Working on the NewRiver Cluster

CMDA3634: Computer Science Foundations for Computational Modeling and Data Analytics
22 February 2018

NewRiver is a computing cluster provided by Virginia Tech's Advanced Research Computing facility (ARC). It includes 173 nodes, with three processor types, several memory sizes, and graphical processing units (GPUs). This document and examples are available at:

```
https://secure.hosting.vt.edu/www.arc.vt.edu/class_note/
```

---

1 ARC

The ARC office is in 3050 Torgersen Hall. The ARC home page is at

```
https://secure.hosting.vt.edu/www.arc.vt.edu/
```

Important information is available under the ARC home page menu items:

```
USER INFO
  NEW ARC USERS INFO: an outline of how to get started
  SCHEDULER INTERACTION: how to run jobs through the queues
  OFFICE HOURS: GRA walk-in office hours in 3050 Torgersen
RESOURCES
  COMPUTING SYSTEMS
    NEWRIVER: description of hardware and queues
  SOFTWARE: list of available software
SUBMIT A REQUEST
  HELP: report a problem and ask for help
```

2 The NewRiver Cluster

The individual cores on NewRiver are no faster than those on your laptop. NewRiver's advantages arise because it has a lot more cores, with rapid connections, and methods of combining them in parallel computations:

- a sequential program runs on one core on one node;
- an OpenMP program can be run on some or all cores on a single node;
- an MPI program can be run on some or all cores on one or more nodes;
- a CUDA program can run on one or both GPU's on one or more nodes;
- a large memory program can run on the IvyBridge subcluster;
- roughly 250 software packages are already installed;
- a work scheduler allows many jobs to be submitted for later execution as resources become available.

NewRiver can be thought of as a collection of smaller clusters, which specialize in certain jobs and activities:

- **Login or ETX**: 8 nodes, 2 Haswell processors per node, 12 cores per processor, 256GB memory, and a K1200 graphics card, where users can log in via ssh, transfer files with sftp or scp, submit jobs and retrieve results; also accessible from web-based ETX interface
- **General**: 100 nodes, 2 Haswell processors per node, 12 cores per processor, 128GB memory, for traditional HPC jobs using OpenMP or MPI;
- **Big Data**: 16 nodes, 2 Haswell processors per node, 12 cores per processor, 512GB of memory. 43.2 TB of storage;
- **Visualization**: 8 nodes, 2 Haswell processors per node, 12 cores per processor, 512GB memory, and NVIDIA K80 GPU attached, for remote graphics rendering;

- **Very Large Memory**: 2 nodes, 4 Ivybridge processors per node, 15 cores per processor, 3 TB memory;
- **GPU**: 39 nodes, 2 Broadwell processors per node, 14 cores per processor, 512 GB memory, and 2 NVIDIA K1200 GPUs attached, for CUDA programs;

3 Access

The ARC computer systems are visible to anyone on the Virginia Tech campus-wide network. Off-campus users need to set up a VPN (Virtual Private Network), which involves installing and configuring the **Pulse** software. Instructions are available on the Virginia Tech 4Help website (search for "VPN").

Authorized ARC users log into NewRiver using their Virginia Tech PID and password. An ARC account gives you limited access (the *open_q*) to *all* ARC computer clusters (BlueRidge, Cascades, DragonsTooth, Huckleberry, and NewRiver); the CMDA3634SP18 class allocation gives you full access to all queues on NewRiver.

