

# HIGH PERFORMANCE RESEARCH COMPUTING

## ACES: Using the Slurm Scheduler on Composable Resources

HPRC Training  
February 13, 2024



High Performance  
Research Computing  
DIVISION OF RESEARCH



# Overview

- HPC Architecture
- Slurm SBATCH Parameters
- Single node jobs
  - single-core
  - multi-core
- Break
- Multi-node jobs
  - MPI jobs
  - TAMULauncher
- Monitoring job resource usage
  - at runtime
  - after job completion
  - job debugging
  - job details

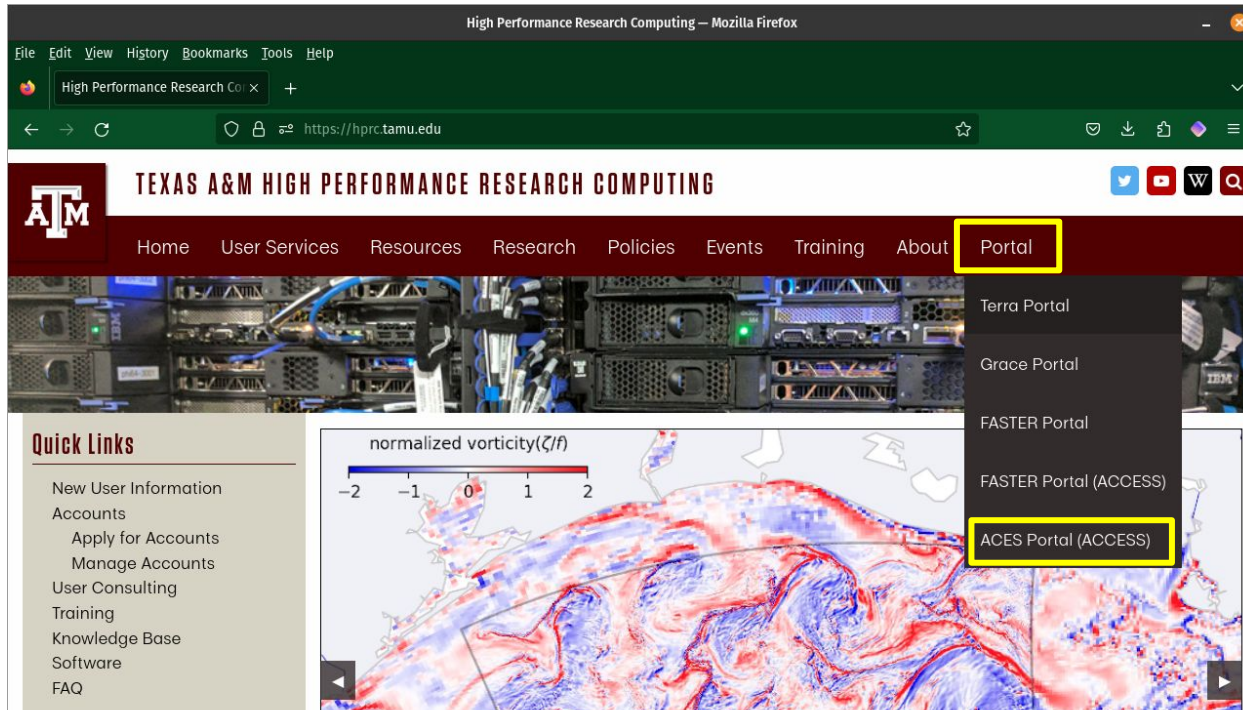
# ACES



- ACES is a Dell cluster with a rich accelerator testbed
  - Intel Max GPUs (PVC)
  - Intel FPGAs (Field Programmable Gate Arrays)
  - NVIDIA H100 and A30 GPUs
  - NEC Vector Engines
  - NextSilicon co-processors
  - Graphcore IPUs (Intelligence Processing Units).

<https://hprc.tamu.edu/kb/User-Guides/ACES>

# Accessing the HPRC ACES Portal

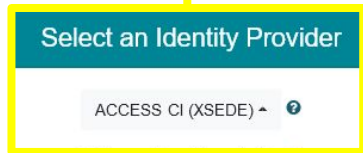
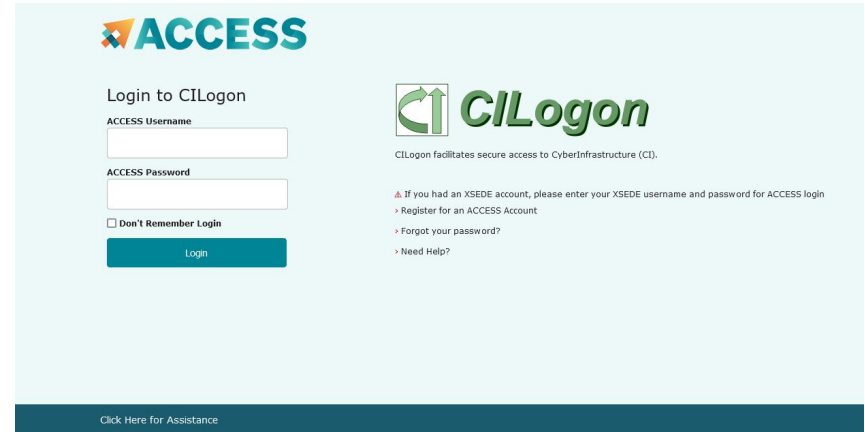
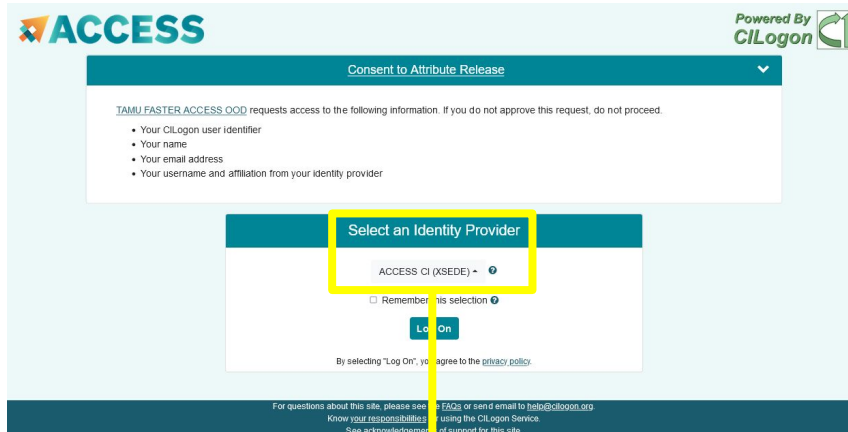


The screenshot shows a Mozilla Firefox browser window displaying the Texas A&M High Performance Research Computing (HPRC) website. The browser's address bar shows the URL <https://hprc.tamu.edu>. The website header includes the TAMU logo and the text "TEXAS A&M HIGH PERFORMANCE RESEARCH COMPUTING". A navigation menu contains links for Home, User Services, Resources, Research, Policies, Events, Training, About, and Portal. The "Portal" link is highlighted with a yellow box, and a dropdown menu is open, listing "Terra Portal", "Grace Portal", "FASTER Portal", "FASTER Portal (ACCESS)", and "ACES Portal (ACCESS)". The "ACES Portal (ACCESS)" option is also highlighted with a yellow box. Below the navigation menu, there is a "Quick Links" section with a list of links: New User Information, Accounts, Apply for Accounts, Manage Accounts, User Consulting, Training, Knowledge Base, Software, and FAQ. To the right of the Quick Links is a visualization of "normalized vorticity( $\zeta/f$ )" over a map of Texas, with a color scale ranging from -2 to 2.

