E4S: Extreme-Scale Scientific Software Stack

https://e4s.io

Sameer Shende
Performance Research Laboratory, OACISS, U. Oregon

sameer@cs.uoregon.edu

Tutorial, NCSA Webinar.
Extreme-scale Scientific Software Stack (E4S)
https://e4s.io

- E4S is a community effort to provide open source software packages for developing, deploying, and running scientific applications on HPC platforms.
- E4S provides both source builds and containers of a broad collection of HPC software packages.
- E4S exists to accelerate the development, deployment and use of HPC software, lowering the barriers for HPC users.
- E4S provides containers and turn-key, from-source builds of 80+ popular HPC software packages:
  - MPI: MPICH and OpenMPI
  - Development tools: TAU, HPCToolkit, and PAPI
  - Math libraries: PETSc and Trilinos
  - Data and Viz tools: Adios, HDF5, and Paraview
Extreme-scale Scientific Software Stack (E4S)
https://e4s.io

- Spack [http://spack.io] is the primary means for software delivery
- SDKs: collection of related ECP ST products where coordination across package teams will improve usability and practices, and foster community growth among teams that develop similar and complimentary capabilities. An SDK involves several products.
- Containers of pre-built binaries of ECP ST products.
- Container runtimes supported
  - Docker: Dockerhub: exascaleproject/sdk:AHM19
  - Charliecloud
  - Shifter
  - Singularity
  - Inception at NCAR
- VirtualBox Open Virtualization Appliance (OVA) image that contains these runtimes
- MPI replacement strategies to use native network interconnect
What are containers

• A lightweight collection of executable software that encapsulates everything needed to run a single specific task
  – Minus the OS kernel
  – Based on Linux only

• Processes and all user-level software is isolated

• Creates a portable* software ecosystem

• Think chroot on steroids

• Docker most common tool today
  – Available on all major platforms
  – Widely used in industry
  – Integrated container registry via Dockerhub

* Container slides from: Andrew Younge, Sandia, “Getting Started with Containers on HPC”, ISC-HPC 2019 tutorial
Hypervisors and Containers

- Type 1 hypervisors insert layer below host OS
- Type 2 hypervisors work as or within the host OS
- Containers do not abstract hardware, instead provide “enhanced chroot” to create isolated environment
- Location of abstraction can have impact on performance
- All enable custom software stacks on existing hardware
• Abstracting hardware and software resources has had profound impact on computing

• Virtual Machines to Cloud computing in the past decade
  – Early implementations limited by performance
  – HPC on clouds: FutureGrid, Magellan, Chameleon Cloud, Hobbes, etc
  – Some initial successes, but not always straightforward

• OS-level virtualization a bit different
  – User level code packaged in container, can then be transported
  – Single OS kernel shared across containers and provides isolation
  – Cgroups traditionally multiplexes hardware resources
  – Performance is good, but OS flexibility is limited
Containers in Cloud Industry

• Containers are used to create large-scale loosely coupled services
• Each container runs just 1 user process – “micro-services”
  – 3 httpd containers, 2 DBs, 1 logger, etc
• Scaling achieved through load balancers and service provisioning
• Jam many containers on hosts for increased system utilization
• Helps with dev-ops issues
  – Same software environment for developing and deploying
  – Only images changes are pushed to production, not whole new image (CoW).
  – Develop on laptop, push to production servers
  – Interact with github similar to developer code bases
  – Upload images to ”hub” or “repository” whereby they can just be pulled and provisioned
Containers

• Containers are gaining popularity for software management of distributed systems
• Enable way for developers to specify software ecosystem
• US DOE High Performance Computing (HPC) resources need to support emerging software stacks
  – Applicable to DevOps problems seen with large HPC codes today
  – Support new frameworks & cloud platform services
• But HPC systems are very dissimilar from cloud infrastructure
  – MPI-based bulk synchronous parallel workloads are common
  – Scale-out to thousands of nodes
  – Performance is paramount
Container features in HPC

- **BYOE - Bring-Your-Own-Environment**
  - Developers define the operating environment and system libraries in which their application runs.

