Open MPI\textsuperscript{(c)}
is now 14 months old …
Current status and more

George Bosilca
<bosilca@cs.utk.edu></b>
MPI From Scratch: Why?

• Merger of ideas from
  – FT-MPI (U. of Tennessee)
  – LA-MPI (Los Alamos, Sandia)
  – LAM/MPI (Indiana U.)
  – PACX-MPI (HLRS, U. Stuttgart)
Open MPI Project Goals

• All of MPI (i.e., MPI-1 and 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 effort
Design Goals

• Extend / enhance previous ideas
  – Message fragmentation / reassembly
  – Design for heterogeneous environments
    • Multiple networks
    • Node architecture (data type representation)
  – Automatic error detection / retransmission
  – Process fault tolerance
Components

• Formalized interfaces
  – Specifies “black box” implementation
  – Different implementations available at run-time
  – Can compose different systems on the fly
Components

• Formalized interfaces
  – Specifies “black box” implementation
  – Different implementations available at run-time
  – Can compose different systems on the fly
Cost of the component
Open MPI’s Point-To-Point Implementation
User application

MPI API

MPI Component Architecture (MCA)

PML

PTL

TCP/IP

Shared Mem

IB

Memory Pooling

Memory Management

Proc Private

Shared

Pinned

Pow 2 binning

Best fit
Pt-2-Pt Components

• PML
  – MPI aware
  – Fragments
  – May request Acks
  – Reassembles
  – Data Fault tolerance

• PTL
  – Data mover
  – Message Matching
  – Responsible for own progress (polling or async)
What About Performance?

• Pack / unpack approach

• 3 distinct steps: pack, network xfer, unpack

• No computation / communication overlap

• How to increase the performance?
Improving Performance

• Pipeline
  – Create computation / communication overlap
  – Split the computations in small slices
Improving Performance

• Other 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?

• Who handles all this?
  – MPI implementation can solve these problems
  – User-level applications cannot
Striped Message Example

Send Side

PML : TEG
First Fragment

IB - 1
IB - 2
GM

Receive Side

PML : TEG
First Fragment

IB-1

PML : TEG
Last Frags

IB - 1
IB - 2
GM

40 %

10 %

a b c
MPI Matching semantics

- **Sender specifies**
  - MPI communicator
  - Source rank in communicator (itself)
  - Destination rank in communicator
  - User defined tag

- **Receiver specifies**
  - MPI communicator
  - Source rank in communicator (*)
  - Destination rank in communicator (itself)
  - User defined tag (*)

**Match Made**

Data from Source
May be delivered to the Destination
Receiver Side Matching

Senders

A
A3
A2
A1

B
B3
B2
B1

C
C3
C2
C1

Receiver:
Matching done on receive side
Multiple NIC’s - Matching

Same:
- Communicator
- Tag

Requirement: M1 matched before M2
Multiple NIC’s - Matching

• Requirement:
  One set of “counters”
  For matching (atomic Access)

• Main memory vs.
  NIC matching
Multiple NIC’s - Striping

- **No** ordering requirement
- Need to identify send/recv completion
Data transfer - Point-To-Point

• Use the most efficient method to send data
  – RDMA is good - minimize main host CPU involvement

• Send header and payload from separate buffers

• If appropriate, order header and buffer sends
  – Header should refer to delivered payload
Message Matching Timing

Send

Match - I

Match - II

Receive - I

Receive - II
Message Matching Timing

• Resource Utilization
  – Temporary buffers
    • Rendezvous Protocol vs. Eager Send
    • “User buffers” vs. Library buffers
    • Reduced latency vs. Reduced resource usage

• Latency considerations are critical for HPC
  – End-to-End (main-memory to main-memory)
    target under 1 uSec - worst case
Rendezvous

Send

First Fragment - Request To Send

Match - II

Receive - II

Ack - Clear To Send

Rest of Send
Get type network

First Fragment - Request To Send

Match - II

Receive - II

Get the rest of the data

Data-Type considerations

Send
Receive Side Notification

- **Polling**
  - “Low” latency
  - High CPU use

- **Event Driven/Asynchronous**
  - “High” latency
  - Low CPU use
  - Good for Compute/Communication overlap
  - Require a specific thread
Alpha stage

Please enjoy yourself using Ompi on the sinrg clusters

Add to your PATH ~bosilca/opt/mpi/bin
Add to your LD_LIBRARY_PATH ~bosilca/opt/mpi/lib
Status

• Test suits
  – IBM  95%
  – Intel  95%
  – HPL 100%
  – Scalapack 60%
  – Opem MPI 100% (!!!)
## PTL devices status

<table>
<thead>
<tr>
<th>PTL/Device</th>
<th>Contiguous data</th>
<th>Heterogeneous &amp; Non contiguous</th>
<th>Threads &amp; MPI-2 dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self (ICL)</td>
<td>100%</td>
<td>100%</td>
<td>MT, yes</td>
</tr>
<tr>
<td>Shared Memory (LANL)</td>
<td>100%</td>
<td></td>
<td>MT, yes</td>
</tr>
<tr>
<td>TCP (LANL &amp; ICL)</td>
<td>100%</td>
<td>100%</td>
<td>MT, yes</td>
</tr>
<tr>
<td>MX (LANL)</td>
<td>100%</td>
<td>100%</td>
<td>MT, yes</td>
</tr>
<tr>
<td>GM (ICL)</td>
<td>100%</td>
<td>100%</td>
<td>UT, yes</td>
</tr>
<tr>
<td>IB (?)</td>
<td>50%</td>
<td>?%</td>
<td>UT, ?</td>
</tr>
<tr>
<td>Profiling (ICL)</td>
<td>50%</td>
<td></td>
<td>MT, yes</td>
</tr>
<tr>
<td>ELAN (LANL &amp; ICL)</td>
<td>80%</td>
<td>?%</td>
<td>UT, ?</td>
</tr>
</tbody>
</table>
### TCP/IP Latency Data (non-blocking)

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Myrinet (μs)</th>
<th>GigeE (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open MPI (poll)</td>
<td>51.5</td>
<td>39.7</td>
</tr>
<tr>
<td>Open MPI (Asn)</td>
<td>51.2</td>
<td>49.9</td>
</tr>
<tr>
<td>LAM/MPI 7</td>
<td>51.5</td>
<td>39.9</td>
</tr>
<tr>
<td>LA-MPI</td>
<td>51.6</td>
<td>42.9</td>
</tr>
<tr>
<td>FT-MPI</td>
<td>51.4</td>
<td>46.4</td>
</tr>
<tr>
<td>MPICH2</td>
<td>51.5</td>
<td>40.3</td>
</tr>
</tbody>
</table>
Bandwidths - 1 NIC GigE (Mb/sec)
Bandwidths - 1 NIC Myrinet (TCP/IP - Mb/sec)
Bandwidth - 2 NIC Myrinet (TCP/IP - Mb/sec)
Bandwidth - Myrinet + GigE (TCP/IP - Gb/sec)
Elan4: 0 Byte Latency - 3.47 uSec