Modular Component Architecture

Jeff Squyres

Why Components?

- Core set included in Open MPI distribution
- 3rd parties can develop / distribute
  - Open MPI development to the community
  - As source or binary (open vs. closed source)
- Can be added to existing Open MPI install
  - Reduce the need for multiple MPI installations
  - Can even be added on a per-user basis
- Run-time decisions (vs. compile-time)

Why Components?

- Better software engineering
  - Enforce strict abstraction barriers
- Small, discrete chunks of code
  - Good for learning / new developers
  - Easier to maintain and extend
- Separate user apps from back-end libraries
  - E.g., user MPI apps not compiled against libibverbs.so / libgm.so / libpbs.a

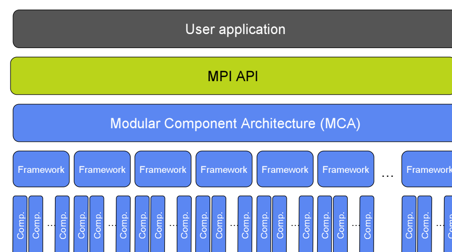
MCA

- MCA
  - Top-level architecture for component services
  - Find, load, unload components
- Frameworks
  - Targeted set of functionality
  - Defined interfaces
  - E.g., MPI point-to-point, high-resolution timers

MCA

- Components
  - Think "plugins"
  - Code that exports a specific interface
  - Loaded / unloaded at run-time
- Modules
  - A component paired with resources
  - E.g., "TCP" component loaded, finds 2 TCP NICs, makes 2 TCP modules
- Component:C++ class :: Module:C++ object

MCA Top-Level View



MCA Organization

- Three entities:
  - MCA base architecture
  - Frameworks
  - Components
    (modules are run-time “instances” of components)
- Everything is versioned
  - (Major, minor, release) triple
  - Allows for backwards compatibility
  - Nothing currently has multiple versions

MCA Organization

- Frameworks
  - Have unique string names
  - One namespace, despite three sections
- Components
  - Belong to exactly one framework
  - Have unique string names
  - Namespace is per framework
- All names must be valid C variable names

Organized by Directory

- <section>/mca/<framework>/<component>
  - Section = opal, orte, ompi
  - Framework = framework name, or “base”
  - Component = component name, or “base”
- Directory names must match
  - Framework name
  - Component name
- Examples
  - ompi/mca/btl/tcp, ompi/mca/btl/openib

“Base”

- Reserved name: “base”
  - opal/mca/base: the MCA itself
  - orte/mca/pls/base: the PLS framework
  - ompi/mca/btl/base: the BTL framework
- Helper functions / header files
  - Common to all components in that framework
  - Public data / methods to be invoked from outside the framework

Header File Conventions

- Framework interface defined in
  - <section>/mca/<framework>/<framework>.h
  - This is mandatory
- Public base functions declared in
  - <section>/mca/<framework>/base/base.h
  - This is not mandatory, but common

OPAL Framework Types

- opal/mca/*
  - maffinity: Memory affinity
  - memory: Memory hooks
  - paffinity: Processor affinity
  - timer: High-resolution timers

ORTE Framework Types

- orte/mca/*
  - errmgr: Error manager
  - iof: I/O forwarding
  - gpr: General purpose registry
  - ns: Name server
  - oob, rml: Communication
  - pls: Process launch / control
  - rmgr, rds, ras, rmaps: Resource manager, discovery, allocation, mapping
  - sds: Startup discovery service
  - soh: State of health monitor

OMPI Framework Types

- ompi/mca/*
  - allocator: Memory allocation
  - coll: Collective operations
  - io: Parallel I/O
  - mpool: Memory pooling
  - osc: One-sided operations
  - pml, bml, btl: Point-to-point
  - rcache: Registration cache
  - topo: Topology management

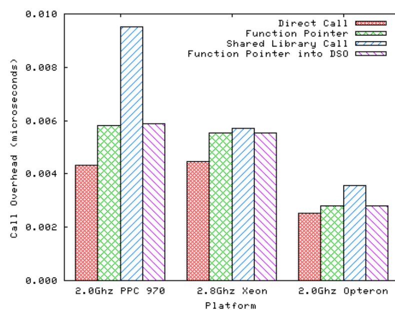
Components

- Back-end technologies
  - Function pointers
  - Usually compiled as dynamic shared objects (DSO's) in .so files ("plugins")
  - But can be included in libmpi (etc.)
- Use GNU Libtool "ltdl" library
  - Portable dlopen(), dlsym()
  - Even works on Windows
  - Not GPL (!)

