Open MPI - A High Performance MPI for Distributed Environments

Richard L. Graham

PPAM 2005

LA-UR-05-6902

Overview

- Introduction to Open MPI
- Rut-Time Environment (ORTE)
- Point-to-Point Communication
- Future - Near term

“Core” Technical Contributors

- Indiana University (LAM/MPI)
- The University of Tennessee (FT-MPI)
- High Performance Computing Center, Stuttgart (PAX-MPI)
- Los Alamos National Laboratory (LA-MPI)

“Core” Team Members

- LANL
  - Tim Woodall
  - David Daniel
  - Ralph Castain
  - Jim Barker
  - Galen Shipman
  - Rich Graham
- U of Tennessee
  - George Bosilca
  - Graham Fagg
  - Thara Angskun
  - Jack Dongarra
- Indiana U.
  - Jeff Squyres
  - Brian Barrett
  - Josh Hursey
  - Andrew Lumsdain
- HLRS
  - Rainer Keller
- U of Houston
  - Edgar Gabriel
- SNL - Livermore
  - Mitch Sukalski

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

Open MPI Project Goals

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

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 data error detection / retransmission
  - Process fault tolerance

Design Goals -Cont'd

- 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)
  - …

Implementation Goals

- All of MPI-2
- Low latency
  - E.g., minimize management traffic
- High bandwidth
  - E.g., stripe messages across multiple networks
- Production quality
- Thread safety and concurrency (MPI_THREAD_MULTIPLE)

Implementation Goals - Cont'd

- Based on a component architecture
  - Flexible run-time tuning
  - "Plug-ins" for different capabilities (e.g., different networks)
- Natively support commodity networks
  - TCP
  - Shared memory
  - Myrinet
    - GM, MX
  - Infiniband
    - mVAPI, OpenIB
  - Portals
  - Quadrics Elan4 §
  - LAPI §

(§ = future)

Operating Systems

- Current
  - Linux
  - OS X (BSD)
  - Solaris
  - AIX
  - Catamount
- Development
  - MS Windows
- Maybe?
  - HP/UX
  - IRIX

Run-Time Environments

- Daemon and daemon-less modes
- Current support
  - rsh / ssh
  - BProc (LANL Bproc Clustermatic)
  - POE
  - PBS/Torque
  - SLURM
- Development
  - Yad (Red Storm)
  - LSF
- Future
  - BProc (Scyld)
  - RMS (Quadrics)
  - Grid (“multi-cell”)

Projects within “Open MPI”

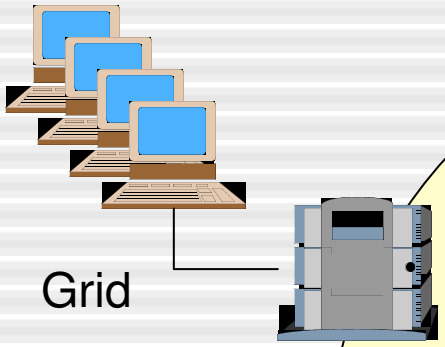
- Open MPI (OMPI) - MPI library
- Open Run-Time-Environment (ORTE) - Run-Time library
- Open Portable Access Layer library - Common support for OMPI and ORTE

Current Status

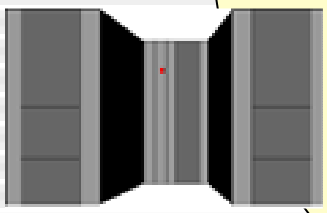
- SVN repository read-only available at www.open-mpi.org
- First production release 4Q05
  - End of September, 2005
  - All of MPI-2 except one-sided operations
  - One sided expected by SC (Nov 2005)

Run-Time Environment (ORTE)

