Take MPI for a Test Drive

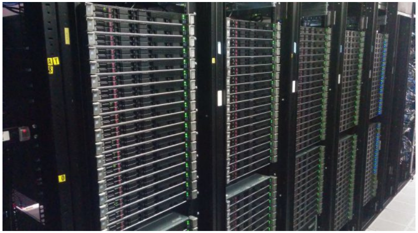
Networked Learning Initiatives Class

John Burkardt: burkardt@vt.edu
Advanced Research Computing
Virginia Tech

10 April 2017

Slides available at:
https://secure.hosting.vt.edu/www.arc.vt.edu/class_note

Take a Test Drive on Our New Model Cluster!



Coverage

Come with me for an hour, and take a test drive on one of ARC's shiny new computer clusters, called NewRiver.

I'll show you how to drive a program that runs in parallel, using MPI.

For our test drive, MPI is all under the hood, making the parallel engine run super-fast, without us worrying about the details of programming.

Taking MPI for granted, we'll concentrate on how you can get access to ARC, what sort of MPI programs are available, how they are run, and how you give them input and collect the output.

Many of our most satisfied customers use the ARC cluster in this way, without having to do any programming.

If you like what you see, you may be able to drive home with an MPI program that solves problems in your research area.

Coverage

This course shows:

- some scientific problems have a natural parallelism;
- multiple programs can cooperate, via MPI;
- many scientific programs already use MPI;
- a cluster organizes many computers to run an MPI program;
- a laptop computation can be transferred to the cluster;;
- how an MPI program can be run, and the results recovered;

This class is not interested in MPI programming itself, only in how to run an MPI program that someone else has already written.

This class assumes little or no knowledge of MPI or programming.

A little experience with Linux, computing on remote computers, or scripting will all be helpful, but is not assumed.

What is Parallel Computing?

A single computer processor has reached a physical speed limit.

To solve problems that are bigger, or to solve them faster, now requires some kind of parallel computing.

To take advantage of parallel computing, a user program must be revised, to identify tasks that can be worked on simultaneously.

The program modifications will depend on the parallel program environment available.

To some extent, this means we can't think only about software and algorithms, but must worry about memory accesses, network bandwidth, and hardware design (multicore machines, multinode clusters, very large shared memory, general purpose GPU's, other hardware accelerators).

MPI has been the top choice for parallel programming for 20 years.

Parallel Programming Choices

- GNU Parallel: execute commands or small scripts on separate computers;
- Posix Threads, ("pthreads"): allows a program to create and control multiple threads of work;
- OpenMP: annotate C/C++/Fortran program for shared memory parallelism;
- MPI: run multiple communicating instances of a program on multiple machines;
- CUDA, OpenCL, OpenACC: allow CPU to offload work onto GPU;
- OpenHMMP: general "heterogeneous" parallel computing (GPU's or other accelerators);
- Hadoop, Spark: parallel processing of big data.

A Perfectly Parallel Problem

Suppose we have received a list of the sequence of chain of amino acids forming each of 10,000 proteins. Each of the 23 amino acids is represented by a unique letter, and each list is **long**.

```
#1:  ANNPERHVWQBBCNZKG...
#2:  WFRGEQRRMYZLAKVSS...
#3: ...
```

The weight of the protein is found by looking up and adding the weight of each amino acid in the sequence.

If we had 10,000 computers, we could do just one problem on each.

We will assume all the results are printed on a single output device, and that we don't care about the order in which this happens.

The programs don't have to start at the same time, nor communicate.

In order to distribute the problems, though, we probably have to specify each file name as input, or assign each computer an ID number that allows it to figure out the problem it is to solve.

Almost Perfectly Parallel Problem

But suppose we need the **maximum** of all the weights we computed.

Then we must wait til all the programs are done, and then we still have a task of scanning the list of weights and finding the maximum.

It would be preferable if the 10,000 programs could somehow bring all their results together and reduce them to the single result we are interested in.

This requires all the programs to run at the same time (synchronization), and be able to send results to a single computer (communication) which determines the maximum.

This is the kind of computation MPI facilitates:

- a big problem is divided into similar subproblems;
- the subproblems are mainly solved on separate computers;
- occasionally, the computers communicate small amounts of data to complete the task.

