CompBioMed

A Centre of Excellence in Computational Biomedicine

e-Seminar series

e-Seminar #18

High Performance Containers?

Presenter:
Michael Bareford (EPCC)

6 October 2021

The e-Seminar will start at 3pm CEST / 2pm BST

Moderator:
Tim Weaving (UCL)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 675451

INSILICO WORLD

https://insilicoworld.slack.com/archives/C0151M02TA4

VPH Institute
Building the Virtual Physiological Human

The e-Seminar series is run in collaboration with:

THE UNIVERSITY of EDINBURGH

epcc

CompBioMed
A Centre of Excellence in Computational Biomedicine

e-Seminar series

e-Seminar #18

High Performance Containers?

Presenter:
Michael Bareford (EPCC)

6 October 2021

Welcome!

Moderator:
Tim Weaving (UCL)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 675451

INSILICO WORLD

https://insilicoworld.slack.com/archives/C0151M02TA4

VPH Institute
Building the Virtual Physiological Human

The e-Seminar series is run in collaboration with:

THE UNIVERSITY of EDINBURGH

epcc

HPC Containers?

THE UNIVERSITY
of EDINBURGH

epcc

Contents

1. Containers, Singularity and MPI

Contents

1. Containers, Singularity and MPI

2. GROMACS on ARCHER2 (4cab)

GROMACS is a molecular dynamics code. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions.

http://www.gromacs.org/

Contents

1. Containers, Singularity and MPI
2. GROMACS on ARCHER2 (4cab)
3. The Container Factory

Contents

1. Containers, Singularity and MPI
2. GROMACS on ARCHER2 (4cab)
3. The Container Factory
4. GROMACS on Cirrus (CPU and GPU)
5. Conclusions...

What is a Container?

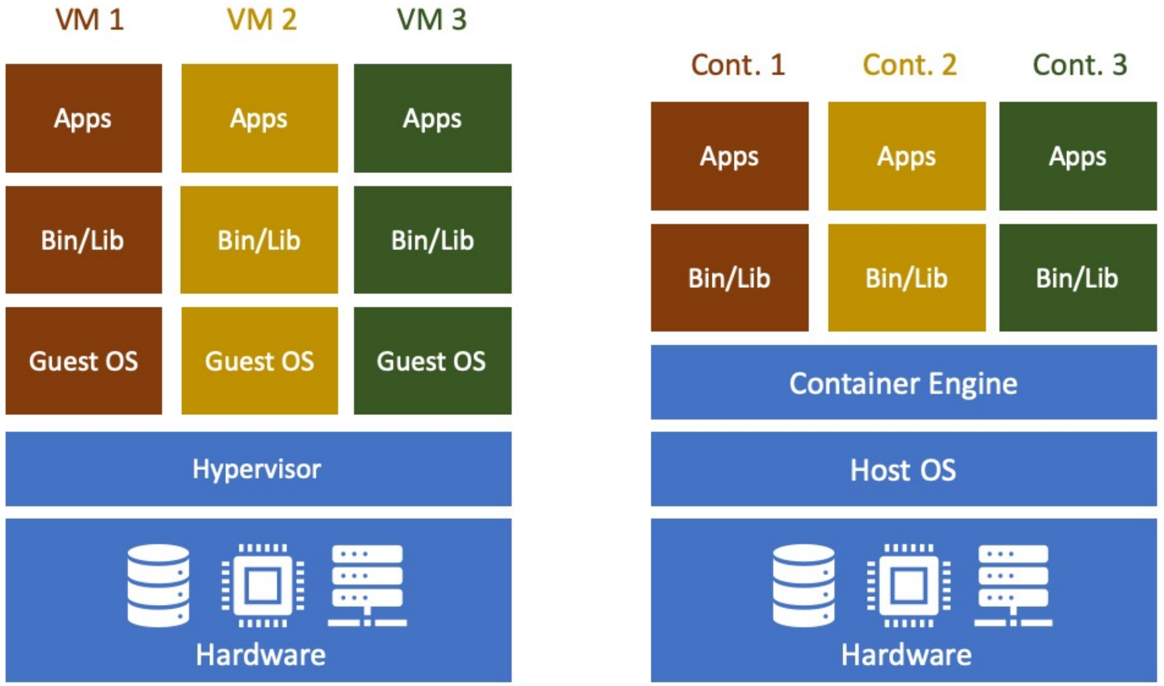
Containers can be thought of as lightweight virtualisations

- are less separate from the host compared to virtual machines
- share the kernel of the host OS
- are a combination of Linux namespaces and controlgroups (cgroups)

Container OS must be compatible with the host OS kernel

- for HPC, container OS must be based on Linux, e.g., Ubuntu

Containers are Lightweight Virtualisations



Virtual Machines (VMs)

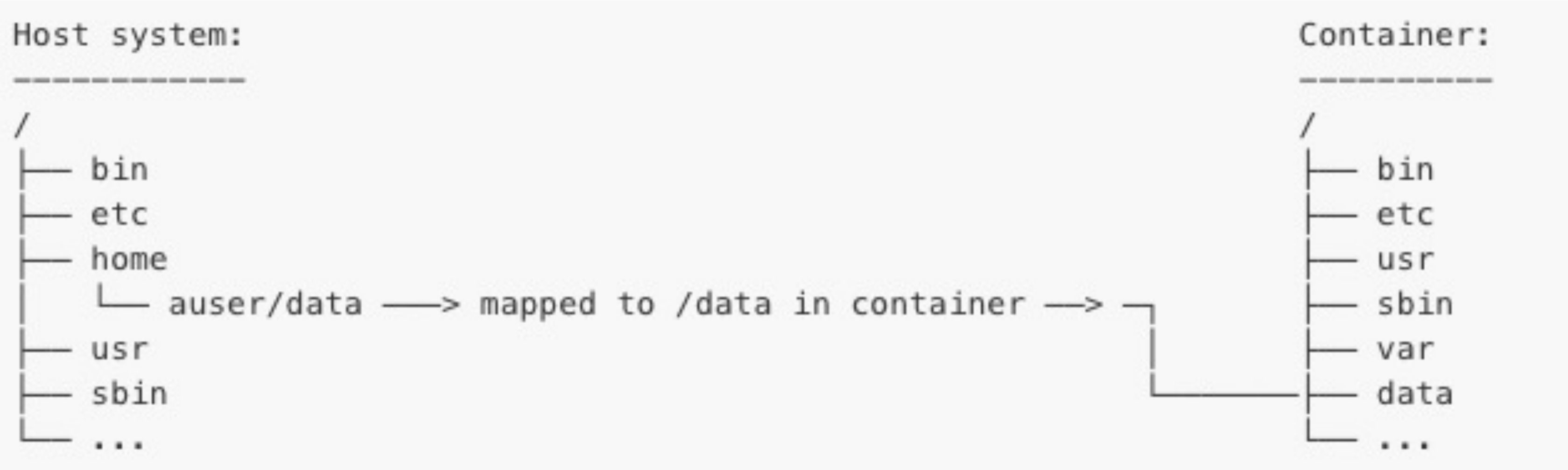
Containers

David Eyers, Sarah Stevens, Andy Turner and Jeremy Cohen
Containers for reproducible research
https://imperialcollegelondon.github.io/2020-07-13-Containers-Online/01-introduction/index.html