- **Composability**
  - Developers explicitly define how their software environment is composed of modular components as container images,
  - Enable reproducible environments that can potentially span different architectures.

- **Portability**
  - Containers can be rebuilt, layered, or shared across multiple different computing systems
  - Potentially from laptops to clouds to advanced supercomputing resources.

- **Version Control Integration**
  - Containers integrate with revision control systems like Git
  - Include not only build manifests but also with complete container images using container registries like Docker Hub.
Container features not wanted in HPC

• Overhead
  – HPC applications cannot incur significant overhead from containers

• Micro-Services
  – Micro-services container methodology does not apply to HPC workloads
  – 1 application per node with multiple processes or threads per container

• On-node Partitioning
  – On-node partitioning with cgroups is not necessary (yet?)

• Root Operation
  – Containers allow root-level access control to users
  – In supercomputers this is unnecessary and a significant security risk for facilities

• Commodity Networking
  – Containers and their network control mechanisms are built around commodity networking (TCP/IP)
  – Supercomputers utilize custom interconnects w/ OS kernel bypass operations
HPC Containers

- Docker not good fit for running HPC workloads
  - Security issues
    - Can’t allow root on shared resources
  - Lack of HPC architecture support
    - No batch integration
    - Assumes local resources
    - Assumes commodity TCP/IP

- Many different container options in HPC
  - Shifter
  - Singularity
  - Charliecloud
  - ...
Spack

- E4S uses the Spack package manager for software delivery
- Spack provides the ability to specify versions of software packages that are and are not interoperable.
- Spack is a build layer for not only E4S software, but also a large collection of software tools and libraries outside of ECP ST.
- Spack supports achieving and maintaining interoperability between ST software packages.
- Acknowledgement: The remaining Spack slides in this presentation are from a talk given by the Spack PI, Todd Gamblin, CASC, LLNL.

- Next: Motivation for Spack!
Scientific software is becoming extremely complex.
Even proprietary codes are based on many open source libraries

- Half of this DAG is external (blue); *more* than half of it is open source
- Nearly *all* of it needs to be built specially for HPC to get the best performance
The Exascale Computing Project is building an entire ecosystem

- Every application has its own stack of dependencies.
- Developers, users, and facilities dedicate (many) FTEs to building & porting.
- Often trade reuse and usability for performance.

= up to 1,260,000 combinations!

<table>
<thead>
<tr>
<th>15+ applications</th>
<th>80+ software packages</th>
<th>5+ target architectures/platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 7 compilers</td>
<td>10+ Programming Models</td>
<td>X</td>
</tr>
<tr>
<td>Intel GCC Clang XL</td>
<td>OpenMPI MPICH MVAPICH</td>
<td>X</td>
</tr>
<tr>
<td>PGI Cray NAG</td>
<td>OpenMP CUDA OpenACC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dharma Legion RAJA</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Kokkos</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2-3 versions of each package + external dependencies</td>
<td>X</td>
</tr>
</tbody>
</table>

We must make it easier to rely on others’ software!
How to install software on a Mac laptop, circa 2013

```
(gluon):~$ port install libelf
```

How to install software on a supercomputer

1. Download all 16 tarballs you need
2. Start building!
3. Run code
4. Segfault!?
5. Start over…
What about modules?

• Most supercomputers deploy some form of environment modules
  – TCL modules (dates back to 1995) and Lmod (from TACC) are the most popular

```
$ gcc
-bash: gcc: command not found

$ module load gcc/7.0.1
$ gcc --dumpversion
7.0.1
```

• Modules don’t handle installation!
  – They only modify your environment (things like PATH, LD_LIBRARY_PATH, etc.)

• Someone (likely a team of people) has already installed gcc for you!
  – Also, you can only `module load` the things they’ve installed
What about containers?

