OpenHPC: Community Building Blocks for HPC Systems

Karl W. Schulz, Ph.D.
Scalable Datacenter Solutions Group, Intel

4th Annual MVAPICH User Group (MUG) Meeting
August 17, 2016 • Columbus, Ohio
Outline

• Motivation and brief history for this new community effort
• Overview of project governance
• Stack overview
• What’s coming
Motivation for Community Effort

• Many sites spend considerable effort aggregating a large suite of open-source projects to provide a capable HPC environment for their users:
  – necessary to build/deploy HPC focused packages that are either absent or do not keep pace from Linux distro providers
  – local packaging or customization frequently tries to give software versioning access to users (e.g. via modules or similar equivalent)

• OpenHPC is focused on establishing a centralizing community effort to:
  – provide a collection of pre-packaged binary components that can be used to help install and manage HPC systems throughout their lifecycle
  – implement integration testing to gain validation confidence
  – allow and promote multiple system configuration recipes that leverage community reference designs and best practices
  – provide additional distribution/integration mechanisms for leading research groups releasing open-source software
  – foster identification and development of relevant interfaces between supported components that allows for simple component replacement and customization
OpenHPC: a brief History…

- ISC’15 (June 2015) – BoF discussion on the merits/interest in a Community Supported HPC Repository and Management Framework
  - discussed convenience of distribution via standard Linux package managers
  - feedback at the time was that most parties were interested in CentOS/SLES
  - consensus that “modules” or equivalent needed to provide capable end-user development environment

- MUG’15 (August 2015) - shamelessly squeezed in a sidebar about potential interest in a community supported HPC repo

- SC’15 (November 2015) – Follow on BoF for a Comprehensive Open Community HPC Software Stack
  - initial seeding of OpenHPC and a 1.0 release
  - variety of interested community members assembled thru the Linux Foundation to work towards establishing a formal, collaborative project

- Nov’15 – May’16
  - Linux Foundation working group collaborating to define participating agreement, initial governance structure and solicit volunteers

➤ June 16, 2016 – Linux Foundation announces technical, leadership and member investment milestones with founding members and formal governance structure
OpenHPC: Project Members

Mixture of Academics, Labs, OEMs, and ISVs/OSVs

Project member participation interest? Please contact Jeff ErnstFriedman jernstfriedman@linuxfoundation.org
Community Governance Overview
Governing Board + Technical Steering Committee

- **Governing Board**
  - Board Representatives
  - Board Representatives for other Members, Indirect Representation (X:1)
  - TSC Director

**Budgetary oversight**
- IP policies, marketing
- Long-term roadmap guidance

**Technical Project Leadership**
- TSC
  - Architecture, Component selection, releases, day-to-day tech work

**HPC Community**
OpenHPC TSC - Role Overview

OpenHPC Technical Steering Committee (TSC)

- Project Leader
- Integration Testing Coordinator(s)
- Upstream Component Development Representative(s)
- End-User / Site Representative(s)
- Maintainers
OpenHPC TSC – Individual Members

- Reese Baird, Intel (Maintainer)
- Pavan Balaji, Argonne National Laboratory (Maintainer)
- David Brayford, LRZ (Maintainer)
- Todd Gamblin, Lawrence Livermore National Labs (Maintainer)
- Craig Gardner, SUSE (Maintainer)
- Yiannis Georgiou, ATOS (Maintainer)
- Balazs Gerofi, RIKEN (Component Development Representative)
- Jennifer Green, Los Alamos National Laboratory (Maintainer)
- Eric Van Hensbergen, ARM (Maintainer, Testing Coordinator)
- Douglas Jacobsen, NERSC (End-User/Site Representative)
- Chulho Kim, Lenovo (Maintainer)
- Greg Kurtzer, Lawrence Berkeley National Labs (Component Development Representative)
- Thomas Moschny, ParTec (Maintainer)
- Karl W. Schulz, Intel (Project Lead, Testing Coordinator)
- Derek Simmel, Pittsburgh Supercomputing Center (End-User/Site Representative)
- Thomas Sterling, Indiana University (Component Development Representative)
- Craig Stewart, Indiana University (End-User/Site Representative)
- Scott Suchyta, Altair (Maintainer)
- Nirmala Sundararajan, Dell (Maintainer)

https://github.com/openhpc/ohpc/wiki/Governance-Overview
Stack Overview

• Packaging efforts have HPC in mind and include compatible modules (for use with Lmod) with development libraries/tools