Containers and File Systems

Host FS

Container FS



```
Host system:                                                      Container:
------------                                                      ----------
/                                                                 /
├── bin                                                           ├── bin
├── etc                                                           ├── etc
├── home                                                          ├── usr
│   └── auser/data ───> mapped to /data in container ──> ─┐       ├── sbin
├── usr                                                   │       ├── var
├── sbin                                                  └───────├── data
└── ...                                                           └── ...
```

David Eyers, Sarah Stevens, Andy Turner and Jeremy Cohen
Containers for reproducible research
https://imperialcollegelondon.github.io/2020-07-13-Containers-Online/01-introduction/index.html

Singularity maps particular directories into the container by default, e.g., `$HOME`, `/etc/passwd`, `/tmp`.

Why Singularity (and not Docker)?

Singularity images are handled as (SIF) files where an image is instantiated as a container.

Why Singularity (and not Docker)?

Singularity images are handled as (SIF) files where an image is instantiated as a container.

Singularity can be run entirely within “user space”: no administrative-level privileges required to run containers on a platform where Singularity has been installed.

Why Singularity (and not Docker)?

Singularity images are handled as (SIF) files where an image is instantiated as a container.

Singularity can be run entirely within “user space”: no administrative-level privileges required to run containers on a platform where Singularity has been installed.

Singularity can support natively high-performance interconnects, such as InfiniBand and Intel Omni-Path Architecture (OPA).

Singularity is designed for parallel execution.

When does Singularity need root permissions?

Installation

- unless you configure with "–`without-setuid`" option
  - all containers must be run within sandbox directories
  - https://sylabs.io/guides/3.8/admin-guide/user_namespace.html - userns-limitations

When does Singularity need root permissions?

Installation

- unless you configure with "–`without-setuid`" option
  - all containers must be run within sandbox directories
  - https://sylabs.io/guides/3.8/admin-guide/user_namespace.html - userns-limitations

Building Containers

- Linux distros package software to be installed by root
- need a container "factory": a Linux host where you have root permissions
- compiling code on one machine but running on another has challenges
  a) less performant
  b) compiler availability

Singularity and MPI

Ideally, we'd build the container such that it contains an MPI implementation that is identical to the implementation running on the host.

The MPI library running on the host and in the container have to be at least ABI compatible*.

*The ABI compatibility initiative is an understanding between various MPICH derived MPI implementations (MPICH, Intel MPI and Cray MPT) to maintain runtime compatibility between each other.

Singularity and MPI

Ideally, we'd build the container so that it contains a version of OpenMPI that is identical to the OpenMPI running on the host.

The MPI library running on the host and in the container have to be at least ABI compatible.

Singularity has two solutions for integrating the container and the host with respect to MPI.

Singularity Hybrid Model

The MPI installation in the container links back to the MPI installation on the host.

```
[host]$ mpirun ... singularity exec ... /path/to/container/sif /path/to/mpiapp ...
```

Singularity Hybrid Model

The MPI installation in the container links back to the MPI installation on the host.

```
[host]$ mpirun ... singularity exec ... /path/to/container/sif /path/to/mpiapp ...
```

- Parallel Job Launcher (e.g., `mpirun`)
  - Process Management Daemon, ORTED
    - Singularity
      - Container and namespace environment
      - MPI application within container
        - MPI libraries
          - *use PMI to connect back to* ORTED

Singularity Hybrid Model

The MPI installation in the container links back to the MPI installation on the host.

```
[host]$ mpirun ... singularity exec ... /path/to/container/sif /path/to/mpiapp ...
```

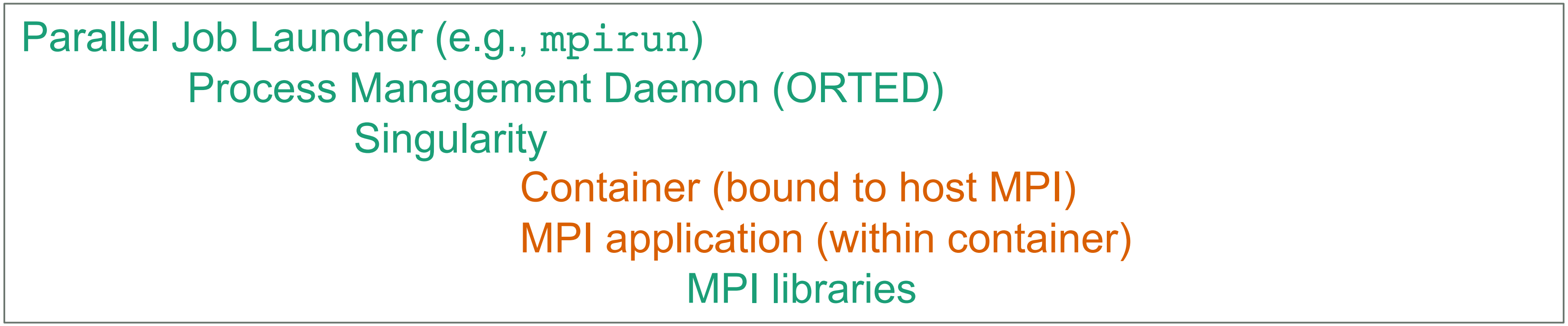
- Parallel Job Launcher (e.g., `mpirun`)
  - Process Management Daemon, ORTED
    - Singularity
      - Container and namespace environment
      - MPI application within container
        - MPI libraries
          - use PMI to connect back to ORTED

- Container MPI must be compatible with host MPI.
- Container MPI must be configured for host hardware if performance is critical.

