Advanced Cluster Usage

Harvard - FAS Research Computing

Topics

- Advanced cluster usage
- Job resource requirement and efficiency
- Job, partition, and queue monitoring
- Job Efficiency
- Fairshare

Training Material

```
# Download slides: https://docs.rc.fas.harvard.edu/kb/training-materials/

# Login to Cannon
ssh <username>@login.rc.fas.harvard.edu

# Clone FASRC User Codes repository:
https://github.com/fasrc/User_Codes/tree/master
SSH - git clone git@github.com:fasrc/User_Codes.git
HTTPS - git clone https://github.com/fasrc/User_Codes.git

# Go to the training folder:
cd User_codes/Training/Advanced_Cluster_Usage
```

Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Workshop Reservation

Add the following to any jobs & be mindful of the partition:

Interactive:

```
salloc --reservation=workshop -p sapphire
```

Batch:

```
#SBATCH --reservation=workshop
#SBATCH -p sapphire
```

If you have problems please let us know.

Introduction to Advanced Cluster Usage

Paula Sanematsu

FASRC clusters

Massachusetts Green HPC Center (MGHPCC)

Cannon cluster

From https://www.servethehome.com/the-harvard-cannon-powered-by-lenovo-neptune/

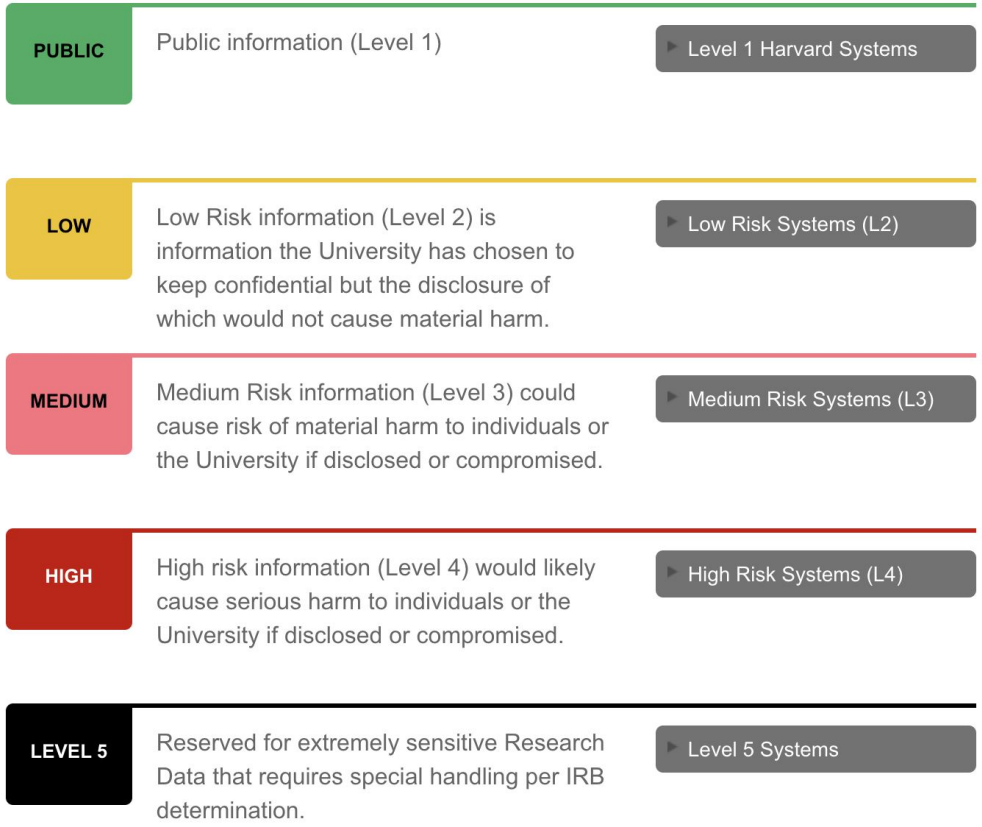
FASRC clusters: Cannon and FASSE

Cannon

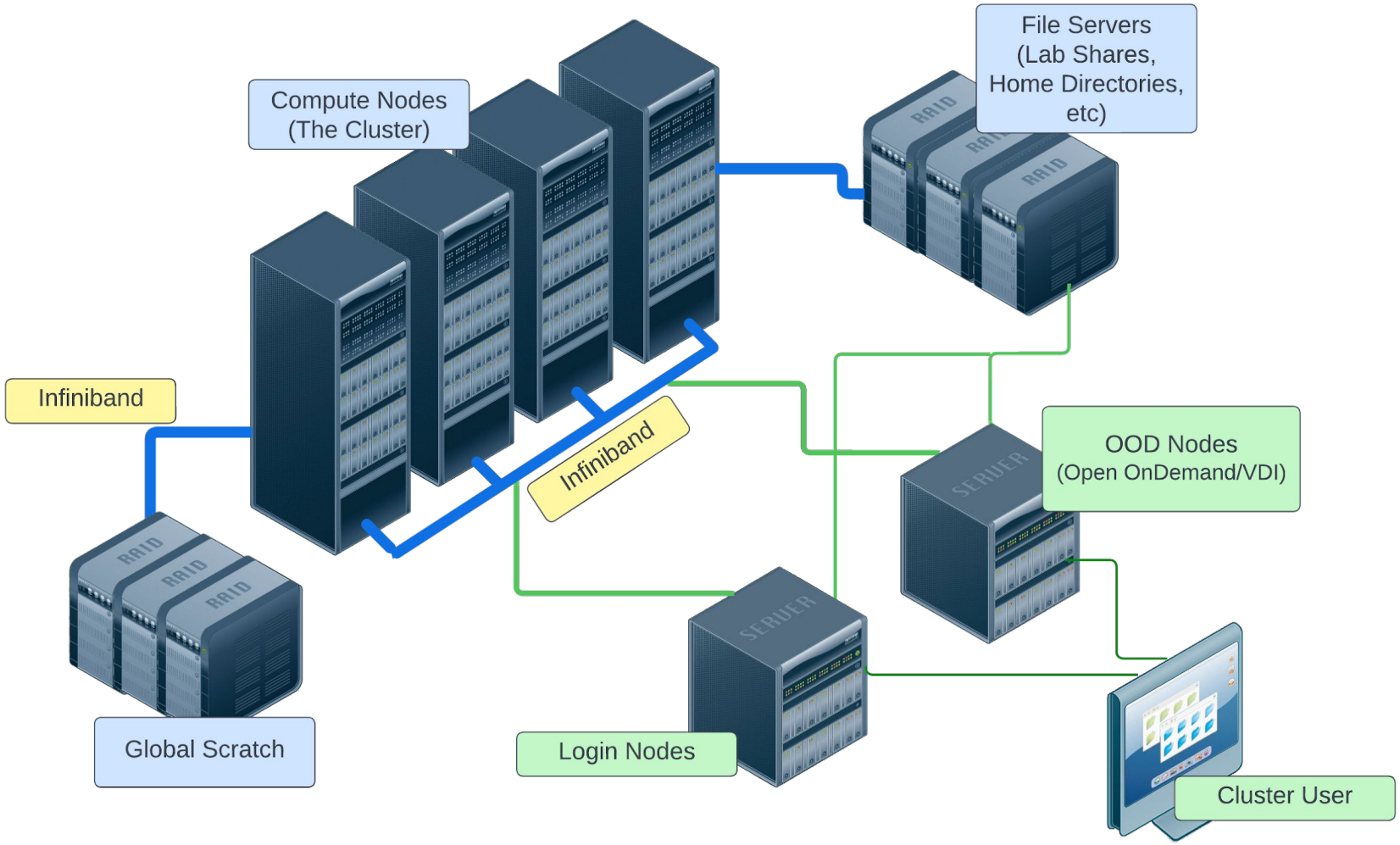
- General purpose
- Only level 1 and 2 data

FASSE

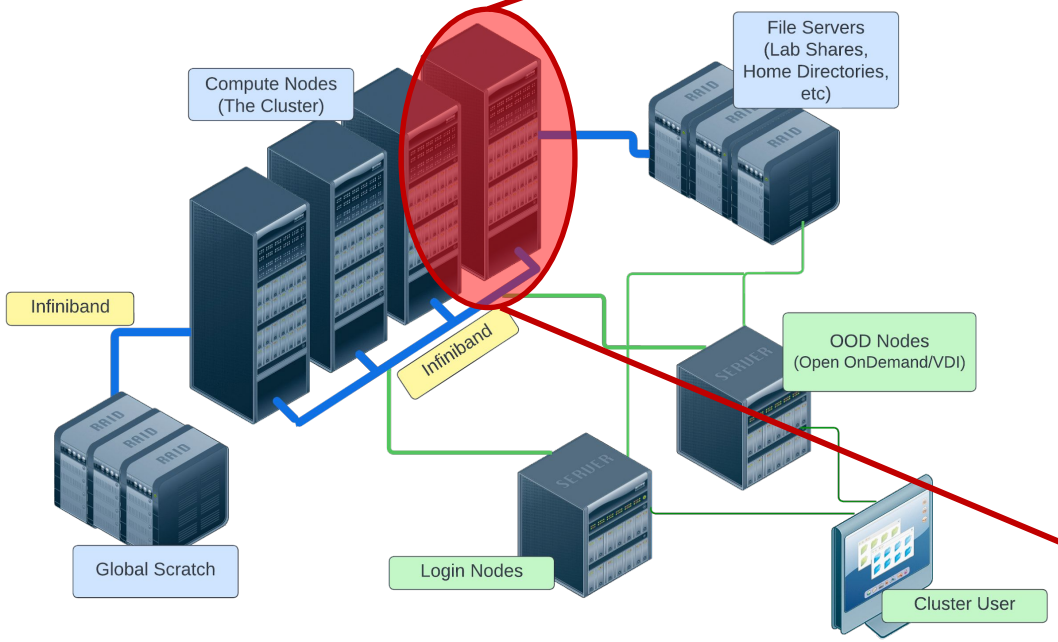
- FAS Secure Environment
- Secure multi-tenant environment
- Analysis of sensitive datasets with DUAs and IRBs
- Level 3 data, no level 4 data
- PI/lab responsibility to know their data
- https://policy.security.harvard.edu/
- https://docs.rc.fas.harvard.edu/kb/data-use-agreements/
- https://security.harvard.edu/
- https://docs.rc.fas.harvard.edu/kb/fasse/

