

Using GPUs at the CHPC

Martin Čuma

*Center for High Performance
Computing University of Utah
m.cuma@utah.edu*



- SLURM scheduler basics.
- GPU types.
- SLURM and GPUs.
- Finding (free) GPUs.
- Submitting job with GPUs
 - Open OnDemand
 - Interactive job on terminal
 - SLURM job script
- Monitoring GPU jobs



- All GPUs are on compute nodes - must run a job to get them
 - Typically GPU partition has "gpu" in its name, e.g. notchpeak-gpu
- Recall basic SLURM commands:
 - `sbatch` - submit a scripted job
 - `salloc` - submit an interactive job
 - `squeue` - see the queued/running jobs (`squeue --me`)
 - `scancel` - cancel a job
 - `sinfo` - list partitions
- Convenience CHPC commands
 - `myallocation` - list accounts/partitions available to user
 - `mysqueue` - list user accounts/partitions and jobs on them
 - `mysinfo` - list user accounts/partitions and state of their nodes



- CHPC has GPUs from many different generations
 - Their performance and capabilities vary widely
 - https://www.chpc.utah.edu/documentation/guides/gpus-accelerators.php#gpu_types
- (Nvidia) GPU classification:
 - Generation (code name - Maxwell, Pascal, Volta, Turing, Ampere, Hopper)
 - Compute Capability (5.2, 6.0, 6.1, 7.0, 7.5, 8.0, 8.6, 8.9, 9.0)
<https://developer.nvidia.com/cuda-gpus#compute>
 - Theoretical compute throughput (single, double precision, tensor)
https://en.wikipedia.org/wiki/Nvidia_Tesla
<https://en.wikipedia.org/wiki/Quadro>
<https://en.wikipedia.org/wiki/GeForce>
https://en.wikipedia.org/wiki/GeForce_40_series
 - Amount of memory (~10-80 GB)

- Example: A40, A100, H100

- Data center GPUs, high double precision performance, large memory
- Expensive (H100 ~\$20000 w/ edu discount), hard to get
- Good for simulations that need high numerical precision (engineering), or high memory (AI)

Model	Micro-architecture	Launch	Core	Core clock (MHz)	Shaders			Memory				Processing power (GFLOPS) ^[a]			CUDA compute capability ^[b]	TDP (W)	
					CUDA cores (total)	Base clock (MHz)	Max boost clock (MHz) ^[c]	Bus type	Bus width (bit)	Size (GB)	Clock (MT/s)	Bandwidth (GB/s)	Half precision Tensor Core FP32 Accumulate	Single precision (MAD or FMA)			Double precision (FMA)
A40 GPU accelerator (PCIe card) ^[43]		October 5, 2020	1× GA102	—	10,752	1,305	1,740	GDDR6	384	48	7,248	695.8	149,680	37,420	1,168	8.6	300
A100 GPU accelerator (PCIe card) ^{[44][45]}		May 14, 2020 ^[46]	1× GA100-883AA-A1	—	6,912	765	1410	HBM2	5,120	40 or 80	1,215	1,555	312,000	19,500	9,700	8.0	250
H100 GPU accelerator (PCIe card) ^[47]	Hopper	March 22, 2022 ^[48]	1× GH100 ^[49]	—	14,592	1,065	1,755 CUDA 1620 TC	HBM2E	5120	80	1,000	2,039	756,449	51,200	25,600	9.0	350
H100 GPU accelerator (SXM card)				—	16,896	1,065	1,980 CUDA 1,830 TC	HBM3	5,120	80	1,500	3,352	989,430	66,900	33,500	9.0	700

- Example: RTX3090

- Consumer grade graphics cards, recent don't fit to compute servers
- Very low double precision performance, good single and tensor performance
- More affordable (a few thousand \$)
- Good for lower precision numerical calculations, AI with small to medium size models

Model	Launch	Launch MSRP (USD)	Code name(s) ^[b]	Transistors (billion)	Die size (mm ²)	Core config ^[c]	SM count ^[d]	L2 cache (MB)	Clock speeds ^[e]		Fillrate ^{[f][g]}		Memory				Processing power (TFLOPs) ^[h]				TDP (watts)
									Core (MHz)	Memory (GT/s) ^[i]	Pixel (Gpx/s)	Texture (Gtex/s)	Size (GB)	Bandwidth (GB/s)	Type	Bus width (bit)	Half (boost)	Single (boost)	Double (boost)	Tensor compute [sparse]	
GeForce RTX 3090 Ti ^{[40][65]}	Mar 29, 2022	\$1,999	GA102-350	28.3	628.4	10752 336:112:84:336	84	6	1560 (1860)	10.5	174.7 (208.3)	524.1 (625)	24	1008	GDDR6X	384	33.55 (39.99)	33.55 (39.99)	0.524 (0.625)	160 ^[66] [320]	450
GeForce RTX 3090 ^{[40][63]}	Sep 24, 2020	\$1,499	GA102-250 ^[64] GA102-300	28.3	628.4	10496 328:112:82:328	82	6	1395 (1695)	9.75	156.2 (189.8)	457.6 (556)	24	936	GDDR6X	384	29.28 (35.58)	29.28 (35.58)	0.458 (0.556)	142 ^[34] [284]	350