HPRC webpage: [hprc.tamu.edu](https://hprc.tamu.edu)

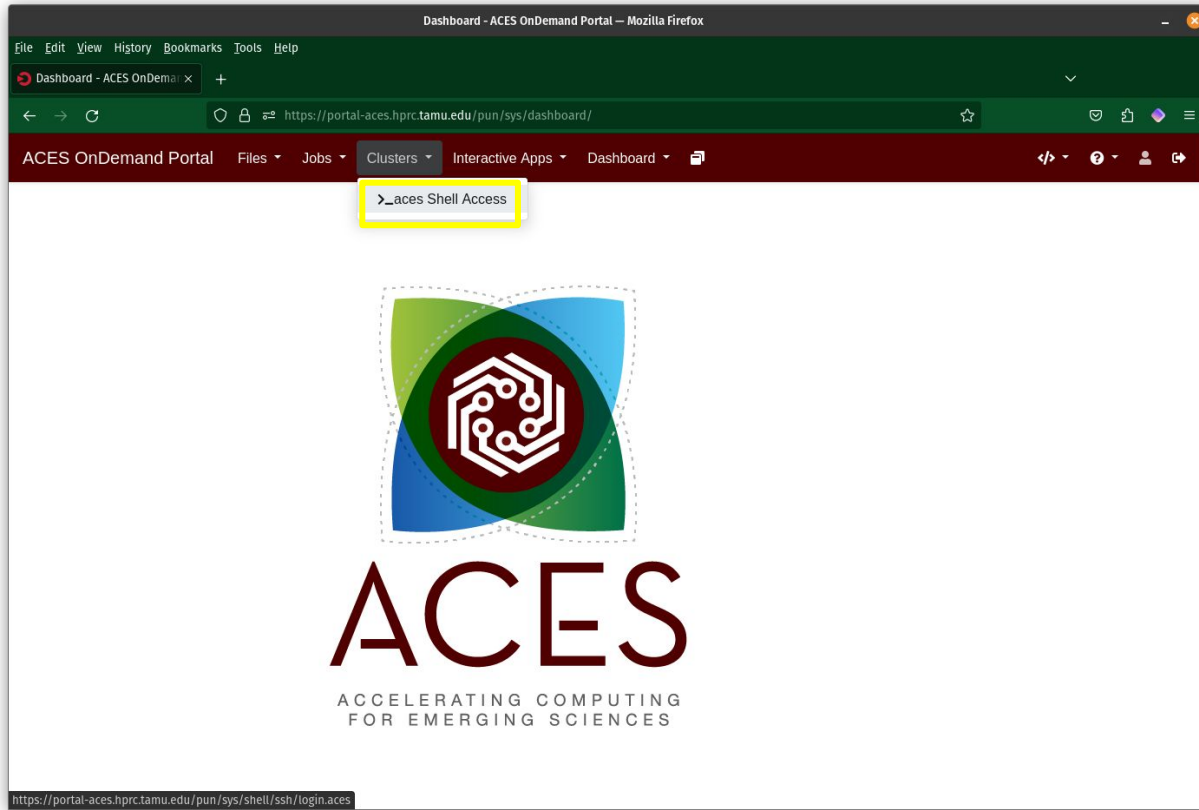
# Accessing ACES via the Portal (ACCESS)

Log-in using your ACCESS credentials.



Select the Identity Provider appropriate for your account.

# Shell Access via the Portal



# Hands-On Activity

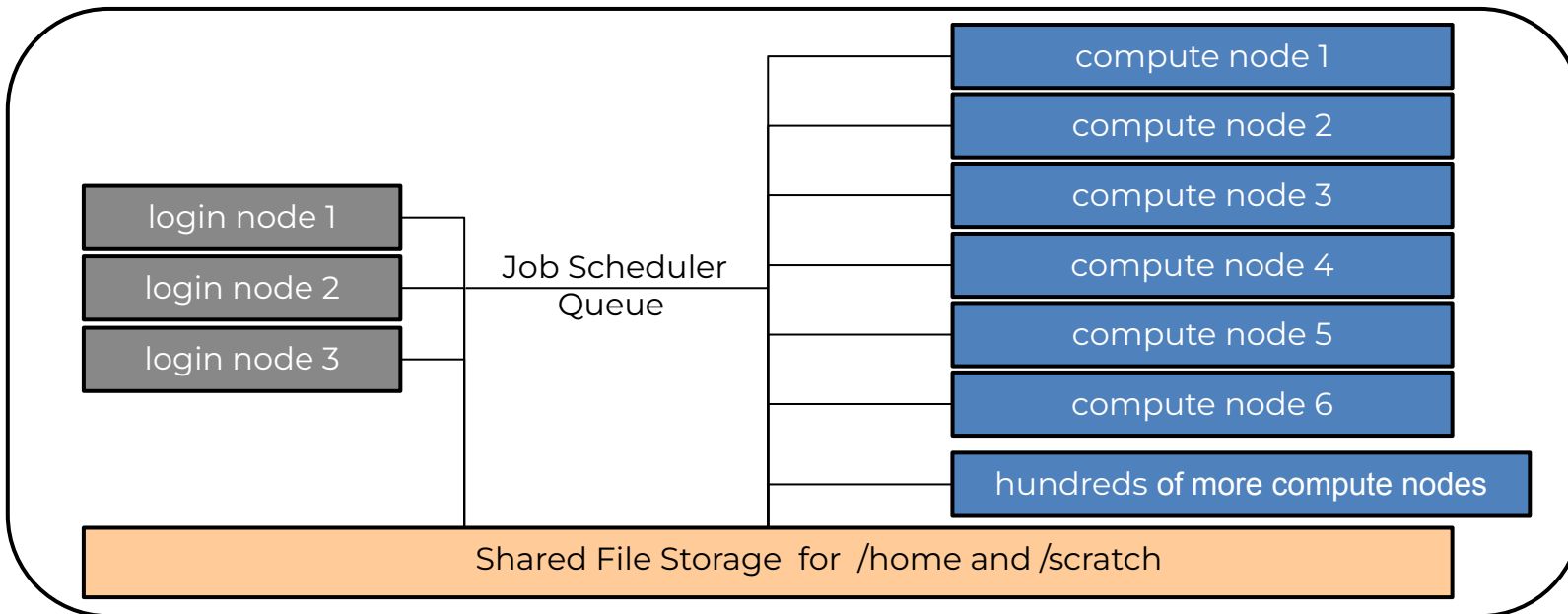
1. Login to the ACES portal.
2. Connect to the shell command line.
3. What message do you see when connecting to the shell command line?
4. On which login node did you land?

