The LAM Implementation of MPI: Features, Dynamic Process Control, and Checkpointing

Jeffrey M. Squyres, Andrew Lumsdaine
Department of Computer Science and Engineering
University of Notre Dame
Overview

- Introduction: Parallel Computing
- MPI (and others)
- The LAM implementation of MPI
- LAM + Condor = Lamdor
- Conclusions / Future Work
Introduction: Parallel Computing

- Shared memory
  - Typically multi-threaded, sometimes multi-process
  - All sharing common memory
  - Not [directly] the focus of this talk
Introduction: Parallel Computing

- Distributed memory
  - Typically separate processes
  - Explicit sending of messages; little [direct] use of shared memory
Message Passing: The Contenders

- Parallel Virtual Machine (PVM)
  - Research project at Oak Ridge National Labs
  - First message passing package on clusters
  - Attracted a *large* user base

- Message Passing Interface (MPI)
  - MPI-1 standardized in 1994, MPI-2 standardized in 1997
  - Vendor and open source implementations
  - Source code portable
PVM: The Good

- Daemon-based run-time environment
- Popularized manager / worker model using dynamic processes
- Load factoring / environment querying
- Inter-implementation communication
- Still has a *large* user following
PVM: The Bad

- Usually forces an extra buffer copy
- No true asynchronous communication
- Nondeterministic behavior (particularly with groups)
- Weak message safety (only one context at a time)
- Losing vendor support
What is MPI?

- A specification for a message passing API
- Two documents:
  - MPI-1: Basic message passing (send, receive, collectives, etc.)
  - MPI-2: Extensions to MPI-1 (one-sided, C++, dynamic processes, etc.)
- Specifically written to enable high performance
- Designed for clusters all the way up to “Big Iron”
MPI: The Good

- Learned from previous designs: NX, Zipcode, PVM, etc.
- No extra memory copy
- Capable of true asynchronous communication
- Deterministic behavior, ease of discovering identity
- Strong message safety
- Strong (and continuing) vendor support
- Uses fastest message passing available (shmeme, TCP, etc.)
MPI: The Bad

- Some of the previous is “implementation dependent”
  - True asynchronous communication
  - Use of fastest message passing channel
- No [portable] fault tolerance
- MPI’s design does not preclude any of these, but much of this is [intentionally] left unspecified
MPI Terminology

- **Rank**: A single entity in a parallel job
- **Communicator**: A group of ranks plus a unique message passing context
- **MPI_COMM_WORLD**: Default communicator that contains all ranks
MPI Code Example: Hello World

```c
int main(int argc, char* argv[]) {
    int me, total;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
    MPI_Comm_size(MPI_COMM_WORLD, &total);

    printf("Hello world: %d of %d\n", me, total);
    MPI_Finalize();
    return 0;
}
```
Overview

- Introduction: Parallel Computing
- MPI (and others)
- The LAM implementation of MPI
- LAM + Condor = Lamdor
- Conclusions / Future Work
The LAM Implementation of MPI

- Originally written at the Ohio Supercomputing Center
  - Targeted at transputers
  - Top MPI layer was added later
  - MPI has since become the main focus of work

- Everyone graduated, moved on
  - LAM/MPI was orphaned for about a year

- Project moved to Notre Dame in 1998
MPI Conformance

- Full MPI-1 conformance
- Much MPI-2 functionality
  - MPI I/O (ROMIO)
  - MPI C++ bindings
  - One-sided communication
  - Dynamic process control
- Interoperable MPI (IMPI)
  - Point-to-point and some collectives
LAM/MPI: Features

● Daemon-based run-time environment
  – Fast startup of user programs
  – Guaranteed cleanup of user programs
  – External monitoring

● Flexible mpirun
  – SMP-friendly syntax
  – SPMD or MPMD
  – Can distribute executables; no global filesystem required
  – Pseudo-tty support
  – Environment variable and working directory export
LAM/MPI: More Features

- Optimized point-to-point message passing
  - “Short-circuit” optimization
  - Short call stack; “uncomplicated” engine
  - Combined TCP / shared memory message passing
  - True asynchronous message passing

- Monitoring tools
  - XMPI: GUI message passing patterns
  - mpimsg: pending messages
  - mpitask: running LAM tasks
LAM/MPI: Still More Features

