

Slurm Basics

June 3, 2025

Timothy H. Kaiser, Ph.D.

tkaiser2@nrel.gov

<https://github.com/timkphd/examples/blob/master/workshop/2025/slurm/slurm25.pdf>

Setup On Kestrel..

```
ssh kestrel.nrel.gov
```

```
cd /scratch/$USER
```

```
mkdir slurm25
```

```
cd slurm25
```

```
tar -xzf /scratch/tkaiser2/shared/slurm25/slurm25.tgz
```

```
export SLURM_ACCOUNT=YOUR_ACCOUNT
```

```
export SALLOC_ACCOUNT=$SLURM_ACCOUNT
```

```
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
```

```
./doall
```

To get the results without running...

```
tar -xzf /scratch/tkaiser2/shared/slurm25/run25.tgz
```


Introduction to Running on High Performance Computing Platforms

slurm

Slurm is the software used to manage user interactions on NREL's HPC platforms. Slurm allows users access to resources (compute nodes) for some duration of time so they can perform work.

This talk will cover the basics of slurm. After this session users should be comfortable running compute jobs on HPC platforms, including NREL's Kestrel and Swift machines.

We will give an overview of slurm. We'll discuss commands for submitting jobs, getting status, deleting jobs, getting node and job information and job history. We'll show some tools to simplify interactions with slurm. We'll discuss priority and partitions. A number of sample run scripts will be given. We'll discuss running interactively. New this year, we'll discuss array jobs and running in shared partitions.

To cover...

- Slurm, what is it, does it do?
- Basic commands for:
 - Submitting
 - Getting status
 - Killing jobs
 - Node information
 - History of jobs
 - Seeing priority
 - Partitions
- Simple run scripts
- Shared partition
- My "shortcuts"
- Array jobs
- Running interactively
- Some important options
- Links
- More scripts as time allows

Also: Start to develop some good habits

Unfortunately there are some chicken and egg issues here

How are HPC platforms laid out?

- Login node ←
- Compute nodes ←
- Service nodes ←
 - File system
 - Monitoring
 - Management
- All connected together via a network
- Kestrel: <https://www.nrel.gov/hpc/kestrel-system-configuration.html>

Kestrel

- Login nodes:
 - kestrel.hpc.nrel.gov (CPU)
 - kl1.hpc.nrel.gov (CPU)
 - kl2.hpc.nrel.gov (CPU)
 - kl3.hpc.nrel.gov (CPU)
 - kestrel-gpu.hpc.nrel.gov
 - kl5.hpc.nrel.gov (GPU)
 - kl6.hpc.nrel.gov (GPU)
- Compute nodes:
 - 2,314 with 104 Intel CPUs
 - 134 nodes with AMD processors and 4 H100 GPUs

On small systems...



- Login nodes might also be the file system and management node
- My home machine (Runs slurm, MPI, OpenMP, Julia, Fortran, C, Spack, Web Server...)
 - pie
 - Login
 - Service (slurm)
 - File Server (nfs)
 - 4 Compute nodes
 - pi00-pi03
 - 4 cores / 4 - 8GB
 - Switch

Slurm, what is it, does it do?

- Simple **L**inux **U**tility for **R**esource **M**anagement
- It manages compute resources
- It allows for fair sharing of resources
- It is the software that allows you, and others to run jobs on compute nodes

<https://slurm.schedmd.com/quickstart.html>

Some definitions

- Node:
 - What is normally thought of as a computer that you could, in theory, login. Slurm manages the **compute nodes** and assigns jobs to them.
- Partition:
 - A collection of compute nodes often that have similar characteristics (memory, disk, core, gpus) or just grouped for management purposes.
 - Every compute node is part of one or more partitions.

Slurm, what is it, does it do?

- Typical usage:
 - You create a "batch" script that says:
 - What you want to run
 - Resources required
 - How long
 - Will most often have a "srun" command to launch multiple instances of the same program in parallel
 - You tell slurm to "run" your script
 - Slurm will run your job on compute nodes (after some waiting for resource to become available)

Running Batch Scripts

- A batch script is submitted to **slurm**
 - Command to submit scripts
 - **sbatch myscript**
 - Options you add to the batch line overrule what you have in the script
 - **sbatch --nodes=1 myscript**
 - Scripts can use variables defined in your environment.
- The scheduler decides where and when to run your job
 - Each job is given a number
 - Jobs are also given a name so that you can track them in the system
 - Each job has a priority that goes up over time
 - Wait for nodes to become available
 - Start the job and let it run until it finishes or runs out of time

A Simple Slurm Script hostname

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

```
#SBATCH --job-name="atest"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o stdout
#SBATCH -e stderr
#SBATCH --export=ALL
##SBATCH --account=hpcapps
##SBATCH --partition=debug
##SBATCH --mail-type=ALL
##SBATCH --mail-user=joeuser@nrel.gov
```

```
#-----
```

```
cd /scratch/$USER
```

```
srun hostname > hostlist
```

Go to your scratch directory

srun - run a command in parallel

Scripts contain comments
designated with a # that are
interpreted by SLURM.
and normal shell commands
Lines with ## are ignored by
slurm

A Simple Slurm Script for a “parallel” job

<code>#!/bin/bash</code>	This is a bash script
<code>#SBATCH --job-name="atest"</code>	Give our job a name in the scheduler
<code>#SBATCH --nodes=2</code>	We want 2 nodes
<code>#SBATCH --ntasks-per-node=8</code>	We expect to run 8 tasks/node
<code>#SBATCH --time=00:02:00</code>	We want the node for 2 minutes
<code>#SBATCH --output=stdout</code>	Output will go to a file “stdout”
<code>#SBATCH --error=stderr</code>	Errors will go to a file “stderr”
<code>#SBATCH --export=ALL</code>	Pass current environment to nodes
<code>##SBATCH --mail-type=ALL</code>	Send email on abort,begin,end
<code>##SBATCH --mail-user=auser@nrel.gov</code>	Address for email
<code>#SBATCH --account=hpcapps</code>	Your account (not hpcapps)
<code>#SBATCH --partition=debug</code>	The scheduler partition
<code>#-----</code>	Just a normal “comment”
<code>cd /scratch/\$USER</code>	Go to this directory first
<code>srun hostname</code>	Run hostname on 8 cores / node (2 nodes)

Submitting and Output

```
[tkaiser2@kl1 system]$ sbatch slurm0
```

```
...
```

```
...
```

```
[tkaiser2@kl1 slurm25]$cat hostlist
```

```
x1005c2s4b0n1
```

```
x1005c2s4b1n0
```

```
x1005c2s4b0n1
```

```
x1005c2s4b0n1
```

```
x1005c2s4b0n1
```

```
x1005c2s4b0n1
```

```
x1005c2s4b0n1
```

```
x1005c2s4b0n1
```

```
x1005c2s4b0n1
```

```
x1005c2s4b1n0
```

```
x1005c2s4b1n0
```

```
x1005c2s4b1n0
```

```
x1005c2s4b1n0
```

```
x1005c2s4b1n0
```

```
x1005c2s4b1n0
```

```
x1005c2s4b1n0
```

```
[tkaiser2@kl1 slurm25]$
```

Color added for clarity

Time format

#SBATCH --time=dd-hh:mm:ss

Days (followed by -)

Hours (followed by :)

Minutes (followed by :)

Seconds

All are optional except Seconds

These are "legal"

--time=48:00:00

--time=120:00

--time=47:59:59

--time=2-00:00:00

--time=10:00

- Time is required for sbatch and salloc
- Can be useful for srun

Mail

--mail-type=

NONE

BEGIN

END

FAIL

ALL

TIME_LIMIT_90

--mail-user=**A_VALID_EMAIL_ADDRESS_PLEASE**

Standard Output/Error

- If you don't specify
 - `#SBATCH -o stdout`
 - `#SBATCH -e stderr`
- Output goes to:
 - `slurm-xxx.out/err` where `xxx` is the job number.
- You can create unique files for each job by including the following in your "file name":

<code>%j</code>	jobid of the running job.
<code>%u</code>	User name.
<code>%x</code>	Job name.

`#SBATCH -o X_%j_%u_%x.out` gives us `X_9366638_tkaiser2_atest.out`

- You can specify an absolute path for your output

Slurm defines Variables:

```
SLURM_MPI_TYPE=pmi2
SLURM_STEP_ID=0
SLURM_NODEID=0
SLURM_TASK_PID=29272
SLURM_PRIO_PROCESS=0
SLURM_CPU_BIND_VERBOSE=quiet
SLURM_SUBMIT_DIR=/scratch/tkaiser2
SLURM_CPUS_PER_TASK=4
SLURM_STEPID=0
SLURM_SRUN_COMM_HOST=192.168.1.1
SLURM_PROCID=0
SLURM_JOB_GID=131364
SLURM_CPU_BIND=....
SLURM_ACCOUNT=hpcapps
SLURMD_NODENAME=c1-28
SLURM_TASKS_PER_NODE=32
SLURM_NNODES=1
SLURM_LAUNCH_NODE_IPADDR=192.168.1.1
SLURM_STEP_TASKS_PER_NODE=32
SLURM_JOB_NODELIST=c1-28
SLURM_CLUSTER_NAME=swift
SLURM_NODELIST=c1-28
SLURM_NTASKS=32
SLURM_UMASK=0002
SLURM_JOB_CPUS_PER_NODE=128
SLURM_TOPOLOGY_ADDR=c1-28

SLURM_WORKING_CLUSTER=swift:192.168.1.4:6817:9472:101
SLURM_STEP_NODELIST=c1-28
SLURM_JOB_NAME=atest
SLURM_SRUN_COMM_PORT=34097
SLURM_JOBID=143033
SLURM_CONF=/etc/slurm.conf
SLURM_JOB_QOS=normal
SLURM_TOPOLOGY_ADDR_PATTERN=node
SLURM_CPUS_ON_NODE=128
SLURM_JOB_NUM_NODES=1
SLURM_JOB_UID=131364
SLURM_JOB_PARTITION=debug
SLURM_PTY_WIN_ROW=60
SLURM_CPU_BIND_LIST=....
SLURM_JOB_USER=tkaiser2
SLURM_PTY_WIN_COL=146
SLURM_NPROCS=32
SLURM_SUBMIT_HOST=swift-login-1.swift.hpc.nrel.gov
SLURM_JOB_ACCOUNT=hpcapps
SLURM_STEP_LAUNCHER_PORT=34097
SLURM_PTY_PORT=37177
SLURM_JOB_ID=143033
SLURM_CPU_BIND_TYPE=mask_cpu:
SLURM_STEP_NUM_TASKS=32
SLURM_STEP_NUM_NODES=1
SLURM_LOCALID=0
```

```
mkdir -p $SLURM_JOB_NAME/$SLURM_JOB_ID
cd $SLURM_JOB_NAME/$SLURM_JOB_ID
cat $0 > script.$SLURM_JOB_ID
```


Using variables defined in your environment

Script with variables

```
[tkaiser2@kl1 slurm0]$cat docommand
#!/bin/bash
#SBATCH --job-name="atest"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:01:00

srun $COMMAND > $OUTFILE
```

Set variables ; submit script

```
[tkaiser2@kl1 slurm0]$export COMMAND='date +%s.%N'
[tkaiser2@kl1 slurm0]$export OUTFILE=spread

[tkaiser2@kl1 slurm0]$sbatch --partition=debug docommand
Submitted batch job 7773917
[tkaiser2@kl1 slurm0]$
```

Results

```
[tkaiser2@kl1 slurm0]$sort spread
1743690724.618852371
1743690724.618885719
1743690724.619073134
1743690724.619128815
1743690724.619943391
1743690724.620182445
1743690724.620212055
1743690724.620769746
1743690724.635477415
1743690724.635745630
1743690724.635746714
1743690724.635955550
1743690724.635969448
1743690724.636509942
1743690724.636712457
1743690724.636839603
[tkaiser2@kl1 slurm0]$
```

Allows you to submit a bunch of different jobs with different data sets without editing your script.

