Open MPI
Join the Revolution

Supercomputing
November, 2005

http://www.open-mpi.org/

Open MPI Mini-Talks

- **Introduction and Overview**
  - Jeff Squyres, Indiana University
- **Advanced Point-to-Point Architecture**
  - Tim Woodall, Los Alamos National Lab
- **Datatypes, Fault Tolerance and Other Cool Stuff**
  - George Bosilca, University of Tennessee
- **Tuning Collective Communications**
  - Graham Fagg, University of Tennessee

Open MPI:
Introduction and Overview

Jeff Squyres
Indiana University

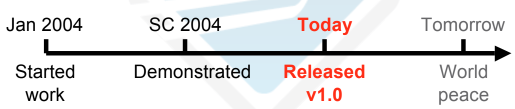
http://www.open-mpi.org/

Technical Contributors

- Indiana University
- The University of Tennessee
- Los Alamos National Laboratory
- High Performance Computing Center, Stuttgart
- Sandia National Laboratory - Livermore

MPI From Scratch!

- Developers of FT-MPI, LA-MPI, LAM/MPI
  - Kept meeting at conferences in 2003
  - Culminated at SC 2003: Let's start over
  - Open MPI was born

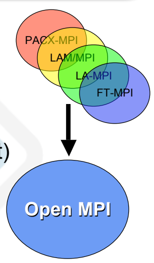


MPI From Scratch: Why?

- Each prior project had different strong points
  - Could not easily combine into one code base
- New concepts could not easily be accommodated in old code bases
- Easier to start over
  - Start with a blank sheet of paper
  - Decades of combined MPI implementation experience

MPI From Scratch: Why?

- Merger of ideas from
  - FT-MPI (U. of Tennessee)
  - LA-MPI (Los Alamos)
  - LAM/MPI (Indiana U.)
  - PACX-MPI (HLRS, U. Stuttgart)



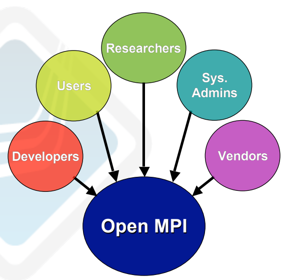
Open MPI Project Goals

- All of MPI-2
- Open source
  - Vendor-friendly license (modified BSD)
- Prevent “forking” problem
  - Community / 3rd party involvement
  - *Production-quality* research platform (targeted)
  - Rapid deployment for new platforms
- Shared development effort

Open MPI Project Goals

- Actively engage the HPC community
  - Users
  - Researchers
  - System administrators
  - Vendors
- Solicit feedback and contributions

➡ *True open source model*



Design Goals

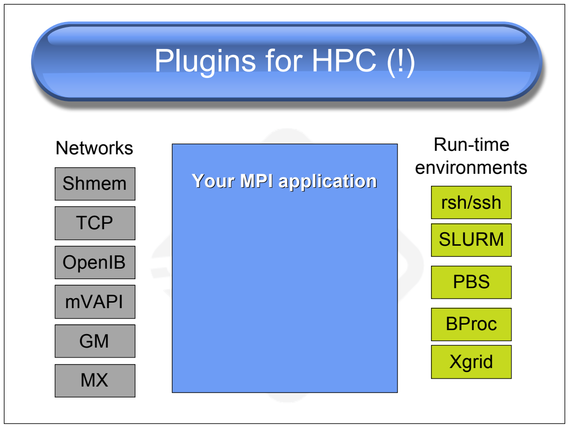
- Extend / enhance previous ideas
  - Component architecture
  - Message fragmentation / reassembly
  - Design for heterogeneous environments
    - Multiple networks (run-time selection and striping)
    - Node architecture (data type representation)
  - Automatic error detection / retransmission
  - Process fault tolerance
  - Thread safety / concurrency