Singularity Bind Model

No container MPI instead Singularity mounts/binds the host MPI in/to the container.

```
[host]$ mpirun ... singularity exec ... /path/to/container/sif /path/to/mpiapp ...
```



Parallel Job Launcher (e.g., `mpirun`)

Process Management Daemon (ORTED)

Singularity

Container (bound to host MPI)

MPI application (within container)

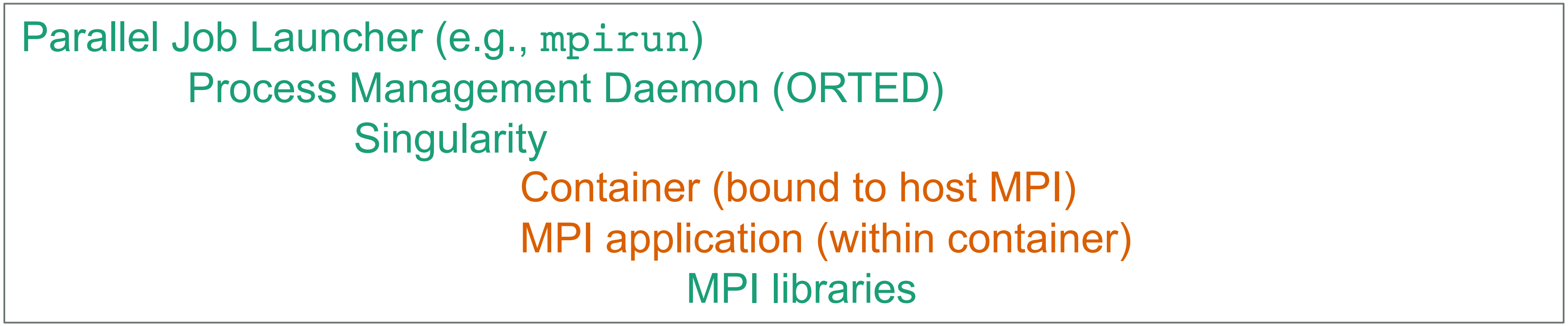
MPI libraries

- MPI configuration should be optimal for host.
- Container easier to build (compared to hybrid)
- Container MPI app must be compatible with host MPI.

Singularity Bind Model

No container MPI instead Singularity mounts/binds the host MPI in/to the container.

```
[host]$ mpirun ... singularity exec ... /path/to/container/sif /path/to/mpiapp ...
```



Parallel Job Launcher (e.g., `mpirun`)

Process Management Daemon (ORTED)

Singularity

Container (bound to host MPI)

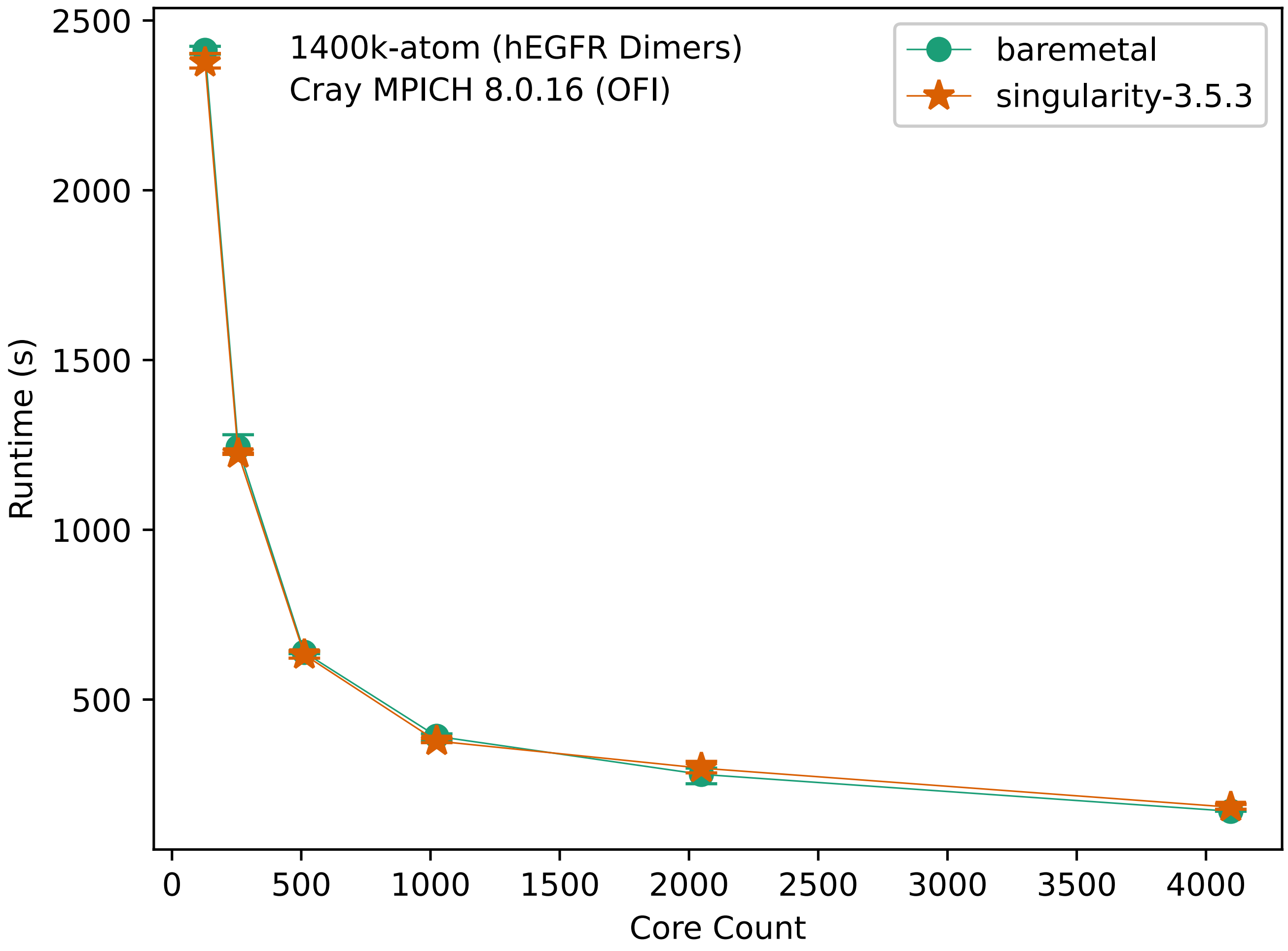
MPI application (within container)

MPI libraries

- MPI configuration should be optimal for host.
- Container easier to build (compared to hybrid)
- Container MPI app must be compatible with host MPI.

- Forthcoming examples use the **bind model**.