Script to build/run a Parallel (MPI/OpenMP) Job

```
#!/bin/bash
##SBATCH --job-name="atest"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH -o /scratch/%u/%j.out
#SBATCH -e /scratch/%u/%j.err
##SBATCH --partition=debug
##SBATCH --account=hpcapps
##SBATCH --mail-type=ALL
##SBATCH --mail-user=joeuser@nrel.gov
```

```
# Setup
export BASE=$SLURM_SUBMIT_DIR
mkdir -p $BASE/$SLURM_JOB_ID
cd $BASE/$SLURM_JOB_ID
```

```
# Save info
cat $0 > $SLURM_JOB_ID.script
printenv > $SLURM_JOB_ID.env
```

```
# Get and build glorified "Hello World"
module load intel-oneapi-mpi
curl -s https://raw.githubusercontent.com/timkphd/examples/master/hybrid/pstream.c -o pstream.c
mpicc -fopenmp pstream.c -o pstream
```

```
echo "First run"
export OMP_NUM_THREADS=1
srun ./pstream -F -D -t 5
echo "Second run:"
export OMP_NUM_THREADS=18
srun --nodes=1 --ntasks=4 --ntasks-per-node=4 ./pstream -F -D -t 5 > pstream.out
```

```
#cp $SLURM_SUBMIT_DIR/$SLURM_JOB_ID.out .
#cp $SLURM_SUBMIT_DIR/$SLURM_JOB_ID.err .
cp /scratch/$USER/$SLURM_JOB_ID.out .
cp /scratch/$USER/$SLURM_JOB_ID.err .
```

Scripts contain “comments”
designated with a # that are
interpreted by SLURM
and normal shell commands
##SBATCH are also comments

srun is the “normal”
command to run
something in parallel

Output

```
[tkaiser2@kl1 9217648]$ls -lt
total 88
-rw-rw---- 1 tkaiser2 tkaiser2 5042 Jun  2 12:42 pstream.out
-rwxrwx--- 1 tkaiser2 tkaiser2 36024 Jun  2 12:42 pstream
-rw-rw---- 1 tkaiser2 tkaiser2 14432 Jun  2 12:42 pstream.c
-rw-rw---- 1 tkaiser2 tkaiser2 23743 Jun  2 12:42 9217648.env
-rw-rw---- 1 tkaiser2 tkaiser2  934 Jun  2 12:42 9217648.script
[tkaiser2@kl1 9217648]$
```

```
[tkaiser2@kl1 9217648]$head pstream.out
```

```
MPI VERSION Intel(R) MPI Library 2021.12 for Linux* OS
```

task	thread	node name	first task	# on node	core
0000	0000	x1005cls0b1n0	0000	0000	0090
0000	0001	x1005cls0b1n0	0000	0000	0000
0000	0011	x1005cls0b1n0	0000	0000	0074
0000	0013	x1005cls0b1n0	0000	0000	0029
0000	0017	x1005cls0b1n0	0000	0000	0079
0000	0005	x1005cls0b1n0	0000	0000	0050
0000	0014	x1005cls0b1n0	0000	0000	0039

```
[tkaiser2@kl1 9217648]$
```

```
[tkaiser2@kl1 9217648]$
```

```
[tkaiser2@kl1 9217648]$tail pstream.out
```

0003	0009	x1005cls0b1n0	0000	0003	0047
0003	0010	x1005cls0b1n0	0000	0003	0068
0002	0001	x1005cls0b1n0	0000	0002	0009
0003	0003	x1005cls0b1n0	0000	0003	0004
0003	0014	x1005cls0b1n0	0000	0003	0082
0002	0015	x1005cls0b1n0	0000	0002	0006
0003	0007	x1005cls0b1n0	0000	0003	0067
0002	0003	x1005cls0b1n0	0000	0002	0077
0002	0006	x1005cls0b1n0	0000	0002	0018

```
total time      5.424
```

```
[tkaiser2@kl1 9217648]$
```

```
[tkaiser2@kl1 9217648]$
```

```
[tkaiser2@kl1 9217648]$ls /scratch/tkaiser2/*9217648*
```

```
/scratch/tkaiser2/9217648.err /scratch/tkaiser2/9217648.out
```

```
[tkaiser2@kl1 9217648]$
```


Output

```
[tkaiser2@kl1 7577744]$cat 7577744.out | egrep "task|^00" | sort -r
task      thread      node name    first task    # on node    core
0015      0000      x3104c0s5b0n0    0008          0007      0008
0014      0000      x3104c0s5b0n0    0008          0006      0007
0013      0000      x3104c0s5b0n0    0008          0005      0006
0012      0000      x3104c0s5b0n0    0008          0004      0005
0011      0000      x3104c0s5b0n0    0008          0003      0004
0010      0000      x3104c0s5b0n0    0008          0002      0003
0009      0000      x3104c0s5b0n0    0008          0001      0002
0008      0000      x3104c0s5b0n0    0008          0000      0001
0007      0000      x3101c0s13b0n0    0000          0007      0009
0006      0000      x3101c0s13b0n0    0000          0006      0008
0005      0000      x3101c0s13b0n0    0000          0005      0007
0004      0000      x3101c0s13b0n0    0000          0004      0006
0003      0000      x3101c0s13b0n0    0000          0003      0005
0002      0000      x3101c0s13b0n0    0000          0002      0004
0001      0000      x3101c0s13b0n0    0000          0001      0003
0000      0000      x3101c0s13b0n0    0000          0000      0002
[tkaiser2@kl1 7577744]$
```

Setting Default Accounts and Partition

- Jobs are “charged” against an account
- The account must be specified for a job to run
- You were given an account when you got your allocation
- One way to set account is with variables
 - SLURM_ACCOUNT - used by srun
 - SALLOC_ACCOUNT - used by salloc
 - SBATCH_ACCOUNT - used by sbatch
 - Most likely, just want to set them all the same

```
export SLURM_ACCOUNT=hpcapps
export SALLOC_ACCOUNT=$SLURM_ACCOUNT
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
```

```
export SBATCH_PARTITION=short
```


More about srun

- sbatch will get you a collection of nodes
- srun will actually launch your program in parallel
- In most cases you will use srun instead of mpirun/mpiexec
- How many instances:
 - Defaults to what is given in your script header

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
```
 - Can also:
 - `srun --tasks-per-node=4 --ntasks=8 myprogrm`
 - Many options See: `man srun`

Getting Status and Information

- `squeue`
- `sinfo`
- `scontrol show node`
- `sacctmgr`

queue

- Shows what is running or waiting to run
- By default it shows everyone's jobs
- Just show yours:
 - `queue -u $USER`
- queue has many potential output fields
- The `--format` option allows to select fields

queue: default output

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
461291	debug	benchmar	slarsen	PD	0:00	1	(ReqNodeNotAvail)
581892	standard	install	joetrip	PD	0:00	1	(Priority)
581891	parallel	install	joetrip	PD	0:00	1	(Priority)
489530	parallel	install	joetrip	PD	0:00	1	(Priority)
489529	parallel	install	joetrip	PD	0:00	1	(Priority)
488899	parallel	install	joetrip	PD	0:00	1	(Priority)
594990	parallel	install	joetrip	PD	0:00	1	(Priority)
594989	parallel	install	joetrip	PD	0:00	1	(Priority)
594988	parallel	install	joetrip	PD	0:00	1	(Priority)
582732	test	jaguar	toads	R	5:51:25	2	c9-[45-47]
582728	test	jaguar	toads	R	5:55:47	1	c4-49
582726	test	jaguar	toads	R	5:57:24	1	c4-49
582712	test	jaguar	toads	R	6:08:34	1	c8-8
581890	test	install	joetrip	PD	0:00	1	(Resources)

queue with a format

I have an alias "sq" for `sqlite3` with a format:

```
alias sq='squeue -u $USER --format="%10A%15I%15L%6D%20S%15P%15r%20V%N%\n"'
```

```
[tkaiser2@el1 runior]$ sq
```

JOBID	TIME_LIMIT	TIME_LEFT	NODES	START_TIME	PARTITION	REASON	SUBMIT_TIME	NODELIST
9382137	1:00:00	1:00:00	2	N/A	debug	QOSMaxJobsPerUs	2022-06-03T10:42:54	
9382131	1:00:00	1:00:00	2	N/A	debug	QOSMaxJobsPerUs	2022-06-03T10:42:37	
9382129	1:00:00	59:42	2	2022-06-03T10:42:39	debug	None	2022-06-03T10:42:22	r102u[34-35]
9382130	1:00:00	59:42	2	2022-06-03T10:42:39	debug	None	2022-06-03T10:42:29	r104u33,r105u33

```
[tkaiser2@el1 runior]$
```

```
queue -i 10 -u $USER
```

Run queue every 10 seconds

```
queue --start -u $USER
```

“Estimate” of when your jobs will start

sprio

List priorities of all jobs

```
[tkaiser2@kl6 pstream]$sq
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
7744567	gpu-h100	runit	tkaiser2	PD	0:00	2	(Resources)

```
[tkaiser2@kl6 pstream]$sprio -n -l --sort="Y" | head
```

JOBID	PARTITION	USER	ACCOUNT	PRIORITY	AGE	ASSOC	FAIRSHARE	JOBSIZE	PARTITION	QOSNAME	QOS	TRES
7308137	standard	linzq25	crerp	0.01030294	1.0000000	0.0000000	0.0216014	0.0056273	0.1000000	normal	0.0000000	
7744467	gpu-h100-	cbu	vonset	0.01085842	0.0015129	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744468	gpu-h100-	cbu	vonset	0.01085842	0.0015129	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744464	gpu-h100-	cbu	vonset	0.01085843	0.0015146	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744465	gpu-h100-	cbu	vonset	0.01085843	0.0015146	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744466	gpu-h100-	cbu	vonset	0.01085843	0.0015146	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744448	gpu-h100-	cbu	vonset	0.01085849	0.0015228	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744450	gpu-h100-	cbu	vonset	0.01085849	0.0015228	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	
7744451	gpu-h100-	cbu	vonset	0.01085849	0.0015228	0.0000000	0.1074309	0.0002063	0.1000000	standby	0.0000000	

```
[tkaiser2@kl6 pstream]$sprio -n -l --sort="Y" -u $USER | head
```

JOBID	PARTITION	USER	ACCOUNT	PRIORITY	AGE	ASSOC	FAIRSHARE	JOBSIZE	PARTITION	QOSNAME	QOS	TRES
7744567	gpu-h100	tkaiser2	hpcapps	0.03688496	0.0000000	0.0000000	0.3885369	0.0004087	0.1000000	normal	0.0000000	

```
[tkaiser2@kl6 pstream]$sprio -n -l | head -1 ;sprio -n -l --sort="Y" | tail
```

JOBID	PARTITION	USER	ACCOUNT	PRIORITY	AGE	ASSOC	FAIRSHARE	JOBSIZE	PARTITION	QOSNAME	QOS	TRES
7744565	short	jreyna	rescore	0.04172777	0.0000000	0.0000000	0.4409562	0.0002043	0.1000000	normal	0.0000000	
7744553	bigmem	pbrown	reedswet	0.04798196	0.0005522	0.0000000	0.5083525	0.0004020	0.1000000	normal	0.0000000	
7744462	bigmem	pbrown	reedswet	0.04798883	0.0015170	0.0000000	0.5083525	0.0004020	0.1000000	normal	0.0000000	
7744463	bigmem	pbrown	reedswet	0.04798883	0.0015170	0.0000000	0.5083525	0.0004020	0.1000000	normal	0.0000000	
7744460	bigmem	pbrown	reedswet	0.04798884	0.0015187	0.0000000	0.5083525	0.0004020	0.1000000	normal	0.0000000	
7744461	bigmem	pbrown	reedswet	0.04798884	0.0015187	0.0000000	0.5083525	0.0004020	0.1000000	normal	0.0000000	
7744459	bigmem	pbrown	reedswet	0.04798885	0.0015203	0.0000000	0.5083525	0.0004020	0.1000000	normal	0.0000000	
7669804	long	khanwale	nawi	0.05424028	0.2594626	0.0000000	0.3352535	0.0514496	0.1000000	high	1.0000000	
7624973	bigmeml	junchich	tkm4nc	0.06269915	0.5372305	0.0000000	0.6255760	0.0012059	0.1000000	normal	0.0000000	
7744480	gpu-h100-	mvansch	questaal	0.08201644	0.0010111	0.0000000	0.8741359	0.0035839	0.1000000	standby	0.0000000	

```
[tkaiser2@kl6 pstream]$
```

The larger the PRIORITY, the higher the job will be positioned in the queue.