• Endeavoring to provide hierarchical development environment that is cognizant of different compiler and MPI families

• Intent is to manage package dependencies so they can be used as building blocks (e.g. deployable with multiple provisioning systems)

• Include common conventions for env variables

• Development library install example:
  
  # yum install petsc-gnu-mvapich2-ohpc

• End user interaction example with above install: (assume we are a user wanting to build a PETSC hello world in C)

  $ module load petsc
  
  $ mpicc -I$PETSC_INC petsc_hello.c -L$PETSC_LIB -lpetsc
# OpenHPC 1.1.1 - Current S/W components

<table>
<thead>
<tr>
<th>Functional Areas</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base OS</td>
<td>CentOS 7.2, SLES12 SP1</td>
</tr>
<tr>
<td>Administrative Tools</td>
<td>Conman, Ganglia, Lmod, LosF, Nagios, pdsh, prun, EasyBuild, ClusterShell, mrsh, Genders, Shine, Spack</td>
</tr>
<tr>
<td>Provisioning</td>
<td>Warewulf</td>
</tr>
<tr>
<td>Resource Mgmt.</td>
<td>SLURM, Munge</td>
</tr>
<tr>
<td>Runtimes</td>
<td>OpenMP, OCR</td>
</tr>
<tr>
<td>I/O Services</td>
<td>Lustre client (community version)</td>
</tr>
<tr>
<td>Numerical/Scientific Libraries</td>
<td>Boost, GSL, FFTW, Metis, PETSc, Trilinos, Hypre, SuperLU, SuperLU_Dist, Mumps, OpenBLAS, Scalapack</td>
</tr>
<tr>
<td>I/O Libraries</td>
<td>HDF5 (pHDF5), NetCDF (including C++ and Fortran interfaces), Adios</td>
</tr>
<tr>
<td>Compiler Families</td>
<td>GNU (gcc, g++, gfortran)</td>
</tr>
<tr>
<td>MPI Families</td>
<td>MVAPICH2, OpenMPI</td>
</tr>
<tr>
<td>Development Tools</td>
<td>Autotools (autoconf, automake, libtool), Valgrind,R, SciPy/NumPy</td>
</tr>
<tr>
<td>Performance Tools</td>
<td>PAPI, IMB, mpiP, pdtoolkit TAU</td>
</tr>
</tbody>
</table>

**Notes:**
- Additional dependencies that are not provided by the BaseOS or community repos (e.g. EPEL) are also included
- 3rd Party libraries are built for each compiler/MPI family (6 combinations typically)
- Resulting repositories currently comprised of ~270 RPMs
Hierarchical Overlay for OpenHPC software

<table>
<thead>
<tr>
<th>General Tools and System Services</th>
<th>Development Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>lmod</td>
<td>gcc</td>
</tr>
<tr>
<td>slurm</td>
<td>hdf5-gnu</td>
</tr>
<tr>
<td>munge</td>
<td>MVAPICH2</td>
</tr>
<tr>
<td>losf</td>
<td>IMPI</td>
</tr>
<tr>
<td>warewulf</td>
<td>OpenMPI</td>
</tr>
<tr>
<td>lustre client</td>
<td></td>
</tr>
<tr>
<td>ohpc</td>
<td></td>
</tr>
<tr>
<td>prun</td>
<td></td>
</tr>
<tr>
<td>pdsh</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compilers</th>
<th>Serial Apps/Libs</th>
<th>MPI Toolchains</th>
<th>Parallel Apps/Libs</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>hdf5-gnu</td>
<td>MVAPICH2</td>
<td>Boost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMPI</td>
<td>pHDF5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OpenMPI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel Composer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hdf5-intel</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Boost</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pHDF5</td>
</tr>
<tr>
<td></td>
<td>boost-gnu-openmpi</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>boost-gnu-mpi</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>boost-gnu-mvapich2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>phdf5-gnu-openmpi</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>phdf5-gnu-mpi</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>phdf5-gnu-openmpi</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Standalone 3rd party components</th>
<th>OHPC Repo</th>
<th>Distro Repo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stack Overview: Bare metal install

• **Step1:** Example OpenHPC 1.1 recipe assumes base OS is first installed on chosen master (SMS) host - e.g. install CentOS7.2 on SMS

  ![CentOS 7 installation screen](image)

  ```
  # export OHPC_GITHUB=https://github.com/openhpc/ohpc/releases/download
  # rpm -ivh ${OHPC_GITHUB}/v1.1.GA/ohpc-release-1.1-1.x86_64.rpm
  ```

• **Step2:** Enable OpenHPC repo using pre-packaged ohpc-release (or mirror repo locally)
Stack Overview: Bare metal install (cont.)