- Example: RTX6000, A6000

- Mid size models (graphical workstations), more memory and a bit pricier than the gaming graphics cards
- Very low double precision performance, good single and tensor performance
- Good for lower precision numerical calculations, AI with small to medium size models

Quadro GPU	Launch	Core	Core clock	Memory clock	Memory size (GB)	Memory type	Memory bandwidth	CUDA cores	Tensor cores	RT cores	Half precision	Single precision	Double precision	CUDA Compute Capability
Units	↕	↕	MHz	MHz	GB	↕	GiB/s	↕	↕	↕	TFLOPS	TFLOPS	GFLOPS	↕
RTX 6000 Ada Generation ^[202]	2022-12-03	AD102-870	915–2505	2500	48	384-bit GDDR6	960	18176	568	142	91.06 ^[203]	91.06	1423	8.9
RTX A6000 ^{[185][186]}	2020-10-05	GA102-875	1410–1800	2000	48 (96 with NVLink 3.0)	384-bit GDDR6	768	10752	336	84	38.709 ^[187]	38.709	1209.677	8.6



- GPUs have to be requested explicitly
 - `#SBATCH --gres=gpu:type:count`
- GPU type is also a *feature*

```
$ sinfo -p notchpeak-gpu
```

PARTITION	NODES	NODES (A/I/O/T)	S:C:T	FREE_MEM	MEMORY	TIMELIMIT	AVAIL_FEATURES	NODELIST
notchpeak-gpu	3	3/0/0/3	2:16:2	77298-136143	188000	3-00:00:00	chpc,skl,tesla,v100,c32,m192,cx4	notch[001-003]
notchpeak-gpu	1	1/0/0/1	2:16:2	25778	188000	3-00:00:00	chpc,skl,geforce,2080ti,tesla,p40,c3	notch004
notchpeak-gpu	1	1/0/0/1	2:32:2	146805	508000	3-00:00:00	chpc,rom,geforce,3090,a100,c64,m512,	notch293

- no feature/gres for GPU memory (yet)
- `#SBATCH --gres=gpu:v100:1`

- Further details on GPU nodes

```
$ sinfo -N -h -p notchpeak-gpu --Format='nodehost,gres:50'
```

```
notch001          gpu:v100:3 (S:0-1)
notch293          gpu:3090:4 (S:0) , gpu:a100:4 (S:1)
```

```
$ sinfo -N -h -p notchpeak-gpu --Format='nodehost,gresused:50'
```

```
notch001          gpu:v100:2 (IDX:0,2)
notch293          gpu:3090:4 (IDX:0-3) , gpu:a100:4 (IDX:4-7)
```


- Open OnDemand web portal auto-fills available GPUs for each account/partition

Cluster

Select the cluster or Frisco node to create this session on.

Account and partition

Choose the **account:partition** combination appropriate to the cluster chosen above. If in doubt, use the default notchpeak cluster and notchpeak-shared-short account and partition.

click on the GPU type again to show the GPU count input box, the form redraw is broken now in the GE

Advanced options (memory, GPU, nodes)

Check the checkboxes to see the entry options. All advanced options need to be at their defaults for them to hide.

Memory per job in GB

GPU type

none ▾

none

any

Nvidia GTX 1080 Ti, SP, 11GB, general, owner

Nvidia RTX 2080 Ti, SP, 11GB, general, owner

Nvidia RTX 3090, SP, 24GB, general, owner

Nvidia A100, DP, 40GB or 80GB, general, owner

Nvidia A40, SP, 48GB, owner

Nvidia RTX A5500, SP, 24GB, owner

Nvidia RTX A6000, SP, 48GB, owner

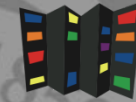
Nvidia L40, SP, 48GB, owner

Nvidia RTX 6000, SP, 48GB, owner

Nvidia T4, SP, 16GB, general, owner

Nvidia V, SP, 12GB, owner

Nvidia V100, DP, 16GB, general, owner



- Special command - `freegpus`

```
$ freegpus -p notchpeak-gpu
```

```
GPUS_FREE: notchpeak-gpu
```

```
v100 (x5)
```

```
2080ti (x6)
```