Just because a job has a high priority it does not necessarily mean it will start soon.

Priorities

- The larger the PRIORITY, the higher the job will be positioned in the queue.
- Priority is a function of several factors including age and FAIRSHARE.
- <https://ni.cmu.edu/computing/knowledge-base/slurm-job-priority-wait-time/>
- Does a good explanation of priorities.
- However, their weights are much different than NREL'S.
 - Their “FAIRSHARE” is 0.
 - NREL - FAIRSHARE is important: give higher priority to accounts which have not used the machine much recently. Usage has a “half-life.”

scancel

- Cancel (kill/stop) your jobs
- `scancel 9382130`
 - Kills a particular job(s)
- `scancel -u $USER`
 - Kills all of your jobs

sinfo

Show information about partitions and nodes

```
[tkaiser2@vs-login-1 ENV]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
sm          up 1-00:00:00      1 drain vs-sm-0016
sm          up 1-00:00:00      4 down  vs-sm-[0001,0024,0028,0032]
sm          up 1-00:00:00      3 drain vs-sm-[0002-0003,0026]
sm          up 1-00:00:00     24 idle  vs-sm-[0004-0015,0017-0023,0025,0027,0029-0031]
gpu         up 1-00:00:00      2 drain vs-gpu-[0004,0006]
gpu         up 1-00:00:00      4 alloc vs-gpu-[0001-0003,0005]
lg          up 1-00:00:00      1 alloc vs-lg-0001
lg          up 1-00:00:00      1 down  vs-lg-0004
lg          up 1-00:00:00      2 drain vs-lg-[0010,0017]
lg          up 1-00:00:00     10 alloc vs-lg-[0002-0003,0006-0008,0012-0014,0016,0018]
std         up 1-00:00:00      2 drain vs-std-[0003,0026]
std         up 1-00:00:00      1 drain vs-std-0014
std         up 1-00:00:00     47 alloc vs-std-[0001-0002,0004-0006,0008,0010-0011,0013]
t           up 4:00:00        2 drain vs-t-[0004,0011]
t           up 4:00:00        4 drain vs-t-[0006,0009,0012-0013]
t           up 4:00:00        9 idle  vs-t-[0001-0003,0005,0007-0008,0010,0014-0015]
[tkaiser2@vs-login-1 ENV]$
```

This is Vermillion not Kestrel

Kestrel Partitions

<https://nrel.github.io/HPC/Documentation/Systems/Kestrel/Running/>

Partition Name	Description	Limits	Placement Condition
debug	Nodes dedicated to developing and troubleshooting jobs. Debug nodes with each of the non-standard hardware configurations are available.	- 1 job with a max of 2 nodes per user. - 2 GPUs per user. - 1x2 GPU node resources per user (Aurora 1.2 nodes). - 01.00.00 max walltime.	<code>--p debug</code> or <code>--partition debug</code>
short	Nodes that prefer jobs with walltimes <= 4 hours.	2016 nodes total. No limit per user.	<code>--time == 4:00:00</code> <code>--mem == 1048576</code> <code>--time == 1:00:00:00</code> [1000 nodes]
standard	Nodes that prefer jobs with walltimes <= 2 days.	2176 nodes total. 1020 nodes per user.	<code>--mem == 1048576</code> <code>--time == 1:00:00:00</code>
long	Nodes that prefer jobs with walltimes > 2 days. Maximum walltime of any job is 10 days.	325 nodes total. 252 nodes per user.	<code>--time == 10:00</code> <code>--mem == 246064</code> <code>--time == 1700000</code> [250 nodes]
bigmem	Nodes that have 2 TB of RAM and 5.6 TB NVMe local disk.	8 nodes total. 4 nodes per user.	<code>--mem == 246064</code> <code>--time == 2:00</code> <code>--time == 1700000</code>
bigmem2	Bigmem nodes that prefer jobs with walltimes > 2 days. Maximum walltime of any job is 10 days.	4 nodes total. 3 nodes per user.	<code>--mem == 246064</code> <code>--time == 2:00</code> <code>--time == 1700000</code>
hpc	CPU compute nodes with dual network interface cards.	312 nodes total. 258 nodes per user. Minimum 2 nodes per job.	<code>--p hpc</code> <code>--time <= 10:00</code> <code>--nodes >= 2</code>
hpc2	CPU compute nodes with 1.7TB NVMe local drives.	258 nodes total. 129 nodes per user.	<code>--p hpc2</code> <code>--time <= 2:00</code>
shared	Nodes that can be shared by multiple users and jobs.	64 nodes total. Half of partition per user. 2 days max walltime.	<code>--p shared</code> or <code>--partition shared</code>
shared2	Nodes that can be shared by multiple users and prefer jobs with walltimes > 2 days.	16 nodes total. 8 nodes per user.	<code>--p shared2</code> or <code>--partition shared2</code>
gpu-h100	Shareable GPU nodes with 4 NVIDIA H100 80GB Compute Accelerators.	130 nodes total. 65 nodes per user.	<code>--time == 4:00:00</code> <code>--time == 4:00:00</code>
gpu-h100a	Shareable GPU nodes that prefer jobs with walltimes <= 4 hours.	130 nodes total. 65 nodes per user.	<code>--time == 4:00:00</code> <code>--time == 4:00:00</code>
gpu-h100b	Shareable GPU nodes that prefer jobs with walltimes > 2 days.	26 GPU nodes total. 13 GPU nodes per user.	<code>--time == 4:00:00</code> <code>--time == 2:00</code>

sinfo

Show information about partitions and nodes

```
[tkaiser2@vs-login-1 ENV]$ sinfo -p lg
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
lg          up 1-00:00:00      1 alloc* vs-lg-0001
lg          up 1-00:00:00      1 down*  vs-lg-0004
lg          up 1-00:00:00      4  comp  vs-lg-[0005,0009,0011,0015]
lg          up 1-00:00:00      2 drain  vs-lg-[0010,0017]
lg          up 1-00:00:00     10 alloc  vs-lg-[0002-0003,0006-0008,0012-0014,0016,0018]
[tkaiser2@vs-login-1 ENV]$
[tkaiser2@vs-login-1 ENV]$
[tkaiser2@vs-login-1 ENV]$ sinfo -N -n vs-t-0001
NODELIST  NODES PARTITION STATE
vs-t-0001      1          t idle
[tkaiser2@vs-login-1 ENV]$
[tkaiser2@vs-login-1 ENV]$
[tkaiser2@vs-login-1 ENV]$ sinfo -N | grep idle      or      (sinfo -N -t idle)
vs-sm-0004      1          sm idle
vs-sm-0005      1          sm idle
vs-sm-0027      1          sm idle
vs-sm-0030      1          sm idle
vs-sm-0031      1          sm idle
vs-t-0001      1          t idle
vs-t-0014      1          t idle
vs-t-0015      1          t idle
[tkaiser2@vs-login-1 ENV]$
```


scontrol

- Many functions, only a few you will most likely ever use:
- Hold/release a job
 - `scontrol uhold 9382268`
 - `scontrol release 9382268`
- Get detailed information about a node(s)/job(s)
 - `scontrol show node`
 - `scontrol show job #####`

scontrol show node

```
[tkaiser2@el1 runior]$ scontrol show node r103u01
```

```
NodeName=r103u01 Arch=x86_64 CoresPerSocket=18
```

```
CPUAlloc=36 CPUTot=36 CPULoad=26.87
```

```
AvailableFeatures=hy
```

```
ActiveFeatures=hy
```

```
Gres=gpu:v100:2
```

```
NodeAddr=r103u01 NodeHostName=r103u01 Version=21.08.5
```

```
OS=Linux 3.10.0-1062.9.1.el7.x86_64 #1 SMP Fri Dec 6 15:49:49 UTC 2019
```

```
RealMemory=751616 AllocMem=0 FreeMem=718868 Sockets=2 Boards=1
```

```
State=ALLOCATED ThreadsPerCore=1 TmpDisk=24000000
```

```
Weight=1 Owner=N/A MCS_label=N/A
```

```
Partitions=gpu,gpu-stdby,gpul,gpul-stdby,bigscratch
```

```
BootTime=2022-04-07T08:39:15 SlurmdStartTime=2022-04-07T08:44:41
```

```
LastBusyTime=2022-06-02T07:55:17
```

```
CfgTRES=cpu=36,mem=734G,billing=36,gres/gpu=2
```

```
AllocTRES=cpu=36,gres/gpu=2
```

```
CapWatts=n/a
```

```
CurrentWatts=0 AveWatts=0
```

```
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```


scontrol show job

```
[tkaiser2@r4i7n35 tkaiser2]$ scontrol show job 9443283
JobId=9443283 JobName=interactive
  UserId=tkaiser2(131364) GroupId=tkaiser2(131364) MCS_label=N/A
  Priority=248099184 Nice=0 Account=hpcapps QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=0 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
  RunTime=00:00:17 TimeLimit=01:00:00 TimeMin=N/A
  SubmitTime=2022-06-08T12:28:31 EligibleTime=2022-06-08T12:28:31
  AccrueTime=2022-06-08T12:28:31
  StartTime=2022-06-08T12:28:36 EndTime=2022-06-08T13:28:36 Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-06-08T12:28:36 Scheduler=Main
  Partition=debug AllocNode:Sid=el3:27289
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=r4i7n35
  BatchHost=r4i7n35
  NumNodes=1 NumCPUs=36 NumTasks=4 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=36,node=1,billing=36
  Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
  Features=[hy|ehy] DelayBoot=00:00:00
  OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
  Command=(null)
  WorkDir=/lustre/eaglefs/scratch/tkaiser2
  Switches=1@2-00:00:00
  Power=
[tkaiser2@r4i7n35 tkaiser2]$
```


scontrol to modify a job

```
todebug () {  
    scontrol update JobId=$1 TimeLimit=01:00:00  
    scontrol update JobId=$1 Partition=debug  
}
```