• Containers provide a great way to reproduce and distribute an already-built software stack

• Someone needs to build the container!
  – This isn’t trivial
  – Containerized applications still have hundreds of dependencies

• Using the OS package manager inside a container is insufficient
  – Most binaries are built unoptimized
  – Generic binaries, not optimized for specific architectures

• Developing with an OS software stack can be painful
  – Little freedom to choose versions

We need something more flexible to build the containers
Spack is a flexible package manager for HPC

• How to install Spack (works out of the box):
  
  $ git clone https://github.com/spack/spack
  $ . spack/share/spack/setup-env.sh

• How to install a package:

  $ spack install hdf5

• HDF5 and its dependencies are installed within the Spack directory.

• Unlike typical package managers, Spack can also install many variants of the same build.
  – Different compilers
  – Different MPI implementations
  – Different build options

Visit spack.io

github.com/spack/spack/spack

@spackpm
Spack provides the **spec** syntax to describe custom configurations

- Each expression is a **spec** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

- Spec syntax is recursive
  - Full control over the combinatorial build space

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ spack install mpileaks</code></td>
<td>unconstrained</td>
</tr>
<tr>
<td><code>$ spack install mpileaks@3.3</code></td>
<td>@ custom version</td>
</tr>
<tr>
<td><code>$ spack install mpileaks@3.3 %gcc@4.7.3</code></td>
<td>% custom compiler</td>
</tr>
<tr>
<td><code>$ spack install mpileaks@3.3 %gcc@4.7.3 +threads</code></td>
<td>+/- build option</td>
</tr>
<tr>
<td><code>$ spack install mpileaks@3.3 cxxflags=&quot;-O3 -g3&quot;</code></td>
<td>setting compiler flags</td>
</tr>
<tr>
<td><code>$ spack install mpileaks@3.3 os=cnl10 target=haswell</code></td>
<td>setting target for X-compile</td>
</tr>
<tr>
<td><code>$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3</code></td>
<td>^ dependency information</td>
</tr>
</tbody>
</table>
`spack list` shows what packages are available

$ spack list

22 packages.

The `spack list` command lists what packages are available.

- Spack has over 3,000 built-in package recipes.
`spack find` shows what is installed

```
$ spack find
===> 103 installed packages.
-- linux-rhel7-x86_64 / gcc@4.4.7 -------------------------------
ImageMagick@6.8.9-10  glib@2.42.1  libtiff@4.0.3  pango@1.36.8  qt@4.8.6
SAMRAI@3.9.1  graphlib@2.0.0  libtool@2.4.2  parmetis@4.0.3  qt@5.4.0
adept-utils@1.0  gtkplus@2.24.25  libxml2@2.9.2  pixman@0.32.6  ravel@1.0.0
atk@2.14.0  harfbuzz@0.9.37  llvmp@3.0  py-dateutil@2.4.0  readline@6.3
boost@1.55.0  hdf5@1.8.13  metis@5.1.0  py-nose@1.3.4  starpu@1.1.4
cairo@1.14.0  icu@54.1  py-numpy@1.9.1  star@2.1.0
callpath@1.0.2  jpeg@9a  py-pty@2014.10  xz@5.2.0
dyninst@8.1.2  libdwarf@20130729  ncurses@5.9  py-setuptools@11.3.1  zlib@1.2.8
freetype@2.9.3  libelf@0.8.13  ocr@2015-02-16  py-six@1.9.0
freetype@2.5.3  libffi@3.1  openmpi@1.8.2
freetype@2.5.3  libpng@2.0.2  openssl@1.0.1h
freetype@2.5.3  libpng@1.6.16  python@2.7.8
freetype@2.5.3  libpng@1.6.16  qhull@1.0
-- linux-rhel7-x86_64 / gcc@4.8.2 -------------------------------
adept-utils@1.0.1  boost@1.55.0  cmake@5.6-special  libdwarf@20130729  mpich@3.0.4
adept-utils@1.0.1  cmake@5.0.0  libelf@0.8.13  mpich@3.0.4
adept-utils@1.0.1  cmake@5.0.0  libelf@0.8.13  openmpi@1.8.2
-- linux-rhel7-x86_64 / intel@14.0.2 -----------------------------
hwloc@1.9  mpich@3.0.4  starpu@1.1.4
-- linux-rhel7-x86_64 / intel@15.0.0 -----------------------------
adept-utils@1.0.1  boost@1.55.0  libdwarf@20130729  libelf@0.8.13  mpich@3.0.4
-- linux-rhel7-x86_64 / intel@15.0.1 -----------------------------
adept-utils@1.0.1  callpath@1.0.2  libdwarf@20130729  libelf@0.8.13  mpich@3.0.4
boost@1.55.0  hwloc@1.9  libelf@0.8.13  starpu@1.1.4
```