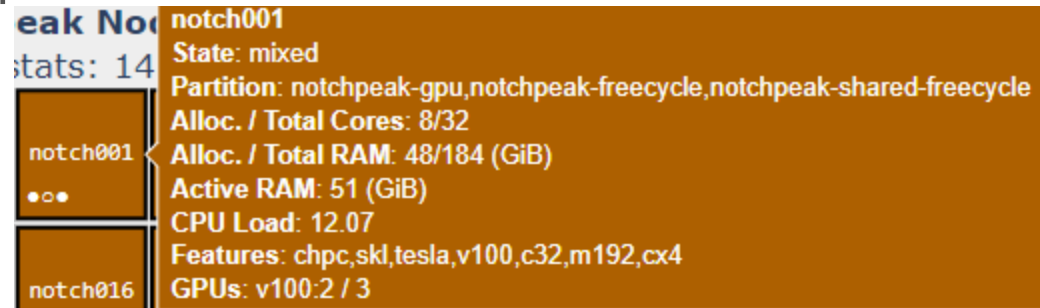
```
3090 (x5)
```

- `x#` is the number of free GPUs in the partition
- `#SBATCH --gres=gpu:v100:1` should get the job started right away
- unless other jobs using GPU(s) on that node use all other resources (CPUs, memory)

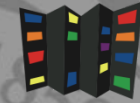
- OnDemand's Node Status App



- empty circle - free GPU, full circle - occupied GPU
- hover with mouse over the node to show more details (incl GPU type)



```
notch001
State: mixed
Partition: notchpeak-gpu,notchpeak-freecycle,notchpeak-shared-freecycle
Alloc. / Total Cores: 8/32
Alloc. / Total RAM: 48/184 (GiB)
Active RAM: 51 (GiB)
CPU Load: 12.07
Features: chpc,skl,tesla,v100,c32,m192,cx4
GPUs: v100:2 / 3
```



- The owner nodes have much more free GPUs than general nodes

```
$ freegpus -p notchpeak-gpu-guest
```

```
GPUS_FREE: notchpeak-gpu-guest
```

```
1080ti (x5)
```

```
2080ti (x26)
```

```
3090 (x8)
```

```
a40 (x6)
```

```
a5500 (x4)
```

```
t4 (x1)
```

```
titanv (x4)
```

```
a6000 (x18)
```

```
a100 (x2)
```

```
l40 (x5)
```

```
rtx6000 (x9)
```

- BUT the job can be preempted - use the owner usage page to correlate the node with free GPUs to the owner partition and to its recent usage

<https://www.chpc.utah.edu/usage/constraints/>



- List of nodes w/ GPUs

```
$ freegpus -p notchpeak-gpu-guest -v
```

```
HOSTNAMES          GRES          GRES_USED
...
notch472           gpu:a40:8 (S:0-1)  gpu:a40:8 (IDX:0-7)
```

- What owner the node has?

```
$ scontrol show node notch472
```

```
...
AvailableFeatures=alhalah,mil,c64,a40,cx6
...
Partitions=notchpeak-gpu-guest,soc-gpu-np,alhalah-gpu-np
...
```

- Notchpeak owner usage

<https://www.chpc.utah.edu/usage/constraints/notchpeak.php>

- find the alhalah feature's recent usage (image so no text search for the features)



- fairly heavily used - may try but there's a good chance for preemption



- Select the Interactive App
- Pick cluster, Account/partition, Advanced options - GPU type and count

Cluster

Select the cluster or Frisco node to create this session on.

 GPU type

Account and partition

- Submit job by clicking 
- Wait till job starts, then click on the Connect to ...

Jupyter (1874654)

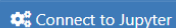
1 node | 1 core | Running

Host: notch309.ipob.int.chpc.utah.edu

Created at: 2024-09-30 16:57:07 MDT

Time Remaining: 3 hours and 59 minutes

Session ID: dc154169-fa8e-4950-a97a-6db7c8dc35b0

Problems with this session? [Submit support ticket](#)



- Interactive

```
salloc -N 1 -n 4 -A notchpeak-shared-short -p notchpeak-  
shared-short --gres=gpu:t4:1 -t 1:00:00
```

- SLURM job script

```
#SBATCH -N 1  
#SBATCH -n 4  
#SBATCH -A notchpeak-shared-short  
#SBATCH -p notchpeak-shared-short  
#SBATCH --gres=gpu:t4:1  
#SBATCH -t 1:00:00
```

