TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

The University of Texas at Austin

Many Core Programming

Stampede2: Skylake and KNL
Hands-on Session

PRESENTED BY:

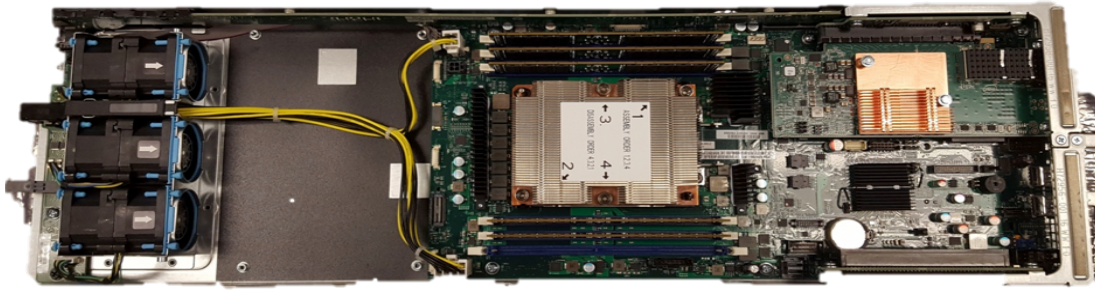
Si Liu
High Performance Computing

Stamede2 at TACC

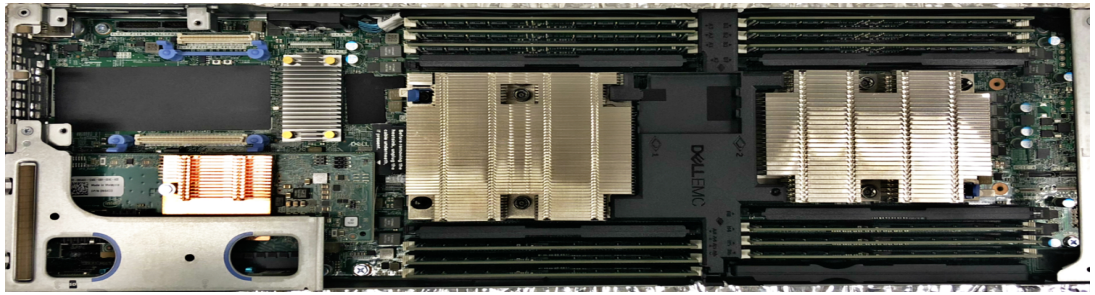
Stampede2
#12 HPC system
in the world
(June, Nov 2017)