# Nodes, Cores and Queues

- **Node**
  - one computer unit of an HPC cluster each containing memory and one or more CPUs. There are generally two classifications of HPC nodes available to users; login and compute.
    - login node
      - This is where users first login to stage their job scripts and do file and directory manipulations with text file editors (vi, gedit, emacs, portal) and Unix commands.
    - compute node
      - These are often referred to as just nodes since jobs are only scheduled on the compute nodes.
      - Some compute nodes contain GPUs or other accelerators.
      - One or more compute nodes can be used for a job, based on how you configure your job script and whether the software supports running on multiple nodes.
- **Core**
  - There are 96 cores (CPUs) on the ACES 512GB memory compute nodes (488GB available).
  - You can use one or all 96 cores in a job script.
    - Check to see that the software you use in your job script supports multi-core usage.
- **Queue**
  - Ordered list of all scheduled jobs of all users both in the PENDING and RUNNING states
  - The queue is made up of multiple partitions (Example partition names: gpu, cpu).
  - The cpu partition is auto-assigned, but you must request gpu and other partitions.



# HPC Diagram



## login nodes are for:

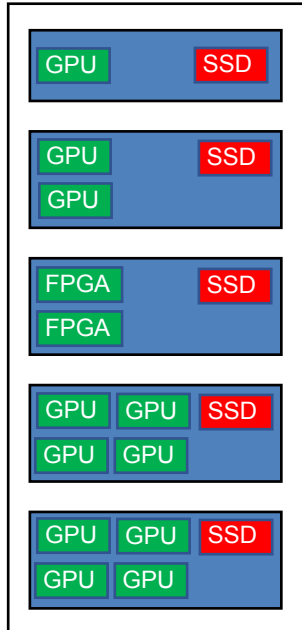
- file manipulation and job script preparation
- software installation and testing
- short tasks (< 60 minutes and max 8 cores)
  - also be aware of amount of memory utilized

## compute nodes are for:

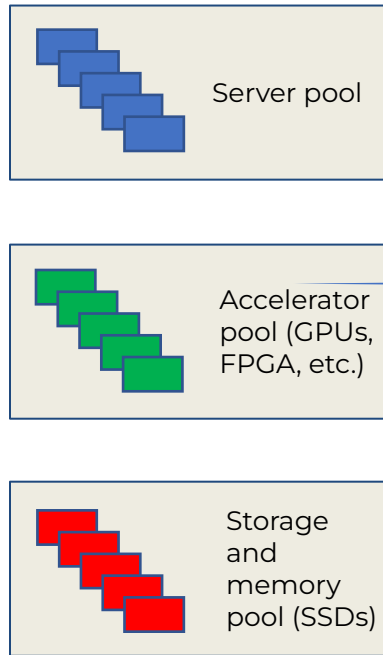
- computational jobs which can use up to 96 cores and/or up to 488GB memory per ACES compute node.
- all jobs running > 60 minutes

# Hardware Composability (Compute Nodes)

## Traditional Server Configuration

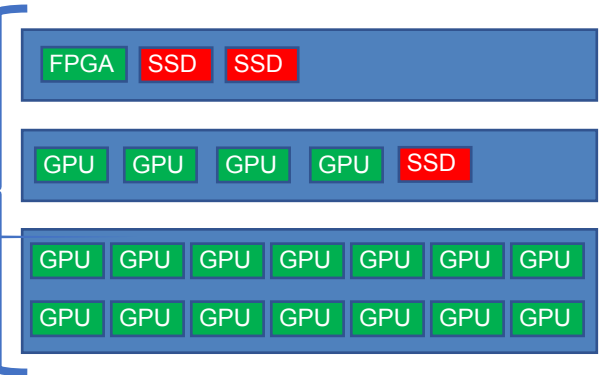


## Composable Resources



Liquid Fabric

## Composable Server Configuration (can be recomposed)



currently composing is performed by system admins at the user's request, but user composability is under development