GROMACS 2021.1 on ARCHER2 4cab (strong scaling)



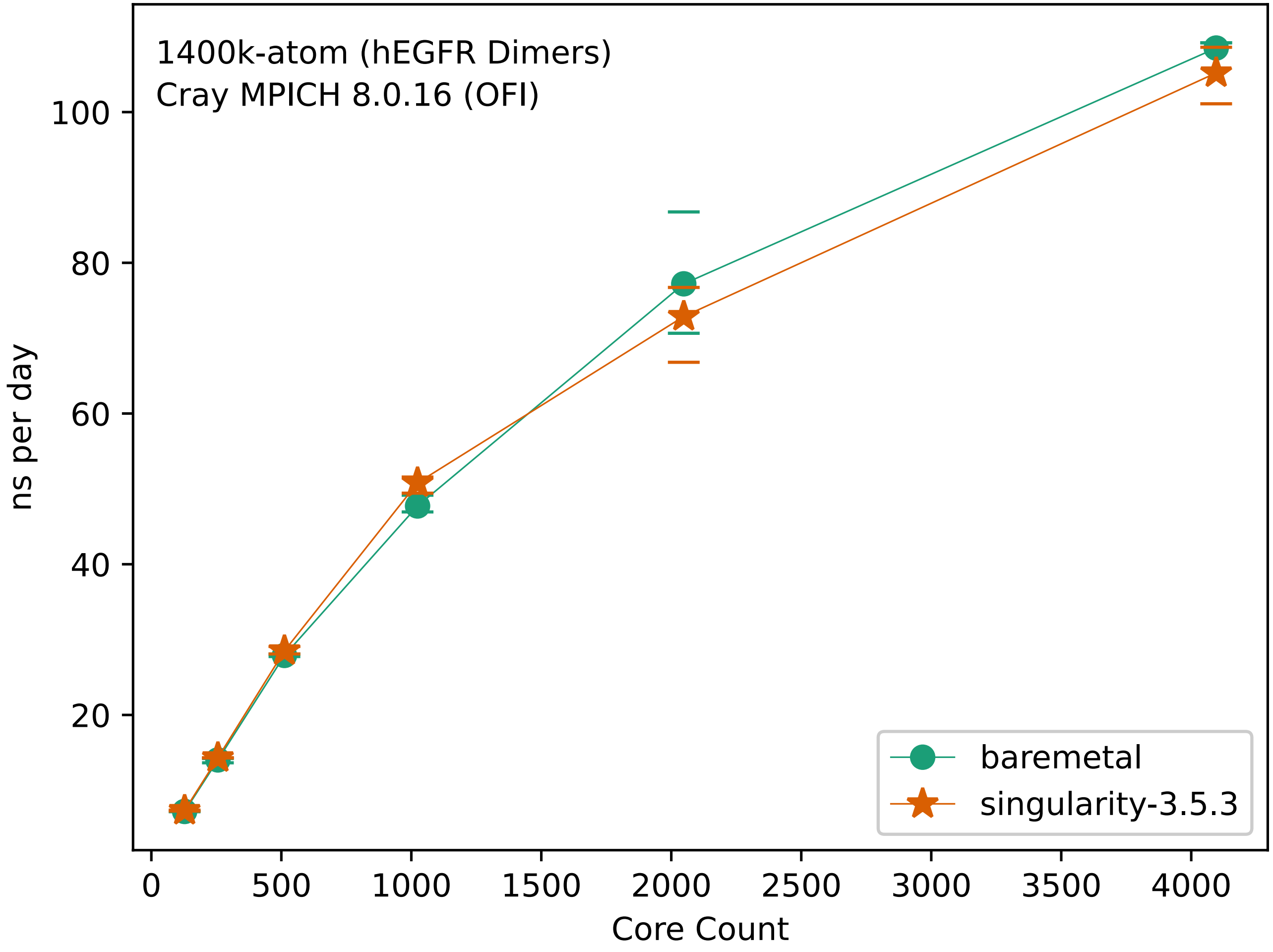
ARCHER2 (4cab)

HPE Cray EX
AMD (EPYC 7742)
128 cores per node
256 GB mem

HPE Slingshot
200 Gb/s

www.archer2.ac.uk

GROMACS 2021.1 on ARCHER2 4cab (strong scaling)



ARCHER2 (4cab)

HPE Cray EX
AMD (EPYC 7742)
128 cores per node
256 GB mem

HPE Slingshot
200 Gb/s

www.archer2.ac.uk

A Larger GROMACS Benchmark

4000k atoms (protein in water) running on 64 ARCHER2 nodes (8192 cores). Average performance was 9.47 ns per day for containerized GROMACS and 9.23 for baremetal.

These noticeable albeit small differences in performance are surprising given that both code instances were built with the same libraries (Cray MPICH v8.0.16 and Cray FFTW v3.3.8.8) using GCC v10 compilers.

Possible that node assignment is a factor here: differences in compute node performance outweigh any overhead due to containerization.

Launching Containerized GROMACS (srun)

🟢 host

🟠 container

```
#!/bin/bash —login

#SBATCH -J sc_gromacs
...


# setup environment variables

# setup singularity bindpaths

# setup singularity environment


...

srun --distribution=block:block --hint=nomultithread --chdir=${APP_RUN_PATH} \

    singularity exec -B ${BIND_ARGS} --env-file ${APP_RUN_PATH}/env.sh  ${SIF} \

        ${APP_EXE} ${APP_PARAMS} &>> ${APP_OUTPUT}

...
```

Launching Containerized GROMACS (singularity bindpaths)

● host

● container

```
#!/bin/bash –login

#SBATCH -J sc_gromacs
...


# setup singularity bindpaths
APP_SCRIPTS_ROOT=/opt/scripts/app/gromacs/host/archer2
BIND_ARGS=`singularity exec ${SIF} cat ${APP_SCRIPTS_ROOT}/bindpaths.lst`
BIND_ARGS=${BIND_ARGS},/var/spool/slurmd/mpi_cray_shasta,${APP_RUN_ROOT}


...

srun --distribution=block:block --hint=nomultithread --chdir=${APP_RUN_PATH} \

    singularity exec -B ${BIND_ARGS} --env-file ${APP_RUN_PATH}/env.sh  ${SIF} \

        ${APP_EXE} ${APP_PARAMS} &>> ${APP_OUTPUT}

...
```

Launching Containerized GROMACS (singularity bindpaths)

🟢 host

🟠 container

```
#!/bin/bash –login

#SBATCH -J sc_gromacs
...


# setup singularity bindpaths
APP_SCRIPTS_ROOT=/opt/scripts/app/gromacs/host/archer2
BIND_ARGS=`singularity exec ${SIF} cat ${APP_SCRIPTS_ROOT}/bindpaths.lst`
BIND_ARGS=${BIND_ARGS},/var/spool/slurmd/mpi_cray_shasta,${APP_RUN_ROOT}

    /work/y07/shared,/opt/cray,/usr/lib64:/usr/lib64/host,/etc/libibverbs.d
...

srun --distribution=block:block --hint=nomultithread --chdir=${APP_RUN_PATH} \

    singularity exec -B ${BIND_ARGS} --env-file ${APP_RUN_PATH}/env.sh  ${SIF} \

        ${APP_EXE} ${APP_PARAMS} &>> ${APP_OUTPUT}

...
```