4 Exercise: SSH + 2FA to Login

Open a terminal on your laptop. Since we will soon be talking to several different computers, I will try to indicate commands you give directly to your laptop by prefacing them with *Laptop:*. Use the **ssh** command to access one of the NewRiver login nodes, *newriver1*, *newriver2*, ..., *newriver8*, with your PID:

```
Laptop: ssh PID@newriver1.arc.vt.edu
```

If you need a connection that allows X-window graphics, then your login requires the **-X** switch:

```
Laptop: ssh -X PID@newriver1.arc.vt.edu
```

Virginia Tech uses Two Factor Authorization (2FA). You must also have the DUO app available on your phone and ready to access, when prompted for your password. There are two ways to handle this:

- enter your password; then your phone will buzz, and you must go to the DUO app and press *Approve* to authorize the login request;
- OR, before typing your password, open the DUO app and press the key symbol to reveal a six digit numeric code. Enter your password, followed immediately by a comma, followed immediately by the numeric code:

  ```
  password: password,123456
  ```

Your terminal window is talking to NewRiver; you've been placed in a home directory on NewRiver, named **/home/PID**, where you can store files, or create directories. Your keyboard commands are sent to NewRiver, executed there, with results sent to your screen, until you break the connection with **logout**.

5 Exercise: Customize Your NewRiver Prompt

Your local computer may have several terminal windows open, and some of them may be connected to NewRiver and others may simply be local. This is an invitation to disaster if you start issuing commands to the wrong system. We can try to avoid confusion by insisting that NewRiver identify itself whenever we are talking to it.

We will need to edit a file on NewRiver. You can use the **vi** or **vim** editor to create or modify the following file that is hidden in your login directory:

```
vi .bashrc
```

Insert the following line into this file:

```
PS1="\h: "
```

After saving the modified file, make your environment change effective immediately:

```
source .bashrc
```

Your prompt should now be something like:

```
nrlogin1:
```

and from now on, every terminal window that is "talking" to NewRiver will include this informative prompt.

6 Exercise: SSH-KEYGEN and SSH-COPY-ID to set up easy access

It can be a pain typing your password and Two Factor Authorization every time you access NewRiver. You can tell NewRiver to "trust" logins from your local machine.

On a Linux/OSX/Unix system, generate a public key for your local system (hitting RETURN repeatedly to choose the default options):

```
ssh-keygen -t rsa
```

and share the public key with the NewRiver:

```
ssh-copy-id PID@newriver1.arc.vt.edu
```

Connecting to NewRiver won't require passwords or 2FA authorization. Check by logging out and back in.

7 Transferring Files

Often, you will have a file on one computer but need it to be copied to another. The **scp** command, short for "Secure Copy", is one way to do this. (You might also be familiar with a similar program, **sftp**.)

It will be easiest if you run **scp** from your laptop, assuming you have a version of Unix available there.

To copy a local file from your laptop to NewRiver:

```
Laptop: scp local_original PID@newriver1.arc.vt.edu:remote_copy
```

To copy a NewRiver file to your laptop:

```
Laptop: scp PID@newriver1.arc.vt.edu:remote_original local_copy
```

If the copied file is to have the same name as the original, then that filename can be abbreviated to **"."**.

8 Exercise: Transfer Files with SCP

I have set up a NewRiver directory of example files as *~burkardt/3634/*. Copy these files:

```
nrlogin1: cp ~burkardt/3634/* .      <- Don't omit the "period" at the end!
```

Let's use **scp** to transfer the file *hello.c* from its NewRiver location to your laptop. Keep your NewRiver terminal window open, but now open a new terminal where you can give some commands to your laptop. Assuming that the file *hello.c* is in your top-level NewRiver directory, the copy command would be:

```
Laptop: scp PID@newriver1.arc.vt.edu:hello.c . <- Don't forget "period"
```

You can also copy files from other people's NewRiver directory to your local system, if they let you. Because my NewRiver directory *3634* and file *quad.c* are world readable, you should be able to copy from my remote directory directly to your laptop:

```
Laptop: scp PID@newriver1.arc.vt.edu:~burkardt/3634/quad.c . <- Don't forget "period"
```

Copying files from your laptop to NewRiver is similar. In order to see this happen, it will be easier if we make a copy of *hello.c* called *goodbye.c*. Then if we transfer this file from your laptop back to NewRiver, it will stand out:

```
Laptop: cp hello.c goodbye.c
Laptop: scp goodbye.c PID@newriver1.arc.vt.edu:
```

Use **ls** to verify that *goodbye.c* was copied to your NewRiver directory.