A Practical Parallel Problem Example

Here is a more realistic problem that MPI can help parallelize.

A sheet of metal is heated by a source field F(x,y,t). We observe the temperature at a MxN array of points at regular time intervals.

Let $T_{i,j,k}$ represent the temperature at location (I, J) and time step K.

A simple model of the process predicts the values at time K+1 from the value at time K of the point (I,J) and its 4 neighbors:

$$\begin{aligned} T_{I,J,K+1} =& T_{I,J,K} + F_{I,J,K} \\ & + \alpha \frac{\Delta t}{\Delta x^2} (T_{I+1,J,K} + T_{I-1,J,K} + T_{I,J+1,K} + T_{I,J-1,K} - 4 * T_{I,J,K}) \end{aligned}$$

where α is the thermal diffusivity (how fast heat spreads in space).

Without parallelism, we simply set up a triple iteration, for all time steps k, for all vertical coordinates j, for all horizontal coordinates i.

Each Process Works on a Subregion; Occasional Messages

So how could we do this in parallel?

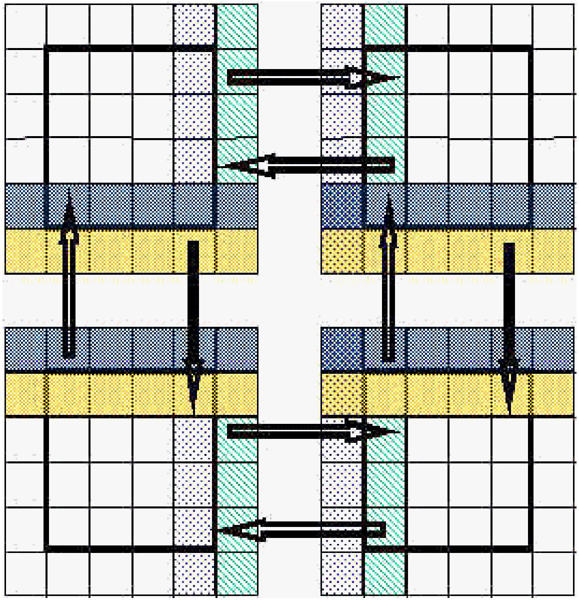
Time is sequential, but space can be split among P computers (or "processes"), and treated in parallel.

A given process P can advance the temperature for every point in its subregion, if that point's 4 neighbors are also available. But that means the points along the boundary have incomplete data.

But MPI allows any process to ask any other process for information; the missing data can be requested, received, and the update done.

As long as the number and size of messages are small, the computation can proceed in parallel at a rate that is almost P times faster.

Ghost Cells: Information on Boundaries Must be Shared



Some Complete MPI Programs Are Ready For You

As you can imagine, it takes some programming effort to identify opportunities for parallelism, and to rewrite a program to use the MPI library in the right way.

In cases where the entire program has already been written, you are excused from having to learn how to program in MPI. In fact, you are even excused from having to learn how to program at all.

So if an MPI program exists for your problem, two tasks remain:

1. how to use the program (what input to give, what to do with output)
2. how to use the program in parallel on a remote cluster.

Item 1 depends on what program you're interested in,
item 2 involves ideas and skills that we can all usefully practice today.

Take a Test Drive on an ARC Cluster

Today we will take an MPI program out for a spin, getting a test drive that allows us to pick up the skills needed to work on a modern computer cluster such as provided by ARC.

We will use canned input files, and won't be too interested in the results.

We will focus on:

- getting access to ARC;
- getting basis information about the program we want to use;
- preparing your input;
- how things relate: laptop, network, login nodes, compute nodes;
- the ways we move around: ssh, sftp, qsub;
- how we configure our computation: job files, module commands;
- MPI: compilers, mpirun command, timing, multiple processes.

How to Get an ARC Account

ARC clusters include BlueRidge, Cascades, DragonsTooth, Huckleberry, NewRiver.

NewRiver is quite suitable for running MPI applications.

Any faculty member or postdoc can apply for an account on any ARC cluster. Students need a faculty member to sponsor their applications.