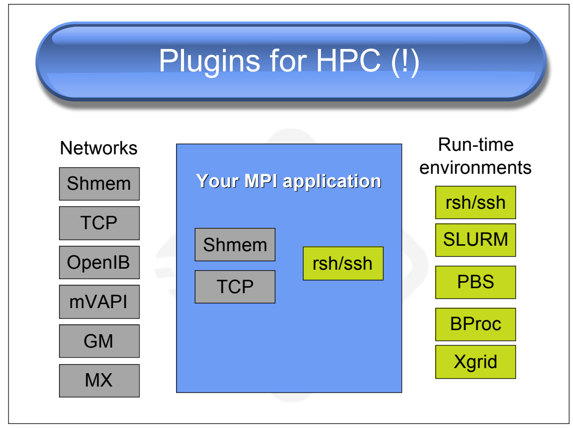
Design Goals

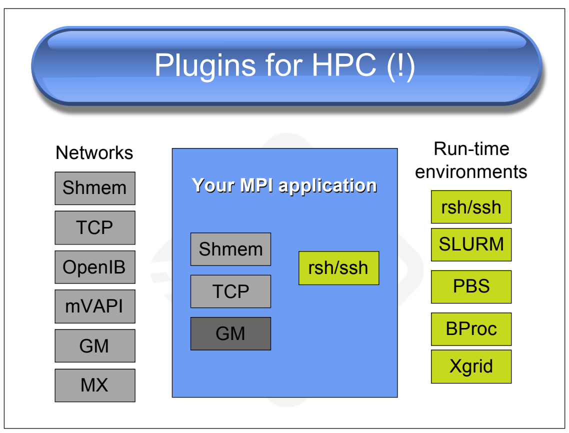
- Design for a changing environment
  - Hardware failure
  - Resource changes
  - Application demand (dynamic processes)
- Portable efficiency on any parallel resource
  - Small cluster
  - “Big iron” hardware
  - “Grid” (everyone a different definition)
  - …

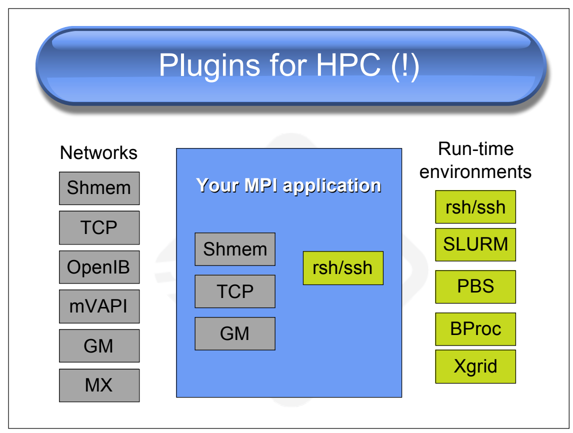
Plugins for HPC (!)

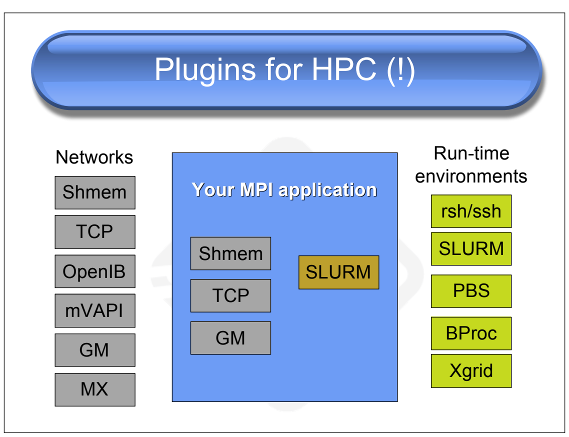
- Run-time plugins for combinatorial functionality
  - Underlying point-to-point network support
  - Different MPI collective algorithms
  - Back-end run-time environment / scheduler support
- Extensive run-time tuning capabilities
  - Allow power user or system administrator to tweak performance for a given platform

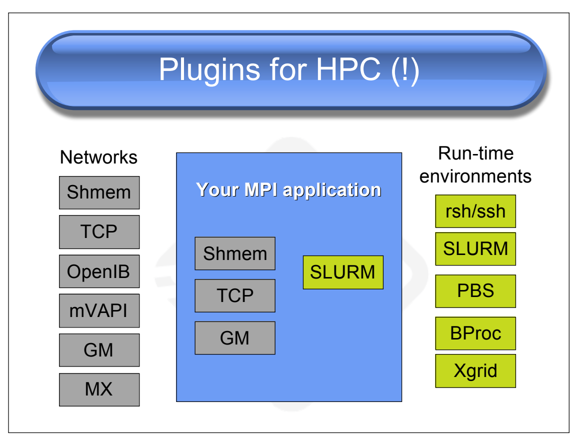

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

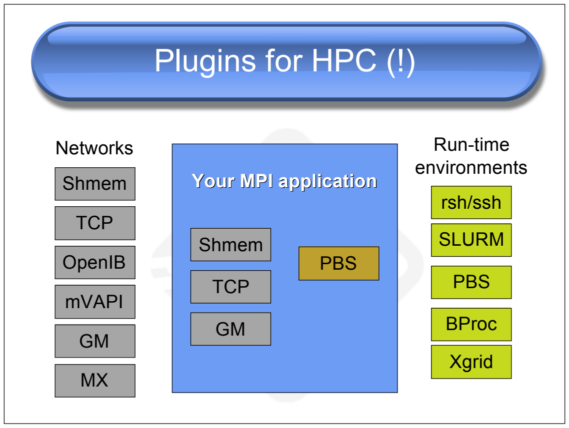

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
rsh/ssh
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