9 Exercise: Compile and Run on a Login Node

To do anything with a C program that you have written, you need a C compiler. On NewRiver, available compilers include **gcc**, **intel** and **pgi**. On the login node, the **gcc** compiler is automatically available. Let's use that compiler to illustrate how to compile and run a simple program:

```
nrlogin1: gcc -o hello hello.c. <-- compiles hello.c, creates executable "hello"
nrlogin1: ./hello               <-- runs the executable "hello" program
```

We committed a small sin, because we ran a program on the login nodes. This program is small (not much memory) and fast (not much time) so it won't be noticed. In general, it's OK to compile programs on the login nodes, but not OK to run programs there, especially large programs that run a long time. If the system administrators catch you doing this, they will warn you once, *and then they can shut off your access.* Instead of using the login nodes, we will see how to run jobs using the job scheduler.

10 The Job Scheduler

Two NewRiver nodes are login nodes, the rest are *compute nodes*. The compute nodes are controlled by a job scheduler called MOAB. To use the compute nodes, you must send a "batch script" to MOAB that describes the resources you want (time, memory, nodes) and the commands you want to execute. MOAB decides when and where your job will run. Once your job is done, you get back a file of results.

Depending on the task you want to carry out, you may need to specify a particular queue for your job. MOAB allows you to specify the desired queue. On NewRiver, these include:

```
open_q
dev_q
normal_q
p100_dev_q
p100_normal_q
largemem_q
vis_q
```

Anyone with an ARC account can use *open_q*, but it is a "low priority" queue, with limits on time and memory. For your work in this class, you will probably be interested in *dev_q* for OpenMP and MPI, or *p100_dev_q* for CUDA work. To qualify to use these higher priority queues, your batch file must include your class allocation code CMDA3634SP18..

11 A Batch Script for open_q

To compile and run the program in the file *quad.c* through the queue *open_q*, we use a batch script called *quad_open_q.sh*. It consists of three parts:

1. #! line, requesting the **bash** interpreter;
2. #PBS lines, for the scheduler, such as job size and time limits;
3. Unix commands you could have entered interactively.

```
#! /bin/bash

#PBS -l walltime=00:05:00    <-- limit of at most 5 minutes of time
```

```
#PBS -l nodes=1;ppn=1        <-- limit of one node please, and just one core
#PBS -W group_list=newriver  <-- Run job on NewRiver cluster
#PBS -q open_q               <-- Run job in NewRiver low priority queue "open_q"
#PBS -j oe                   <-- please only return one output file, not two

cd $PBS_O_WORKDIR            <-- Start the job in the directory where
                                 this batch file is.

module purge                 <-- Set up the environment.
module load gcc
gcc -o quad quad.c -lm       <-- Compile the program.
./quad 1000                  <-- Run the program with input "1000"
```

Note that the line **cd $PBS_O_WORKDIR** tells MOAB that when this job starts, it should start in the directory from which the batch script was submitted. We put the file *quad.c* in that same directory, and when the job is completed, the output file will appear here as well.

12 Exercise: Run the Batch Script in open_q

Verify that, in your NewRiver directory, you have copies of *quad.c* and *quad_open_q.sh*. Now we are ready to run the job in batch. Because it only asks for a small amount of computer resources, we should expect it to run quickly.

1. submit your job request using the command **qsub quad_open_q.sh**.
2. note the number in the system response, such as *123456.master.cluster*, (I'm pretending it's 123456!).
3. check your directory using the **ls** command.
4. check your job with the command **checkjob -v** *123456*.
5. keep issuing **ls** commands until you see a file called *quad_open_q.sh.o123456*.
6. use **vim**, **cat** or **more** to display *quad.sh.o123456*. You should see an estimate for the integral.