# Slurm SBATCH Parameters

# Slurm Job Script Example

```
#!/bin/bash
#SBATCH --job-name=spades           # keep job name short with no spaces
#SBATCH --time=1-00:00:00          # request 1 day; Format: days-hours:minutes:seconds
#SBATCH --nodes=1                  # request 1 node
#SBATCH --ntasks-per-node=1        # request 1 task (command) per node
#SBATCH --cpus-per-task=1          # request 1 cpu (core, thread) per task
#SBATCH --mem=5G                    # request 5GB total memory per node
#SBATCH --output=stdout.%x.%j      # save stdout to a file with job name and JobID appended to file name
#SBATCH --error=stderr.%x.%j      # save stderr to a file with job name and JobID appended to file name

# unload any modules to start with a clean environment
module purge
# load software modules
module load GCC/11.3.0  SPAdes/3.15.5
# run commands
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```

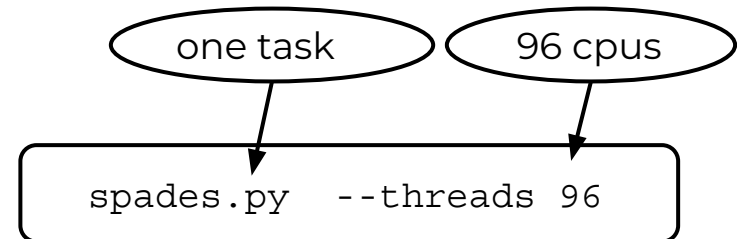
- Always include the first line exactly as it is; no trailing spaces or comments.
- Slurm job parameters begin with **#SBATCH**, and you can add comments afterwards as above.
- Name the job script whatever you like, but be consistent to make it easier to search for job scripts
  - **my\_job\_script.job**
  - **my\_job\_script.sbatch**
  - **run\_program\_project.sh**
  - **job\_program\_project.slurm**

# Commonly Used Slurm SBATCH Parameters

- **--nodes**
  - number of nodes to use where a node is one computer unit of many in an HPC cluster
    - **--nodes=1** # request 1 node
  - used for multi-node jobs
    - **--nodes=10**
- if number of cpus per node is not specified then defaults to 1 cpu
- default is 1 node if --nodes not used & can be used with --ntasks-per-node and --cpus-per-task
- **--ntasks**

either --ntasks, --ntasks-per-node or --nodes needs to be provided.

  - a task can be considered a command such as blastn, bwa, script.py, etc.
  - **--ntasks=1** # total tasks across all nodes where each task is scheduled a max of 1 cpu
- when using --ntasks > 1 without --nodes=1, the job might be scheduled on multiple compute nodes
- **--ntasks-per-node**
- use together with --cpus-per-task
  - **--ntasks-per-node=1**
- **--cpus-per-task**
  - number of CPUs (cores) for each task (command)
  - **--cpus-per-task=96**



# Commonly Used Slurm SBATCH Parameters

- **--time**
  - max runtime for job (*required*); format: days-hours:minutes:seconds (days- is optional)
  - `--time=24:00:00` # set max runtime 24 hours (same as `--time=1-00:00:00`)
  - `--time=7-00:00:00` # set max runtime 7 days
- **--mem**
  - total memory for each node (*required*)
  - `--mem=488G` # request 488GB total memory (max available for 512gb nodes)
- **--job-name**
  - set the job name, keep it short and concise without spaces (*optional but highly recommended*)
  - `--job-name=myjob`
- **--output**
  - save all stdout to a specified file (*optional but highly recommended for debugging*)
  - `--output=stdout.%x.%j` # saves stdout to a file named `stdout.jobname.JobID`
- **--error**
  - save all stderr to a specified file (*optional but highly recommended for debugging*)
  - `--error=stderr.%x.%j` # saves stderr to a file named `stderr.jobname.JobID`
  - use just `--output` to save stdout and stderr to the same output file: `--output=output.%x.%j.log`
- **--partition**
  - specify a partition (queue) to use (*optional, use as needed*)
  - partition is automatically assigned to `cpu` so you don't need `--partition` unless you want to use accelerators.
    - need to specify `--partition` parameter to use `gpu`, `bittware`, `memverge`

# Commonly Used Optional Slurm Parameters

- **--gres**
  - used for requesting 1 or more GPUs; use GPU type in lowercase
  - use **gpuavail** command to see number of GPUs per compute node
  - **--gres=gpu:h100:1** # request 1 H100 GPU; use replace :1 with :2 for two GPUs, etc
  - **--partition=gpu** # also include this line when requesting GPUs
- **--account**
  - specify which account to use; use **myproject** to see your accounts
  - **--account=ACCOUNTNUMBER**
  - default account from **myproject** output is used if not specified
- **--mail-user**
  - **--mail-user=myemail@myuniversity.edu**
- **--mail-type**
  - send email per job event: BEGIN, END, FAIL, ALL
  - **--mail-type=ALL**
- **--dependency**
  - schedule a job to start after a previous job successfully completes
  - **--dependency=afterok:JobID**
    - get the JobID of the previous job with **squeue --me**

# Submitting Slurm Jobs

- A job script is a text file of Unix commands with **#SBATCH** parameters.
- **#SBATCH** parameters provide resource configuration request values.
  - time, memory, nodes, cpus, output files, ...
- Jobs can be submitted using a job script or directly on the command line.
  - start time depends on available resources
- Submit the job using sbatch command with the job script name.
  - Your job script provides a record of commands used for an analysis.
  - `sbatch my_job_script.job`
- Submit command on the command line by specifying all necessary parameters.
  - `sbatch -t 01:00:00 -n 1 -J myjob --mem 5G -o stdout.%j commands.sh`
- You can start an interactive job on the command line using the **srun** command instead of **sbatch**. Your srun job ends when you exit the terminal.
  - Do not to use more than the requested memory and CPUs when your **srun** job starts.
  - `srun --time=04:00:00 --mem=5G --ntasks=1 --cpus-per-task=1 --pty bash`

[slurm.schedmd.com/sbatch.html](http://slurm.schedmd.com/sbatch.html)



# How Busy is the ACES Cluster?

Check on the command line of a cluster using the sinfo command or view the main HPRC webpage

## sinfo

PARTITION	AVAIL	TIMELIMIT	JOB_SIZE	NODES (A/I/O/T)	CPUS (A/I/O/T)
cpu*	up	7-00:00:00	1-64	0/65/13/78	0/6240/1248/7488
gpu	up	2-00:00:00	1-15	13/2/2/17	1072/368/192/1632
pvc	up	2-00:00:00	1-15	0/0/20/20	0/0/1920/1920
bittware	up	2-00:00:00	1	0/0/2/2	0/0/192/192
d5005	up	2-00:00:00	1	0/2/0/2	0/192/0/192
memverge	up	2-00:00:00	1	0/5/3/8	0/480/288/768
nextsilicon	up	2-00:00:00	1	0/1/1/2	0/96/96/192
staff	up	2-00:00:00	1-110	13/77/20/110	1072/7568/1920/10560

### A/I/O/T

A = Active (in use by running jobs)  
I = Idle (available for jobs)  
O = Offline (unavailable for jobs)  
T = Total

not all partitions are available to users

## hprc.tamu.edu

### Cluster Status

#### ACES

Nodes 36/92 (39%)  
Cores 3194/8832 (36%)  
Jobs 17R-3Q

#### FASTER

Nodes 62/119 (52%)  
Cores 2713/7552 (36%)  
Jobs 61R-115Q

#### Grace

Nodes 562/679 (83%)  
Cores 22545/32792 (69%)  
Jobs 477R-17Q

#### Terra

Nodes 141/264 (53%)  
Cores 2435/7772 (31%)  
Jobs 238R-24Q

Historical Status

# Checking GPU Availability on ACES

- Use the command line (shell) to see the current GPU configuration and availability
- The GPU configuration can change since ACES is a composable resource cluster

```
gpuavail
```

```
CONFIGURATION
NODE          NODE
TYPE          COUNT
-----
gpu:pvc:4    19
gpu:h100:2   15
gpu:a30:4     1
gpu:pvc:2    1

AVAILABILITY
NODE   GPU   GPU   GPU   CPU   MEM
NAME   TYPE  COUNT AVAIL AVAIL AVAIL
-----
ac040  h100  2     2    96   488
ac046  h100  2     1    48   244
ac055  h100  2     2    96   488
ac064  a30   4     4    96   488
```

# Viewing Maximum Resources

The **maxconfig** command will show the recommended Slurm parameters for the maximum resources (cores, memory, time) per node for a specified accelerator or partition (default partition: cpu).

```
[username@aces ~]$ maxconfig

ACES partitions:  cpu  gpu  pvc  bittware  d5005  memverge  nextsilicon  gpu-hybrid
ACES GPUs in gpu partition:  a30:4  h100:2  h100:4  pvc:2  pvc:4

Showing max parameters (cores, mem, time) for partition cpu

#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --time=7-00:00:00
#SBATCH --nodes=1          # max 64 nodes for partition cpu
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

# Hands-on Activity: Exploring the maxconfig Command

Run the `maxconfig -h` command to answer the following questions.

---

1. What `maxconfig` option(s) can you use to see max resources for the H100 GPUs nodes?
  - a. What is the maximum number of H100 GPUs available per node?
  - b. What is the maximum runtime for an H100 GPU job?
2. What `maxconfig` option(s) can you use to see max resources for just one H100 GPU?
  - a. Why doesn't this option show all available memory and cores on the compute node?
3. What `maxconfig` option(s) can you use to see max resources for the bittware partition?