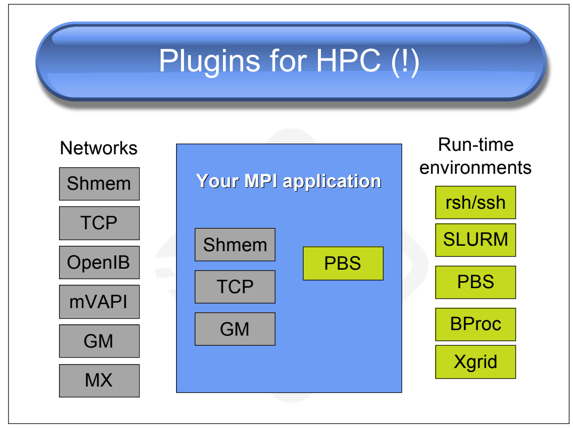

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
GM
rsh/ssh
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

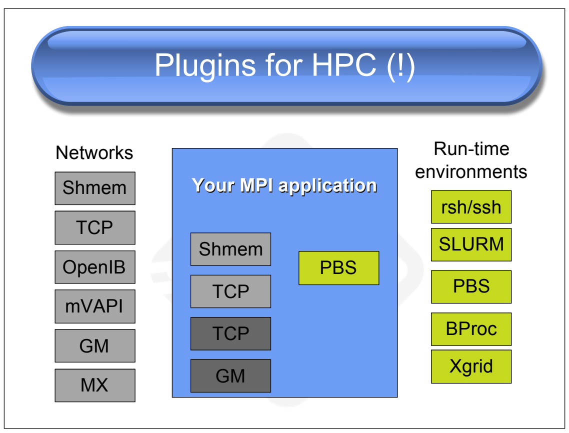

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
GM
rsh/ssh
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

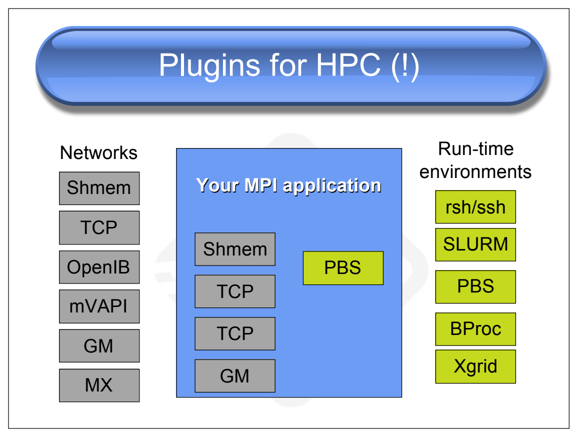

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
GM
SLURM
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

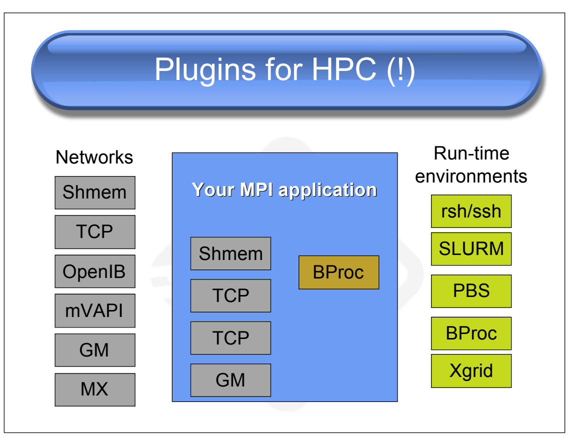

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
GM
SLURM
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