Some other useful commands:

- **qdel** *123456* cancels a job (whether or not it has started running);
- **qstat -u** *PID* shows that status of all your jobs;
- **showstart** *123456* estimates when your job will start running;

13 Exercise: Change the Batch Script to run in dev_q

Test out the steps needed to access other queues:

1. Make a copy of *quad_open_q.sh* called *quad_dev_q.sh*;
2. Edit *quad_dev_q.sh*, perhaps with **vi**;
3. Change #PBS -q open_q to #PBS -q dev_q
4. Insert #PBS -A CMDA3634SP18
5. Submit *quad_dev_q.sh*, using the **qsub** command.
6. Expect a result file *quad_dev_q.sh.o123456*.

In general, all your batch scripts should include the class allocation, and should go to *dev_q* for the near future, and to *p100_dev_q* later on for your CUDA jobs.

14 Nodes and Processors and Cores

NewRiver involves 173 nodes. A node is like a supercharged laptop. It has several processors, a chunk of shared memory, possibly some GPU's. A processor controls a set of 12, 14, 15 cores. A core can be assigned a process to execute.

A simple sequential program can be executed on a single core. Any computation involving more than one core is a parallel computation. A typical NewRiver node has 24 cores, and so it could be running 24 different sequential programs at once.

Any number of cores on a single node can run a single OpenMP program in parallel.

Any number of cores on any number of nodes can run a single MPI program in parallel.

At least one core must be associated with a GPU, to run a CUDA program; it's possible for a single program to involve both GPU's, or to use GPU's on multiple nodes.

We will need to understand these ideas in order to run parallel jobs.

15 Exercise: Run an OpenMP Job on 1 Node, Multiple Cores

OpenMP is a simple parallel programming method that requires shared memory. We can run as many processes together as we like, as long as they are all on the same node. On NewRiver, an OpenMP program can therefore run on between 1 to 24 cores.

Let's say we want to run on 4 cores. Our batch script might be:

```
#! /bin/bash

#PBS -l walltime=00:05:00
#PBS -l nodes=1:ppn=4                      <-- Request 1 node, 4 cores
#PBS -W group_list=newriver
#PBS -q dev_q
#PBS -A CMDA3634SP18
#PBS -j oe

cd $PBS_O_WORKDIR

module purge
module load gcc

gcc -o julia_openmp -fopenmp julia_openmp.c <-- use ‘-fopenmp‘ switch

export OMP_NUM_THREADS=4                    <-- Tell the program to use 4 processes
./julia_openmp

rm julia_openmp
```

The **qsub** command will submit this job, and if it runs successfully, it will create *julia.tga*, a TARGA graphics file. The image can be converted to *JPEG* format:

```
convert julia.tga julia.jpg
```

or, if you logged into NewRiver with X Windows, you can display it immediately:

```
module load ImageMagick
display julia.tga
```

16 The Module Command

The **module** command allows you to temporarily make certain software programs and libraries available. We have already used this command to enable **gcc** and **ImageMagick**, and you will need it for MPI and CUDA jobs as well.

To see what is currenly loaded:

```
nrlogin1: module list
```

To return to the default "empty" module environment:

```
nrlogin1: module purge
```

To inquire about the available versions of software, such as **gcc**:

```
nrlogin1: module spider gcc
```

To get details about a particular version of gcc

```
nrlogin1: module spider gcc/5.2
```

MPI requires loading a compiler and a version of MPI. We could do this in two steps:

```
nrlogin1: module load gcc
nrlogin1: module load openmpi
```

or combine them:

```
nrlogin1: module load gcc openmpi
```

17 Exercise: Run an MPI Program

An MPI program must be compiled and executed using special commands whose names depend on the version of MPI being used. For *openmpi*, these commands are **mpicc** and **mpirun** respectively. The file *quad_mpi.c* is another MPI program. It approximates the integral $I = \int_0^1 \frac{1}{1+x}\,dx = \ln(2) \approx 0.6931471805$. It reads an input value **n** which specifies the number of subintervals to use in the approximation. We'll take this number to be 1,000. The program could be compiled and run as 4 processes:

```
nrlogin1: module load gcc openmpi
nrlogin1: mpicc -o quad_mpi quad_mpi.c
nrlogin1: mpirun -np 4 quad_mpi 1000
```

