# Our Workshop Environment

John Urbanic
Parallel Computing Specialist
Pittsburgh Supercomputing Center

# Our Environment Today

- Your laptops or workstations: only used for ssh access

- Stampede is our HPC platform

We will here briefly go through the steps to login, edit, compile and run before we get into the real materials

We want to get all of the distractions and local trivia out of the way here. Everything *after* this talk applies to any MPI environment you will encounter.

# Our Environment

**We are using the Stampede system at the Texas Advanced Computing Center**

- 6400 nodes
- 102,400 cores
- 205 TB memory
- 9.5 PFLOP (2.2 from Xeon E5 Sandy Bridge CPUs and 7.3 from Xeon Phi)
- 14 PB disk (76 Lustre IO servers)
- 128 nodes with NVIDIA Kepler K20s

# Getting Connected

From anywhere that you can ssh (using a Putty terminal, for example), ssh with your XSEDE credentials:

```
ssh urbanic@stampede.tacc.utexas.edu
```

## Editors

For editors, we have several options:

- emacs
- vi
- nano : use this if you aren’t familiar with the others

# Compiling

We will be using standard Intel Fortran and C compilers for this session.  The mpi wrappers around them save us a little extra linking typing:

- mpicc for C
- mpif90 for Fortran

## Batch System

No, we don’t just do it to be control freaks

- Parallel machines are big 4D jigsaws that we need to fit together.
- If we didn’t annoy you with this, other users would annoy you more.
- This will be in place on every serious GPU resource that you don’t own.

## SLURM Commands

Only 3 simple commands:

- *sbatch* to submit a job
- *showq* to see how it and others are doing
- *scancel* to delete a job you don’t care about

If you know about PBS, you can easily understand the basic jobfiles that we are using here.  If you don’t, you can probably make out what they are doing anyway, or you can simply just use them obliviously without any loss.

# A SLURM jobscript

We can often get by with only a few simple commands in a jobscript, and you will usually just slightly modify an existing jobscript for your use.  You can get through our entire workshop with just the test.job that we will give you for our first exercise.

```
#!/bin/bash
#SBATCH -A TG-TRA130024     # the account to charge to
#SBATCH -n 4                # total number of mpi tasks requested
#SBATCH -p development      # queue (partition) -- normal, development, etc.
#SBATCH -t 00:05:00         # run time (hh:mm:ss) - 1.5 hours

ibrun ./a.out               # run the MPI executable named a.out
```

SLURM can do much more, but this will suffice for all of our purposes.

# Running a SLURM job

Here is a typical session of running a job and checking it.  Your input in orange.

```
login3$ sbatch test.job
-----------------------------------------------------------------
              Welcome to the Stampede Supercomputer
-----------------------------------------------------------------

--> Verifying valid submit host (login3)...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/00940/urbanic)...OK
--> Verifying availability of your work dir (/work/00940/urbanic)...OK
--> Verifying availability of your scratch dir (/scratch/00940/urbanic)...OK
--> Verifying access to desired queue (development)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (TG-TRA130024)...OK
Submitted batch job 1621590
login3$ showq -u

SUMMARY OF JOBS FOR USER: <urbanic>

ACTIVE JOBS--------------------
JOBID     JOBNAME    USERNAME      STATE   CORE   REMAINING  STARTTIME
================================================================================
1621590   test.job   urbanic       Running 16       0:04:58  Fri Aug 30 02:09:56

WAITING JOBS------------------------
JOBID     JOBNAME    USERNAME      STATE   CORE     WCLIMIT  QUEUETIME
================================================================================

Total Jobs: 1     Active Jobs: 1     Idle Jobs: 0     Blocked Jobs: 0
```

## SLURM Returns Your Results

By default, your job will return its output in a file named with the job name appended by the job number. Something like *slurm-1621590.out*

```
login3$ more slurm-1621589.out
TACC: Starting up job 1621589
TACC: Setting up parallel environment for MVAPICH2+mpispawn.
TACC: Starting parallel tasks...
 Congratulations!

TACC: Shutdown complete. Exiting.
login3$ urbanic$ more test.job.o3255
```

# Batch System Workflow

1. Edit your source files
2. Compile
3. sbatch the (likely same) jobscript (“sbatch test.job”)
4. Wait for your output file (“more slurm-1621589.out”)

OR

## idev

For short development and debugging jobs, most batch system have some way of giving you some “interactive” nodes.  These usually have severe time (30 minute) and size limitations, but for all of our exercises we can live within those.