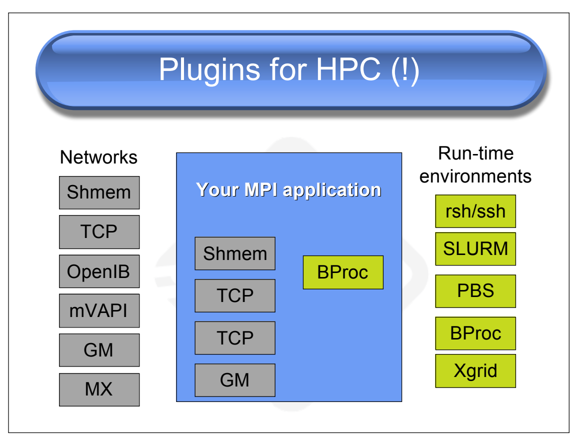

Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
GM
PBS
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid


Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
GM
PBS
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid


Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
TCP
GM
PBS
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid


Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
TCP
GM
PBS
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid


Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
TCP
GM
BProc
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid


Plugins for HPC (!)
Networks
Shmem
TCP
OpenIB
mVAPI
GM
MX
Your MPI application
Shmem
TCP
TCP
GM
BProc
Run-time environments
rsh/ssh
SLURM
PBS
BProc
Xgrid

Current Status

- v1.0 released (see web site)
- Much work still to be done
  - More point-to-point optimizations
  - Data and process fault tolerance
  - New collective framework / algorithms
  - Support more run-time environments
  - New Fortran MPI bindings
  - …
- *Come join the revolution!*

Open MPI: Advanced Point-to-Point Architecture

Tim Woodall

Los Alamos National Laboratory

`http://www.open-mpi.org/`

Advanced Point-to-Point Architecture

- Component-based
- High performance
- Scalable
- Multi-NIC capable
- Optional capabilities
  - Asynchronous progress
  - Data validation / reliability

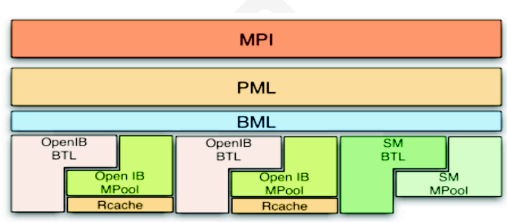
Component Based Architecture

- Uses Modular Component Architecture (MCA)
  - Plugins for capabilities (e.g., different networks)
  - Tunable run-time parameters

Point-to-Point Component Frameworks

- Byte Transfer Layer (BTL)
  - Abstracts lowest native network interfaces
- Point-to-Point Messaging Layer (PML)
  - Implements MPI semantics, message fragmentation, and striping across BTLs
- BTL Management Layer (BML)
  - Multiplexes access to BTL's
- Memory Pool
  - Provides for memory management / registration
- Registration Cache
  - Maintains cache of most recently used memory registrations

Point-to-Point Component Frameworks



Network Support

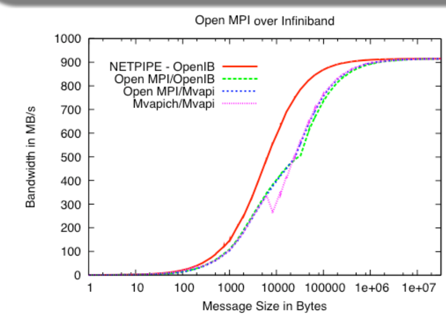
- Native support for:
  - Infiniband: Mellanox Verbs
  - Infiniband: OpenIB Gen2
  - Myrinet: GM
  - Myrinet: MX
  - Portals
  - Shared memory
  - TCP
- Planned support for:
  - IBM LAPI
  - DAPL
  - Quadrics Elan4

Third party contributions welcome!

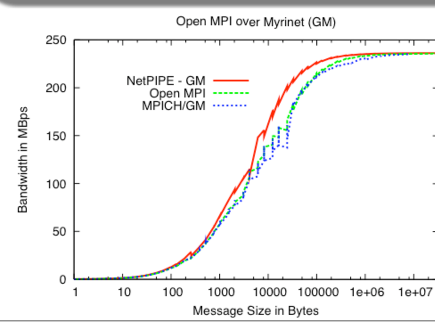
High Performance

- Component-based architecture *does not impact performance*
- Abstractions leverage network capabilities
  - RDMA read / write
  - Scatter / gather operations
  - Zero copy data transfers
- Performance on par with (*and exceeding*) vendor implementations

