Working with HPC Apps

Abhinav Thota
Scientific Applications and Performance Tuning
Research Technologies
Indiana University

July 09, 2014
What is this class about?

- Working with applications on HPC machines
- How do you install applications?
  - Who can install applications?
  - What can be installed?
  - How to choose a compiler?
  - How to choose optimizations and libraries?
- How to use the applications once they are installed?
  - How do you run GPU apps?
  - Dealing with interactive jobs
  - Dealing with GUIs and X-forwarding
- We will then do a lab session to try all of these out on BR 2
What’s so hard about working with HPC apps?

• It’s not, if:
  • you developed the application yourself
  • were involved in the development
  • had training on how to use the application
  • there is really good documentation on the web
• Many of the popular Linux applications are easy to figure out because there is a lot of documentation on the web
• But HPC is a small world and even the popular HPC applications are not as popular as your average windows or mac application, unfortunately
Most apps are 3\textsuperscript{rd} party

- What are 3\textsuperscript{rd} party apps?
  - The software that you or us did not develop
  - The software that did not come with the OS or from Cray
- Why do we care about them?
  - Most of the applications that people run and use are 3\textsuperscript{rd} party
  - Most people don’t develop their own software
  - Could be applications, libraries, compilers, etc.
- How do we deal with them?
  - Will talk about installation, usage, etc.
  - How to get support?
- Discuss examples of CPU and GPU apps
- What do we have at IU?
The Apps and Tools and Libraries we have

• You can take a look on the machines
  OR
• Go to cybergateway.uits.iu.edu and search for apps you are interested in
• You can request software to be installed on one of the machines
• We ask for a justification before we do that
  • Why not install this in your home directory?
  • Are there multiple users who would benefit from a common system wide installation?
  • Is this a particularly difficult application to install?
Examples of software at IU

- NAMD, AMBER, GROMACS, LAMMPS
- WRF, SAS, MATLAB, R
- Compilers
  - GNU, PGI, Intel, Cray
- Not just applications, but tools like debuggers, tracers and libraries such as Boost, NetCDF, Vampir
- OpenMPI, mpich, etc.
- We might have expertise in installing these applications, but we are just installing software developed by someone else most of the time
- We are also not root on the machine
- We are not domain science experts!
Installing software in a Linux Environment

• Obviously, the first step in using an application is installing it
• Who here has installed anything in a Linux/Unix environment?
  • Installing on HPC machines is not much different than installing on your Linux desktop
    - But, not root
    - Can’t install RPMs, can’t install in /usr/bin or /usr/local
    - Must specify non-default location
• We do system wide installs in /N/soft
  • /N/soft/rhel6 – for Quarry and Mason, which run RHEL 6
  • /N/soft/cle4 – for Big Red 2, which runs Cray Linux Environment 4.2
Common procedures

- Compiling is installing
  - gcc main.c gives you a.out
    - a.out is the binary that you just installed
    - More complex applications involve more steps
- Common procedures are:
  - configure, make, make install is the most common and we will see how it works
  - cmake
  - Bjam
  - rpms
  - Binaries
Choosing a compiler

- We have GNU, Intel, PGI and Cray
- What should you pick?
  - No straight answer
  - Depends on the code and CPU make
  - On BR 2, Cray will claim that they have the best compiler, usually true, but not every application builds with the Cray compiler
  - GNU compilers are the most forgiving, but not generally known for performance
  - Most applications build with Intel and PGI and generally Intel and PGI perform well
Configure is the most important step!

- Most applications have a configure step, as in configure make make install
- Other procedures involve a configure step as well, although it may be called something else
- We will see more of this during the lab, but here is a short run down:
  - The make utility comes with Linux, used to maintain groups of programs
  - If you just have one program, you could just do gcc main.c
  - Configure goes through the environment and finds things that the applications needs, tries out different versions and finds something that works
  - Saves you from finding everything that the application needs
- Most widely used and standard applications display available options when you type ./configure --help
Configure options

Some of the common and important options include the following:

- help
- prefix – defines install location
- shared /static – defines shared or static build
- CC, CXX, FC, F77, CFLAGS, etc

The configure help for OpenMPI prints more than 450 lines of options and help information.

After running configure, look at the output to see if the configure step was successful.

No point in proceeding if configure failed.

There could be cases where configure was partially successful:
  - Proceed if the failure is not a deal breaker.
Choose the options now

- Configure is when you choose all the important options and features for an application, for example:
  - Compiler
  - Libraries
  - Optimization
  - Enable optional programs that you want to install that come with the main program
  - For example, enable or disable the GUI feature for an application
    - Enabling the GUI might mean that the application needs additional libraries, which you have to provide if configure can’t find them
  - Configure will find the libraries if they are in your path, if not add them to your path or tell configure where to find them
How to link libraries?