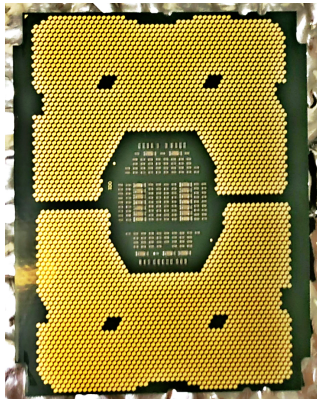
Stampede2

KNL Node



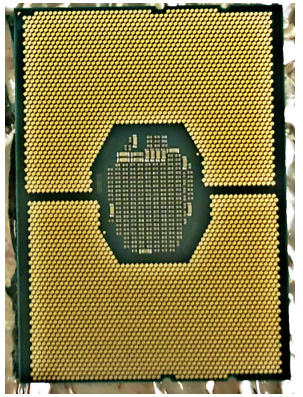
Skylake Node





KNL

Skylake



Stampede2

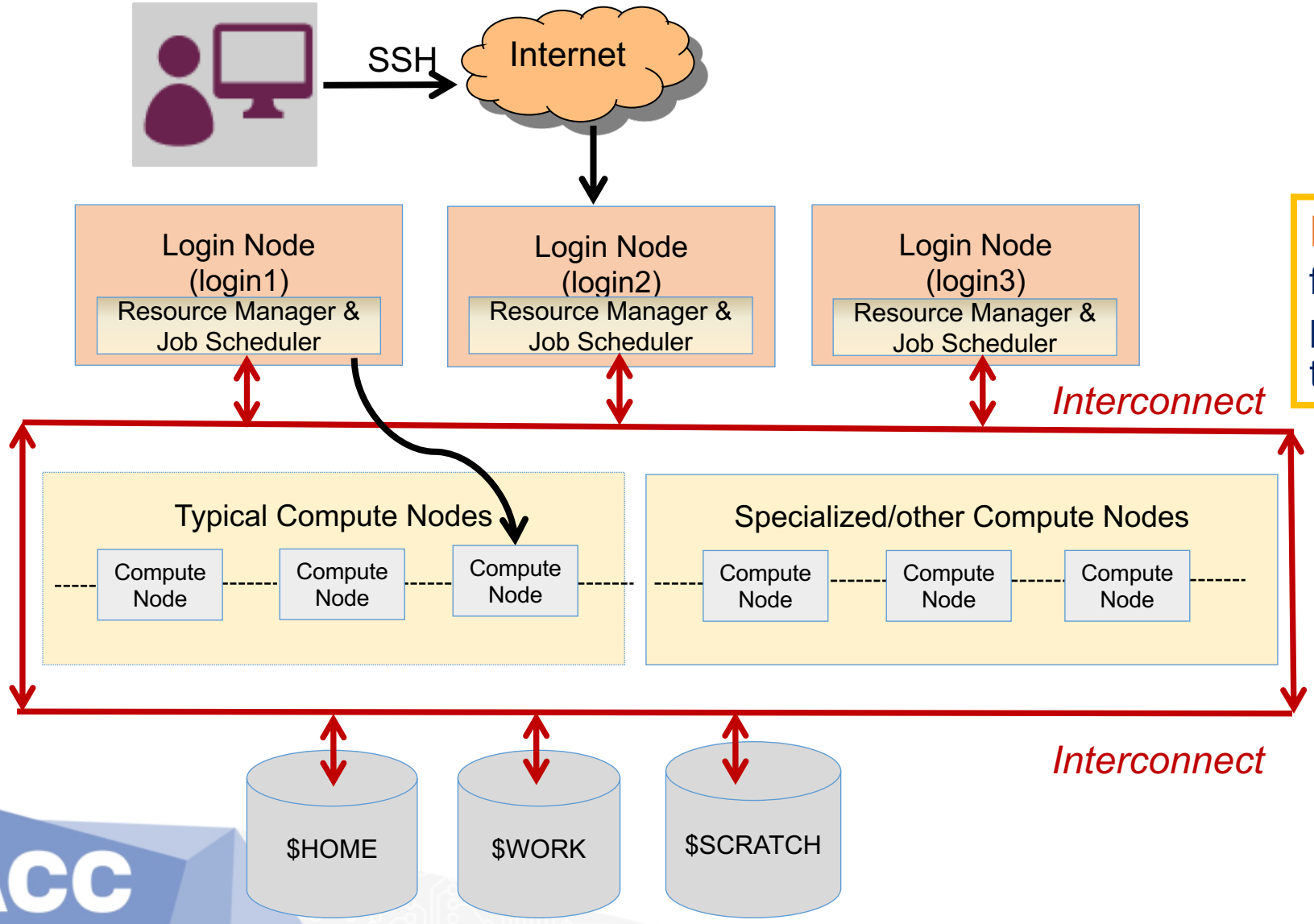
- Peak performance: 18 PF, rank 12 in Top 500 (2017)
- 4,200 68-core Knights Landing (KNL) nodes
- 1,736 48-core Skylake (SKX) nodes
- 368,928 cores and 736,512GB memory in total
- Interconnect: Intel's Omni-Path Fabric Network
- HOME,WORK,SCRATCH Lustre Filesystems
- Large-memory nodes in the near future
- 3D Xpoint non-volatile memory in the future
- Funded by NSF through grant #ACI-1134872

https://www.tacc.utexas.edu/systems/stampede2

Stampede2 Compute Node Specification

	Knights Landing (KNL)	Skylake (SKX)
Model	Intel Xeon Phi 7250	Intel Xeon Platinum 8160
Clock rate	1.4GHz	2.1GHz nominal
Cores per node	68 cores on a single socket	48 cores on two sockets
Hardware threads/core	4	2
Hardware threads/node	4 x 68 = 272	2 x 48 = 96
RAM	96GB DDR4 + 16GB MCDRAM	192GB
Cache	32KB L1 data cache per core 1MB L2 per two-core tile (optional) 16GB MCDRAM	32KB L1 data cache per core 1MB L2 per core 33MB L3 per socket
Local storage	132GB or 58GB	132GB

Stampede2 (Oversimplified Diagram)



Login nodes:
for installing software, compiling programs and requesting access to compute nodes

Compute nodes:
for running jobs in batch or interactive mode

File Systems to Store Data

Access Stampede2

- For Windows users:

  Use client programs on Windows machines like

  Putty:

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

  Cygwin:

  https://cygwin.com/install.html

- For MAC users:

  Use your Terminal application

  After opening the terminal, type the ssh command with your own username (you will be prompted for password after that)