# Requesting all Cores and Memory on One Compute Node

## Example 1 (maxconfig)

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --time=7-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

## Example 2

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --time=7-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=96
#SBATCH --cpus-per-task=1
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

## Example 3

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --time=7-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

select configuration based on your mpi jobs

## Example 4

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --time=7-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

use --nodes if not  
using  
--ntasks\_per\_node

## Example 5

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --time=7-00:00:00
#SBATCH --ntasks=96
#SBATCH --ntasks_per_node=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

use  
--ntasks\_per\_node  
if not using  
--nodes

# Incorrect Resource Request Parameters for a single-node job

```
#!/bin/bash
#SBATCH --job-name=myjob
#SBATCH --time=1-00:00:00
#SSBATCH --ntasks=96
#SSBATCH --mem=24G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

- This job could be scheduled across multiple nodes since `--nodes` or `--ntasks_per_node` were not used with `--ntasks=96`.
- Request all the memory if requesting all the cores.
- Request all the cores if requesting all the memory.

# Single-Node Jobs

# Single vs Multi-Core Jobs

- When to use single-core jobs
  - The software being used only supports commands utilizing a single-core
- When to use multi-core jobs
  - If the software supports multiple cores (--threads, --cpus, ...), then configure the job script and software command options to utilize all CPUs on a compute node to get the job done faster, unless the software specifically recommends a limited number of cores.
    - ACES 512GB memory compute nodes
      - 96 CPUs (cores) per compute node
      - 488GB of available memory per compute node
  - Can group multiple single-core commands into a "multi-core" job using [TAMULauncher](#) on one or multiple nodes



# Slurm Parameter: --ntasks

```
#!/bin/bash
#SBATCH --job-name=myjob           # job name
#SBATCH --time=1:00:00            # set the wall clock limit to 1 hour
#SBATCH --ntasks=1                # request 1 task (command) per node
#SBATCH --mem=5G                  # request 3GB of memory per node
#SBATCH --output=stdout.%x.%j     # create a file for stdout
#SBATCH --error=stderr.%x.%j     # create a file for stderr
```

If you use `--ntasks=2` or more, the job could land on one node per task and your software may not support running on multiple nodes

- `--ntasks=1`
  - NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1
- `--ntasks=96` # without `-nodes=1` could result in multiple nodes
  - NumNodes=1 NumCPUs=96 NumTasks=96 CPUs/Task=1
  - NumNodes=96 NumCPUs=1 NumTasks=96 CPUs/Task=1

# Requesting one GPU on ACES

## Compute Nodes that Have Two or More GPUs

- Request less than the available CPU and memory resources for a compute node that has 2 or more installed GPUs when you need just 1 GPU so that someone else can use the other GPUs, unless you need more CPUs and memory resources for your job.
  - **maxconfig** will scale the cores and memory based on the fraction of GPUs selected
- If you request 1 GPU with 96 cores and 488GB memory, any other GPUs on the compute node are unavailable for other jobs.

### gpuavail

CONFIGURATION		AVAILABILITY					
NODE TYPE	NODE COUNT	NODE NAME	GPU TYPE	GPU COUNT	GPU AVAIL	CPU AVAIL	MEM AVAIL
gpu:pvc:4	19	ac046	h100	2	1	48	244
gpu:h100:2	15	ac055	h100	2	2	96	488
gpu:a30:4	1	ac064	a30	4	4	96	488
gpu:pvc:2	1						

### maxconfig -g a30 -G 1

```
#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --time=2-00:00:00
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=24
#SBATCH --mem=122G
#SBATCH --gres=gpu:a30:1
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

# maxconfig for a CPU-only Job

```
[username@aces ~]$ maxconfig
```

```
ACES partitions:  cpu gpu pvc bittware d5005 memverge nextsilicon gpu-hybrid  
ACES GPUs in gpu partition:  a30:4 h100:2 h100:4 pvc:2 pvc:4
```

```
Showing max parameters (cores, mem, time) for partition cpu
```

```
#!/bin/bash  
#SBATCH --job-name=my_job  
#SBATCH --time=7-00:00:00  
#SBATCH --nodes=1          # max 64 nodes for partition cpu  
#SBATCH --ntasks-per-node=1  
#SBATCH --cpus-per-task=96  
#SBATCH --mem=488G  
#SBATCH --output=stdout.%x.%j  
#SBATCH --error=stderr.%x.%j
```

# maxconfig for a GPU Job

```
[username@aces ~]$ maxconfig -g h100

ACES partitions:  cpu gpu pvc bittware d5005 memverge nextsilicon gpu-hybrid
ACES GPUs in gpu partition:  a30:4 h100:2 h100:4 pvc:2 pvc:4

Showing max parameters (cores, mem, time) for partition gpu and h100:2

#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --time=2-00:00:00
#SBATCH --nodes=1          # max 15 nodes for partition gpu
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --partition=gpu
#SBATCH --gres=gpu:h100:4
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

max number of GPUs per node can change because ACES is a composable resource cluster

# Select GPU type on ACES Cluster

```
#!/bin/bash
#SBATCH --job-name=my_gpu_job
#SBATCH --time=1-00:00:00           # request 1 day of time for the job
#SBATCH --ntasks-per-node=1       # request 1 task (command)
#SBATCH --cpus-per-task=48        # request 1/2 of the available cores since using 1 of 2 GPUs
#SBATCH --mem=244G                # request 1/2 of the available memory since using 1 of 2 GPUs
#SBATCH --gres=gpu:h100:1         # request 1 x H100 GPU; replace :1 with :2 for two GPUs
#SBATCH --partition=gpu           # use partition=gpu when selecting GPUs
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

# unload modules to start with a clean environment; then load required modules
module purge
module load CUDA/11.7.0
# run your gpu command
my_gpu_command
```

- There are two types of GPUs on ACES compute nodes. Select the type and quantity with `--gres`
  - H100      `--gres=gpu:h100:N`      (N can be 1 to **max**)      30 x H100 on ACES
  - A30      `--gres=gpu:a30:N`      (N can be 1 to **max**)      4 x A30 on ACES

The **max** value for **N** can change since ACES is a composable cluster

# Single-Node Multi-Core Job Scripts

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=244G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/11.2.0 SPAdes/3.15.3
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 48
```

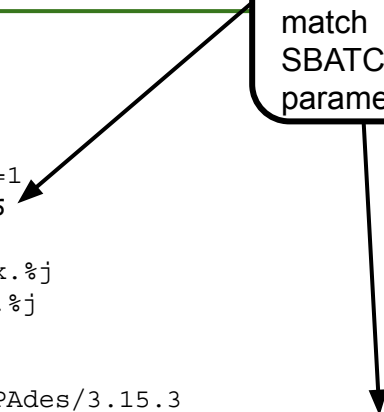
**Example 1**

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/11.2.0 SPAdes/3.15.3
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 96
```

**Example 2**

specify  
number of  
threads to  
match  
SBATCH  
parameters



It is best to request all cores if requesting all the memory because no other jobs will be scheduled when all the memory is requested and vice versa.

# Slurm Environment Variables

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/11.3.0 SPAdes/3.15.5
spades.py -1 s1_R1 -2 s1_R2 -o outdir_s1 --threads $SLURM_CPUS_PER_TASK
spades.py -1 s2_R1 -2 s2_R2 -o outdir_s2 --threads $SLURM_CPUS_PER_TASK
```

`$SLURM_CPUS_PER_TASK`  
variable used in  
commands to match  
SBATCH parameters

You can use the environment variable `$SLURM_CPUS_PER_TASK` to capture the value in the `#SBATCH --cpus-per-task` parameter so that you only need to adjust the cpus in one place.