```
tolong ()  
{  
    scontrol update JobId=$1 Partition=long  
}
```

```
toshort ()  
{  
    scontrol update JobId=$1 Partition=short  
}
```

```
to01 ()  
{  
    scontrol update JobId=$1 TimeLimit=00:01:00  
}
```

```
to05 ()  
{  
    scontrol update JobId=$1 TimeLimit=00:05:00  
}
```

```
to60 ()  
{  
    scontrol update JobId=$1 TimeLimit=00:60:00  
}
```

Here are some functions I have defined to modify jobs.

These are in the "utils" directory; to be discussed shortly.

sacct

```
[tkaiser2@vs-login-1 ENV]$ sacct -S 2025-01-04 -E 2025-01-06 -u $USER
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
50005503	rfm_Hello+	gpu	hpcapps	60	COMPLETED	0:0
50005503.ba+	batch		hpcapps	30	COMPLETED	0:0
50005503.0	HelloTest+		hpcapps	60	COMPLETED	0:0
50005504	rfm_Hello+	gpu	hpcapps	60	CANCELLED+	0:0
50005504.ba+	batch		hpcapps	30	CANCELLED	0:15
50005504.0	HelloTest+		hpcapps	60	CANCELLED	0:15
50005505	rfm_Hello+	lg	hpcapps	120	COMPLETED	0:0
50005505.ba+	batch		hpcapps	60	COMPLETED	0:0
50005505.0	HelloTest+		hpcapps	120	COMPLETED	0:0
50005506	rfm_Hello+	lg	hpcapps	120	COMPLETED	0:0
50005506.ba+	batch		hpcapps	60	COMPLETED	0:0
50005506.0	HelloTest+		hpcapps	120	COMPLETED	0:0
50005507	rfm_Hello+	lg	hpcapps	120	COMPLETED	0:0
50005507.ba+	batch		hpcapps	60	COMPLETED	0:0
50005507.0	HelloTest+		hpcapps	120	COMPLETED	0:0
50005508	rfm_Hello+	lg	hpcapps	120	COMPLETED	0:0
50005508.ba+	batch		hpcapps	60	COMPLETED	0:0
50005508.0	HelloTest+		hpcapps	120	COMPLETED	0:0

```
[tkaiser2@vs-login-1 ENV]$
```

Show history of my jobs between January 4 and January 6

sacct

- Like most commands sacct has a format option
- Things important to me:
 - JobID
 - Start
 - Nodelist
 - Starting directory
- I have a bash function (recent) that
 - Takes days range (not date)
 - Calls sacct with the above requested
 - Does some filtering

recent - sacct

```
[tkaiser2@vs-login-1 ENV]$ recent 4 1
50022417 2022-05-31T12:18:06 vs-sm-0025 /home/tkaiser2
50022418 2022-05-31T12:22:11 vs-sm-0025 /home/tkaiser2
50022433 2022-06-01T14:30:09 None assigned /projects/hpcapps/tkaiser2/runior
50022434 2022-06-01T14:30:38 None assigned /projects/hpcapps/tkaiser2/runior
50022435 2022-06-01T14:31:26 vs-sm-[0017-0020]/projects/hpcapps/tkaiser2/runior
50022436 2022-06-01T14:33:51 vs-sm-[0017-0020]/projects/hpcapps/tkaiser2/runior
50022437 2022-06-01T14:37:40 vs-sm-[0017-0020]/projects/hpcapps/tkaiser2/runior
50022438 2022-06-01T14:39:13 vs-sm-[0017-0020]/projects/hpcapps/tkaiser2/runior
50022439 2022-06-01T14:42:13 vs-sm-[0029-0030]/projects/hpcapps/tkaiser2/runior
50022440 2022-06-01T14:44:31 vs-sm-[0030-0031]/projects/hpcapps/tkaiser2/runior
50022441 2022-06-01T14:46:29 vs-sm-[0030-0031]/projects/hpcapps/tkaiser2/runior
50022442 2022-06-01T14:50:40 vs-sm-[0030-0031]/projects/hpcapps/tkaiser2/runior
50022443 2022-06-01T14:54:33 vs-sm-[0030-0031]/projects/hpcapps/tkaiser2/runior
50022444 2022-06-01T14:58:43 vs-sm-[0030-0031]/projects/hpcapps/tkaiser2/runior
50022445 2022-06-01T15:06:47 vs-sm-[0030-0031]/projects/hpcapps/tkaiser2/runior
50022446 2022-06-02T14:47:12 None assigned /projects/hpcapps/tkaiser2/runior
50022447 2022-06-01T16:29:33 vs-sm-[0029-0030]/projects/hpcapps/tkaiser2/runior
50022448 2022-06-01T16:31:14 vs-sm-[0017-0018]/projects/hpcapps/tkaiser2/runior
50022449 2022-06-01T16:33:26 vs-sm-0025 /projects/hpcapps/tkaiser2/runior
[tkaiser2@vs-login-1 ENV]$
```

Recent is function I defined. Source is in the examples.

sacctmgr

Another command with many functions but
you will most likely only use it to find out
what accounts you have on a machine

```
[thk2@e11 ENV]$ sacctmgr show associations user=$USER format=account%15
Account
-----
hpcapps
continental
mobility
msoc
wks
naermpcm
[thk2@e11 ENV]$
```

```
alias accounts='sacctmgr show associations user=$USER format=account%15'
```

salloc

- **salloc**
- Used to allocate resources for a job in an interactive mode. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

```
salloc --nodes=2 --ntasks=72 --time=01:00:00 --partition=debug --account=hpcapps
```


Running Interactively

- Sometimes you want to grab a node or nodes and run commands
- Almost like running on login node except you are actually logged in to a compute node and you have access to all of the nodes you have allocated
- After you get started you can run simple commands or run in parallel with srun
- The command to do this is **salloc**
- Note: The way this works has recently changed

Grab nodes and run with salloc

```
[thk@el1 ~]$ salloc --nodes=2 --ntasks=72 --time=01:00:00 --partition=debug --account=hpcapps
salloc: Pending job allocation 9382710
salloc: job 9382710 queued and waiting for resources
salloc: job 9382710 has been allocated resources
salloc: Granted job allocation 9382710
salloc: Waiting for resource configuration
salloc: Nodes r102u[34-35] are ready for job
[thk@r102u34 ~]$
[thk@r102u34 ~]$
[thk@r102u34 ~]$ printenv SLURM_NODELIST
r102u[34-35]
[thk@r102u34 ~]$
[thk@r102u34 ~]$
[thk@r102u34 ~]$ srun -n 4 --distribution=block -l hostname
1: r102u34
0: r102u34
2: r102u35
3: r102u35
[thk@r102u34 ~]$
```


Some Notes:

- Interactive (**salloc**) session: `.bashrc` gets sourced
 - Your environment looks like a new login shell
 - Will be extra variables for SLURM
 - Setting you changed after getting on Kestrel will not be on the login nodes
- **sbatch**: `.bashrc` does not get sourced
 - Environment defined on submitting node more or less gets passed as is
 - Module “counts” will change

More sbatch /salloc options

- Any option specified with #SBATCH can be overwritten with an option on the command line
 - `sbatch --time=1-00:00:00 my script`
- You can ask for nodes with a specific amount of memory, scratch disk, gpus
- `--x` Request that a specific list of hosts **not** be included for your job
- These options are specific to Kestrel but other machines will have similar options
 - `--mem=700000 700 Gbytes` (big memory nodes)
 - `--tmp=1500000 1.5 Tbytes` of scratch
 - On Kestrel, setting specifics can put you in different partitions

Directory for today...

```
[tkaiser2@k11 slurm25]$ls -Rr
```

```
.:  
utils  slurm1  slurm0  pstream.c  mstream.cu  mpmd  makefile  gpu  doall  
array  affinity
```

```
./utils:  
tymer  slurmnodes  onnodes  ongpus  myalias  doall  allow
```

```
./slurm1:  
script  makefile
```

```
./slurm0:  
script  makefile  docommand
```

```
./mpmd:  
script  mlist.py  makefile  hellof.f90  helloc.c
```

```
./gpu:  
mstream.cu  makefile  doit  dogpu
```

```
./array:  
uselist  usedirs  useboth  tymer  setarray.py  makefile  invertp.py  invertc.c  
doit  doboth  bot.tgz
```

```
./affinity:  
makefile  doaff  break  
[tkaiser2@k11 slurm25]$
```

What's in the utils dir

source utils/myalias

- Get a nice node list for a job
 - **thenodes**
- Show your accounts
 - **accounts**
- Show information about a job
 - **showjob**
- Show what I have in the queue
 - **sq**
- Print a clean module list in load order
 - **mylist**
- Show my recent jobs
 - **recent**
- Show my recent jobs with run time
 - **recentt**
- Show a list of nodes scheduled to soon (maybe) be used
 - **pnodes**
- Do a salloc of various flavors
 - **alloc**
- Send a job to debug, long, short, set time to 1, 5, 60 minutes
 - **todebug**

What's in the utils dir

```
[tkaiser2@kl1 utils]$onnodes -h
```

```
./onnodes [JOB_ID ...]
```

OR

```
./ongpus [JOB_ID ...]
```

For each specified or running JOB

For each NODE of the JOB

Show what user has running on each core

If NODE has GPUs show what is running on the GPUs

Must be called as ongpus to get gpu info

```
[tkaiser2@kl1 utils]$slurmnodes -h
```

USAGE:

slurmnodes compressed node list + -Fattribute1,attribute2,...

Where a compressed node list is of the form:

```
x1002c4s5b1n1,x1002c5s2b0n0,x1002c5s5b1n0,x1002c6s3b1n[0-1]
```

Without any inputs:

default node is x1000c0s0b0n1

default attributes are: ['CPUAlloc', 'CPULoad', 'FreeMem']

all is a valid input for both nodes and attributes

Examples:

Get the state of all nodes

```
slurmnodes all -FState
```

Get a list of attributes

```
slurmnodes -Fall
```

OR

```
slurmnodes -F
```

```
[tkaiser2@kl1 utils]$tymer -h
```

USAGE:

```
./tymer [file] [comment]
```

With no input on the command line prints time in seconds and date to the screen.

With a file name on the command line it reads the file, if it exists, and prints a delta time from the last time this program updated the file and appends the time information to the file.

Note: file can be /dev/null or ""

If the file does not exist it creates it and write the current time information.

You can add optional comments that will be added to the end of the line.

tymer can be called as a function:

```
from tymer import *
```

```
tymer() prints to stdout
```

```
tymer("file") prints to file
```

```
tymer("-i") use an internal file for saving time
```

```
tymer(["file","comments"]) prints to file with comments
```

```
tymer(["","comments"]) prints to stdout with comments
```

```
tymer(["/dev/null","comments"]) prints to stdout with comments
```

```
[tkaiser2@kl1 7802275]$allow
```

Generate a task/thread mask for srun

Slurm array jobs

- Slurm allows array jobs:

Job arrays offer a mechanism for submitting and managing collections of similar jobs quickly and easily. All jobs must have the same initial options (e.g. size, time limit, etc.), however it is possible to change some of these options after the job has begun execution using the command specifying the *JobID* of the array or individual *ArrayJobID*.

Job arrays will have two additional environment variable set. **SLURM_ARRAY_JOB_ID** will be set to the first job ID of the array. **SLURM_ARRAY_TASK_ID** will be set to the job array index value. For example a job submission of this sort:

```
sbatch --array=1-3 some_script
```

will generate a job array containing three jobs. If the sbatch command responds

```
Submitted batch job 36
```

then the environment variables will be set as follows:

```
SLURM_JOBID=36  
SLURM_ARRAY_JOB_ID=36  
SLURM_ARRAY_TASK_ID=1
```

```
SLURM_JOBID=37  
SLURM_ARRAY_JOB_ID=36  
SLURM_ARRAY_TASK_ID=2
```

```
SLURM_JOBID=38  
SLURM_ARRAY_JOB_ID=36  
SLURM_ARRAY_TASK_ID=3
```


Apology

- Our example is complex
- It could be a talk unto itself
- Goal is not complete understanding
- Show what is possible
- You can dig through the example and use it as a basis for your work.