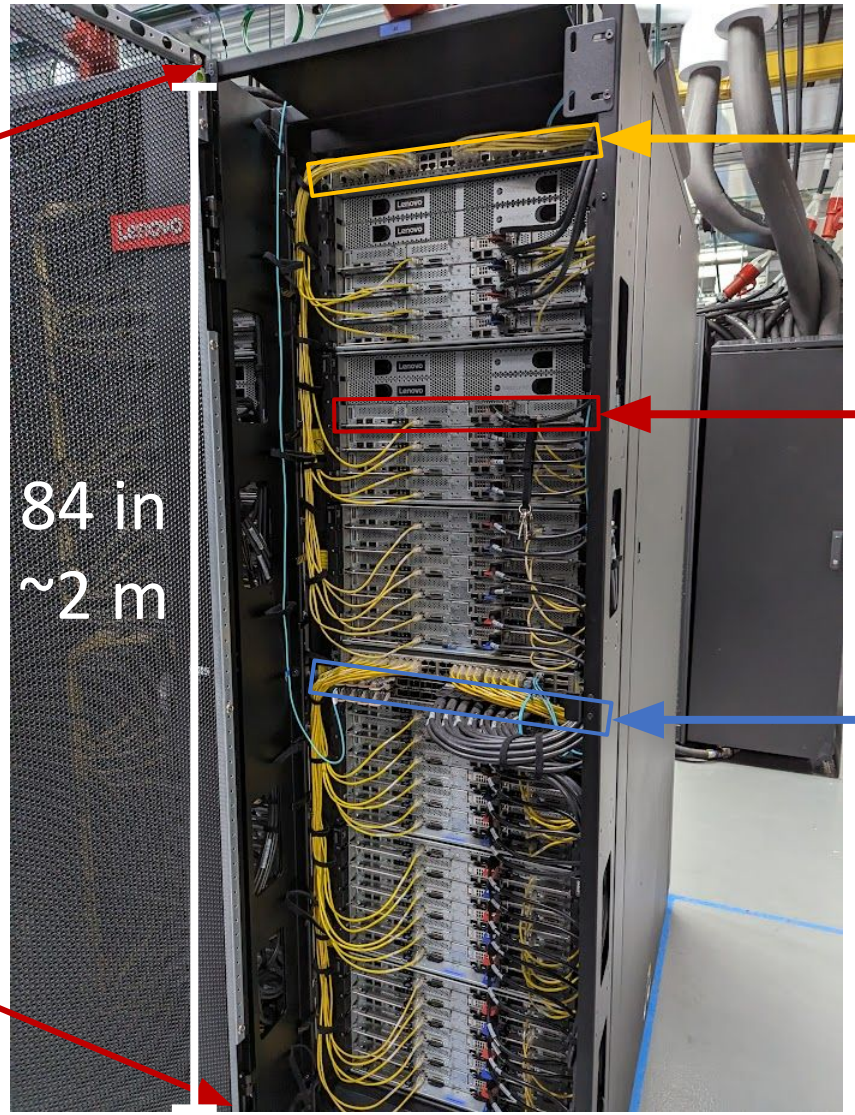


Cluster architecture



Rack



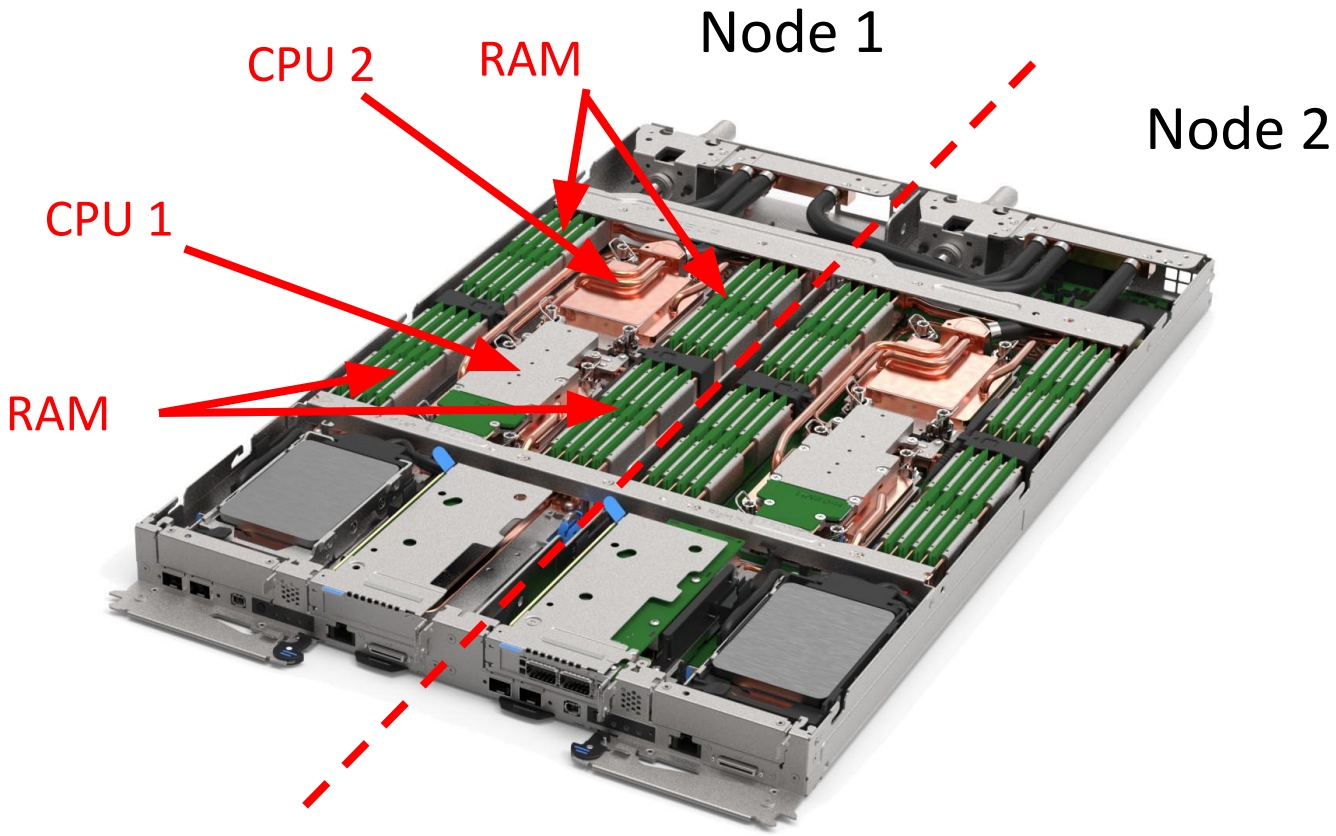


Ethernet switch

Compute nodes

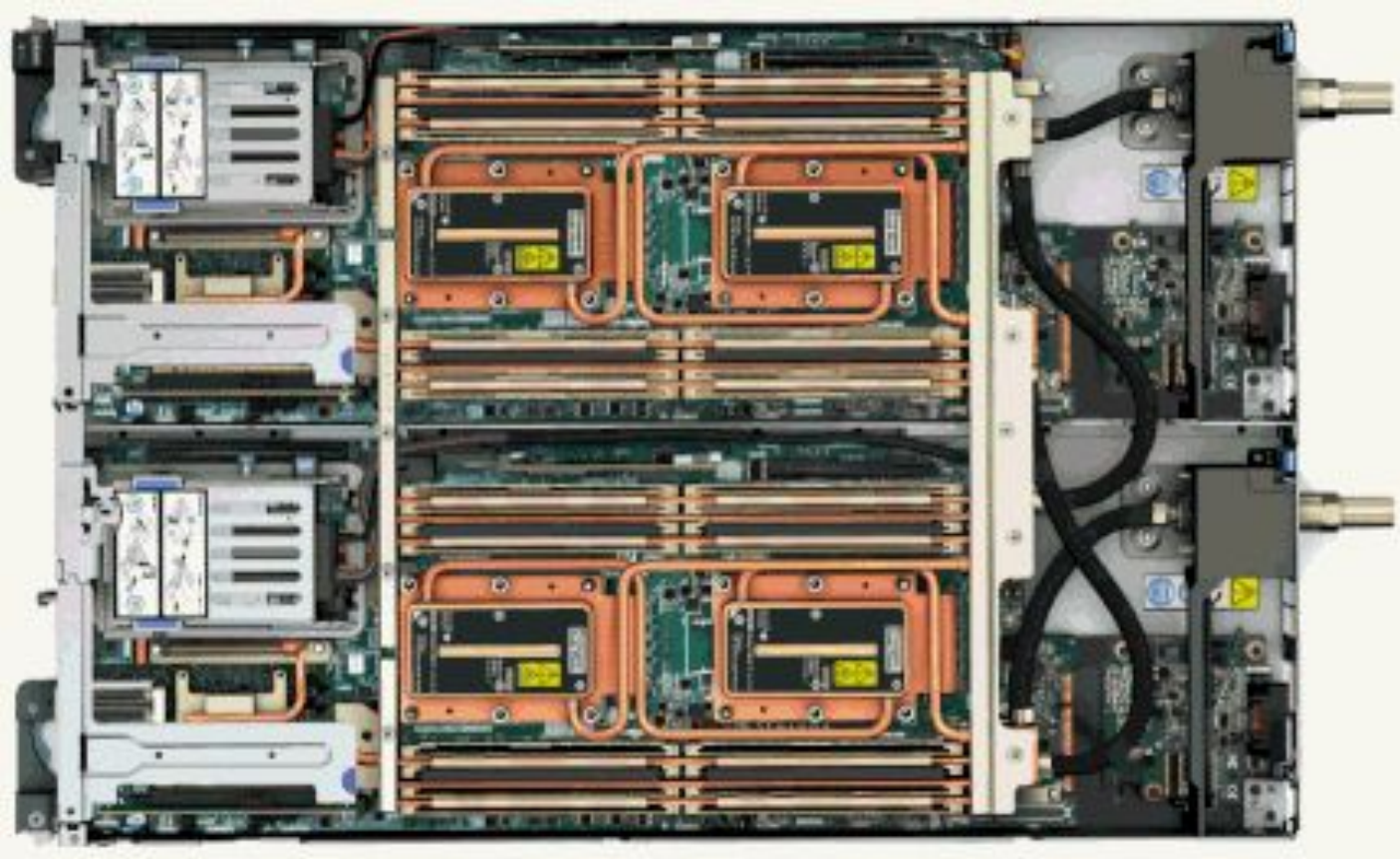
Infiniband switch

Node



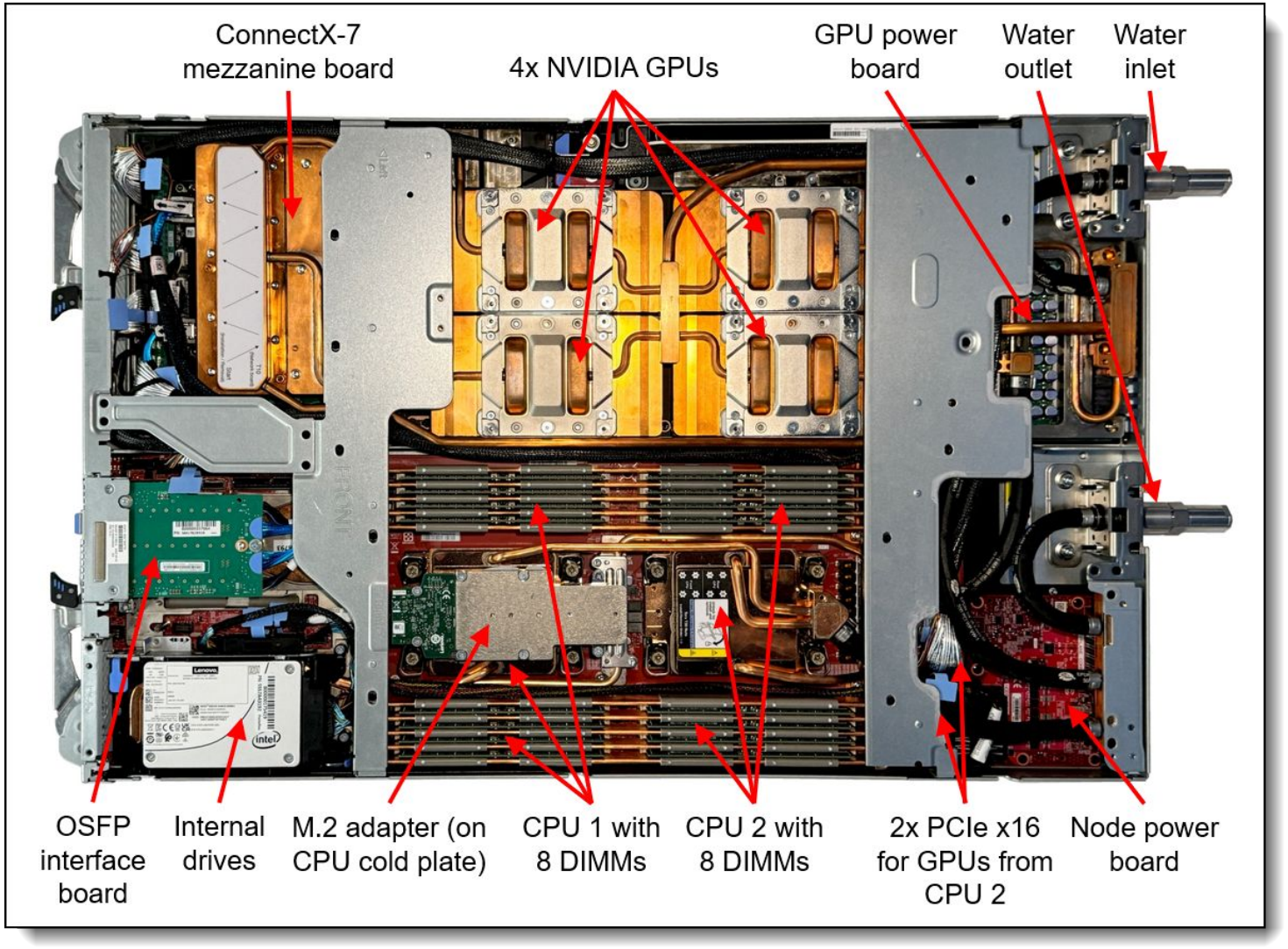
From https://lenovopress.lenovo.com/lp1603-thinksystem-sd650-v3-server

