THE UNIVERSITY OF QUEENSLAND AUSTRALIA | CREATE CHANGE

Level up your HPC skills

Dr Emily Kahl, Australian Institute for Bioengineering and Nanotechnology (AIBN), The University of Queensland

(e.kahl@uq.edu.au)

Acknowledgements

- Rika Kobayashi (NCI)
- Marco De La Pierre (Pawsey)
- Marlies Hankel (RCC/QCIF)
- Shern Tee (UQ)
- Stephen Sanderson (UQ)

NCI AUSTRALIA

RCC RESEARCH COMPUTING CENTRE

pawsey

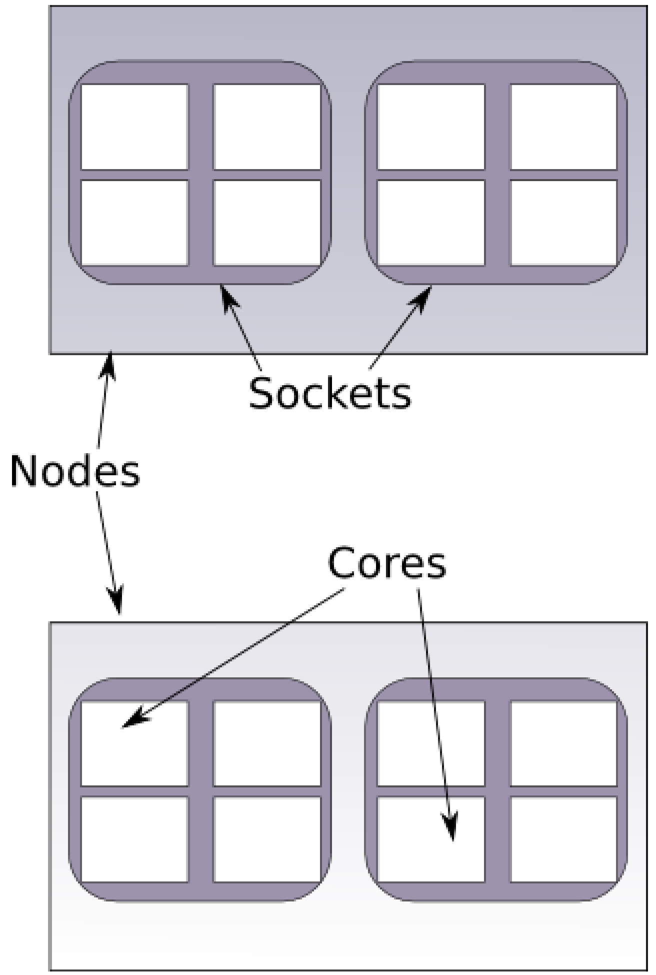
THE UNIVERSITY OF QUEENSLAND AUSTRALIA | CREATE CHANGE

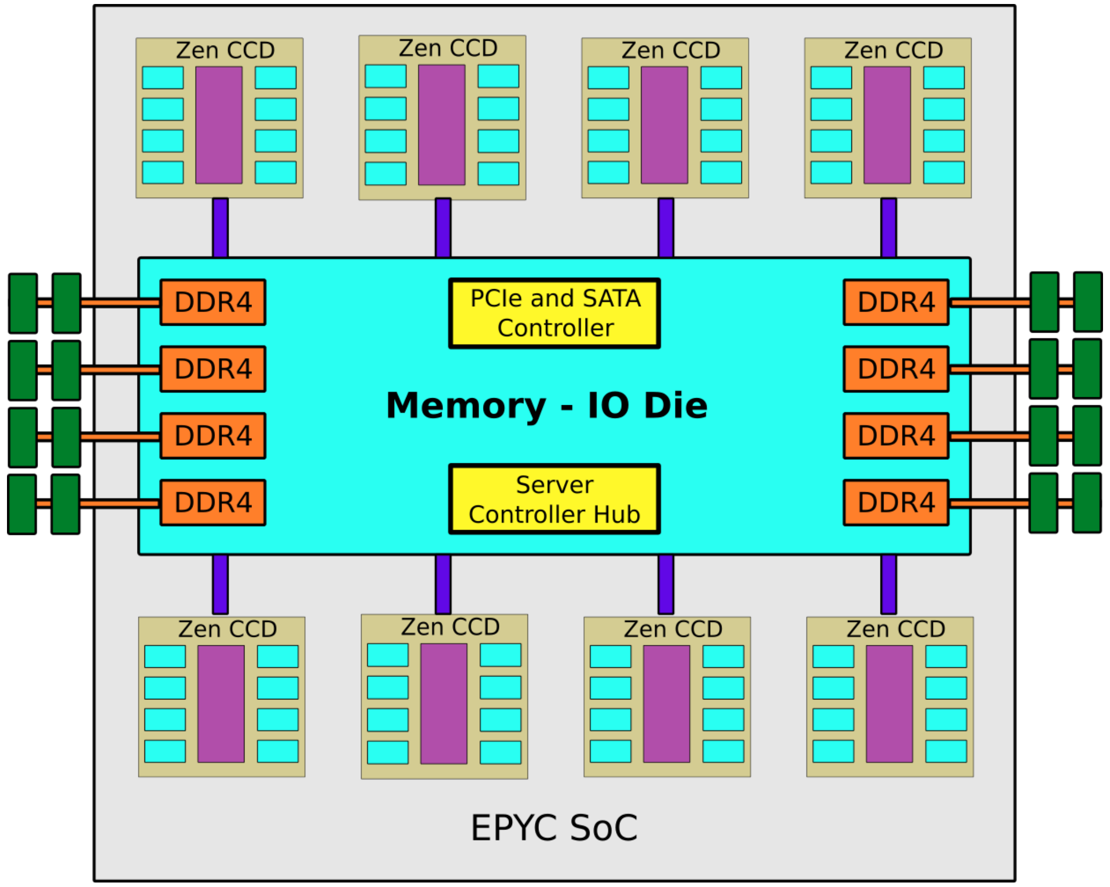
HPC architecture

Some terminology

- **MPI** *(Message Passing Interface)* - software library facilitating parallelism across multiple, potentially heterogeneous computational resources.
- **OpenMP**: software library facilitating shared-memory parallelism (e.g. within a single server or computer).
- **Process**: basic unit of parallelism employed by MPI. Processes operate as independent, persistent instances of a given program and do not share resources.
- **Thread**: basic unit of parallelism employed by OpenMP. Threads can be conceptually created and destroyed at will and share all resources on a machine.