Slurm array jobs

```
[joeuser@lognode memory]$ sbatch --nodelist=node002 --array=1-4 array
Submitted batch job 11968
```

```
[joeuser@lognode memory]$ squeue -u joeuser
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
11968_1	debug	hybrid	joeuser	R	0:05	1	node002
11968_2	debug	hybrid	joeuser	R	0:05	1	node002
11968_3	debug	hybrid	joeuser	R	0:05	1	node002
11968_4	debug	hybrid	joeuser	R	0:05	1	node002

Here we run 4 jobs on the same node producing:

```
[joeuser@lognode memory]$ ls -lt | head
total 12768
-rw-rw-r-- 1 joeuser joeuser 1791 Sep 11 14:50 stderr.11968
-rw-rw-r-- 1 joeuser joeuser 803 Sep 11 14:50 stderr.11970
-rw-rw-r-- 1 joeuser joeuser 803 Sep 11 14:50 stderr.11971
-rw-rw-r-- 1 joeuser joeuser 803 Sep 11 14:50 stderr.11969
-rw-rw-r-- 1 joeuser joeuser 2763 Sep 11 14:50 fillmemc.sinput1.output.11968.11968.1
-rw-rw-r-- 1 joeuser joeuser 2763 Sep 11 14:50 fillmemc.sinput4.output.11971.11968.4
-rw-rw-r-- 1 joeuser joeuser 2763 Sep 11 14:50 fillmemc.sinput2.output.11969.11968.2
-rw-rw-r-- 1 joeuser joeuser 2763 Sep 11 14:50 fillmemc.sinput3.output.11970.11968.3
```

A set of array jobs can run on the same node or many different nodes.

On Kestrel running in the shared partition will "allow" subjobs to run on the same node at the same time but other considerations might prevent it. **export SBATCH_PARTITION=shared**

Our Array Job Example Overview

- Standard header
- We want to share a node for multiple instances so we set the memory for a node
- We created a directory hierarchy with a directory for each job and under that a directory for each subjob
- Gets different inputs for each subjob. We show how to do that in two different ways.
- Run our program

Array Job Example

- Our program does 4 matrix inversions (twice so you should end up with the same matrices with which you started)
- The "array" aspect of this example is that each subjob will invert a different set of matrices.
- The program takes an input from the command line of the form
 - 10 20 30 40 200
- The first 4 integers give the diagonal values for the matrices and the last value is the size
- The program returns times and errors for the inversions

Array Job Example

- We run two different ways
 - Input for each run is in a separate directory
 - Our script selects a directory
 - Each instance will read data from a different directory
- There is a single input files
 - Our script selects a line from the input
 - Each instance will use a different line of input

Array Job Example

- Recall that when you run an array job you get both
 - `$SLURM_ARRAY_JOB_ID`
 - Like the normal `SLURM_JOB_ID`, say 2197923
 - `$SLURM_ARRAY_TASK_ID`
 - Numbers in the range n1 to n2; the array values set on your sbatch line. Example 1-24
 - `sbatch --array=1-24 array.job`

Array Job Example

- Our script sets

```
export JOB_ID=$SLURM_ARRAY_JOB_ID  
export SUB_ID=$SLURM_ARRAY_TASK_ID
```

- and uses these values to create a main directory for all our jobs and a sub directory for each subjob

Our script(s) in details

```
#!/bin/bash -e
#SBATCH --job-name="array_job"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=00:05:00
##### sending output to /dev/null will suppress it
##### this is a good idea for array jobs lest it
##### create extra output for each subjob
##SBATCH -o /dev/null
##SBATCH --exclusive=user
##SBATCH --account=hpcapps
```

For Kestrel don't use
exclusive

```
#SBATCH --mem=5G
```

mem is what is required for each subjob.
If this is not set your subjobs might not be able
to share cores on a node

```
#-----
# example invocation
# sbatch --array=1-24 array_simple
```

```
module load python
```

```
export OMP_NUM_THREADS=4
```

```
#run with either the C or python version of our program
#export EXE=invertp.py
export EXE=invertc
```

```
# go to our starting directory
cd $SLURM_SUBMIT_DIR
```

```
# get the JOB and SUBJOB ID
if [[ $SLURM_ARRAY_JOB_ID ]] ; then
    export JOB_ID=$SLURM_ARRAY_JOB_ID
    export SUB_ID=$SLURM_ARRAY_TASK_ID
else
    export JOB_ID=$SLURM_JOB_ID
    export SUB_ID=1
fi
```

We will use
\$SLURM_ARRAY_JOB_ID
and \$SLURM_ARRAY_TASK_ID
to use as directory names.
\$SLURM_ARRAY_TASK_ID
Will also be used to grab the correct line from
our input.


```
# make a top level directory for the job
# if it does not already exist
mkdir -p $JOB_ID
cd $JOB_ID

# make a directory for the subjob and go there
mkdir -p $SUB_ID
cd $SUB_ID
```

```
# Make a copy of our script
cat $0 > myscript
```

```
# Get the name of our LIST, default to list
if [ -z ${LIST+x} ]; then echo "LIST is unset"; export LIST=list ; else echo "LIST is set to '${LIST}'"; fi
```

We are creating a directory for each job and subjob.

Directory \$JOB_ID

Subdirectories
\$JOB_ID/\$SUB_ID

Get the name of our file list or assume it is “list”

Example #1

Data is in a directory

```
# Here we assume that our LIST is a set of directories  
# under a "main" directory "inputs" each containing data sets.  
# For our current program we require each directory contain an  
# input file myinput.  
#
```

```
# We get the directory from the list  
dir=`head -n $SUB_ID $SLURM_SUBMIT_DIR/$LIST | tail -1`
```

```
# Copy our input from the directory  
cp -r $SLURM_SUBMIT_DIR/inputs/$dir/* .  
printenv > envs
```

```
# Run our job  
hostname > node  
echo $SLURM_SUBMIT_DIR/inputs/$dir > directory
```

```
$SLURM_SUBMIT_DIR/tymer timer start_time  
$SLURM_SUBMIT_DIR/$EXE `cat myinput` > output  
$SLURM_SUBMIT_DIR/tymer timer end_time
```


Example #2

Each line contains input

```
# Here we assume that our each line of our LIST contains
# data for our program.
#
# Grab the line
  export input=`head -n $SUB_ID $SLURM_SUBMIT_DIR/$LIST | tail -1`
  printenv > envs

# Run our job
  hostname > node

$SLURM_SUBMIT_DIR/tymer timer start_time
echo $input > myinput
eval $SLURM_SUBMIT_DIR/$EXE `cat myinput` > output
$SLURM_SUBMIT_DIR/tymer timer end_time
```

Our makefile

```
[timk@colostate.edu@login12 array]$ vi makefile
```

```
all:invertc input
```

```
invertc:invertc.c  
    gcc -O3 -fopenmp invertc.c -o invertc
```

```
input:setarray.py  
    rm -rf in_list list inputs  
    ./setarray.py 100
```

```
clean:  
    rm -rf invertc in_list dir_list inputs
```

```
backup:  
    tar -czf array.tgz array.job doit invertc.c makefile setarray.py tymer invertp.py bot
```

The makefile does two things. It builds the matrix inversion program `invertc` and runs the python program `setarray.py`.

The python program creates our input files “`in_list`” and “`dir_list`” and a directory “`inputs`.”

The directory “`inputs`” contains 100 subdirectories each with input for our program. The file `dir_list` contains a list of these directories.

The file `in_list` contains 100 lines of input for our program, one line per iteration.

The file doit runs make to create our program and data sets. It also will submit our batch script as an array

```
[timk@colostate.edu@shas0136 array]$ cat doit
# make our program and data set
make clean
make
```

```
#run assuming each line of "in_list" contains our input
export LIST=in_list
sbatch --array=1-24    usedirs
sbatch --array=25-48  usedirs
```

It also will submit our batch script 4 times. Each submission will have 24 subjobs.

```
#run assuming data in in directories specified in "dir_list"
export LIST=dir_list
export USEDIRS=yes
sbatch --array=1-24    uselist
sbatch --array=25-48  uselist
```

For our first set of submissions we assume that our data is in a set of directories

```
unset LIST
unset USEDIRS
```

For the second two we assume our input is all in the single file in_list

```
source doit
rm -rf invertc in_list dir_list inputs slurm*out
gcc -O3 -fopenmp invertc.c -o invertc
rm -rf in_list dir_list inputs
./setarray.py 100
Submitted batch job 7749134
Submitted batch job 7749135
Submitted batch job 7749136
Submitted batch job 7749137
[tkaiser2@kl6 array]$
```

We source the file which builds the program and submits our jobs.


```
[tkaiser2@kl6 array]$cat makefile
all:invertc input
```

```
invertc:invertc.c
    gcc -O3 -fopenmp invertc.c -o invertc
```

```
# Our python program creates a file in_list that
# contains 100 lines of input, one for each run
# and a directory "inputs" that contains 100
# subdirectories each with an input file. The
# file dir_list contains a list of the directories
```

```
input:setarray.py
    rm -rf in_list dir_list inputs
    ./setarray.py 100
```

```
clean:
    rm -rf invertc in_list dir_list inputs slurm*out
```

```
backup:
    tar -czf array.tgz bot doseperate dotogether invertc.c invertp.py makefile setarray.py tymer useboth usedirs uselist
```

makefile

```
[tkaiser2@kl6 array]$head -5 in_list
74 50 62 29 400
66 13 7 69 400
60 36 88 84 400
17 84 19 57 400
42 38 14 91 400
[tkaiser2@kl6 array]$
```

```
[tkaiser2@kl6 array]$head -5 dir_list
set001
set002
set003
set004
set005
```

```
[tkaiser2@kl6 array]$cat inputs/set001/myinput
63 67 47 55 200
[tkaiser2@kl6 array]$cat inputs/set005/myinput
44 65 99 50 200
[tkaiser2@kl6 array]$
```

Data sets


