

THE STATE OF CONTAINERS

On HPC Systems where I have played with Singularity containers

Michael Bareford



THE UNIVERSITY
of EDINBURGH



Contents

1. Containers and Singularity
2. Container Factory (UoE specific)
3. Running Containers on a HPC Platform
4. Baremetal Performance Comparisons
(using openmpi v4 and gcc)
5. Summary...

What is a Container?

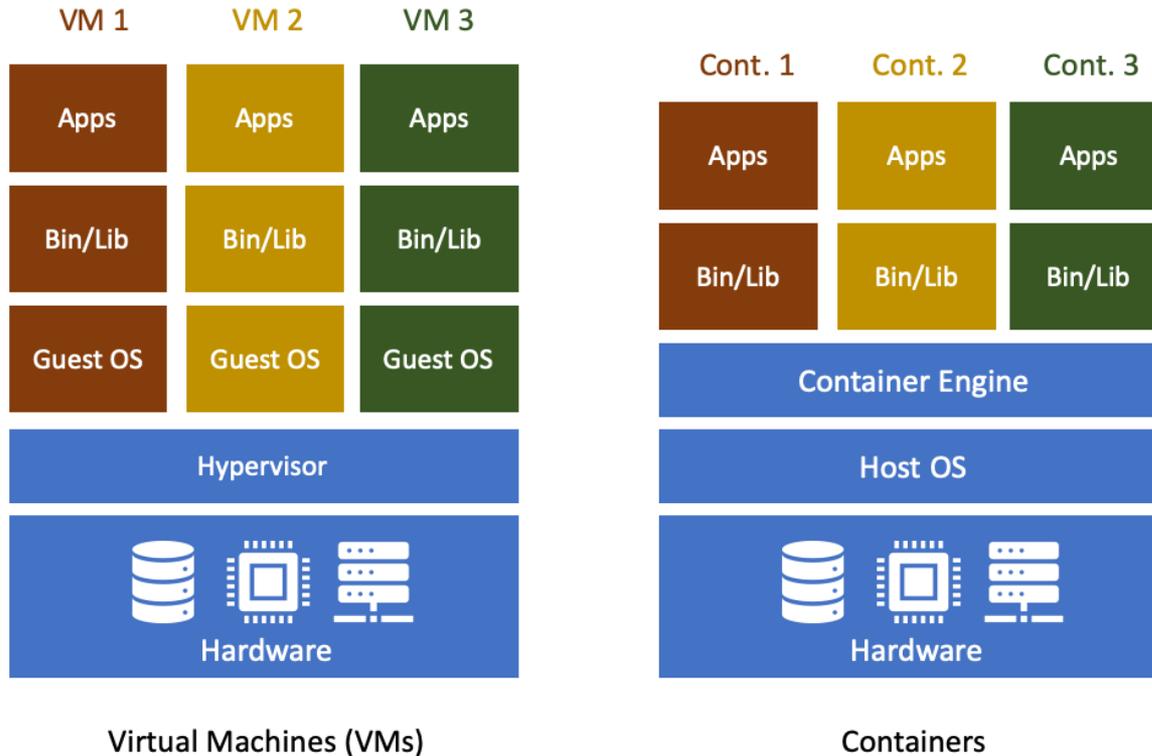
Containers can be thought of as lightweight virtualisations

- 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



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

Host system:

```

-----
/
├── bin
├── etc
├── home
│   └── auser/data
├── usr
├── sbin
└── ...
  
```

└── auser/data → mapped to /data in container →

Container FS

Container:

```

-----
/
├── bin
├── etc
├── usr
├── sbin
├── var
├── 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/3.6/admin-guide/user_namespace.html#usersns-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/3.6/admin-guide/user_namespace.html#usersns-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 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*.

*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 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 MPI app must be compatible with host MPI.
- Forthcoming examples use the **hybrid model**.

Container Factory

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

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

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

<https://www.wiki.ed.ac.uk/display/ResearchServices/Eleanor>

Please note, links may not be accessible to users outside UoE.

Container Factory

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

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

Once you have setup an instance (running Ubuntu 19.10 say), you can then create an *instance* image (or snapshot), which can then be made public or shared with specific projects (or users).

Eleanor Research Cloud Service

All UoE staff have access to the “**Free Tier**” level of resource, e.g., 1 vCPU, 1 GB memory, 20 GB disk (**t1.small** flavor).

First, you need to create a *free* new cloud project within your **ECDF* Storage Manager** Account.

*Edinburgh Compute and Data Facility

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

Eleanor Research Cloud Service

All UoE staff have access to the “**Free Tier**” level of resource, e.g., 1 vCPU, 1 GB memory, 20 GB disk (**t1.small** flavor).

First, you need to create a *free* new cloud project within your **ECDF Storage Manager** Account.

Eleanor Cloud instances are managed via web-based interface called **Horizon**.

Eleanor Horizon

<https://horizon.ecdf.ed.ac.uk/dashboard/project/instances/>

(link not accessible to non-UoE personnel)

openstack ed • ft_mbarefor mbarefor

Project / Compute / Instances

Instances

Instance ID = Filter Launch Instance (Quota exceeded) Delete Instances More Actions

Displaying 1 item

<input type="checkbox"/>	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
<input type="checkbox"/>	container-factory-my_instance-f4noi643rlzf	Ubuntu 18.04	192.168.1.7, 172.16.50.15	t1.small	mbcloudkey	Active	nova	None	Running	1 year, 1 month	Create Snapshot

Displaying 1 item

https://horizon.ecdf.ed.ac.uk/dashboard/home/

```
ssh -i mbcloudkey.pem ubuntu@172.16.50.15
```

Eleanor Command Line

1. Create your own public-private key pair.
2. Download your Eleanor user credentials (OpenStack RC v3 file).
3. Setup a miniconda3 environment on your local machine, installing `python-heatclient` and `python-openstackclient` packages.

Eleanor Command Line

1. Create your own public-private key pair.
2. Download your Eleanor user credentials (OpenStack RC v3 file).
3. Setup a miniconda3 environment on your local machine, installing `python-heatclient` and `python-openstackclient` packages.

You can now setup a cloud instance from the command line.

```
. ./ft_mbarefor-openrc.sh  
openstack stack create -t ubuntu.yml container-factory
```

See next slide for example of a cloud instance script (`ubuntu.yml`).

Eleanor Instance Script (ubuntu.yml)

heat_template_version: 2015-04-30

description: Quickly create a Ubuntu instance and give it an IP address

resources:

```
my_instance:
  type: OS::Nova::Server
  properties:
    key_name: mbcloudkey
    image: Ubuntu 18.04
    flavor: t1.small
    networks:
      - network: VM Network Private
my_ip:
  type: OS::Nova::FloatingIP
  properties:
    pool: Floating Network Private (UoE access only)
assoc:
  type: OS::Nova::FloatingIPAssociation
  properties:
    floating_ip: { get_resource: my_ip }
    server_id: { get_resource: my_instance }
```

outputs:

```
instance_ip:
  description: IP address of the instance
  value: { get_attr: [my_ip, ip] }
```

Eleanor Command Line

1. Create your own public-private key pair.
2. Download your Eleanor user credentials (OpenStack RC v3 file).
3. Setup a miniconda3 environment on your local machine, installing `python-heatclient` and `python-openstackclient` packages.

You can now setup a cloud instance from the command line.

```
. ./ft_mbarefor-openrc.sh  
openstack stack create -t ubuntu.yml container-factory
```

Find external IP address at

<https://horizon.ecdf.ed.ac.uk/dashboard/project/instances/>

(link not accessible to non-UoE personnel)

```
ssh -i mbcloudkey.pem ubuntu@172.16.50.15
```

Building a Container at the Factory

Install Singularity 3.6.1 (and Go 1.14.1)

possible to support multiple Singularity versions

Build a container

```
DEFS=$HOME/work/scripts/def  
NAME=ramses-ubuntu19-openmpi4
```

```
sudo singularity build $NAME.sif \  
                        $DEFS/$NAME.def \  
                        &> $NAME.out &
```

- build takes around 30 mins
- resulting SIF is approx 600 MB
- **singularity** inspect -d ramses-ubuntu19-openmpi4.sif

Singularity Container Definition File

```
Bootstrap: library
From: ubuntu:19.10
```

https://sylabs.io/guides/3.6/user-guide/definition_files.html

```
...

%files
~/work/scripts/post_start.sh /opt/
~/work/scripts/post_stop.sh /opt/
~/work/builds/dirac/ramses/scripts.tar.gz /opt/
~/arc/apps/dirac/ramses-18.09.tar.gz /opt/

...

%post
. /opt/post_start.sh

ubuntu-19.10.sh

hwloc.sh 2.2.0
openmpi.sh 4.0.3

ramses.sh 18.09

. /opt/post_stop.sh

...
```

```
Bootstrap: library
From: ubuntu:19.10

%setup...

%files...

%environment...

%post...

%runscript...

%startscript...

%test...

%labels...

%help...
```

Container OS Script for Ubuntu 19.10

ubuntu-19.10.sh

```
#!/bin/bash

# ubuntu 19.10 is updated such that it supports infiniband comms

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

apt-get -y update
apt-get -y install build-essential uuid-dev libssl-dev \
                  libseccomp-dev libgpgme11-dev iputils-ping \
                  squashfs-tools wget git \
                  subversion pkg-config m4 \
                  gfortran zlib1g-dev vim \
                  bc autoconf autogen \
                  environment-modules libtool libibverbs-dev

apt-get -y upgrade
apt-get -y dist-upgrade

# install support for intel omni-path comms architecture
apt-get -y update
apt-get -y install opa-fm opa-fastfabric libpsm2-2 \
                  libpsm2-2-compat libpsm2-dev libpmix2

apt-get -y upgrade
apt-get -y dist-upgrade
```

Container Component Script for OpenMPI

openmpi.sh 4.0.3

```
#!/bin/bash
```

```
VERSION=$1
```

```
MAJOR_VERSION=`echo "${VERSION}" | cut -d"." -f 1-2`
```

```
NAME=openmpi-${VERSION}
```

```
ROOT=/opt/${NAME}
```

```
ARC_LINK=https://download.open-mpi.org/release/open-mpi/v${MAJOR_VERSION}/${NAME}.tar.gz
```

```
CFG_ARGS="CC=gcc CXX=g++ FC=gfortran --enable-mpi-compatibility  
--enable-mpi-fortran --with-verbs --with-hwloc=/opt/hwloc-2.2.0"
```

```
install_cmp.sh ${NAME} ${ROOT} ${ARC_LINK} "${CFG_ARGS}"
```

```
update_env.sh ${ROOT} OPENMPI_NAME ${NAME}
```

```
update_env.sh ${ROOT} OPENMPI_ROOT ${ROOT}
```

```
update_env.sh ${ROOT} MPI_HOME ${ROOT}
```

```
update_env.sh ${ROOT} MPI_RUN ${ROOT}/bin/mpirun
```

```
# add support for running hybrid mpi
```

```
update_env.sh ${ROOT} OMPI_DIR ${ROOT}
```

```
update_env.sh ${ROOT} SINGULARITY_OMPI_DIR ${ROOT}
```

```
echo ". ${ROOT}/env.sh" >> /opt/env.sh
```

```
echo ". ${ROOT}/env.sh" >> $SINGULARITY_ENVIRONMENT
```

Container Component Script for Ramses

ramses.sh 18.1

```
#!/bin/bash
```

```
VERSION=18.09
```

```
NAME=ramses
```

```
ROOT=/opt/apps/${NAME}
```

```
mkdir -p /opt/apps
```

```
mv /opt/ramses-${VERSION}.tar.gz /opt/apps/
```

```
cd /opt/apps
```

```
tar -xvzf ramses-${VERSION}.tar.gz
```

```
rm ramses-${VERSION}.tar.gz
```

```
if test -f "/opt/env.sh"; then
```

```
    . /opt/env.sh
```

```
fi
```

```
cd ramses/bin
```

```
make clean
```

```
make
```

```
make clean
```

```
update_env.sh ${ROOT} RAMSES_NAME ${NAME}
```

```
update_env.sh ${ROOT} RAMSES_ROOT ${ROOT}
```

```
echo ". ${ROOT}/env.sh" >> $SINGULARITY_ENVIRONMENT
```

Factory Sustainability (initial thoughts)

Scripts for Singularity definition files and container builds held on github.

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

Should be possible for other (EPCC) Eleanor users to launch factory instance from snapshot image.

Typical workflow would include the following.

- 1) login to personal cloud instance
- 2) fork and clone “[container-factory](#)” git repo
- 3) build containers
- 4) push new scripts to forked repo
- 5) create pull request for original (upstream) repo

At some point, “[container-factory](#)” repo will be moved to <https://github.com/EPCCed>

Factory Snapshot Image (initial thoughts)

Multiple Singularity/Go installations

Basic packages

- `build-essential, squashfs-tools, wget, git, vim`

ssh config info for uploading SIF files to HPC hosts

- individual factory users would need to create their own ssh keys within their own factory instance

Running Container on a Tier-2 HPC Host

Batch submission script

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

```
...
```

```
module load openmpi/4.0.3
```

```
module load singularity/3.6.1
```

```
...
```

```
MPIRUN_PREFIX_OPT="--prefix ${OPENMPI_ROOT}"
```

```
MPIRUN_RES_OPTS="-N ${NCORESPN} -n ${NCORES} --hostfile ${APP_RUN_PATH}/hosts --bind-to core"
```

```
MPIRUN_MCA_OPTS="--mca btl ^sm --mca btl_openib_allow_ib true"
```

```
MPIRUN_OPTS="${MPIRUN_PREFIX_OPT} ${MPIRUN_RES_OPTS} ${MPIRUN_MCA_OPTS}"
```

```
SINGULARITY_OPTS="exec -B /etc/libibverbs.d"
```

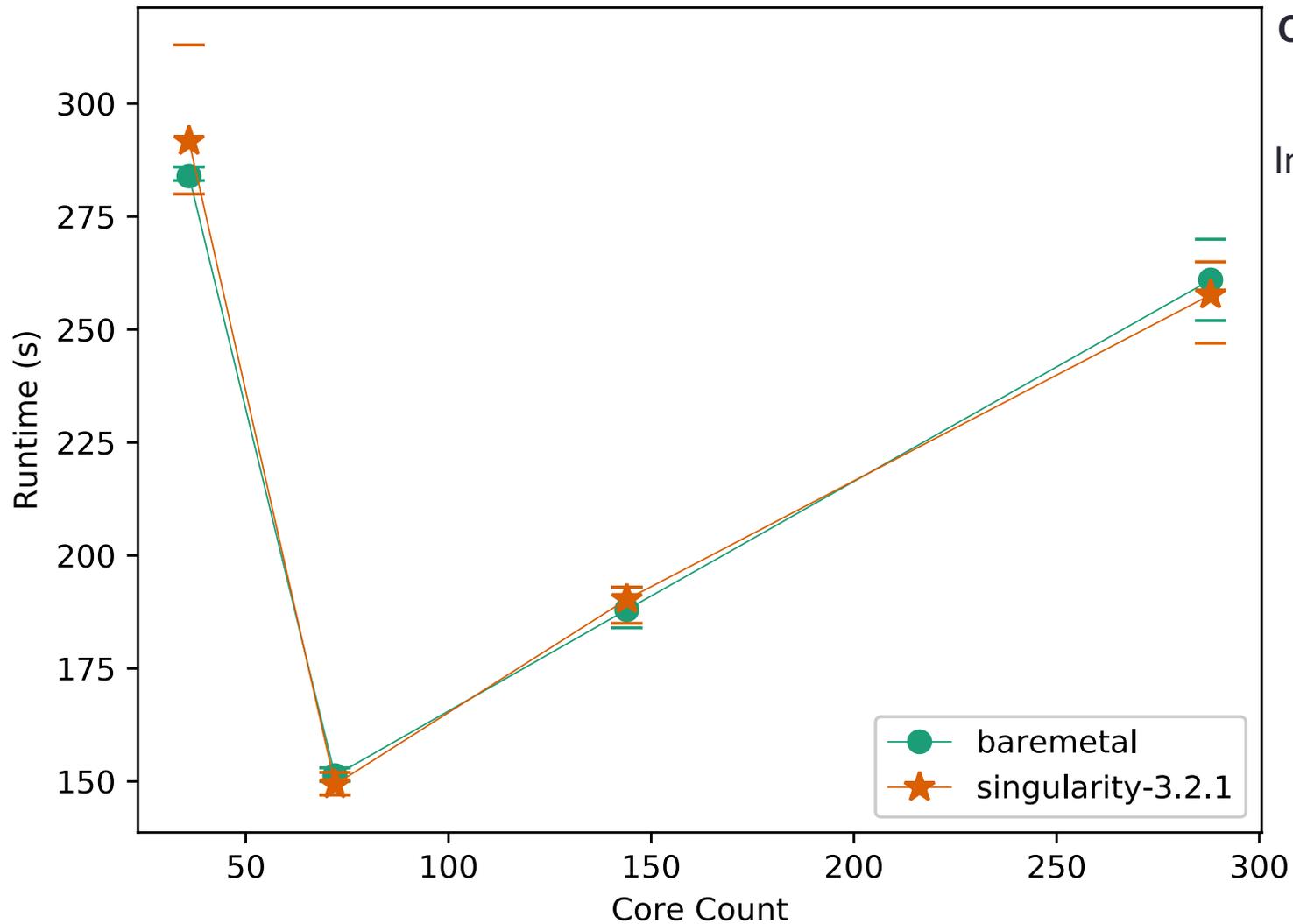
```
mpirun $MPIRUN_OPTS singularity exec -B /etc/libibverbs.d \
```

```
    $HOME/containers/dirac/arc/ramses-ubuntu19-openmpi4.sif
```

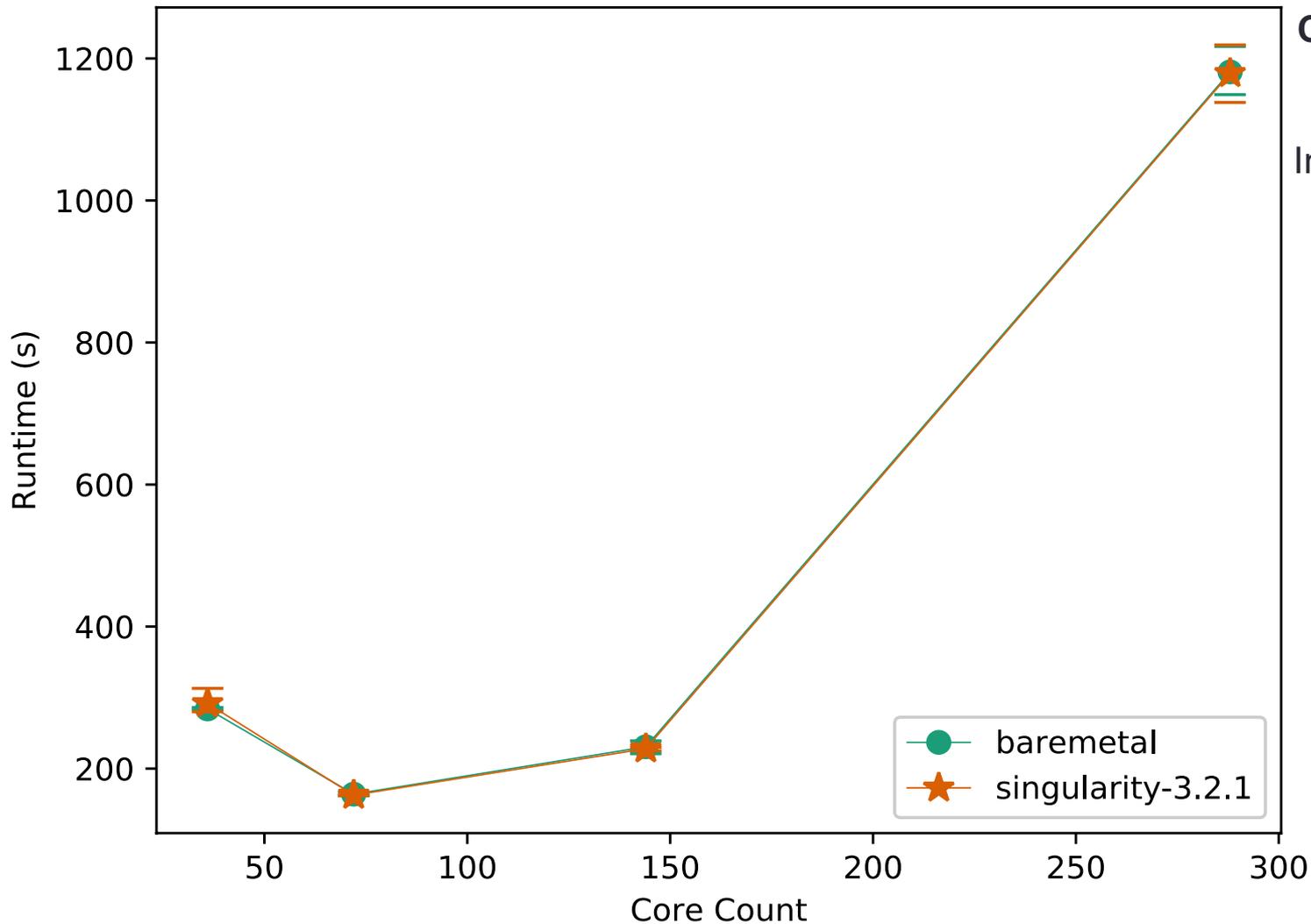
```
    /opt/apps/ramses/bin/ramses3d $APP_PARAMS &> $APP_OUTPUT
```

```
...
```

RAMSES Strong Scaling Performance on Cirrus

**Cirrus (pre-upgrade)**SGI ICE XA
Intel Xeon (Broadwell)
36 cores per node
256 GB memInfiniband (FDR)
54.5 Gb s⁻¹...but using
verbs interface

RAMSES Weak Scaling Performance on Cirrus

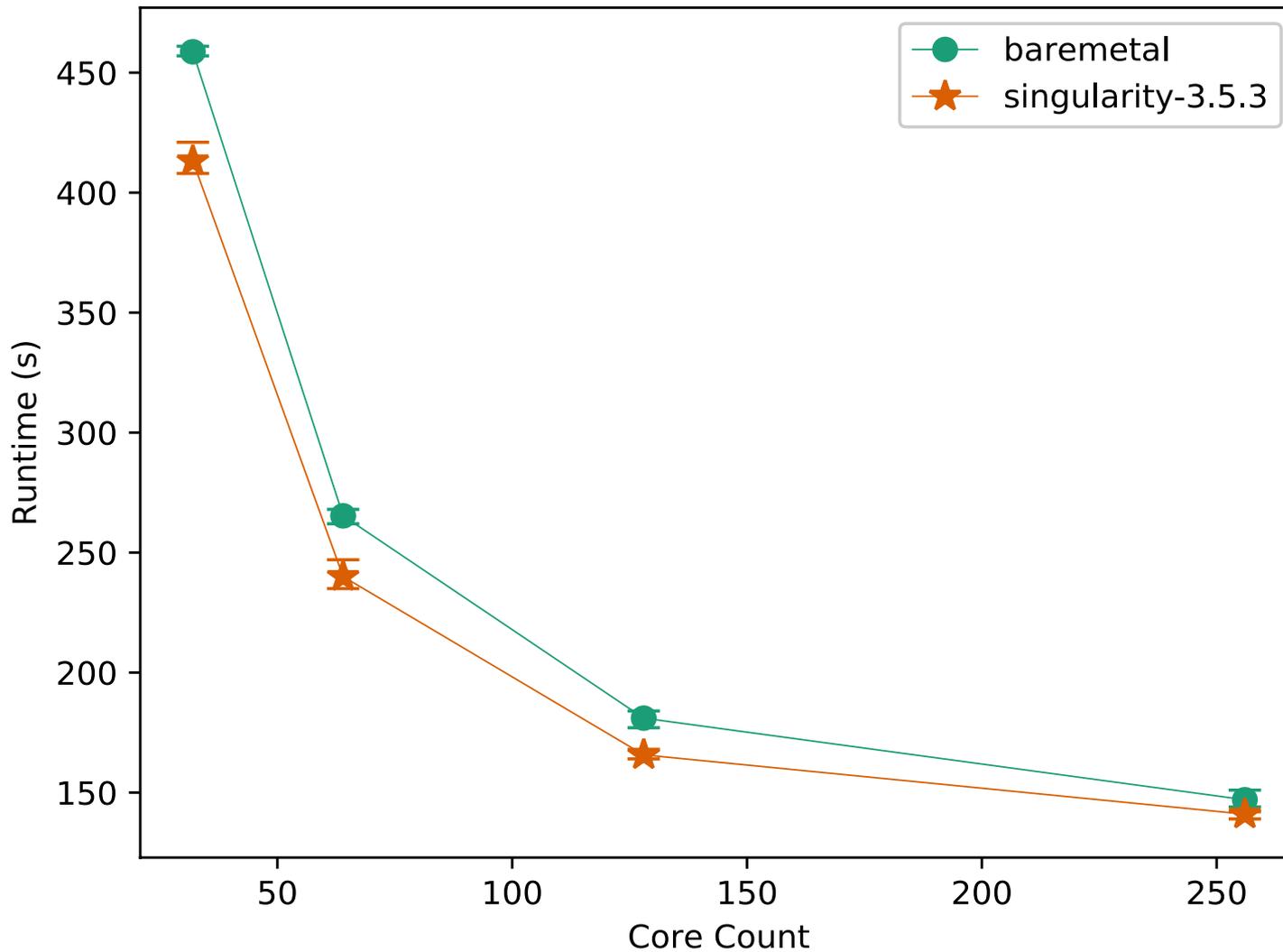
**Cirrus (pre-upgrade)**

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

Infiniband (FDR)
54.5 Gb s⁻¹

...but using
verbs interface

RAMSES Strong Scaling Performance on CSD3

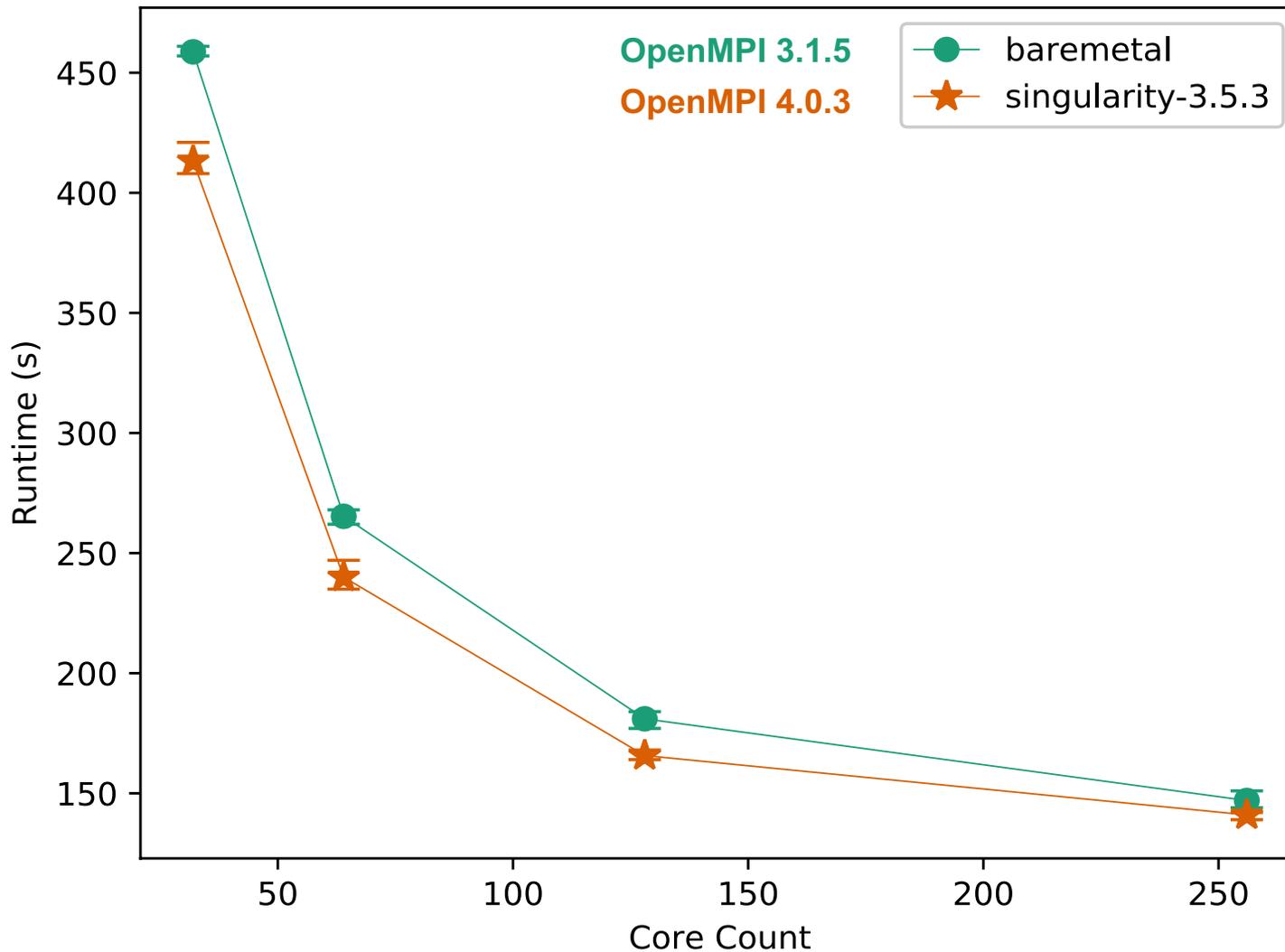
**CSD3 (Peta4)**

Intel Xeon Skylake
32 cores per node
192 GB mem

Intel OPA (PSM2)
100 Gb s⁻¹

...but again using
verbs interface

RAMSES Strong Scaling Performance on CSD3



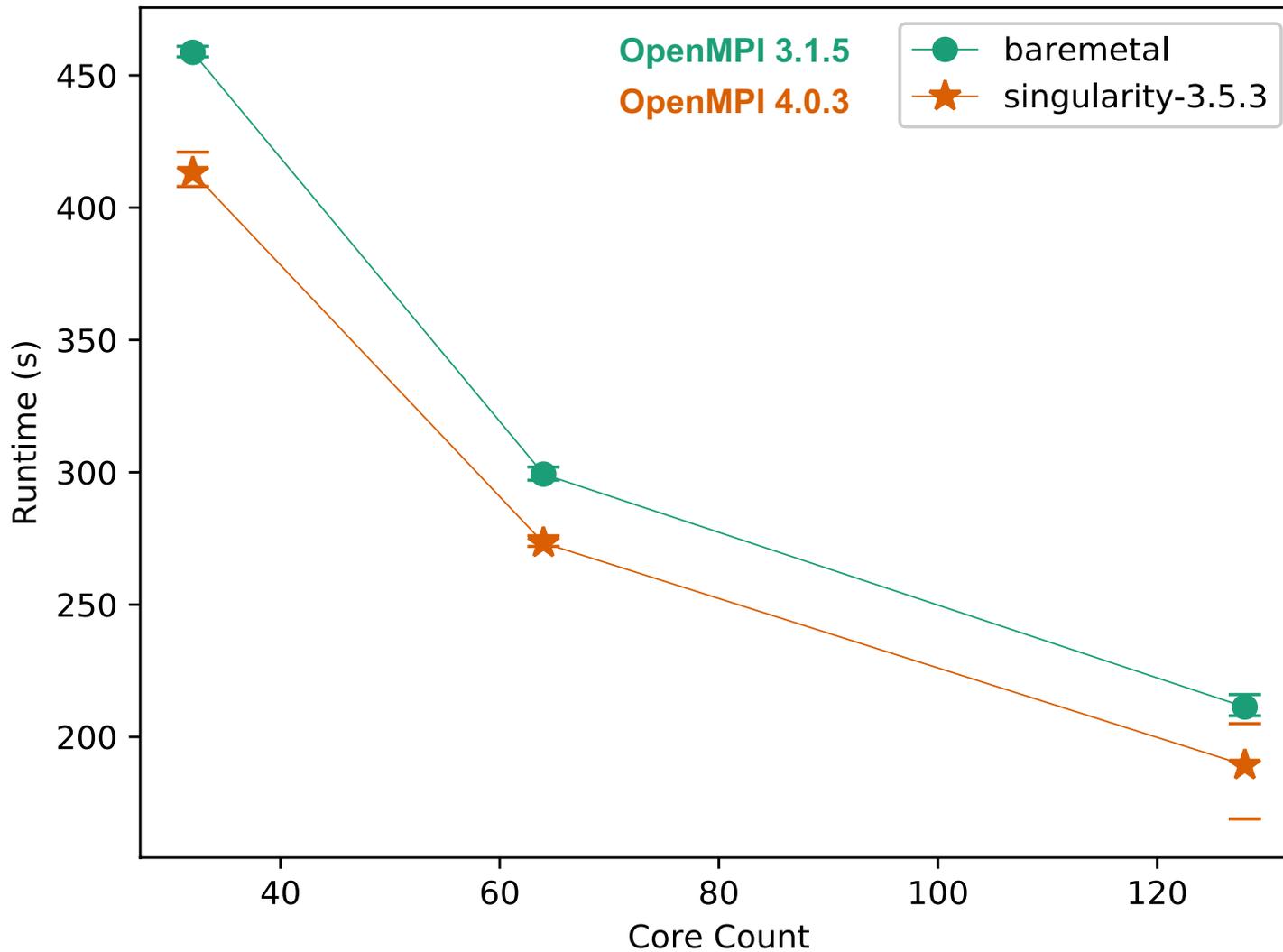
CSD3 (Peta4)

Intel Xeon Skylake
32 cores per node
192 GB mem

Intel OPA (PSM2)
100 Gb s⁻¹

...but again using
verbs interface

RAMSES Weak Scaling Performance on CSD3

**CSD3 (Peta4)**

Intel Xeon Skylake
32 cores per node
192 GB mem

Intel OPA (PSM2)
100 Gb s⁻¹

...but again using
verbs interface

Some Established ARCHER Codes

GROMACS 2016.1: 1400k-atom (pair of hEGFR Dimers of 1IVO and 1NQL)

CASTEP 18.1: 270-atom sapphire surface with a vacuum gap (al3x3)

CP2K 6.1: H2O-1024

Some Established ARCHER Codes

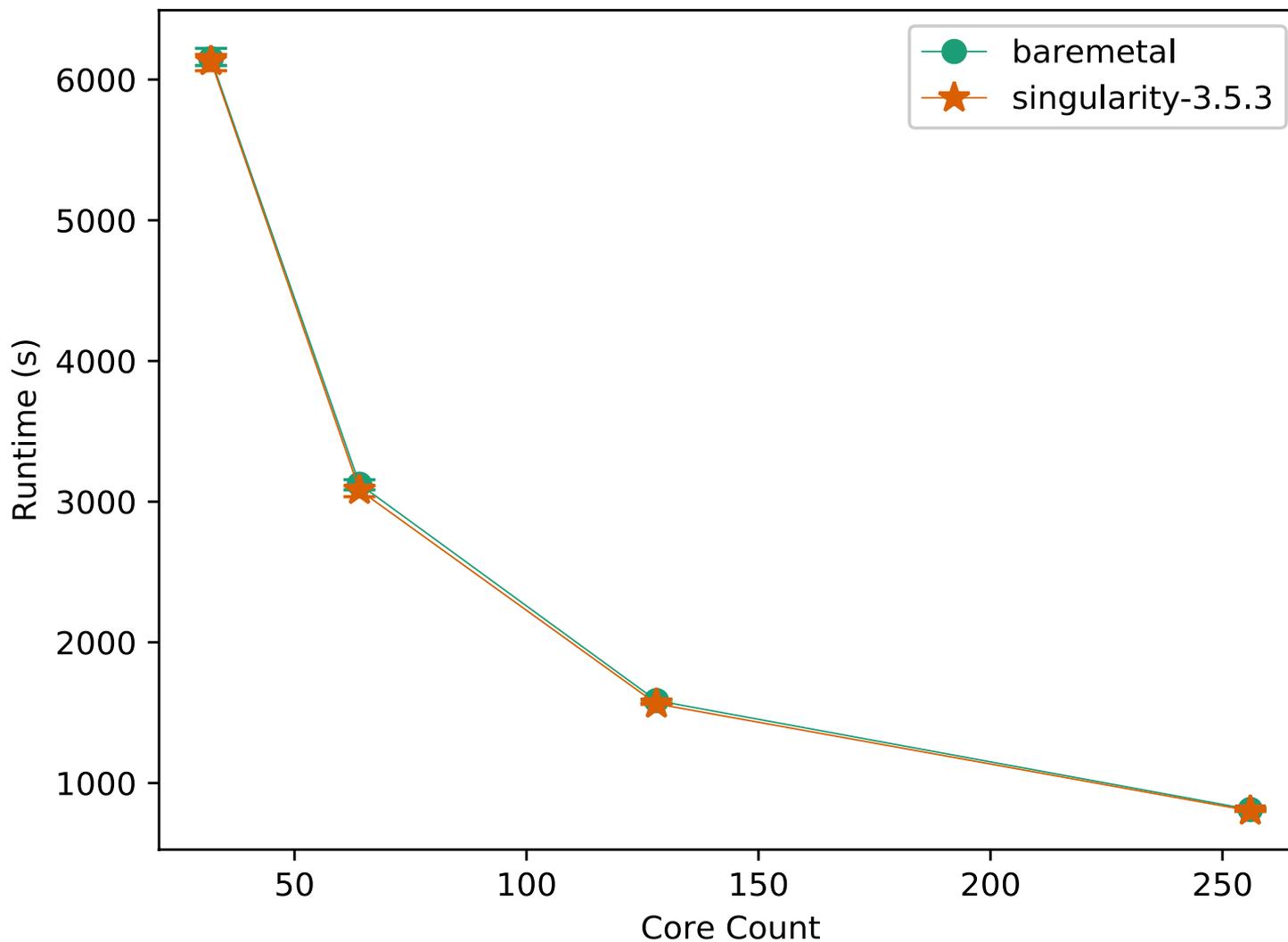
GROMACS 2016.1: 1400k-atom (pair of hEGFR Dimers of 1IVO and 1NQL)

CASTEP 18.1: 270-atom sapphire surface with a vacuum gap (al3x3)

CP2K 6.1: H2O-1024

It was straightforward to build GROMACS container at the factory (GROMACS itself was built using [cmake 3.14.1](#)).

GROMACS Strong Scaling Performance on CSD3

**CSD3 (Peta4)**

Intel Xeon Skylake
32 cores per node
192 GB mem

Intel OPA (PSM2)
100 Gb s⁻¹

...but using
verbs interface

Some Established ARCHER Codes

GROMACS 2016.1: 1400k-atom (pair of hEGFR Dimers of 1IVO and 1NQL)

CASTEP 18.1: 270-atom sapphire surface with a vacuum gap (al3x3)

CP2K 6.1: H2O-1024

It was straightforward to build GROMACS container at the factory (GROMACS itself was built using cmake 3.14.1).

CASTEP could not be compiled at the factory due to the Eleanor instance having insufficient memory (free tier provides just 1 GB).

- [mpif90](#) compiler unable to allocate memory for compilation of "ion.f90"

Post Factory Customization (CASTEP)

Container `/opt/scripts`

```
aux
  install_cmp.sh
  update_env
  ...

os
  ubuntu-19.10

cmp
  openmpi.sh
  mkl.sh
  fftw.sh
  ...

app
  castep
    csd3
      castep.sh
    cirrus
      castep.sh
    ...
  ...
```

Post Factory Customization (CASTEP)

Container /opt/scripts

```
aux
  install_cmp.sh
  update_env
  ...

os
  ubuntu-19.10

cmp
  openmpi.sh
  mkl.sh
  fftw.sh
  ...

app
  castep
    csd3
      castep.sh
    cirrus
      castep.sh
    ...
  ...
```

Host Customization Workflow

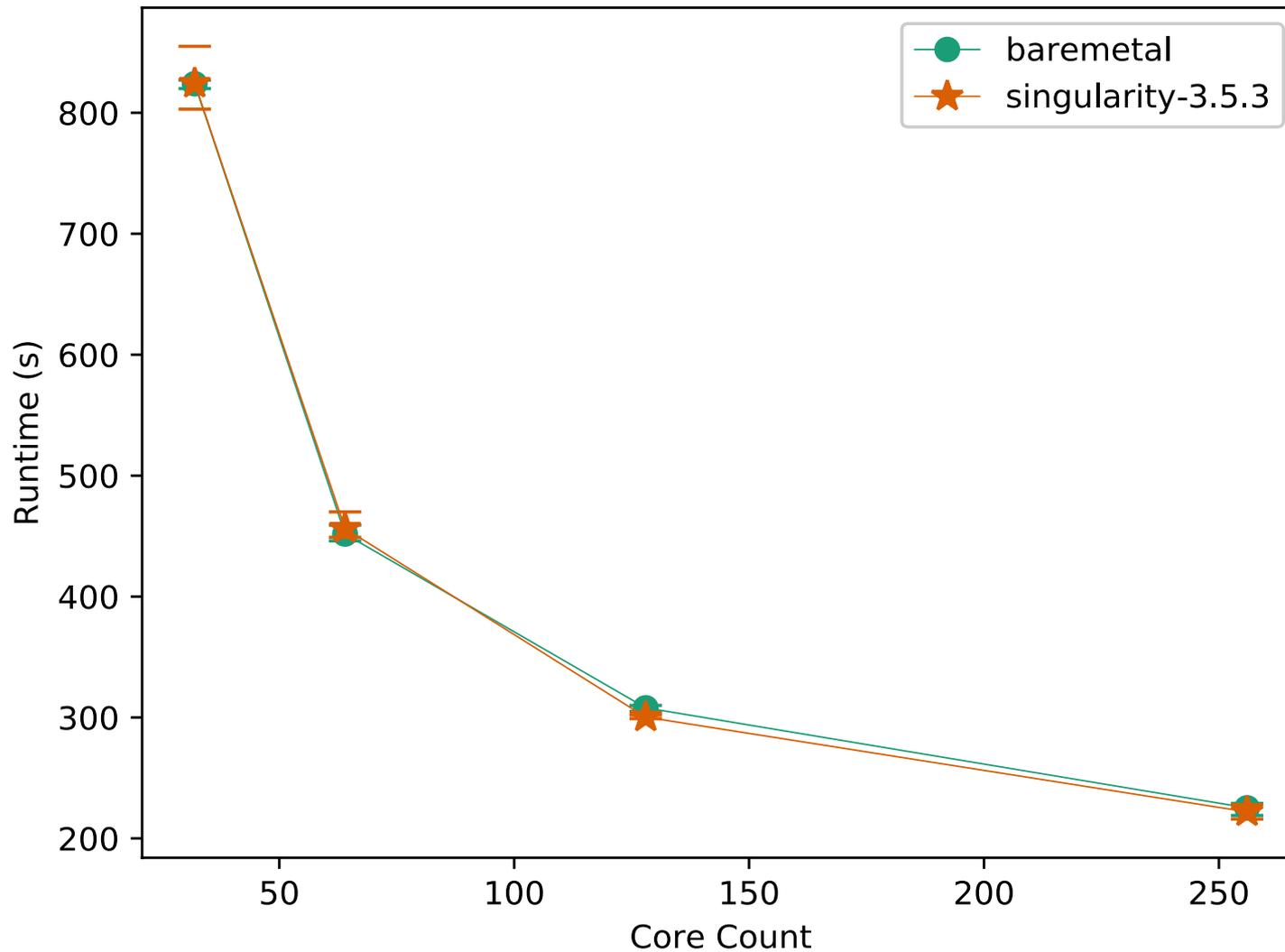
```
ssh <hostname>

...

tar -xvf castep-18.1.tar.gz
rm castep-18.1.tar.gz
cd castep-18.1

singularity shell /path/to/sif
  . /opt/scripts/app/castep/<hostname>/castep.sh
  exit
```

CASTEP Strong Scaling Performance on CSD3

**CSD3 (Peta4)**

Intel Xeon Skylake
32 cores per node
192 GB mem

Intel OPA (PSM2)
100 Gb s⁻¹

...but using
verbs interface

Some Established ARCHER Codes

GROMACS 2016.1: 1400k-atom (pair of hEGFR Dimers of 1IVO and 1NQL)

CASTEP 18.1: 270-atom sapphire surface with a vacuum gap (al3x3)

CP2K 6.1: H2O-1024

It was straightforward to build GROMACS container at the factory (GROMACS itself was built using cmake 3.14.1).

CASTEP could not be compiled at the factory due to the Eleanor instance having insufficient memory (free tier provides just 1 GB).

- mpif90 compiler unable to allocate memory for compilation of "ion.f90"

CP2K is easily the hardest of the three codes to compile.

- also hits factory memory limit during compilation of **libxc**
- some components such as **libgrid** have to be built on host

Post Factory Customization (CP2K)

Building CP2K first requires building many third-party libraries and some of these are host specific.

- **libxsmm**: targets Intel platforms for specialized matrix operations
- **libgrid**: automatically tunes CP2K's kernel routines for particular hardware

/opt/scripts/app/cp2k/

```
csd3
  libxsmm.slurm
  libxsmm.sh
  libgrid.slurm
  libgrid.sh
  cp2kexe.slurm
  cp2kexe.sh
```

Post Factory Customization (CP2K)

Building CP2K first requires building many third-party libraries and some of these are host specific.

- **libxsmm**: targets Intel platforms for specialized matrix operations
- **libgrid**: automatically tunes CP2K's kernel routines for particular hardware

/opt/scripts/app/cp2k/

```
csd3
  libxsmm.slurm
  libxsmm.sh
  libgrid.slurm
  libgrid.sh
  cp2kexe.slurm
  cp2kexe.sh
```

Host Customization Workflow

```
ssh <hostname>

...
cd cp2k-6.1

SIFPATH=/path/to/sif
singularity shell $SIFPATH
  HOSTPATH=/opt/scripts/app/cp2k/<hostname>
  cp $HOSTPATH/libxsmm.slurm ./
  cp $HOSTPATH/libgrid.slurm ./
  cp $HOSTPATH/cp2kexe.slurm ./
  exit

sbatch -export=ALL,SIF=$SIFPATH libxsmm.slurm
sbatch -export=ALL,SIF=$SIFPATH libgrid.slurm
sbatch -export=ALL,SIF=$SIFPATH cp2kexe.slurm
```

Post Factory Customization (CP2K – libgrid on CSD3)

libgrid.slurm

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

#SBATCH -J libgrid
#SBATCH -o libgrid.o%j
#SBATCH -e libgrid.o%j
#SBATCH -p skylake
#SBATCH -A dirac-ds007-cpu
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=12:00:00

module purge
module load singularity/current

singularity exec $SIF /opt/scripts/app/cp2k/csd3/libgrid.sh
```

Many variants ($8 \times 64 \times 4 = 2048$) of the *integrate* and *collocate* routines are built, run and timed; the variant that produces the correct answer in the shortest time is retained to generate the optimal binaries.

Running CP2K container on CSD3

The entire containerised CP2K build was done on CSD3.

- **login:** **ELPA**, **libint**, **libxc** and Plumed libraries
- **compute:** **libxsmm**, **libgrid** and **CP2K executables** as per previous slide

Running CP2K container on CSD3

The entire containerised CP2K build was done on CSD3.

- **login:** **ELPA**, **libint**, **libxc** and Plumed libraries
- **compute:** **libxsmm**, **libgrid** and **CP2K executables** as per previous slide

Unfortunately, attempting to run H2O-32 with containerised CP2K resulted in a segmentation fault (`cp_fm_diag.F`).

```

SUBROUTINE cp_fm_syevd_base(...)
  ...
  CALL pdsyevd(...)
  ...
  lwork = NINT(work(1)+100000)
  DEALLOCATE (work)
  ALLOCATE (work(lwork))
  ...
  CALL pdsyevd(...)
  ...
END SUBROUTINE cp_fm_syevd_base

```

2nd call to ScaLAPACK
routine, `pdsyevd()`



The “baremetal” CP2K, also built from source, ran H2O-32 and H2O-1024 successfully.

Summary and Further Work

No significant difference between baremetal and containerised performance.

See also <https://ieeexplore.ieee.org/document/8950978>.

Get containerised CP2K working on CSD3.

- problem could be due to `/proc/meminfo` differences
- compare performance not just with baremetal CP2K but also with containerised CP2K built entirely at factory

Repeat strong scaling runs on Cirrus and ARCHER2.

- extend runs over higher node counts
- do weak scaling runs

Facilitate access to Eleanor Container Factory within EPCC (and perhaps beyond).

Factory Costs

Factory spec (`t1.small`) shown in this presentation was insufficient to build CASTEP and CP2K (`libxc`).

A “production” factory instance will have running costs.

For example, assuming a spec of 8 vCPUs, 16 GB memory and 160 GB disk (`m1.xlarge` flavor), £100 would buy ~1176 hours per user.

<https://www.wiki.ed.ac.uk/display/ResearchServices/Cloud+Costs>

(link not accessible to non-UoE personnel)

(8 hours x 3 days x 45 weeks = 1080 hours)

Eleanor cloud instances can be “shelved” when not in use to avoid charging.

Conclusions

Those codes that are trivial to containerise (e.g., Ramses and GROMACS) are perhaps not worth containerising since reproducibility overhead is low.

Codes that are difficult to containerise are worth running in containers as these hide many compilation details that could potentially delay scientific output.

Conclusions

Those codes that are trivial to containerise (e.g., Ramses and GROMACS) are perhaps not worth containerising since reproducibility overhead is low.

Codes that are difficult to containerise are worth running in containers as these hide many compilation details that could potentially delay scientific output.

We (EPCC) should identify those codes that are particularly time-consuming to build and therefore would most benefit HPC users if containerised.

- CP2K, NEMO and FEniCS are perhaps strong candidates