ARCHER2 GPU Development Platform

ARCHER2 CSE Team, EPCC, The University of Edinburgh

support@epcc.ed.ac.uk

www.archer2.ac.uk

archer2

epcc

Outline

- GPU development platform overview
- GPU node hardware
- Scheduler configuration
  - Primary resource – requesting and using resources
  - Limits
- Example job submission scripts
- Example interactive jobs
- GPU eCSE Call
- Future work planned by ARCHER2 CSE team

ARCHER2 Partners

epcc

epcc

THE UNIVERSITY of EDINBURGH

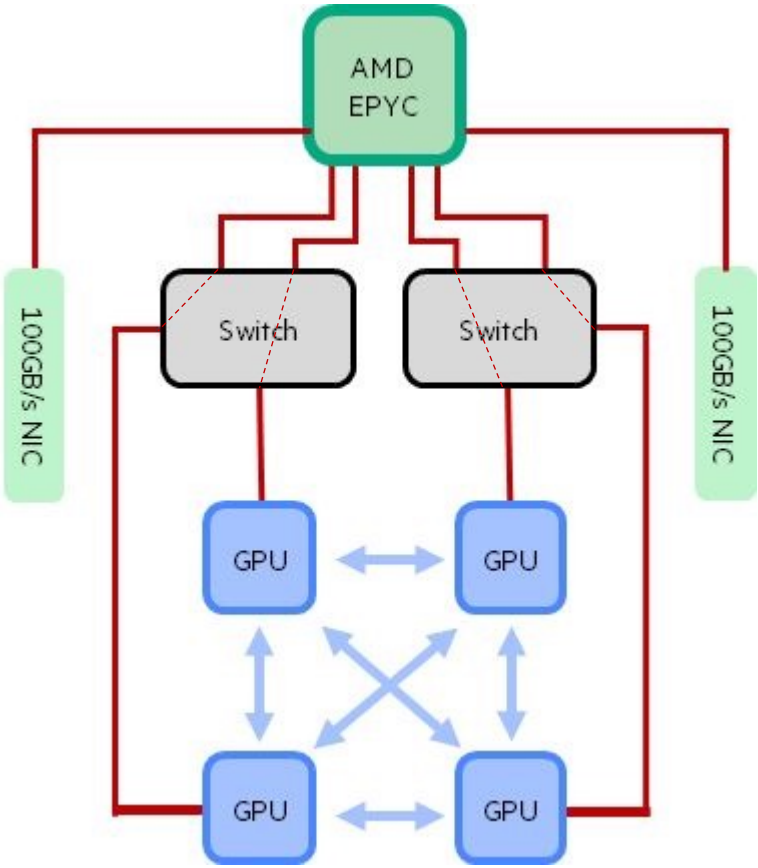
Hewlett Packard Enterprise

GPU Development Platform overview

- Very small resource
  - 4 nodes, 16 GPU in total
- For development and testing, not production use
- Integrated into ARCHER2 system
  - No separate login nodes, same scheduler
- Available to all ARCHER2 users with a positive CU budget
  - **No charge for GPU node use!**
- Documentation at: https://docs.archer2.ac.uk/user-guide/gpu/
  - Basic documentation at the moment
  - Will be added to and updated over the next few weeks

GPU node hardware

- Each node:
  - 1x AMD CPU
  - 4x AMD Instinct MI210 GPU
  - 2x single port 100 Gbps Slingshot interconnect
- /work and solid state (NVMe) file systems available



Compiling

- Programming environment with GPU support available on:
  - Login nodes
  - GPU compute nodes (terminal access via Slurm)
  - Serial (data analysis nodes)
- Full details of offload methods, instructions and examples in HPE CoE presentation and in documentation

Scheduler configuration

- Submit from ARCHER2 login nodes as for other job types
- Use your job script to select the number of GPU you require
  - `--gpus=N` option
  - Do not specify number of CPU cores or amount of memory
  - CPU cores and memory assigned pro rata based on number of GPU you request
  - 8 CPU cores and 128 GiB memory allocated per GPU requested
- Once you have your allocated resources, use `srun` to specify how many tasks (MPI processes) and threads you need and in what configuration

QoS and job limits

- Partition: gpu
- QoS:
  - gpu-shd: access GPU while sharing node with other users
    - Typically used for 1-2 GPU on a single node or 2 GPU spread across two nodes
    - Higher priority
  - gpu-exc: node-exclusive access
    - 4 GPU on a single node or 8 GPU across two nodes
    - Lower priority
- Limits (both QoS):
  - 2 jobs submitted (1 may be running) per user
  - 1 hour maximum walltime
  - Maximum resources:
    - gpu-shd: 2 GPU maximum
    - gpu-exc: 2 nodes maximum (8 GPU)