Open RTE - Design Overview



Grid

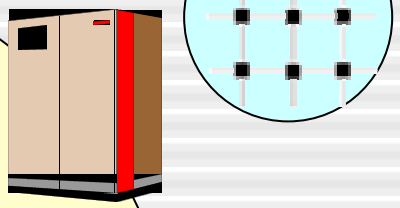


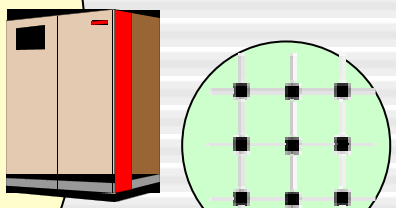
Single Computer

Seamless, transparent environment for high-performance applications

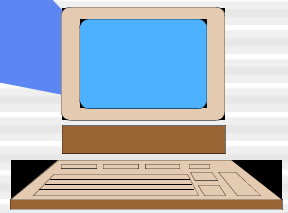
- Inter-process communications within and across cells
- Distributed publish/subscribe registry
  - Supports event-driven logic across applications, cells
  - Persistent, fault tolerant
- Dynamic “spawn” of processes, applications both within and across cells

Cluster

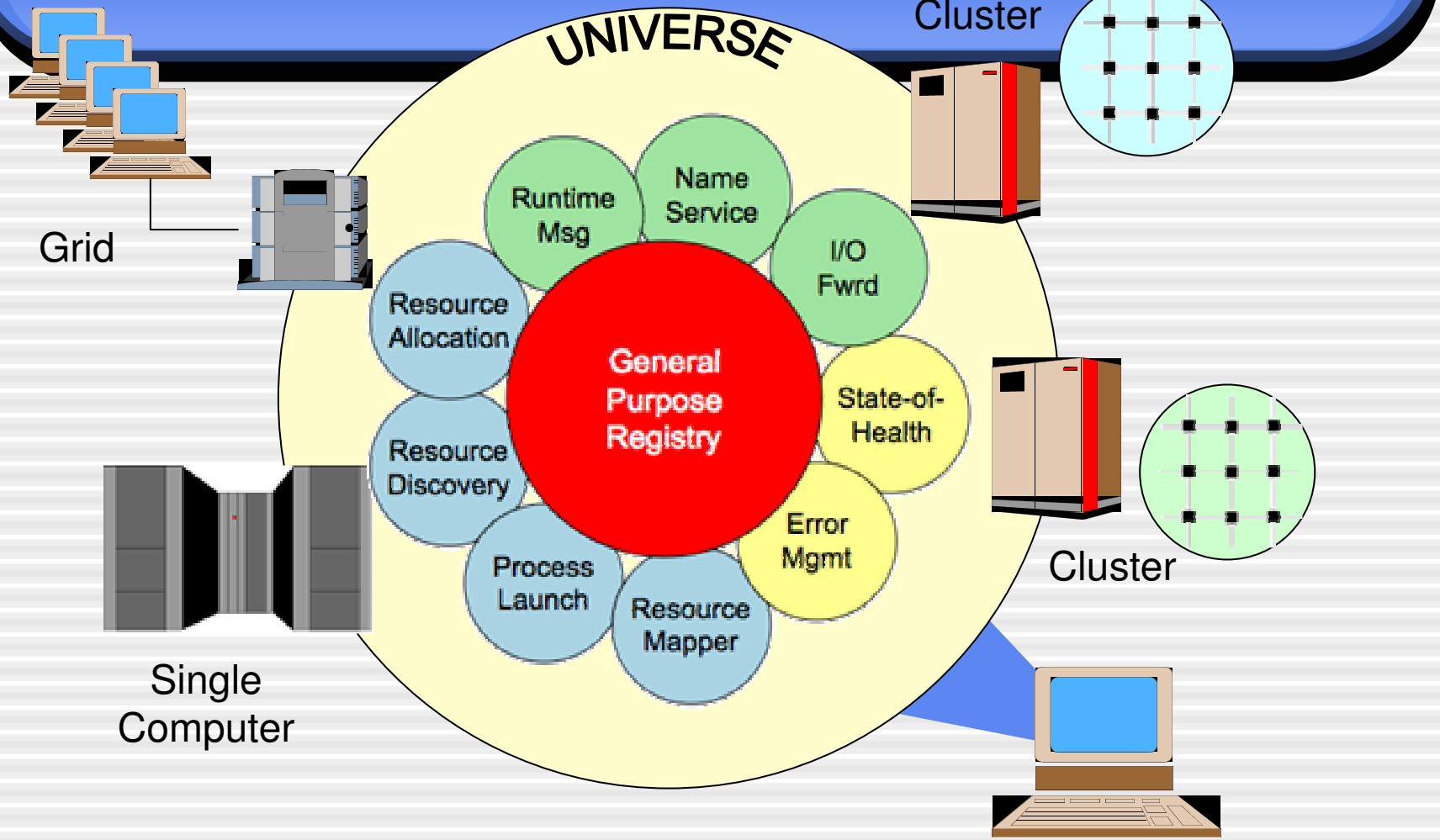




Cluster



Open RTE - Components



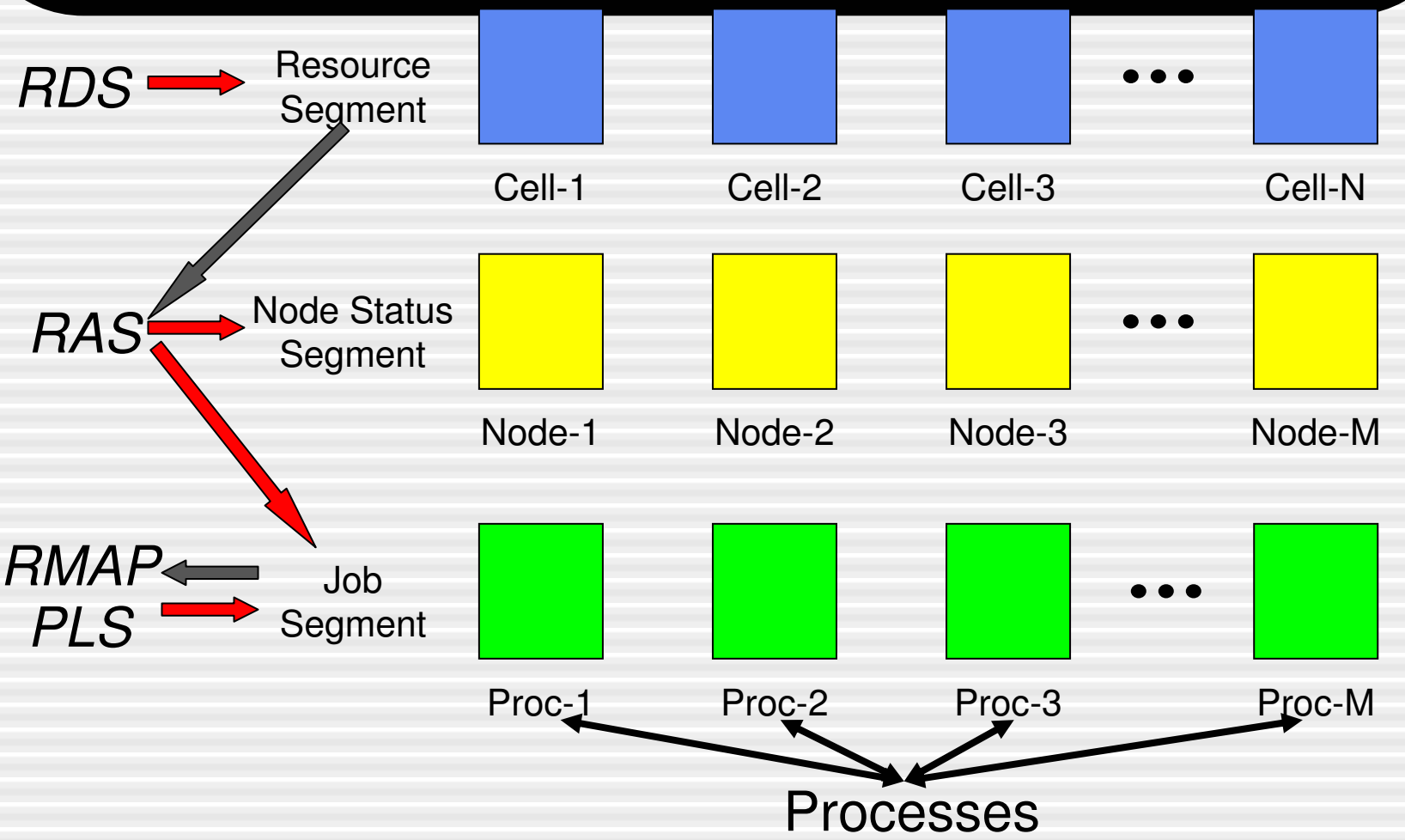
General Purpose Registry

- Cached, distributed storage/retrieval system
  - All common data types plus user-defined
  - Heterogeneity between storing process and recipient automatically resolved
- Publish/subscribe
  - Support event-driven coordination and notification
  - Subscribe to individual data elements, groups of elements, wildcard collections
  - Specify actions that trigger notifications

