Advanced Research Computing Center

UNIVERSITY OF CENTRAL FLORIDA

Newton GPU Cluster Workshop

R. Paul Wiegand
Glenn Martin

Summer 2018

Get the presentation

What is the ARCC?

- Advanced computing facility
  - Intended to facilitate research computing at UCF
  - Subsidized by the Office of the Provost, Office of Research & Commercialization, and the Institute for Simulation & Training
  - We help over 800 users from nearly 200 research groups across almost every college at UCF

- Manage many research computing resources:
  - Stokes high performance computing (HPC) cluster
  - Newton GPU cluster
  - UCF Research network
  - Small staff to help with problems

Types of Jobs on Clusters

- High-Throughput Computing (HTC)
  - Separate “small” multicore jobs
  - E.g.: Multiple replications of stochastic algorithms
  - E.g.: Explicit, external problem decomposition
  - Network communication is less important than memory, CPU, and GPU resources on individual nodes
- High Performance Computing (HPC)
  - Large-scale, distributed jobs
  - E.g.: Problem decomposed in software (FEM methods)
  - Fast, reliable network communication is more important than individual node resources

What is Stokes?

- A mid-sized compute cluster
  - Suited mainly for highly parallel, distributed computation
  - Supports many users concurrently

- Not a single computer, but many computers:
  - 237 compute nodes
  - Older nodes are a mixture of 12-core and 16-core nodes w/ 2 GB of memory per core
  - Newer nodes are a mixture of 28-core and 32-core nodes w/ 128 GB or 192 GB of memory per node
  - All total, a little over 4,396 cores

What is Newton?

- A small GPU-based compute cluster
  - Suited mainly HTC, GPU-based computation
  - Supports some users concurrently
- Not a single computer, but many computers:
  - 10 compute nodes
  - Each node 32 cores and 192 GB of memory per node
  - Each node has two Nvidia V100 GPUs
  - All total 320 cores and 20 GPUs
  - We will be *doubling* this in the next few months
- Newton was supported by your *Technology Fees*!

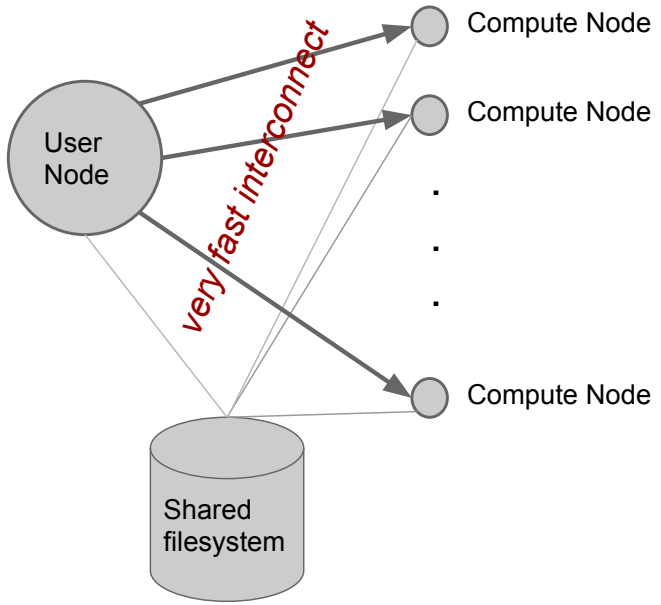
ARCC Clusters, Logical View

Users:

- Log into a *user node*
- Copy data to the user node
- Manipulate data, scripts, and code on the user node
- Submit jobs from the user nodes that will run on *compute nodes*
- Copy the results back when the job is complete

The System:

- Determines when your job can be run
- Decides which compute nodes will be used
- Executes user's job on subset of compute nodes



Managing Expectations

- Writing, debugging, and running parallel and distributed programs on a cluster is harder than dealing with individual, sequential programs on your desktop
- HPC users *must* have some core technical skills:
  - Command-line use of linux
  - Basic understanding of SSH, SCP, etc.
  - Basic understanding and use of job queuing and scheduling
  - Some understanding of basic issues in parallel/distributed computing
- You have to change your workflow to use a cluster
- Think about it more like a *shared printer*

Requesting Resources

- You don't just run a program on an HPC, you *request* resources
- Scheduler / resource manager runs your command or job when those resources are available
- The most important resources to specify:
  - **compute** – how many nodes/cores that your job will need
  - **memory** – how much memory your job will need (/core or /node)
  - **time** – how long your job will run (by "wallclock")
  - **gres** — *generic* additional resource (e.g., a **GPU**)

Requesting Resources

- In general, it takes a little bit of tuning to get these right:
  - Ask for way more than you need $\rightsquigarrow$ wait for longer
  - Ask for too little $\rightsquigarrow$ won't get needed resources:
    - Jobs that need more compute than you request may run very slow
    - Jobs that need more memory than you request may crash
    - Jobs that need more time than you request will be cancelled

What is SSH?