CPU node and water cooling



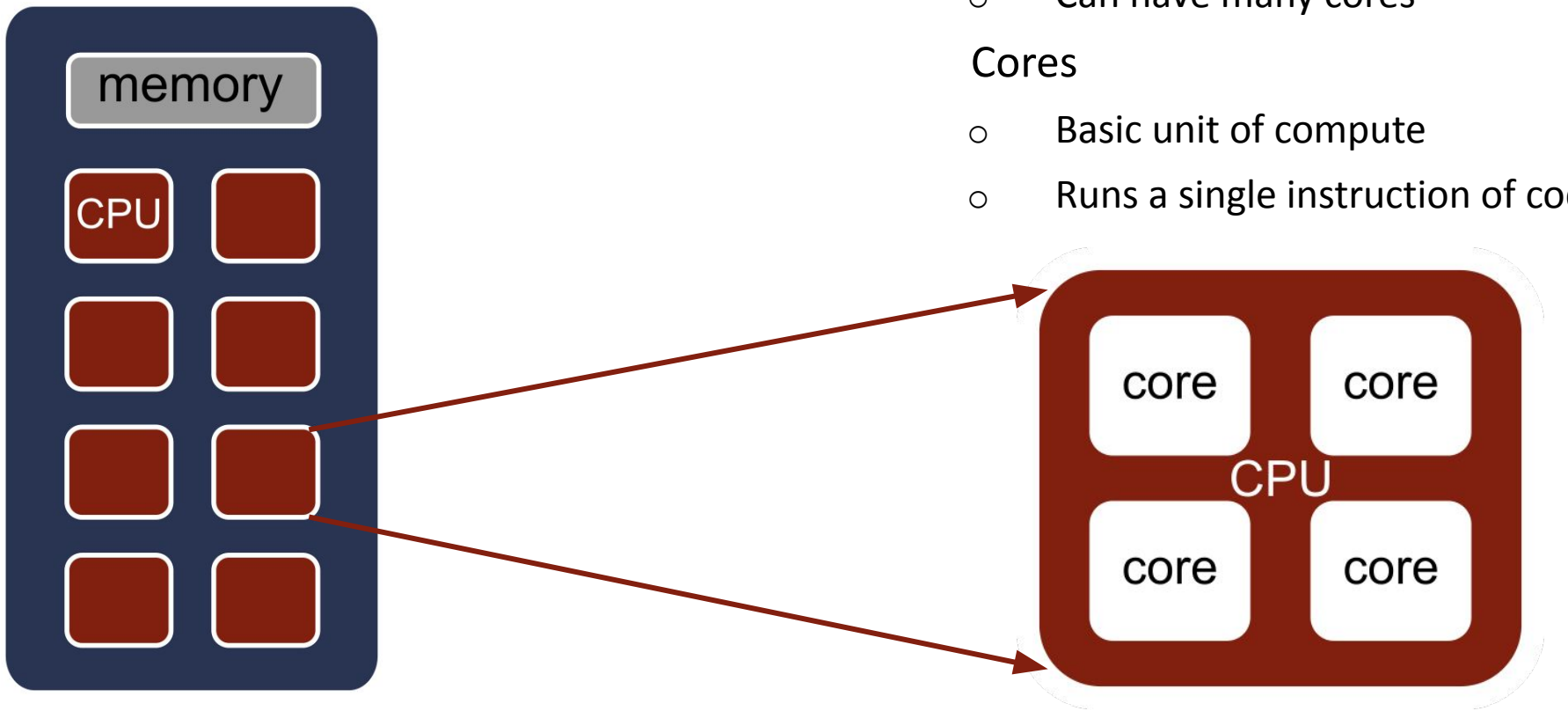
From https://www.rc.fas.harvard.edu/blog/cannon-makes-top-500-list/

GPU node


ConnectX-7 mezzanine board
4x NVIDIA GPUs
GPU power board
Water outlet
Water inlet
OSFP interface board
Internal drives
M.2 adapter (on CPU cold plate)
CPU 1 with 8 DIMMs
CPU 2 with 8 DIMMs
2x PCIe x16 for GPUs from CPU 2
Node power board

Node, processors, core

Node: a computer in the cluster



CPU

- Central processing unit, processor
- Can have many cores

Cores

- Basic unit of compute
- Runs a single instruction of code

Job scheduler - SLURM

- The Cluster is a multi-tenant environment, so how can everyone use it fairly?
- Job scheduler!
- Slurm: Simple Linux Utility for Resource Management
  - Manages job queue for a cluster of resources
  - Prioritizes jobs
  - Provides status of running, queue, completed and failed jobs
  - Determines the order jobs are executed
  - On which node(s) jobs are executed

Adapted from HPC@LSU training (http://www.hpc.lsu.edu/training/weekly-materials/2021-Summer/HPC_UserEnv_2021_Summer_session_2.pdf)

Cannon partitions

Documentation: https://docs.rc.fas.harvard.edu/kb/running-jobs/

Partitions	sapphire	shared	gpu	test	gpu_test	serial_requeue	gpu_requeue	bigmem	intermediate	bigmem_ intermediate	unrestricted	**pi_lab**
Time Limit	3 days	3 days	3 days	12 h	12 h	3 days	3 days	3 days	3-14 days	3-14 days	no limit	**varies**
# Nodes	186	288	36	18	14	varies	varies	4	12	3	8	**varies**
# Cores / Node	112	48	64 + 4 A100	112	64 + 8 A100 MIG	varies	varies	112	112	64	48	**varies**
Memory / Node (GB)	990	184	990	990	487	varies	varies	1988	990	2000	184	**varies**

FASSE partitions

Documentation: https://docs.rc.fas.harvard.edu/kb/fasse/

Partitions	fasse	fasse_gpu	test	serial_requeue	fasse_bigmem	fasse_ultramem	remoteviz	**pi_lab**
Time Limit	7 days	7 days	12 h	7 days	7 days	7 days	7 days	**varies**
# Nodes	42	4	5	varies	17	1	1	**varies**
# Cores / Node	48	64 + 4 A100	48	varies	64	64	32	**varies**
Memory / Node (GB)	184	487	184	varies	499	2000	373	**varies**

Which partitions can I use?

Documentation: https://docs.rc.fas.harvard.edu/kb/convenient-slurm-commands/

```
[jharvard@boslogin08 ~]$ spart
Partition                State       Cores       GPUs        Average Mem/Node(GB)      Nodes       Time Limit
bigmem                   UP          448         0           2015                      4           3-00:00:00
bigmem_intermediate      UP          192         0           2015                      3           14-00:00:00
gpu                      UP          2304        144         1007                      36          3-00:00:00
gpu_requeue              UP          22448       1150        1174                      278         3-00:00:00
gpu_test                 UP          896         112         503                       14          12:00:00
intermediate             UP          1344        0           1007                      12          14-00:00:00
remoteviz                UP          32          0           377                       1           3-00:00:00
sapphire                 UP          20832       0           1007                      186         3-00:00:00
serial_requeue           UP          107952      1150        592                       1546        3-00:00:00
shared                   UP          17760       0           188                       370         3-00:00:00
test                     UP          2016        0           1007                      18          12:00:00
unrestricted             UP          384         0           188                       8           UNLIMITED
```

Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Job Resource Requirements

Paul Edmon

Resource Specification - Overview

Key Computational Resources

- Partition
- Nodes
- CPUs
- Memory
- Time limits
- GPUs

Key SLURM Directives

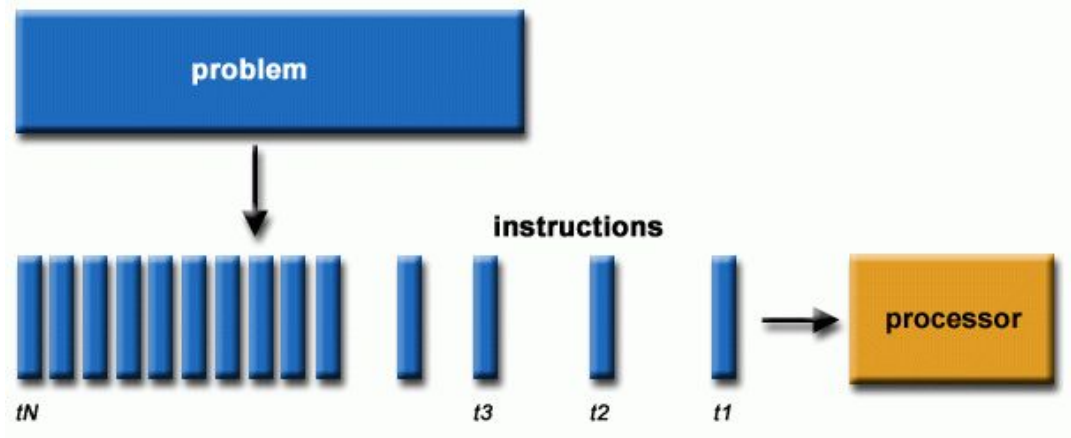
```
--partition / -p
--nodes / -N
--ntasks / -n
--cpus-per-task / -c
--mem
--mem-per-cpu
--time / -t
--gres
```

Resource Specification - Serial Jobs

Example serial job submission script

```
#!/bin/bash
#SBATCH -J serial_test
#SBATCH -N 1
#SBATCH -c 1
#SBATCH -p shared
#SBATCH -t 0-12:00:00
#SBATCH --mem=8G
#SBATCH -o serial_test_%A.out
#SBATCH -e serial_test_%A.err

module load <Required Software Modules>  # Optional
srun -c $SLURM_CPUS_PER_TASK  ./serial_test.x
```



Job Characteristics

- Single node
- One task / core per node
- Memory per node

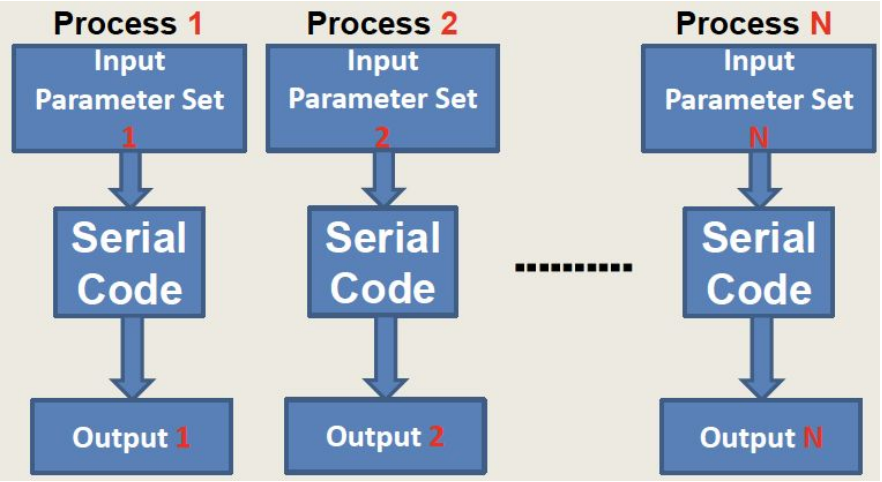
Resource Specification - Array Jobs

Example array job submission script

```
#!/bin/bash
#SBATCH -J array_test
#SBATCH -p test
#SBATCH -c 1
#SBATCH -t 00:20:00
#SBATCH --mem=4G
#SBATCH -o %A-%a.o
#SBATCH -e %A-%a.e
#SBATCH --array=100,200,300

# Load required software modules
module load python/3.10.13-fasrc01

srun -c 1 python serial_sum.py >
output_${SLURM_ARRAY_TASK_ID} .out
```



Job Characteristics

- Multiple job instances
- One core per job instance
- Memory per job instance

https://github.com/fasrc/User_Codes/tree/master/Parallel_Computing/EP/Example1

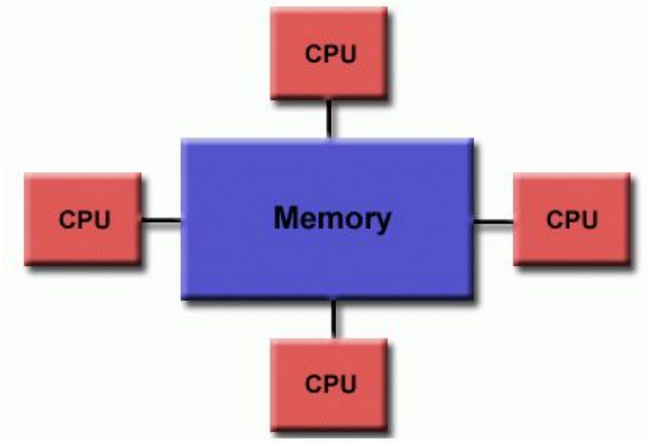
Resource Specification - Threaded (OpenMP) Jobs

Example threaded job submission script

```
#!/bin/bash
#SBATCH -J omp_diag
#SBATCH -o omp_diag.out
#SBATCH -e omp_diag.err
#SBATCH -p test
#SBATCH -t 00:30:00
#SBATCH -N 1
#SBATCH -c 4
#SBATCH --mem=4G

# Set up environment
module load intel/24.0.1-fasrc01
module load intel-mkl/24.0.1-fasrc01

# Run program
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK  ./omp_diag.x
```



Job Characteristics

- Single node
- Multiple cores on the same node
- Memory per node (shared memory parallelism)

https://github.com/fasrc/User_Codes/tree/master/Parallel_Computing/OpenMP/Example2

Resource Specification - Rank-based (MPI) Jobs

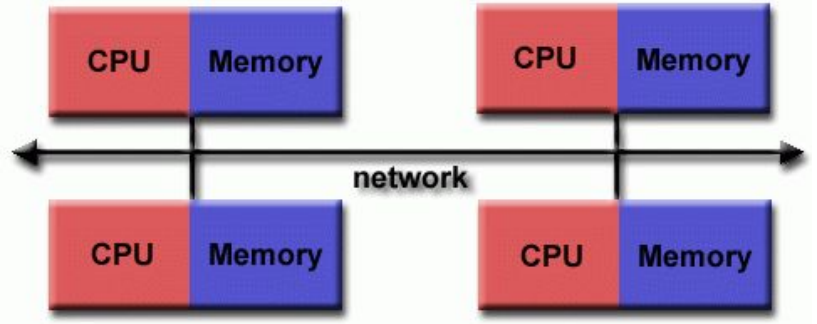
Example MPI job submission script

```
#!/bin/bash
#SBATCH -J mpitest              # job name
#SBATCH -o mpitest.out          # standard output file
#SBATCH -e mpitest.err          # standard error file
#SBATCH -p shared               # partition
#SBATCH -n 8                    # ntasks
#SBATCH -t 00:30:00             # time in HH:MM:SS
#SBATCH --mem-per-cpu=500       # memory in megabytes

# --- Load the required software modules., e.g., ---
module load gcc/13.2.0-fasrc01 openmpi/4.1.5-fasrc03

# --- Run the executable ---
srun -n $SLURM_NTASKS --mpi=pmix  ./mpitest.x
```

https://github.com/fasrc/User_Codes/tree/master/Parallel_Computing/MPI



Job Characteristics

- Single node or multiple nodes
- Many tasks on the same node or many nodes
- Memory per task (distributed memory parallelism)

