MPICH/UCX status update

Pavan Balaji

Computer Scientist and Group Leader

Argonne National Laboratory
The MPICH Project

- MPICH and its derivatives are the world’s most widely used MPI implementations
  - Supports all versions of the MPI standard including the recent MPI-3
- Funded by DOE for 23 years (turned 23 this month)
- Has been a key influencer in the adoption of MPI
- Award winning project
  - DOE R&D100 award in 2005

MPICH and its derivatives in the Top 10

1. Tianhe-2 (China): TH-MPI
2. Titan (US): Cray MPI
3. Sequoia (US): IBM PE MPI
4. K Computer (Japan): Fujitsu MPI
5. Mira (US): IBM PE MPI
6. Trinity (US): Cray MPI
7. Piz Daint (Germany): Cray MPI
8. Hazel Hen (Germany): Cray MPI
9. Shaheen II (Saudi Arabia): Cray MPI
10. Stampede (US): Intel MPI and MVAPICH

MPICH and its derivatives power 9 of the top 10 supercomputers (Nov. 2015 Top500 rankings)
**MPICH layered structure**

**Platform independent code**
- Collectives
- Communicator management

**“Device” layer:** Machine-specific functionality
- Derived datatype management
- RMA

**“Channel” layer**
- Some “core” functionality, such as shared memory

**“Netmod” layer**
- Implements some mandatory functionality
- Can override any amount of optional functionality (better bcast, better barrier)

- CH3
- **MPI Layer**
- **Platform independent code**
  - Collectives
  - Communicator management
- **“Device” layer:** Machine-specific functionality
  - Derived datatype management
  - RMA
- **“Channel” layer**
  - Some “core” functionality, such as shared memory
- **“Netmod” layer**
  - Implements some mandatory functionality
  - Can override any amount of optional functionality (better bcast, better barrier)
- tcp, mxm, ofi, ugni, ...
- CH3, psmipi, pamid, ...
- “Netmod” layer
  - Implements some mandatory functionality
  - Can override any amount of optional functionality (better bcast, better barrier)
- Platform independent code
  - Collectives
  - Communicator management
- “Device” layer: Machine-specific functionality
  - Derived datatype management
  - RMA
- “Channel” layer
  - Some “core” functionality, such as shared memory
- “Netmod” layer
  - Implements some mandatory functionality
  - Can override any amount of optional functionality (better bcast, better barrier)
MPICH/CH4

- New Effort for low-overhead communication
- Directed towards network APIs with high-level semantics
  - E.g., UCP
- Collaborative effort between multiple institutes
  - Argonne, Mellanox, Intel, RIKEN, ...
**CH4 Design Goals**

### High-Level Netmod API
- Give more control to the network
  - `netmod_isend`
  - `netmod_irecv`
  - `netmod_put`
  - `netmod_get`
- Fallback to Active Message based communication when necessary
  - Operations not supported by the network

### Provide default shared memory implementation in CH4
- Disable when desirable
  - Eliminate branch in the critical path
  - Enable better tuned shared memory implementations
  - Collective offload

### “Netmod Direct”
- Support two modes
  - Multiple netmods
    - Retains function pointer for flexibility
  - Single netmod with inlining into device layer
    - No function pointer

#### MPI

#### CH4

#### Netmod

#### OFI

#### UCX

#### Portals 4

### No Device Virtual Connections
- Global address table
  - Contains all process addresses
  - Index into global table by translating `(rank+comm)`
- VCs can still be defined at the lower layers
UCX support for MPICH 3.3

- CH4-netmod API
- Uses UCP implementation
- Isend/Irecv are directly implemented using tag-matching of UCP
## Measurement Setup

- Mellanox Connect-IB EDR Infiniband
- osu-microbenchmarks
- OFED-3.1-1.0.5
- Set UCX_BCOPY_TRSHOLD to 0 (get better performance)
- Use different parameters to get best performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>UCX</th>
<th>MXM</th>
</tr>
</thead>
<tbody>
<tr>
<td>osu_mbw_mr</td>
<td>-w 1024</td>
<td>-w 64</td>
</tr>
<tr>
<td>osu_latency</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>osu_bw</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Preliminary Improvements to MXM

Message Rate / MMPS

MPICH/UCX
MPICH/MXM

UCX BoF (11/17/2015)
Preliminary Improvements to MXM

Latency (us) for small messages:
- MPICH/UCX
- MPICH/MXM

Latency (us) for large messages:
- MPICH/UCX
- MPICH/MXM

UCX BoF (11/17/2015)
Preliminary Improvements to MXM

Bandwidth

MPICH/UCX
MPICH/MXM

UCX BoF (11/17/2015)
Instruction Count

- 25% less instructions for UCX compared to MXM

- Intel compiler
- Dynamic linking
- Not counting in: Instructions inside UCX/MXM lib

ch3/mxm

ch4/ucx