Launching Containerized GROMACS (singularity env)

● host

● container

```
#!/bin/bash –login

#SBATCH -J sc_gromacs
...


# setup singularity environment
APP_SCRIPTS_ROOT=/opt/scripts/app/gromacs/host/archer2
singularity exec ${SIF} \
    cat ${APP_SCRIPTS_ROOT}/cmpich8-ofi/gcc10/env.sh > ${APP_RUN_PATH}/env.sh


...

srun --distribution=block:block --hint=nomultithread --chdir=${APP_RUN_PATH} \

    singularity exec -B ${BIND_ARGS} --env-file ${APP_RUN_PATH}/env.sh  ${SIF} \

        ${APP_EXE} ${APP_PARAMS} &>> ${APP_OUTPUT}

...
```

Launching Containerized GROMACS (singularity env)

● host

● container

```
#!/bin/bash –login

#SBATCH -J sc_gromacs
...


# setup singularity environment
APP_SCRIPTS_ROOT=/opt/scripts/app/gromacs/host/archer2
singularity exec ${SIF} \
    cat ${APP_SCRIPTS_ROOT}/cmpich8-ofi/gcc10/env.sh > ${APP_RUN_PATH}/env.sh

sed -i -e 's/LD_LIBRARY_PATH/export SINGULARITYENV_LD_LIBRARY_PATH/g’ \
    ${APP_RUN_PATH}/env.sh
. ${APP_RUN_PATH}/env.sh


...

srun --distribution=block:block --hint=nomultithread --chdir=${APP_RUN_PATH} \

    singularity exec -B ${BIND_ARGS}  ${SIF} \

        ${APP_EXE} ${APP_PARAMS} &>> ${APP_OUTPUT}

...
```

Older versions of Singularity (e.g., v3.5.3) do not support the “--env-file” option.

Launching Containerized GROMACS (singularity env)

```
# /opt/scripts/app/gromacs/host/archer2/cmpich8-ofi/gcc10/env.sh


MPI_ROOT=/opt/cray/pe/mpich/8.0.16/ofi/gnu/9.1
...

FFTW_ROOT=/opt/cray/pe/fftw/3.3.8.8/x86_rome

LIBSCI_ROOT=/opt/cray/pe/libsci/20.10.1.2/GNU/9.1/x86_64
BLAS_LIBRARIES=${LIBSCI_ROOT}/lib/libsci_gnu_82_mpi_mp.so
LAPACK_LIBRARIES=${BLAS_LIBRARIES}

LD_LIBRARY_PATH=${FFTW_ROOT}/lib:${LIBSCI_ROOT}/lib:${MPI_ROOT}/lib: \
    /opt/cray/pe/lib64:/opt/cray/libfabric/1.11.0.0.233/lib64: \
    /usr/lib64/host:/usr/lib64/host/libibverbs: \
    /lib/x86_64-linux-gnu:/.singularity.d/libs

...
```

● host

● container

Container Factory

A container factory can be setup as an instance within the UoE's Research Cloud Service, **Eleanor**, based on OpenStack.

https://www.ed.ac.uk/information-services/research-support/research-computing/ecdf/cloud

A user automatically has root access to any cloud instance that is created from within their Eleanor account.

Why not setup a container factory on your personal laptop?

- establishing the factory as a cloud-based instance separates that work from any details peculiar to an individual's machine
- the building of a factory can also be scripted allowing others to create their own container factories

https://github.com/mbareford/container-factory

Eleanor Horizon

Factory instance has 8 vCPUs, 16 GB RAM and 160 GB of disk space.

At present, factory OS is Ubuntu 20.04.2 and the container software is (Sylabs) SingularityCE 3.8.3.

Creating the Initial Container Image

The making of an application-specific container takes place within the factory builds folder, e.g., `~/work/builds/gromacs`.

```
#!/bin/bash --login

# bundle various scripts into a tar archive such that it is
# accessible to the container definition file
...

sudo singularity build gromacs.sif.0 \
    ${HOME}/work/scripts/def/gromacs.def &> create.log
```

● factory

Singularity Container Definition File

https://sylabs.io/guides/3.8/user-guide/definition_files.html

```
Bootstrap: library
From: ubuntu:20.04

...

%files
    ~/work/scripts/post_start.sh /opt/
    ~/work/scripts/post_stop.sh /opt/
    ~/work/builds/gromacs/scripts.tar.gz /opt/

...

%post
    . /opt/post_start.sh

    ubuntu-20.04.sh 10

    miniconda.sh 3 4.8.3 38
    conda_install.sh numpy,scipy,matplotlib

    cmake.sh 3.18.4

    source.sh gromacs 2021.1

    . /opt/post_stop.sh

...
```

```
Boostrap: library
From: ubuntu:20.04

%setup...

%files...

%environment...

%post...

%runscript...

%startscript...

%test...

%labels...

%help...
```

● factory

● container

Container OS Script for Ubuntu 20.04

● container

```
ubuntu-20.04.sh 10
```

```
#!/bin/bash

echo "deb http://archive.ubuntu.com/ubuntu \
      focal main restricted universe multiverse" > /etc/apt/sources.list
...

apt-get -y install build-essential uuid-dev libssl-dev libseccomp-dev \
                   libgpgme-dev zlib1g-dev iputils-ping squashfs-tools \
                   m4 lzip liblz4-1 wget curl git \
                   ...
...

if [ !z “${1}” ]; then
    apt-get -y install gcc-${1} g++-${1} gfortran-${1}
    ...
fi
```

Container Source Script for GROMACS

● container

```
source.sh gromacs 2021.1
```

```
#!/bin/bash

VERSION=$2
LABEL=$1
NAME=${LABEL}-${VERSION}
ROOT=/opt/app/${LABEL}

mkdir -p ${ROOT}
cd ${ROOT}

wget https://ftp.gromacs.org/${LABEL}/${NAME}.tar.gz
tar -xzf ${NAME}.tar.gz
rm ${NAME}.tar.gz
```

Container Provenance

● factory

● container

```
singularity inspect -H gromacs.sif.0
```

```
This GROMACS (http://www.gromacs.org/) container image file was created at the EPCC Container Factory,
an OpenStack Ubuntu 20.04 instance (ID 859596f3-6683-4951-82d4-f9e080c30d1f) hosted by the University
of Edinburgh Eleanor Research Cloud.

The container is based on Ubuntu 20.04 and features GCC 10.3.0, Miniconda3 4.8.3, CMake 3.18.4 and the
GROMACS source code version 2021.1.

See the container creation log at "/opt/logs/create.log.0" and the original definition file at
"/opt/scripts/def/gromacs.def".

Submission script templates can be found under "/opt/scripts/app/gromacs/host/". These script files are
named "submit.sh" and are organised by "<host name>/<MPI library>/<compiler>".
```

```
/.singularity.d/runscript.help
```

```
singularity exec gromacs.sif.0 cat /opt/logs/create.log.0
```

The provenance history grows every time the containerized application is built on (or targeted at) a HPC platform.