Useful SLURM Options for MPI Jobs

```
--mem-per-cpu
--ntasks-per-node
--nodes/-N
--contiguous
--distribution/-m
```

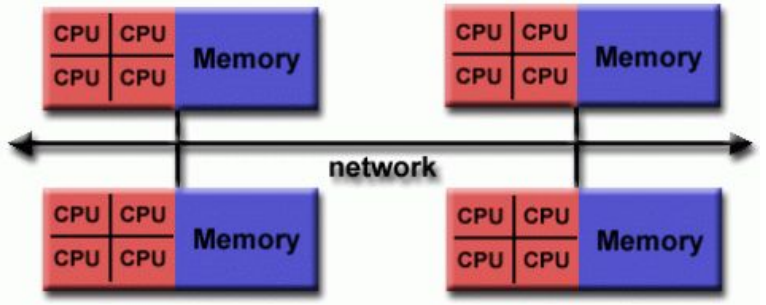
Resource Specification - Hybrid (MPI+OpenMP) Jobs

Example hybrid MPI+OpenMP job submission script: 8 nodes x 1 MPI tasks per node x 64 OMP threads per MPI task = **512** cores across 8 compute nodes

```
#!/bin/bash
#SBATCH -J hybrid_test          # job name
#SBATCH -o hybrid_test.out      # standard output file
#SBATCH -e hybrid_test.err      # standard error file
#SBATCH -p sapphire             # partition
#SBATCH --nodes=8               # Number of nodes
#SBATCH --ntasks-per-node=1     # Tasks per node
#SBATCH --cpus-per-task=64      # Number of cores per MPI task
#SBATCH --mem-per-cpu=256GB     # memory in megabytes
#SBATCH -t 00:30:00             # time in HH:MM:SS

# --- Load the required software modules., e.g., ---
module load gcc/13.2.0-fasrc01 openmpi/4.1.5-fasrc03

# --- Run the executable ---
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK -n $SLURM_NTASKS --mpi=pmix
./hybrid_test.x
```



Resource Specification - GPU Jobs

```
#!/bin/bash
#SBATCH -p gpu_test
#SBATCH -c 1
#SBATCH --gres=gpu:1
#SBATCH --mem=12000
#SBATCH -J cuda_test
#SBATCH -o cuda_test.out
#SBATCH -e cuda_test.err
#SBATCH -t 30

# Load required modules
module load cuda/12.4.1-fasrc01

# Run the executable
./saxpy.x
```

Job Characteristics

- Single node
- One CPU core
- One GPU
- Memory per node

https://github.com/fasrc/User_Codes/tree/master/GPU_Computing/CUDA

Advanced Options

--contiguous

Ensures you end up with an adjacent block of nodes

--tmp=<size>

Specifies the size of /scratch

--exclusive[={user|mcs}]

Ensures only you or members of your lab are on a node

--no-requeue

Slurm will cancel the job rather than requeue it

Hint

--hint=<type>

Lets the scheduler know that this calculation is of a specific type

compute_bound

Fills up one socket, and then the next

memory_bound

Lays out only one core per socket

Constraints

--constraint

List of required features

--prefer

List of preferred features

Logic Symbols

&: And

|: Or

Example:

#SBATCH --constraint=”intel&holyndr”

#SBATCH --prefer=”sapphirerapids”

Example Features

Processor Manufacturer: amd, intel

Processor Type: cascadelake, sapphirerapids

Processor Feature: avx, avx2, avx512

GPU Type: a100, h100

Infiniband Network Type: holyhdr, holyndr

https://docs.rc.fas.harvard.edu/kb/running-jobs/#Job_Constraints

Dependencies

--dependency=<option>

Allows for a job to only run when a previous condition is met

afterok:JOBID

Job will run after JOBID completes successfully

afternotok:JOBID

Job will run after JOBID fails

singleton

Only one job of this name can run at a time

https://docs.rc.fas.harvard.edu/kb/running-jobs/#Job_dependencies

Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Job/Queue/Partition Monitoring

Manasvita Joshi

Job Monitoring - `sacct`

Documentation: https://docs.rc.fas.harvard.edu/kb/convenient-slurm-commands/

- Every 30 sec, node collects the amount of cpu and memory usage that all of the process ID are using for a given job. After the job ends, this data is sent to SLURM database (slurm db)
- `sacct`: displays accounting data for jobs & job steps in slurm db. Lets you examine pending, running, & finished jobs.
- Common flags (i.e., options)
  - `-j jobid or --name=jobname`
  - `-S starttime YYYY-MM-DD and -E endtime YYYY-MM-DD`
  - `-o output_options: sacct -j <jobid> -o submitline -P` (command line arguments while submission)
  - See slurm docs for more options: https://slurm.schedmd.com/sacct.html

```
[jharvard@boslogin01 ~]$ sacct -j 2742999 --format=JobID,Jobname,partition,state,time,start,end,elapsed,MaxRss,MaxVMSize,nnodes,ncpus,nodelist --units=G

JobID           JobName  Partition      State  Timelimit               Start                 End    Elapsed     MaxRSS  MaxVMSize   NNodes      NCPUS        NodeList
------------ ---------- ---------- ---------- ---------- ------------------- ------------------- ---------- ---------- ---------- -------- ---------- ---------------
2742999          py_job       test  COMPLETED   00:30:00 2023-09-21T12:03:20 2023-09-21T12:03:21   00:00:01                              1          1     holy7c02410
2742999.bat+      batch              COMPLETED            2023-09-21T12:03:20 2023-09-21T12:03:21   00:00:01      0.01G      0.21G         1          1     holy7c02410
2742999.ext+     extern              COMPLETED            2023-09-21T12:03:20 2023-09-21T12:03:21   00:00:01      0.00G      0.17G         1          1     holy7c02410
```

Job Monitoring - `scontrol`

- `scontrol`: for viewing & controlling queued or running jobs.
- Options: `show, suspend, release`
- See https://curc.readthedocs.io/en/latest/running-jobs/slurm-commands.html

```
[jharvard@boslogin08 ~]# scontrol show job 1858242

JobId=1858242 JobName=snakejob.duplicates_23S.203831.sh
   UserId=dhelekal(64639) GroupId=grad_lab(5085) MCS_label=N/A
   Priority=12681081 Nice=0 Account=grad_lab QOS=normal
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:50 TimeLimit=00:06:00 TimeMin=N/A
   SubmitTime=2025-02-02T20:16:17 EligibleTime=2025-02-02T20:16:17
   AccrueTime=2025-02-02T20:16:17
   StartTime=2025-02-02T20:16:18 EndTime=2025-02-02T20:17:08 Deadline=N/A
   PreemptEligibleTime=2025-02-02T20:16:18 PreemptTime=None
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2025-02-02T20:16:18 Scheduler=Main
   Partition=serial_requeue AllocNode:Sid=holy8a32312:1602528
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=holy7c02405
   BatchHost=holy7c02405
   StepMgrEnabled=Yes
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   ReqTRES=cpu=1,mem=2000M,node=1
   AllocTRES=cpu=1,mem=2000M,node=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=2000M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/n/netscratch/grad_lab/Lab/dhelekal/cdc_GISP_2020-2023/.snakemake/tmp.u7auutal/snakejob.duplicates_23S.203831.sh
   WorkDir=/n/netscratch/grad_lab/Lab/dhelekal/cdc_GISP_2020-2023
   StdErr=/n/netscratch/grad_lab/Lab/dhelekal/cdc_GISP_2020-2023/logs/slurm/duplicates_23S_sample=SRR23441151.err
   StdIn=/dev/null
   StdOut=/n/netscratch/grad_lab/Lab/dhelekal/cdc_GISP_2020-2023/logs/slurm/duplicates_23S_sample=SRR23441151.out
```

Queue Monitoring - `squeue`

- `squeue`: list all current jobs, running or pending, in Slurm queue;
- Default display:
  - jobid, partition, job name, username, job status,
  - time running or pending depending on the status,
  - number of nodes,
  - name of nodes or reason if pending

- See https://slurm.schedmd.com/squeue.html

```
# List all current jobs for a user
[jharvard@boslogin08 ~]# squeue -u <username>

# List all current jobs for a users on a desired partition
[jharvard@boslogin08 ~]# squeue -u <username> -p <partitioname>
```

Partition Monitoring - `sinfo`

`sinfo`: View information about SLURM nodes and partitions

Useful Options

```
  -a, --all                  show all partitions (including hidden and those not accessible)
  -N, --Node                 Node-centric format
  -o, --format=format        format specification
  -O, --Format=format        long format specification
  -p, --partition=PARTITION  report on specific partition
  -r, --responding           report only responding nodes
  -R, --list-reasons         list reason nodes are down or drained
  -s, --summarize            report state summary only
```

```
[jharvard@b-sa01 ~]# sinfo -p gpu_test
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
gpu_test     up   12:00:00      5   comp holygpu7c[26201-26202,26303-26304,26306]
gpu_test     up   12:00:00      8    mix
holygpu7c[26105-26106,26203,26205-26206,26301-26302,26305]
gpu_test     up   12:00:00      1  alloc holygpu7c26204
```

https://slurm.schedmd.com/sinfo.html

Partition Monitoring - `showq` (Non-native)

```
[jharvard@holylogin08 ~]# showq -p itc_gpu -o

SUMMARY OF JOBS FOR QUEUE: <itc_gpu>

ACTIVE JOBS--------------------
JOBID     JOBNAME    USERNAME      STATE   CORE   GPU    REMAINING  STARTTIME
================================================================================
1205801   0129kastau hyerincho     Running 1      1       61:35:25  Wed Jan 29 11:54:16
1328292   0109oz128  hyerincho     Running 64     4       75:36:22  Thu Jan 30 09:55:13
1351732   0130rdepgm hyerincho     Running 1      1       87:20:52  Thu Jan 30 13:39:43
1356731   0130beta10 hyerincho     Running 1      1       88:08:45  Thu Jan 30 14:27:36
1451334   0131beta10 hyerincho     Running 1      1       106:31:28  Fri Jan 31 08:50:19

     5 active jobs :   68 of  256 cores ( 26.56 %):    8 of   16 gpus ( 50.00 %):    4 of    4
nodes (100.00 %)

WAITING JOBS------------------------
JOBID     JOBNAME    USERNAME      STATE   CORE   GPU     WCLIMIT  QUEUETIME
================================================================================
1571976   .fasrcood/ cfpark00      Waiting 64     4       72:00:00  Sat Feb  1 00:06:13

Total Jobs: 6     Active Jobs: 5     Idle Jobs: 1     Blocked Jobs: 0
```

Shows summary of all running, pending, and blocked jobs

Options:

- `-p`: partition
- `-o`: pending jobs by priority
- `-u`: display jobs for current user
- `-U`: jobs by username
- `-s`: only summary information

https://github.com/fasrc/slurm_showq

Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Job Efficiency

Plamen Krastev

Job Efficiency Summary - Memory: `seff`

`seff` - Provides statistics on how efficiently resources were utilized by a completed job.

```
[user@boslogin01 home]# seff 1234567
Job ID: 1234567
Cluster: [cluster name] User/Group: user/user_lab
State: COMPLETED (exit code 0)
Nodes: 8
Cores per node: 64
CPU Utilized: 37-06:17:33
CPU Efficiency: 23.94% of 155-16:02:08 core-walltime
Job Wall-clock time: 07:17:49
Memory Utilized: 1.53 TB (estimated maximum)
Memory Efficiency: 100.03% of 1.53 TB (195.31 GB/node)
```