# Multi-Node Jobs



# Slurm Parameters: --nodes --ntasks-per-node

```
#!/bin/bash
#SBATCH --job-name=myjob           # job name
#SBATCH --time=1:00:00            # set the wall clock limit to 1 hour
#SBATCH --nodes=2                 # request 2 nodes
#SBATCH --ntasks-per-node=1       # request 1 task (command) per node
#SBATCH --cpus-per-task=96        # request 96 cores per task
#SBATCH --mem=488G                # request 488GB of memory per node
#SBATCH --output=stdout.%x.%j     # create a file for stdout
#SBATCH --error=stderr.%x.%j     # create a file for stderr
```

It may be easier to scale jobs by using --nodes with --ntasks-per-node instead of with --ntasks. If you use --nodes with --ntasks, you need to calculate total CPUs for all nodes as the --ntasks value.

- **--nodes=2 --ntasks-per-node=96**
  - NumNodes=2 NumCPUs=192 NumTasks=192 CPUs/Task=1 mem=488G per node
- **--nodes=2 --ntasks=192**
  - NumNodes=2 NumCPUs=192 NumTasks=192 CPUs/Task=1 mem=488G per node
- **--nodes=1 --ntasks=96**
  - NumNodes=1 NumCPUs=96 NumTasks=96 CPUs/Task=1 mem=488G per node
- **--nodes=2 --ntasks=96**
  - will allocate 48 cores on one node and 48 cores on a second node
- **when --nodes is > 1, make sure the software you are using supports multi-node processing**

# MPI Multi-Node Multi-Core Job Script

```
#!/bin/bash
#SBATCH --job-name=my_mpi_job
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge

mpirun my_mpi_script -i input_file
```

mpirun reads the Slurm parameters to know how to distribute the job across compute nodes

960	Total CPUs requested
96	CPUs per node
488	Total requested GB memory per node

# TAMULauncher

- <https://hprc.tamu.edu/kb/Software/tamulauncher>
- Use when you have hundreds or thousands of commands to run, each utilizing a single-core or a few cores.
  - tamulauncher keeps track of which commands completed successfully
    - To see the list of completed commands:
      - `tamulauncher --status commands_file.txt`
      - If time runs out, tamulauncher can be restarted, and it will know which was the last successfully completed command
    - Submit tamulauncher as a batch job within your job script.
    - You can run tamulauncher interactively on login node-limited to 8 cores
    - You can check the --status on the command line from the working directory.
- Run a single command of your thousands to make sure the command is correct and to get an estimate of resource usage (CPUs, memory, time).
- Request all cores and memory on the compute node(s) and configure your commands to use all available cores.

# TAMULauncher Multi-Node Single-Core Commands

`commands.txt`

(500 lines for example)

`run_spades_tamulauncher.sh`

```
spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 1
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 1
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 1
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 1
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 1
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 1
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 1
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 1
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 1
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 1
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 1
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 1
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 1
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 1
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 1
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 1
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 1
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 1
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 1
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 1
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 1
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=96
#SBATCH --cpus-per-task=1
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

```
module purge
module load GCC/11.3.0 SPAdes/3.15.5

tamulauncher commands.txt
```

run 96 spades.py commands per node with each command using 1 core. Requesting all 96 cores on ACES reserves entire node for your job

- run 96 single-core commands per node; useful when each command requires  $\leq$  5GB memory
- create a commands file (named whatever you want) to go with the the job script
- load the software module in the job script not the commands file

# TAMULauncher Multi-Node Multi-Core Commands

`commands.txt`

(500 lines for example)

`run_spades_tamulauncher.sh`

```
spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 4
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 4
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 4
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 4
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 4
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 4
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 4
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 4
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 4
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 4
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 4
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 4
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 4
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 4
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 4
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 4
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 4
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 4
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 4
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 4
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 4
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 4
```

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --cpus-per-task=4
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

```
module purge
module load GCC/11.3.0 SPAdes/3.15.5

tamulauncher commands.txt
```

run 24 spades.py commands per node with each command using 4 cores. Requesting all 96 cores on ACES reserves entire node for your job

- useful when each command requires more than 5GB but less than all available memory
- use `OMP_NUM_THREADS` if needed when running fewer commands than requested cores
  - add on the line before the tamulauncher command
  - `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`

# Making a TAMULauncher Commands File

## Part 1

Input files are two files per sample and named: `s1_R1.fastq.gz` `s1_R2.fastq.gz`  
Run this command to create the example files:

```
mkdir seqs && touch seqs/s{1..40}_R{1,2}.fastq.gz
```

Run the following commands to get familiar with useful shell commands for creating and manipulating variables:

```
file=seqs/s1_R1.fastq.gz
echo $file
basename $file
sample=$(basename $file)
echo $sample
echo ${sample/_R1.fastq.gz}
echo ${sample/R1/R2}
```

```
# create variable named file
# show contents of variable
# strip off path of variable
# create variable named sample
# show contents of variable
# strip off _R1.fastq.gz
# substitute R1 text with R2
```

# Making a TAMULauncher Commands File

## Part 2

Input files are two files per sample and named: `s1_R1.fastq.gz` `s1_R2.fastq.gz`

- Run the following commands to loop through all R1 files in the reads directory and create the commands.txt.
- Use just the R1 files because we only need to capture the sample names once.

```
for file in seqs/*_R1.*gz
do
read1=$file
read2=${read1/_R1/_R2.}
sample=$(basename ${read1/_R1.fastq.gz})
echo spades.py -1 $read1 -2 $read2 -o ${sample}_out --threads 1
done > commands.txt
```

Match as much  
as possible to  
avoid matching  
sample names

# Other Useful Unix Commands

```
${variable##*SubStr} # will drop beginning of variable value up to first occurrence of 'SubStr'  
${variable###*SubStr} # will drop beginning of variable value up to last occurrence of 'SubStr'  
${variable%SubStr*} # will drop part of variable value from last occurrence of 'SubStr' to the end  
${variable%%SubStr*} # will drop part of variable value from first occurrence of 'SubStr' to the end
```

These are useful if the part of the filename for each sample that needs to be removed is not the same.

`s1_S1_R1.fastq.gz`    `s2_S2_R1.fastq.gz`    `s3_S3_R1.fastq.gz`

Make a new directory and create a new set of files for this exercise.

want to remove this part from each file name

```
mkdir seqs2  
for i in {1..10}; do touch seqs2/s${i}_S${i}_R{1,2}.fastq.gz; done
```

Unix Command

```
file=seqs2/s1_S1_R1.fastq.gz  
echo $file  
echo ${file%_S*}  
basename ${file%_S*}  
sample=$(basename ${file%_S*})  
echo $sample
```

Output