HPC cluster architecture - CPUs





Schematic of AMD Zen3 node. Credit: Pawsey Supercomputing Centre

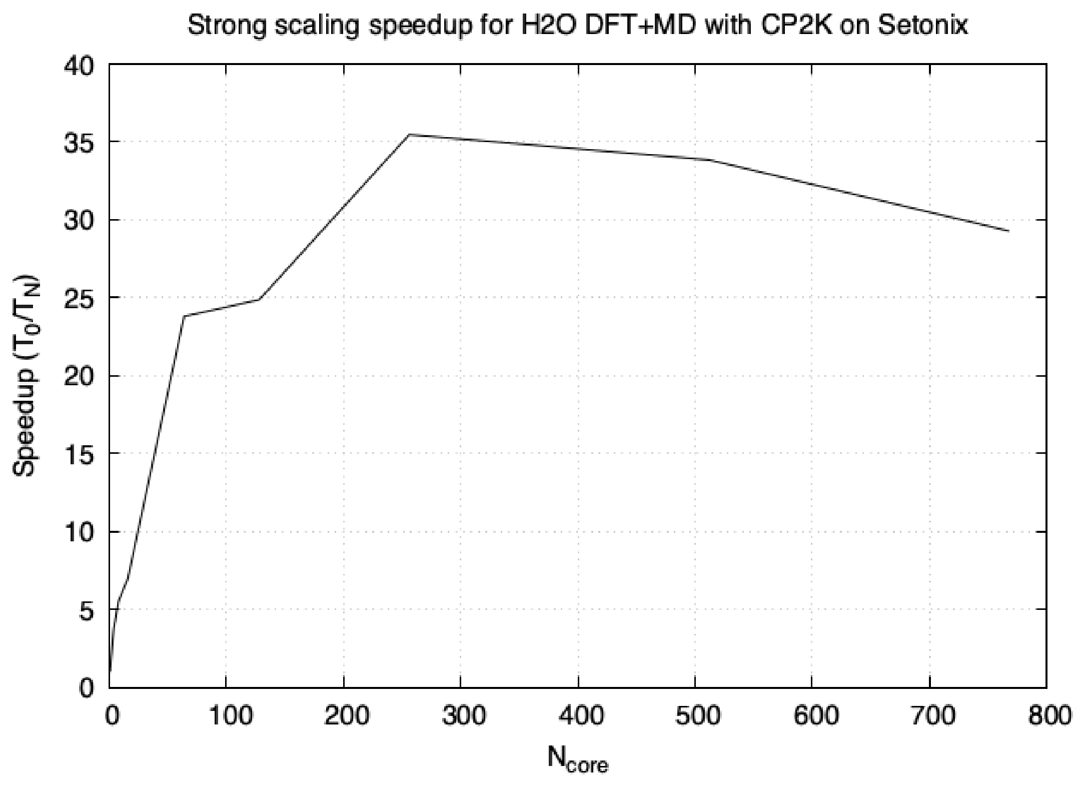
HPC cluster architectures

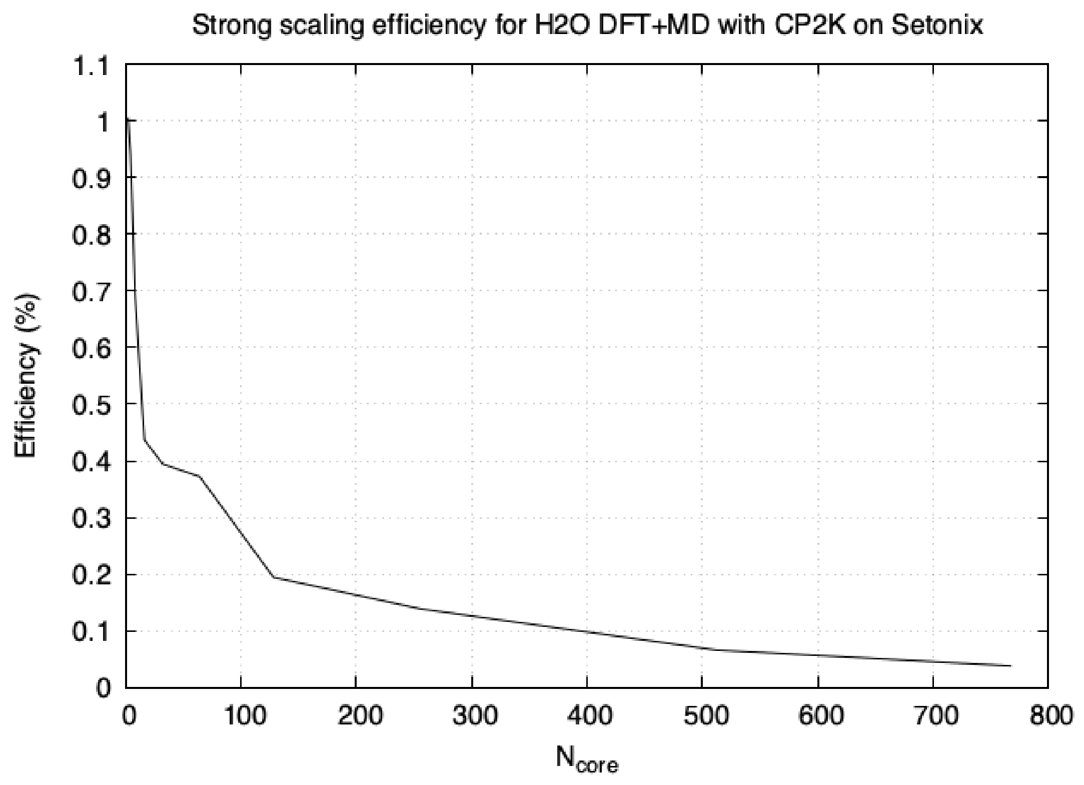
- Modern clusters are becoming increasingly heterogeneous (even for CPUs)
- Thread/core count per node can be misleading
  - Non-uniform performance
  - Threads on a single socket often share e.g. I/O bus – bandwidth saturation!
- Modern HPC processors are really multiple smaller processors that happen share a chip
- Memory hierarchies are also getting deep
  - Non-uniform memory access
  - Deep shared caches
- *Where* your code executes (e.g. MPI ranks, OpenMP threads) can make a big difference for performance

HPC performance – profile your code

- Can't know if you're doing the right thing if you don't measure
- We're scientists: treat optimising and deploying code like an experiment
  - Collect data
  - Analyse performance patterns
  - Form a hypothesis for how to improve
  - Collect data to test hypothesis
- Can be as simple as checking walltime when job's done (scheduler will usually do this for you)
- Large codes like LAMMPS, CP2K sometimes have this built in
- Dedicated profiling tools available on most clusters – ask your helpdesk or see the list at the end of these slides