Subscription Services

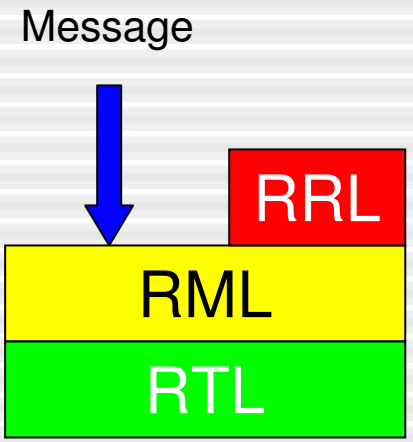
- Subscribe to container and/or keyval entry
  - Can be entered before data arrives
  - Specifies data elements to be monitored
    - Container tokens and/or data keys
    - Wildcards supported
  - Specifies action that generates event
    - Data entered, modified, deleted
    - Number of matching elements equals, exceeds, is less than specified level
    - Number of matching elements transitions (increases/decreases) through specified level
- Events generate message to subscriber
  - Includes specified data elements
  - Asynchronously delivered to specified callback function on subscribing process

Resource Manager



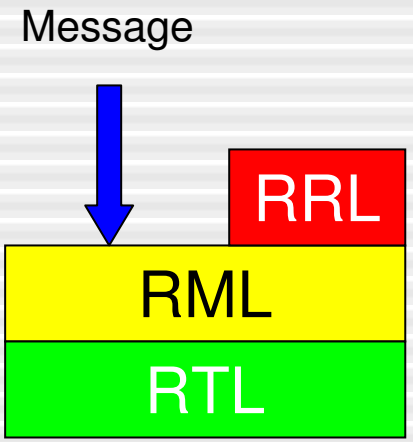
ORTE Messaging

- Runtime Messaging Layer (RML)
  - API for sending messages
  - Provides usual interfaces
    - Send/recv, send_buffer/recv_buffer
    - Blocking, non-blocking
    - Reliable (guaranteed delivery - default)
    - Non-guaranteed (send-and-forget)
  - Message size limited by
    - Available addressable memory
    - Configuration parameter
- Runtime Routing Layer (RRL)
- Runtime Transport Layer (RTL)



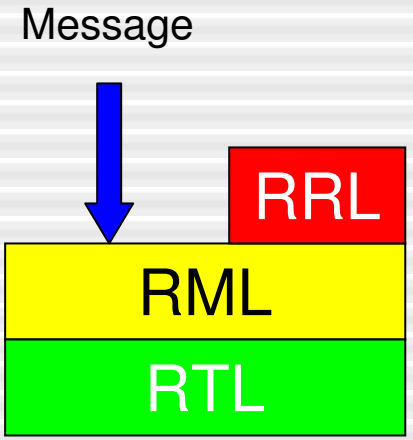
ORTE Messaging

- Runtime Messaging Layer (RML)
- Runtime Routing Layer (RRL)
  - Determines optimal routing for messaging system
  - Defines inter-cell routing
  - Selects transport to be used
    - Priorities set at compile, execution
    - Fastest layer used according to message size, priority
- Runtime Transport Layer (RTL)



ORTE Messaging

- Runtime Messaging Layer (RML)
- Runtime Routing Layer (RRL)
- Runtime Transport Layer (RTL)
  - Actual transport for messaging system
  - Multiple pathways supported in parallel
    - Identified at compile, execution, and dynamically by program
    - TCP/IP (default)
    - High-speed paths (InfiniBand, Myrinet, …) as available and appropriate



Data Packing Subsystem

- Compensates for heterogeneity of sender/recipient
  - Transparent to user
- Packs all native C data types
- All ORTE-defined data types

Naming Service

- Assign each application a unique name
- Assign each process a unique name
- Used for communications purposes

I/O Forwarding

- Stream-level redirection
- Captures all standard I/O
  - from process initiation to completion (daemon-based systems)
  - From MPI_INIT to MPI_FINALIZE (non-daemon systems)