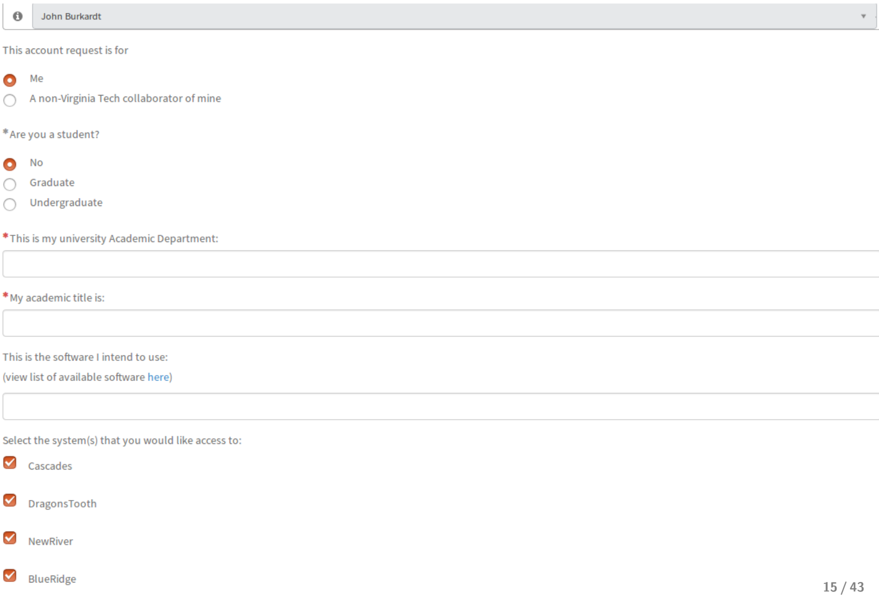
To request an account, you go to the web page:

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

and select the menu choices

USER INFO / Submit a Request / Account.

The Online ARC Account Request Form



John Burkardt

This account request is for

[✓ Me]
A non-Virginia Tech collaborator of mine

*Are you a student?

[✓ No]
Graduate
Undergraduate

*This is my university Academic Department:

*My academic title is:

This is the software I intend to use:
(view list of available software here)

Select the system(s) that you would like access to:

[✓ Cascades]

[✓ DragonsTooth]

[✓ NewRiver]

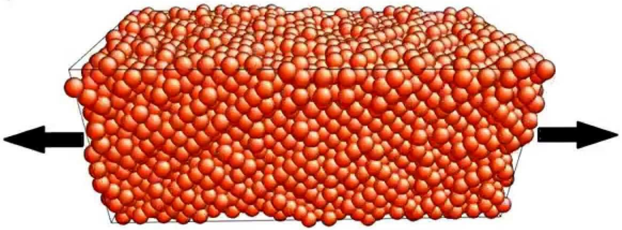
[✓ BlueRidge]

A Little Bit about LAMMPS:

The program we are going to run is called **LAMMPS**; it's a molecular modeling program. It simulates the behavior of thousands of atoms, something like tracking the motion of many billiard balls over time.

For our exercise, all we really need to know is that LAMMPS:

- expects to read a single input file;
- is available on NewRiver;
- can run faster by using MPI.



LAMMPS Problems Are Big:

A LAMMPS computation is a little bit like making a movie. We start with a single frame of data, knowing the positions and velocities of each particle at a starting time.

Each particle experiences an acceleration, or change in velocity, which is the sum of the force exerted on it by every other particle, along with, possibly, the effects of externally applied forces, such as gravity or applied electromagnetic fields.

The inter-particle forces depend on distance, so to compute the force on a particle, we need to know where all the other particles are.

Using the position, velocity, and acceleration, the position and velocity at the next time step can be estimated, so we have our next frame.

A typical calculation involves millions of particles, millions of time steps.

LAMMPS Problems Are Parallelizable:

It is easy to see opportunities for parallelism in LAMMPS.

To compute the next frame of data, the system “stands still” while every particle’s information is updated.

This means that if we have P processors available, we can divide the updates among them, and hope to run faster.

There is a substantial amount of communication, since a processor needs to have updated information about the locations of all the particles, in order to work out the forces to be applied to its subset of particles.

For a very large problem, this communication cost could substantially reduce the benefit of having P processes.

Getting the Example Files

To use an MPI program on the cluster, you typically have an input file of data, and a job control file. Both of these might initially be created on your laptop, with an editor you prefer.

For our MPI test drive, we will just copy an input file and job file to our laptops, and pretend we created them.