- All the versions coexist!
  - Multiple versions of same package are ok.
- Packages are installed to automatically find correct dependencies.
- Binaries work regardless of user’s environment.
- Spack also generates module files.
  - Don’t have to use them.
The Spack community is growing rapidly

- **Spack simplifies HPC software for:**
  - Users
  - Developers
  - Cluster installations
  - The largest HPC facilities

- **Spack is central to ECP’s software strategy**
  - Enable software reuse for developers and users
  - Allow the facilities to consume the entire ECP stack

- **The roadmap is packed with new features:**
  - Building the ECP software distribution
  - Better workflows for building containers
  - Stacks for facilities
  - Chains for rapid dev workflow
  - Optimized binaries
  - Better dependency resolution

Visit spack.io

github.com/spack/spack

@spackpm
Exascale Platform Preparation

- SDK Exascale platform preparation is focused on interoperable delivery.
- ST products from SDKs are released in the Extreme-scale Scientific Software Stack (E4S) [https://e4s.io].
  - E4S: a community effort to provide open source software packages for developing, deploying, and running scientific applications on HPC platforms
- E4S containers and Spack based builds currently support the following pre-exascale systems:
  - Theta at ALCF (Cray XC).
  - Cori at NERSC (Cray XC).
  - Summit, Sierra, Butte, RZAnsel (IBM Power 9 AC922).
  - Linux x86_64 systems at LANL (Grizzly), Sandia (Voltrino), LLNL (Quartz).
  - Other NSF platforms including Frontera (TACC).
- E4S preparation for future Exascale systems includes testing on AMD and Intel systems.
Integration and Interoperability: E4S

- E4S is released twice a year. Two versions have been released to date and we are planning for a release at SC19. The E4S 0.2 release supports:
  - Containers and turn-key, from-source builds of 80+ popular HPC software packages
  - 37 full release ECP ST products including:
    - MPI: MPICH and OpenMPI
    - Development tools: TAU, HPCToolkit, and PAPI
    - Math libraries: PETSc and Trilinos
    - Data and Viztools: Adios, HDF5, and Paraview
  - Limited access to 10 additional ECP ST products
    - Docker
    - Singularity
    - Shifter
    - Charliecloud
    - Inception
  - Open Virtualization Appliance (OVA) for VirtualBox features Spack, E4S containers, and support for container environments
Integration and Interoperability: E4S on AWS

- E4S AWS public image ami-063e830287b86155c (US-West-2 Oregon) has following container runtimes:
  - Docker
  - Shifter
  - Singularity
  - Charliecloud
- Spack with base PMR components
- E4S full featured Singularity image
  - (exascaleproject/sdk:AHM19)
- Used in ISC-HPC 2019 tutorials
- Used as base image for NASA GEOS-Chem E4S public image
- Resources provided by AWS AI/ML team
Reproducible, Customizable Container Builds & Spack Mirrors