State-of-Health Monitor

- Process state-of-health
  - Tracks process successful startup, notifies of abnormal terminations
- System state-of-health
  - Tracks node status (up, down, booting, …)
  - Updates system
    - Provides info for identifying where to move processes on faulty nodes
    - Where to launch processes when user doesn't specify

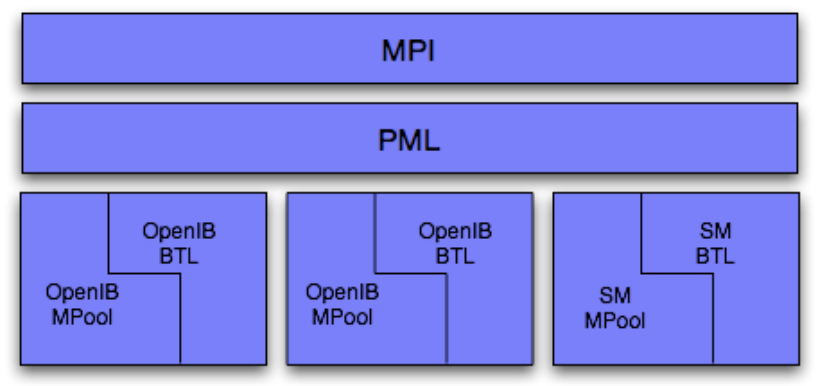
Error Manager

- Log ORTE errors for reporting, future analysis
- Centralized strategy for responding to non-recoverable errors
  - Ensures notification results in appropriate action
- Forwards all signals to processes
  - Specific recipient, broadcast

Point-To-Point Communications

Architecture

- Component Architecture:
  - “Plug-ins” for different capabilities (e.g. different networks)
  - Tuneable run-time parameters
- Three component frameworks:
  - Point-to-point messaging layer (PML) implements MPI semantics
  - Byte Transfer Layer (BTL) abstracts network interfaces
  - Memory Pool (mpool) provides for memory management/registration



PML Framework

- Single PML manages multiple BTL modules
  - Maintains set of BTLs on a per-peer basis
  - Message fragmentation and scheduling
- Implements MPI semantics
  - Synchronous / buffered / ready / normal sends
  - Persistent requests / Request completion
- OB1 : Eager/Rendezvous protocol
  - Eager send of short messages
  - Configurable threshold (short vs. long)
  - Multiple long protocols

PML Protocols

- Send / receive pipeline to / from pre-registered buffers (non-contiguous data)
- MPI_Alloc_mem support
  - Red/black tree of memory registrations
  - BTL associated with registration is used by scheduler
  - Xfer of contiguous data with 1 RDMA (after match)
- “Leave pinned” run-time parameter
  - Registration on first-use
  - MRU cache (configurable size) of registrations
  - Bandwidth equivalent to pre-registered buffers (MPI_Alloc_mem)

PML Protocols (Continued)