Targeting the Container (the top-level command)

🟣 factory

🟢 host

```
target.sh ~/work/scripts ${PWD} gromacs \
    archer2 /work/z19/z19/mrb4cab/containers/build \
    "2021.1 cmpich8-ofi gcc10"
```

Targeting the Container (the top-level command)

🟣 factory

🟢 host

```
target.sh ~/work/scripts ${PWD} gromacs \
    archer2 /work/z19/z19/mrb4cab/containers/build \
    "2021.1 cmpich8-ofi gcc10"
```

Recent versions of Singularity (>= 3.7.x) require that bind paths already exist within the container before those same paths can be used with the –**B** option.

```
singularity exec -B ${BIND_ARGS} --writable ${SIF}.sandbox ...
```

An example: for Cirrus, it is necessary to run a "`target_init.sh`" script that creates the "**`/lustre/sw`**", "**`/opt/hpe`**" and "**/etc/libibverbs.d**" file paths within the container.

Targeting the Container (the target script)

● factory

● host

```
# target.sh


# upload singularity image from factory to host
scp gromacs.sif.0 ...


# run deployment script on host
...


# download new singularity image from host to factory
scp archer2:${DEPLOY_PATH}/gromacs.sif.1 ...
```

Targeting the Container (the target script)

● factory

● host

```
# target.sh


# upload singularity image from factory to host
scp gromacs.sif.0 ...


# run deployment script on host
DEPLOY_SCRIPT=~/work/scripts/app/gromacs/host/archer2/deploy.sh
DEPLOY_PATH=/work/z19/z19/mrb4cab/containers/build
DEPLOY_ARGS=“gromacs ${DEPLOY_PATH} 2021.1 cmpich8-ofi gcc10”

ssh archer2 “bash –ls” < ${DEPLOY_SCRIPT} ${DEPLOY_ARGS}


# download new singularity image from host to factory
scp archer2:${DEPLOY_PATH}/gromacs.sif.1 ...
```

Targeting the Container (the deploy script)

● host

● container

```
# deploy.sh

...


# convert singularity image to sandbox
singularity build --sandbox ${SIF}.sandbox ${SIF}


# build app within container sandbox
...


# convert singularity sandbox back to image
singularity build --force ${SIF} ${SIF}.sandbox
```

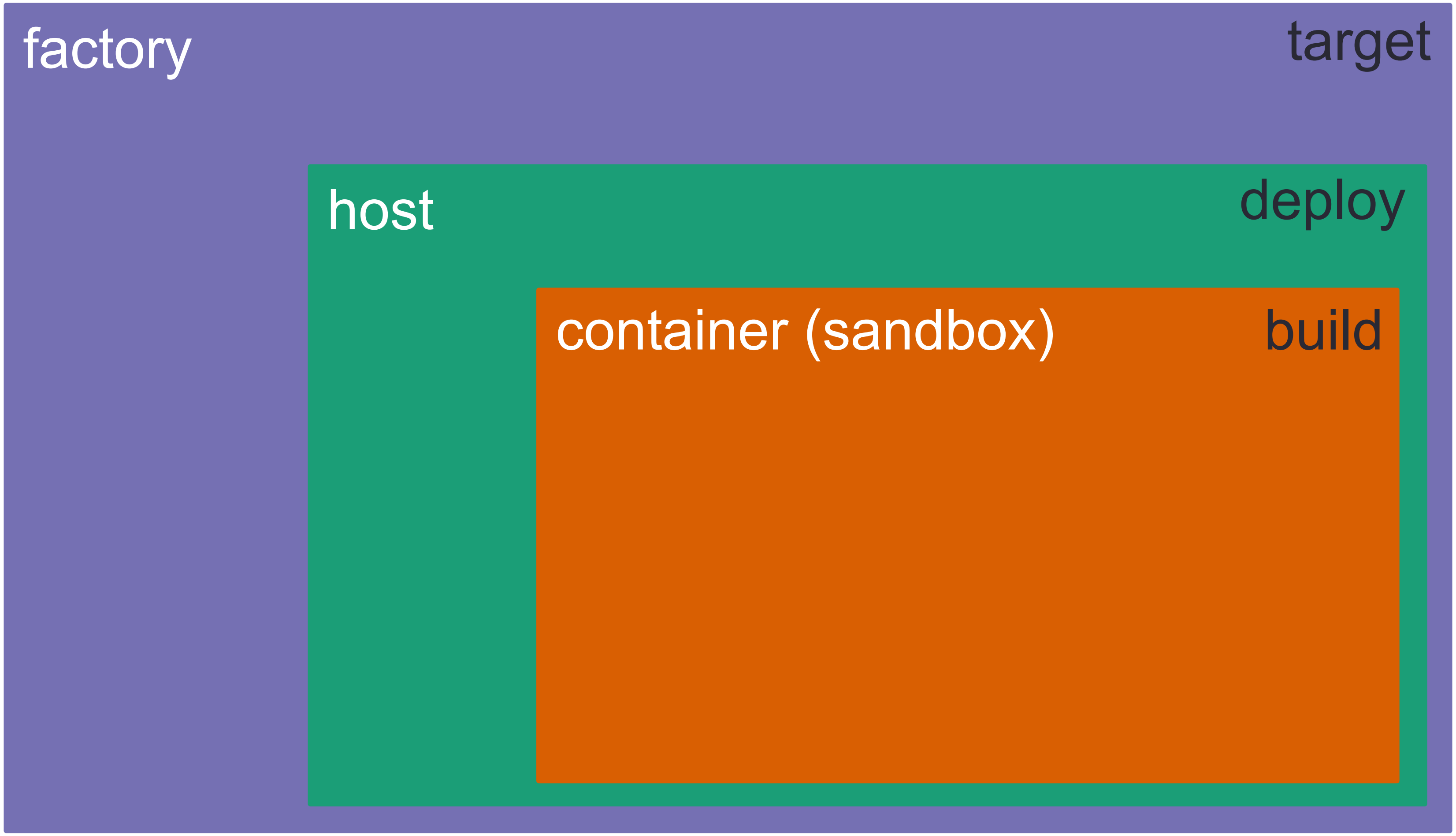
Targeting the Container (the deploy script)