• E4S provides base images and recipes for building Docker containers based on SDKs
  - Git: https://github.com/UO-OACISS/e4s
  - Base images released (September 2019):
    • UBI 7.6 (RHEL Universal Binary Image for container builds) for x86_64
    • Centos 7.6 for x86_64
    • Ubuntu 18.04 for x86_64
    • UBI 7.6 (RHEL) for ppc64le

• E4S provides build caches for Spack for native bare-metal as well as container builds based installation of ST products
  - Build caches: https://oaciss.uoregon.edu/e4s
    • The build cache model can be extended to target platforms, and can be managed by facilities staff when appropriate.
E4S Build Cache Binaries

Spacc E4S Build Cache

Last updated: 13-Oct-2019 07:43 PST

Click on one of the packages below to see a list of all available variants.

1363 Spack binaries in the build cache

Search

ados2@2.4.0  adlbx@0.9.2  adol-c@develop  alquimia@xsdk-0.4.0  aml@0.1.0  annex@18.10.1  ant@1.10.0  argbots@1.0rc1  arpack-ng@3.7.0  autoconf@2.69
automake@1.16.1  axl@0.1.1  bdflopfc@1.0.5  binutils@2.32  bison@3.0.5  bmpi@develop  bolt@1.0rc1  boost@1.70.0  butterflyphack@1.0.1  butterflyphack@master
bzip2@1.0.8  c-block@1.17.0  cairo@1.16.0  caliper@2.0.1  cgns@develop  cinch@develop  cmake@3.15.1  cmake@3.15.3  cuda@10.0.130  cuda@10.1.243
curl@7.63.0  darshan-runtime@3.1.7  darshan-util@3.1.7  dealii@9.0.1  diffutils@3.7  double-conversion@2.0.1  doxygen@1.8.15  dtcmp@1.1.0
dyninst@10.1.0

Click on the full spec link to find out more.

<table>
<thead>
<tr>
<th>Link</th>
<th>Arch</th>
<th>OS</th>
<th>Compiler</th>
<th>Created</th>
<th>Full Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Spec</td>
<td>x86_64</td>
<td>centos7</td>
<td>gcc@7.3.0</td>
<td>19-Sep-2019 19:07</td>
<td>m+6b manages 8Ly3g4w4US8m7vJr8x</td>
</tr>
<tr>
<td>Full Spec</td>
<td>x86_64</td>
<td>rhel7</td>
<td>gcc@7.3.0</td>
<td>19-Sep-2019 19:11</td>
<td>v2+863L3g7v7v4sds3v281mg53Qp7o</td>
</tr>
<tr>
<td>Full Spec</td>
<td>x86_64</td>
<td>ubuntu18.04</td>
<td>gcc@7.3.0</td>
<td>19-Sep-2019 19:22</td>
<td>cwadthw68h58tw2kv8745f7uxy</td>
</tr>
<tr>
<td>Full Spec</td>
<td>x86_64</td>
<td>ubuntu18.04</td>
<td>gcc@7.3.0</td>
<td>19-Sep-2019 19:23</td>
<td>nekynvwy4pove4pslepqtm2bts7kn</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e25n3763g75e5ac9w6btnty3jbgfvt</td>
</tr>
</tbody>
</table>

effutils@0.176  emacs@26.2  environment-modules@4.3.0  environment-modules@4.3.1  er@0.0.3  exmcutil@0.5.7  expat@2.2.5  faodel@1.1906.1  findingutil@4.6.0
flatcc@0.5.3  flecs@develop  flex@2.6.4  font-utility@1.3.2  fontconfig@2.12.3  fontsproto@2.1.3  freetype@2.9.1  gasnet@2019.3.0  gawk@4.1.4  gcc@7.3.0
gdbm@1.18.1  geopm@1.0.0-rc2  gettext@0.19.8.1  git@2.21.0  glib@2.56.3  glm@0.9.7.1  globalarrays@5.7  glproto@1.4.17  gmp@6.1.2  googletest@1.8.1

https://oaciss.uoregon.edu/e4s
Reproducible Container Builds using E4S Base Images

