Presented in conjunction with the EasyBuild community
Webex Logistics

- This session is being recorded
- Ask questions in the Q&A panel
Overview

● Background
● PMIx: What is it?
● Building Open MPI
● A breakdown of Open MPI:
  ○ The run-time stuff
  ○ The MPI stuff
● Configuration / debugging tips
● The upcoming Open MPI v4.1.x series
● The upcoming Open MPI v5.0.x series

We’ll get as far as we get today

Next session: Wednesday, July 8
Background
Open MPI Overall Architecture Terminology

- Modular Component Architecture (MCA)
  - Semantic architecture of the Open MPI software package
  - Hierarchy: Project → Framework → Component
Open MPI Overall Architecture Terminology
Open MPI Framework + Component Examples

(NOTE: not a comprehensive list of all projects, frameworks, and components)
Names of Frameworks and Components

- The Open MPI community has proven to be terrible at naming things

- There are several frameworks and components with Star Wars-inspired names (i.e., that have nothing to do with their functionality)
  - Most famous: “vader” = shared memory message transport
  - A few non-Star-Wars science fiction names, too (e.g., Star Trek, Highlander)
PMLx: What Is It?
Origin: Changing Landscape

Launch time limiting scale

Programming model & runtime proliferation

Legion

Hybrid applications

Model-specific tools

Container technologies
Start Someplace!

- Resolve launch scaling
  - Pre-load information known to RM/scheduler
  - Pre-assign communication endpoints
  - Eliminate data exchange during init
  - Orchestrate launch procedure
What is PMIx?

PMI-1 $\leftrightarrow$ PMI-2

RM SLURM ALPS

MPICH

wireup support
dynamic spawn
keyval publish/lookup

years go by...

Exascale systems on horizon
Launch times long
New paradigms

PMIx $\times 10^{18}$

PMIx v1.2

OMPI Spectrum OSHMEM

Exascale launch in < 30s

OMPI MPICH Spectrum OSHMEM SOS PGAS TV/DDT ...

Exascale launch in < 10s
Workflow orchestration

2015

2016

2020

RM All but UGE

PMIx v4
Three Distinct Entities

- **PMIx Standard**
  - Defined set of APIs, attribute strings
  - Nothing about implementation

- **OpenPMIx Library**
  - Full-featured implementation of the Standard
  - Intended to ease adoption

- **PMIx Reference RTE (PRRTE)**
  - Full-featured “shim” to a non-PMIx RM
  - Provides development environment

v4.0 soon!

v2.0 soon!
Messenger not Doer

SMS

APP

Tool
What Is Its Role?
“Doer” Exceptions

- Interactions with non-PMIx systems
  - Fabric manager, credential subsystems, storage systems
- Aggregate local collective operations
  - Fence, connect/disconnect
- Environment “support”
  - Inventory collection, process monitoring, logging
Where Does It Fit?

Open MPI

PMIx
- PTL
- GDS
- TCP
- usock
- hash
ds12
ds21

MPI
- PML
- BTL
- MTL
- ofi
- psam2
- portals4
- basic
- ucx
- ...

SHMEM
- atomic
- scoll
- sshmem
- basic
- mpi
- ...
- map
- ev
- ucx
- pthreads
- qthreads
- nobots
- posix
- mmap
- sysv
- linux
- darwin
- solaris

OPAL (open portable access layer)
Building Open MPI
Tl;dr

wget \\
    https://download.open-mpi.org/release/open-mpi/vx.y/openmpi-x.y.z.tar.bz2 \n\ntar xf openmpi-x.y.z.tar.bz2 \ncd openmpi-x.y.z

./configure --prefix=$HOME/my-ompi <options> |& tee config.out

# Most <options> typically deal with network communications
# libraries (e.g., libfabric, UCX)

make -j 8 |& tee make.out
\nmake install |& tee install.out
Building from a Distribution Tarball vs. Git Clone

- Distribution tarballs are bootstrapped
- Building from a Git clone requires more tools
  - GNU Autotools
  - Flex
  - Pandoc (as of May 2020 git master / upcoming v5.0.0)

- See the HACKING file for more details about building from a Git clone
Configure Script Philosophies

- The configure script looks around your system
  - Searches for support for optional dependencies
  - If it finds them, builds support for them
  - If it does not find them, skip them (i.e., it's not an error)

- If user specifies `--with-FOO` (e.g., `--with-libfabric`)
  - The configure script will fail / abort if it cannot find / build support for FOO

- If user specifies `--without-FOO`
  - The configure script will (effectively) skip looking for FOO

- In short: *if a human asks for something that configure can’t do, abort*
Specifying Compilers

- Via the usual GNU Autoconf method: shell variables
  - `CC` (C compiler)
  - `CXX` (C++ compiler)
  - `FC` (Fortran compiler)
    - → F77 and F90 are no longer used!
    - → FC is used to compile all Open MPI Fortran code

- Best practice: specify these values **to the right of the configure token**
  - `./configure CC=/path/to/clang CXX=/path/to/clang++ FC=/path/to/gfortran ...
  - This way, these values end up in config.log
Project Libraries: Static or Shared?

