Introduction to the SWC

High Performance

Computing Systems

Introduction

High Performance Computing is a term given to the use of supercomputers and computer clusters to solve large computational problems.

Mostly nowadays its how to scale up a single desktop PC to a resource with 100's or 1000's of systems.

It is a combination of:

Compute hardware: CPUs & GPUs to perform computation

Data storage: for high-speed, low-latency, parallel access to large amounts of data

Networking: to link the devices together at high-speeds (10Gbit/sec or higher)

Efficiency: The hardware and running costs (mostly power) from having a large computing resources are high
So it makes sense to concentrate them and use them efficiency (24-7 operations)

Software: to provide an common environment in which to run your code and distribute efficiently across the computing hardware.

At SWC we have our own locally hosted and managed HPC resources. These are designed with SWC requirements in mind and are a mixture of CPUs, GPUs and large storage.

Compared with the large HPC sites we have a small number of the latest systems, but we also tend to keep older systems running all the time they are useful. So the HPC hardware resources are fairly heterogeneous. In total we have around 1500 CPU cores and 45 GPUS.

All the systems have fast access to main filestores (ceph, winstor, gatsbystor etc.).

All the systems have the same Operating System (currently Ubuntu 20.04 LTS) and the user environment is the same on all systems (same s/w filesystems etc).

The HPC resources are normally accessed and regulated via a system called SLURM. This system queues-up and distributes the workload to appropriate resources.

We are very flexible, so If the systems or services do not match your requirements, please raise a ticket to discuss this with the IT Project team.

CPU-only nodes

We have a lot of CPU (~1000 cores) capacity in blade servers, some of these are fairly old, but can still provide a lot of bulk capacity.

These tend to have lower contention.

Note: Many don't has much memory/core (so its good to request the correct amount)

Note2: Most only have 1Gb networking

CPU Platform Architecture

Architecture	Cores per CPU	RAM	Interconnect	Network	SLURM Feature	Intel Codename
Intel Xeon E5-2660 v3 @ 2.60GHz	10	64 - 128 GB	Gbe	Gbe	GenuineIntel-6-63	Haswell
Intel Xeon E5-2650 v4 @ 2.2GHz	12	64 - 256 GB	Gbe	Gbe	GenuineIntel-6-79	Broadwell
Intel Xeon E5-2660 v4 @ 2.2GHz	14	128 - 512 GB	10 Gbe	10 Gbe	GenuineIntel-6-79	Broadwell

GPU nodes

We currently have 18 nodes fitted with Nvidia GPU's of 6 different types:

Card	Architecture (Note 1)	SLURM gres	Cores (cuda:tensor:RT)	GFLOPS (s)	GFLOPS (d)	GPU Memory
GTX 1080	GP104-100	gtx1080	2560:160:64	8228	257	8GB
GTX 1080 TI	GP102-350		3584:224:88	10609	332	11GB
Quadro P5000	GP104	p5000	2560:160:64	8900	~300	8GB / 16GB ECC
Quadro P6000	GP102		3840:240:96	10882	~375	8GB ECC
RTX 2080	TU104	rtx2080	2944:184:64	8920	278	8GB
Quadro RTX 4000	TU104		2304:144:64	7119	223	8GB ECC
Quadro RTX 5000	TU104	rtx5000	3072:192:64	11151	349	16GB ECC
Quadro RTX 6000	TU102		4608:288:96	16312	510	24GB ECC
A100	GA100	a100	6912:432:108	19500	9700	40GB ECC
H100	GH100		14592:456:114	51200	25600	80GB ECC

Notes

1. GP = Pascal, GM = Maxwell, TU=Turing, GA = Ampere, GH = Hopper
2. Cores - Unified shaders: texture mapping units: render output units
3. GFLOPS given are those at the base clock rate : (s) single-precision (d) double precision

GPU nodes

About half of the capacity is in three quad-A100 systems, these are very similar in specs to those used as building bricks for very large Supercomputing systems (2 years ago).

They have 64 cores, 512GB RAM, multiple 25Gb (potentially 100Gb) networking, NVMe drives.

Most of the current workload doesn't need to use a whole a100 GPU (40GB memory) so we normally configure two of the systems in what Nvidia refer to as MIG mode. The allows a single GPU to act as multiple virtual GPU's

Currently we have a100's configured in a mode with three a100_2g.10gb instances so we can have effectively 12 (lower capacity) GPU's in a single server.

These lower capacity GPUs are actually still better in performance to some of the older GPU's...so if your task isn't too demanding (can fit into a rtx1080). You can generically request a GPU without worrying too much.

Getting Started on HPC

Get an SWC account - you will need this to be able to submit jobs using SLURM.

Be familiar with basic Unix commands.

You will need to be logged into a machine that is a SLURM client - e.g. ssh to a HPC gateway (or login to a managed Linux desktop system)

The HPC gateway node is called: hpc-gw1.hpc.swc.ucl.ac.uk You can use this for lightweight development work, file transfers etc.

There is also a bastion node (sgw2) that can be accessed externally as ssh.swc.ucl.ac.uk

Make sure that you are using the correct filesystems, file sizes etc. Look at things to avoid.

Familiarise yourself with submitting jobs to SLURM and requesting resources.

Develop your workflow - where is your input data and output data going, who should be able to access it etc.

File Systems

The HPC systems have fast access to all of the core storage systems either as a native mount (ceph) or via nfs. All the mounts are done via an automounter. The same maps are applied to all systems so that changes are made globally.

The main filesystems are:	/nfs/nhome and /nfs/ghome	home filesystems for SWC and Gatsby users
	/nfs/winstor/<Lab Group>	Winstor data storage for Lab Groups
	/nfs/gatsbystor	Gatsby data storage
	/ceph/scratch	Scratch storage (not backed up)
	/ceph/apps	Application area
	/ceph/<Lab Group>	Ceph data storage for Lab groups

The /ceph storage refers to what is becoming our main storage platform. cephfs is a distributed filesystem (currently using about 20 servers) where the data are stored with redundancy (currently we are using 8+3 encoding) using a mixture of NVMe drives and hard drives and has dedicated 100Gb switches on the HPC network. Currently we have around 8PB of storage.

Access via the Automounter:

```
amartin@hpc-gw1:~$
amartin@hpc-gw1:~$ ls  /ceph
aeon  apps  erlich  neuroinformatics  scratch
amartin@hpc-gw1:~$ ls  /ceph/akrami
Edmund  Peter    TEST                capsid_testing        neuropixels_recordings
George  Quentin  akrami_transfer.log  mouse_reconstruction
amartin@hpc-gw1:~$ ls  /ceph
aeon  akrami  apps  erlich  neuroinformatics  scratch
amartin@hpc-gw1:~$ ls  /ceph/margrie
ls: cannot open directory '/ceph/margrie': Permission denied
amartin@hpc-gw1:~$ ls  /ceph
aeon  akrami  apps  erlich  margrie  neuroinformatics  scratch
amartin@hpc-gw1:~$
amartin@hpc-gw1:~$
amartin@hpc-gw1:~$ id amartin
uid=801150777(amartin) gid=801150777(amartin) groups=801150777(amartin),801100504(gatsby),801151006(akrami)
```

In general you will only have access to the data shares of the groups you are a member of. In you think this is wrong you will need to contact the helpdesk.

How much storage space can I use?

Most of the file storage has quotas. Individually on the home file systems and for the Lab shares. For your home space, its fairly limited and you should not be using it for data storage:

```
amartin@sgw2:~/sample_jobs$ quota
Disk quotas for user amartin (uid 801150777):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
swc-homes.id.swc.ucl.ac.uk:/vol_homes
               8871864  209715200 262144000           45350  4294967295 4294967295
amartin@sgw2:~/sample_jobs$
```

For ceph volumes the quota defines total space on the volume:

```
amartin@hpc-gw1:~$ df -BT | grep ceph
192.168.234.154,192.168.234.155,192.168.234.156,192.168.234.157,192.168.234.158,192.168.234.159:6789:/erlich                91T   28T       64T  31% /ceph/erlich
192.168.234.154,192.168.234.155,192.168.234.156,192.168.234.157,192.168.234.158,192.168.234.159:6789:/aeon                273T  189T       85T  70% /ceph/aeon
192.168.234.154,192.168.234.155,192.168.234.156,192.168.234.157,192.168.234.158,192.168.234.159:6789:/neuroinformatics      19T    4T       16T  17% /ceph/neuroinformati
192.168.234.154,192.168.234.155,192.168.234.156,192.168.234.157,192.168.234.158,192.168.234.159:6789:/apps                 5T    1T        5T   9% /ceph/apps
amartin@hpc-gw1:~$
```

If you run out of space you will need to raise it with the HelpDesk.

SWC HPC Software Environment

The HPC (and other managed Linux systems) have a base installation based on a version of Ubuntu with long term support (currently 20.04 LTS) and because upgrades take some effort to manage the changes and fix the bugs, we aim to maintain a stable environment foe a few years.

If a package comes with the standard OS or supported repo, we generally add it to a list of packages that are automatically installed (by puppet). This ensures its on all systems, even if a system needs to be reinstalled.

If you would like a standard package installed, please ask.

For some software, particularly python based packages, you may find it best to setup your own environments using e.g. conda/pip?

For other software which is not pre-packaged or here we need to have multiple versions we install in a shared area and use the module system to manage it.

Linux Environmental Modules

The HPC (and other managed Linux systems) uses a package called environmental modules for managing the shell environment including software paths. The modules system permits users to set up the shell environment to make running and compiling software easier. It also allows users to avail of many software packages and libraries that might otherwise conflict with one another and to maintain multiple versions of the same package.

The module system is a script based system used to manage the user environment and to “activate” software packages. In order to access software that is installed on the cluster, you must first load the corresponding software module.

In the background these scripts are setting up the appropriate PATHS and environmental variables to use a particular package

.

Useful modules commands

module avail	lists all available software modules
module avail [name]	lists modules matching [name]
module load [name]	loads the named module
module unload [name]	unloads the named module
module list	lists the modules currently loaded for the user
module help	help

examples

To find out which packages are available do; > module avail

```
amartin@hpc-gw1:~$ module avail
---------------------------- /usr/share/modules/modulefiles ----------------------------
dot  module-git  module-info  modules  null  use.own

---------------------------- /ceph/apps/ubuntu-20/modulefiles ----------------------------
brainglobe/2022-07-05  cuda/11.6              matlab/R2018a     neuron/8.0
cuda/9.0               cuda/11.8              matlab/R2019b     pycharm/2022.2.1
cuda/10.1              cuda/12.0              matlab/R2021a     SLEAP/2023-03-13
cuda/10.2              deeplabcut/2022-07-06  matlab/R2022a
cuda/11.2              julia/1.7.3            matplotlib/3.3.4
cuda/11.5              kilosort3/2022-10-06   miniconda/4.9.2
```

To find out which modules are loaded > module list

```
amartin@hpc-gw1:~$ module list
No Modulefiles Currently Loaded.
amartin@hpc-gw1:~$ module load cuda
amartin@hpc-gw1:~$ module list
Currently Loaded Modulefiles:
 1) cuda/12.0
amartin@hpc-gw1:~$ module unload cuda
amartin@hpc-gw1:~$ module load cuda/11.8
amartin@hpc-gw1:~$ module list
Currently Loaded Modulefiles:
 1) cuda/11.8
```

To load a module > module load <name>/<version> if no <version> there is a default

To unload a module > module unload <name>

Useful modules commands

module avail	lists all available software modules
module avail [name]	lists modules matching [name]
module load [name]	loads the named module
module unload [name]	unloads the named module
module list	lists the modules currently loaded for the user
module help	help

Overview of SLURM job scheduler

The HPC cluster is a shared resource with different users who may want to run different jobs at the same time. To make sure two or more users don't run two different conflicting programs on the same node, the cluster uses a job scheduler that schedules different jobs from different users. This means making sure it allocates each job to a different node and, if all the nodes are being used, putting any extra jobs that can't currently be run in a queue so that they are sequentially allocated to different nodes in some useful way (e.g. run all the short jobs first, then run all the long ones). The job scheduler that we use is called SLURM (Simple Linux Utility for Resource Management). It performs the following functions:

Schedules submitted jobs.

Allocates requested compute resources.

Processes submitted jobs.

Like any other job scheduler, SLURM requires that you submit jobs to the queue in a particular way. Learning this is basically all there is to it.

Our SLURM setup is very similar to that of other organisations. You should be able to utilise examples scripts from anywhere with only slight modification for the local queues and software stack.

Useful SLURM commands

SLURM offers a number of helpful commands for tasks ranging from job submission, monitoring and modifying resource requests for jobs that have already been submitted to the queue. Below is a list of SLURM commands with details on how to use them. You must be logged into a system that is a SLURM client to use these commands.

sinfo The command allows users to view information about SLURM nodes and partitions.

sbatch This command is used for submitting jobs to the cluster. sbatch accepts a number of options either from the command line or from a batch script.

srun The command is used to submit jobs for real-time execution. It can also be used to submit interactive jobs.

squeue This command shows you the current queue, i.e. the jobs currently running and which nodes they are running on, and the jobs not yet running but on the queue.

scancel The command removes a job from the queue, or cancel a job that is currently running. If the job number i is 1234, cancel it with scancel 1234. To cancel all the jobs submitted by user myname, use scancel -u myname.

sacct This command is used for viewing and display information for completed jobs. This can be useful for monitoring job progress or diagnosing problems that occurred during job execution.

salloc The command allocates resources for an interactive job.

SLURM partitions

The HPC cluster has generally two main partitions (or queues) 'cpu' and 'gpu' for production work, there is also a small 'fast' partition intended for development work. The purpose of the 'fast' partition is to allow users to quickly test a job before submitting a larger number of jobs to the production partition. You can display information about these partitions using the sinfo command. The following is a typical output for the SWC cluster sinfo:

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
cpu*         up   infinite      7 drain~ enc0-node[1-7]
cpu*         up   infinite     26  idle~ enc1-node[1-6,8-10,12,14],enc2-node[1-6,8-12],enc3-node[5-8]
cpu*         up   infinite      2    mix enc1-node7,enc2-node7
cpu*         up   infinite      3  alloc enc1-node[11,13],enc2-node13
cpu*         up   infinite      4   idle enc3-node[1-4]
gpu          up   infinite     11    mix gpu-350-[01-05],gpu-380-[11-13],gpu-sr670-[20-22]
gpu          up   infinite      6   idle gpu-380-[10,14-18]
fast         up    3:00:00      2   idle enc1-node16,gpu-erlich01
```

This also should the status of the compute nodes: nodes that are idle are currently free of jobs..so a submitted job should run if it can fit into node.

Nowadays we have enabled power saving on the cluster and shut down most systems when thay have been idle for a while. You can see this with the nodes having “~” suffix to their state. Nodes in power saving mode are powered backup and put back to work when allocated new jobs, with a latency
of 5-10 mins. This would be indicated by a “#” suffix. This allows us to keep a lot of the older
kit still available as “surge capacity” without wasting power.

Types of Jobs

Batch jobs (sbatch command)

A batch job is a non-interactive way of running a job on the cluster. There's no user input as the job script controls the job. When a batch job is submitted to the cluster, it is put in a queue and then started later. The advantage of using batch jobs is that you can queue many jobs simultaneously, which can start automatically once resources are available. This is the primary method of running applications on the cluster

Interactive jobs (srun command)

An interactive job is a job that returns a command line prompt instead of running a script, when the job runs. Interactive jobs are useful when debugging or interacting with an application. To run interactive jobs, you use the srun command. When the job starts, a command line prompt will appear on the compute node assigned. From here commands can be executed using the resources allocated on the local node. The main advantage of interactive jobs is that you get immediate feedback.

Preparing a batch script

The first step for submitting a job to SLURM is to write a batch script. This instructs the scheduler how to run the script and what to do with the results.The script is essentially a simple shell file that includes several #SBATCH directive lines that tell SLURM details about your job, including the resource requirements.

Below is an example of how to write a simple job batch script file. ~amartin/sample_jobs/test1.sh

```
#!/bin/bash
#
#SBATCH -p cpu              # partition (queue)
#SBATCH -N 1                # number of nodes
#SBATCH -n 1                # number of cores
#SBATCH --mem 100           # total memory required  for all cores  (N.B.  default units are MB)
#SBATCH -t 0-2:00:00        # time (D-HH:MM:00)
#SBATCH -o slurm.%N.%j.out # STDOUT    (%N expands to the hostname   %j to the job nuumber)
#SBATCH -e slurm.%N.%j.err # STDERR
#
#
#
hostname
sleep 10
exit
```

Submitting a job script

Once your script is prepared, you are ready to submit your job. To submit your job to the queuing system, use the command sbatch. SLURM will then try to find or wait for available resources matching your request and run your job there. For example, if your script is in 'test1.sh' the command would be:

sbatch test1.sh

This will return a message with your job id.

```
amartin@hpc-gw1:~/sample_jobs$ sbatch test1.sh
Submitted batch job 3446190
amartin@hpc-gw1:~/sample_jobs$ ls
slurm.enc1-node7.3446190.err  slurm.enc1-node7.3446190.out  test1.sh
amartin@hpc-gw1:~/sample_jobs$ cat slurm.enc1-node7.3446190.out
enc1-node7
amartin@hpc-gw1:~/sample_jobs$
```

Monitor your job status

Once your job is submitted, you can monitor the progress of the job using several commands. To see your jobs, use the squeue -u command and specify your username as shown below:

squeue -u <Your username> (if you miss out your username you get all jobs)

```
amartin@hpc-gw1:~/sample_jobs$ squeue -u amartin
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           3446192       cpu test2.sh  amartin  R       0:54      1 enc1-node7
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 3446123_[6,36-41]       cpu synth_sl  emmettt PD       0:00      1 (Resources)
           3434778       cpu     bash  pierreg  R 11-19:53:56      1 enc1-node7
           3439724       cpu     bash  pierreg  R 8-20:01:42      1 enc2-node7
           3446192       cpu test2.sh  amartin  R       1:00      1 enc1-node7
           3445708       cpu     bash     jlee  R 1-04:32:28      1 enc2-node7
        3446123_21       cpu synth_sl  emmettt  R      26:11      1 enc2-node10
        3446123_35       cpu synth_sl  emmettt  R      26:40      1 enc1-node6
        3446123_28       cpu synth_sl  emmettt  R      26:41      1 enc1-node5
        3446123_34       cpu synth_sl  emmettt  R      26:42      1 enc1-node3
        3446123_33       cpu synth_sl  emmettt  R      26:43      1 enc3-node6
        3446123_24       cpu synth_sl  emmettt  R      27:11      1 enc1-node1
        3446123_25       cpu synth_sl  emmettt  R      27:11      1 enc1-node2
        3446123_27       cpu synth_sl  emmettt  R      27:11      1 enc1-node4
```

Check your job output

Once your job is submitted and has started, it will write its standard output and standard error to files that you can read. SLURM will put the output in the directory where you submitted the job in a file named slurm- followed by the job id with the extension out.

Below is an example of how to write a simple job batch script file. ~amartin/sample_jobs/test2.sh

```
#!/bin/bash
#
#SBATCH -p cpu              # partition (queue)
#SBATCH -N 1                # number of nodes
#SBATCH -n 1                # number of cores
#SBATCH --mem 100           # total memory required  for all cores  (N.B.  default units are MB)
#SBATCH -t 0-2:00:00        # time (D-HH:MM:00)
#SBATCH -o slurm.%j.out # STDOUT    (%N expands to the hostname   %j to the job nuumber)
#SBATCH -e slurm.%j.err # STDERR
#
hostname
cat nosuchfile.txt
exit
```

```
amartin@hpc-gw1:~/sample_jobs$ sbatch test2.sh
Submitted batch job 3446195
amartin@hpc-gw1:~/sample_jobs$ ls
slurm.enc1-node7.3446195.err  slurm.enc1-node7.3446195.out  test1.sh  test2.sh
amartin@hpc-gw1:~/sample_jobs$ cat slurm.enc1-node7.3446195.out
enc1-node7
amartin@hpc-gw1:~/sample_jobs$ cat slurm.enc1-node7.3446195.err
cat: nosuchfile.txt: No such file or directory
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
amartin@hpc-gw1:~/sample_jobs$
```

How do I find out what happened to my Job and what Resources it used?

You need to have the Job_ID!

Then you see what happened to it and what resources it used. Using the sacct command

```
amartin@sgw2:~/sample_jobs$
amartin@sgw2:~/sample_jobs$ sacct --format="NNodes,NCPUS,Start,End,CPUTime,MaxRSS" -j 3446122
  NNodes      NCPUS               Start                 End    CPUTime     MaxRSS
-------- ---------- ------------------- ------------------- ---------- ----------
       1          1 2023-05-11T14:09:45 2023-05-11T14:14:42   00:04:57    169276K
amartin@sgw2:~/sample_jobs$
```

There are a lot more more possible parameters...do sacct -l or man sacct to see them.

Using the GPU systems

To use a GPU you need to reserve one. You won't get one if you just submit a job to the gpu partition

To reserve a GPU use the sbatch --gres directive

It takes the general form:

--gres gpu:<GPU Type>:Number of GPU's

<GPU Type> is optional

```
examples:

#SBATCH --gres gpu:1                 # requust  any gpu

#SBATCH --gres gpu:rtx5000:2         # if you wanted two rtx5000  gpus

#SBATCH --gres gpu:a100_2g.10gb:1    # a 2g.10gb MIG instance on a A100 GPU
                                     # (You can't have more than one of them)
```

You can check what GPU's are available in you job using the nvidia-smi command

Example of GPU Job gpu_test.sh

```
/bin/bash
#SBATCH -p gpu
#SBATCH --mem=1G
#!
### request GPU
#SBATCH --gres gpu:1                # requust  any gpu

### Other examples

###SBATCH --gres gpu:rtx5000:2      # if you wanted two rtx5000  gpus
###SBATCH --gres gpu:a100_2g.10gb:1 # request  a100 2g.10gb instance


hostname

###  load CUDA  module

module load cuda/11.8
nvidia-smi
printenv | grep CUDA

###  load tensorflow module

module load tensorflow
python  ./test_tf.py


exit
```

```
amartin@sgw2:~/sample_jobs$ more test_tf.py
import tensorflow as tf
print("TensorFlow version:", tf.__version__)


with tf.device('/gpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)
    print(c)
```

In this simple tensorflow example we specify to use the first GPU device /gpu:0 (The GPU allocated by SLURM will always appear as the first GPU)

