MPI for the Next Generation of Supercomputing

Andrew Lumsdaine
Open Systems Laboratory
Indiana University
lums@cs.indiana.edu
Message Passing

- “We are the first generation of message passing programmers”

- “Hopefully we are the last generation of message passing programmers”
Message Passing and MPI

The rumors are greatly exaggerated.
Outline

- Background
- Design and functionality of MPI
- What is good about MPI
- Challenges
  - Next generation software
  - Next generation hardware
- Evolution of MPI
- Conclusion
Parallel Computing

- Multiple processes that work together to solve large problems (fast)
- Parallelization process
- Programming model
  - Describes logical relationship of CPUs, memory, network
- Hardware model
  - Describes physical relationship of CPUs, memory, network
- Logical and physical relationships can be independent
Parallelization Process

Sequential Computation → Tasks → Processes → Parallel Program → Processors

Decomposition → Assignment → Orchestration → Mapping
Hardware Models

- Single processor model
- Shared memory
- Distributed memory
- *All memory is distributed*
Programming Models

- A programmer-oriented taxonomy
  - Data-parallel: Same operations on different data
    (High-Performance Fortran)
  - Task-parallel: Different operations on different data
  - SPMD: Single program, multiple data (MPI)

- SPMD and task-parallel are essentially equivalent
  - Any task-parallel program can be expressed as SPMD
Enter MPI

- Message passing well understood as parallel programming paradigm
- Early vendor systems (NX, EUI, CMMD) were not portable
- Early portable systems (PVM, p4, TCGMSG, Chameleon) were mainly research efforts
  - Did not address the full spectrum of message-passing issues
  - Lacked vendor support
  - Were not implemented in the most efficient way
Enter MPI

- The MPI Forum organized in 1992 with broad participation by vendors, library writers, and end users
- MPI Standard (1.0) released June, 1994; many implementation efforts (LAM/MPI, MPICH, vendors)
- MPI-2 Standard (1.2 and 2.0) released July, 1997
- MPI forum recently reconvened (MPI 2.1, 2.2, and 3.0)
What is MPI?

- A message-passing library specification
  - Message-passing model
  - Not a compiler specification
  - Not a specific product
- Interfaces for C, C++, and Fortran 77
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Two parts: MPI-1 (1.2) and MPI-2 (2.0)
MPI Features

- **General**
  - Communicators combine context and group for message security
  - Thread safety

- **Point-to-point communication**
  - Structured buffers and derived datatypes, heterogeneity
  - Modes: normal (blocking and non-blocking), synchronous, ready (to allow access to fast protocols on some systems), buffered

- **Collective**
  - Both built-in and user-defined collective operations
  - Large number of data movement routines
  - Subgroups defined directly or by topology
MPI Features (cont’d)

- **Dynamic process control**
  - Allows creation and cooperative termination of processes after an MPI application has started.
  - Mechanism to establish communication between existing MPI applications, which may have been started separately.

- **One-sided operations**
  - Remote Memory Access (RMA) communication mechanisms
  - Communication: Put (remote write), Get (remote read) and Accumulate (remote update)
  - Support for both active and passive target synchronization.
MPI Features (cont’d)

- Parallel I/O
  - Portable interface for optimized parallel file access
  - Support for synchronous and asynchronous I/O
  - Allows for close coupling with parallel filesystems
  - Data partitioning expressed using derived datatypes.
MPI Features (cont’d)

- Application-oriented process topologies
  - Built-in support for grids and graphs (based on groups)
- Profiling
  - Hooks allow users to intercept MPI calls to install their own tools
- Environmental
  - Inquiry
  - Error control
Features Not in MPI

- Non-message-passing concepts not included:
  - Remote memory transfers
  - Active messages
  - Threads
  - Virtual shared memory
  - Fault tolerance

- MPI does not address these issues, but has tried to remain compatible with these ideas (e.g. thread safety as a goal, etc.)
Is MPI Large or Small?

- MPI is large (MPI-1 has 128 functions, MPI-2 adds 152 functions)
  - MPI’s extensive functionality requires many functions
  - Number of functions not necessarily a measure of complexity
- MPI is small (6 functions)
  - Many parallel programs can be written with just 6 basic functions
- MPI is just right
  - One can access flexibility when it is required
  - One need not master all parts of MPI to use it
MPI is the Worst Way to Program

Except for all the others!
Why has MPI Succeeded?