- Open MPI supports building static and/or shared libraries
  - `--enable-static / --disable-static`
    - Referring to `libmpi.a`
  - `--enable-shared / --disable-shared`
    - Referring to `libmpi.so`

- Default (recommended):
  - `--disable-static`
  - `--enable-shared`
By default, components are built as Dynamic Shared Objects (DSOs)
- Individual files that are opened at runtime (e.g., via `dlopen()`)
Components: DSO or Included?

• By default, components are built as Dynamic Shared Objects (DSOs)
  ○ Individual files that are opened at run time (e.g., via dlopen())

• But the components can also be included in their respective project library
  ○ ./configure --disable-dlopen ...
Dependencies: libevent and hwloc

- Open MPI requires these two packages
  - Most modern Linux distros come with these packages
  - But installing the header files is not common
- Open MPI therefore (still) embeds full copies of these packages
  - If configure finds system-installed versions, it will use them (“external”)
  - If not, it will use the embedded copies (“internal”)

- Can use CLI options to force the “internal” or “external” versions:
  - `./configure --with-hwloc=/path/to/external/hwloc/install/tree` ...
  - `./configure --with-libevent=internal` ...
Communication Libraries

The most common two libraries these days are Libfabric and UCX:

- **Libfabric ("OpenFabrics Interfaces")**
  - `--with-libfabric[=LIBFABRIC_INSTALL_DIR]

- **UCX (Unified Communication X)**
  - `--with-ucx[=UCX_INSTALL_DIR]

But other communication libraries are also available, such as:

- **PSM2 (OmniPath) and Portals4** are also supported
  - `--with-psm2[=PSM2_INSTALL_DIR]`
  - `--with-portals4[=PORTALS4_INSTALL_DIR]`
Libfabric ("OFI") and UCX

- Libfabric was originally created by network vendors who wanted an HPC network API that wasn't tied to the abstractions of InfiniBand
  - Cisco (usNIC)
  - Cray (uGNI)
  - Intel (PSM, PSM2)
- It has since grown to support many additional network types
  - AWS EFA (Elastic Fabric Adapter)
  - BlueGene Q
  - IB Verbs (IB, RoCE, iWARP)
  - NetDirect
  - POSIX TCP and UDP sockets
  - Shared memory

- UCX became the next generation, higher-abstraction InfiniBand support, supporting:
  - InfiniBand
  - RoCE
- It also grew to support additional network types:
  - Cray uGNI
  - POSIX TCP sockets
  - Shared memory
NOTE: Open MPI does not use Libfabric or UCX for (pure) shared memory or TCP
Accelerators

Open MPI has CUDA support

- Nvidia (Mellanox) recommends building UCX with GDRcopy support
  - GDR = GPUDirect RDMA (there are multiple flavors of GPUDirect; this is the RDMA flavor)
  - Consult UCX documentation for GDRcopy build information
- Then build Open MPI with CUDA and UCX support
  - `./configure --with-cuda[=/path/to/cuda] --with-ucx[=/path/to/ucx]
- PSM2 also supports CUDA

When built with CUDA support, Open MPI can send messages from / receive messages to GPU device memory without copying through main RAM
Open MPI Installation Details

- Use the ompi_info command to see information about your installation
- Useful CLI options:
  - `--parsable`: machine-friendly format
  - `--all`: see all available MCA run-time parameters

```bash
$ ompi_info
Package: Open MPI jsquyres@laptop Distribution
Open MPI: 5.0.0a1
Open MPI repo revision: v2.x-dev-7856-ge1e8b2a373
Open MPI release date: Unreleased developer copy
MP API: 3.1.0
Ident string: 5.0.0a1
Prefix: /Users/jsquyres/bogus
Configured architecture: x86_64-apple-darwin19.5.0
Configured by: jsquyres
Configured on: Sat Jun 20 13:46:35 EDT 2020
Configure host: laptop
Configure command line: '--prefix=/Users/jsquyres/bogus'
Built by: jsquyres
Built on: Sat Jun 20 14:00:44 EDT 2020
Built host: laptop
C bindings: yes
Fort mpif.h: no
Fort use mpi: no
Fort use mpi size: deprecated-ompi-info-value
Fort use mpi_f08: no
...
Questions?

That’s it for part 1!

Join us for part 2 in two weeks:
July 8, 2020
8am US Pacific / 11am US Eastern / 3pm UTC / 5pm CEST
Thank you!

Join us for part 2 in two weeks:
July 8, 2020
8am US Pacific / 11am US Eastern / 3pm UTC / 5pm CEST