```
amartin@sgw2:~/sample_jobs$ more slurm-3446206.out
gpu-sr670-20
Thu May 11 17:27:06 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  On   | 00000000:31:00.0 Off |                   On |
| N/A   31C    P0    62W / 400W |   2635MiB / 40960MiB |     N/A      Default |
|                               |                      |              Enabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared        |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                       |
|==================+======================+===========+=======================|
|  0    3   0   0  |     13MiB /  9856MiB | 28      0 |  2   0    1    0    0 |
|                  |      0MiB / 16383MiB |           |                       |
+------------------+----------------------+-----------+-----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
CUDA_DEVICE_ORDER=PCI_BUS_ID
CUDA_VISIBLE_DEVICES=MIG-GPU-c37dfdb2-af80-ac6d-782f-1517794733be/3/0
```

```
2023-05-11 17:27:08.869741: I tensorflow/core/util/port.cc:110] oneDNN custom operations are on. You ma
y see slightly different numerical results due to floating-point round-off errors from different comput
ation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-05-11 17:27:09.054611: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary
 is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild Tensor
Flow with the appropriate compiler flags.
2023-05-11 17:27:13.239371: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cou
ld not find TensorRT
2023-05-11 17:27:21.114005: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device /jo
b:localhost/replica:0/task:0/device:GPU:0 with 7960 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-40
GB MIG 2g.10gb, pci bus id: 0000:31:00.0, compute capability: 8.0
2023-05-11 17:27:24.828278: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloa
t-32 will be used for the matrix multiplication. This will only be logged once.
TensorFlow version: 2.12.0
tf.Tensor(
[[22. 28.]
 [49. 64.]], shape=(2, 2), dtype=float32)
```

Parallel jobs in SLURM

There are many ways to do parallel work with SLURM.

Running a multi-threaded job or a multi-process job (e.g.: MPI) (The current HPC is not really setup to do this efficiently across servers)

Running several instances of the same job using job arrays.

Multi-process & Multi-threading

Multi-threaded programs are applications that are able to execute in parallel across multiple CPU cores within a single node using a shared memory execution model.

To request that a parallel script spawns children processes tasks (e.g.: MPI), use the `--ntasks` option.

To request that a job uses multiple threads, use the option `--cpus-per-task

Examples:

you want 16 processes to stay on the same node: --ntasks=16 --ntasks-per-node=16

you want one process that can use 16 cores for multithreading: --ntasks=1 --cpus-per-task=16

you want 16 processes to spread across 8 nodes to have two processes per node: --ntasks=16 --ntasks-per-node=2 (Not recommended)

Array batch jobs

Job arrays are useful for submitting and managing a large number of similar jobs. This can be used when you need to run the same script multiple times in parallel (for different random seeds or choice of parameter for instance).

Here is an examples of a slurm script using a job arrays.

The environment variable `SLURM_ARRAY_TASK_ID` holds the value of the job array.

Example of CPU Job array sbatch script array_test.sh

```
#!/bin/bash
#
#SBATCH -p cpu                # partition (queue)
#SBATCH -N 1                  # number of nodes
#SBATCH -n 8                  # 8 cores
#SBATCH --mem 1G              # total memory required  for all cores  (N.B.  default units are MB)
#SBATCH -t 0-2:00:00          # time (D-HH:MM:00)
#
#SBATCH --array=0-3           #  create an array of 4 tasks  0,1,2,3
#
#SBATCH -o slurm.%A_%a.out # STDOUT    (%A  JOB_ID    %a  is task ID)
#SBATCH -e slurm.%A_%a.err # STDERR
#
#
#
hostname

JOB_ARGS=(10 50 1000 2000)

./array_job.sh ${JOB_ARGS[$SLURM_ARRAY_TASK_ID]}
exit
```

When run this will effectively run four jobs with different arguments for the following script

```
amartin@sgw2:~/sample_jobs$
amartin@sgw2:~/sample_jobs$ cat array_job.sh
#!/bin/bash
echo "array_job called with argument: " $1
sleep 100
amartin@sgw2:~/sample_jobs$
```