```
seqs2/s1_S1_R1.fastq.gz  
seqs2/s1  
s1  
  
s1
```



# Useful Slurm Runtime Environment Variables

- **\$TMPDIR**
  - This is a temporary local disk space (1.4TB) created at runtime and is deleted when the job completes.
  - The directory is mounted on the compute node, and files created in \$TMPDIR do not count against your file and disk quotas.
  - **samtools sort -T \$TMPDIR/sort**
- **\$SLURM\_CPUS\_PER\_TASK**
  - returns how many CPU cores were allocated on this node
  - can be used in your command to match requested #SBATCH cpus
    - **#SBATCH --cpus-per-task=96**
    - **spades.py --threads \$SLURM\_CPUS\_PER\_TASK**
- **\$SLURM\_ARRAY\_TASK\_ID**
  - can be used to select or run one of many commands when using a job array
- **\$SLURM\_JOB\_NAME**
  - populated by the --job-name parameter
  - **#SBATCH --job-name=spades\_job**
- **\$SLURM\_JOB\_NODELIST**
  - can be used to get the list of nodes assigned at runtime
- **\$SLURM\_JOBID**
  - can be used to capture JobID at runtime

# Useful Unix Environment Variables

- Type `env` to see all Unix environment variables for your login session.
- `$USER`
  - This will be automatically populated with your username.
    - `echo $USER`
- `$SCRATCH`
  - You can use this to change to your `/scratch/user/username` directory.
    - `cd $SCRATCH`
- `$OMP_NUM_THREADS`
  - used when software uses OpenMP for multithreading; default is 1
  - value is not updated based on Slurm parameters selected
  - set it manually by exporting the variable with the new value
    - `export OMP_NUM_THREADS=96`
- `$PWD`
  - contains the full path of the current working directory

# Finding NGS job template scripts using GCATemplates

Genomic Computational Analysis Templates

```
mkdir $SCRATCH/slurm_class
```

```
cd $SCRATCH/slurm_class
```

```
gcatemplates
```

For practice, we will copy a template file.

- Enter 1, then continue through the menus to find the template that contains bmap.
  - or use the search to find bmap
- The final step will save a template job script file to your current working directory.

```
BIOINFORMATICS GCATemplates (ACES)

CATEGORY
1. FASTQ files (QC, trim, SRA)
2. Protein tools

s search
q quit

Select:1
```

# Monitoring Job Resource Usage

# Submit a Slurm Job

- Submit your job script.
  - `sbatch run_bbmap_38.90_bbnorm_aces.sh`
- See status and JobID of all your submitted jobs.
  - `squeue --me`

JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMIT	NODES	NODELIST (REASON)
119112	cpu	bbnorm	username	RUNNING	3:54	1:00:00	1	ac005

- Can cancel (kill) a PENDING or RUNNING job using JOBID
  - `scancel JOBID`

# Monitor a Running Job

- See a lot of info about your running or recently completed (~10 minutes) job.

```
scontrol show job JobID
```

- You can add this command at the beginning of your job script to capture job info into the stdout file.

```
scontrol show job $SLURM_JOB_ID
```

example #SBATCH parameters

```
#!/bin/bash
#SBATCH --job-name=bbnorm
#SBATCH --time=01:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

```
JobId=731601 JobName=bbnorm
UserId=username(99999) GroupId=username(99999) MCS_label=N/A
Priority=22505 Nice=0 Account=myaces QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:09 TimeLimit=01:00:00 TimeMin=N/A
SubmitTime=2023-09-06T10:03:44 EligibleTime=2023-09-06T10:03:44
AccrueTime=2023-09-06T10:03:44
StartTime=2023-09-06T10:03:45 EndTime=2023-09-06T11:03:45 Deadline=N/A
SuspendTime=None SecsPreSuspend=0 LastSchedEval=2023-09-06T10:03:45
Scheduler=Main
Partition=cpu AllocNode:Sid=login2:513947
ReqNodeList=(null) ExcNodeList=(null)
NodeList=ac009
BatchHost=ac009
NumNodes=1 NumCPUs=96 NumTasks=1 CPUs/Task=96 ReqB:S:C:T=0:0:*:*
TRES=cpu=96,mem=488G,node=1,billing=96
Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
MinCPUsNode=96 MinMemoryNode=488G MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)

Command=/scratch/user/username/slurm_class/run_bbmap_38.96_bbnorm_aces.sh
WorkDir=/scratch/user/username/slurm_class
StdErr=/scratch/user/username/slurm_class/stderr.bbnorm.119112
StdIn=/dev/null
StdOut=/scratch/user/username/slurm_class/stdout.bbnorm.119112
Power=username
```

# See Completed Job Efficiency Stats

```
seff JobID
```

will show CPU and Memory efficiency based on selected resources

```
seff 119112

Job ID: 119112
Cluster: aces
User/Group: username/username
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 96
CPU Utilized: 03:20:22
CPU Efficiency: 57.80% of 05:46:40 core-walltime
Job Wall-clock time: 00:05:25
Memory Utilized: 228.01 GB
Memory Efficiency: 46.74% of 488.00 GB
```

CPU load was at 100% for 57% of the run time, or 37 cores were at 100% for the job time, or some other combination.

max memory utilized was 47% of requested memory

# Monitor GPU and CPU usage for a Job

You can use the jobstats command to monitor resource usage and create graphs.

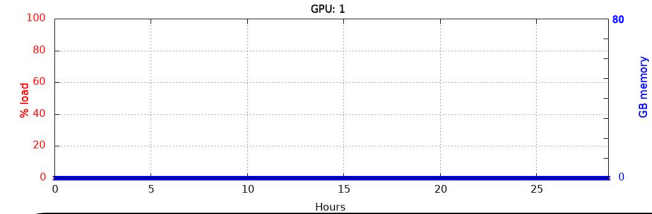
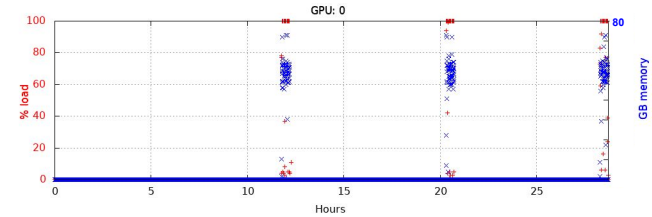
```
#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --time=2-00:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=96
#SBATCH --mem=488G
#SBATCH --gres=gpu:h100:2
#SBATCH --partition=gpu
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load CUDA/11.7.0

# run jobstats in the background (&) to monitor resource usage
jobstats &

my_gpu_command

# run jobstats to create graphs of resource usage for this job
jobstats
```



- Notice that GPU:1 was not used in this job.
- Adjust your software code to use 2 GPUs or the similar job can be configured to use just 1 GPU: `--gres=gpu:h100:1`

<https://hprc.tamu.edu/kb/Software/useful-tools/jobstats>



# ACES Service Unit Calculations

For the ACES compute nodes, when SU charging is enabled, you are charged Service Units (SUs) based on one of the following values whichever is greater.

- 1 SU per CPU per hour or 1 SU per 5GB of requested memory per hour

Number of Cores	Total Memory per node (GB)	Accelerator* count, type	Hours	SUs charged**
1	5	0	1	1
1	6	0	1	2
1	488	0	1	96
96	488	0	1	96

**As of Spring 2024, SU charging is not enabled**

\* Each Accelerator will be an additional SU charge per hour in the near future.

\*\* Currently SUs charging is not enabled for CPU or Accelerators.