- PMR SDK base image (UBI 7.6) has Spack build cache mirror and GPG key installed.
- Base image has GCC and MPICH configured for MPICH ABI level replacement (with system MPI).
- Customized container build using binaries from E4S Spack build cache for fast deployment.
- No need to rebuild packages from the source code.
- Same recipe for container and native bare-metal builds with Spack!
Reproducible Base Images on Dockerhub

- ecpe4s
- x86_64
- ppc64le
- aarch64

- Centos 7.6
- Ubuntu 18.04
- RHEL/UBI 7.6
Docker Recipes on GitHub

- Base images
- SDKs
- E4S

https://github.com/UO-OACISS/e4s
Spack Build Caches from E4S Base Images

**Index of /e4s/x86_64/build_cache/linux-rhel7-x86_64/gcc-7.3.0**

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Directory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x86_64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>build cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>○ 40 GB on disk</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Index of /e4s/ppc64le/build_cache/linux-centos7-ppc64le/gcc-7.3.0**

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Directory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppc64le</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>build cache</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>○ 2.6 GB on disk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>○ early stages of effort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>○ Initial ARM 64 build cache is underway</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Docker container of E4S

% docker pull exascaleproject/sdk:AHM19

• Using USB stick or images from https://e4s.io:
  • % gunzip –c ecp.tgz | docker load
    % docker images
  • Mount home directory:

% docker -i -v $HOME:$HOME -t exascaleproject/sdk:AHM19 /bin/bash
% which spack
% cp –r /usr/local/packages/ecp/demo . ; cd demo; cat README
Using Shifter at NCSA BlueWaters

Load shifter module and E4S image on the compute node

• Allocate a node
  – % qsub -l -l nodes=1:ppn=32 -l walltime=01:15:00 -l gres=shifter16

• Load the shifter module
  – % module load shifter

• Pull the image (once)
  – % shifterimg pull exascaleproject/sdk:AHM19

• Launch the image
  – % shifter --image=exascaleproject/sdk:AHM19 -- /bin/bash
  – % unset CRAYPE_VERSION; . /etc/bashrc
  – % spack find
Extreme-scale Scientific Software Stack (E4S)  
https://e4s.io

- Containers for HPC that include ECP ST products.
E4S Second Release (37+ ST products) exascaleproject/sdk:AHM19 (on Dockerhub)
Extreme-scale Scientific Software Stack (E4S)  
https://e4s.io
Running MPI applications on other systems

• Applications built with MPI in the E4S container can be replaced by the system MPI!
• This allows fast inter-node communication using the native interconnect.
• Application and data are external to the E4S container.
• Programming models, compilers, runtime libraries, and tools are inside the container.
• We can replace MPI using the MPICH ABI compatibility layer.
• Goal: Build an MPI binary once and run it un-modified on all HPC Linux x86_64 clusters!
Using E4S on NCSA BlueWaters and replacing MPI

**Step 1: Allocate a node with the E4S image**

- `qsub -l -l nodes=2:ppn=32 -l walltime=01:15:00 -l gres=shifter16 -v UDI=exascaleproject/sdk:AHM19`
- This allocates a single node for 1:15h
- Specifies the use of Shifter as the container environment
- The image is `exascaleproject/sdk:AHM19`
- This image was pulled on a compute node previously using:
  - `%module load shifter; shifterimg pull exascaleproject/sdk:AHM19`
- After this `qsub` step, we can now launch the job using `aprun`
Using E4S on NCSA BlueWaters Replacing MPI

Step 2: Create a file called ~/shifter_mpi.sh

```bash
#!/bin/bash
# set up LD_LIBRARY_PATH
for dir in $(echo $CRAY_LD_LIBRARY_PATH:/opt/cray/wlm_detect/default/lib64 | tr ':' ' ')
do
    realpath=$(readlink -f "$dir")
    if [[ -z $LD_LIBRARY_PATH ]]; then
        eval 'export LD_LIBRARY_PATH=/dsl'$realpath
    else
        eval 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/dsl'$realpath
    fi
done
```
Replacing MPI using cray-mpich-abi package