🟢 host

🟠 container

```
# deploy.sh

...


# convert singularity image to sandbox
singularity build --sandbox ${SIF}.sandbox ${SIF}


# build app within container sandbox
singularity exec -B ${BIND_ARGS} --writable ${SIF}.sandbox
    /opt/scripts/app/gromacs/build.sh 2021.1 cmpich8-ofi gcc10


# convert singularity sandbox back to image
singularity build --force ${SIF} ${SIF}.sandbox
```

Targeting the Container (the whole process)



Container Provenance after Targeting

`singularity inspect -H` **`gromacs.sif.2`** ≈ 1.8 GB

```
This GROMACS (http://www.gromacs.org/) container image file was created at the EPCC Container Factory,
an OpenStack Ubuntu 20.04 instance (ID 859596f3-6683-4951-82d4-f9e080c30d1f) hosted by the University
of Edinburgh Eleanor Research Cloud.

The container is based on Ubuntu 20.04 and features GCC 10.3.0, Miniconda3 4.8.3, CMake 3.18.4 and the
GROMACS source code version 2021.1.

See the container creation log at "/opt/logs/create.log.0" and the original definition file at
"/opt/scripts/def/gromacs.def".

Submission script templates can be found under "/opt/scripts/app/gromacs/host/". These script files are
named "submit.sh" and are organised by "<host name>/<MPI library>/<compiler>".


2021-09-25 12:39:53: Built gromacs 2021.1 (cmpich8-ofi-gcc10) on archer2 (/opt/logs/make.log.1)

2021-09-25 13:42:56: Built gromacs 2021.1 (ompi4-ofi-gcc10) on archer2 (/opt/logs/make.log.2)

2021-09-29 12:28:37: Built gromacs 2021.1 (mpt2-ib-gcc10) on cirrus  (/opt/logs/make.log.3)
```

`/.singularity.d/runscript.help`

Targeting the Container (the **Cirrus** deploy script)

● host

● container

```
# deploy.sh

...

# convert singularity image to sandbox
singularity build --sandbox ${SIF}.sandbox ${SIF}

# copy mellanox drivers to container sandbox
LIBMLX_HOST=${SIF}.sandbox/lib/x86_64-linux-gnu/libmlx-cirrus
mkdir -p ${LIBMLX_HOST}
cp /lib64/libmlx* ${LIBMLX_HOST}/
cp /lib64/libib* ${LIBMLX_HOST}/
...

# build app within container sandbox
...

# convert singularity sandbox back to image
singularity build --force ${SIF} ${SIF}.sandbox
```

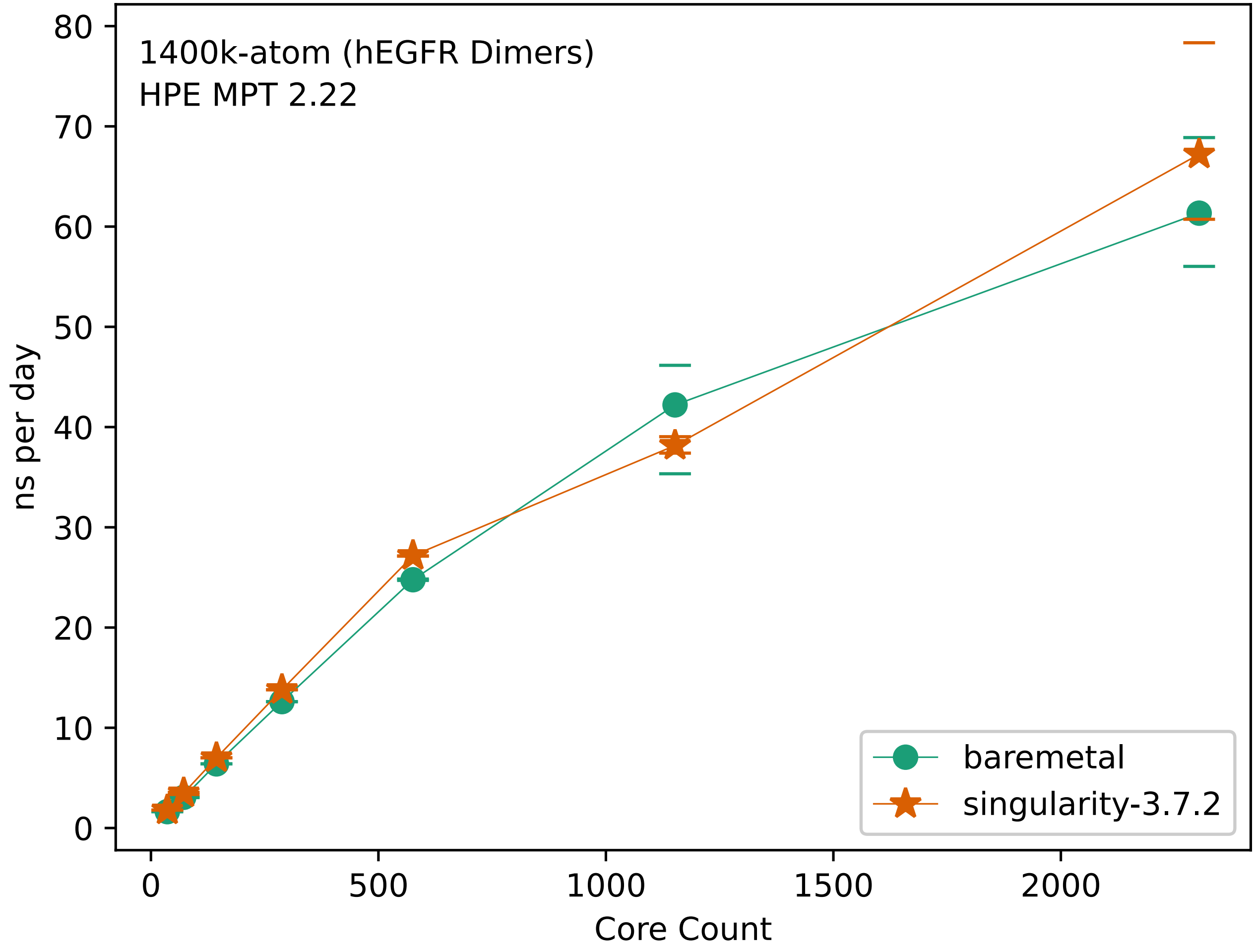
Targeting the Container (the **Cirrus** deploy script)

● host

● container

```
# deploy.sh

...

# convert singularity image to sandbox
singularity build --sandbox ${SIF}.sandbox ${SIF}

# copy mellanox drivers to container sandbox
...

# build app within container sandbox
BIND_ARGS=/lustre/sw:/opt/hpe:/etc/libibverbs.d
singularity exec -B ${BIND_ARGS} --no-home --writable ${SIF}.sandbox
    /opt/scripts/app/${APP}/build.sh 2021.1 mpt2-ib gcc10

# convert singularity sandbox back to image
singularity build --force ${SIF} ${SIF}.sandbox
```