- Know your Code
  - Numerical Methods
  - Size of Data
  - Type of Parallelism
- Experimentation
  - Validate Memory Size Requirements
  - Scaling Tests
  - Profiling
- Select Appropriate Partition and Hardware

https://docs.rc.fas.harvard.edu/kb/how-do-i-figure-out-how-efficient-my-job-is

Job Efficiency Summary - `seff-account`

```
[root@holy7c22501 ~]# seff-account -u user
------------------------------------------------------------
Job Information
Names: interactive, column_integration, stilt
Cluster: odyssey
Users: user
Account: user_lab
Start Time: now-1days
End Time: now
Average Requested CPUs: 7.89 cores on 1.00 node(s)
Average Requested Memory: 123.52G
Average Requested Time: 86248.08s
------------------------------------------------------------
Job Status
CANCELLED by 0: 3004
COMPLETED: 59
FAILED: 2237
PREEMPTED: 8
------------------------------------------------------------
------------------------------------------------------------
Finished Job Statistics
(excludes pending, running, and cancelled jobs)
Average CPU Efficiency 15.74%
Average Memory Usage 3.43G
Average Run-time 2386.95s
---------------------

CPU Efficiency (%)
--------------------
+0.00e+00 - +1.00e+01   [2619]  ████████████████████████████
+1.00e+01 - +2.00e+01   [   0]
+2.00e+01 - +3.00e+01   [   0]
+3.00e+01 - +4.00e+01   [  75]  ▉
+4.00e+01 - +5.00e+01   [ 982]  ██████████
+5.00e+01 - +6.00e+01   [ 131]  ▉
+6.00e+01 - +7.00e+01   [  53]  ▌
+7.00e+01 - +8.00e+01   [  10]  ▏
+8.00e+01 - +9.00e+01   [  10]  ▏
+9.00e+01 - +1.00e+02   [   3]
```

```
Memory Efficiency (%)
---------------------
+0.00e+00 - +1.00e+01   [3448]  ████████████████████████████
+1.00e+01 - +2.00e+01   [ 265]  ▉
+2.00e+01 - +3.00e+01   [   3]
+3.00e+01 - +4.00e+01   [   1]
+4.00e+01 - +5.00e+01   [   2]
+5.00e+01 - +6.00e+01   [   6]  ▏
+6.00e+01 - +7.00e+01   [  10]  ▏
+7.00e+01 - +8.00e+01   [   5]
+8.00e+01 - +9.00e+01   [   1]
+9.00e+01 - +1.00e+02   [  42]  ▌

Time Efficiency (%)
---------------------
+0.00e+00 - +1.00e+01   [3769]  ████████████████████████████
+1.00e+01 - +2.00e+01   [   4]
+2.00e+01 - +3.00e+01   [   5]
+3.00e+01 - +4.00e+01   [   0]
+4.00e+01 - +5.00e+01   [   0]
+5.00e+01 - +6.00e+01   [   0]
+6.00e+01 - +7.00e+01   [   4]
+7.00e+01 - +8.00e+01   [   1]
+8.00e+01 - +9.00e+01   [   0]
+9.00e+01 - +1.00e+02   [   0]
------------------------------------------------------------
```

Options

- `-u` user
- `-a` account (will not show data for jobs outside your user)
- `-S` start time
- `-E` end time

https://github.com/fasrc/seff-account

Job Efficiency - Memory

Memory per node or per CPU / task ?

- Memory per node `--mem`
- Memory per CPU / task `--mem-per-cpu`

How much memory should I request?

1. **Initial Guess:** 2-3 times the size of data (input data or generated/compute data). Or, if you don't know, request 4GB per core (most of the cluster has 4GB/core)
2. **Test:** Run a test job on the `test` partition with your guess
3. **If Out Of Memory:** Double memory and go to step 2
4. **If complete with no errors:** Inspect the result with `seff`, or `sacct` and note the actual used memory
5. **Adjust memory:** Update memory to match used memory plus 10% buffer (aim at memory efficiency of at least 80%)

Memory Usage - `sacct`

1. Run a test batch job
2. Check memory usage after the job has completed (with `sacct` or `seff`)

```
[jharvard@boslogin01 ~]$ sacct -j 7212727 -o ReqMem,MaxRSS
    ReqMem     MaxRSS
---------- ----------
       28G
                7860K
                 256K
            28036696K

[jharvard@boslogin01 ~]$ sacct -j 7212727 -o ReqMem,MaxRSS --units=G
    ReqMem     MaxRSS
---------- ----------
       28G
                0.01G
                0.00G
               26.74G
```

Memory Usage - `seff`

1. Run a test batch job
2. Check memory usage after the job has completed (with `sacct` or `seff`)

```
[jharvard@boslogin01 ~]$ seff 7212727
Job ID: 7212727
Cluster: odyssey
User/Group: user/user_lab
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:02:02
CPU Efficiency: 85.92% of 00:02:22 core-walltime
Job Wall-clock time: 00:01:11
Memory Utilized: 26.74 GB
Memory Efficiency: 95.49% of 28.00 GB (14.00 GB/core)
```

Hands-On Practice: Exercise Setup

Create space for the workshop materials, e.g.,

```
> mkdir /n/netscratch/<LAB_NAME>/Lab/<USER_NAME>/Workshop
```

Clone the `User_Codes` GitHub repository

```
> cd /n/netscratch/<LAB_NAME>/Lab/<USER_NAME>/Workshop/
> git clone https://github.com/fasrc/User_Codes.git
```

Go to the exercise directory

```
> cd User_Codes/Training/Advanced_Cluster_Usage/
```

NOTE

- Exercises are implemented in C, C++, Fortran, Python, and R
- Contents for each implementation are located in the corresponding folders in the root directory for each exercise

Hands-On Practice: Memory Efficiency per Node (1)

Exercise 1: Memory Efficiency per Node (`--mem`)

We use a C code, `mem_test.c`, to generate a random matrix of dimension 60,000. Using double precision, the program needs ~28.8 GB of memory (since, 60000 x 60000 x 8 bytes = 28,800,000,000 bytes) to execute successfully.

(1) Compile the code. On a login node execute:

```
> module load gcc/14.2.0-fasrc01
> make # we use a Makefile to compile the C code
```

(2) Create a batch-jobs submission script. Here we intentionally request less memory than what the job needs, e.g.,

```
#SBATCH --mem=20G # We request deliberately less memory
```

(3) Submit the job and check the job status

```
> sbatch run.sbatch
> sacct -j <JOB_ID>
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise1/C

Hands-On Practice: Memory Efficiency per Node (2)

Exercise 1: Memory Efficiency per Node (`--mem`)

NOTE: The job fails due to insufficient memory

```
sacct -j 3249906
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
3249906        mem_test       test   rc_admin          1 OUT_OF_ME+    0:125
3249906.bat+      batch              rc_admin          1 OUT_OF_ME+    0:125
3249906.ext+     extern              rc_admin          1  COMPLETED      0:0
```

(4) Adjust the memory request and resubmit the job

```
#SBATCH --mem=40G  # Double the original memory request
```

(5) Check the job status and memory efficiency

```
> sacct -j <JOB_ID> # Check job status (Job completes with no errors)
> seff <JOB_ID>     # Check efficiency (Memory Efficiency is about 60%)
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise1/C

Hands-On Practice: Memory Efficiency per Node (3)

Exercise 1: Memory Efficiency per Node (`--mem`)

NOTE: Memory Efficiency is 60%.

```
> seff 3253896
...
Memory Utilized: 23.83 GB
Memory Efficiency: 59.57% of 40.00 GB (40.00 GB/node)
```

(6) Adjust the memory request so that the efficiency is at least 80% and resubmit the job

```
#SBATCH --mem=28G
```

(7) Check memory efficiency

```
Memory Utilized: 23.51 GB
Memory Efficiency: 83.96% of 28.00 GB (28.00 GB/node)
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise1/C

Hands-On Practice: Memory Efficiency per CPU (1)

Exercise 2: Memory Efficiency per CPU/core (`--mem-per-cpu`)

We use a C code, `omp_mem_test.c`, to generate a random matrix of dimension 60,000. Using double precision, the program needs ~28.8 GB of memory (since, 60000 x 60000 x 8 bytes = 28,800,000,000 bytes) to execute successfully. We use OpenMP to parallelize the creation of the random matrix. The specific example uses 2 OMP threads. The purpose of this exercise is to illustrate requesting the memory via the `--mem-per-cpu` option.

(1) Compile the code. On a login node execute:

```
> module load gcc/14.2.0-fasrc01
> make              # We use a Makefile to compile the code
```

(2) Create a batch-jobs submission script. Here we intentionally request less memory than what the job needs, e.g.,

```
#SBATCH --mem-per-cpu=10G # We request deliberately less memory
```

(3) Submit the job and check the job status

```
> sbatch run.sbatch
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise2/C

Hands-On Practice: Memory Efficiency per CPU (2)

Exercise 2: Memory Efficiency per CPU/core (`--mem-per-cpu`)

NOTE: The job fails due to insufficient memory

```
> sacct -j 6959634
JobID           JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
6959634      omp_mem_t+       test   rc_admin          2     FAILED      1:0
6959634.bat+      batch              rc_admin          2     FAILED      1:0
6959634.ext+     extern              rc_admin          2  COMPLETED      0:0
6959634.0    omp_mem_t+              rc_admin          2 OUT_OF_ME+    0:125
```

(4) Adjust the memory request and resubmit the job

```
#SBATCH --mem-per-cpu=20G   # Double the original memory request
```

(5) Check the job status and memory efficiency

```
> sacct -j <JOB_ID>  # Check job status (Job completes with no errors)
> seff <JOB_ID>      # Check efficiency
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise2/C

Hands-On Practice: Memory Efficiency per CPU (3)

Exercise 2: Memory Efficiency per CPU/core (`--mem-per-cpu`)

Check memory efficiency, e.g.,

```
> seff 6960135
…
Memory Utilized: 25.27 GB
Memory Efficiency: 63.17% of 40.00 GB (20.00 GB/core)
…
```

We need to adjust the memory further so that the memory efficiency is at least 80%, resubmit the job, and re-check the job status and efficiency

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise2/C

Hands-On Practice: Memory Efficiency per CPU (4)

Exercise 2: Memory Efficiency per CPU/core (`--mem-per-cpu`)

Adjust the memory and resubmit the job, e.g.,

```
#SBATCH --mem-per-cpu=14G
```

Check the job status and efficiency, e.g.,

```
> seff 6960985
…
Memory Utilized: 25.60 GB
Memory Efficiency: 91.42% of 28.00 GB (14.00 GB/core)
…
```

Memory efficiency is above 80% (~91%).

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise2/C

Monitoring Memory Usage in Real-Time (1)

You can monitor memory usage in real time by starting an interactive session in one terminal, `ssh` to the host where the interactive job started from another terminal, and then use the `top` or `htop` command

In terminal 1 you do:

```
# Start an interactive job
[username@LOGIN_NODE ~]$ salloc -p test --mem=40G -t 0-06:00

# Land on a compute node
[username@holy8a26602 ~]$ cd /n/netscratch/<LAB_NAME>/Lab/<USER_NAME>/Workshop/
[username@holy8a26602 ~]$ cd User_Codes/Training/Advanced_Cluster_Usage/Exercise1/R