```
[tkaiser2@kl6 array]$ls slurm-774929*
slurm-7749293_10.out  slurm-7749293_23.out  slurm-7749294_29.out  slurm-7749294_43.out  slurm-7749295_18.out  slurm-7749295_8.out  slurm-7749296_37.out
slurm-7749293_11.out  slurm-7749293_24.out  slurm-7749294_30.out  slurm-7749294_44.out  slurm-7749295_19.out  slurm-7749295_9.out  slurm-7749296_38.out
slurm-7749293_12.out  slurm-7749293_2.out  slurm-7749294_31.out  slurm-7749294_45.out  slurm-7749295_1.out  slurm-7749296_25.out  slurm-7749296_39.out
slurm-7749293_13.out  slurm-7749293_3.out  slurm-7749294_32.out  slurm-7749294_46.out  slurm-7749295_20.out  slurm-7749296_26.out  slurm-7749296_40.out
slurm-7749293_14.out  slurm-7749293_4.out  slurm-7749294_33.out  slurm-7749294_47.out  slurm-7749295_21.out  slurm-7749296_27.out  slurm-7749296_41.out
slurm-7749293_15.out  slurm-7749293_5.out  slurm-7749294_34.out  slurm-7749294_48.out  slurm-7749295_22.out  slurm-7749296_28.out  slurm-7749296_42.out
slurm-7749293_16.out  slurm-7749293_6.out  slurm-7749294_35.out  slurm-7749295_10.out  slurm-7749295_23.out  slurm-7749296_29.out  slurm-7749296_43.out
slurm-7749293_17.out  slurm-7749293_7.out  slurm-7749294_36.out  slurm-7749295_11.out  slurm-7749295_24.out  slurm-7749296_30.out  slurm-7749296_44.out
slurm-7749293_18.out  slurm-7749293_8.out  slurm-7749294_37.out  slurm-7749295_12.out  slurm-7749295_2.out  slurm-7749296_31.out  slurm-7749296_45.out
slurm-7749293_19.out  slurm-7749293_9.out  slurm-7749294_38.out  slurm-7749295_13.out  slurm-7749295_3.out  slurm-7749296_32.out  slurm-7749296_46.out
slurm-7749293_1.out  slurm-7749294_25.out  slurm-7749294_39.out  slurm-7749295_14.out  slurm-7749295_4.out  slurm-7749296_33.out  slurm-7749296_47.out
slurm-7749293_20.out  slurm-7749294_26.out  slurm-7749294_40.out  slurm-7749295_15.out  slurm-7749295_5.out  slurm-7749296_34.out  slurm-7749296_48.out
slurm-7749293_21.out  slurm-7749294_27.out  slurm-7749294_41.out  slurm-7749295_16.out  slurm-7749295_6.out  slurm-7749296_35.out
slurm-7749293_22.out  slurm-7749294_28.out  slurm-7749294_42.out  slurm-7749295_17.out  slurm-7749295_7.out  slurm-7749296_36.out
```

```
[tkaiser2@kl6 array]$cat slurm-7749294_36.out
LIST is set to 'dir_list'
1743539148.516247 Tue Apr 1 14:25:48 2025 0.000 0.000 start_time
1743539148.576869 Tue Apr 1 14:25:48 2025 0.061 0.061 end_time
[tkaiser2@kl6 array]$

[tkaiser2@kl6 array]$ls 7*
7749293:
1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 23 24 3 4 5 6 7 8 9

7749294:
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

7749295:
1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 23 24 3 4 5 6 7 8 9

7749296:
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
```

```
[tkaiser2@kl6 array]$ls 7749293/1
directory envs myinput myscript node output timer

[tkaiser2@kl6 array]$cat 7749293/1/output
45 56 19 17 200
section 1 start time= 0.00016499 end time= 0.0090261 error= 4.21146e-11
section 2 start time= 0.00016713 end time= 0.0094981 error= 2.35085e-11
section 3 start time= 0.00016904 end time= 0.0093021 error= 3.41071e-10
section 4 start time= 0.00016904 end time= 0.0093081 error= 4.80289e-11

[tkaiser2@kl6 array]$ls 7749295/1
envs myinput myscript node output timer

[tkaiser2@kl6 array]$cat 7749295/1/output
48 12 38 77 400
section 1 start time= 0.00024509 end time= 0.073825 error= 1.78e-10
section 2 start time= 0.00024605 end time= 0.074992 error= 1.76297e-09
section 3 start time= 0.00024891 end time= 0.074665 error= 7.64311e-11
section 4 start time= 0.00024891 end time= 0.073795 error= 4.19306e-10
[tkaiser2@kl6 array]$
```

When finished we get 4 top level directories, each with 24 subdirectories.


```
[timk@colostate.edu@shas0136 test]$ ls 2197925  
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
```

```
[timk@colostate.edu@shas0136 test]$ ls 2197925/25  
directory envs myinput myscript node output timer
```

```
[timk@colostate.edu@shas0136 test]$ cat 2197925/25/directory  
/projects/timk@colostate.edu/test/inputs/set024
```

```
[timk@colostate.edu@shas0136 test]$ cat 2197925/25/myinput  
10 84 20 50 200
```

```
[timk@colostate.edu@shas0136 test]$ cat 2197925/25/output  
10 84 20 50 200  
section 1 start time= 5.7936e-05 end time= 0.019413 error= 3.87013e-12  
section 2 start time= 5.7936e-05 end time= 0.019923 error= 9.82607e-12  
section 3 start time= 0.019456 end time= 0.038625 error= 4.5729e-12  
section 4 start time= 0.019967 end time= 0.039389 error= 6.69376e-12
```

```
[timk@colostate.edu@shas0137 test]$ cat 2197925/25/timer  
1555962929.441435 Mon Apr 22 13:55:29 2019 0.000 0.000 start_time  
1555962929.523498 Mon Apr 22 13:55:29 2019 0.082 0.082 end_time  
[timk@colostate.edu@shas0137 test]$
```

Each sub directory contains the output from a single run. Here we have the output from the set of runs where our input came from a set of directories.

```
[timk@colostate.edu@shas0136 test]$ ls 2197922  
1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 23 24 3 4 5 6 7 8 9
```

```
[timk@colostate.edu@shas0136 test]$ ls 2197922/1  
envs input myscript node output timer
```

```
[timk@colostate.edu@shas0136 test]$ cat 2197922/1/input  
87 94 27 17 400
```

```
[timk@colostate.edu@shas0136 test]$ cat 2197922/1/output  
87 94 27 17 400  
section 1 start time= 6.2943e-05 end time= 0.15121 error= 4.31382e-11  
section 2 start time= 6.2943e-05 end time= 0.15126 error= 1.53059e-11  
section 3 start time= 0.15138 end time= 0.30201 error= 3.08275e-11  
section 4 start time= 0.15144 end time= 0.30183 error= 2.21327e-11
```

Here we have the output from the set of runs where our input came from lines in a file.

Ask for GPU nodes

--partition=gpu-h100 --gres=gpu:h100:4

```
[tkaiser2@kl1 slurm25]$cat dogpu
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SUBATCH --partition=gpu-h100
#SUBATCH --gres=gpu:h100:4
```

```
# What node(s) are we on?
echo $SLURM_NODELIST
```

```
# What partition
echo $SLURM_JOB_PARTITION
```

```
# GPU information
nvidia-smi -L
```

```
[tkaiser2@kl1 slurm25]$cat slurm-7583410.out
x3100c0s21b0n0
gpu-h100
GPU 0: NVIDIA H100 80GB HBM3 (UUID: GPU-96224b76-119e-b3d5-bcf1-de1c07aaac45)
GPU 1: NVIDIA H100 80GB HBM3 (UUID: GPU-e6b0d645-48b9-0341-0b87-b8ae88ac0761)
GPU 2: NVIDIA H100 80GB HBM3 (UUID: GPU-c1fb2da2-5567-abdb-1b1b-039ef0062f7b)
GPU 3: NVIDIA H100 80GB HBM3 (UUID: GPU-e5172eda-9fd6-954e-0d21-31b1afb358f1)
[tkaiser2@kl1 slurm25]$
```

```
[tkaiser2@kl1 slurm25]$cat nogres
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH --partition=gpu-h100
##### #SBATCH --gres=gpu:h100:4 #####
```

```
# What node(s) are we on?
echo $SLURM_NODELIST
```

```
# What partition
echo $SLURM_JOB_PARTITION
```

```
# GPU information
nvidia-smi -L
```

```
[tkaiser2@kl1 slurm25]$cat slurm-7583402.out
x3100c0s13b0n0
gpu-h100
No devices found.
[tkaiser2@kl1 slurm25]$
```

If you land on a node that has GPUs but don't specify the --gres option you will not see the GPUs.

As of 06/09/22 on Vermilion gres is not currently required/supported

Ask for GPU nodes

--gres=gpu:h100:4

```
[tkaiser2@kl1 slurm25]$cat dogpu
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH --partition=gpu-h100
#SBATCH --gres=gpu:h100:4
```

```
# What node(s) are we on?
echo $SLURM_NODELIST
```

```
# What partition
echo $SLURM_JOB_PARTITION
```

```
# GPU information
nvidia-smi -L
```

```
[tkaiser2@kl1 slurm25]$cat slurm-7583410.out
x3100c0s21b0n0
gpu-h100
GPU 0: NVIDIA H100 80GB HBM3 (UUID: GPU-96224b76-119e-b3d5-bcf1-de1c07aaac45)
GPU 1: NVIDIA H100 80GB HBM3 (UUID: GPU-e6b0d645-48b9-0341-0b87-b8ae88ac0761)
GPU 2: NVIDIA H100 80GB HBM3 (UUID: GPU-c1fb2da2-5567-abdb-1b1b-039ef0062f7b)
GPU 3: NVIDIA H100 80GB HBM3 (UUID: GPU-e5172eda-9fd6-954e-0d21-31b1afb358f1)
[tkaiser2@kl1 slurm25]$
```

```
[tkaiser2@kl1 slurm25]$cat nogres
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=00:02:00
#SBATCH --partition=gpu-h100
##### #SBATCH --gres=gpu:h100:4 #####
```

```
# What node(s) are we on?
echo $SLURM_NODELIST
```

```
# What partition
echo $SLURM_JOB_PARTITION
```

```
# GPU information
nvidia-smi -L
```

```
[tkaiser2@kl1 slurm25]$cat slurm-7583402.out
x3100c0s13b0n0
gpu-h100
No devices found.
[tkaiser2@kl1 slurm25]$
```

If you land on a node that has GPUs but don't specify the --gres command you will not see the GPUs.

As of 06/09/22 on Vermilion gres is not currently required/supported

GPU Example

```
#!/bin/bash
#SBATCH --job-name="gpu_job"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:02:00
#SBATCH --partition=gpu-h100
#SBATCH --gres=gpu:h100:4
#SBATCH --exclusive=user
#SBATCH --account=hpcapps

# Check we are launching from a GPU node

if echo $SLURM_SUBMIT_HOST | egrep "k15|k16" > /dev/null ; then : ; else echo Run script from a GPU node; echo exit ; fi

mkdir $SLURM_JOBID
cp mstream.cu makefile doit dogpu $SLURM_JOBID
cd $SLURM_JOBID

nvidia-smi -L > gpuinfo

# Run a mpi/cuda version of the stream benchmark
ml PrgEnv-nvhpc
CC mstream.cu -o mstream
srun -n 4 ./mstream -n 5000 > mstream.out

mv slurm-$SLURM_JOBID.out .
```

Two additions:

- Check we are launching from a GPU node
- Run a “mpi/cuda” version of the stream benchmark
- Our "doit" script will login into a gpu node and launches from there

GPU Example

```
template <typename T>
__global__ void STREAM_Copy(T const * __restrict__ const a, T * __restrict__ const b, long len)
{
    long idx = (long)threadIdx.x + (long)blockIdx.x * (long)blockDim.x;
    if (idx < len)
        b[idx] = a[idx];
}
```

```
template <typename T>
__global__ void STREAM_Scale(T const * __restrict__ const a, T * __restrict__ const b, T scale, long len)
{
    long idx = (long)threadIdx.x + (long)blockIdx.x * (long)blockDim.x;
    if (idx < len)
        b[idx] = scale * a[idx];
}
```

```
template <typename T>
__global__ void STREAM_Add(T const * __restrict__ const a, T const * __restrict__ const b, T * __restrict__ const c, long len)
{
    long idx = (long)threadIdx.x + (long)blockIdx.x * (long)blockDim.x;
    if (idx < len)
        c[idx] = a[idx] + b[idx];
}
```