Preparation

1. Login to Stampede2

   $ ssh <username>@stampede2.tacc.utexas.edu

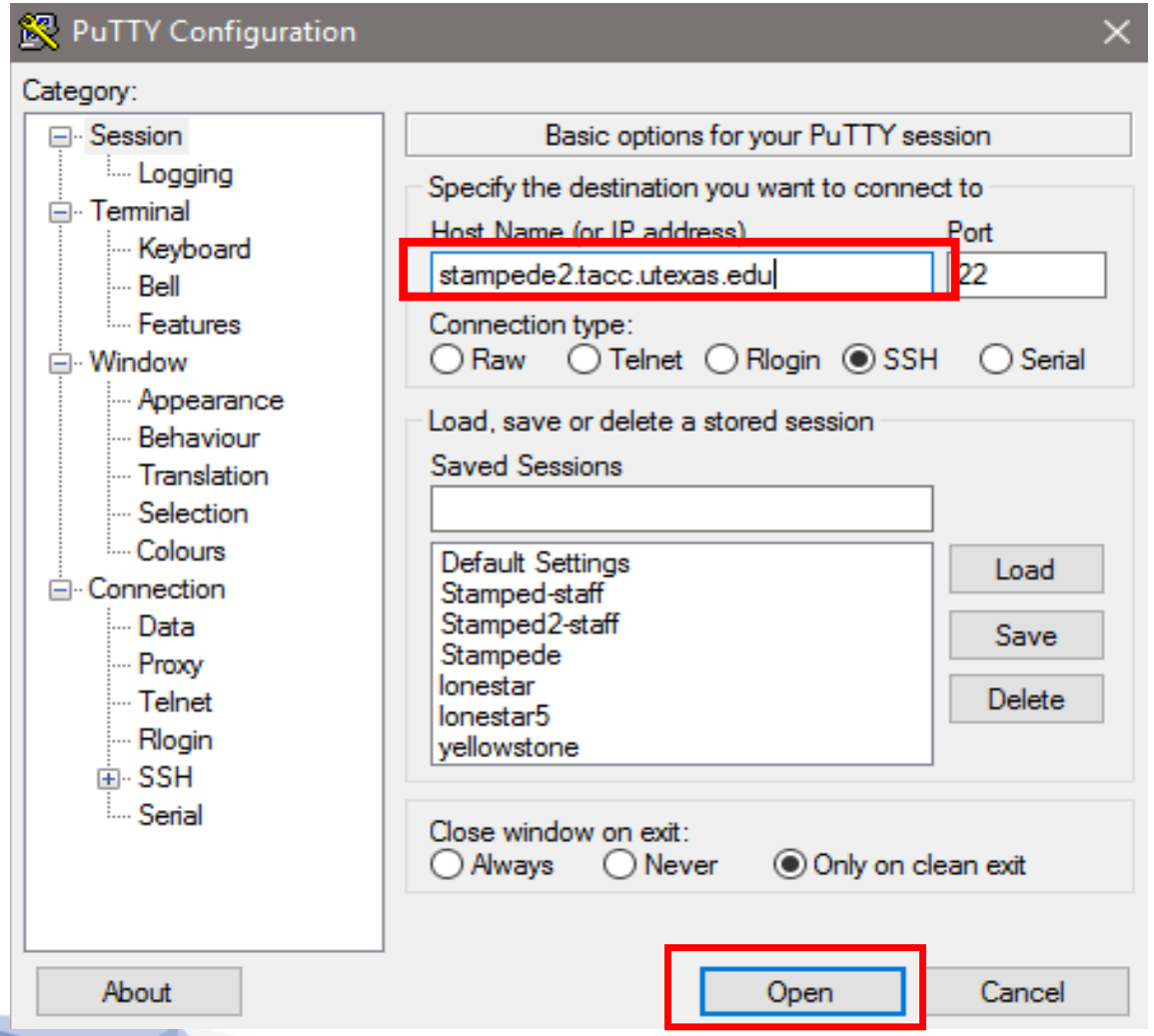
2. Enter your password
3. Extract the lab tarfile