# Load software modules and run calculations, e.g.,
[username@holy8a26602 ~]$ module load R/4.4.1-fasrc01
[username@holy8a26602 ~]$ Rscript mem_test.R
```

Monitoring Memory Usage in Real-Time (2)

You can monitor memory usage in real time by starting an interactive session in one terminal, `ssh` to the host where the interactive job started from another terminal, and then use the `top` or `htop` command

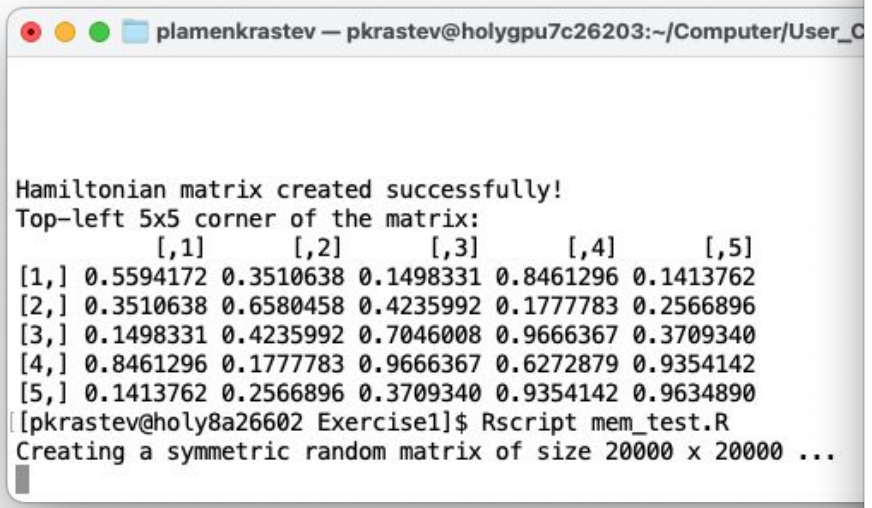
Then, in terminal 2 you do:

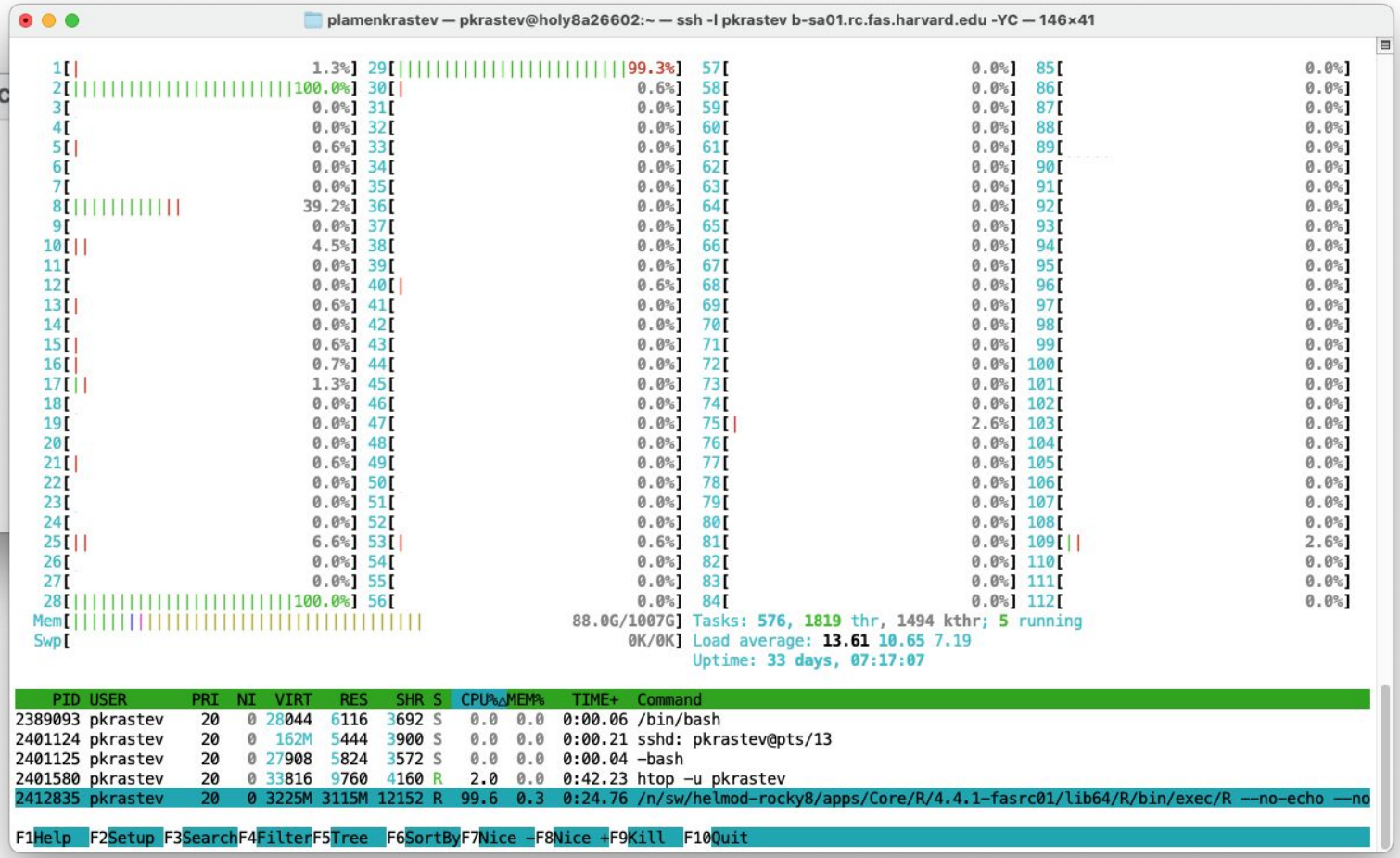
```
# SSH to the compute node where the interactive job started, e.g.,
[username@LOGIN_NODE ~]$ ssh holy8a26602

# Land on the compute node
[username@holy8a26602 ~]$

# Use top or htop to monitor memory usage in real-time, e.g.,
[username@holy8a26602 ~]$ htop -u username # Monitor only the processes you own
```

Monitoring Memory Usage in Real-Time (3)





Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Break

```
# Set up your conda environment for scaling Exercise 3
# On a login node execute:
> cd /n/netscratch/<LAB_NAME>/Lab/<USER_NAME>/Workshop/
> cd User_Codes/Training/Advanced_Cluster_Usage/Exercise3/
> sbatch run_speedup.sbatch

# This will create a conda environment named "speedup_env"
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise3

Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Job Efficiency - Scaling (Cores)

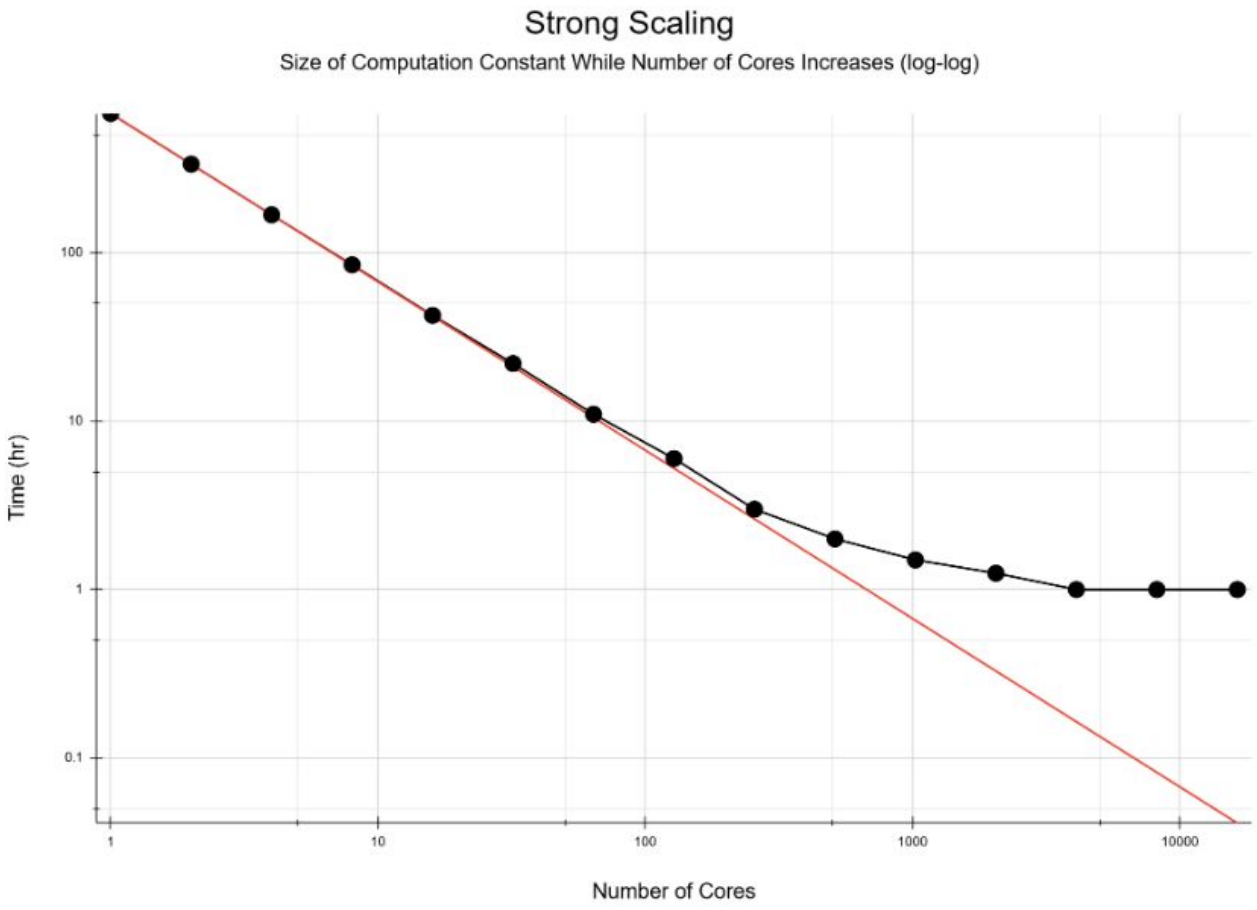
Note

- SLURM does not automatically parallelize your code. I.e., if it is serial, no matter how many more cores you request, it will not run faster
- Check code's documentation, or with the primary author, if the code is parallel or not

How many cores (and nodes) should I request ?

1. Determine if the code is thread-based (e.g., OpenMP), rank-based (e.g., MPI) or hybrid (e.g., OpenMP+MPI)
2. Test with a small-scale example on the test partition
3. Perform scaling tests

Strong Scaling



- Compute bound
- Keep problem size the same, increase parallel process count
- At some point, communication overhead surpasses computation time

Strong Scaling

How much faster will the program run?

Speedup:

$$S(n) = \frac{T(1)}{T(n)}$$

Time to complete the job on **one** process

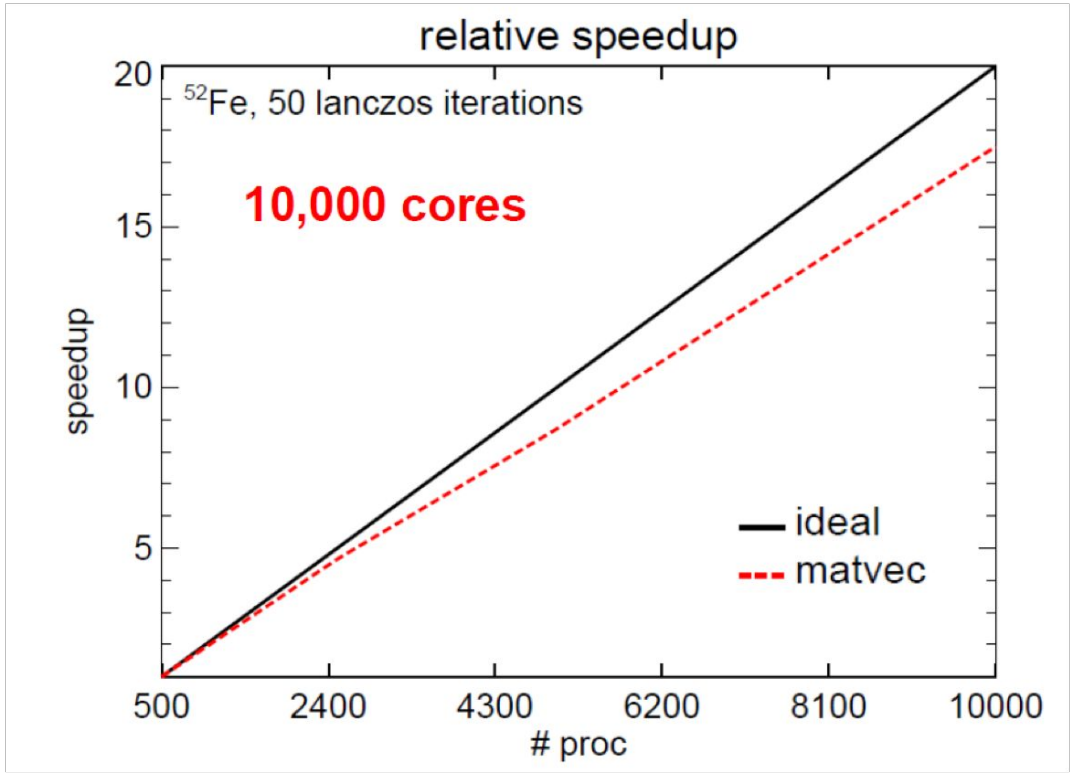
Time to complete the job on **n** process

Efficiency:

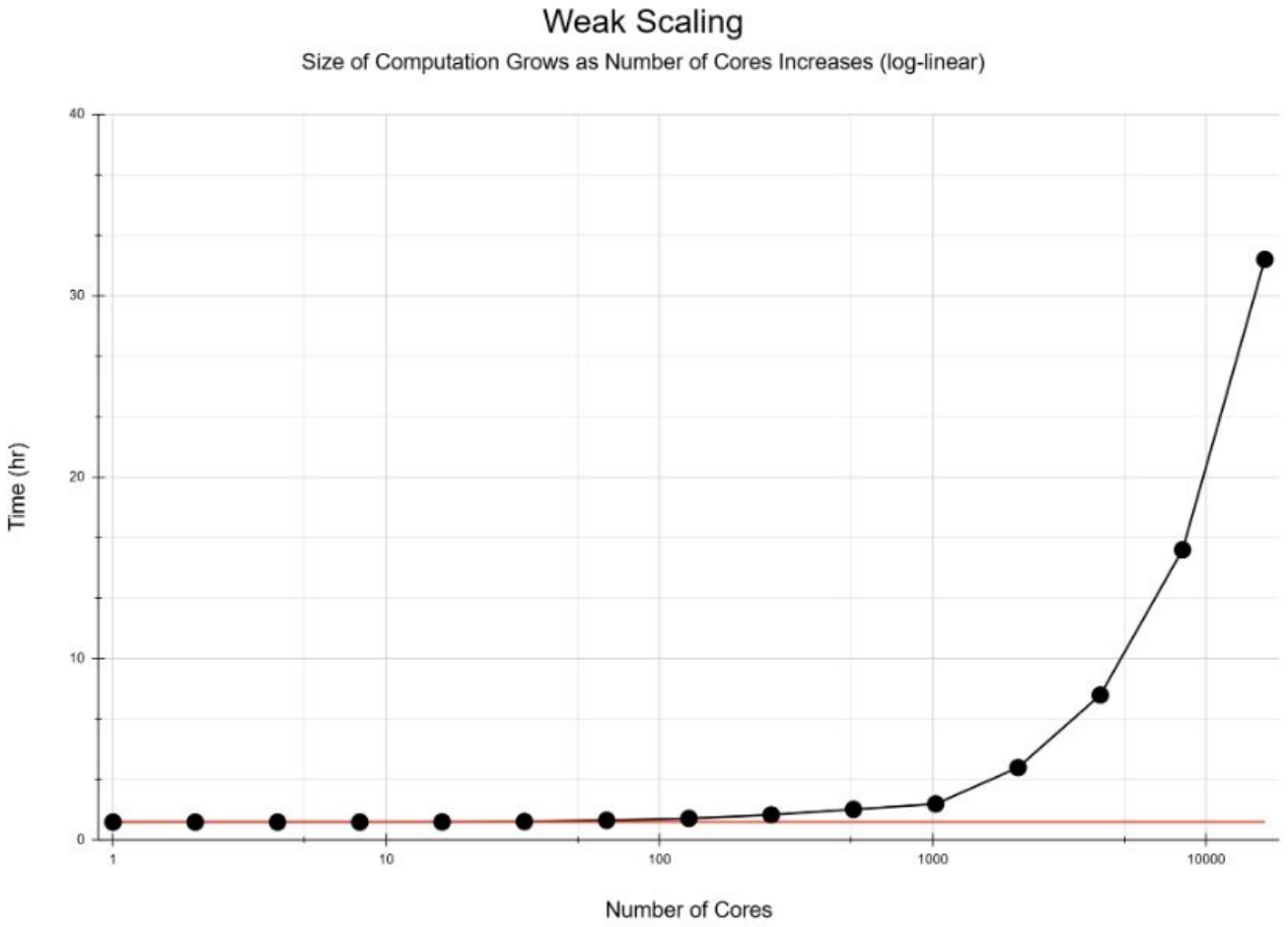
$$E(n) = \frac{S(n)}{n}$$

Tells you how efficiently you parallelize your code

Realistic example: Speedup of matrix vector multiplication in large scale shell-model calculations