- CPU memory

- many GPU programs require substantial CPU (host) memory
- default is 2GB per CPU - often not enough
- you'll see "OOM" and/or "killed" messages in output if that's the case
- ask for more CPUs, or, even better, specify CPU memory:

```
#SBATCH --mem=32G
```



- Check job state

```
$ squeue --me
```

```
JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
1894063 notchpeak interact u0101881 R      0:06      1 notch308
```

```
$ scontrol show job 1894063
```

```
...
```

```
NumNodes=1 NumCPUs=4 NumTasks=4 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
```

```
ReqTRES=cpu=4,mem=16000M,node=1,billing=4,gres/gpu=1
```

```
AllocTRES=cpu=4,mem=16000M,node=1,billing=4,gres/gpu=1
```

```
...
```

```
TresPerNode=gres/gpu:t4:1
```



- Check GPU status (inside job)

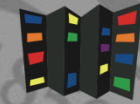
```
$ nvidia-smi
```

```
Wed Oct 2 09:55:03 2024
```

```

+-----+
| NVIDIA-SMI 550.54.14                Driver Version: 550.54.14          CUDA Version: 12.4         |
+-----+-----+-----+-----+-----+-----+
| 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   Tesla V100-PCIE-16GB             On          | 00000000:3B:00.0 Off  |           Off        |
| N/A    33C    P0              36W / 250W | 607MiB / 16384MiB |    11%    Default    |
|                                           |              N/A     |
+-----+-----+-----+-----+-----+
|
| Processes:
| GPU   GI    CI          PID    Type    Process name                        GPU Memory
|       ID    ID                                   | Usage
+-----+-----+-----+-----+-----+-----+
|    0   N/A  N/A      2269935    C     /usr/bin/python                     604MiB
+-----+

```

- Check GPU parameters (inside job)

```
$ ml nvhpc
```

```
$ nvaccelinfo
```

```
CUDA Driver Version:          12040
NVRM version:                NVIDIA UNIX x86_64 Kernel Module  550.54.14  Thu Feb 22 01:44:30
UTC 2024
```

```
Device Number:                0
Device Name:                  Tesla V100-PCIE-16GB
Device Revision Number:       7.0
Global Memory Size:           16928342016
Number of Multiprocessors:     80
Concurrent Copy and Execution: Yes
Total Constant Memory:        65536
Total Shared Memory per Block: 49152
Registers per Block:          65536
Warp Size:                    32
Maximum Threads per Block:    1024
Maximum Block Dimensions:     1024, 1024, 64
Maximum Grid Dimensions:      2147483647 x 65535 x 65535
...
Clock Rate:                   1380 MHz
...
Default Target:               cc70
```



- Check GPU utilization

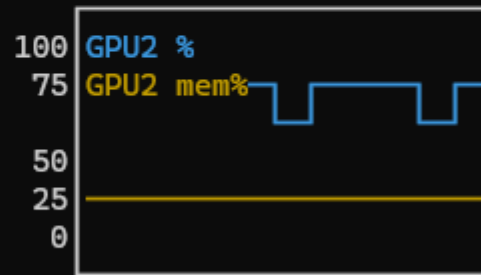
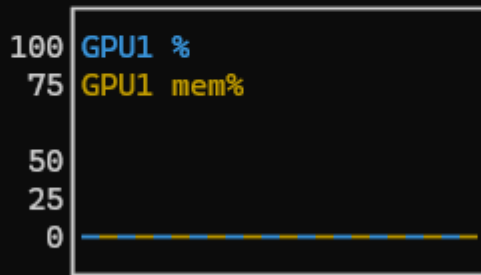
 - ssh to the node with the job, run nvidia-smi

In Open OnDemand:

```
$ ssh notch001
$ ml nvidia-smi
$ nvidia-smi
```

Host: >_ notch308.ipoib.int.chpc.utah.edu

```
Device 0 [Tesla V100-PCI-E-16GB] PCIe GEN 3@16x RX: 47.00 KiB/s TX: 69.00 KiB/s
GPU 1245MHz MEM 877MHz TEMP 33°C FAN N/A% POW 37 / 250 W
GPU[|| 6%] MEM[|| 0.828Gi/16.000Gi]
Device 1 [Tesla V100-PCI-E-16GB] PCIe GEN 3@16x RX: 0.000 KiB/s TX: 0.000 KiB/s
GPU 135MHz MEM 877MHz TEMP 29°C FAN N/A% POW 24 / 250 W
GPU[ 0%] MEM[ 0.234Gi/16.000Gi]
Device 2 [Tesla V100-PCI-E-16GB] PCIe GEN 3@16x RX: 118.0 KiB/s TX: 26.00 KiB/s
GPU 1380MHz MEM 877MHz TEMP 38°C FAN N/A% POW 78 / 250 W
GPU[|||||||||||||||||||| 76%] MEM[||| 1.672Gi/16.000Gi]
```