- As a paradigm
  - Powerful organizing principle for parallelization
- As a library / interface
  - Portability
  - Performance
  - Simplicity
  - Symmetry
  - Modularity
  - Composability
  - Completeness
Prospects for a Discipline

- Advance what works

Why MPI Succeeded

- Distributed Memory Hardware
- New Problems
- Ad-Hoc Solutions
- Folklore
- Models/Theory
- Codification
- Improved Practice
- “Legacy MPI codes”

MPI

NX
Shmem
P4, PVM
Sockets

Message Passing Rules!
Future of Parallel Programming

Progress, far from consisting in change, depends on retentiveness. Those who cannot remember the past are condemned to repeat it.
Future of Parallel Programming

It will be called MPI

New Problems

Ad-Hoc Solutions

Folklore

Models/ Theory

Codification

Improved Practice
One Challenge

Not the only challenge!
Ubiquitous Parallelism

- Emphasizes challenges
- Programmability
- Scalability
- Heterogeneity
- Fault tolerance
Developer Concerns

- Productivity
- Composability
- Reusability
- Maintainability
- Reliability
- Performance
Mainstream Ecosystem

- Modern technologies developed to address these goals
  - Paradigms (OO, generic programming)
  - Languages (C++, Java, C#, Python, …)
  - Tools (Visual Studio, Eclipse, Xcode)
  - Architectures (client/server, multitier, GUI)
  - Libraries (STL, MFC, libc, …)
- To be relevant, parallelism must fit into the mainstream ecosystem
- Mixed results with MPI
Boost.MPI: High-Level C++ MPI

- Modernized C++ interface to MPI
  - Seamlessly supports user-defined types
  - Auto-serialization via Boost.Serialization library for objects and containers
  - Collectives operate on user-defined types, function objects
  - Integration with C++ Standard Template Library, third party libraries
  - Far easier to use than existing C++ bindings

- Benchmarks and performance:
  - Zero abstraction penalty relative to C

http://www.boost.org
MPI.NET: Message Passing in C#

- High-performance MPI library for C#
  - Supports all .NET languages (C#, C++, F#, ..., even Visual Basic!)

- Natural expression of MPI in C#
  - Retains familiar MPI concepts
  - Provides natural C# interfaces
    - Seamless support for user-defined types
    - Eliminates pitfalls of low-level languages
  - Easier to use than C, C++, or Fortran bindings

- High productivity, high performance

http://www.osl.iu.edu/research/mpi.net
MPI.NET: Natural C# Interfaces

- Automatic serialization of objects as needed

```csharp
if (world.Rank == 0)
    world.Send("Hello, World!", 1, 0);
else
    string msg = world.Receive<string>(0, 0);
```

- Collectives extended to user-defined types, lambdas and methods

```csharp
string[] hostnames = comm.Gather(MPI.Environment.ProcessorName, 0);

double pi
    = 4.0*comm.Reduce(dartsInCircle,(x, y) => return x + y, 0) / totalDartsThrown;
```

http://www.osl.iu.edu/research/mpi.net
MPI.NET: Performance

- Applies aggressive run-time optimizations
  - Maps high-level operations to their most specific native MPI operation
  - Never serializes primitive types or structures (constructs derived datatypes when needed)
- NetPIPE micro-benchmark performance:
  - Negligible overhead (relative to C) over TCP
  - 7-12% slower than C in shared memory

http://www.osl.iu.edu/research/mpi.net
Scalability

- MPI is sometimes called the assembly language of parallel programming
- Local descriptions of data exchanges lead to complex, implicitly defined, interactions
- Collective operations are an important higher-level mechanism
- Limited (somewhat) in MPI-1 and MPI-2
Non-Blocking Collectives

- Delivers real performance increases
- Proposed for inclusion in the MPI standard
- Currently available in LibNBC
  - Portable implementation using MPI 1

http://www.unixer.de/research/nbcoll/libnbc
Checkpoint/Restart in Open MPI

- C/R Coordination Protocol
  - **OMPI CRCP**
- Snapshot Coordination
  - **ORTE SnapC**
- Remote File Management
  - **ORTE FileM**
- Checkpoint/Restart Service
  - **OPAL CRS**
- Internal Synchronization
  - **Interlayer Notification Callbacks**

Functioning in Open MPI svn, included in Open MPI 1.3
Open Research Issues

- Interactions with threads
  - Programming model
  - Hardware model
- Tool support
- Language support
Conclusion

☐ Not dead yet
Checkpoint/Restart in Open MPI

- C/R Coordination Protocol
  - OMPI CRCP
- Snapshot Coordination
  - ORTE SnapC
- Remote File Management
  - ORTE FileM
- Checkpoint/Restart Service
  - OPAL CRS
- Internal Synchronization
  - Interlayer Notification Callbacks

Functioning in Open MPI svn, included in Open MPI 1.3