- Note that ohpc-release enables two repos:

<table>
<thead>
<tr>
<th># yum repolist</th>
<th>repo id</th>
<th>repo name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OpenHPC</td>
<td>OpenHPC-1.1 - Base</td>
</tr>
<tr>
<td></td>
<td>OpenHPC-updates</td>
<td>OpenHPC-1.1 - Updates</td>
</tr>
<tr>
<td></td>
<td>base</td>
<td>CentOS-7 - Base</td>
</tr>
<tr>
<td></td>
<td>epel</td>
<td>Extra Packages for Enterprise Linux 7 - x86_64</td>
</tr>
</tbody>
</table>

- Step3: install desired building blocks to build cluster or add development tools. **Convenience aliases** are provided to group related functionality.

```bash
[sms]# yum -y groupinstall ohpc-base
[sms]# yum -y groupinstall ohpc-warewulf

[sms]# yum -y groupinstall ohpc-slurm-server

[sms]# cp /opt/ohpc/pub/examples/network/centos/ifcfg-ib0 /etc/sysconfig/network-scripts

# Define local IPoIB address and netmask
[sms]# perl -pi -e "s/master_ipoib/{sms_ipoib}/" /etc/sysconfig/network-scripts/ifcfg-ib0
[sms]# perl -pi -e "s/ipoi0_netmask/{ipoib_netmask}/" /etc/sysconfig/network-scripts/ifcfg-ib0

# Initiate ib0
[sms]# ifup ib0
```

*note that community recipe is purposefully very transparent on config file edits and assumes Linux familiarity*
Stack Overview: Bare metal install (cont.)

- Recipe guides necessarily have a number of things to “cut-and-paste” if you want to reproduce them
- Have motivating need to automate during the validation process:
  - Cull out relevant commands automatically for use during CI testing
  - Seemed reasonable to make available directly, so there is a template starting script available with the documentation RPM which can be used for local installation and customization

Install the docs-ohpc package

```
[sms]# yum -y install docs-ohpc
```

Copy the provided template input file to use as a starting point to define local site settings:

```
[sms]# cp /opt/ohpc/pub/doc/recipes/vanilla/input.local input.local
```

Update `input.local` with desired settings

Copy the template installation script which contains command-line instructions culled from this guide.

```
```

Tip

Note that the template script provided is intended for use during initial installation and is not designed for repeated execution. If modifications are required after using the script initially, we recommend running the relevant subset of commands interactively.
Development Infrastructure
OpenHPC Development Infrastructure

What are we using to get the job done....?

The usual software engineering stuff:

- GitHub (SCM and issue tracking/planning)
- Continuous Integration (CI) Testing (Jenkins)
- Documentation (LaTeX)

Capable build/packaging system

- At present, we target a common delivery/access mechanism that adopts Linux sysadmin familiarity - ie. `yum/zypper` repositories for supported distros
  - ultimately delivering RPMs
  - `base` + `update` repositories to support life-cycle management
- Require flexible system to manage builds for multiple distros, multiple compiler/MPI family combinations, and dependencies across packages
- Have engineered a system using Open Build Service (OBS) which is supported by back-end git
  - git houses `.spec` files, tarballs, patches, documentation recipes, and integration tests
  - OBS performs automated builds and dependency analysis

https://github.com/openhpc/ohpc

https://build.openhpc.community
Build System - OBS

https://build.openhpc.community

- Using the **Open Build Service (OBS)** to manage build process
- OBS can drive builds for multiple repositories
- Repeatable builds carried out in chroot environment
- **Generates binary and src rpms**
- Publishes corresponding package repositories
- Client/server architecture supports distributed build slaves and multiple architectures
OpenHPC Build Architecture Conventions

- Motivation is to have single input to drive multiple output configurations (e.g. hierarchy for compiler/MPI families)
- Also want to establish baseline install path conventions
- Leverage variety of macros to aid in this effort