Weak Scaling



- Memory bound
- Increase problem size proportionally with resources
- At some point, communication overhead surpasses computation time

Hands-On Practice: Scaling - OpenMP (1)

Exercise 3: Scaling of threaded parallel applications (OpenMP)

This example illustrates evaluating the speedup of parallel applications. The specific example is an OpenMP (OMP) implementation of a Monte-Carlo algorithm for calculating π in parallel. We will run the program on 1, 2, 4, 8, 16, 32, and 64 OMP threads, calculate the speedup and create a speedup figure.

(1) Compile the code. On a login node execute:

```
> module load gcc/14.2.0-fasrc01
> make                    # We use a Makefile to build the code
```

(2) Submit the job with different number of OpenMP threads and record the execution times

```
#SBATCH -c 1               # Here we will vary c from 1 to 64 (1, 2, 4, 8, 16, 32, 64)
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise3/C

Hands-On Practice: Scaling - OpenMP (2)

Exercise 3: Scaling of threaded parallel applications (OpenMP)

- Upon execution of the code, timing results are recorded in the file `omp_pi.dat`
- For each OpenMP thread number, collect the execution times in a text file, `scaling_results.txt`, used to generate the speedup figure, e.g.,

```
> cat scaling_results.txt
 1 10.92
 2  5.47
 4  2.74
 8  1.37
16  0.68
32  0.34
64  0.17
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise3/C

Hands-On Practice: Scaling - OpenMP (3)

Exercise 3: Scaling of threaded parallel applications (OpenMP)

(3) Generate speedup figure

- Use the Python code , `speedup.py`, to compute the speedup and parallel efficiency, and generate the speedup figure
- You can use the batch-job submission script `run_speedup.sbatch` to send the figure-generation job to the queue

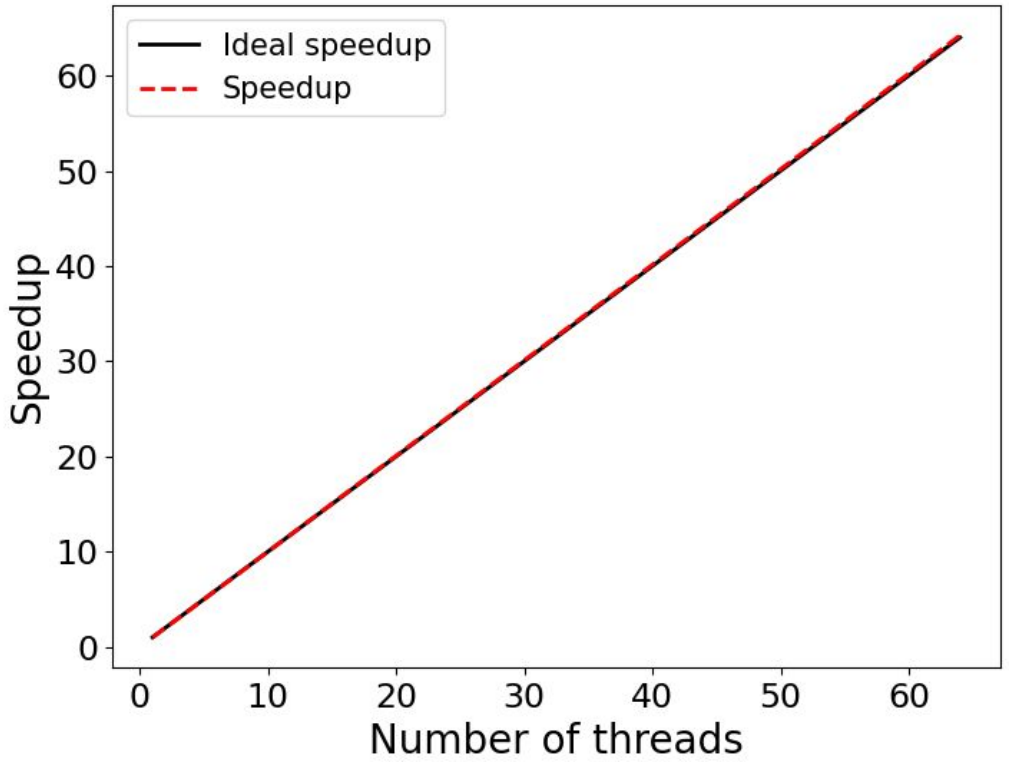
NOTE: This assumes that you already have the `conda` environment `speedup_env` set up

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise3/C

Hands-On Practice: Scaling - OpenMP (4)

Speedup and Parallel Efficiency

Nthreads	Walltime	Speedup	Efficiency (%)
1	10.92	1.00	100.00
2	5.47	2.00	99.82
4	2.74	3.99	99.64
8	1.37	7.97	99.64
16	0.68	16.06	100.00
32	0.34	32.12	100.00
64	0.17	64.24	100.00



https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise3/C

Schedule

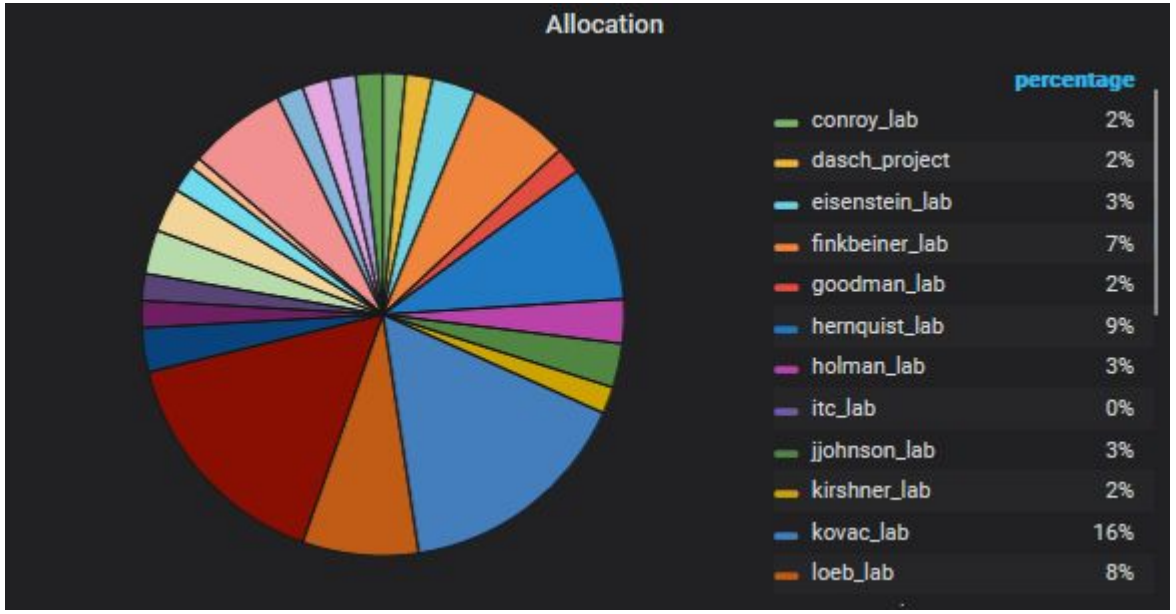
- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

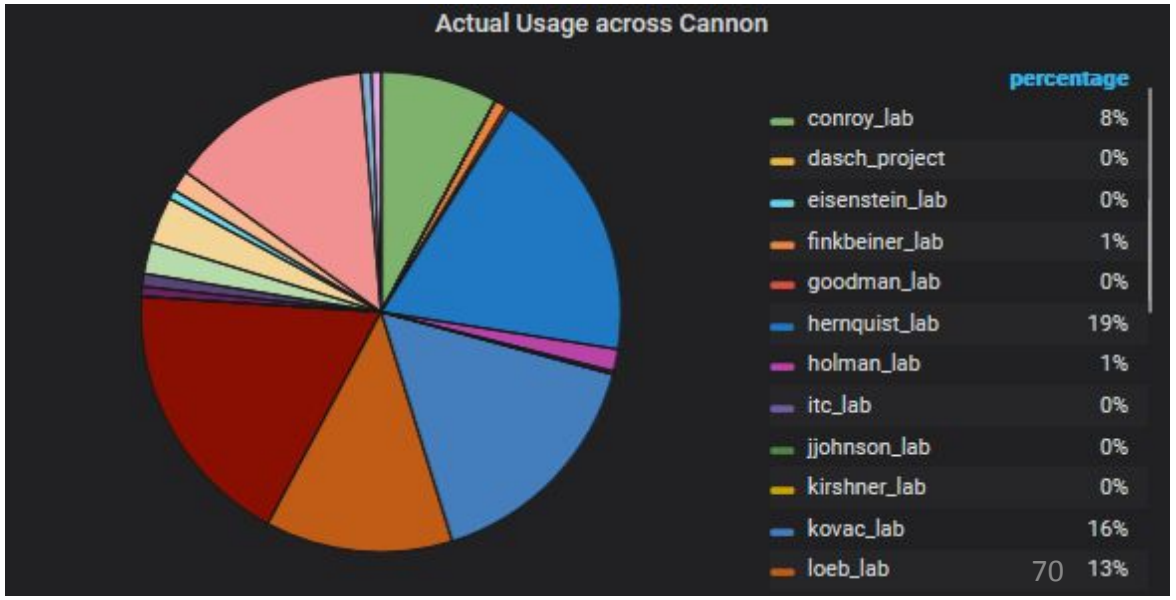
Fairshare

Paul Edmon

Fairshare

- A method for ensuring the equitable use of a cluster
- The fraction of the cluster a user/group gets
- The score assigned by Slurm to a user/group based on usage
- Priority that users/groups get based on usage





Trackable RESources (TRES)

- Method of weighting different types of resources
- CPU/GPU: Based on theoretical peak FLOps (Floating Point Operations per second)
- Memory: NumCore*CoreTRES/TotalMem
- Usage = Runtime * (CoreTRES*CoreAlloc + MemTRES*MemAlloc + GPUTRES*GPUAlloc)
- requeue partitions half cost
- scontrol show partition <partitionname>
  - TRESBillingWeights

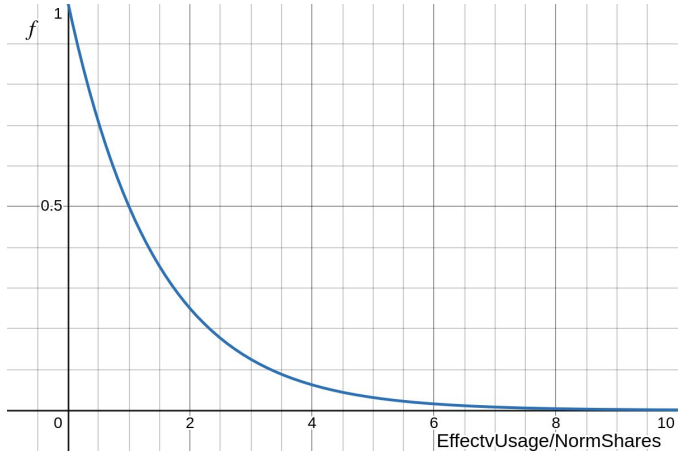
Processor Type	TRES
Intel Skylake	0.5
AMD Milan	0.5
AMD Genoa	0.6
Intel Sapphire Rapids	0.6
Intel Cascade Lake	1.0
Intel Ice Lake	1.15
Nvidia A40	10
Nvidia V100	75
Nvidia A100	209.1
Nvidia H100	546.9