So, here is our shortcut around the batch system.

# idev

```
login3$ idev

Defaults file    : ~/.idevrc
Default  project : TG-TRA130024
Default  time    : 30 min.
Default  queue   : development
System           : Stampede
-----------------------------------------------------------------
              Welcome to the Stampede Supercomputer
-----------------------------------------------------------------

--> Verifying valid submit host (login3)...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/00940/urbanic)...OK
--> Verifying availability of your work dir (/work/00940/urbanic)...OK
--> Verifying availability of your scratch dir (/scratch/00940/urbanic)...OK
--> Verifying access to desired queue (development)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (TG-TRA130024)...OK
Submitted batch job 1621628

 After your idev job begins to run, a command prompt will appear,
 and you can begin your interactive development session.
 We will report the job status every 4 seconds: (PD=pending, R=running).

job status:  R
--> Job is now running on masternode= c558-302...OK
--> Sleeping for 7 seconds...OK
--> Checking to make sure your job has initialized an env for you....OK
--> Creating interactive terminal session (login) on master node c558-302.
TACC Stampede System
LosF 0.40.0 (Top Notch)
Provisioned on 23-Sep-2012 at 15:48
c558-302$
```

# idev

You may or may not need this ./ in front of your executables. This tells the shell to look in your current directory for your command. That is not always the default.

```
c558-302$
c558-302$ mpif90 test.f
c558-302$ ibrun -np 4 ./a.out
TACC: Starting up job 1621628
TACC: Setting up parallel environment for MVAPICH2+mpispawn.
TACC: Starting parallel tasks...
 Congratulations!

TACC: Shutdown complete. Exiting.
c558-302$
```

By default, your shell prompt has your node name. You can quickly see that you are not on one of the login nodes. You must be in an interactive session.

# idev details

idev defaults to 16 PEs and 30 minutes. Then you will see a message indicating that you are being kicked back into your login node shell. You can alter these parameters with –n (PEs) and –m (minutes) flags. Larger numbers may cause an extended start up time or exceed the limits of the development queue.

If you have multiple accounts, idev may not choose the right default. You must use the “-A account#” option if so.

Recommended practice is to not use the compute nodes you get with idev for anything but running an executable. That might even be enforced here by removing the compilers from the compute nodes. That means you should compile on the login nodes. The practical way to do this while developing is to open one window to the login nodes for compiling and open another for use with an idev session to run.

You may find it convenient and sufficient to just use idev for the entire workshop. Note that the default “development” queue is not set up to accommodate a large workshop. If it gets backed up, you will want to fall back to the “normal” queue:

```
idev –p normal
```

PITTSBURGH SUPERCOMPUTING CENTER

## Our Setup For This Workshop

**After you copy the files from the instructor directory, you will have:**

```
/MPI_Course
       /Test
       /MPI
              laplace_serial.f90/c
              laplace_template.f90/c
              /Solutions
              /Examples
```

# Preliminary Exercise

Let's get the boring stuff out of the way now.

1. Login from your laptop or workstation to Stampede using the procedure described earlier

```
ssh username@stampede.tacc.utexas.edu
```

2. Copy over the training directory

```
cp -r ~train00/MPI_Course .
```

3. Make sure you can edit a file

```
emacs file.c    or    nano file.c
```

4. Test the compiler

```
cd MPI_Course/Test         then
mpicc test.c    or    mpif90 test.f90
```

5. Try idev:

```
idev
ibrun -np 4 a.out
```

When you are done, let us know your coding preference (C or Fortran) with the survey at bit.ly/XSEDE-Workshop

PITTSBURGH SUPERCOMPUTING CENTER