I should be able to give you a copy of the files from a USB drive:

- **colloid.in**, an input file;
- **colloid.sh**, a job control file.

These are both text files, so you can type them, or look at them with an editor, search, modify or copy them.

If you can log into NewRiver, then you can copy them from there instead:

```
cp ~burkardt/demo_mpi/* .
```

LAMMPS Can Run on a Laptop

The **lammps** program can be installed on your laptop after downloading it from **http://lammps.sandia.gov**

The program could be run right on your laptop, with a command like

```
lammps < colloid.in
```

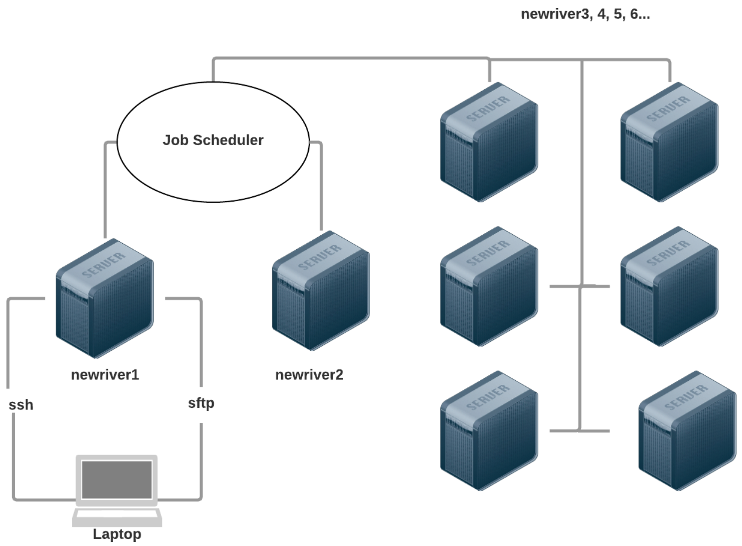
if you happen to be running a version of Linux or Mac OSX.

If your laptop has multiple cores (2, 4, maybe 8) and an MPI compiler, you can even run **lammps** with a mild degree of parallelism.

The reason for running **lammps** on a cluster is to be able to take advantage of much higher degrees of parallelism.

The rest of the class concentrates on making **lammps** < **colloid.in** happen fast on a big cluster.

A Map of You and the NewRiver Cluster



Getting Around on the Cluster

So now let us assume that the program you want to run, which today is **lammps**, is available on the cluster compute nodes.

Your input files are sitting on your laptop, where you probably prefer to do editing and visualization.

The cluster login nodes are a halfway place; you can login there; you can transfer files there; from the login nodes you can send jobs to the compute nodes, and receive their output.

We need to learn how to communicate with the login nodes, and how to manage our work there.

Connect Linux or OSX Laptop to Login Nodes

From Mac OSX, or Linux, you log into your ARC account using **ssh**, and you transfer files back and forth with **sftp**.

You'll probably want to open two terminals on your laptop, one for interaction and one for file transfer.

In the interactive terminal:

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

(The optional -**X** switch ensures that graphics and emacs work correctly.)

In the file transfer terminal, first move to the laptop directory where you have files you want to transfer. Then type:

```
sftp PID@newriver1.arc.vt.edu
```

In both cases, you will be prompted for a password to complete the login. But in the **ssh** window, you'll now be talking directly with the NewRiver node, while in the **sftp** window you'll be talking to the **sftp** program about which files to move.

Connect Windows Laptop to Login Nodes

From Windows, the **putty** program enables logging in and file transfer.

If you don't have it installed already, you can go to

```
http://www.chiark.greenend.org.uk/~sgtatham/putty/
```

putty can be run either by clicking on its icon, or from the Windows command line.

If you are using the graphic interface, then in the dialog box fill in the HostName field with **newriver1.arc.vt.edu**, and the ConnectionType with **ssh** and then choose **Open**,

or, from the command line, type:

```
putty.exe -ssh PID@newriver1.arc.vt.edu
```

From Window Laptop to Login Nodes

The **putty** package includes **psftp**, a file transfer program, which can only be operated from the Windows command line.