Shares

- Gratis Shares
  - Cannon: 200
  - FASSE: 100
- PI Purchased
  - NumCores*CoreTRES + NumGPUS*GPUTRES
- Group Purchased
- RawShares = Gratis + PI + Group

Processor Type	TRES
Intel Skylake	0.5
AMD Milan	0.5
AMD Genoa	0.6
Intel Sapphire Rapids	0.6
Intel Cascade Lake	1.0
Intel Ice Lake	1.15
Nvidia A40	10
Nvidia V100	75
Nvidia A100	209.1
Nvidia H100	546.9

Fairshare - `sshare`




```
[user1@holyitc01 ~]$ sshare --account=jharvard_lab -a

Account  User  RawShares NormShares RawUsage  EffectvUsage FairShare
----------------------------------------------------------------------
jharvard_lab          244    0.001363   45566082       0.000572   0.747627
jharvard_lab user1 parent     0.001363   8202875  0.000572   0.747627
jharvard_lab user2 parent     0.001363   248820   0.000572   0.747627
jharvard_lab user3 parent     0.001363   163318   0.000572   0.747627
jharvard_lab user4 parent     0.001363   18901027 0.000572   0.747627
jharvard_lab user5 parent     0.001363   18050039 0.000572   0.747627
```

Fairshare Equation:
f = 2^(-EffectvUsage/NormShares)

Fairshare Regimes:

- f =1 : Unused
- 1.0 > f > 0.5: Underutilized
- 0.5: Average utilization
- 0.5> f > 0: Over-utilized
- f = 0: No share left

Halflife: 3 days

Priority

- Priority = Fairshare*FairshareWeight + JobAge*JobAgeWeight
- FairshareWeight = 20,000,000
- JobAgeWeight = 10,000,000
  - Grows to maximum value over 3 days
  - Weighted so that maximum Job Age Priority is equivalent to Fairshare 0.5
- sprio

Fairshare - `scalc`: Projecting Fairshare

```
[jharvard@holy8a24507 general]# scalc
What do you want to calculate?
1) Projected FairShare Based on New RawShare
2) Additional RawShare Need for FairShare Score
3) Projected Time to Reach FairShare Score Assuming No New Jobs
4) Projected Usage and Fairshare Based on Job
5) Calculate New RawShare Based on Additional Hardware
Option: 1
1
We will now calculate the projected fairshare score based on the current usage of the Account and RawShare you specify.
First we need to know what account you want to calculate for: jharvard_lab
Now we need to know the new RawShare: 400
jharvard_lab has a current RawShare of 200 and Fairshare of 0.352874
With a new RawShare of 400 jharvard_lab would have a Fairshare of 0.5943373717651835
```

Fairshare - `scalc`: Projecting RawShares

```
[jharvard@holy8a24507 general]# scalc
What do you want to calculate?
1) Projected FairShare Based on New RawShare
2) Additional RawShare Need for FairShare Score
3) Projected Time to Reach FairShare Score Assuming No New Jobs
4) Projected Usage and Fairshare Based on Job
5) Calculate New RawShare Based on Additional Hardware
Option: 2
2
We will now calculate the Rawshares required to reach a specific Fairshare based on the current usage of an Account you
specify.
First we need to know what account you want to calculate for: jharvard_lab
Now we need to know the target Fairshare: 0.5
jharvard_lab has a current RawShare of 200 and Fairshare of 0.352787
For a fairshare of 0.5 jharvard_lab will need a RawShare of 301
```

Fairshare - `scalc`: Hardware RawShare Projection

```
[jharvard@holy8a24507 general]# scalc
What do you want to calculate?
1) Projected FairShare Based on New RawShare
2) Additional RawShare Need for FairShare Score
3) Projected Time to Reach FairShare Score Assuming No New Jobs
4) Projected Usage and Fairshare Based on Job
5) Calculate New RawShare Based on Additional Hardware
Option: 5
5
We will now calculate the new RawShare for a specified account based on the additional hardware specified. Available
hardware types are Intel Sapphire Rapids cores and Nvidia H100 GPUs
First we need to know what account you want to calculate for: jharvard_lab
How many Intel Sapphire Rapids cores (TRES Weight = 0.6) do you want to add: 500
How many Nvidia H100 GPUs (TRES Weight = 546.9) do you want to add: 20
Account jharvard_lab has current RawShares of 200
The additional hardware would add 11238 RawShares for a total of 11438
```

Fairshare - `scalc`: Halflife

```
[jharvard@holy8a24507 general]# scalc
What do you want to calculate?
1) Projected FairShare Based on New RawShare
2) Additional RawShare Need for FairShare Score
3) Projected Time to Reach FairShare Score Assuming No New Jobs
4) Projected Usage and Fairshare Based on Job
5) Calculate New RawShare Based on Additional Hardware
Option: 3
3
We will now calculate how long it will take for the specified account to reach the specified fairshare if no new jobs are
submitted.
First we need to know what account you want to calculate for: jharvard_lab
Now we need to know the target Fairshare: 0.75
jharvard_lab has a current Normalized Usage of 0.000582 a Normalized Allocation of 0.000387 and Fairshare of 0.352615
jharvard_lab will need 5.572115968734966 days to reach a fairshare of 0.75 if they submit no further jobs
```

Fairshare - `scalc`: Projected Usage

```
[jharvard@holy7c22501 ~]# scalc
What do you want to calculate?
1) Projected FairShare Based on New RawShare
2) Additional RawShare Need for FairShare Score
3) Projected Time to Reach FairShare Score Assuming No New Jobs
4) Projected Usage and Fairshare Based on Job
5) Calculate New RawShare Based on Additional Hardware
Option: 4
4
We will now calculate how much TRES your jobs will cost as well as how it will impact the specified account's usage and
fairshare.
First we need to know what account you want to calculate for: jharvard_lab
Next we need the partition you want to submit to: shared
How many cores will you use per job: 1024
How much memory in GB will you use per job: 4000
How many total GPUs will you user per job: 0
How long will the job run for (DD-HH:MM:SS): 1-00:00:00
How many jobs (or array elements) will you run of this type: 1
jharvard_lab has a current Raw Usage of 9725230 a Normalized Usage of 0.000026 a Normalized Allocation of 0.000759 and
Fairshare of 0.976085
This partition has a TRES charge per second of CPU: 1.0 | Mem (per GB): 0.25 | GPU (per GPU): 0
This set of jobs has a total TRES usage of: 174873600.0
For jharvard_lab this will give a new Normalized Usage of 0.0004935173337802807 and a Fairshare of 0.6371829348127839
```

Fairshare Graphs Demo

Get on RC VPN

Go To: dash.rc.fas.harvard.edu

Select either:

- Cannon Lab Fairshare
- FASSE Lab Fairshare

Schedule

- 12:00-12:30p: Lunch & set up
- 12:30-1:00p: Introduction to Advanced Cluster Usage (Paula Sanematsu)
- 1:00-1:30p: Job Resource Requirements (Paul Edmon)
- 1:30-1:45p: Job/Partition/Queue Monitoring (Manasvita Joshi)
- 1:45-2:30p: Job Efficiency - Memory (Plamen Krastev)
- 2:30-2:45p: Break
- 2:45-3:15p: Job Efficiency - Scaling (Plamen Krastev)
- 3:15-3:45p: Fairshare (Paul Edmon)
- 3:45-4:00p: Survey, General Q&A

Training Survey

Please, fill out our course survey. Your feedback is essential for us to improve our trainings!

https://tinyurl.com/FASRCworkshop

Questions, Comments, Concerns?

Extra Material

Hands-On Practice: Scaling - MPI (1)

Exercise 4: Scaling of distributed parallel applications (MPI)

This example illustrates evaluating the speedup of parallel applications. The specific example is a MPI implementation of a Monte-Carlo algorithm for calculating PI in parallel. We will run the program with 1, 2, 4, 8, 16, 32, and 64 MPI tasks, and evaluate the speedup and scaling of the application with the provided Python code.

(1) Compile the code

```
> module load intel/24.2.1-fasrc01 openmpi/5.0.5-fasrc01
> make                    # We use a Makefile to build the code
```

(2) Submit the job with different number of MPI ranks and record the execution times

```
#SBATCH -n 1            # Here we will vary n from 1 to 64 (1, 2, 4, 8, 16, 32, 64)
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise4

Hands-On Practice: Scaling - MPI (2)

Exercise 4: Scaling of distributed parallel applications (MPI)

For each MPI task number collect the execution times in a text file, `scaling_results.txt`, used to generate the speedup figure, e.g.,

```
> cat scaling_results.txt
 1   74.10
 2   37.09
 4   18.55
 8   9.41
16   4.65
32   3.38
64   1.90
```

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise4

Hands-On Practice: Scaling - MPI (3)

Exercise 4: Scaling of distributed parallel applications (MPI)

(3) Generate speedup figure

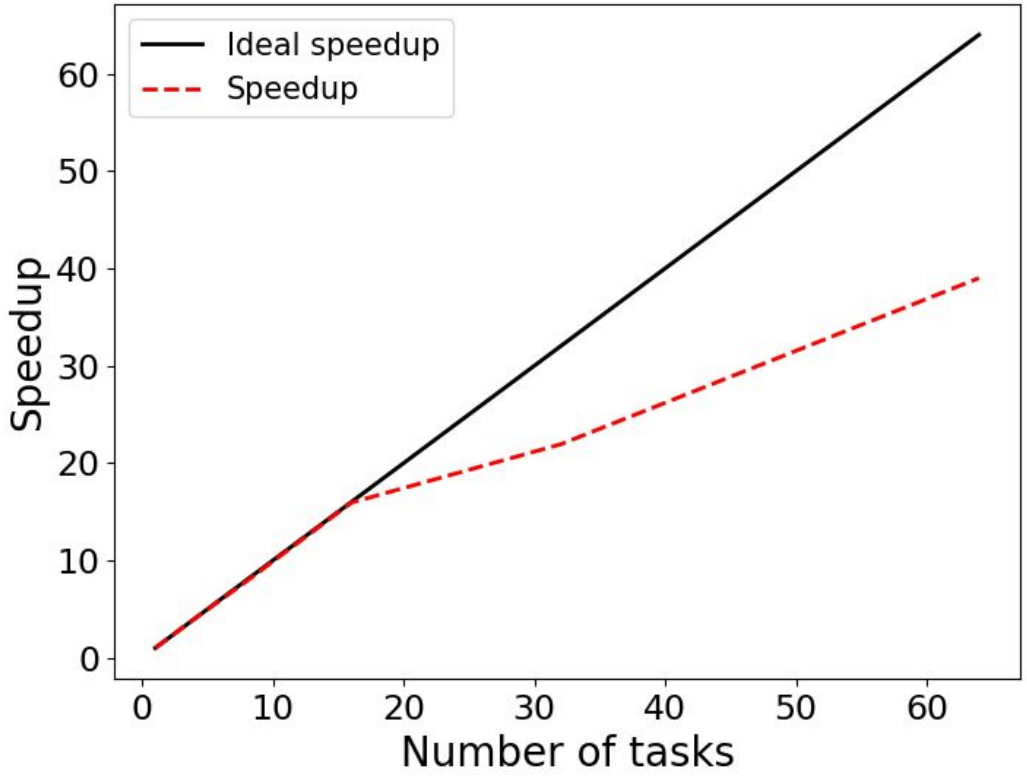
- Use the Python code , `speedup_mpi.py`, to compute the speedup and parallel efficiency, and generate the speedup figure
- You can use the batch-job submission script `run_speedup.sbatch` to send the figure-generation job to the queue

https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise4

Hands-On Practice: Scaling - MPI (4)

Speedup and Parallel Efficiency

Ntasks	Walltime	Speedup	Efficiency (%)
1	74.10	1.00	100.00
2	37.09	2.00	99.89
4	18.55	3.99	99.87
8	9.41	7.87	98.43
16	4.65	15.94	99.60
32	3.38	21.92	68.51
64	1.90	39.00	60.94



https://github.com/fasrc/User_Codes/tree/master/Training/Advanced_Cluster_Usage/Exercise4

Improve Speedup and Scaling of Distributed (MPI) Apps

- Make use of SLURM options

```
--nodes/-N
--ntasks-per-node
--contiguous
--distribution/-m
   block
   cyclic
   plane
```

- Reduce time spent in communication

- Improve load balance (give equal amount of work to each parallel process)

Improve Speedup and Scaling of Distributed (MPI) Apps

Make use of SLURM options, e.g., have all MPI tasks on a single node:

```
#SBATCH --nodes=1
#SBATCH --ntasks=32
```

Exercise 4: Scaling of distributed parallel applications (MPI)

Ntasks	Walltime	Speedup	Efficiency (%)
1	74.05	1.00	100.00
2	37.16	1.99	99.64
4	18.59	3.98	99.58
8	9.41	7.87	98.37
16	4.65	15.92	99.53
32	2.40	30.85	96.42
64	1.35	54.85	85.71