# ACES Dashboard

ACES OnDemand Portal Files Jobs Clusters Interactive Apps **Dashboard** My Interactive Sessions

ACES Dashboard

**ACES DASHBOARD** [Create Help Ticket](#) [Request Software](#)

### CLUSTER STATISTICS

#### Node Utilization

#### Core Utilization

**Jobs**

Running	9
Pending	18

### SUMMARY

#### Accounts

Account ↑↓	Default ↑↓	Allocation ↑↓	Used ↑↓	Balance ↑↓
154669186753	Set Default	150000	0	150000
155062417651	Set Default	20000	0.53	19999.47
156171559762	Set Default	200000	0	200000

#### Disk Quotas

Disk	Disk Usage	Limit	File Usage	Limit
/home	19.72 MB (0.19 %)	10 GB	1023 (10.23 %)	10000
/scratch	32.08 GB (3.13 %)	1 TB	32391 (12.96 %)	250000

[Request Quota Increase](#)

D. Pham et al. (2022) PEARC '22  
<https://doi.org/10.1145/3491418.3535182>

# Debugging Job Submission

- The job was not scheduled.
  - Make sure the job CPU/GPU count and memory specification exist.

- ```
sbatch: error: CPU count per node can not be satisfied
sbatch: error: Batch job submission failed: Requested node configuration is not available
```

# PENDING Jobs

- If your job is in the PENDING state for a long time:
  - check to see if the cluster is busy using `sinfo` or see [hprc.tamu.edu](http://hprc.tamu.edu)
  - use `gpuavail` to see if the gpu partition is busy
  - check to see if your job walltime overlaps with a scheduled `maintenance`

```
The scheduled 11 hour ACES maintenance will start in:
```

```
3 days 16 hours 41 minutes
```

```
Scheduled jobs will not start if they overlap with this maintenance window.
```

A 7-day job submitted at the time of the above message will remain queued and will not start until after the maintenance is complete.

# Debugging COMPLETED Jobs

- Look in the stderr output file for an out of memory error message.
  - could occur in only one index of a job array

```
slurmstepd: error: Exceeded job memory limit at some point.
```

- Increase the amount of SBATCH memory in your job script, and resubmit the job.
- If you see an 'Out of disk space' or 'No space left on device' error,
  - check your file and disk quotas using the showquota command.
    - `showquota`
  - reduce the number of files you have generated.
    - delete any nonessential or temporary files.
    - use `$TMPDIR` in your command if software supports a temporary directory.
    - create and download a .tar.gz package of completed projects and delete the original directory to free up disk space.
  - request a temporary increase in file and/or disk quota for your project.

# Show Your Job Details using myjob

- The `myjob` command can be used to see detailed information related to your job
  - Status (PENDING, RUNNING, COMPLETED, FAILED, ...)
  - Node List
  - Submit time, Start time, End time, Total runtime
  - CPU Efficiency
  - Memory Utilized, Memory Efficiency
- will advise you if your job is PENDING due to a scheduled maintenance.
- will advise you if your job FAILED due to CRLF characters in the job script and provide a link to the HPRC documentation on how to resolve this issue.
- will advise you if your job FAILED due to file or disk quota being reached.
  - will show you the directory in your \$HOME directory that has the most files when \$HOME file quota is reached.

# Show Your Job Details using myjob

```
[username@aces ~]$ myjob 8862586
```

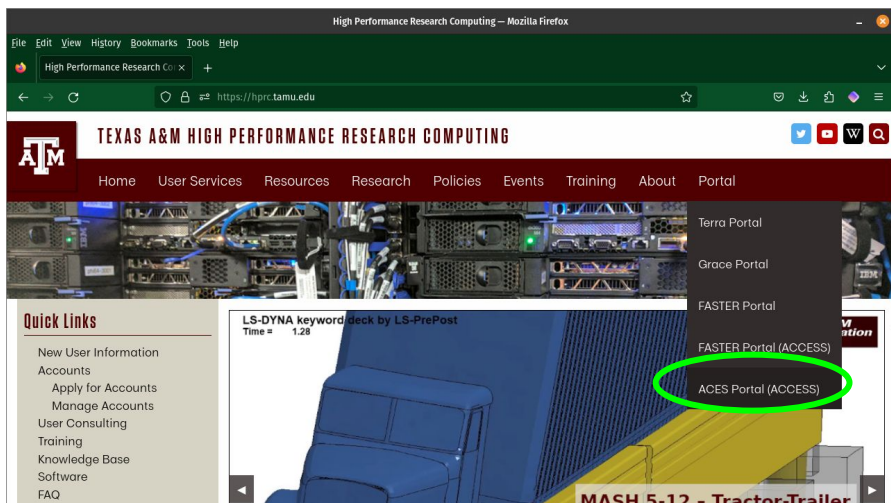
```
    Job ID: 8862586
    Cluster: grace
    User/Group: netid/netid
    Account: 123456789101
    State: COMPLETED (exit code 0)
    Partition: gpu
    Node Count: 1
    NodeList: g090
    Cores per node: 24
    CPU Utilized: 08:19:34
    CPU Efficiency: 2.39% of 14-12:55:36 core-walltime
    Submit time: 2023-08-09 15:29:07
    Start time: 2023-08-09 15:29:08
    End time: 2023-08-10 06:01:27
    Job Wall-clock time: 14:32:19
    Memory Utilized: 51.43 GB
    Memory Efficiency: 28.57% of 180.00 GB
    Job Name: alphafold
    Job Submit Directory: /scratch/user/netid/alphafold/2.3.1
    Submit Line: sbatch run_alphafold_2.3.1_a100_grace.sh
```

use the -h flag to view usage  
`myjob -h`

# hprc.tamu.edu

The portal allows users to do the following:

- Browse files on the the ACES filesystem.
- Access the ACES Unix command line.
  - runs on login node; limit your processes to 8 cores
- Compose and launch job scripts.
- Launch interactive GUI apps.
- Monitor and stop running jobs and interactive sessions.
- Request software installation and quota increases.



| Interactive Apps |
|------------------|
| GUI              |
| VNC              |
| Nextsilicon VNC  |
| Imaging          |
| CryoSPARC        |
| ImageJ           |
| cisTEM           |
| Servers          |
| Jupyter Notebook |
| JupyterLab       |
| RStudio          |



# Need Help?

First check the KnowledgeBase Documentation [hprc.tamu.edu/kb](http://hprc.tamu.edu/kb)

- ACES User Guide [hprc.tamu.edu/kb/User-Guides/ACES](http://hprc.tamu.edu/kb/User-Guides/ACES)
- Email your questions to [help@hprc.tamu.edu](mailto:help@hprc.tamu.edu)

Help us help you -- we need more info

- Which Cluster
- Username
- JobID(s) if any
- Location of your jobfile, input/output files
- Software Application used if any
- Module(s) loaded if any
- Error messages
- Steps you have taken, so we can reproduce the problem

Let us know when the issue has been resolved so we can close the helpdesk ticket.



High Performance  
Research Computing

DIVISION OF RESEARCH

Thank you

*Questions?*

