
  - Selection scope is per-communicator
- Intuitive interface
- Minimize overhead
- Allow different implementation models
- Allow less-than-full components

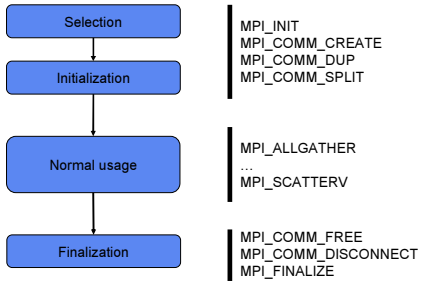
Typical Implementation Models

- Layered over point-to-point
  - Use MPI_SEND, MPI_RECV
- Alternate communication channels
  - Native hardware support for collectives
- Hierarchical coll components
  - Let one coll componet use another
  - …explained later

Interface

- Simple 1-to-1 mapping
  - Selected module hangs off communicator
  - Module has pointers to back-end functions
- → Switch to show file ompi/mca/coll/coll.h
  - Component interface
  - Module interface

Lifecycle



Selection

- For any communicator constructor
  - And when MPI_COMM_WORLD and SELF are created during MPI_INIT
- Query all available components
  - See if they want to run on this communicator
  - Those who do return a module and a priority
  - Keep modules in priority order
  - Highest priority module is initialized
  - All others are "unselected"

Selection Flaw

- Currently assumes that all processes in a communicator make identical selections
  - No attempt to ensure selections match
- Works well in homogeneous environments
  - No one has complained… yet
- Implementation issue; not design issue

Initialization

- Module with highest priority is initialized
- Typically creates / initializes private data
  - Pre-computes data for faster invocation
  - Allocates resources
  - Stored on comm->c_coll_selected_data
- If module contains NULL for any function
  - Replaced with "basic" version
  - Technically, this is an abstraction violation
  - Lots of special case code in coll base for this

Normal Usage

- Module is cached on the communicator
  - Can also have private / opaque data hung off communicator
- Example: MPI_BCAST
  - Invokes comm->c_coll.coll_bcast(…)
- Module can use cached private data

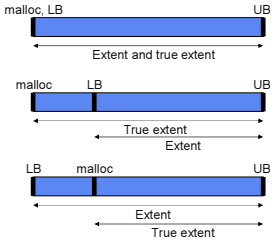
Sidenote: Temporary Buffers

→ From mca/coll/basic/coll_basic_reduce

- Sometimes you need a temporary buffer
  - E.g., tree-based reduce
- Need two different values:
  - How many bytes to malloc
  - Pointer to give to MPI_SEND (etc.)

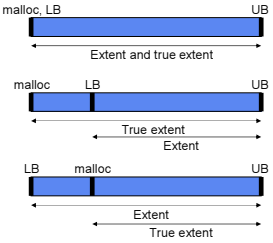
Sidenote: Temporary Buffers

- 3 buffer cases
  - malloc == LB
  - malloc < LB
  - malloc > LB
- MPI buffer is always (malloc – LB)



Sidenote: Temporary Buffers

- For count > 1
  - Need to malloc more than (N * true_extent)
- Easiest to malloc:
  - 1 true_extent +
  - (N-1) * extent
- This is more than necessary



Temporary Buffers

```
/* Get extent and true extent */
ompi_ddt_get_extent(dtype, &lb, &extent);
ompi_ddt_get_true_extent(dtype, &true_lb,
   &true_extent);

/* Allocate more space than we need */
free_buffer = malloc(true_extent + (count - 1) *
   extent);
/* Pointer that we give to MPI_SEND (etc.) */
pml_buffer = free_buffer - lb;
```

Existing Components

- Basic
  - Baseline linear and logarithmic algorithms
  - Intra- and intercommunicators
- Shmem
  - Intracommunicator only
  - 4 NUMA-aware shared memory collectives
  - Barrier, Broadcast, Reduce (1 flavor), Allreduce

Existing Components

- Tuned
  - The new “basic”
  - Results from UTK collective research
  - Lots of different algorithms for each operation
- Hiearch
  - Hierarchical collectives, divided by latency
  - Make sub-communicators at latency boundaries
  - Invoke relevant collectives in sub-comms

Existing Components

- Self
  - For MPI_COMM_SELF (and clones)
  - If one process in communicator, returns priority of 75 (otherwise, NULL)
  - Simple no-op’s or memcpy’s (depending on operation)
  - Intended so that no other coll components need to handle this case
  - → Show ompi/mpi/coll/self/*.c

Coll v2 Framework

- Under active design
  - Will likely wholly replace v1
- Much more ambitious than v1
- Optional session tonight to discuss current thoughts / designs (not yet complete)

MPI Topologies

MPI Topology Overview

- MPI_CART_* and MPI_GRAPH_*
  - N dimensional Cartesian
  - Arbitrary graphs
- Allow MPI to re-order ranks
  - If it has “special” knowledge
- Allow user app to send “up,” “down,” “left,” “right,” etc.

Framework Overview

- Scope is per-communicator
- Components lazily loaded
  - Decrease memory footprint
  - Loaded at first CART/GRAPH invocation

Topo Base

- Implements all functions via:
  - MPI_CART_MAP
  - MPI_GRAPH_MAP
- Hence, components only need to provide these
  - Can provide NULL pointers in the module
  - Replaced with base functions

Unity Component

- Makes a 1-to-1 mapping
  - Only provide MPI_*_MAP functions
  - Very simplistic
  - All other functions are NULL
- Interested to have others implement “more interesting” mappings
  - Allow re-ordering to map network layout