GROMACS 2021.1 on Cirrus (strong scaling)



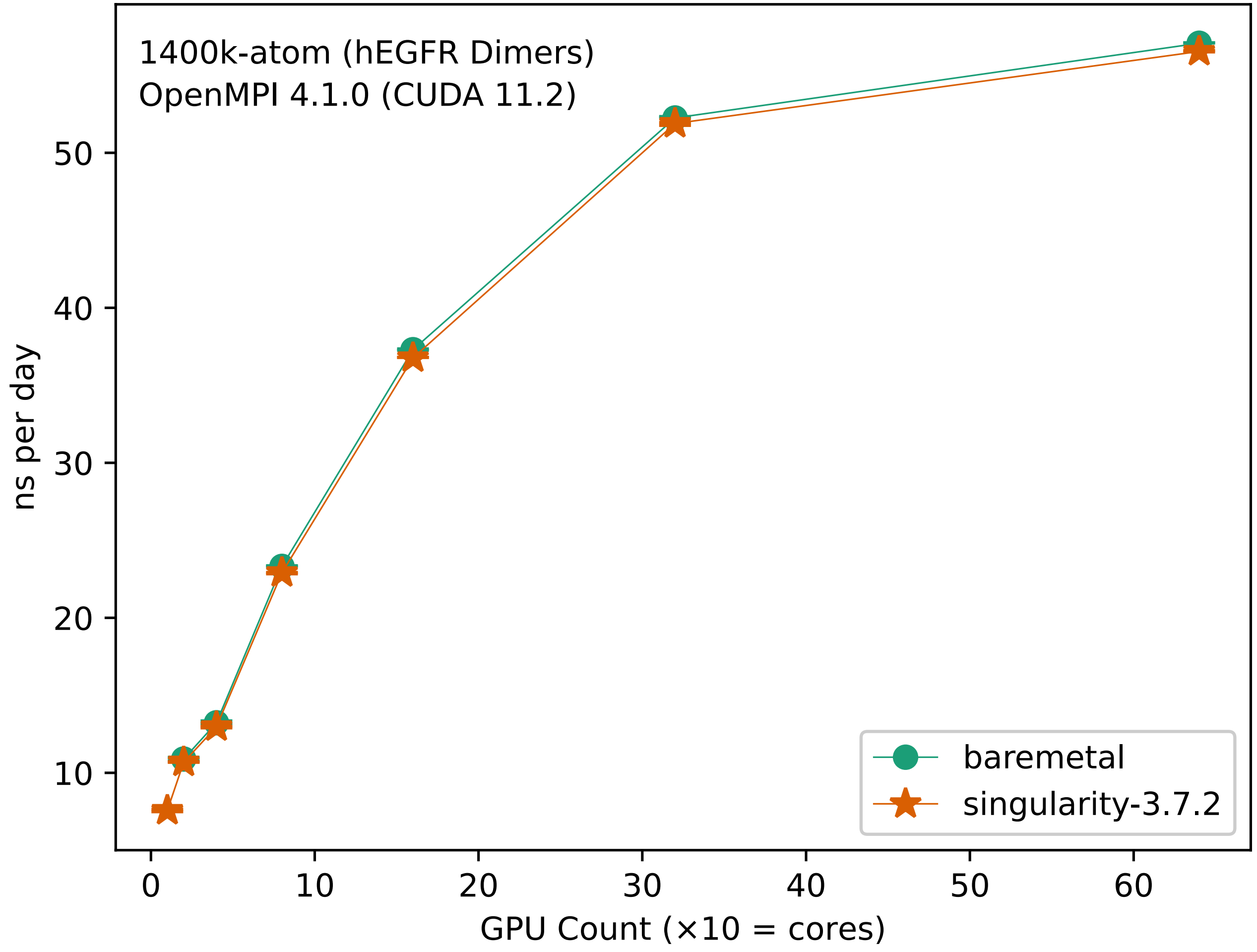
Cirrus

SGI ICE XA
Intel Xeon (Broadwell)
36 cores per node
256 GB mem

Infiniband (FDR)
54.5 Gb/s

www.cirrus.ac.uk

GROMACS 2021.1 on Cirrus GPU (strong scaling)



Cirrus GPU

SGI ICE XA
Intel Cascade Lake
40 cores per node
384 GB mem

4 GPUs per node
NVIDIA Tesla
(V100-SXM2-16GB)

Infiniband (FDR)
54.5 Gb/s

www.cirrus.ac.uk

```
srun ... singularity exec —nv ... ${SIF} ...
```

Conclusions

No significant difference between containerized and baremetal performance... so far...

Similar results seen with CASTEP (materials modelling code) and RAMSES (astrophysical code) on ARCHER2 4cab.

https://github.com/mbareford/container-factory

Conclusions

No significant difference between containerized and baremetal performance.... so far...

Similar results seen with CASTEP (materials modelling code) and RAMSES (astrophysical code) on ARCHER2 4cab.

Other compilers?

Code compilation done using GNU compiler installed within container. What about compilers that might require a licence?

Is it possible to build using compiler on host (e.g., Cray or Intel), thereby avoiding licence restrictions?

https://github.com/mbareford/container-factory

Conclusions

No significant difference between containerized and baremetal performance.... so far...

Similar results seen with CASTEP (materials modelling code) and RAMSES (astrophysical code) on ARCHER2 4cab.

ARM Platform?

Container images compatible with x86-64 (amd64) processor architecture only. Separate image file necessary for ARM machines.

SingularityCE provide a remote build facility.

```
    singularity build --remote --arch=arm64 ...
```

https://github.com/mbareford/container-factory

Conclusions

No significant difference between containerized and baremetal performance.... so far...

Similar results seen with CASTEP (materials modelling code) and RAMSES (astrophysical code) on ARCHER2 4cab.

Persistent Overlays?

Is it possible to use an overlay as a sort of *lens* that when applied to a base image allows a containerized app to run on a particular HPC platform?

This would make an interesting MSc project.

https://github.com/mbareford/container-factory

Q&A

To pose a question, you can write your question in the "Questions" tab

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 675451

Thank you for participating!

…don’t forget to fill in our feedback questionnaire…

Visit the CompBioMed website (www.compbiomed.eu/training)
for a full recording of this and other e-Seminars,
to download the slides
and to keep updated on our upcoming trainings

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 675451

https://insilicoworld.slack.com/archives/C0151M02TA4

The e-Seminar series is run in collaboration with: