ARCHER2 hardware environment

Adrian Jackson, EPCC, The University of Edinburgh
a.jackson@epcc.ed.ac.uk

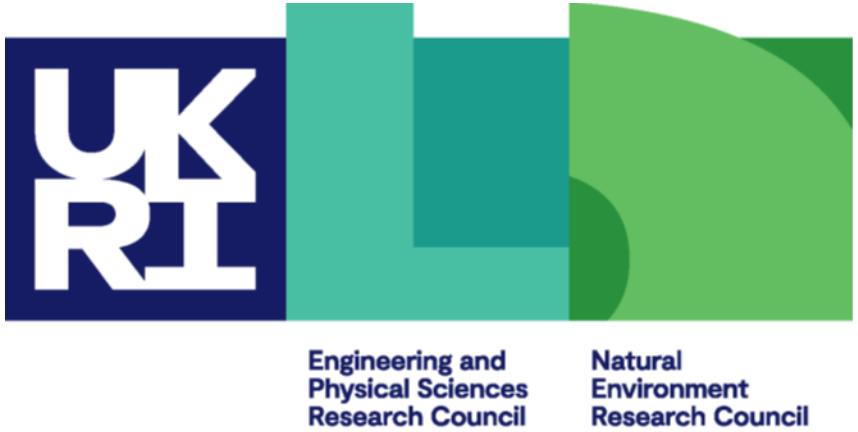
archer2

epcc

Reusing this material

Partners

epcc



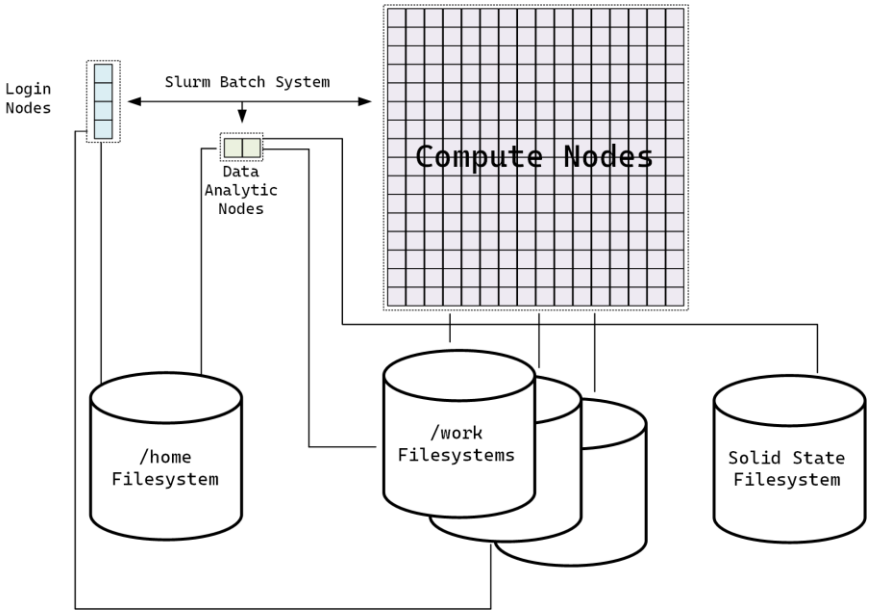
epcc

THE UNIVERSITY of EDINBURGH

Hewlett Packard Enterprise

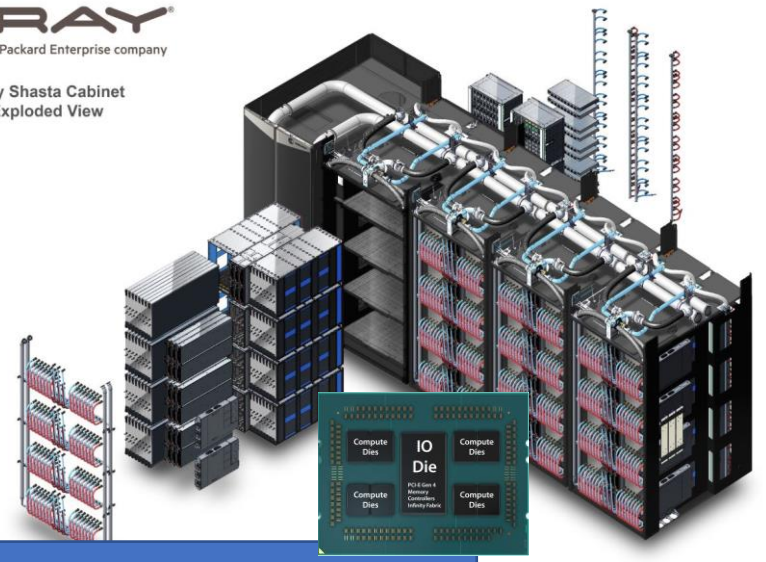
Hardware

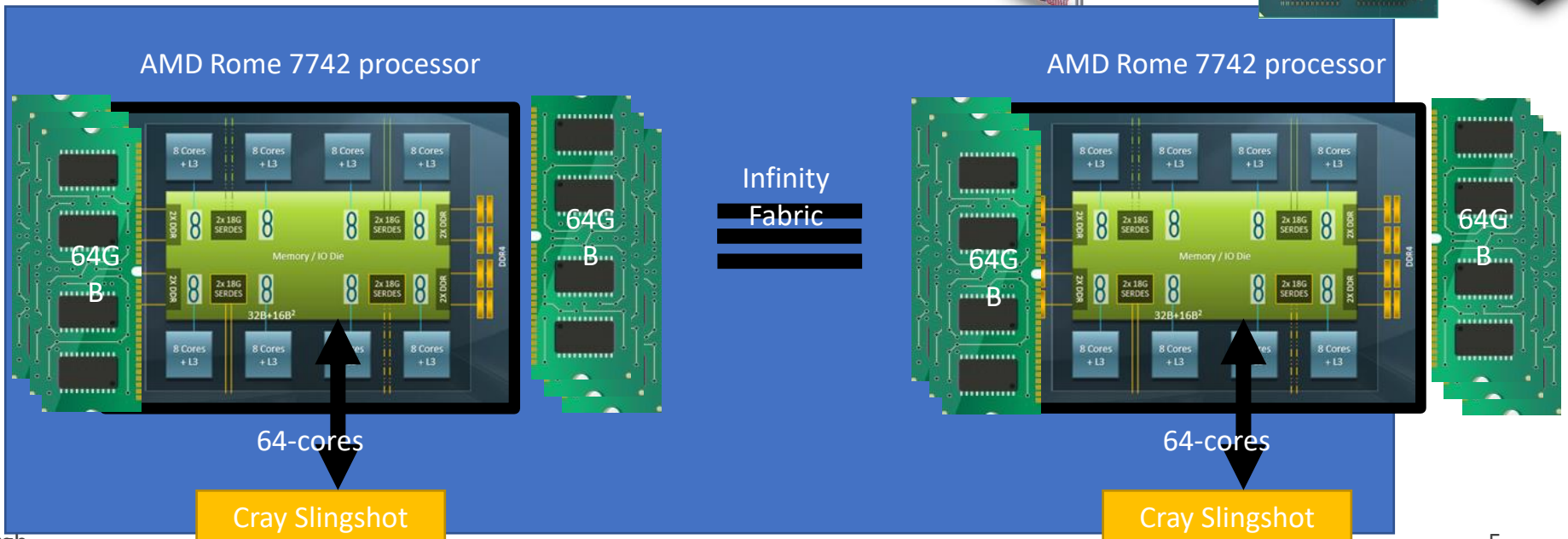
- ARCHER2 is composed of:
  - 5,860 compute nodes
  - High performance network
  - home filesystem
  - work filesystem
  - 4 login nodes
  - 2 data analytics nodes
- Filesystems
  - /home: 1 PB network filesystem
    - Available on login and data analysis nodes
  - /work: 14.5 PB lustre filesystem
    - Available on login, data analysis and compute nodes
  - Solid state storage: 1.1 PB NVMe storage



Hardware

- Nodes comprise of:
  - 128 processor cores:
    - 2x AMD EPYC Zen2 (Rome) 7742, 2.25 GHz, 64-core
  - 256/512 GB DDR 3200, 8 memory channels
    - 300 high memory nodes (512 GB)





Software

- Linux operating system
  - HPE Cray Linux Environment (based on SLES 15)
- Slurm scheduler
  - Access compute and data nodes
- Three compilers
  - Cray: crayftn, craycc, crayCC
  - GNU: gfortran, gcc, g++
  - AMD: clang, clang++, flang
- HPE/Cray MPI and communication libraries
- HPE Cray scientific and numerical libraries:
  - HPE Cray LibSci: BLAS, LAPACK, ScaLAPACK
  - FFTW 3
  - NetCDF
  - HDF5
  - etc…
  - cray-python
- Programming tools:
  - gdb4hpc parallel debugger
  - valgrind4hpc parallel memory debugging
  - STAT stack trace analysis tool
  - ATP abnormal program termination analysis tool
  - HPE Cray Performance Analysis Toolkit (CrayPAT)
- https://docs.archer2.ac.uk/

Usage

- Login nodes for compilation and general development
- Data nodes for longer running data movement/processing tasks (limited amounts of resources)
- Compute nodes for everything else
  - Compute nodes require runtime budget
- Usage of cores is restricted
  - Login nodes have fair share
  - Compute nodes having binding/placement enforcement

Batch system

- Full configuration details
  - https://docs.archer2.ac.uk/user-guide/scheduler/

```
#!/bin/bash
#SBATCH --job-name=xthi
#SBATCH --nodes=2
#SBATCH --tasks-per-node=128
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=ta144
#SBATCH --time=0:10:0

srun -n 256 ./xthi
```

Multiple tasks on the same node

```
#!/bin/bash
#SBATCH --job-name=xthi
#SBATCH --nodes=2
#SBATCH --tasks-per-node=128
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=ta144
#SBATCH --time=0:10:0

srun -n 64 ./xthi &
srun -n 64 ./xthi &
srun -n 64 ./xthi &
srun -n 64 ./xthi &
wait
```

Multiple tasks on the same node

```
#!/bin/bash
#SBATCH --job-name=xthi
#SBATCH --nodes=2
#SBATCH --tasks-per-node=128
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=ta144
#SBATCH --time=0:10:0

for i in {1..256}
do
   srun -n 1 ./xthi &
done
wait
```

Running multiple jobs in a single script

- Slurm allows job arrays
  - Same script run on differing indexes for a range of indexes

```
#!/bin/bash
# Slurm job options (job-name, compute nodes, job time)
#SBATCH --job-name=Example_Array_Job
#SBATCH --time=04:00:00
#SBATCH --nodes=1
#SBATCH --tasks-per-node=128
#SBATCH --cpus-per-task=1
#SBATCH --array=0-55
# Replace [budget code] below with your budget code (e.g. ta144)
#SBATCH --account=[budget code]
#SBATCH --partition=standard
#SBATCH --qos=standard

srun --distribution=block:block --hint=nomultithread /path/to/exe $SLURM_ARRAY_TASK_ID
```

Interactive access to compute nodes

- Run a program on the compute nodes from the login nodes

```
srun --nodes=1 --tasks-per-node=2 --cpus-per-task=1 \
 --exclusive --partition=standard --qos=short \
 --reservation=shortqos --account=ta144 \
 --time=0:20:0 python ./example4.py
```

- Get a shell on the compute nodes:

```
srun --nodes=1 --tasks-per-node=2 --cpus-per-task=1 \
 --exclusive --partition=standard --qos=short \
 --reservation=shortqos --account=ta144 --time=0:20:0 \
 --pty /bin/bash
```

  - Requires the addition of `--oversubscribe` if using `srun` within that shell

- Can allocate resources then launch a run from the compute nodes as well

```
salloc --nodes=8 --tasks-per-node=128 --cpus-per-task=1 \
       --time=00:20:00 --partition=standard --qos=short \
       --account=ta144
```

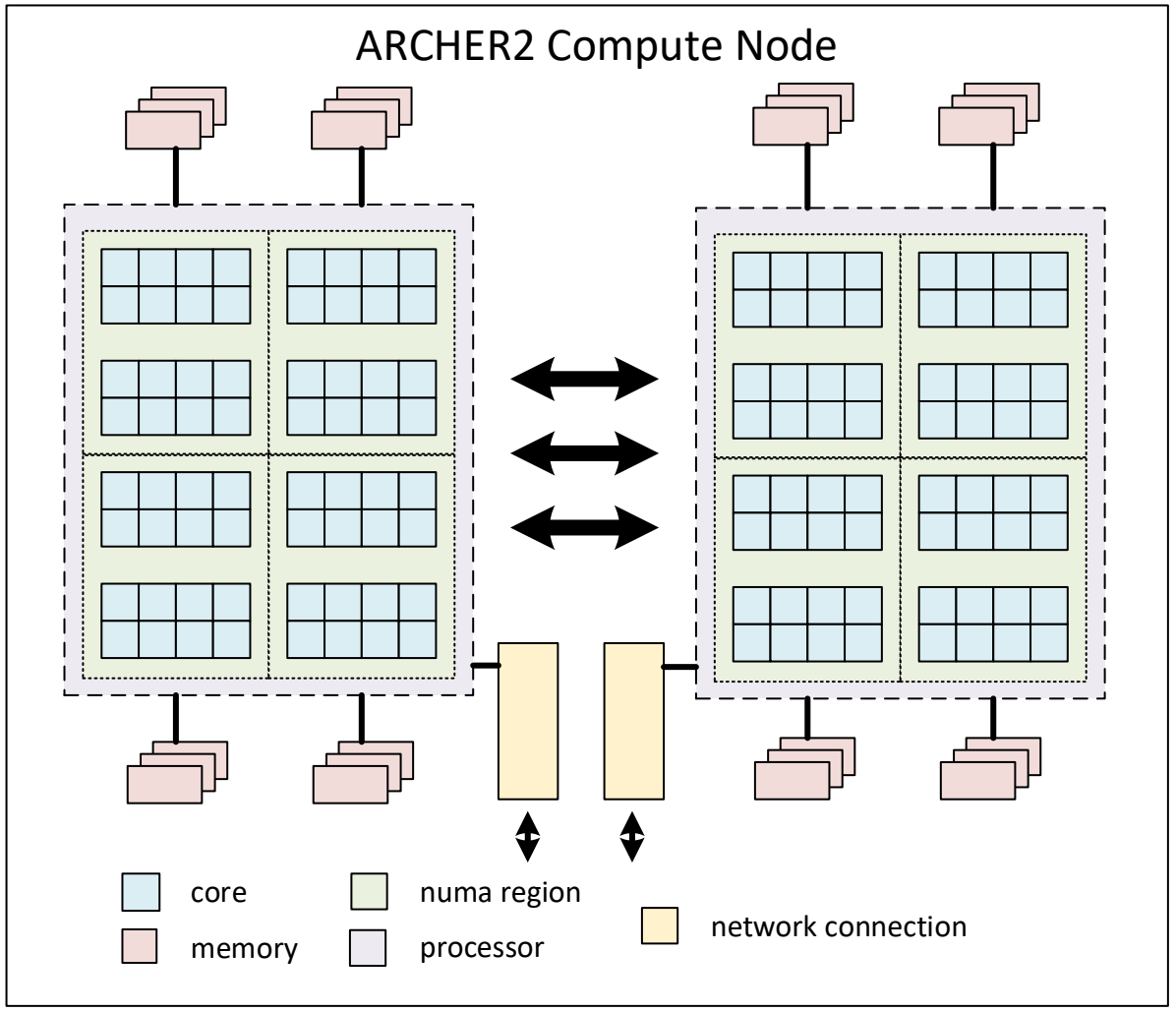
  - Wait for the allocation

```
srun --distribution=block:block --hint=nomultithread python ./my_code.py
exit
```

Batch system configuration

QoS	Partition(s)	Max Nodes Per Job	Max Walltime	Jobs Queued	Jobs Running	Notes
standard	standard	1024	24 hrs	64	16	Maximum of 1024 nodes in use by any one user at any time
highmem	highmem	256	24 hrs	16	16	Maximum of 512 nodes in use by any one user at any time
taskfarm	standard	16	24 hrs	128	32	Maximum of 256 nodes in use by any one user at any time
short	standard	32	20 mins	16	4	
long	standard	64	96 hrs	16	16	Minimum walltime of 24 hrs, maximum 512 nodes in use by any one user at any time, maximum of 2048 nodes in use by QoS
largescale	standard	5860	12 hrs	8	1	Minimum job size of 1025 nodes
lowpriority	standard	2048	24 hrs	16	16	Jobs not charged but requires at least 1 CU in budget to use.
serial	serial	32 cores and/or 128 GB memory	24 hrs	32	4	Jobs not charged but requires at least 1 CU in budget to use. Maximum of 32 cores and/or 128 GB in use by any one user at any time.

Process and thread placement



- https://docs.archer2.ac.uk/user-guide/hardware/

Batch system

```
#!/bin/bash
#SBATCH --job-name=xthi
#SBATCH --nodes=2
#SBATCH --tasks-per-node=128
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=ta144
#SBATCH --time=0:10:0

srun -n 256 ./xthi
```

Batch system

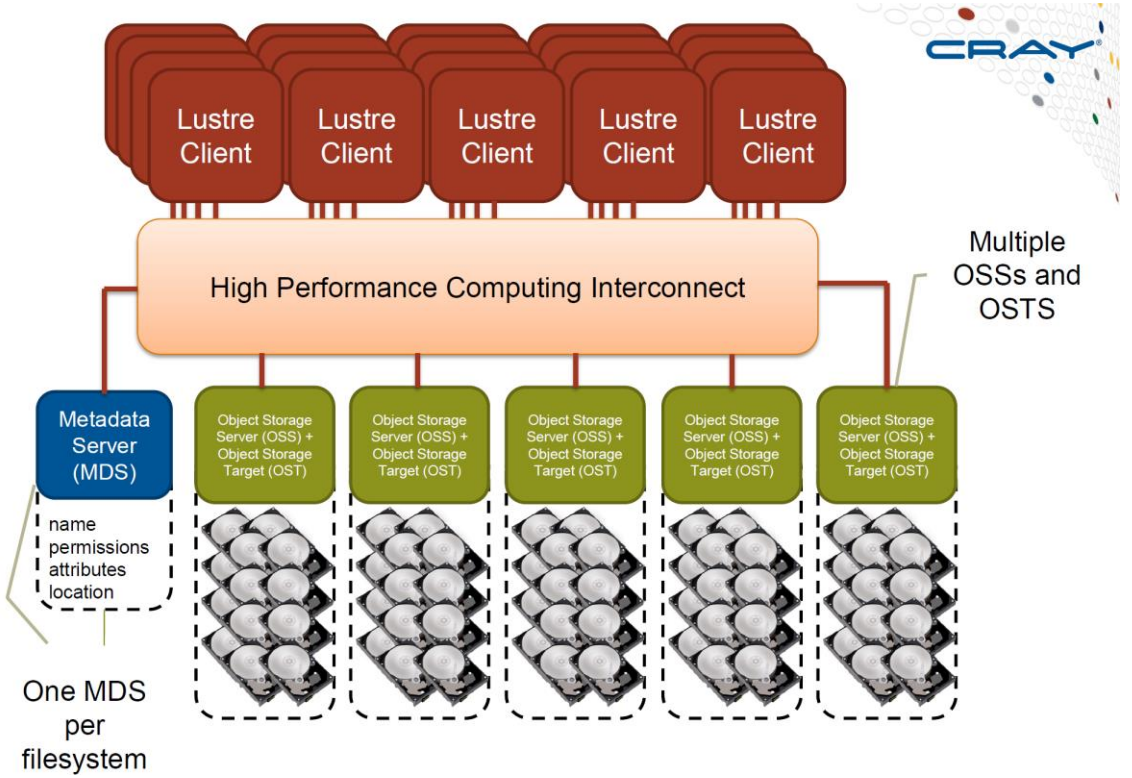
```
#!/bin/bash
#SBATCH --job-name=xthi
#SBATCH --nodes=2
#SBATCH --tasks-per-node=64
#SBATCH --cpus-per-task=2
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=ta144
#SBATCH --time=0:10:0
export OMP_NUM_THREADS=2
srun -n 128 ./xthi
```

Process and thread placement

```
#!/bin/bash
#SBATCH --job-name=MultiParallelOnCompute
#SBATCH --time=0:10:0
#SBATCH --nodes=1
#SBATCH --tasks-per-node=64
#SBATCH --cpus-per-task=2
#SBATCH --partition=standard
#SBATCH --qos=standard
module load cray-python
module load genmaskcpu
# Set the number of threads to 2 as required by all subjobs
export OMP_NUM_THREADS=2
# Loop over 8 subjobs
for i in $(seq 1 8)
do
    echo $j $i
    # Generate mask: 8 subjobs per node, subjob number in sequence given by i,
    # 8 MPI processes per subjob, 2 OpenMP threads per process
    maskcpu=$(genmaskcpu 8 ${i} 8 2)
    srun --cpu-bind=mask_cpu:${maskcpu} --nodes=1 --ntasks=8 --tasks-per-node=8 --
cpus-per-task=2 \
     --oversubscribe --mem=12500M xthi > placement${i}.txt &
done
# Wait for all subjobs to finish
wait
```

Filesystem usage

- /work is a Lustre filesystem
  - Parallel filesystem
  - External storage and metadata nodes
  - OSS containing OSTs
  - MDS containing MDT
- I/O bandwidth
  - Requires splitting files across OSS
  - Striping
- Striping increases metadata overheads
- Default configuration
  - A default stripe count of 1
  - A default stripe size of 1 MiB (2^{20} = 1048576 bytes)



Filesystem usage

- Striping cont.
  - Improves bandwidths, overall performance available, and maximum file size
  - Incurs communication overhead and contention potentials including serialisation if multiple processes accessing same units
- `lfs` command for more information and configuration

```
adrianj@nid16958:~>lfs df –h
```

(query number of OSTs)

```
adrianj@nid16958:~> lfs getstripe dirname
```

(query `stripe count, stripe size`)

```
adrianj@nid16958:~> lfs setstripe dirname 0 –1 –1
```

(set large file `stripe size, start index, stripe count`)

```
adrianj@nid16958:~> lfs setstripe dirname 0 –1 1
```

(set lots of files `stripe size, start index, stripe count`)

Summary

- Basic hardware usage is straightforward
  - Distributed memory jobs will map to the system as expected
- Running lots of small or serial jobs together is possible
  - Array jobs, combined `srun` loops etc…
- Placement/binding important to ensure workers get hardware available
  - Ensure `srun` and `sbatch` options match what you actually want
  - Login nodes don’t provide this functionality in a straightforward way
- Filesystem configuration may also be important for your use case
  - Large scale parallel jobs, single file access, striping very important to turn on
  - Lots of individual file access, default configuration is ok
- Understanding your specific use case and how to make it to the hardware important for efficient usage