Putting all pieces together

<table>
<thead>
<tr>
<th>Quick Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC Home</td>
</tr>
<tr>
<td>Getting an account</td>
</tr>
<tr>
<td>Gentle Introduction to using HPC</td>
</tr>
<tr>
<td>Getting started on Prince</td>
</tr>
<tr>
<td>Prince How-to Articles</td>
</tr>
<tr>
<td>Logging in</td>
</tr>
<tr>
<td>Windows</td>
</tr>
<tr>
<td>Mac / Linux</td>
</tr>
<tr>
<td>Clusters and Storage</td>
</tr>
<tr>
<td>Prince (HPC )</td>
</tr>
<tr>
<td>Dumbo (Hadoop)</td>
</tr>
<tr>
<td>Brooklyn (OpenStack)</td>
</tr>
</tbody>
</table>
Dalmaca (NYU Abu Dhabi)

Transferring data to/from the clusters

Transferring data to/from Prince cluster using Globus

Submitting jobs with sbatch

Available software

Licensed Software Available on the HPC Cluster

Building Software

Slurm Tutorial

Tutorials

FAQs

Scratch Area Cleanup

Programming for Biologists
Running jobs on the Prince Cluster

Accessing the Prince Cluster
- From Windows workstation
- From Mac workstation

Software and Environment Module
- Job script and resource request
  - Introduction to job scheduling
  - Submitting jobs with sbatch
  - Requesting resources
Pulling it all together - Preparing, submitting and monitoring a job on Prince

In this section we will prepare, submit and monitor a small Amber (molecular dynamics) job. Our test case comes from the test suite that comes with Amber.

Exercise
Start a terminal session on Prince and replicate this example in it.

Choose your own example
After - or instead of - following this example through, prepare and submit a run of something genuinely relevant to your research. This way, if you are doing this tutorial in a classroom, the presenter will be available should you have questions or strike difficulties

We're using Amber, so first we'll look for available modules. On Prince:
There's quite a few versions there. We'll select a recent one - and purge first to ensure we start from a clean environment

```bash
$ module purge
$ module list
No Modulefiles Currently Loaded.
$ module load amber/openmpi/intel/16.06
```

Take a look at what it did:

```bash
$ module list
Currently Loaded Modulefiles:
  1) centos/7        5) hdf5/intel/1.10.0p1   9) pnetcdf/openmpi/intel/1.6.0
  2) python/intel/2.7.12  6) netcdf/intel/4.4.1.1 10) amber/openmpi/intel/16.06
  3) cuda/8.0.44      7) intel/17.0.1
  4) zlib/intel/1.2.8  8) openmpi/intel/2.0.1
```

... clearly, Amber uses a lot of other packages. The modulefile has looked after loading the correct ones.
$ module show amber/openmpi/intel/16.06
--------
/share/apps/modulefiles/amber/openmpi/intel/16.06.lua:
--------
whatis("Amber: moleculare dynamics package")
whatis("Name: amber version: 16.06 compilers: openmpi/intel")
load("centos/7")
load("python/intel/2.7.12")
load("cuda/8.0.44")
load("netcdf/intel/4.4.1.1")
load("pnetcdf/openmpi/intel/1.6.0")
prepend_path("MANPATH","/share/apps/amber/16.06/openmpi/intel/amber16/share/man")
prepend_path("PYTHONPATH","/share/apps/amber/16.06/openmpi/intel/amber16/lib/python2.7/site-packages")
prepend_path("PATH","/share/apps/amber/16.06/openmpi/intel/amber16/bin")
prepend_path("LD_LIBRARY_PATH","/share/apps/amber/16.06/openmpi/intel/amber16/lib")
prepend_path("PKG_CONFIG_PATH","/share/apps/amber/16.06/openmpi/intel/amber16/lib/pkgconfig")
setenv("AMBER_ROOT","/share/apps/amber/16.06/openmpi/intel/amber16")
setenv("AMBER_HOME","/share/apps/amber/16.06/openmpi/intel/amber16")
setenv("AMBER_INC","/share/apps/amber/16.06/openmpi/intel/amber16/include")
setenv("AMBER_LIB","/share/apps/amber/16.06/openmpi/intel/amber16/lib")
setenv("AMBERHOME","/share/apps/amber/16.06/openmpi/intel/amber16")
family("AMBER")

Loading the module set a useful environment variable, AMBERHOME. We’ll get our test case from there:

```bash
$ ls $AMBERHOME/test/amoeba_jac/
amoeba_jac.ips.mdot/save
amoeba_jac.mdout.save
amoeba_jac.pmemd.mdot.save
inpcrd
inpcrd rst7
prmtop
Run.amoeba_jac
Run.amoeba_jac.ips
Run.amoeba_jac.pmemd
```

We are interested in the files highlighted in blue. Make a directory in your $HOME and copy them there.