Strong scaling – Amdahl’s law





HPC performance - MPI

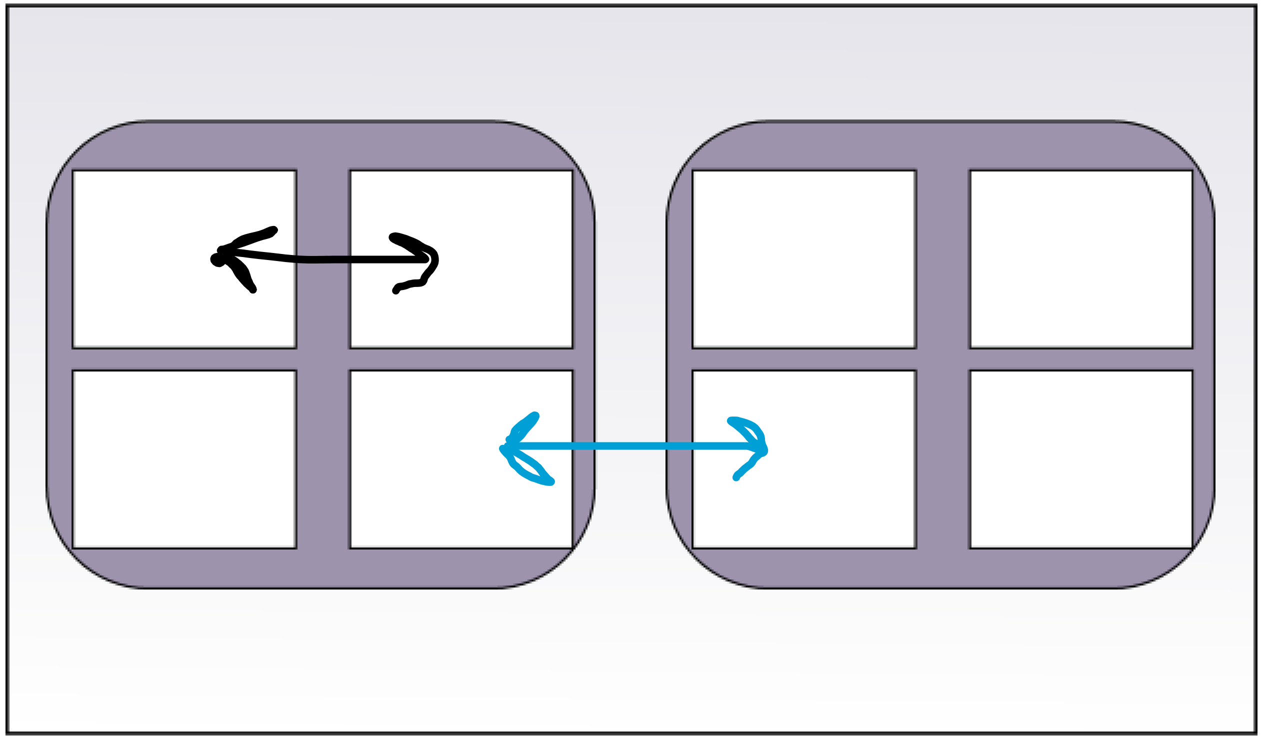
- MPI performance determined by two main hardware factors:
  1. Network latency
  2. Network bandwidth
- Relative importance varies between apps
  - "Speed" can refer to either of these, or both. Be clear about your needs
- Network congestion affects both, can't control this as a user
- You *can* control where processes are placed
  - Need to be aware of network topology for maximum performance

HPC network topology - intra-node

- MPI messages between processes on the same node use *shared memory*
  - Message passing performance dependent on memory access
- Modern CPU architecture has *Non-Uniform Memory Access* (NUMA):

● = low-latency

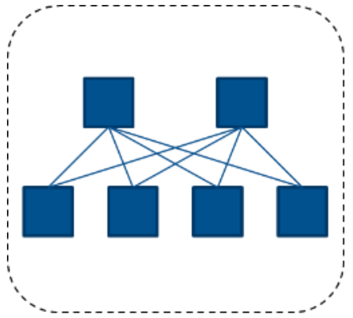
● = higher-latency

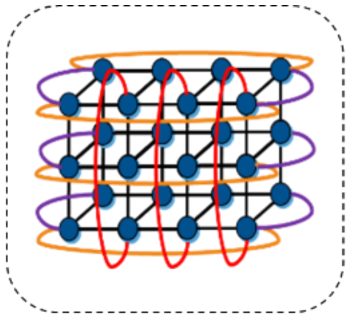


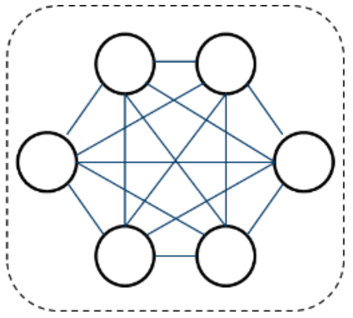
Single HPC node

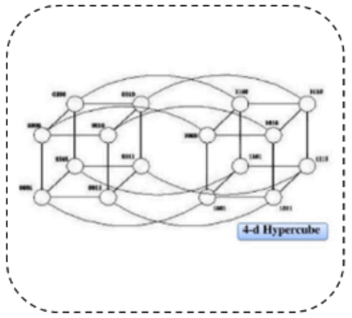
HPC network topology – inter-node

- Fast network interconnects put the "super" in "supercomputer"
- Multi-node latency very sensitive to *network topology* and process placement
- Many different topologies, differ between clusters
- Learn what your cluster uses and how to optimise for it
- E.g. Setonix and Gadi use *Dragonfly* and *Dragonfly*+ topologies, respectively