- **Secure Shell (SSH)** – A network protocol for secure data transmission
- Provides command-line access and data file transfer between machines on a network
- Facilitates certain types of graphical access between machines
- Authenticates and establishes identities using *keys*
- Available on all platforms:
  - Linux & Mac OS X: ssh and scp (via OpenSSH)
  - Windows: Putty and WinSCP, or MoabXTerm

Logging into Newton from Unix

Step 1 – Make sure you have OpenSSH installed

Step 2 – Copy all the files we gave you, including the key information somewhere on your machine

Step 3 – Bring up a terminal window

Step 4 – SSH to Stokes:

```
ssh -i <wherever>/<your_username>_id_rsa_1 <your_username>@newton.ist.ucf.edu
```

Step 5 – Enter your passphrase

Logging into Newton from Windows

- Download an SSH tool (e.g., **PuTTY** or **MoabXTerm**)
- Configure an SSH session with the keys we give you
- For instance, we have instructions:

`https://arcc.ist.ucf.edu/index.php/help/tutorials/logging-into-arcc-clusters`

- Open the session you configured, type your passphrase

Activity: Copy Your Key Files

Let's take the next few minutes to copy the files we gave you to your local macine

- Put them somewhere secure
- Put them where you can find them
- On my machine, I put them in `~/.ssh/arcc`

Activity: Log into Newton

Let's take the next few minutes to make sure we can all log in!

```
ssh -i <wherever>/<your_username>_id_rsa_1 <your_username>@newton.ist.ucf.edu
```

OR:

```
https://arcc.ist.ucf.edu/index.php/help/tutorials/logging-into-arcc-clusters
```

Copying Files to/from Stokes in Unix

Step 1 – Bring up a terminal window

Step 2a – SCP a local file to Stokes

```
scp -i --/<your_username>_id_rsa_1 <somefile> <your_username>@newton.ist.ucf.edu:--
```

Step 2b – SCP a file on Stokes to local machine

```
scp -i --/<your_username>_id_rsa_1 <your_username>@newton.ist.ucf.edu:<somefile> --
```

Copying Files to/from Stokes in Windows

- Download an SCP tool (e.g., **WinSCP**)
- Configure an SSH session with the keys we give you
- For instance, we have instructions:

`https://arcc.ist.ucf.edu/index.php/help/tutorials/transferring-files-with-secure-copy`

- Open the session you configured, type your passphrase

Dealing with Files

- Seeing your files:

`ls`	List files
`ls -a`	List all files, including hidden files
`ls -l`	Long list of files (with details)
`ls -la`	Long list all files (details & including hidden)
`ls -lrt`	Long list of files reverse sorted by size

- Wildcards:

`ls works*`	List files starting with “works”
`ls *shop*`	List files with string “shop” in the name
`ls ^w`	List files whose name starts with “w”

- Getting around:

`pwd`	Print the current working directory
`mkdir foo`	Make a directory called “foo”
`cd foo`	Change to directory called “foo”
`cd`	Change to your user’s home directory
`cd ~/foo`	Change to the directory called “foo” right off the user’s home directory
`rmdir foo`	Remove the directory called “foo”

Viewing & Editing Files

- Seeing the content of a text file:

`cat bar`	Display all the contents of the file "bar" to the screen
`less bar`	Show the contents of the file "bar", but pause for page breaks
`head bar`	Just show the first few lines of "bar"
`tail bar`	Just show the last few lines of "bar"

- Edit a text file:

`vi bar`	Edit the file "bar" with the `vi` text editor
`emacs -nw bar`	Edit the file "bar" with the `emacs` text editor
`nano bar`	Edit the file "bar" with the `nano` text editor

- Note that:
  - `vi` and `emacs` are advanced editors
  - They are powerful but hard to learn
  - `nano` is easier to use but not as powerful

Manipulating Files

`cp fromfile tofile`	Copy the file "fromfile" to the file "tofile"
`mv here/fromfile there/.`	Move the file "here/fromfile" to the directory "there"
`mv oldfile newfile`	Rename "oldfile" to be named "newfile"
`rm bar`	Delete the file "bar"

Note that in Linux:

- Directories are separated by a **forward slash**, /
- File and directory names are **case sensitive**
- The . character refers to the current directory
- The .. character refers to the parent directory
- The ~ character refers to your user's home directory
- There *is no undelete*!!