Performance Results: Infiniband



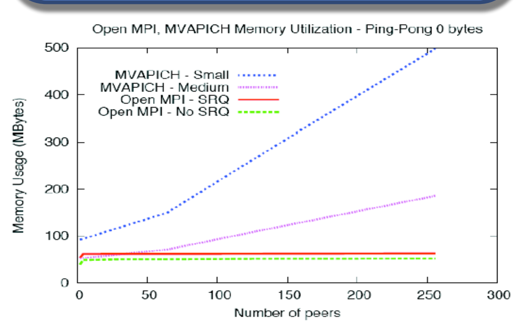
Performance Results: Myrinet



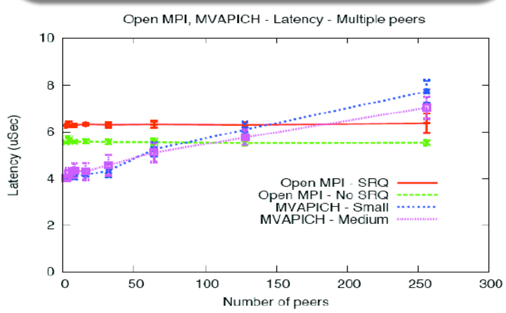
Scalability

- On-demand connection establishment
  - TCP
  - Infiniband (RC based)
- Resource management
  - Infiniband Shared Receive Queue (SRQ) support
  - RDMA pipelined protocol (dynamic memory registration / deregistration)
  - Extensive run-time tuneable parameters:
    - Maximum fragment size
    - Number of pre-posted buffers
    - ....

Memory Usage Scalability



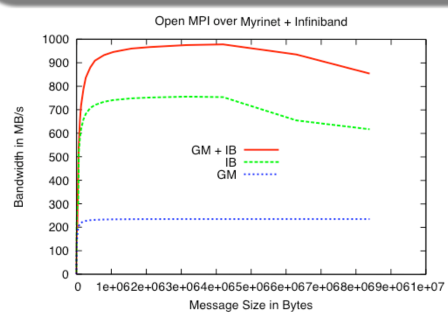
Latency Scalability



Multi-NIC Support

- Low-latency interconnects used for short messages / rendezvous protocol
- Message stripping across high bandwidth interconnects
- Supports concurrent use of heterogeneous network architectures
- Fail-over to alternate NIC in the event of network failure (work in progress)

Multi-NIC Performance



Optional Capabilities (Work in Progress)

- Asynchronous Progress
  - Event based (non-polling)
  - Allows for overlap of computation with communication
  - Potentially decreases power consumption
  - Leverages thread safe implementation
- Data Reliability
  - Memory to memory validity check (CRC/checksum)
  - Lightweight ACK / retransmission protocol
  - Addresses noisy environments / transient faults
  - Supports running over connectionless services (Infiniband UD) to improve scalability

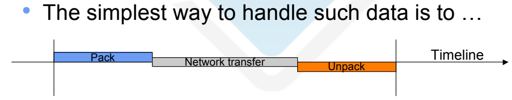
Open MPI: Datatypes, Fault Tolerance, and Other Cool Stuff

George Bosilca
University of Tennessee

http://www.open-mpi.org/

User Defined Data-type

- MPI provides many functions allowing users to describe non-contiguous memory layouts
  - MPI_Type_contiguous, MPI_Type_vector, MPI_Type_indexed, MPI_Type_struct
- The send and receive type must have the same signature, but not necessary have the same memory layout
- The simplest way to handle such data is to …



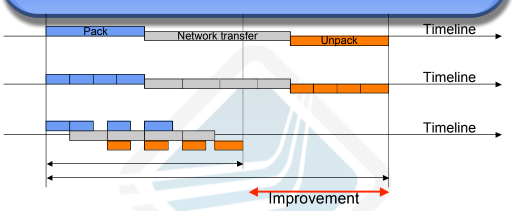
Problem With the Old Approach

- [Un]packing: intensive CPU operations.
  - No overlap between these operations and the network transfer
  - The requirement in memory is larger
- Both the sender and the receiver have to be involved in the operation
  - One to convert the data from its own memory representation to some standard one
  - The other to convert it from this standard representation in it's local representation.

How Can This Be Improved?