To run **psftp**, you need to tell Windows the name of the directory where **putty** and **psftp** are stored. This is done by a command like:

```
set PATH=C:\path to putty directory;%PATH%
```

If you give the location correctly, you can then set up a file transfer session with the command:

```
psftp PID@newriver1.arc.vt.edu
```

EXERCISE 1: Transfer the Example Files to a Login Node

Here's how we might transfer the example files to NewRiver.

The **sftp** session starts in the laptop directory with the example files.

SSH session	SFTP session
ssh -X PID@newriver1.arc.vt.edu	
	sftp PID@newriver1.arc.vt.edu
	mkdir colloid
	cd colloid
	put colloid.sh
	put colloid.in
cd colloid	
ls	
	quit <– Don't do this today!
logout <– Don't do this today!	

Useful ssh Commands

ARC systems use the LINUX operating system; you need a few LINUX commands for any interactive session with **ssh**:

- **ls** to list files;
- **cd** to move to a new directory;
- **cd ..** move "up" one level;
- **pwd** what directory am I in now?;
- **mkdir** to make a new directory;
- **mv** to rename a file;
- **rm** to delete a file;
- **logout** to terminate the **ssh** session.
- **command** run a command or program.
- **command** < **infile** run a command or program with input.
- **command** > **outfile** run a command or program saving output.
- **command** < **infile** > **outfile** use this input, save that output.
- **mpirun -np 4 command** run an MPI program using 4 processes.
- **mpirun -np 4 command** < **infile** same, but with input.

Useful sftp Commands

The **sftp** command can "talk" to both the remote (ARC) system and the local laptop system. This can be confusing! It uses many commands similar to ssh when talking to the remote system. For the local system, the commands are modified with an initial letter L:

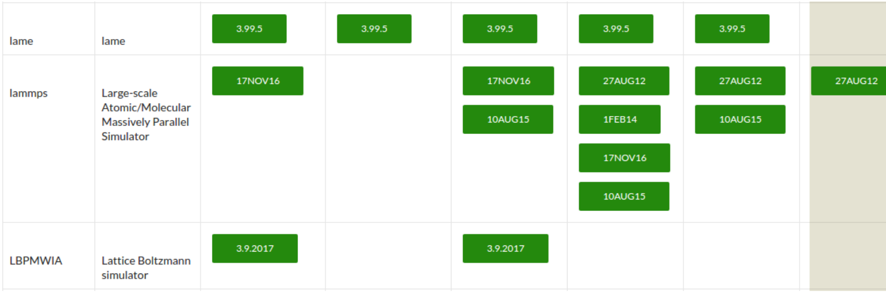
- **put** *file* to copy a local file to the remote system;
- **get** *file* to copy a remote file back to the local system;
- **ls** or **lls** to list files;
- **cd** or **lcd** to move to a new directory;
- **mkdir** or **lmkdir** to make a new directory;
- **pwd** or **lpwd** to show the current location;
- **mv** to rename a file;
- **rm** to delete a file;
- **quit** to terminate the **sftp** session.

Software Installed on ARC Systems

There are about 250 software packages installed on the ARC systems.

A package may be installed on only some systems; several versions may be available on a system.

The current software list is given at
https://secure.hosting.vt.edu/www.arc.vt.edu/software/



lame	lame	3.99.5	3.99.5	3.99.5	3.99.5	3.99.5	
lammps	Large-scale Atomic/Molecular Massively Parallel Simulator	17NOV16		17NOV16 10AUG15	27AUG12 1FEB14 17NOV16 10AUG15	27AUG12 10AUG15	27AUG12
LBPMWIA	Lattice Boltzmann simulator	3.9.2017		3.9.2017			

Some Executable MPI Programs on NewRiver:

- **ABAQUS:** finite element analysis;
- **ABYSS:** de novo biological sequence assembler;
- **ANSYS:** engineering simulation;
- **EXAML:** phylogenetic inference using maximum likelihood;
- **GROMACS:** biomolecular molecular dynamics package;
- **HMMER:** hidden Markov models for biological sequence analysis;
- **LAMMPS:** molecular modeling;
- **LS-DYNA:** general purpose finite element program;
- **MOTHUR_MPI:** bioinformatics data processing;
- **MPIBLAST:** genomic sequence database search;
- **MRBAYES:** Bayesian inference, phylogenetic/evolutionary models;
- **NAMD:** molecular dynamics for large biomolecular systems;
- **OPENFOAM:** fluid dynamics;
- **QUANTUM ESPRESSO:** electronic structure;
- **VASP:** atomic scale material modeling;
- **WANNIER90:** maximally localized Wannier functions;
- **WRF:** weather research and forecasting model.