```
template <typename T>
__global__ void STREAM_Triad(T const * __restrict__ a, T const * __restrict__ b, T * __restrict__ const c, T scalar, long len)
{
    long idx = (long)threadIdx.x + (long)blockIdx.x * (long)blockDim.x;
    if (idx < len)
        c[idx] = a[idx] + scalar * b[idx];
}
```


Output

```
[tkaiser2@kl1 7764252]$ls
dogpu doit gpuinfo makefile mstream mstream.cu mstream.out strm.0000 strm.0001 strm.0002 strm.0003
```

```
[tkaiser2@kl1 7764252]$cat strm.0000
STREAM Benchmark implementation in CUDA on device 0 of x3103c0s9b0n0
Device name: NVIDIA H100 80GB HBM3
Array elements 5000 Array size (double precision) = 0.04 MB
Total memory for 3 arrays = 0.00 GB
NTIMES 200000
using 192 threads per block, 27 blocks
output in IEC units (KiB = 1024 B)
```

Function	Rate (GiB/s)	Avg time(s)	Min time(s)	Max time(s)
Copy:	10.7759	0.00000736	0.00000691	0.00014782
Scale:	10.7759	0.00000734	0.00000691	0.00015306
Add:	16.1638	0.00000734	0.00000691	0.00004506
Triad:	16.1638	0.00000737	0.00000691	0.00005603

```
Total time 5.91 seconds
STREAM Benchmark implementation in CUDA on device 1 of x3103c0s9b0n0
Device name: NVIDIA H100 80GB HBM3
Array elements 5000 Array size (double precision) = 0.04 MB
Total memory for 3 arrays = 0.00 GB
NTIMES 200000
using 192 threads per block, 27 blocks
output in IEC units (KiB = 1024 B)
```

We run 4 MPI tasks/per
node and
each task “hits” one GPU

```
[tkaiser2@kl1 7764252]$cat strm.0003
STREAM Benchmark implementation in CUDA on device 3 of x3103c0s9b0n0
Device name: NVIDIA H100 80GB HBM3
Array elements 5000 Array size (double precision) = 0.04 MB
Total memory for 3 arrays = 0.00 GB
NTIMES 200000
using 192 threads per block, 27 blocks
output in IEC units (KiB = 1024 B)
```

Function	Rate (GiB/s)	Avg time(s)	Min time(s)	Max time(s)
Copy:	12.5000	0.00000716	0.00000596	0.00003409
Scale:	12.5000	0.00000713	0.00000596	0.00003409
Add:	18.7500	0.00000714	0.00000596	0.00003815
Triad:	18.7500	0.00000718	0.00000596	0.00005794

```
Total time 5.75 seconds
[tkaiser2@kl1 7764252]$
```

Shared partition

```
[tkaiser2@kl1 slurm25]$salloc --nodes=1 --time=01:00 --account=hpcapps --partition=shared --tasks-per-node=4
```

```
...  
...
```

```
[tkaiser2@x1008c0s0b1n1 slurm25]$printenv | grep SLURM | grep "NODE="
```

```
SLURM_TASKS_PER_NODE=4  
SLURM_STEP_TASKS_PER_NODE=1  
SLURM_NTASKS_PER_NODE=4  
SLURM_JOB_CPUS_PER_NODE=4  
SLURM_CPUS_ON_NODE=4
```

```
[tkaiser2@kl1 slurm25]$salloc --nodes=1 --time=01:00 --account=hpcapps --partition=shared --tasks-per-node=4 --cpus-per-task=2
```

```
...  
...
```

```
[tkaiser2@x1008c0s0b1n1 slurm25]$printenv | grep SLURM | grep "NODE="
```

```
SLURM_TASKS_PER_NODE=4  
SLURM_STEP_TASKS_PER_NODE=1  
SLURM_NTASKS_PER_NODE=4  
SLURM_JOB_CPUS_PER_NODE=8  
SLURM_CPUS_ON_NODE=8
```

You only get “charged” for what you use.

Shared partition

```
[tkaiser2@kl1 slurm25]$salloc --nodes=1 --time=01:00:00 --account=hpcapps --partition=shared  
--tasks-per-node=4
```

```
..  
..  
[tkaiser2@x1008c0s0b1n1 slurm25]$/home/tkaiser2/hog  
1 :: 0.1630130 0.1630130 6.587  
2 :: 0.3267691 0.1637561 6.557  
3 :: 0.4938800 0.1671109 6.425  
Killed
```

```
salloc --nodes=1 --time=01:00:00 --account=hpcapps --partition=shared  
--tasks-per-node=4 --cpus-per-task=2
```

```
..  
..  
[tkaiser2@x1008c0s0b1n1 slurm25]$/home/tkaiser2/hog  
1 :: 0.1458972 0.1458972 7.360  
2 :: 0.2909060 0.1450088 7.405  
3 :: 0.4365361 0.1456301 7.373  
4 :: 0.5813200 0.1447840 7.416  
5 :: 0.7271311 0.1458111 7.364  
6 :: 0.8707340 0.1436028 7.477  
7 :: 1.0156901 0.1449561 7.407  
Killed
```

```
[tkaiser2@kl1 slurm25]$salloc --nodes=1 --time=01:00:00 --account=hpcapps --partition=shared  
--tasks-per-node=4 --cpus-per-task=2 --mem=16000
```

```
..  
..  
[tkaiser2@x1008c0s0b1n1 slurm25]$/home/tkaiser2/hog  
1 :: 0.1295950 0.1295950 8.285  
2 :: 0.2586551 0.1290600 8.320  
3 :: 0.3880930 0.1294379 8.295  
4 :: 0.5172129 0.1291199 8.316  
5 :: 0.6463461 0.1291332 8.315  
6 :: 0.7754819 0.1291358 8.315  
7 :: 0.9048121 0.1293302 8.302  
8 :: 1.0338349 0.1290228 8.322  
9 :: 1.1628139 0.1289790 8.325  
10 :: 1.2923260 0.1295121 8.291  
11 :: 1.4220200 0.1296940 8.279  
12 :: 1.5522740 0.1302540 8.243  
13 :: 1.6825619 0.1302879 8.241  
14 :: 1.8127539 0.1301920 8.247  
15 :: 1.9497881 0.1370342 7.836  
Killed
```

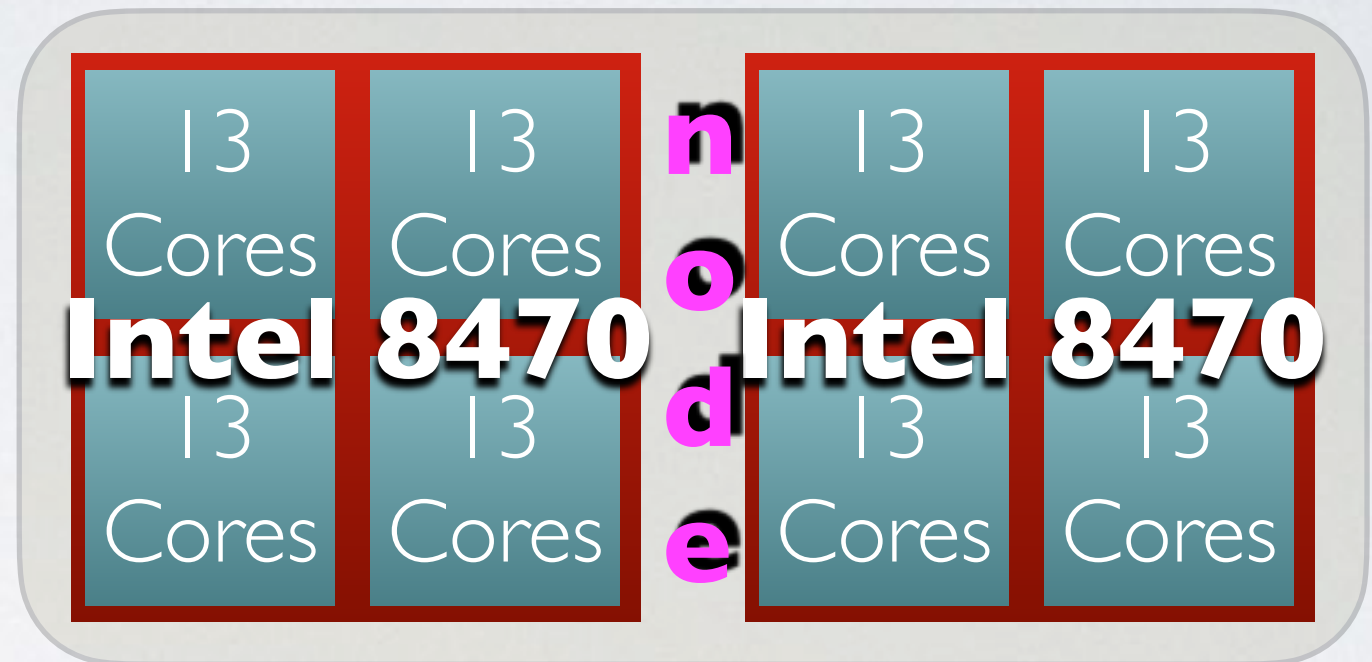
In shared, by default,
you get 1GB of
memory for each
core requested

We're running a code that just
keeps asking for more memory!

(We're trained experts. Don't
try this at home)

AFFINITY & WHY IMPORTANT

- Affinity - mapping of threads/tasks to cores
- Kestrel 104 cores/node
 - 2 chips (Intel 8470)
 - 52 cores each
 - 4 "tiles" with 13 cores each



- **Worst case: Multiple threads/tasks can end up on the same core potentially reducing performance by 2X or maybe much more**
- Also: You may want to put threads/tasks on particular tiles to maximize communications or memory access
- Possible to have different MPI tasks to have different # threads

0-12	26-38
13-25	39-51

52-64	78-90
65-77	91-103

Affinity

- Mapping of tasks and threads to cores
- If you have two or more tasks or threads on the same core performance can tank.
- Our example code, pstream.c is designed to show mappings of tasks/threads to cores
- Mappings are effected by environmental variables and command line arguments
- Desired mappings can be application specific
- Default mappings might not be very good

Our script for affinity testing

```
#!/usr/bin/bash
#SBATCH --job-name="affinity"
#SBATCH --nodes=1
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=00:10:00
#SBATCH --partition=short

# Select MPI version based on the variable MYMPI, default to intel-oneapi-mpi
if [ -z "${MYMPI+x}" ]; then export MYMPI=intel-oneapi-mpi ; fi

# Setup
export BASE=$SLURM_SUBMIT_DIR
mkdir -p $BASE/$SLURM_JOB_ID
cd $BASE/$SLURM_JOB_ID

# Save info
cat $0 > $SLURM_JOB_ID.script
printenv > $SLURM_JOB_ID.env

# Get and build glorified "Hello World"
curl -s https://raw.githubusercontent.com/timkphd/examples/master/hybrid/pstream.c -o pstream.c
module load $MYMPI
echo MYMPI $MYMPI
echo MPI= `which mpicc`
mpicc -fopenmp pstream.c -o pstream

for nmpi in 1 2 4 8 13 26 52 104 ; do
  for nthreads in 2 4 8 13 ; do
    cores=`echo "$nmpi*$nthreads" | bc`
    if [ "$cores" -le "104" ] ; then
      echo Running on a total of $cores cores
      export OMP_NUM_THREADS=$nthreads
      export OMP_PROC_BIND=spread
      # The last column of output from pstream is the core on which a task/thread is run


      srun --cpu-bind=v --threads-per-core=1 --cpus-per-task=$nthreads --ntasks $nmpi ./pstream -F -D -t 2 > ${nmpi}_${nthreads}
# If we don't use the extra settings some tasks/threads will get mapped to the same cores and you'll see FAILED
      #srun --cpu-bind=v --ntasks $nmpi ./pstream -F -D -t 2 > ${nmpi}_${nthreads}
      # We grab the core #s. There should be unique set and the size should be equal to $cores
      used=`cat ${nmpi}_${nthreads} | grep ^0 | awk '{print $6}' | sort -u | wc -l`
      if [ "$cores" -eq "$used" ] ; then
        echo SUCCESS ${nmpi}_${nthreads}
      else
        echo FAILED ${nmpi}_${nthreads}
      fi
    fi
  done
done
cp $SLURM_SUBMIT_DIR/slurm-$SLURM_JOB_ID.* .
```

On our run line we add:

--cpu-bind=v --threads-per-core=1 --cpus-per-task=\$nthreads
--cpu-bind=v produces a “verbose” output mappings

Output from one of the runs

Core on which a
task/thread is running