- No conversion to standard representation (XDR)
  - Let one process convert directly from the remote representation into its own
- Split the packing / unpacking into small parts
  - Allow overlapping between the network transfer and the packing
- Exploit gather / scatter capabilities of some high performance networks

Open MPI Approach



- Reduce the memory pollution by overlapping the local operation with the network transfer

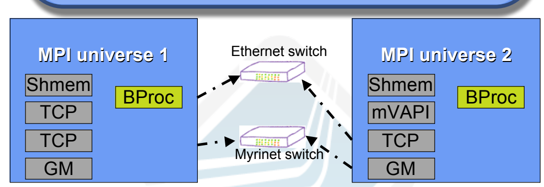
Improving Performance

- Others questions:
  - How to adapt to the network layer?
  - How to support RDMA operations?
  - How to handle heterogeneous communications?
  - How to split the data pack / unpack?
  - How to correctly convert between different data representations?
  - How to realize data type matching and transmission checksum?
- Who handles all this?
  - MPI library can solve these problems
  - User-level applications cannot

MPI 2 Dynamic Processes

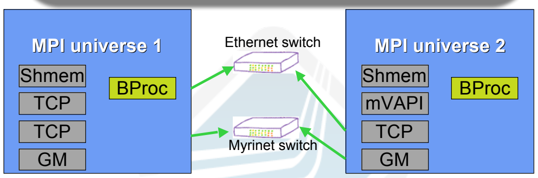
- Increasing the number of processes in an MPI application:
  - MPI_COMM_SPAWN
  - MPI_COMM_CONNECT / MPI_COMM_ACCEPT
  - MPI_COMM_JOIN
- Resource discovery and diffusion
  - Allows the new universe to use the "best" available network(s)

MPI 2 Dynamic processes



- Discover the common interfaces
  - Ethernet and Myrinet switches
- Publish this information in the public registry

MPI 2 Dynamic processes



- Retrieve information about the remote universe
- Create the new universe

Fault Tolerance Models Overview

- Automatic (no application involvement)
  - Checkpoint / restart (coordinated)
  - Log Based (uncoordinated)
    - Optimistic, Pessimistic, Casual
- User-driven
  - Depends on application specifications, then the application recover the algorithmic requirements
  - Communication mode: rebuild, shrink, blank
  - Message mode: reset, continue

Open Questions

- Detection
  - How can we detect that a fault happens?
  - How can we globally decide the faulty processes?
- Fault management
  - How to propagate this information to everybody involved?
  - How to handle this information in a dynamic MPI-2 application?
- Recovery
  - Spawn new processes
  - Reconnect the new environment (scalability)
- How can we handle the additional entities required by the FT models (memory channels, stable storages …) ?

Open MPI: Tuning Collective Communications; Managing the Choices

Graham Fagg

Innovative Computing Laboratory

University of Tennessee

http://www.open-mpi.org/

Overview

- Why collectives are so important
- One size doesn't fit all
- Tuned collectives component
  - Aims / goals
  - Design
  - Compile and run time flexibility
- Other tools
  - Custom tuning
- The Future

Why Are Collectives So Important?

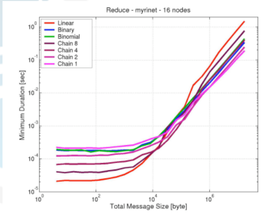
- Most applications use collective communication
  - Stuttgart HLRS profiled T3E/MPI applications
  - 95% used collectives extensively (i.e. more time spent in collectives than point to point)
- The wrong choice of a collective can increase runtime by orders of magnitude
- This becomes more critical as data and node sizes increase

One Size Does Not Fit All

- Many implementations perform a run-time decision based on either communicator size or data size (or layout, etc.)

The reduce shown for just a **single small** communicator size has **multiple** 'cross over points' where **one** method performs better than the rest

(note the **LOG** scales)



Tuned Collective Component: Aims and Goals

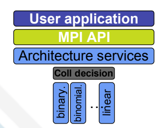
- Provide a number of methods for each of the MPI collectives
  - Multiple algorithms/topologies/segmenting methods
  - Low overhead efficient call stack
  - Support for low level interconnects (i.e. RDMA)
- Allow the user to choice the best collective
  - Both at compile time and at runtime
- Provide tools to help users understand which, why and how some collectives methods are chosen (including application specific configuration)

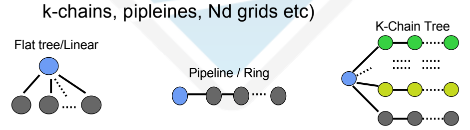
Four Part Design

- The MCA framework
  - The tuned collectives behaves as any other Open MPI component
- The collectives methods themselves
  - The MPI collectives backend
  - Topology and segmentation utilities
- The decision function
- Utilities to help users tune their system / application

Implementation

1. MCA framework
   - Has normal priority and verbose controls via MCA parameters
2. MPI collectives backend
   - Supports: Barrier, Bcast, Reduce, Allreduce, etc.
   - Topologies: Trees (binary, binomial, multi-fan in/out, k-chains, pipleines, Nd grids etc)