Example job submission script (gpu-shd)

- Request: 1 GPU, 20 mins
- Use case: 1 CPU process, 8 CPU threads, 1 GPU

```
#!/bin/bash
#SBATCH --job-name=single-GPU
#SBATCH --account=[budget]
#SBATCH --partition=gpu
#SBATCH --qos=gpu-shd
#SBATCH --gpus=1
#SBATCH --time=0:20:0

# Check assigned GPU
srun --ntasks=1 rocm-smi

export OMP_NUM_THREADS=8
srun --ntasks=1 –cpus-per-task=8 ./my_gpu_program.x
```

Remember, you only ask for the number of GPU you require in the `sbatch` options and CPU cores and host memory are assigned pro rata

`rocm-smi` is useful to check you have the number of GPU you expect

Once you have your assigned block of resource: GPUs, CPU cores and memory; you can use `srun` within the job script to divide them up however you want.

Example job submission script (gpu-exc)

- Request: 8 GPU, 20 mins
- Use case: 1 MPI process per GPU, 8 CPU threads per MPI process, 8 GPU

```
#!/bin/bash
#SBATCH --job-name=multi-GPU
#SBATCH --account=[budget]
#SBATCH --partition=gpu
#SBATCH --qos=gpu-exc
#SBATCH --gpus=8
#SBATCH --nodes=2
#SBATCH --exclusive
#SBATCH --time=0:20:0

# Check assigned GPU
nodelist=$(scontrol show hostname $SLURM_JOB_NODELIST)
for nodeid in $nodelist
do
   echo $nodeid
   srun --ntasks=1 --nodelist=$nodeid rocm-smi
done

module load xthi
srun --ntasks=8 --cpus-per-task=8 \
     --hint=nomultithread --distribution=block:block \
     xthi

srun --ntasks=8 --cpus-per-task=8 \
     --hint=nomultithread --distribution=block:block \
     ./my_gpu_program.x
```

When you use the exclusive QoS (gpu-exc), you also must specify the number of nodes you want and the `--exclusive` flag

xthi is useful to check you have the process and thread pinning you expect. Use exactly the same `srun` options you use for your executable

Interactive jobs

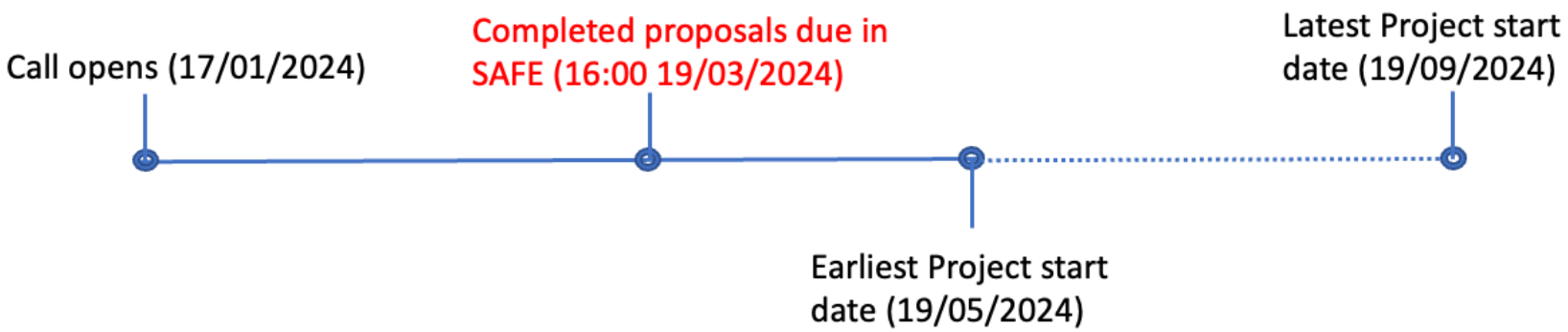
- Use `salloc` to reserve resources and `srun` to launch again to launch the executable
- Example, 1 GPU, 1 process
- Reserve resources:

```
salloc --gpus=1 --time=00:20:00 --partition=gpu --qos=gpu-shd --account=z19
```

- Launch interactive job:

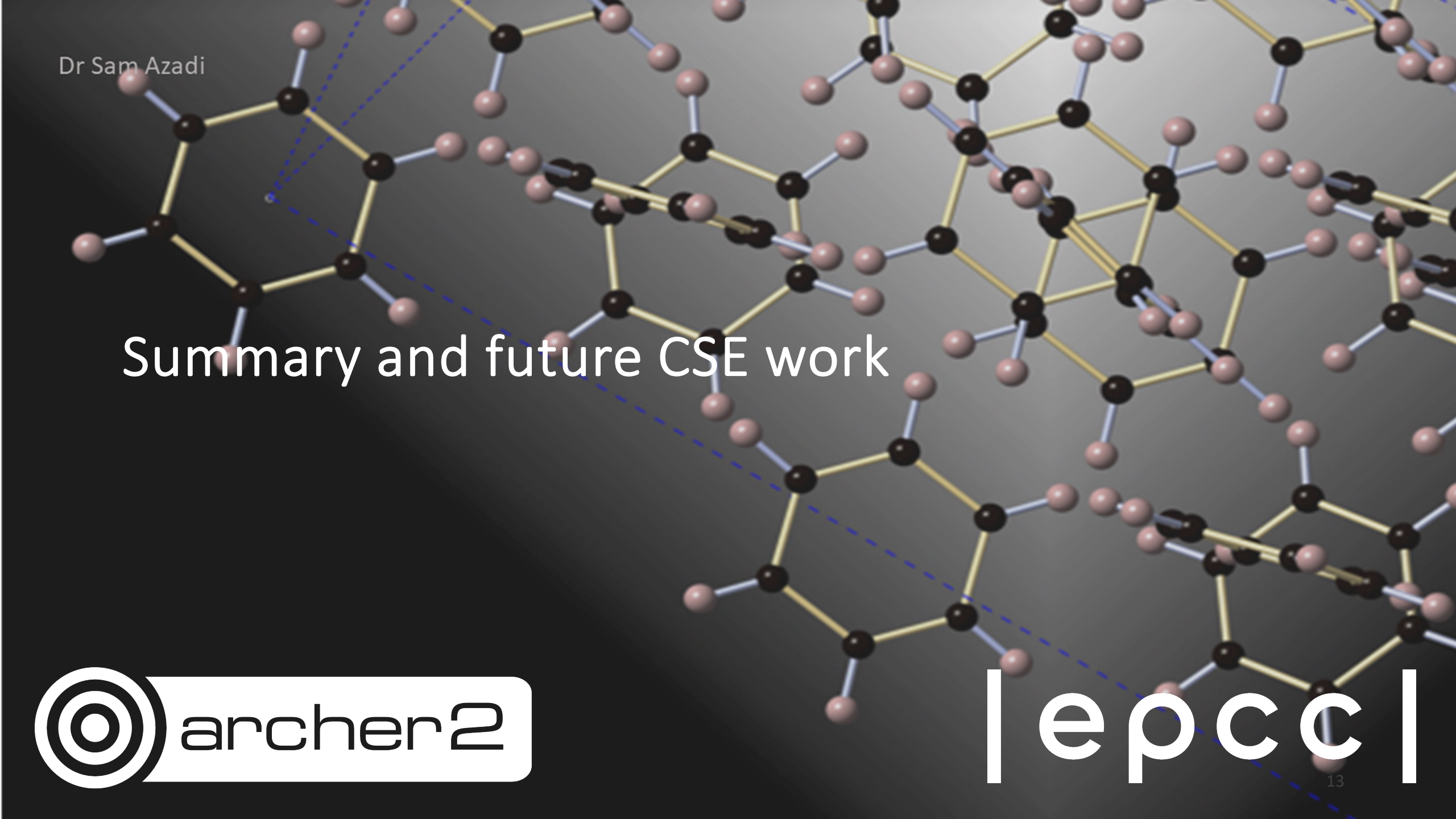
```
srun --ntasks=1 –cpus-per-task=8 ./my_gpu_program.x
```

GPU eCSE Programme



- Programme of GPU eCSE software development calls
  - Expect to run 3 calls this year
  - 1st call open now
- Open to proposals to support research across all of UKRI remit
- Up to 36 person months of effort available per call
  - Max duration 2 years
  - Flexibility of how effort spend (e.g. 1 person 50% for 2 years, 2 people 100% for 18 months, etc.)
  - Funding can be for RSE at PI's institution, RSE at third-party institution, member of ARCHER2 CSE team – or combination of the above

https://www.archer2.ac.uk/ecse/calls/



Dr Sam Azadi

Summary and future CSE work

archer2

epcc

CSE future work

- Porting software packages:
  - e.g. Gromacs, LAMMPS, NAMD, VASP
- Porting ML frameworks
  - PyTorch and Tensorflow
- Continue to investigate and document functionality
  - Performance libraries
  - Debugging and profiling
- Performance investigations

Summary

- GPU Development Platform available to all ARCHER2 users
  - Small resource – aimed at development and testing, not production research
- 4 nodes, each with 4 AMD Instinct MI210 GPU
- Shared node access, can request a single GPU
- Exclusive node access – maximum of 2 nodes per job
- Documentation and guidance are work in progress and will be expanded
- GPU-based eCSE call open – closing date 19 Mar 2024

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