Function Pointers

- Most common criticism
  - "Using pointers to invoke functions are slow!"
- Not so, Grasshopper
  - Euro PVM/MPI 2004 paper proved otherwise
  - Always faster than a shared library call

Indirect Function Call Overhead



Base Component Interface

- Common structure for all components
  - "Parent" class
- Switch to show opal/mca/mca.h

Base Component Fields

- MCA version (triple)
- Framework name / version (triple)
- Component name / version (triple)
  - Simplifying convention: included component versions = Open MPI version
  - Unless difference is meaningful (e.g., bug fix release)
- Open and close methods
  - Open can return failure

Definition: Availability

- Components are "available" if:
  - Can be found at run-time (e.g., they were compiled)
  - Can be opened at run-time (e.g., they can find all the shared libraries that they need)
  - The "open" function returns SUCCESS

Definition: Selection

- Act of picking which components to use
  - Typically involves querying each available
  - Strongly discourage having framework know specifics about any individual component
- Each framework has different selection rules and criteria
  - Must select >= 0 components
  - Must select >= 1 components
  - Must select exactly 1 component

Definition: Scope

- Applicability of component selection
- Example: per-process
  - Open: MPI_INIT
  - Selection: MPI_INIT
  - Finalize: MPI_FINALIZE
  - Close: MPI_FINALIZE

Definition: Scope

- Example: per-communicator
  - Open: MPI_INIT (or lazy)
  - Selection: Communicator constructors
  - Finalize: Communicator destructors
  - Close: MPI_FINALIZE
- …defined by framework, so other scenarios possible

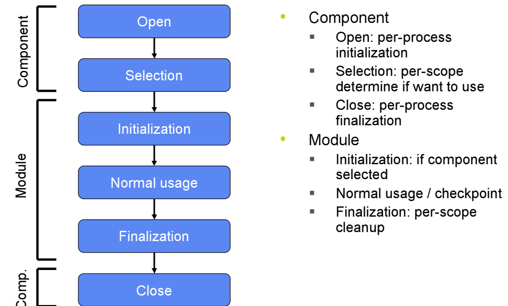
Amorphousness

- MCA base is strictly defined
- Each framework builds upon the base
  - But definitions are framework-specific
  - Every framework is different
  - Depends on what the framework is for
- Therefore somewhat difficult to describe
- But most follow common conventions

Component Interface

- Defined by the framework
  - But guaranteed to have the base component as the first member
- Typically has some kind of selection function
  - “Do you want to be used with X?”
  - Where “X” is relevant to the framework
  - E.g.: Coll components – “Do you want to be used with communicator X?”

Component / Module Lifecycle



Run-Time Tunable Parameters

Tunable Parameters

- Philosophy: do not use constants
  - Use run-time parameters instead
- Referred to as “MCA parameters”
  - Somewhat misleading name
  - Means: service provided by the MCA base
  - Does not mean that they are restricted to MCA components or frameworks
  - OPAL, ORTE, and OMPI layers have parameters

Rationale

- Make everything a run-time decision
  - Give every param a “sensible” default
  - Open question what to do about params that cannot have globally sensible defaults
- Parameters usually indicate:
  - Values (e.g., short/long message size)
  - Behavior (e.g., selection of algorithm)
- Much easier than recompiling

Intrinsic MCA Params

- Each framework name is an MCA param
  - Specifies which components to open
- MCA base automatically registers it
  - Value is a comma-delimited list of component names
  - Default value is empty (meaning “all”)
- Inclusionary or exclusionary behavior
  - btl=tcp,self,sm
  - btl=^tcp

MCA Param Lookup Order

1. "Override" value
2. mpirun command line
   - mpirun –mca <name> <value>
3. Environment variable
   - setenv OMPI_MCA_<name> <value>
4. File
   - $HOME/.openmpi/mca-params.conf
   - $prefix/etc/openmpi-mca-params.conf (these locations are themselves tunable)
5. Default value

Using MCA Parameters

- Characteristics
  - Strings and integers
  - Read-only (information) and read-write
  - Private and public
- Components *must* register params during component open
- WARNING: Lookup is slow!
  - Do not put it in critical performance path
  - Initialize at beginning of scope

MCA Param Examples

- btl_gm_version
  - Read-only, string version of the GM library that the BTL gm component was compiled against
- btl_tcp_if_include
  - Read-write, string list of TCP interfaces to use
- btl
  - Read-write, list of BTL components to use
- orte_base_singleton
  - Private, whether this process is a singleton

Sidenote: ompi_info Command

- Tells everything about OMPI installation
  - Finds all components and all params
  - Great for debugging
- Can look up specific component
  - ompi_info --param <framework> <component>
  - Shows params and current values
  - Can also use keyword "all"
- "--parsable" option
- → Run ompi_info command

MCA Param API

- → Show opal/mca/base/mca_base_param.h
- Register and lookup
  - Several variations of each
- Components *must* register during open
  - ompi_info calls open/close on every component that it finds (to find parameters)