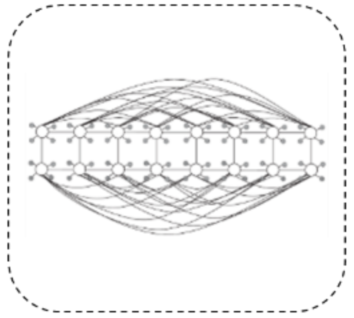

Fat Tree


Torus


Dragonfly



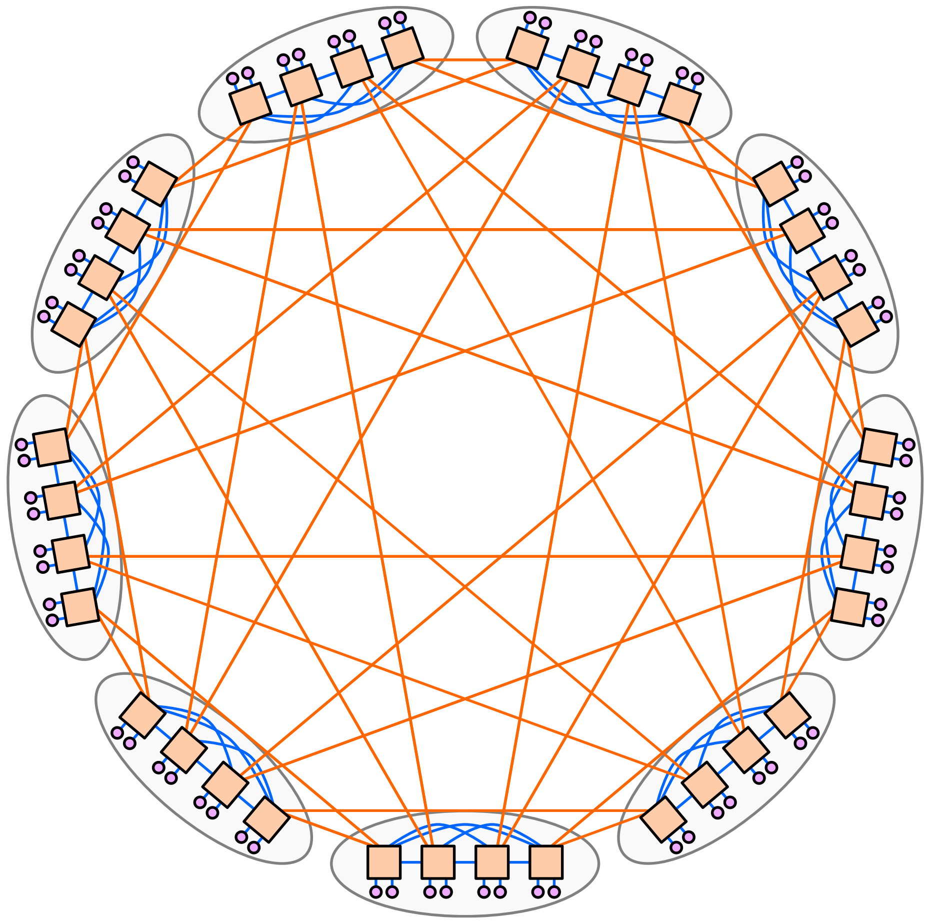
Hypercube


HyperX

Source: Mellanox Technologies Inc.

Example: Dragonfly topology

- Processing elements (PE, e.g. nodes) organised into *groups*
- Groups are *all-to-all* connected to each other:
  - One "external" switch per group
  - Few "long links"
- PEs within a group connected by different sub-topology (e.g. tree or torus)
- Latency (almost) constant for messages *within* groups, much worse for messages *between* groups



Graph of dragonfly topology. Source: https://commons.wikimedia.org/wiki/File:Dragonfly-topology.svg

Job schedulers

The lay of the land

- Multi-tenant clusters need some way to allocate and manage jobs/resources
- SLURM or PBS/Torque/SGE most common on HPC clusters
  - Typically focused on relatively homogeneous compute jobs (e.g large DFT jobs) - batch processing
  - Provide fine-grained control of resource placement/topology
  - Lots of monitoring/reporting (even profiling) tools
- Traditional schedulers poorly suited to multi-stage heterogeneous workflows
  - Data-intensive ML or bioinformatics pipelines are especially tricky
- New set of *workflow managers* gaining adoption:
  - Nextflow
  - Flux
  - Snakemake