Install Path Macros

```
$ cat OHPC_macros
%define OHPC_BUILD 1
%define PROJ_NAME ohpc
%define OHPC_HOME /opt/%{PROJ_NAME}/
%define OHPC_ADMIN %{OHPC_HOME}/admin
%define OHPC_PUB %{OHPC_HOME}/pub
%define OHPC_COMPILERS %{OHPC_PUB}/compiler
%define OHPC_MPI_STACKS %{OHPC_PUB}/mpi
%define OHPC_APPS %{OHPC_PUB}/apps
%define OHPC_LIBS %{OHPC_PUB}/libs
%define OHPC_MODULES %{OHPC_PUB}/modulefiles
%define OHPC_MODULEDEPS %{OHPC_PUB}/moduledeps
```

MPI macros

```
$ cat OHPC_setup_mpi
if [ -z "$OHPC_MPI_FAMILY" ]; then
    echo "Unknown OHPC_MPI_FAMILY"
    exit 1
fi

if [ "$OHPC_MPI_FAMILY" = "openmpi" ]; then
    module load openmpi
elif [ "$OHPC_MPI_FAMILY" = "impi" ]; then
    module load impi
elif [ "$OHPC_MPI_FAMILY" = "mvapich2" ]; then
    module load mvapich2
else
    echo "Unsupported OHPC_MPI_FAMILY -> $OHPC_MPI_FAMILY"
    exit 1
fi
```

/opt/ohpc  

---  

*Top-level path convention for installs*
OpenHPC Build Architecture Conventions (cont.)

Example of compiler hierarchy template

- Default family choice defined, but can be overridden
- Family dependencies embedded for package managers

```
#include %{_sourcedir}/OHPC_macros

# OpenHPC convention: the default assumes the gnu compiler family;  
# however, this can be overridden by specifying the compiler_family  
# variable via rpmbuild or other mechanisms.

%!compiler_family: %define compiler_family gnu}

# Compiler dependencies
BuildRequires: lmod{PROJ_DELIM}
%if %{compiler_family} == gnu
BuildRequires: gnu-compilers{PROJ_DELIM}
Requires:     gnu-compilers{PROJ_DELIM}
%endif
%if %{compiler_family} == intel
BuildRequires: gcc-c++ intel-compilers-devel{PROJ_DELIM}
Requires:     gcc-c++ intel-compilers-devel{PROJ_DELIM}
%endif

Snippet from METIS .spec file

```

yum/zypper install:  

- hypre-gnu-mvapich2-ohpc
-metis-gnu-ohpc
- gnu-compilers-ohpc
Build System - OBS

- OBS manages dependency resolution and rebuilds all downstream packages
- Leveraging ability within OBS to link related packages
  - Convenient for packages with compiler and MPI dependencies
  - Single commit drives all package permutations
- OBS builds automatically triggered via git commit hooks

```
$ lsmetis-*
metis-gnu:
    _service
metis-intel:
    _link

$ catmetis-intel/_link
<link project='OpenHPC:1.1:Factory' package='metis-gnu'>
<patches>
    <topadd>%define compiler_family intel</topadd>
</patches>
</link>
```
Integration Testing
Integration/Test/Validation

Testing is a key element for us and the intent is to build upon existing validation efforts and augment component-level validation with targeted cluster-validation and scaling initiatives including:

- install recipes
- cross-package interaction
- development environment

- mimic use cases common in HPC deployments
- upgrade mechanism
Integration/Test/Validation

• To facilitate global efforts in diagnostics/validation, we have devised a standalone integration test infrastructure

• Intent was to create families of tests that could be used during:
  - initial install process (can we build a system?)
  - post-install process (does it work?)
  - developing tests that touch all of the major components (can we compile against 3rd party libraries, will they execute under resource manager, etc)

• Expectation is that each new component included will need corresponding integration test collateral

• These integration tests and harness are included in GitHub repo
Post Install Integration Tests - Overview

• Where do we get the tests? Ideally, we leverage directly from the packages we are testing:
  – as an example, we went down this path originally with HDF5
    • discovered the tests that ship with their “make check” actually test internal (non-public) API’s
    • did not make sense as the internal header files are not part of a normal HDF5 install
    • ended up using separate collection of tests from HDF5 community that are used to illustrate APIs (only C and Fortran though)
    • we integrated these as a subcomponent and added some companion C++ tests
  – in other cases, we have to cook up the tests from scratch (great opportunity for community participation)