Hardware: Clusters/Nodes/Processors/Cores

In order to take advantage of MPI, it will be helpful to understand the somewhat complicated structure of a typical computer cluster.

A cluster is, roughly speaking, a connected bunch of stripped-down laptops; but we call them nodes. The ARC cluster called **NewRiver** includes a general subsystem consisting of 100 nodes.

A node consists of processors, and memory shared by the processors. Each NewRiver node has 2 Haswell processors, sharing 128 gigabytes of memory. Nodes are named *newriver1*, *newriver2*, and so on.

Users can ssh to either login node *newriver1* and *newriver2*; the remaining compute nodes are only accessible using a queueing sytem.

Each processor has local memory, and, on NewRiver, 12 Haswell cores.

Each core is an efficient, stripped down computing device.

An MPI computation can use each core as a separate parallel computer.

An MPI job that grabbed all of NewRiver could effectively have $100 * 2 * 12 = 2{,}400$ processes.

Software: MPI “processes”

When we run our program in parallel, we have to say how many copies of the program we want to run. MPI calls each program a process. For a big run of **lammps**, we might want to run tens or hundreds of processes.

We will not have trouble with this issue today, but keep in mind that there is a difference between the software MPI processes (program copies) and the hardware cores that will run the programs.

For large problems, it is necessary to keep these ideas distinct, because the way you request cores depends in part on knowing how many you can get on any one node...

Modules Set up the Environment

Programs installed on NewRiver expect a particular environment, including compilers, the MPI version, and other features.

The **module** command allows a user to specify exactly how the environment should be set up, before a program is run.

When you login, a few modules are loaded (active) by default. If you want a customized environment, you start by clearing out the defaults:

```
module purge
```

then researching the preferred enviroment for your program:

```
module spider lammps
```

and follow this up by a full specification:

```
module spider lammps/10Aug15
```

Module Requirements for LAMMPS

module spider lammps/10Aug15

```
----------------------------------------------------------------------------
lammps: lammps/10Aug15
----------------------------------------------------------------------------
Description:
  Large-scale Atomic/Molecular Massively Parallel Simulator

You will need to load all module(s) on any one of the lines
below before the "lammps/10Aug15" module is available.

  gcc/4.7.2  openmpi/1.6.4
  gcc/4.7.2  openmpi/1.8.5

Help:
  LAMMPS is a classical molecular dynamics code that models
  an ensemble ofparticles in a liquid, solid, or gaseous
  state. It can model atomic, polymeric, biological, metallic,
  granular, and coarse-grained systems using a variety of
  force fields and boundary conditions.
```

followed by stuff we don't care about.

Exercise 2: Check the LAMMPS Module Requirements

Even though we aren't going to run **lammps** interactively, we can go through the motions of setting up the environment with the appropriate module commands. In fact, this is a good test.

Use **ssh** to log into `newriver1`

Issue the following commands:

```
module purge
module load gcc/4.7.2
module load openmpi/1.8.5
module load lammps/10Aug15
```

We get a warning!

```
 Lmod has detected the following error:  Cannot load module
lammps/10Aug15" without these module(s) loaded:
   mkl, cuda, fftw
```

What do you think this means? What do we do?

Exercise 3 (Instructor Only!): Run LAMMPS Interactively

I said we wouldn't run **lammps** interactively. However, our example data is for a small problem, and it's interesting to see how the program works, and even though you should not run jobs on the login nodes, I will risk a short small test for you because I want to make a point.

```
cd colloid          (move to directory containing input file)
module purge
module load gcc/4.7.2
module load openmpi/1.8.5
module load mkl
module load cuda
module load fftw
module load lammps/10Aug15
mpirun -np 4 lmp_mpi < colloid.in
```

On NewRiver, the MPI version of **lammps** is named **lmp_mpi**.

These are almost exactly the user commands we will need to run **lammps** on the compute nodes, with any number of processes.

The Job File User Commands

On the ARC systems, you don't run a big program by typing its name. Instead, you create a job file, which includes the commands you would normally type interactively, plus some information about how the job should be run, including the number of cores to be assigned.