SLURM

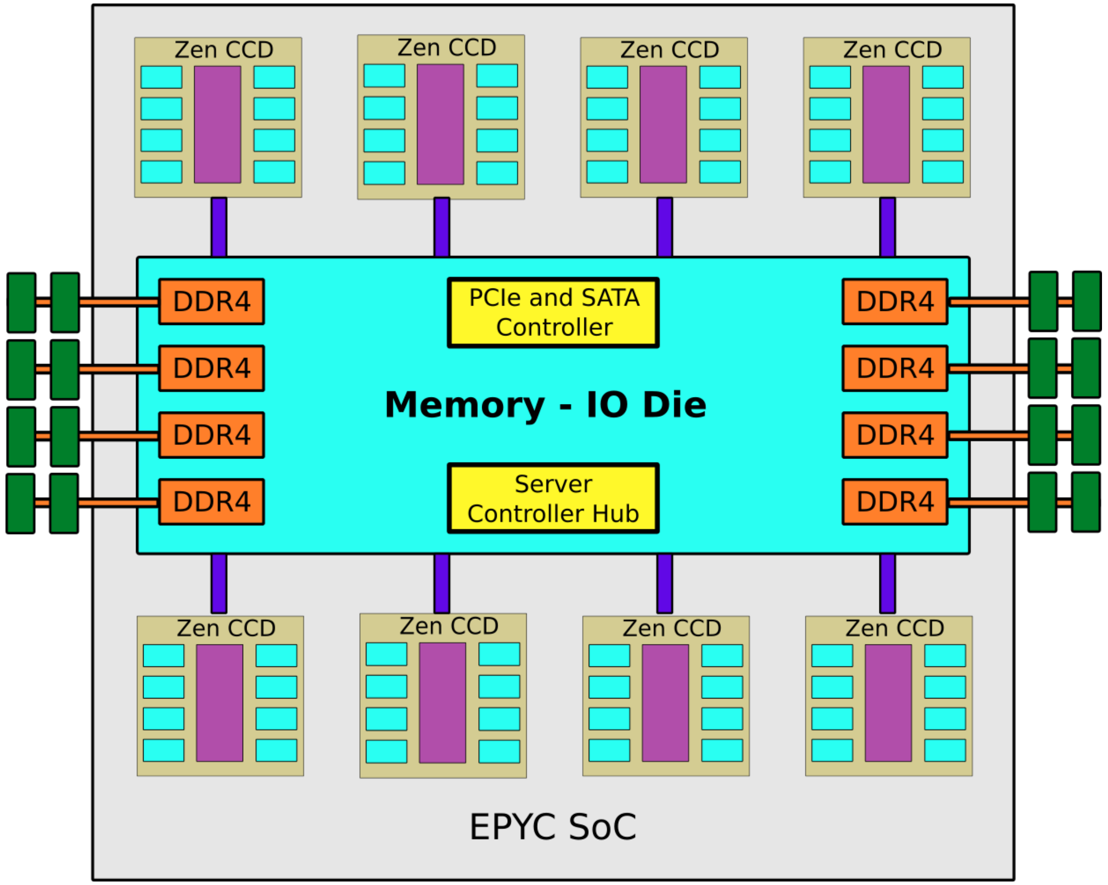
- Used by Setonix, UQ's Bunya
- Good support for heterogeneous architectures (e.g. GPUs, highly-nonuniform nodes, future weird stuff?)
- Extensive monitoring and logging capabilities
- UQ Research Computing Centre has a cool tool to generate SLURM script templates: https://shiny.rcc.uq.edu.au/SLURM/

SLURM – resource management

SLURM concept	Logical interpretation
NTASKS	MPI Ranks
CPUS	Physical CPU core
CPUS_PER_TASK	CPUs per MPI process (for e.g. MPI+OpenMP)
NODES	Physical compute node
SOCKETS	Physical socket/chiplet
GRES	Generic consumable resource (e.g. GPU)

SLURM – submitting

- Manage process/thread placement with srun
- Modern HPC processors are really multiple smaller processors that share a chip
- Memory access/messaging across boundaries (even within a node) is *slow*
- Lots of cores, so resource contention is also important
- HPC administrators will typically set good defaults for *most* jobs, not necessarily good for *your* job



AMD Zen 3 CPU architecture
Source: pawsey.org.au

SLURM - submitting jobs

```
srun --distribution={block|cyclic|arbitrary|plane}:{block|cyclic|arbitrary|plane}
```

- First option distribution of tasks to nodes, second controls distribution *within* nodes
- Example 1: MPI-only, processes only talk to "nearby" processes (e.g. neighbour list MD)
  - Scheduling adjacent processes on the same node/socket = optimise communication

```
srun --nodes 4 --tasks-per-node=128 --distribution=block:block lmp […]
```

- Example 2: MPI+OpenMP, MPI ranks talk to to "nearby" processes, one process per socket
  - Schedule nearby processes on adjacent nodes, spread out processes within nodes to minimise contention between threads

```
srun --nodes=4 --ntasks-per-node=8 --cpus-per-task=16 --distribution=block:cyclic
```

- As always, experiment and profile!