• Dev environment tests are a mixture of flavors:
  – interactive execution to verify certain binaries are in working order
  – successful compilation to test libraries provided via OpenHPC
  – successful interactive execution
  – tests for module usability and consistency
  – successful remote execution under resource manager

> 1,000 jobs submitted to RM as part of the current test suite
Global testing harness includes a number of embedded subcomponents:

- major components have configuration options to enable/disable
- end user tests need to touch all of the supported compiler and MPI families
- we abstract this to repeat the tests with different compiler/MPI environments:
  - gcc/Intel compiler toolchains
  - Intel, OpenMPI, MVAPICH2 MPI families

Package version.............. : test-suite-1.0.0
Build user................... : jilluser
Build host................... : master4-centos71.localdomain
Configure date............... : 2015-10-26 09:23
Build architecture.......... : x86_64-unknown-linux-gnu
Test suite configuration.... : long

Submodule Configuration:

User Environment:
- RMS test harness
- Munge
- Apps
- Compilers
- MPI
- HSN
- Modules
- OOM

Dev Tools:
- Valgrind
- R base package
- TBB
- CILK

Performance Tools:
- mpiP Profiler
- Papi
- PETSc
- TAU

Libraries:
- Adios : enabled
- Boost : enabled
- Boost MPI : enabled
- FFTW : enabled
- GSL : enabled
- HDF5 : enabled
- HYPRE : enabled
- IMB : enabled
- Metis : enabled
- MUMPS : enabled
- NetCDF : enabled
- Numpy : enabled
- OPENBLAS : enabled
- PETSc : enabled
- PHDF5 : enabled
- ScaLAPACK : enabled
- Scipy : enabled
- Superlu : enabled
- Superlu_dist : enabled
- Trilinos : enabled

Apps:
- MiniFE : enabled
- MiniDFT : enabled
- HPCG : enabled
- PRK : enabled
Integration Tests - Let’s see one submodule test in action

**Lmod user environment**

- These are examples that primarily test interactive commands
- We are using the Bash Automated Testing System (Bats) for these tests
  - a TAP-complaint framework for Bash
  - available on GitHub
- We have extended Bats to:
  - create JUnit output for parsing into Jenkins CI environment
  - capture execution runtimes

```bash
$ ./interactive_commands
✓ [modules] module purge
✓ [modules] module list
✓ [modules] module help
✓ [modules] module load/unload
✓ [modules] module whatis
✓ [modules] module swap
✓ [modules] path updated

7 tests, 0 failures
```

```bash
$ ./rm_execution
✓ [modules] env variable passes through (slurm)
✓ [modules] loaded module passes through (slurm)
✓ [modules] module commands available (slurm)
✓ [modules] module load propagates thru RMS (slurm)

4 tests, 0 failures
```

**Test Result: modules**

<table>
<thead>
<tr>
<th>Test name</th>
<th>Duration</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>[module] module commands available (slurm)</td>
<td>0.49 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[module] module load propagates thru RMS (slurm)</td>
<td>0.62 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] env variable passes through (slurm)</td>
<td>0.44 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] loaded module passes through (slurm)</td>
<td>0.54 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] module help</td>
<td>0.47 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] module list</td>
<td>0.37 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] module load/unload</td>
<td>2.9 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] module purge</td>
<td>0.14 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] module swap</td>
<td>0.94 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] module whatis</td>
<td>0.46 sec</td>
<td>Passed</td>
</tr>
<tr>
<td>[modules] path updated</td>
<td>1.4 sec</td>
<td>Passed</td>
</tr>
</tbody>
</table>
What’s Coming

• Some known big ticket items on the horizon for the TSC
  – establishing a process and prioritization/selection process for including new software components
  – establish minimum integration test expectations
  – establish packaging conventions:
    • naming schemes
    • dependency hierarchy management
    • installation paths
    • upgrade/rollback? mechanisms
  – roadmap timeline for next release (and cadence strategy for future releases)
  – addition of public CI infrastructure, roll out of additional architecture builds (e.g. ARM)
Thanks for your Time - Questions?

karl.w.schulz@intel.com

http://openhpc.community
https://github.com/openhpc/ohpc
https://build.openhpc.community (repo)