Ownership and Permissions

- Relevant properties of a file/directory:
  - **owner** – the user who has direct ownership of the file or directory
  - **group** – the group to which the file or directory belongs
  - **other** – all the other users that are neither the owner or in the group of the file/directory
- Relevant permissions of a file/directory:
  - **read** – whether or not the file/directory contents can be read
  - **write** – whether or not the file/directory can be written to
  - **execute** – whether a file can be executed, or whether a directory can be *passed through* to reach another file or directory
- That makes nine different permissions properties (read, write, and execute for user, group and other)

Ownership and Permissions, p. 2

- A long directory listing tells you all of the permissions
  - The first character tells you what kind of object it is

    (- means file, d means directory)
  - The next three characters tell you what permissions it has for *user*
  - The next three tell you what permissions it has for *group*
  - The last three characters tell you what permissions it has for *other*
  - For example, the follow string indicates a file that is read, write, and executable by the user, but only readable and writable by the group, and only readable by other:

    `-rwxrw-r--`

- Modify file/directory permissions with `chmod`:

`chmod ug+rw bar`	Make "bar" readable and writable to user and group
`chmod o-wx bar`	Remove write and executable permission to "bar" by others

Your SSH Directory

Don't *ever* change the permissions of your ~/.ssh directory!!!

Module System

- Newton has a lot of different pieces of software, including:
  - Different versions of the same software
  - Different versions of software compiled using different compilers
  - Different versions of software linked to different libraries
  - Etc.
- Getting all these to play nicely together is challenging
- So we use a *module* system to help setup specific software to run correctly
- Use our modules, don't try to run (our) binaries outside of those

Module Commands

`module list`	List all modules you currently have loaded
`module avail`	List all available modules
`module load blah/blah-2.0-gcc-7.1.0`	Load the 2.0 version of “blah“ compiled with GCC v7.1
`module unload blah/blah-2.0-gcc-7.1.0`	Unload the 2.0 version of “blah“ compiled with GCC v7.1
`module purge`	Unload all modules

Relevant Modules for Today

- We will be using three modules today:
  - `gcc/gcc-7.1.0`
  - `cuda/cuda-9.0`
  - `tensorflow/tensorflow-1.6.0`

Activity: Load Some Modules

Let's try to load a module!

1. Log into Newton
2. Type the following:
```
module list
module avail cuda
which nvcc
```
3. We can see we have no modules loaded, what CUDA modules are available, and we have no access to `nvcc`
4. Now type the following:
```
module load cuda/cuda-9.0
module list
which nvcc
```
5. Now we see what have the CUDA 9.0 version loaded and the `nvcc` from that install is available
6. Now clear your modules:
```
module purge
module list
```

What is SLURM?

- Open-source workload manager designed for Linux clusters
- Both resource manager and scheduler, like Torque & Moab
- Designed to be portable, scalable, fault-tolerant, and *simple*
- System becoming widely adopted by HPCs around the country

Why SLURM?

- SLURM is free
- SLURM is easier to configure and maintain than other tools
- SSERCA is standardizing on SLURM
- Developer and administrator communities are active on SLURM support lists
- Most of HPCs on XSEDE now use SLURM

Useful Commands For Gathering Information

Command	Purpose
`sinfo`	Get information about queue states
`squeue`	List contents of the queue
`srun`	Run some command via scheduler
`sbatch`	Submit a script to run via scheduler
`scontrol`	Perform a variety of more detailed operations

Queues in SLURM

- What other systems call a "queue" SLURM calls a **partition**
- We have three partitions:
  - **normal** – the default partition
  - **preemptable** – a partition that doesn't charge against an allocation but on which jobs can be *preempted*
  - **display** – a special partition set aside for a machine in our workroom for display purposes
- Most people don't ever care about the "preemptable" partition and so just ignore this
- Almost all SLURM commands allow you to focus just on the *normal* partition using `-p normal`