- The make system will do this for you, but if you are building a stand alone application, then do this:
  - gcc main.c -lmath -o main
  - -l assumes that libmath.so or libmath.a is available in your path
- If the libraries are not in a standard location, the you can explicitly specify the library location:
  - gcc main.c -L/path/to/library/location/lib -o main
- Static builds are encouraged on Big Red II, and by default the compilers try to build static binaries for performance reasons
- Include missing header files:
  - gcc main.c -I/path/to/header/files/include -o main
Compiler performance comparison

• It is not uncommon to see 10-20% difference in runtime for the same application built with different compilers, on the same machine
• Depending on the application, usage patterns and the number of users:
  • It makes sense to build the same application with multiple compilers and compare the performance
  • If you are going to use an application to do a large workload over months, saving a month of walltime is not negligible
  • When multiple people are using the same application, it adds up
• We generally try to install applications with the compiler that performs best on that platform
  • Generally Intel on Quarry and Cray/Gnu on BR 2
    - Which means just GNU on BR 2
Optimization

- During compile process, there is usually a way to specify optimization flags
  - Processor specific
  - Easily get 10% speedup with the right optimization flags
  - This adds up cumulatively, when many people run the same application over and over again
  - Man pages will help
  - Again, no guarantee that a higher level of optimization will result in better performance
  - Experiment with different optimizations
- Optimization levels: O1, O2, O3
  - Check man pages for other helpful flags
  - man gcc
  - man icc
make, make test and make install

• After the configure step, you run make
• make tries to compile everything according to the rules that the developer specified
  • usually in a file called Makefile, which is generated by configure
• Some programs have an option to run make test now, which will test the executable
• Then run make install, which will install the program in the location that you specified
• Will demo this in the lab
Compiling GPU applications

• You follow the same steps as you do for CPU applications
• You need additional libraries such as the CUDA toolkit, but –
• You don’t actually need a GPU to build a GPU application
• We do not have GPUs on the login and aprun nodes of BR 2
• Running GPU applications is straightforward as well –
  • Unless you want to use multiple GPUs on multiple nodes
Execution environments on Big Red II (Cray specific, does not concern Quarry or Mason)

• Due to the architecture/design, there are three distinct types of nodes on Big Red II
• Which leads to this, there are two execution modes on Big Red II
  • ESM – Extreme Scalability Mode
  • CCM – Cluster Compatibility Mode
ESM

- The ESM environment is the native execution environment on Big Red II
- It is designed to run large, complex, highly scalable applications, but does not provide the full set of Linux services needed to run standard cluster-based applications
- Need to launch applications with the “aprun” command
- Can run parallel applications with aprun:
  - aprun –n 64 binary_name
The CCM environment provides the Linux services needed to run applications that run on most standard x86_64 cluster-based systems.

The CCM execution environment emulates a Linux-based cluster, allowing you to launch standard applications that won't run in the ESM environment.

Need to launch applications with the “ccmrun” command.

Cannot run parallel applications without mpirun.

For example:
- ccmrun mpirun -np 64 binary_name
Running GPU jobs

- You saw yesterday and the day before how to run CPU jobs
- Is running a GPU job all that different?
- Not really, but it depends on how you want to do it
- If you just want to run a GPU binary, then you do this:
  - aprun –n 1 gpu_binary
  - Most GPU applications need to be run with a single core, which they read as a single GPU
- If you want to run on multiple GPUs:
  - aprun –n 16 –N 1 gpu_binary
  - This is read as 16 GPUs, but only 1 GPU per node
  - Each BR 2 node has only 1 GPU, so
    - aprun –n 16 gpu_binary
    » Will fail and report can’t find enough GPUs
Interactive Jobs

- Why interactive jobs?
  - Testing purposes, where you want to own a node for a brief period, for multiple runs
  - To use interactive features of your application
  - To use the GUI
- You can request interactive nodes on all of IU’s HPC machines
- Just pass “-l” flag to qsub
- Debug queues available on BR 2 and Quarry, for short jobs, interactive or not
- Can request an interactive job from any of the queues
X forwarding

- You need to enable something called X forwarding to interact with the GUI of an application running on a HPC machine
- The X Window system is a software package that is available on all our HPC systems
- Most Linux distributions have X server installed
  - Install Xquartz on Mac OS X
- Just add “-Y” flag to your ssh command when connecting from a Linux/Unix computer
- From a Windows machine
  - Install an X server application, usually Xming or Exceed (from IUWare)
  - Start Xming, and then check the box in Putty that says enable X11
  - Start your session as you always do
Interactive Jobs and GUls

• Now that you have X forwarding setup, how do you use it?
  • Can’t use it on the login node!
  • Except for sessions shorter than 20 mins
• You need to pass “-X” flag to qsub to enable X forwarding
• Once you have the node, on Quarry and Mason, you can start your session
• On Big Red II, you need to login to the compute node before starting your session
Questions?

We will meet in a few minutes to try out the things we talked about.