- Heterogeneous support
  - Portable to most POSIX systems
  - On-the-fly endian conversion (if necessary)

- Debugging support
  - Ability to **mpirun** non-MPI executables (e.g., debuggers)
  - Totalview support on the way
  - Purify clean
  - Open source (some users actually *do* source dive!)
  - Lots of online help: web pages, man pages
Daemon-Based Run-Time Environment

- User level, not root level
- Launch the LAM RTE: lamboot <hostfile>
Running MPI Programs

- `mpirun` sends a message to the local daemon

```
mpirun -np 4 foo
```
Running MPI Programs: Step 2

- The daemons **fork** / **exec** the child, setup stdin / stdout, etc.
During `MPI_Init()`, each MPI rank connects all others.
**MPI_Init: Mutual Awareness**

- Out-of-band messaging is used during `MPI_Init()`
  - Each rank contacts `mpirun`
  - `mpirun` sends full list of out-of-band peer addresses

```bash
mpirun -np 4 foo
```

**Diagram:**
1. `address_0` connects to `foo`
2. `address_1` connects to `foo`
3. `address_0` connects to `address_1`
4. `address_1` connects to `address_0`
MPI_Init: Peer-to-Peer Setup

- More out-of-band messaging used between MPI ranks
  - Each rank opens a dynamic “listening” socket
  - Pairwise, “acceptor” rank sends port number to “listener”
MPI Dynamic Processes

- **MPI_Comm_spawn()** is used to launch a group of children processes
  - It is a collective (blocking) call across the spawning communicator

- Children processes will have their own unique **MPI_COMM_WORLD**

- Parents and children will share a “bridge” communicator that they can communicate with
**MPI_Comm_spawn: Same as MPI_Init**

- Replace `mpirun` instance with `MPI_Comm_spawn()` instance
- Do same out-of-band messaging
MPI_Comm_spawn: Slight Differences

- Children must create parent communicator
- `MPI_Comm_spawn()` must also send parent addresses
MPI Bandwidth Performance: TCP

- MPI performance on the Coral cluster
- ICASE / NASA Langley Research
- 32 nodes
  - Phase 2: 16 dual pentium III/500, PC100
  - Phase 3: 16 dual pentium III/800, PC133
- Gigabit ethernet interconnection network
MPI Bandwidth Performance: TCP

- MPICH latency over TCP: 175us
- LAM latency over TCP: 129us
MPI Bandwidth Performance: Myrinet

- Linux pentium cluster at Indiana University
- Myrinet interconnection network
- Numbers collected using NetPIPE 2.4 benchmark application
MPI Bandwidth Performance: Myrinet
MPI Latency Performance: Myrinet

NetPIPE 2.4 Latency Performance

- MPICH-gm
- LAM/gm

Message size (bytes)

Latency (sec.)
Overview

- Introduction: Parallel Computing
- MPI (and others)
- The LAM implementation of MPI
- LAM + Condor = Lamdor
- Conclusions / Future Work
LAM + Condor = Lamdor

- We want to run LAM jobs under Condor
  - Get MPI-2 features in Condor (spawn, put / get, etc.)
  - Take MPI support burden away from Condor team

- Bring MPI into distributed [dynamic] computing
  - Manager / worker process model
  - Some degree of fault tolerance

- Eventual goal: cycle-scavenging parallel jobs
  - Transparent checkpoint/migrate/restart support for MPI jobs
Applies Outside of Condor Environments

- Clusters aren’t as stable as Beowulfers would have you believe
  - Nodes die, switches and routers fail, etc.
  - That is: even dedicated clusters are dynamic clusters

- Scale LAM up to grid-sized problems: dynamic MPI environments

- Use the standalone checkpoint library outside of a Condor flock
  - Resource maintenance (e.g., Beowulf-style clusters)
  - Queue management / dedicated resource time
  - Program fault protection
Overview

- Introduction: Parallel Computing
- MPI (and others)
- The LAM implementation of MPI
- LAM + Condor = Lamdor
- Conclusions / Future Work
Conclusions / Future Work

- Get LAM/MPI to run under the Condor static scheduler
- Extend LAM/MPI to dynamic and unreliable environments
  - Properly define (redefine?) MPI semantics / models for dynamic environments
- Extend LAM/MPI to handle grid-sized problems
- Experiment with checkpoint / migrate / restart schemes