Listing Information about Partitions/Queues (`sinfo`)

We can list resources available using the `sinfo` command

```
[pwiegand@evuser1]$ sinfo

PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*         up   infinite     10   idle evc[1-10]
preemptable     up 2-00:00:00     10   idle evc[1-10]
display         up   infinite      1   idle evd1
```

- State of “drain” means *offline*
- State of “down” means *down*
- State of “mix” means some, but not all cores are *allocated*
- State of “alloc” means completely busy
- State of “idle” means all cores are available

Listing Information about Running Jobs (squeue)

We can get information about running jobs using `squeue`

```
[pwiegand@evuser1]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               369    normal submit.s pwiegand  R       0:01      1 evc1
```

- State (ST) of "R" means *running*
- State (ST) of "PD" means *pending*
- State (ST) of "CG" means *completing*

Detailing Information about a Job (`scontrol`)

We can show detailed information about a particular job using **`scontrol show job`**

```
[pwiegand@evuser1]$ scontrol show job 369

JobId=369 JobName=submit.slurm
   UserId=pwiegand(5031) GroupId=pwiegand(5031) MCS_label=N/A
   Priority=793 Nice=0 Account=pwiegand QOS=pwiegand
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:51 TimeLimit=00:15:00 TimeMin=N/A
   SubmitTime=2018-06-19T16:45:47 EligibleTime=2018-06-19T16:45:47
   StartTime=2018-06-19T16:45:47 EndTime=2018-06-19T17:00:47 Deadline=N/A
   ...
   NodeList=evc1
   BatchHost=evc1
   NumNodes=1 NumCPUs=8 NumTasks=1 CPUs/Task=8 ReqB:S:C:T=0:0:*:*
   TRES=cpu=8,mem=47200M,node=1,billing=10,gres/gpu=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=8 MinMemoryCPU=5900M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=gpu:1 Reservation=(null)
   ...
   GresEnforceBind=Yes
```

Detailing Information about a Node (`scontrol`)

We can show detailed information about a particular node using **`scontrol show node`**

```
[pwiegand@evuser1]$ scontrol show node evc1

NodeName=evc1 Arch=x86_64 CoresPerSocket=16
   CPUAlloc=8 CPUErr=0 CPUTot=32 CPULoad=0.01
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=gpu:2
   NodeAddr=ivc1 NodeHostName=evc1 Version=17.11
   OS=Linux 3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017
   RealMemory=191908 AllocMem=47200 FreeMem=176317 Sockets=2 Boards=1
   State=MIXED ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=normal,preemptable
   BootTime=2018-05-24T14:24:01 SlurmdStartTime=2018-06-15T07:34:29
   CfgTRES=cpu=32,mem=191908M,billing=36,gres/gpu=2
   AllocTRES=cpu=8,mem=47200M,gres/gpu=1
   CapWatts=n/a
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Accounting on Newton

- The *wall time* of a job is the time on the clock that the job takes
- DPH ("dedicated processor hour") — the *cpu time* a job takes
  - A job running on 4 nodes, 2 cores per node for 10 hours has consumed $4 \times 2 \times 10 = 80$ DPH
- GPU-hours are accounted by entire GPU (not GPU cores)
  - job that runs on a machine for 100 hours over 8 cores and using both GPUs would consume 800 CPU-hours and 200 GPU-hours
- You are charged for what you *consume*, not what you *request*
- The current base allocation for each PI account is 10,000 CPU hours and 2,000 GPU hours per month
- This may change once we get a sense for general usage patterns
- All users attached to an allocation share those resources

Gathering Information about Account Activity (myusage)

We can get a variety of information about your account, including recent activity via **myusage**

```
[pwiegand@evuser1]$ myusage

Usage for Account pwiegand  on the Newton cluster for  start=2018-06-01T00:00:00  end=2018-07-01T00:00:00
================================================
  CPU Used:  35.4  hours of  10,000.0  (0.4%)
  GPU Used:  4.8  hours of  2,000.0  (0.2%)
================================================
                USER               CPU-Hours   GPU-Hours
                ----               ---------   ---------
            *pwiegand                   35.4   4.8
```

Running Commands Directly (`srun`)

- To simply run a command or program via the workload manager, use `srun`