Run.amoeba_jac is a script for running the test after building Amber - it’s not exactly what we want, but we’ll take some parts of it. First, we need the namelist file it creates, so we’ll cut-and-paste the lines between “cat > mdin” and “EOF” into a file called “mdin”. (Note the liberal use of shortcuts in the snapshot below. The TAB key is also useful here!)

```bash
$ mkdir $HOME/tutorial-2-ex1
$ cd !$
$ cp $AMBERHOME/test/amoeba_jac/[ip]* .
$ cp $AMBERHOME/test/amoeba_jac/Run.amoeba_jac .
$ cp Run.amoeba_jac mdin
$ vi mdin
# ... and selectively delete the unwanted lines

It should come out looking like this: (don't worry about the blue text just yet)
short md, nve ensemble
&cntl
ntx=1, rest=0,
nstlim=10,
ntpr=1, ntwr=10000,
dt=0.001, vlim=10.0,
cut=8., jfastw=4,
rtt=1, temp=50.0, tempi=0.0,
iamoeba=1,
/  
&ewald
nfft1=80, nfft2=80, nfft3=80,
skinnb=2., nbtell=0, order=5, ew_coeff=0.45,
/  
&amoeba
do_bond=1, do_urey_b=1, do_reg_angle=1, do_trig_angle=1,
do_opbend=1, do_torsion=1, do_pi_torsion=1, do_strbend=1,
do_torsion_torsion=1, do_amoeba_nonbond=1,
dipole_scf_tol = 0.01, dipole_scf_iter_max = 20,
soe_coefficient=0.7, ee_damped_cut=4.5, ee_dsum_cut=6.7,
beeman_integrator=1,
/  

Next we need to write a job script.

What resources will we need?

- It's a serial job, so nodes=1 and ntasks=1
- This test uses about 800MB of memory. We'll request 1GB to be sure
- The test takes less than a minute. We'll request 5 minutes and you can optionally increase the run length (currently 10 fs) by increasing nstlim in the md file (that blue text above).

As for the workflow, the script needs to:

- Load the Environment Module for Amber
- Set up a run directory. We'll run in $SCRATCH, even though this is a very small test
  - **NOTE:** currently $SCRATCH is only defined for login (usually just interactive) shells. We can make any shell a login shell (even if not interactive) by calling bash (or csh) with the -l option. See the first line of the script below.
  - copy the input files there
- Change into the run directory and run the job
  - The command to start this Amber job can be determined from the script Run.amoeba_jac:

```bash
sander -O -i mdin -o amoeba_jac.mdout
```
  - We can get a report at the end of the run on the time and memory resources used by using the "time" command. Here we give it an explicit path because bash has a builtin function called "time", and we want the more powerful command from /usr/bin
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --mem=1GB
#SBATCH --time=5:00

module purge
module load amber/openmpi/intel/16.06

# make a unique directory (by using the PBS job id in its name)
# The ./*/ shortens the job id to the part before the first ".", see
# the "Parameter expansion" section of "man bash" for more
RUNDIR=${SCRATCH}/sbatch_amber_tut.${SLURM_JOB_ID/.*/}
mkdir $RUNDIR

# copy the input data from where we placed in in $HOME to the $RUNDIR
cd $RUNDIR
cp $SLURM_SUBMIT_DIR/inpcrd .
cp $SLURM_SUBMIT_DIR/prmtop .
cp $SLURM_SUBMIT_DIR/mdin .

# the command to start the simulation is:
# sander -O -i mdin -o amoeba_jac.mdout
# we can get timing and resource use information by running a command or program via "time":
# /usr/bin/time -v will report the time and memory used afterwards
/usr/bin/time -v sander -O -i mdin -o amoeba_jac.mdout

Finally, we can submit our job, and monitor its progress:

$ sbatch my_job_script.s

You'll get a job id returned.

Is it running yet?

$ squeue -u $USER

You could watch the output in the run directory:

$ ls -l ${SCRATCH}

Look for a directory whose suffix is the job id. Inside it, a file called amoeba_jac.mdout will be growing, with the output of the Amber run.

Finally, when the job finishes, you should see a slurm-*.out file in the directory you submitted from.

$ ls -lt
-rw-rw-r-- 1 johd johd 766 Jan 20 14:35 slurm-12418.out

Use 'cat' to see what is in the .out file. It's the timing information from /usr/bin/time -v. The job was very short! Try increasing nstlim (as mentioned above) so the job will take at least two or three minutes.

Exercise
Experiment with sbatch options for the job name, output and error file merging and location, resource limits (what if the job exceeds them? You might need to increase nstlim quite a lot to hit a resource limit).