MPI on BG/L

Bill Gropp and Rusty Lusk

(other team members: Ralph Butler, Rob Latham, David Ashton, Brian Toonen, Rob Ross, Rajeev Thakur, Anthony Chan)

Mathematics and Computer Science Division
Argonne National Laboratory
{gropp,lusk}@mcs.anl.gov
Outline

• Why MPI on BG/L?
• Challenges for MPI implementations provided by BG/L
• MPICH
  • The Abstract Device Approach to MPI implementation
  • MPICH-2 and scalability
  • Early results of IBM/Argonne Collaboration
• Scalable Process Management
  • MPD and the SciDAC Scalable Systems Software Project
  • MPI / Process Manager Interface
  • MPD and LoadLeveler on BG-L
  • Early results of IBM/Argonne Collaboration
• Conclusion
  • BG/L will provide at least one convenient and familiar programming and job scheduling environment
IBM Collaborators

• T. J. Watson
  • Jose Moreira
  • Gheorghe Almasi
  • Silvius Rus

• Haifa
  • Edi Shmueli
  • Yariv Aridor
  • Tamar Domany
  • Yosef Moatti
Why Have MPI on BG/L?

• BG/L does support MPI model
  • Separate address spaces (though small) for separate processes
• Vast number of parallel applications ready to run, or at least ready to begin work on
  • No barrier at programming model level (familiar message-passing model)
  • No barrier at language level (C, Fortran, C++, Fortran 90)
  • No barrier at communication library level (MPICH)
  • Memory requirement barriers likely at data/process level
  • Scalability barriers likely at algorithm level
• Demonstration of general purpose nature of machine
  • If MPI can be implemented, so can anything else
Challenges for an MPI Implementation on BG/L

- Small memory footprint (fingerprint?) per MPI process
- Scalability of data structures
  - Local size must be independent of total number of processes
  - Buffer management
- Scalability of algorithms
  - Must take advantage of BG/L hardware support, especially for collective operations
  - MPI topology routines will become more important
- Scalability of process manager interactions
  - Interaction with MPI library
  - Interaction with user
    - Convenient familiar direct interface to process manager (mpirun, mpiexec) or to batch scheduler (LoadLeveler)
MPICH

• Goals
  • Supply research vehicle for MPI implementation issues
  • Promote standard programming model for users
  • Provide vendors and others with starting point for specialized MPI implementations (both commercial and research)
    • Architected to support replacement of components
  • MPICH-1
    • Began during MPI standardization process
    • Current version 1.2.4, 2500 downloads/month
    • Complete implementation of MPI-1.2, plus I/O from MPI-2
    • Basis of many research and vendor implementations
      • MPICH-GM from Myrinet
      • MPI on ASCI Red (scalable to 3000+ nodes)
      • Early Cray, Meiko, SGI, HP/Compaq, NEC, other implementations
      • Research groups experimenting with lower levels
    • Windows version
MPICH-2

• Original goals of MPICH, plus
  • Scalability to 100,000 processes
  • Improved performance in multiple areas
  • Portability to new interconnects
  • Thread safety
  • Full MPI-2 Standard (I/O, RMA, dynamic processes, more)

• Not yet released
  • Detailed design complete and publicly available
  • Core functionality (point-to-point and collective operations) from MPI-1 complete
    • Early performance results
  • MPI-1 part to be released this fall
Structure of MPICH-2

MPICH-2

PMI
- Fork
- MPD
- Vendors

ADI-3
- Multi-Method
- Channel Interface

ADIO
- Existing parallel file systems

Vendors

For others

TCP

BG/L

MM

Portals

PVFS

Existing

In Progress

Existing
The Abstract Device Interface

- Key to Performance and Portability
- MPICH-2 based on 3rd-generation ADI design (ADI-3)
- Research Topics
  - Combining performance with portability
  - Latency reduction
  - Multi-method
  - Thread safety
  - High-performance MPI datatype processing
  - Interaction with process management, MPI topology routines
  - Multiple approaches to collective operations
    - (For example, need not be in terms of point-to-point operations)
  - Sophisticated implementation of remote-memory operations
  - Dealing with faults
Possible Implementations of the ADI