   $ tar -xvf ~train00/lab_knl_intro_2018.tgz

4. Move into the directory

   $ cd lab_knl_intro

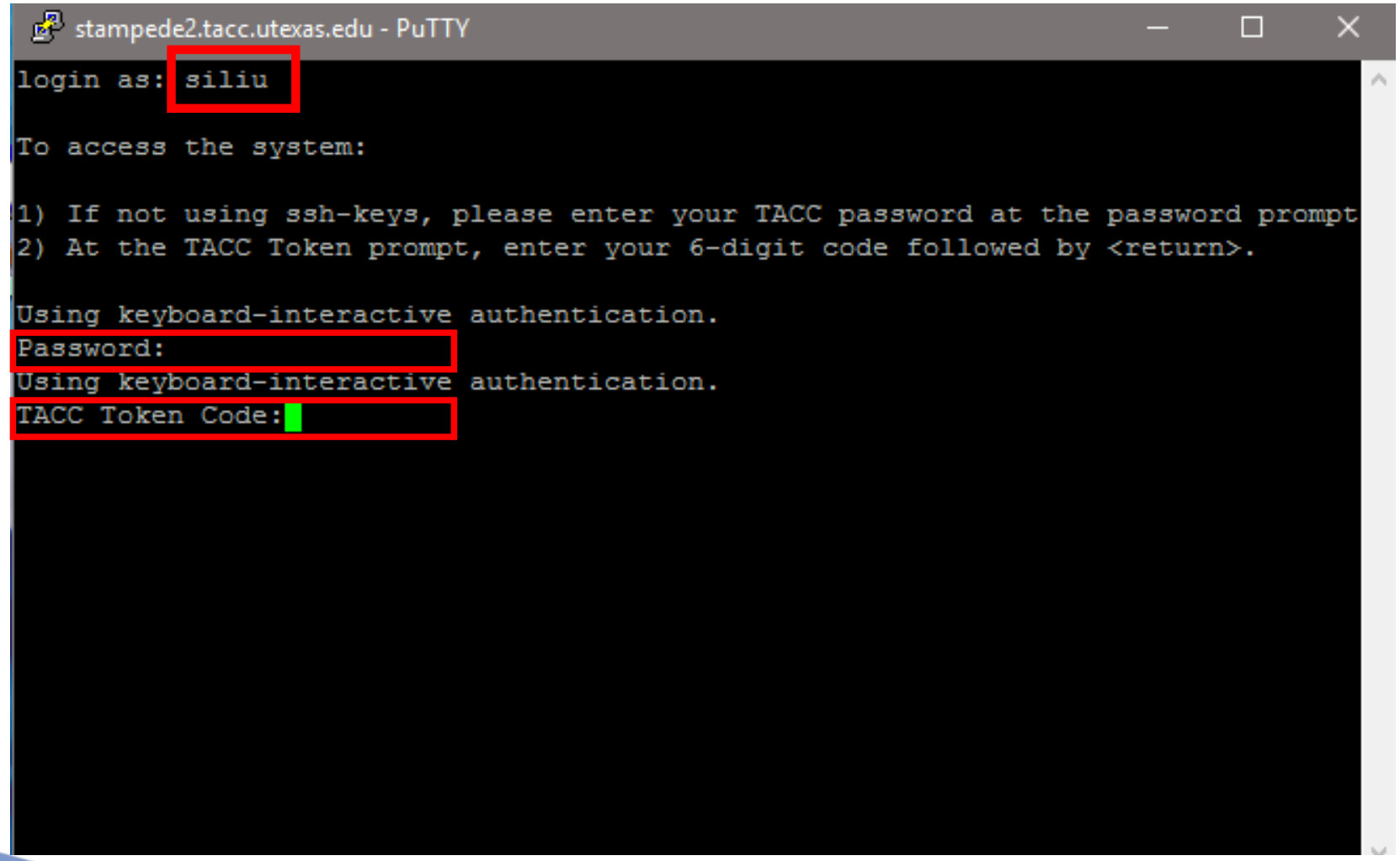
Access Stampede2 Using Putty (1)



Enter Host Name
stampede2.tacc.utexas.edu

Click “Open”

Access Stampede2 Using Putty (2)



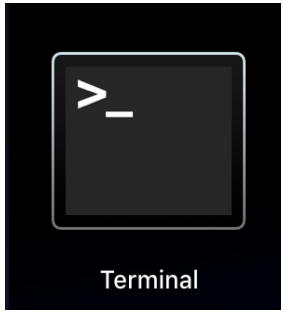
Enter your user name

Enter your Password

Enter your Token

Access Stampede2 from a Mac Machine (1)

- You can have remote access to servers (Stampede2)
  through your “Terminal” application



- After opening the terminal, type the SSH command below after replacing username with the one provided to you
  – you will be prompted for password after that

siliu@mypc> **ssh –X username@stampede2.tacc.utexas.edu**

ssh –l username –X stampede2.tacc.utexas.edu

Access Stampede2 from a Mac Machine (2)

siliu@mypc> ssh -X siliu@stampede2.tacc.utexas.edu

To access the system:

1) If not using ssh-keys, please enter your TACC password at the password prompt

2) At the TACC Token prompt, enter your 6-digit code followed by <return>.