```
[tkaiser2@kl1 7772992]$cat 2_4
```

```
MPI VERSION Intel(R) MPI Library 2021.12 for Linux* OS
```

task	thread	node name	first task	# on node	core
0001	0000	x1001c0s3b0n1	0000	0001	0052
0001	0003	x1001c0s3b0n1	0000	0001	0055
0001	0002	x1001c0s3b0n1	0000	0001	0054
0001	0001	x1001c0s3b0n1	0000	0001	0053
0000	0000	x1001c0s3b0n1	0000	0000	0000
0000	0001	x1001c0s3b0n1	0000	0000	0001
0000	0003	x1001c0s3b0n1	0000	0000	0003
0000	0002	x1001c0s3b0n1	0000	0000	0002

```
total time      3.545
```

```
[tkaiser2@kl1 7772992]$
```

Output from a “broken” run

```
MPI VERSION MPI VERSION      : CRAY MPICH version 8.1.28.15 (ANL base 3.4a2)
MPI BUILD INFO : Wed Nov 15 21:00 2023 (git hash 1cde46f)
```

task	thread	node name	first task	# on node	core
0000	0000	x1003c0s0b1n0	0000	0000	0000
0001	0000	x1003c0s0b1n0	0000	0001	0000
0002	0000	x1003c0s0b1n0	0000	0002	0000
0003	0000	x1003c0s0b1n0	0000	0003	0000
0000	0001	x1003c0s0b1n0	0000	0000	0052
0001	0001	x1003c0s0b1n0	0000	0001	0052
0002	0001	x1003c0s0b1n0	0000	0002	0052
0003	0001	x1003c0s0b1n0	0000	0003	0052
total time		2.928			

All tasks/threads end up on core 0 or 52.

Results

--threads-per-core=1

--cpus-per-task=\$nthreads

MPI tasks _ Threads	With extra options	Time (sec)
1_2	SUCCESS	5.26
1_4	SUCCESS	3.54
1_8	SUCCESS	2.72
1_13	SUCCESS	2.31
2_2	SUCCESS	5.26
2_4	SUCCESS	3.55
2_8	SUCCESS	2.73
2_13	SUCCESS	2.31
4_2	SUCCESS	5.26
4_4	SUCCESS	3.55
4_8	SUCCESS	2.74
4_13	SUCCESS	2.38
8_2	SUCCESS	5.27
8_4	SUCCESS	3.57
8_8	SUCCESS	2.96
8_13	SUCCESS	2.55
13_2	SUCCESS	5.29
13_4	SUCCESS	3.79
13_8	SUCCESS	2.94
26_2	SUCCESS	5.46
26_4	SUCCESS	3.44
52_2	SUCCESS	4.57

MPI tasks _ Threads	Without extra options	Time (sec)	Slowdown
1_2	SUCCESS	5.25	1.0
1_4	SUCCESS	3.55	1.0
1_8	SUCCESS	2.73	1.0
1_13	SUCCESS	2.30	1.0
2_2	SUCCESS	5.26	1.0
2_4	SUCCESS	3.55	1.0
2_8	SUCCESS	2.73	1.0
2_13	SUCCESS	2.31	1.0
4_2	FAILED	14.22	2.7
4_4	FAILED	7.35	2.1
4_8	FAILED	5.69	2.1
4_13	FAILED	3.88	1.6
8_2	FAILED	28.23	5.4
8_4	FAILED	14.56	4.1
8_8	FAILED	7.66	2.6
8_13	FAILED	5.10	2.0
13_2	FAILED	46.06	8.7
13_4	FAILED	23.77	6.3
13_8	FAILED	12.42	4.2
26_2	FAILED	91.56	16.8
26_4	FAILED	47.72	13.9
52_2	FAILED	183.78	40.3

Affinity Settings

- srun options that effect affinity
 - https://slurm.schedmd.com/mc_support.html#srun_lowlevelmc
 - --cpu-bind
 - --cpus-per-task
 - --distribution
 - --ntasks-per-core
 - --ntasks-per-socket
 - --threads-per-core
- Openmp variables that effect affinity
 - OMP_NUM_THREADS
 - OMP_PROC_BIND
 - OMP_PLACES

What's a Reservation

- Nodes can be set aside, temporarily, for use for a particular purpose or set of people
- Only "special" people can assess these nodes
 - `--reservation=rrrr`
 - `export SBATCH_RESERVATION=rrrr`
- Can still access other nodes/partitions
- Sometimes used to "drain" nodes.

man pages

- All of the commands from today have man pages
 - man sbatch
 - man squeue
 - man srun
 - man sinfo
 - man scancel
 - man sacctmgr

<https://slurm.schedmd.com/quickstart.html>

Some Links

- <https://www.nrel.gov/hpc/>
 - NREL's HPC docs front page
- <https://github.com/NREL/HPC/tree/master/slurm>
 - My slurm script examples (old but still might be useful)
- https://nrel.github.io/HPC/Documentation/Slurm/batch_jobs/
 - Local slum info with script tutorial
- <https://nrel.github.io/HPC/Documentation/Systems/>
 - Systems page
- <https://slurm.schedmd.com/quickstart.html>
 - Slurm Docs
- <https://nrel.github.io/HPC/Documentation/Systems/Kestrel/Environments/gpubuildandrun/>
 - Building and running go Kestrel's GPU nodes
- <https://github.com/timkphd/examples/tree/master/workshop/2025/slurm>
 - Public copy of examples

Different # tasks on each node

```
[tkaiser2@x1004c6s0b0n1 mpmd]$cat ./mlist.py
#!/usr/bin/env python3
# usage
# scontrol show hostnames | ./mlist.py 4 2 6
import sys
node_list = sys.stdin.read()
node_list=node_list.split()
k=0
nnodes=len(node_list)
for n in sys.argv[1:]:
    for j in range(0,int(n)):
        print(node_list[k % nnodes])
        k=k+1
```

```
[tkaiser2@x1004c6s0b0n1 mpmd]$scontrol show hostnames | ./mlist.py 4 2 6 4 > hlist
[tkaiser2@x1004c6s0b0n1 mpmd]$cat hlist
```

```
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s4b1n0
x1004c6s4b1n0
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s0b0n1
x1004c6s4b1n0
x1004c6s4b1n0
x1004c6s4b1n0
x1004c6s4b1n0
```

1. Create a node list file "hlist" containing a node for each task
2. Export SLURM_HOSTFILE to tell run to use it
3. srun as normal

```
[tkaiser2@x1004c6s0b0n1 mpmd]$tasks=`wc -l hlist | cut -f 1 -d" "`
[tkaiser2@x1004c6s0b0n1 mpmd]$echo $tasks
16
```

```
[tkaiser2@x1004c6s0b0n1 mpmd]$export SLURM_HOSTFILE=hlist
```

```
[tkaiser2@x1004c6s0b0n1 mpmd]$srun -n $tasks ./helloc > output
```

```
[tkaiser2@x1004c6s0b0n1 mpmd]$grep Hello output | sort -nk5,5
```

```
Hello from x1004c6s0b0n1 (C) 0 of 16
Hello from x1004c6s0b0n1 (C) 1 of 16
Hello from x1004c6s0b0n1 (C) 2 of 16
Hello from x1004c6s0b0n1 (C) 3 of 16
Hello from x1004c6s4b1n0 (C) 4 of 16
Hello from x1004c6s4b1n0 (C) 5 of 16
Hello from x1004c6s0b0n1 (C) 6 of 16
Hello from x1004c6s0b0n1 (C) 7 of 16
Hello from x1004c6s0b0n1 (C) 8 of 16
Hello from x1004c6s0b0n1 (C) 9 of 16
Hello from x1004c6s0b0n1 (C) 10 of 16
Hello from x1004c6s0b0n1 (C) 11 of 16
Hello from x1004c6s4b1n0 (C) 12 of 16
Hello from x1004c6s4b1n0 (C) 13 of 16
Hello from x1004c6s4b1n0 (C) 14 of 16
Hello from x1004c6s4b1n0 (C) 15 of 16
```

```
[tkaiser2@x1004c6s0b0n1 mpmd]$
```


MPMD - different apps for various tasks

```
[tkaiser2@x1001c1s0b1n0 mpmd]$scontrol show hostnames | ./mlist.py 4 2 6 4 > hlist
[tkaiser2@x1001c1s0b1n0 mpmd]$ export SLURM_HOSTFILE=hlist
```

```
[tkaiser2@x1001c1s0b1n0 mpmd]$cat apps
0-5 ./helloF
6-8 ./helloc
9 ./helloF
10 ./helloc
11 ./helloF
12-15 ./helloc
```

```
[tkaiser2@x1001c1s0b1n0 mpmd]$srun -n 16 --multi-prog apps
```

Sorted output

Hello from x1001c1s0b1n0 (F)	0	of	16
Hello from x1001c1s0b1n0 (F)	1	of	16
Hello from x1001c1s0b1n0 (F)	2	of	16
Hello from x1001c1s0b1n0 (F)	3	of	16
Hello from x1001c1s1b0n0 (F)	4	of	16
Hello from x1001c1s1b0n0 (F)	5	of	16
Hello from x1001c1s0b1n0 (C) 6 of 16			
Hello from x1001c1s0b1n0 (C) 7 of 16			
Hello from x1001c1s0b1n0 (C) 8 of 16			
Hello from x1001c1s0b1n0 (F)	9	of	16
Hello from x1001c1s0b1n0 (C) 10 of 16			
Hello from x1001c1s0b1n0 (F)	11	of	16
Hello from x1001c1s1b0n0 (C) 12 of 16			
Hello from x1001c1s1b0n0 (C) 13 of 16			
Hello from x1001c1s1b0n0 (C) 14 of 16			
Hello from x1001c1s1b0n0 (C) 15 of 16			

1. Here we also use a host list file but that is not required

2. Create an "apps" file that contains a numbered list of the app for each task

3. srun with **--multi-prog apps**

4. No arguments after this new option

5. If you need arguments for your app they go in the "apps" file

6. Broken for Cray MPICH



Slurm Basics

June 3, 2025

Timothy H. Kaiser, Ph.D.

tkaiser2@nrel.gov

Setup On Kestrel..

```
ssh kestrel.nrel.gov
```

```
cd /scratch/$USER
```

```
mkdir slurm25
```

```
cd slurm25
```

```
tar -xzf /scratch/tkaiser2/shared/slurm25/slurm25.tgz
```

```
export SLURM_ACCOUNT=YOUR_ACCOUNT
```

```
export SALLOC_ACCOUNT=$SLURM_ACCOUNT
```

```
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
```

```
./doall
```

To get the results without running...

```
tar -xzf /scratch/tkaiser2/shared/slurm25/run25.tgz
```