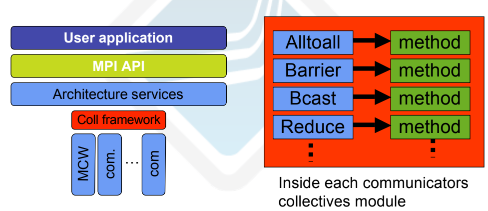


Implementation

3. Decision functions
   - Decided which algorithm to invoke based on:
     - Data previously provided by user (e.g., configuration)
     - Parameters of the MPI call (e.g., datatype, count)
     - Specific run-time knowledge (e.g., interconnects used)
   - Aims to choose the optimal (or best available) method

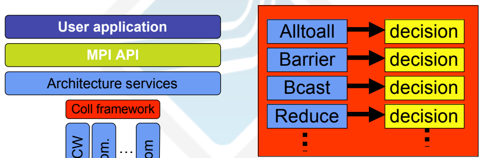
Method Invocation

- Open MPI communicators each have a function pointer to the backend collective implementation



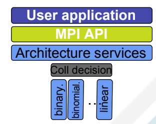
Method Invocation

- The tuned collective component changes the method pointer to a decision pointer



Inside each communicators collectives module

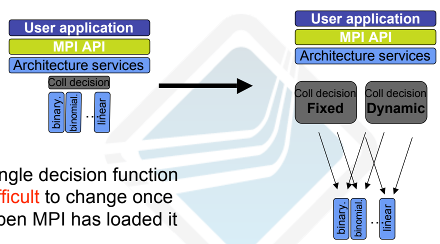
How to Tune?



Single decision function difficult to change once Open MPI has loaded it

One decision function per Communicator per MPI call

How to Tune?



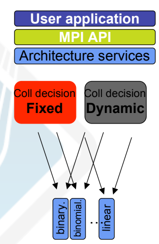
Single decision function difficult to change once Open MPI has loaded it

One decision function per Communicator per MPI call

Fixed Decision Function

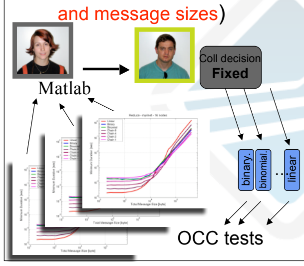
- *Fixed* means the decision functions are as the module was ***compiled***
- You can change the component, recompile it and rerun the application if you want to change it

➡ Since this is a plugin, there is no need to re-compile or re-link the application



Fixed Decision Function

The fixed decision functions *must* decide a method for *all* possible [valid] input parameters (i.e., ALL communicator and message sizes)

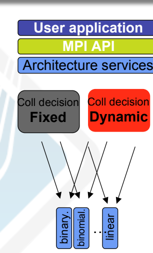



```
commute = _atb_op_get_commute(op);
  if ( gcommode != FT_MODE_BLANK ) {
    if ( commute ) {
      /* for small messages use linear algorithm */
      if (msgsize <= 4096) {
         mode = REDUCE_LINEAR;
         *segsize = 0;
      } else if (msgsize <= 65536 ) {
         mode = REDUCE_CHAIN;
         *segsize = 32768;
         *fanout = 8;
      } else if (msgsize < 524288) {
         mode =  REDUCE_BINTREE;
         *segsize = 1024;
         *fanout = 2;
      } else {
         mode = REDUCE_PIPELINE;
         *segsize = 1024;
         *fanout = 1;
      }
```

Dynamic Decision Function

- *Dynamic* means the decision functions are changeable as each communicator is created
- Controlled from a file or MCA parameters

➡ Since this is a plugin, there is no need to re-compile or re-link the application

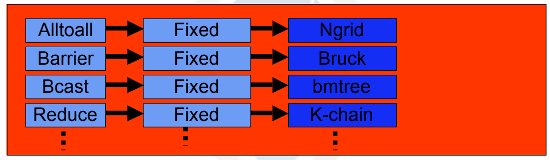


Dynamic Decision Function

- Dynamic decision = run-time flexibility
- Allow the user to control each MPI collective individually via:
  - A fixed override (known as *"forced"*)
  - A per-run configuration file
  - Or both
- Default to fixed decision rules if neither provided

MCA Parameters

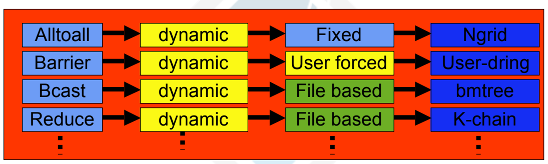
- Everything is controlled via MCA parameters