- The “Channel” device
  - Small number of functions
  - Straightforward to implement
  - Sacrifices some opportunities for optimization
  - Current approach for BG/L

- The “Multimethod” device
  - Allows mixing of communication methods
    - TCP, Shared memory, NIC-based (Myrinet, Infiniband, others)
  - Made more difficult by MPI’s “ANY_SOURCE” in MPI_Recv
  - Intermethod interface by which new methods may be added

- The “Custom” device
  - Specialized to a particular environment
  - Usable by vendors (e.g., Myricom, who have studied ADI-3)
  - Optimum performance
  - Under discussion for BG/L
ADI Status and Plans

• Status
  • TCP implementation of the CH3 implementation of ADI-3 done
  • Multimethod implementation of ADI-3 under way
  • Both faster than in MPICH-1 (see following charts)

• Plans
  • Complete implementation of multimethod device
  • Tune and port to other environments (shared memory, Infiniband)
  • Continued vendor collaboration
    • Myricom plans to implement ADI-3
    • Current discussions with IBM on ADI/CH interface for BG/L
    • Collaborations with multiple Infiniband vendors in progress
An Example: CH3 Implementation over TCP

- Pollable and active-message data paths
- RMA Path
Early Results on Channel/TCP Device

- Conclusion: little added overhead over low-level communication
  - But will become more critical with high-performance network
BG/L and the MPICH2 Architecture

Interface

Message Passing Interface

Abstract Device Interface

Channel Interface

Implementation

MPI

Types, key values
notion of requests

MPID

Transform to
pt2pt ops

CH3

Request
progress engine

TCP/IP

BG/L Torus

BG/L Tree

BG/L GI

Special opportunities:
• collective bypass
• scalable buffer mgmnt
• out-of-order network
Abstract Device Interface/MPID

Channel Interface/CH3

Transport: Torus Message Layer

Torus Packet Layer

Torus hardware

MPICH2

BG/L software

MPI

USER

MPI_Send

MPI_Bcast

MPID_Send

CH_Write

Channel_Write

Torus_Send

lfpdux()

sfpdnx()
Some Questions That You Are About to Ask

- Out-of-order delivery of packets in the network
  - Channel device enhanced to simplify support
  - Few MPI communications require ordering; channel supports ordering of message headers to enable message tracing tools such as Jumpshot

- Implementation of collectives without MPI point-to-point (e.g., using the other network)
  - Improved version of mechanism used in MPICH-1 (introduced for the Meiko) allows each collective operation to use special routines on a communicator-by-communicator basis

- Scalable eager buffer and connection management
  - Dynamic buffer allocation and connection management is consistent with the ADI design (virtual connection table, currently an array, can be replaced with a sparse array).

- Polling and non-polling
  - Design supports both. Neither is always best.
Some More Questions

- **Thread safety**
  - Careful use of atomic operations avoids locks in many cases. Both configure-time and runtime control of the level of thread safety. All versions support OpenMP-style loop parallelism

- **Process(or) topology**
  - Interface through MPI_Cart_create and MPI_Graph_create

- **RMA (one-sided)**
  - Design uses operation aggregation to eliminate extra operations and access windows to eliminate serialization present in other implementations of MPI RMA

- **Rendezvous optimizations**
  - Single communication method case can use an “unexpected receive” approach, already used in some prototype MPI implementations, to avoid one handshake message
Another Question

• How to best use the second CPU?
  • Multiple modes possible
    • 2nd CPU idle ("heater" mode)
    • 2nd CPU runs 2nd thread in same MPI process ("symmetric" mode)
      – At least initially
      – Exploring using for 2nd MPI process ("virtual node" mode)
    • 2nd CPU acts as communication co-processor
      – Allows true overlap of computation and communication
      – Allows peak performance
        • “middle packet” optimization
  • Current plan: support all modes
MPI in BG/L: Using the 2\textsuperscript{nd} CPU

- **Processing modes:**
  - heater mode
  - symmetric
    - 1 MPI rank per ASIC
    - *communication co-processor*

- **Compute processor:**
  - post, allocate, match graduate MPI requests
  - progress at channel protocol level

- **Comm. processor:**
  - progress at transport level
    - packets
    - messages