```
[pwiegand@evuser1]$ srun /usr/bin/hostname
evc7
```

- You can also use `srun` to run the same program in parallel

```
[pwiegand@evuser1]$srun --nodes=3 --ntasks-per-node=2 hostname
evc2
evc1
evc2
evc1
evc3
evc3
```

Useful SLURM switches

Switch	Purpose
`--nodes=`	Number of nodes you are requesting
`--ntasks=`	Number of "tasks" you would like done concurrently (overall)
`--ntasks-per-node=`	Number of tasks requested per node
`--cpus-per-tasks=`	Number of cores requested per task
`--mem=`	Amount of memory per node requested (defaults to MB)
`--mem-per-cpu=`	Amount of memory per core requested (defaults to MB)

Running Commands Directly, Redux

- If you simply ask SLURM for `ntasks`, it will find them wherever it can (possibly across multiple nodes)
- This command runs `hostname` four times in parallel, but allocates 2 cores for each run:

```
[pwiegand@evuser1]$ srun --ntasks=4 --cpus-per-task=2 hostname
evc1
evc1
evc2
evc1
```

Requesting *Generic Resources*

- The GPUs are considered *generic resources*
- If you don't ask for them, you won't get them
- You request them using the `--gres` switch, and you get what you ask per node
- You can request up to *two* GPUs per node, `--gres=gpu:2`
- SLURM is smart enough to try to assign cores and GPUs optimally in terms of where they are on the bus
- But you can *insist* that SLURM only permit GPU and core assignments that are optimal using `--gres-flags=enforce-binding`

Activity: Getting a GPU

Let's take a look at a GPU:

1. Load the CUDA module:
   ```
   module load cuda/cuda-9.0
   ```
2. Now ask SLURM to run `nvidia-smi` on a node with 1 GPU:
   ```
   srun --nodes=1 --gres=gpu:1 --ntasks-per-node=1 nvidia-smi
   ```
3. Note that you may have to wait since there are more participants than GPUs

Activity: Results

- Your output probably looked something like this:

```
Tue Jun 19 17:15:29 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.46                 Driver Version: 390.46                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  Off  | 00000000:3B:00.0 Off |                    0 |
| N/A   34C    P0    37W / 250W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Batch Scripts

Usually, it is better to use a **submit script** than to run directly

- Submit scripts document precisely what you requested
- More easily repeatable
- Easier for us to help you when you have a problem
- Can more easily be given to others
- You don't have to wait at the console for the job to run
- Any switch that works for `srun` works for `sbatch` (almost)

Submitting a Batch Script (`sbatch`)

To submit a script to the scheduler, use `sbatch`

```
[pwiegand@evuser1]$ sbatch whatever-my-script-name-is.slurm
Submitted batch job 372

[pwiegand@evuser1]$ squeue
     JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
       372 normal      PaulEg pwiegand R       0:01      2 evc[1-2]
```

SLURM Submit Script

```
#!/bin/bash
#SBATCH --nodes=1                       # Get one node
#SBATCH --cpus-per-task=2               # Two cores per task
#SBATCH --ntasks=1                      # But only one task
#SBATCH --gres=gpu:1                    # And one GPU
#SBATCH --gres-flags=enforce-binding    # Insist on good CPU/GPU alignment
#SBATCH --time=00:10:00                 # Run for 10 minutes, at most
#SBATCH --job-name=GPU-Example          # Name the job so I can see it in squeue

#SBATCH --mail-type=BEGIN,END,FAIL      # Send me email for various states
#SBATCH --mail-user XXX@YYY.ZZZ         # Use this address

# Load modules
module load cuda/cuda-9.0
./myprogram
```

Batch Job Output

Text output from your job will go to files

- By default, the output stream goes to `slurm-`⟨*jobid*⟩`.out`
- By default, the error stream goes to `slurm-`⟨*jobid*⟩`.err`
- But you can specify these with `--output=` and `--error=`
- You can also *join* these — resist the temptation

Canceling Jobs (scancel)

You can cancel your job using scancel

```
[pwiegand@evuser1]$ sbatch submit.slurm
Submitted batch job 386

[pwiegand@evuser1]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               386    normal submit.s pwiegand  R       0:01      1 evc3
[pwiegand@evuser1]$ scancel 386

srun: Force Terminated job 386

[pwiegand@evuser1]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

Allocating Resources Interactively (srun)

We can request an **interactive session** on compute resources:

```
[pwiegand@evuser1]$ srun --nodes=1 --ntasks-per-node=2 --cpus-per-task=16\
                        --time=00:15:00  --gres=gpu:2  --pty bash

[pwiegand@evc2]$ echo $SLURM_NODELIST, $SLURM_NTASKS, $SLURM_CPUS_ON_NODE
evc2, 2, 4

[pwiegand@evc2]$ nvidia-smi topo -m
GPU0 GPU1 mlx5_0 CPU Affinity
GPU0  X  SYS SYS  0-0,2-2,4-4,6-6,8-8,10-10,12-12,14-14,16-16,18-18,20-20,22-22,24-24,26-26,28-28,30-30
GPU1 SYS  X  NODE 1-1,3-3,5-5,7-7,9-9,11-11,13-13,15-15,17-17,19-19,21-21,23-23,25-25,27-27,29-29,31-31
mlx5_0 SYS NODE  X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing a single PCIe switch
  NV#  = Connection traversing a bonded set of # NVLinks

[pwiegand@evc2]$ exit
```

Activity: Getting an Interactive Session

Let's get a (non-GPU) interactive session!

1. There are a lot of us, so let's keep our request as small as possible
   ```
   srun --ntasks=1 --time=00:10:00  --pty bash
   ```
2. Take a look at who is running:
   ```
   squeue
   ```
3. Try to "see" the GPU now (noting you haven't asked for one):
   ```
   nvidia-smi
   ```
4. Exit your interactive session
   ```
   exit
   ```

Biconjugate Gradient Solver

- The ARCC provides a number of examples on Stokes and Newton at:

  `/groups/arcc/shared/examples/current/`

- In the `cuda` directory there, we have some examples

- Specifically, there is test code that uses C++ and CUDA to solve large, sparse systems via the GPU

- Copy that to your account, then `cd` into that directory:

```
cp -R /groups/arcc/shared/examples/current/cuda/biconjgrad ~/.
cd ~/biconjgrad
```

Directory Contents

`./src/` – Directory where main C++ and CUDA source code is kept

`./inc/` – Directory where C++ and CUDA header code is kept

`./data/` – Directory matrix and loading vectors are kept

- ThermalBarrierCoating
- TurbineBladeStresses

`./obj/` – Directory object files are stored during compilation

`./bin/` – Directory binary executables are built

`./readme.txt` – A textfile explaining this sample

`./Makefile` – The build file for the source

`./submit-simple.slurm` – A simple SLURM batch submit script

`./submit.slurm` – A more general SLURM batch submit script

Building the Source

- Don't build on the user node
- Better to request an interactively node and build there
- You *share* the user node with a lot of people
- We set aggressive `ulimits` on the user node
- You don't need a GPU to build, so:

```
srun  --ntasks=1  --time=00:10:00  --pty bash
module load cuda/cuda-9.0
make
exit
```

Simple Submit Script, Page 1

- Let's take a look at the submit script SLURM directives:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=2
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1
#SBATCH --gres-flags=enforce-binding
#SBATCH --time=00:10:00
#SBATCH --job-name=GPU-Example

# Give this process 1 task (per GPU, but only one GPU), then assign two
# cores per task (so two cores overall).  Then enforce that slurm assigns
# only CPUs that are on the socket closest to the GPU you get.
```

Simple Submit Script, Page 2

- Let's look at how the script prepares to run:

```
# Select between the different Solvers and Problems by changing the
# SOLVER and PROBLEM Variables
SOLVER=biconjgrad
PROBLEM=ThermalBarrierCoating

# Output some preliminaries before we begin
date
echo "Slurm nodes: $SLURM_JOB_NODELIST"
NUM_GPUS=1
echo "You were assigned $NUM_GPUS gpu(s)"

# Load modules
module load cuda/cuda-9.0
module load gcc/gcc-6.2.0

# List the modules that are loaded
module list
```

Simple Submit Script, Page 3

- Finally, let's see how it launches the program:

```
# Have Nvidia tell us the GPU/CPU mapping so we know
nvidia-smi topo -m

# Okay, finally run the darned code
echo
echo Run the solver...
echo
./bin/${SOLVER} data/${PROBLEM}/K.ell\
                data/${PROBLEM}/f.vct\
                data/${PROBLEM}/my-solution.vct\
                0 $NUM_GPUS
echo

# You're done!
echo "Ending script..."
date
```