We want to run this program on the compute nodes. To do so, we just have to modify our job script to specify the combination of nodes and cores we want. Let's suppose we want to use 8 MPI processes, but for some reason, we wish to use 2 nodes, and 4 cores on each node:

```
#! /bin/bash

#PBS -l walltime=00:05:00
#PBS -l nodes=2:ppn=4          <-- 2 nodes x 4 cores = 8 cores
#PBS -W group_list=newriver
#PBS -q dev_q
#PBS -A CMDA3634SP18
#PBS -j oe

cd $PBS_O_WORKDIR

module purge
module load gcc openmpi

mpicc -o quad_mpi quad_mpi.c -lm

mpirun -np 8 ./quad_mpi 1000   <-- run with 8 MPI processes
```

Submit this job using **qsub**.

18 Interactive Jobs

Sometimes you really would prefer interactive access for your computing work, such as for certain graphics operations, or debugging. Simply running your jobs on a login node is a bad idea, because it is shared by many users, is not intended for heavy computational use, and will get you a warning or worse from the system administrator.

Interactive access to 1 compute node, and all 24 cores, for one hour can be requested:

```
interact -A CMDA3634SP18
```

The system will respond with *qsub: waiting for job 14156.master.cluster to start* and there will be a wait, often just 5 or 10 minutes. Once the resources requested are available, the system will say *qsub: job 14156.master.cluster ready* and then show a prompt on a compute node. You can issue commands on the compute node as you would on the login node or any other system. To exit the interactive session, simply type **exit**.

To modify the default behavior, you can add arguments that would normally be included as part of a batch script: Request two hours (nondefault) on one node with all cores (default):

```
interact -l walltime=2:00:00 -A CMDA3634SP18
```

Request one hour (default) on 1 node, 1 core and 1 GPU (nondefault):

```
interact -lnodes=1:ppn=1:gpus=1 -A CMDA3634SP18
```

19 Exercise: Run an Interactive Job

Practice using **interact** for an MPI job:

1. **interact -lnodes=1:ppn=1 -A CMDA3634SP18** *Only ask for 1 core, so others can share node.*
2. **who** *Other classmates may be logged in right now!*
3. **module purge**
4. **module load gcc openmpi**
5. **mpicc -o quad_mpi -o quad_mpi.c -lm**
6. **mpirun -np 1 ./quad_mpi 1000** *Can only use one core today!*
7. **exit**

20 Exercise: Run a CUDA Program

We will not have time to discuss the structure of a GPU, or how to write a CUDA program. I only want to tell you that a few simple changes are needed to use the GPU facilities on NewRiver.

Your CUDA program will be written in an extended version of C, and to indicate this, the file containing the program will have the extension *.cu*. Your CUDA program will need to be compiled by the NVIDIA CUDA compiler, known as **nvcc**, so you will need to load CUDA.

To compile and run a CUDA program on the GPU, your batch script must request access to a GPU, and it must be submitted to one of the two queues *p100_dev_q* or *p100_normal_q*:

```
#! /bin/bash
#
#PBS -l walltime=00:05:00
#PBS -l nodes=1:ppn=1:gpus=1         <-- request GPU
#PBS -W group_list=newriver
#PBS -q p100_dev_q                   <-- request the appropriate queue
#PBS -A CMDA3634SP18                 <-- class allocation

cd $PBS_O_WORKDIR

module purge
module load cuda                     <-- load CUDA

nvcc -o collatz -arch=sm_60 collatz.cu <-- compile

./collatz                            <-- run
```