SLUM – monitoring and reporting

- sinfo – information about node status and partition occupancy
- squeue – what's the status of my jobs in the queue?
- sprio – what's my job's priority? Why isn't it running yet?

```
$ sinfo -O Partition,NodeList,Nodes,Gres,CPUs
PARTITION      NODELIST        NODES          GRES          CPUS
debug*         bun[006-067]    62             (null)        192
general        bun[006-067]    62             (null)        192
ai             bun[003-005]    3              gpu:a100:3    256
gpu            bun[001-002]    2              gpu:mi210:2   192
gpu            bun068          1              gpu:a100:2    192
```

SLURM - monitoring and reporting

- sstat – what's the status of my (running) job? CPU usage, memory usage, I/O patterns
- sacct – as above, but for historical jobs. Can specify ranges. Useful for coarse-grained profiling, keeping track of which jobs have finished/failed

```
$ sstat --format=JobID,AveCPU,MinCPU,AveVMSize -j 965786
JobID          AveCPU    MinCPU  AveVMSize
------------ ---------- ---------- ----------
965786       00:01:59  00:01:57   117936K
```

```
$ sacct -o jobid,jobname,NNodes,NCPUS,elapsed -S now-30days --user=$USER
JobID         JobName   NNodes     NCPUS    Elapsed
------------ ---------- -------- ---------- ---------- ----------
909263          cp2k-256        1          8   00:24:33
966144          build-lammps   1       256   00:00:35
```

SLURM – monitoring and reporting

- sreport - generate nice reports from SLURM job data for a range of jobs
  - Very useful when estimating resource requirements for grant applications

```
sreport cluster UserUtilizationByAccount -t Hours start=2022-01-01 Users=$USER
--------------------------------------------------------------------------------
Cluster/User/Account Utilization 1 Jan 2022 - Ystday 23:59 (36201600 secs)
Usage reported in CPU Hours
--------------------------------------------------------------------------------
  Cluster     Login     Proper Name         Account     Used   Energy
--------- --------- --------------- --------------- -------- --------
  setonix     ekahl     Emily Kahl              fc8     8577        0
```

SLURM – monitoring and reporting

- seff – generate detailed report about a *single* job, including CPU and memory efficiency
  - Important to gauge how well you're using the cluster
  - Want to aim for as close to 100% utilisation as possible
  - Sometimes number of cores can be misleading – figure to the right reports 4 cores, but that counts virtual cores which are unused and not included in job accounting (re-scale by 2 to get the physical cores): more info from Pawsey here

Example CP2K job:

```
$ seff 1026815
Job ID: 1026815
Cluster: setonix
User/Group: ekahl/ekahl
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 4
CPU Utilized: 01:23:52
CPU Efficiency: 49.39% of 02:49:48 core-walltime
Job Wall-clock time: 00:42:27
Memory Utilized: 188.32 MB (estimated maximum)
Memory Efficiency: 62.77% of 300.00 MB (300.00 MB/node)
```

PBS

- Scheduler used by Gadi, many older clusters
- Less rich support for thread/process placement than SLURM
- Have to use OpenMP environment variables and mpirun to control job placement/affinity
- Usually don't need these with SLURM (use srun instead)

OMP_NUM_THREADS	Number of OpenMP threads
OMP_PROC_BIND	Pin threads to CPU cores
OMP_PLACES	How to distribute threads on a node
mpirun --map-by	How to distribute MPI ranks

THE UNIVERSITY OF QUEENSLAND AUSTRALIA | CREATE CHANGE

High performance storage

POSIX filesystems

- UNIX-like systems use the POSIX standard
- Everything is a file, files live in a hierarchical directory structure
- /home, /scratch on clusters
- Directories spread across multiple disks/servers on clusters, but still have a single namespace: distributed file system, e.g. Lustre
- Familiar, convenient, works the same on every machine
- Doesn't scale well for very large data – abstractions are expensive!
- Strong limits on *number* of files – searching through metadata gets slow
- Usually have quotas on storage *and* number of files

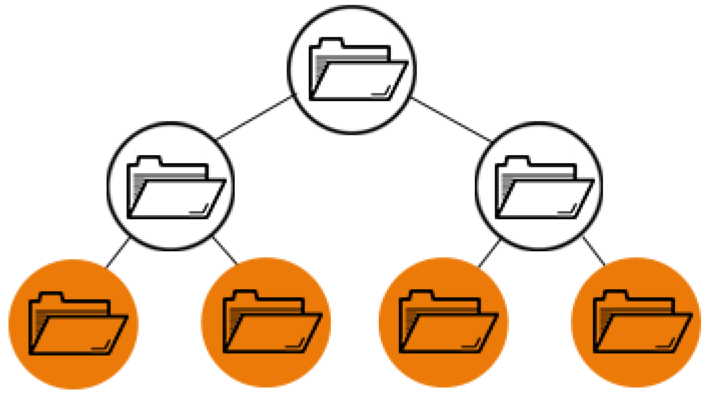


Image credit: RedHat, Inc.

Python on HPC

- Python packages can be very bad for HPC filesystem performance
  - e.g. Conda, pip
- Small, isolated packages probably okay
- Recursive dependencies can create LOTS of tiny files – really bad for Lustre performance
- Easy to exhaust quota on "number of files" (inodes)
- Try not to install python environments on the clusters
- Use containers where possible – image acts as a single file on disk but expands into a filesystem in memory
- Good resource/tutorial: https://pawseysc.github.io/singularity-containers/
- `(pip|conda) install --dry-run` is your friend!