PID	USER	DEV	TYPE	GPU	GPU MEM	CPU	HOST MEM	Command
2321720	u1470608	2	Compute	76%	1464MiB 9%	100%	3469MiB	python3 seed_run.py
2269935	u1361738	0	Compute	9%	604MiB 4%	564%	3429MiB	/usr/bin/python run_



- Check CPU and host memory utilization

- ssh to the node with the job, run top or htop

In Open OnDemand:

```
$ ssh notch001
```

Host: [>_notch308.ipuib.int.chpc.utah.edu](https://_notch308.ipuib.int.chpc.utah.edu)

```
$ top
```

```
top - 12:58:18 up 26 days, 1:58, 2 users, load average: 7.82, 8.14, 8.01
Tasks: 821 total, 3 running, 818 sleeping, 0 stopped, 0 zombie
%Cpu(s): 10.6 us, 0.1 sy, 0.0 ni, 89.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 191904.0 total, 105313.6 free, 44884.0 used, 41706.4 buff/cache
MiB Swap: 49152.0 total, 49027.3 free, 124.7 used, 144894.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2269935	u1361738	20	0	13.1g	3.5g	414344	S	481.5	1.9	15297:35	pt_main_thread
2321720	u1470608	20	0	11.7g	3.7g	244284	R	101.3	2.0	1034:37	python3
2336952	u1415428	20	0	36.4g	35.0g	87672	R	99.7	18.7	45:06.45	rsession
9609	root	20	0	0	0	0	S	2.6	0.0	689:08.08	nv_queue
2338236	u0101881	20	0	276076	5712	4076	R	0.7	0.0	0:00.07	top
1	root	20	0	240180	11744	8032	S	0.0	0.0	0:39.55	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:04.29	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp

- Check if program recognizes GPU

- Tensorflow

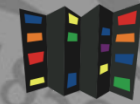
```
$ ml apptainer
$ apptainer pull docker:///tensorflow/tensorflow:latest
$ apptainer exec --nv tensorflow_latest-gpu.sif python -c "from
tensorflow.python.client import device_lib;
print(device_lib.list_local_devices())"
...
incarnation: 6542723605025785350
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14433386496
locality {
  bus_id: 1
  links {
  }
}
incarnation: 11460771345138423495
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:3b:00.0, compute
capability: 7.5"
xla_global_id: 416903419
]
```



- Check if program recognizes GPU

- PyTorch

```
$ ml apptainer
$ apptainer pull docker://pytorch/pytorch:2.4.1-cuda12.1-cudnn9-devel
$ apptainer exec --nv pytorch_2.4.1-cuda12.1-cudnn9-devel.sif python -c
"import torch;print(torch.cuda.device_count())"
1
$ apptainer shell --nv pytorch_2.4.1-cuda12.1-cudnn9-devel.sif
Apptainer> python
Python 3.11.9 | packaged by conda-forge | (main, Apr 19 2024, 18:36:13) [GCC
12.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.cuda.device_count()
1
>>> device_name = torch.cuda.get_device_name(0)
>>> print("Device name:", device_name)
Device name: Tesla T4
```



- Single node
 - ML/AI frameworks should pick up multiple GPUs within a node and use them
 - Other programs may need MPI to run one process per one GPU
 - Consult the program's documentation for how to run
- Multi node
 - Not used much at the CHPC, we don't have too many GPU nodes
 - ML/AI frameworks need a way to communicate - e.g. Horovod library
 - uses MPI or Nvidia's NCCL - contact us if you have a need for this
 - Other programs - should work if they are built with MPI and CUDA, using CUDA aware MPI
 - e.g GROMACS on 2 nodes with 2 GPUs each

```
ml gcc/11.2.0-gpu openmpi/5.0.3-gpu gromacs/2024.2-gpu
mpirun -np 4 gmx_mpi mdrun -gpu_id 01
```



- Be aware of many different GPU types.
- Use `freegpus` to find what GPUs are free
- Use SLURM's `--gres=gpu` to request a GPU
- Ask for fair amount of host memory (SLURM `--mem`)
- Consider using owner GPUs for quick turnaround
- Monitor job's GPU usage with `nvidia-smi` and `nvtop`
- Watch job's output for information about GPU discovery and utilization