---

**Ground Rule #1:**

MPI primitives are executed by compute processor
Communication co-processor

- co-processor looks like a **big** virtual torus device
  - high performance
  - no coherency problem
  - compatible
  - perfect for a first cut
  - latency
  - can do better
“Middle packet” optimization

- aligned packets of matched/allocated requests
- coprocessor streams to/from request buffers
  + truly 0-copy
  + good latency
  + true comm. overlap
- needs co-ordination
- fragile
- only for aligned packets
Status of BG/L MPI Implementation Today

- Running (!) in
  - Emulation
    - native Linux/IA32
    - 2:1 slowdown
  - Simulation
    - Linux/bglsim
    - $\sim 10^3$ slowdown

- Message Layer supports CH3 “eager protocol”
  - Does not yet provide correct inter-message ordering
  - Does not implement optimistic error control
  - Does not yet have specialized collective operations
  - MPICH-2 does not yet have all of MPI, but:

- NAS parallel benchmarks
  - experiments on 2 to 4 processors
Process Manager Research Issues

• Identification of proper process manager functions
  • Starting (with arguments and environment), terminating, signaling, handling stdio, …
• Interface between process manager and communication library
  • Process placement and rank assignment
  • Dynamic connection establishment
  • MPI-2 functionality: Spawn, Connect, Accept, Singleton Init
• Interface between process manager and rest of system software
  • Cannot be separated from system software architecture in general
  • Process manager is important component of component-based architecture for system software, communicating with multiple other components
• Scalability
  • A problem even on existing large systems
  • BG/L presents new challenges
Process Manager Research at ANL

- MPD – prototype process management system
- Original Motivation: faster startup of interactive MPICH programs
- Evolved to explore general process management issues, especially in the area of communication between process manager and parallel library
- Laid foundation for scalable system software research in general
- MPD-1 is part of current MPICH distribution
  - Much faster than earlier schemes
  - Manages stdio scalably
  - Tool-friendly (e.g. supports TotalView)
Requirements on Process Manager from Message-Passing Library

- Individual process requirements
  - Same as for sequential job
  - To be brought into existence
  - To receive command-line arguments
  - To be able to access environment variables

- Requirements derived from being part of a parallel job
  - Find size of job: MPI_Comm_size( MPI_COMM_WORLD, &size )
  - Identify self: MPI_Comm_rank( MPI_COMM_WORLD, &myrank )
  - Find out how to contact other processes: MPI_Send( ... )
Finding the Other Processes

- Need to identify one or several ways of making contact
  - Shared memory (queue pointer)
  - TCP (host and port for connect)
  - Other network addressing mechanisms (Infiniband)
  - (x,y,z) torus coordinates in BG/L
- Depends on target process
- Only process manager knows where other processes are
- Even process manager might not know everything necessary (e.g. dynamically obtained port)
- “Business Card” approach
Approach

- Define interface from parallel library (or application) to process manager
  - Allows multiple implementations
  - MPD is a scalable implementation (used in MPICH ch_p4mpd device)
- PMI (Process Manager Interface)
  - Conceptually: access to spaces of key=value pairs
  - No reserved keys
  - Allows very general use
  - Basic part: for MPI-1, other simple message-passing libraries
  - Advanced part: multiple keyval spaces for MPI-2 functionality, grid software
- Provide scalable PMI implementation with fast process startup
- Let others do so too
The PMI Interface

- PMI_Init
- PMI_Get_size
- PMI_Get_rank
- PMI_Put
- PMI_Get
- PMI_Fence
- PMI_End

- More functions for managing multiple keyval spaces
  - Needed to support MPI-2, grid applications
MPD

Architecture of MPD:

Scheduler

mpirun

mpd’s managers application processes
Interesting Features

- **Security**
  - “Challenge-response” system, using passwords in protected files and encryption of random numbers
  - Speed not important since daemon startup is separate from job startup
- **Fault Tolerance**
  - When a daemon dies, this is detected and the ring is reknit => minimal fault tolerance
  - New daemon can be inserted in ring
- **Signals**
  - Signals can be delivered to clients by their managers
More Interesting Features