Some Useful Tools and Resources

Observability tools

NOTE: These tools may not be available on all HPC systems. All of these are command-line tools and their output may require some interpretation.

- lstopo: print information about the topology of the node (CPU cores, memory regions, network bus, etc). Sub-program distributed with hwloc (itself a sub-project of OpenMPI). Will need to run via a batch or interactive job to see details of compute nodes
- lscpu: print detailed information about the CPU on the current node. Will also need to run via a batch or interactive job to see details of compute nodes
- LIKWID: set of tools for reporting and managing hardware and HPC resources (e.g. MPI). Some tools are more for system administrators, but some can be installed without superuser privileges. Link
- sinfo: SLURM tool to print information about nodes and partitions, including some information on hardware configuration

Profilers

- gprof: old-school, kinda rough, limited support for parallelism, built into GNU compilers and available everywhere: https://support.pawsey.org.au/documentation/display/US/Profiling+with+gprof
- Caliper: extremely fine-grained information, supports MPI/OpenMP/CUDA, open-source, requires source-code modifications: https://software.llnl.gov/Caliper/index.html
- Arm Forge: Very fine-grained metrics, tracing (can view program performance as a time-series), really good GUI, works with MPI/OpenMP/CUDA, proprietary, available on Gadi: https://opus.nci.org.au/display/Help/Arm+HPC+Tools
- Cray PAT/perftools: decent coverage of diagnostics, optional GUI, proprietary, available on Setonix (documentation)
- Intel VTune: good coverage of low-level diagnostics and tracing, good support for OpenMP, installed on Gadi, proprietary but costs $0: https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html
- HPCToolkit: fine-grained metrics and profiling, works with MPI/OpenMP and CUDA, open-source and widely available, somewhat complicated workflow: http://hpctoolkit.org/
- TAU: good support for profiling and tracing with MPI and OpenMP, optional GUI, open-source: https://www.cs.uoregon.edu/research/tau/home.php

Debuggers

- Can start and stop program execution and inspect program's state almost arbitrarily
- GDB is a superpower!
  - Available almost anywhere
  - Almost arbitrary control and observability for CPU-based programs.
  - Terminal based, but TUI exists
  - Good tutorial at Dive Into Systems
- NVIDIA/AMD have GPU-aware debuggers with GDB-like interface: cuda-gdb/rocm-gdb
- Distributed, MPI-aware debugging is hard, few available tools:
  - Arm DDT (proprietary) on Gadi: https://opus.nci.org.au/display/Help/Arm+HPC+Tools
  - gdb4hpc (Cray proprietary) on Setonix
  - TotalView, available on Gadi: https://opus.nci.org.au/display/Help/TotalView

Good resources

- CTCMS tutorials and guide (suggestions and contributions welcome!): https://ctcms-uq.github.io/
- UQ RCC docs: https://github.com/UQ-RCC/hpc-docs
- NCI's Gadi user guide: https://opus.nci.org.au/display/Help/Gadi+User+Guide
- Pawsey's Setonix user guide: https://support.pawsey.org.au/documentation/display/US/Setonix+User+Guide
- Victor Eijkhout, *The Art of HPC* (series of textbooks and guides)
- Paul E McKenney, Is Parallel Programming Hard, And, If So, What Can You Do About It? (textbook)
- Brendan Gregg's website and *and book* on performance engineering
- Julia Evans, *The Pocket Guide to Debugging*

THE UNIVERSITY OF QUEENSLAND AUSTRALIA CREATE CHANGE

Thank you!

Object storage

- Qualitatively different from POSIX filesystems
- Stores *unstructured* data – no folders, no hierarchy
- Scales better than POSIX for very large data sets
- *Objects* stored in *buckets* – unique ID + rich metadata for search
- Splits reads/writes from namespace manipulation
- Pull data -> read/modify -> push data
- Good for data which is read more often than it's written (e.g. molecular geometry files)
- Used by Pawsey (Acacia), commercial cloud services (e.g. Amazon S3) - get used to it!

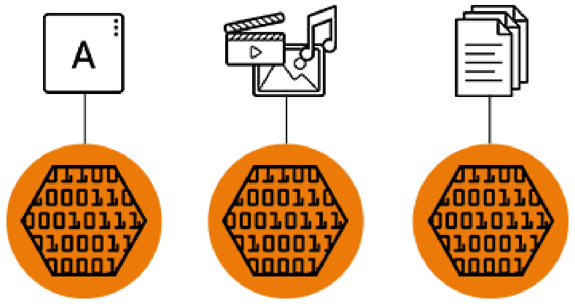


Image credit: RedHat, Inc.