Activity: Run the Job

Let's run this job:

1. Submit the script to the scheduler:
   `sbatch submit-simple.slurm`
2. Check your job to see when it deploys, runs, then completes
   `squeue -u 'whomai'`
3. Note that you may have to wait since there are more participants than GPUs
4. Note the `jobid`
5. When the job is done, check the output:
   `cat slurm-<jobid>.out`

TensorFlow

- TensorFlow is an open source machine learning framework that supports *deep learning* in *Python*
- It is capable of running on CPUs and/or GPUs
- There are a number of pre-built "*models*" already available in TensorFlow
- And there are a number of example programs, including the standard image recognition benchmark CIFAR-10
- Copy `cifar10` to your account, then `cd` into that directory:

```
cp -R /groups/arcc/shared/examples/current/cuda/tensorflow/cifar10/ ~/.
cd ~/cifar10
```

Directory Contents

`./models/` – Directory where TF CIFAR-10 source is kept

`./out/` – Directory TF will place the output

`./readme.txt` – A textfile explaining this sample

`./Makefile` – The build file for the source

`./submit-simple.slurm` – A simple SLURM batch submit script

`./submit.slurm` – A more general SLURM batch submit script

Simple Submit Script, Page 1

- Let's take a look at the submit script SLURM directives:

```
#!/bin/bash

#SBATCH --time=00:15:00
#SBATCH --cpus-per-task=8
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1
#SBATCH --gres-flags=enforce-binding
#SBATCH --output=cifar10-train-slurm-%J.out

# Give this process 1 task (per GPU, but only one GPU), then
# assign eight 8per task (so 8 cores overall).  Then enforce
# that slurm assigns only CPUs that are on the socket closest
# to the GPU you get.
```

Simple Submit Script, Page 2

- Let's look at how the script prepares to run:

```
# Output some preliminaries before we begin
date
echo "Slurm nodes: $SLURM_JOB_NODELIST"
NUM_GPUS=1
echo "You were assigned $NUM_GPUS gpu(s)"

# Load the TensorFlow module
module load tensorflow/tensorflow-1.6.0

# List the modules that are loaded
module list

# Have Nvidia tell us the GPU/CPU mapping so we know
nvidia-smi topo -m

echo
```

Simple Submit Script, Page 3

- Finally, let's see how it launches the program:

```
# Activate the GPU version of TensorFlow
source activate tensorflow-gpu

# Run TensorFlow
echo
time python models/tutorials/image/cifar10/cifar10_train.py --train_dir=./out  --tmp_dir=./tmp
echo

# You're done!
echo "Ending script..."
date
```

Activity: Run the Job

Let's run this job:

1. Submit the script to the scheduler:
   `sbatch submit-simple.slurm`
2. Check your job to see when it deploys, runs, then completes
   `squeue -u 'whomai'`
3. Note that you may have to wait since there are more participants than GPUs
4. Note the `jobid`
5. When the job is done, check the output:
   `cat cifar10-train-slurm-<jobid>.out`
6. Checkout and logs are now in `./out/`

Our Staff

We have a small staff to help with problems with things like login, submission issues, software installs, etc.

- **R. Paul Wiegand** – part-time co-manager of ARCC resources, full-time faculty
- **Glenn Martin** – part-time co-manager of ARCC resources, full-time faculty
- **Derrick Greenspan** – student staff member
- **Jamie Schnaitter** – student staff member

Website, Email, & Social Media

- Our website (`http://arcc.ist.ucf.edu`) has a lot of information
  - Some tutorials
  - A *node matrix* with some information about recent cluster activity
  - Forms for requesting more resources, obtaining an account, etc.
  - News (e.g., downtimes, etc.)
- We also have a Facebook page for news (`https://www.facebook.com/ucfarcc/`)
- We also have a mail list (`stokes-arcc@listserv.cc.ucf.edu`)

Requesting Help

- Usually it is better not to send us email directly
- Instead, you should use our **ticket tracking system**
- To see how to submit to our ticket system, go here:
  `https://arcc.ist.ucf.edu/index.php/help/help`

Questions & Comments

Advanced Research Computing Center

UNIVERSITY OF CENTRAL FLORIDA

Get the presentation

R. Paul Wiegand
wiegand@ist.ucf.edu
Advanced Research Computing Center
Institute for Simulation & Training
http://arcc.ist.ucf.edu