--mca coll_tuned_use_dynamic_rules 0

MCA Parameters

- Everything is controlled via MCA parameters



--mca coll_tuned_use_dynamic_rules 1

User-Forced Overrides

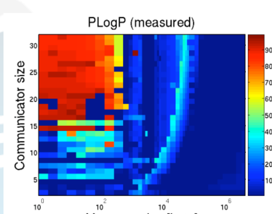
- For each collective:
  - Can choose a specific algorithm
  - Can tune the parameters of that algorithm
- Example: MPI_BARRIER
  - Algorithms
    - Linear, double ring, recursive doubling, Bruck, two process only, step-based bmree
  - Parameters
    - Tree degree, segment size

File-Based Overrides

- Configuration file holds detailed rule base
  - Specified for each collective
  - Only the overridden collectives need be specified
- The rule base is only loaded once
  - Subsequent communicators share the information
  - Saves memory footprint

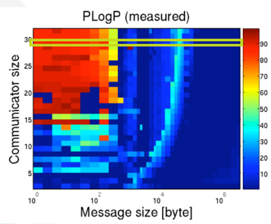
File-Based Overrides

- Pruned set of values
  - A complete set would have to map every possible comm size and data size/type to a method and its parameters (topology, segmentation etc)
- Lots of data!
- And lots of measuring to get that data

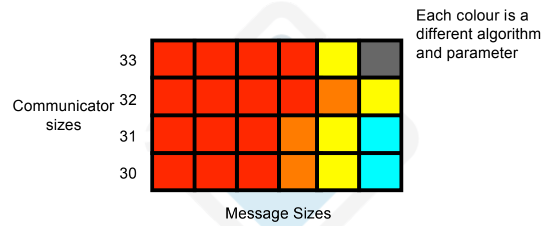


Pruning Values

- We know some things in advance
  - Communicator size
- Can therefore prune
  - 2D grid of values
  - Communicator size vs. message size
  - Maps to algorithm and parameters

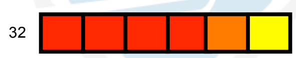


How to Prune



How to Prune

- Select communicator size, then search all elements
  - Linear: slow, but not too bad
  - Binary: faster, but more complex than linear



How to Prune

- Construct “clusters” of message sizes
- Linear search by cluster
  - Number of compares = number of clusters

File-Based Overrides

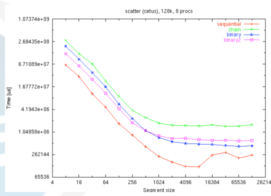
- Separate fields for each MPI collective
- For each collective:
  - For each communicator size:
    - Message sizes in a run length compressed format
- When a new communicator is created it only needs to know its communicator size rule

Automatic Rule Builder

- Replaces dedicated graduate students who love Matlab!
- Automatically determine which collective methods you should use
  - Performs a set of benchmarks
  - Uses intelligent ordering of tests to prune test set down to a manageable set
- Output is a set of file-based overrides

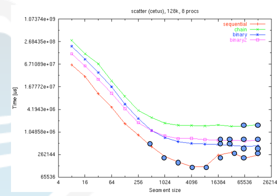
Example: Optimized MPI_SCATTER

- Search for:
  - Optimal algorithm
  - Optimal segment size
  - For 8 processes
  - For 4 algorithms
  - 1 message size (128k)
- Exhaustive search
  - 600 tests
  - *Over 3 hours (!)*



Example: Optimized MPI_SCATTER

- Search for:
  - Optimal algorithm
  - Optimal segment size
  - For 8 processes
  - For 4 algorithms
  - 1 message size (128k)
- Intelligent search
  - 90 tests
  - 40 seconds



Future Work

- Targeted Application tuning via Scalable Application Instrumentation System (SAIS)
- Used on DOE SuperNova TeraGrid application
  - Selectively profiles an application
  - Output compared to a mathematical model
  - Decide if current collectives are non-optimal
  - Non-optimal collective sizes can be retested
  - Results then produce a tuned configuration file for a particular application

Join the Revolution!

- **Introduction and Overview**
  - Jeff Squyres, Indiana University
- **Advanced Point-to-Point Architecture**
  - Tim Woodall, Los Alamos National Lab
- **Datatypes, Fault Tolerance and Other Cool Stuff**
  - George Bosilca, University of Tennessee
- **Tuning Collective Communications**
  - Graham Fagg, University of Tennessee

http://www.open-mpi.org/