The user commands we just saw that ran **lammps** interactively are part of what I need, but let me make one modification.

Instead of writing **cd colloid**, the job file will use the command:

```
cd $PBS_O_WORKDIR
```

For us, this will end up meaning the same thing, but technically, it says:

"move to the directory from which the job file was just submitted".

The Job File With Scheduler Commands

```
#! /bin/bash                  <-- always like this
#PBS -l walltime=00:05:00     <-- Time limit in HH:MM:SS
#PBS -l nodes=1:ppn=16        <-- Request 16 cores on one node
#PBS -W group_list=newriver <-- always this on NewRiver
#PBS -q open_q                <-- Queue available to all users
#PBS -j oe                    <-- Include errors in output file

  cd $PBS_O_WORKDIR           <-- our user commands begin here
  module purge
  module load gcc/4.7.2
  module load openmpi/1.8.5
  module mkl
  module cuda
  module fftw
  module lammps/10Aug15

  mpirun -np 16 lmp_mpi < colloid.in
```

Using a Job File

Our job file is called *colloid.sh* and it's in the *colloid* directory we created and filled in Exercise 1.

To run the job, we move to this directory, and then issue the command

```
qsub colloid.sh
```

which should result in a response like

178725`.master.cluster`

where 178725 is the jobid, the identification number for the job.

We can use the jobid to check the status:

```
checkjob -v o178725
```

and in particular, when the job is done, the output file will have the name of the job file, followed by the letters ".o" and the jobid:

```
colloid.sh.o178725
```

Exercise 4: Run Your Job

Log into NewRiver, and move to your *colloid* directory and issue the appropriate command to submit the job *(what was that command?)*

Note your jobid, and issue the command that checks on your job's status.

When you see the output file appear in your directory, print out the first few lines. You can do this with a command like

```
  more colloid.sh.o178725    <-- use your jobid, not mine!
```

Your results should begin something like this:

more colloid.sh.o178725

```
    LAMMPS (18 Aug 2015)
    Lattice spacing in x,y,z = 10 10 10
    Created orthogonal box = (0 0 -5) to (300 300 5)
      4 by 4 by 1 MPI processor grid
    Created 900 atoms
    Setting atom values ...
      861 settings made for type/fraction
    Setting atom values ...
      39 settings made for mass
```

Exercise 5: Retrieve the Output to Your Laptop

After a successful run, you want to bring results back to your laptop for reading, printing, analysis, visualization or archiving.

We need **sftp** program, but this time our important command will be **get**.

The output file is in the *colloid* directory, with a name something like *colloid.sh.o123456*. We can figure out the correct name of the file using the **ls** command.

From your laptop, open a terminal, and use **sftp** to connect to NewRiver. *If you did not quit your original* **sftp** *session, you don't need this step.*

Move your remote **sftp** directory to the *colloid* directory. Remember this is done with the **cd** command.

Have **sftp** issue the **ls** command to see what's there. You're looking for the exact name of your output file.

Have **sftp** issue the **get** command to copy the output file to your laptop.

Now quit your **sftp** session.

RECAP: How to Use an MPI Program

Did your output file make it back? Is it in the directory you expected? You should be able to view the output file with any text editor.

I would guess this test drive was somewhat exhausting! But so is learning to drive. Realize that the second time you run **lammps** will be much easier. You might simply change an item in the input file, and everything else would stay the same.

To use other MPI programs, the procedure will be similar:

1. Try the program out on your laptop if possible, and prepare a suitable input file and job file.
2. Use the **module spider** command to find the appropriate modules for this program.
3. Decide how many MPI processes you want to use.
4. Transfer the input and job file to the cluster.
5. Submit your job and then copy results back to laptop.

If You Liked Your MPI Test Drive...

If you want to run MPI programs on your laptop, you need to make sure the MPI library is installed. A quick test is to see what your laptop thinks of the command **mpicc**.

If MPI is installed on your laptop, you can download and run programs like LAMMPS, and see a speed improvement that depends on how many cores your laptop has and how well the program has been parallelized.

To get access to larger MPI systems, you can apply for an ARC account.

If you want to know more about how MPI works, or want to write your own MPI programs, start with the MPI tutorial at **https://computing.llnl.gov/tutorials/mpi/**