```
amartin@sgw2:~/sample_jobs$ sbatch array_test.sh
Submitted batch job 3446381
amartin@sgw2:~/sample_jobs$ squeue -u  amartin
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         3446381_0       cpu array_te  amartin  R       0:07      1 enc1-node7
         3446381_1       cpu array_te  amartin  R       0:07      1 enc3-node2
         3446381_2       cpu array_te  amartin  R       0:07      1 enc3-node2
         3446381_3       cpu array_te  amartin  R       0:07      1 enc3-node2
amartin@sgw2:~/sample_jobs$
amartin@sgw2:~/sample_jobs$ ls -alrt
total 68
-rw-rw-r--  1 amartin amartin   452 May 11 15:05 test1.sh
-rw-rw-r--  1 amartin amartin   471 May 11 15:27 test2.sh
-rw-rw-r--  1 amartin amartin   483 May 11 15:45 test3.sh
-rw-rw-r--  1 amartin amartin   288 May 11 17:21 test_tf.py
-rw-rw-r--  1 amartin amartin  2503 May 11 17:22 slurm-3446205.out
-rwxr-xr-x  1 amartin amartin   356 May 11 17:26 gpu_test.sh
-rw-rw-r--  1 amartin amartin  3761 May 11 17:27 slurm-3446206.out
-rwxrwxr-x  1 amartin amartin    65 May 12 09:31 array_job.sh
-rw-rw-r--  1 amartin amartin   515 May 12 09:33 array_test.sh
drwxr-xr-x 48 amartin root    24576 May 12 09:33 ..
-rw-rw-r--  1 amartin amartin     0 May 12 09:34 slurm.3446381_0.err
-rw-rw-r--  1 amartin amartin     0 May 12 09:34 slurm.3446381_3.err
-rw-rw-r--  1 amartin amartin     0 May 12 09:34 slurm.3446381_1.err
-rw-rw-r--  1 amartin amartin     0 May 12 09:34 slurm.3446381_2.err
drwxrwxr-x  2 amartin amartin  4096 May 12 09:34 .
-rw-rw-r--  1 amartin amartin    47 May 12 09:34 slurm.3446381_0.out
-rw-rw-r--  1 amartin amartin    49 May 12 09:34 slurm.3446381_3.out
-rw-rw-r--  1 amartin amartin    49 May 12 09:34 slurm.3446381_2.out
-rw-rw-r--  1 amartin amartin    47 May 12 09:34 slurm.3446381_1.out
amartin@sgw2:~/sample_jobs$ cat slurm.3446381_3.out
enc3-node2
array_job called with argument:  2000
```

Using variables to define the array limits

SLURM does not support using variables in the #SBATCH lines within a job script.

However, values passed from the command line have precedence over values defined in the job script. So the array can be passed on the sbatch command line.

So in the previous example if you do

```
sbatch --array=1-2 ./array_test.sh
```

you will find that two of the array tasks are submitted with task_ids 1 and 2

Interactive Jobs: srun command

The srun command launches a task interactively, it has a bunch of command-line options which basically correspond to the #SBATCH directives (do man srun to look them up)

```
amartin@sgw2:~/sample_jobs$
amartin@sgw2:~/sample_jobs$
amartin@sgw2:~/sample_jobs$ srun -p cpu hostname
enc2-node7
amartin@sgw2:~/sample_jobs$
amartin@sgw2:~/sample_jobs$ srun -p gpu --gres=gpu:1 nvidia-smi
Fri May 12 09:00:48 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Quadro P5000        Off  | 00000000:0B:00.0 Off |                  Off |
| 19%   34C    P0    39W / 180W |      0MiB / 16384MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
amartin@sgw2:~/sample_jobs$ srun -p fast --nodes=1 --ntasks-per-node=1 --time=01:00:00 --pty bash -i
(base) amartin@enc1-node16:~/sample_jobs$
(base) amartin@enc1-node16:~/sample_jobs$
```