Step 3: Source this ~/shifter_mpi and setup LD_LIBRARY_PATH

```bash
% cat run.sh
#!/bin/bash
export CRAY_ROOTFS=SHIFTER
module load shifter
module unload PrgEnv-cray # or any other PrgEnv module currently loaded
module load PrgEnv-gnu # or PrgEnv-intel
module unload cce
module unload cray-mpich
module load cray-mpich-abi
export LD_LIBRARY_PATH=$CRAY_MPICH_DIR/lib:$LD_LIBRARY_PATH
source ~/shifter_mpi.sh
export LD_LIBRARY_PATH=/usr/local/packages/ecp/spack/opt/spack/linux-centos7-x86_64/gcc-7.3.0/hwloc-1.11.9-7xxgxbg65an7zmrztfcuu3hs73puj6v3/lib:$LD_LIBRARY_PATH
export OMP_NUM_THREADS=2
aprun -b -n 64 -- ./lulesh.host -i 100
```
Step 4: run the example

% ./run.sh
Running problem size $30^3$ per domain until completion
Num processors: 64
Num threads: 2
Total number of elements: 1728000

... Run completed:
- Problem size = 30
- MPI tasks = 64
- Iteration count = 100
- Final Origin Energy = $8.465100e+07$
- Testing Plane 0 of Energy Array on rank 0:
  - \( \text{MaxAbsDiff} = 7.916242e-09 \)
  - \( \text{TotalAbsDiff} = 3.030168e-08 \)
  - \( \text{MaxRelDiff} = 1.224484e-13 \)

- Elapsed time = 16.58 (s)
- Grind time (us/z/c) = 6.1409471 (per dom) (0.095952298 overall)
- FOM = 10421.845 (z/s)
- Elapsed time = 16.58 (s)
- Grind time (us/z/c) = 6.0131382 (per dom) (0.22270882 overall)
- FOM = 4490.1679 (z/s)

Application 81575093 resources: utime ~442s, stime ~20s, Rss ~45404, inblocks ~9110
NAS Parallel Benchmark Example

Compare a native build with a container based build

% cd ~/scratch/host/NPB3.1
% make suite
% cd bin
% ./run.sh

% cd ~/scratch/demo/NPB3.1/bin/
% ./run.sh
E4S VirtualBox OVA image

Contains all four container runtimes and the E4S Singularity image!

• Docker
• Singularity
• Shifter
• Charliecloud
E4S image on Amazon AWS

Contains all four container runtimes and the E4S Singularity image!

- AWS AMI ID (Oregon, us-west-2 region):
  - ami-063e830287b86155c

- Royalty free, public image with HPC, AI, and 4 container runtimes

- Launch EC2 instance with this AMI
  - Login: livetau
  - Password: ****
Future work, issues…

● Increasing the number of ST packages in E4S
● Porting to IBM and ARM platforms
● Support for GPUs and visualization tools
● Addition of CI testing
● Facility deployment
● Scalable startup with full-featured “Supercontainers”
● Improving the launch of MPI applications
E4S: How to get involved

● E4S BoF at SC19
  ● Tuesday, Nov. 19, 12:15pm – 1:15pm, Room 405-406-407

● CANOPIE-HPC Workshop at SC19
  ● 1st Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC
  ● Monday, Nov. 18, 2019, 2pm – 5:30pm, Room 704-706
  ● https://canopie-hpc.nersc.gov/

● “Container Computing for HPC and Scientific Workflows”
  ● Tutorial at SC19, Sunday, Nov. 17, 2019, 1:30pm – 5pm, Room 201
Acknowledgment

“This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation’s exascale computing imperative.”