- **Uses of signal delivery**
  - signals delivered to a job-starting console process are propagated to the clients
    - so can suspend, resume, or kill an mpirun
  - one client can signal another
    - can be used in setting up connections dynamically
  - a separate console process can signal currently running jobs
    - can be used to implement a primitive gang scheduler

- **Mpirun also represents parallel job in other ways**
  totalview mpirun –np 32 a.out
  runs 32-process job under TotalView control
More Interesting Features

• Support for parallel libraries
  • implements the PMI process manager interface, used by MPICH.
    • groups, put, get, fence, spawn
    • simple distributed database maintained in the managers
    • solves “pre-communication” problem of startup
    • makes MPD independent from MPICH while still providing needed features
Handling Standard I/O

- Managers capture `stdout` and `stderr` (separately) from their clients.
- Managers forward `stdout` and `stderr` (separately) up a pair of binary trees to the console, optionally adding a rank identifier as line label.
- Console’s `stdin` is delivered to `stdin` of client 0 by default, but can be controlled to broadcast or go to specific client.
The Scalable Systems Software SciDAC Project

- Multiple Institutions (most national labs, plus NCSA)
- Targeting systems software for large systems, particularly clusters
- Component architecture
- Currently using XML for inter-component communication
- Status
  - Early demos; watch for more at SC’02, some components in use at Argonne on Chiba City cluster
  - Detailed XML interface to PM component, implemented by MPD
- One powerful effect: forcing rigorous (and aggressive) definition of what a process manager should do and what should be encapsulated in other components
  - Start (with arguments and environment variables), terminate, cleanup
  - Signal delivery
  - Interactive support (e.g. for debugging) – requires stdio management
What Does This Have to Do with MPI on BGL?

- MPI library needs PMI interface implementation
- LoadLeveler desirable as scheduler
  - It exists!
  - Provides sophisticated scheduling capabilities
  - Familiar to large class of users
- LoadLeveler can be used as scheduling component in Scalable System Software Center sense
  - Interface to process manager well defined
  - Interface has needed features
  - MPD-based process manager ready for use
  - Currently collaborating with IBM/Haifa group on this approach to scheduling and process management for BG/L
- LoadLeveler only one option for scheduling component
  - Clear definitions of interfaces will support use of other schedulers
    - (e.g., SLURM)
MPD Supports Multiple Styles of Process Management

• Scheduler can compose and execute mpirun command that communicates with MPD ring
  • Easy to write BG/L-specific mpirun scripts
    • (e.g. to specify topology information)
• Scheduler can communicate directly with mpd ring
• Scheduler, other components of system software can communicate with persistent process manager component, using public XML interface
• Scheduler can allocate nodes for interactive use and user can run mpirun interactively
  • (e.g. for debugging)
• User can set up own MPD ring in user mode
  • (e.g. for development)
LoadLeveler and MPD for BG/L

- Goals
  - Provide functional and familiar job submission, scheduling, and process management environment on BG/L
  - Change existing code base (LL, MPICH, MPD) as little as possible
- Current Plan: Run MPD’s as root and have LL submit job to MPD’s to start user job as user
- LL can schedule set of nodes for user to use interactively; then user can use mpirun to run series of short interactive jobs on subsets of allocated nodes
  - Ensure that user can only use scheduled nodes
- Build foundation for development of other scheduling and process management approaches
BG/L Architecture

- Example: 2 I/O nodes, each with 64 compute nodes
Proxy processes

• A proxy process (Linux process) is created for each MPI task
• The task is not visible to the operating-system scheduler
• The proxy interfaces between the operating-system and the task, passing signals, messages etc…
• It provides transparent communication with the MPI task
• MPD will start these proxy processes
  • Need to be able to pass separate arguments to each
Running the Proxies on the Linux Nodes

LL daemon

mpd

mpd

mpdrun

mpdman

mpdman

mpdman

mpdman

mpdman

mpdman

proxy

proxy

proxy

proxy

proxy

proxy

proxy

proxy

proxy

proxy

proxy

proxy

Proxy cpi 5

Proxy cpi 6

Proxies still under discussion

Run as root

Run as user
Conclusion

- IBM and ANL are collaborating in two related areas to improve the usability of BG/L
  - MPI implementation
  - Process management
- In each case timing seemed to be perfect to connect existing research projects to new scalability challenges
- Early results are promising