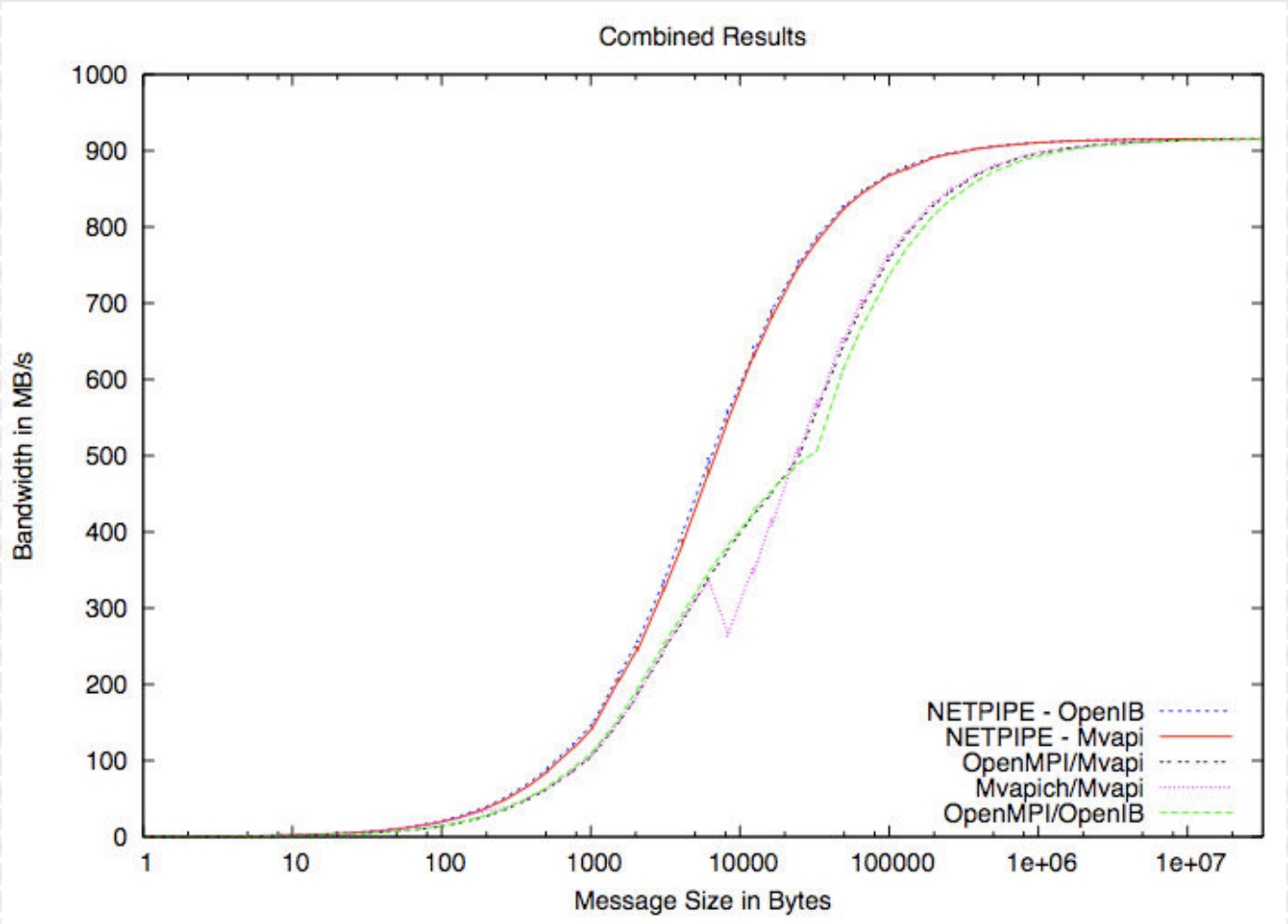
- Dynamic memory registration/deregistration
  - Fragment message and build pipeline of RDMA requests
  - Overlap [de-]registration with RDMA
  - Bandwidth 97% of pre-registered memory at large message sizes (8Mbytes - IB)
  - Performance impacted by bus type/bandwidth

Latency (Ping-Pong)

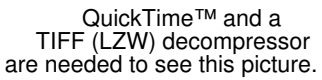
	Average Latency
Open MPI - OpenIB - *optimized	5.13 usec
Open MPI - OpenIB - *defaults	5.43 usec
Open MPI - Mvapi - *optimized	5.64 usec
Open MPI - Mvapi - *defaults	5.94 usec
Mvapich - Mvapi (rdma/mem poll)	4.19 usec
Mvapich - Mvapi (send/recv)	6.51 usec
Open MPI - GM	6.86 usec
MPICH-GM	7.10 usec
Open MPI - Shared Memory	0.94 usec
MPICH2 - Shared Memory	1.07 usec

* Send/Recv based protocol

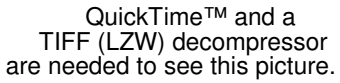
Bandwidth - Infiniband



Bandwidth - GM



Bandwidth - Shared Memory



Future - Near term

Collectives

- “Self Tuning” collectives
- Hierarchical

Fault-Tolerance: Active Research

- LAM/MPI supports checkpoint / restart
  - Open MPI will too…soon
  - LAM/MPI - style
  - MPICH-V - style
- FT-MPI supports user-level restarting
  - Including use of MPI_SPAWN
  - This will eventually be rolled into Open MPI

Data Fault Tolerance

- Usually an unrecognized problem
  - Irrelevant with TCP (guaranteed protocol)
  - It matters when no end-to-end data integrity (Myrinet and Infiniband)
- LA-MPI data reliability
  - Will eventually be rolled into Open MPI
  - FY2006

Questions?