Password:

TACC Token Code:

Last login: Mon Feb 12 14:25:39 2018 from 140.221.145.35

Welcome to the Stampede2 Supercomputer

Texas Advanced Computing Center, The University of Texas at Austin

---

siliu@login1.stampede2.tacc.utexas.edu:~>

File Systems on Stampede2

User-owned storage on a system can be available in different directories

- On Stampede2, three directories that are identified by $HOME, $WORK, and $SCRATCH
- Use cdh, cdw, and cds to access different directories quickly
- These directories are separate Lustre file systems
- They are accessible from any node (login node, compute node) on the Stampede2 system

$HOME	$WORK	$SCRATCH
10 GB quota, maximum 200K files allowed	1 TB quota, maximum 3M files allowed	No Quota Restriction
Backed up	Not backed up	Not backed up
No purge	No purge	Files with no access for a while can be purged
Store your source code and important files	Store large files here Large package/software	Store large files here Productive job, I/O work

Job Scheduling

- TACC supercomputer resources use a job scheduler for running jobs
- Simple Linux Utility For Resource Management (SLURM) Workload Manager
- All jobs are placed in a queue after they are submitted

- If you do not use scheduler or equivalent, you are running on the front end (login) nodes – do NOT do this!
  - Do NOT launch exe (./a.out or ./a.exe) directly on the command line
    One of the easiest way to get your account suspended
  - "make –j 4" may be fine, but not "make –j 64"

Start an Interactive Job

***siliu*@login1.stampede2.tacc.utexas.edu:~>** idev -A A-ccsc -t 1:00:00 -p development -N 1 -n 4

```
 -> Checking on the status of development queue. OK
 -> Defaults file    : ~/.idevrc
 -> System           : stampede2
 -> Queue            : development    (cmd line: -p       )
 -> Nodes            : 1              (cmd line: -N       )
 -> Total tasks      : 4              (cmd line: -n       )
 -> Time (hh:mm:ss)  : 1:00:00        (cmd line: -t       )
 -> Project          : A-ccsc         (cmd line: -A       )

---------------------------------------------------------------------
        Welcome to the Stampede 2 Supercomputer
---------------------------------------------------------------------
No reservation for this job
--> Verifying valid submit host (login2)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/01255/siliu)...OK
--> Verifying availability of your work dir (/work/01255/siliu/stampede2)...OK
--> Verifying availability of your scratch dir (/scratch/01255/siliu)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (development)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (A-ccsc)...OK
Submitted batch job 133511
```

```
-> 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:  PD
 -> job status:  PD
 -> job status:  PD
 -> job status:  PD
 -> job status:  PD
 -> job status:  PD

 -> job status:  R
 -> Job is now running on masternode= c455-073...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 c455-073.

TACC Stampede2 System
Provisioned on 12-Feb-2017 at 11:48

siliu@c455-073 > hostname
c455-073.stampede2.tacc.utexas.edu
```

Script for Submitting a Batch Job

A **sample SLURM job script**, named **myJob.sh**, that can be used for Stampede2

```
#!/bin/bash
#SBATCH -J myMPI          # Job Name
#SBATCH -o myMPI.o%j      # Name of the output file
#SBATCH -e myMPI.e%j      # Name of the error file
#SBATCH -n 64             # Requests 64 tasks/node, 64 cores total
#SBATCH -N 1              # Request 1 compute node
#SBATCH -p normal         # Queue name normal
#SBATCH -t 00:30:00       # Run time (hh:mm:ss) - 30 mins
#SBATCH -A XXXXX.         # Set your account name (XXXXX)

ibrun ./myprogram
```

https://portal.tacc.utexas.edu/user-guides/stampede2#running-sbatch-jobscripts

Submitting, Monitoring, Cancelling a Batch (SLURM) Job

```
Stampede2> sbatch myJob.sh
------------------------------------------------------------
  Welcome to the Stampede 2 Supercomputer
------------------------------------------------------------
No reservation for this job
--> Verifying valid submit host (login3)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/01255/siliu)...OK
--> Verifying availability of your work dir (/work/01255/siliu/stampede2)...OK
--> Verifying availability of your scratch dir (/scratch/01255/siliu)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (normal)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (A-ccsc)...OK
Submitted batch job 123456

Stampede2> squeue -u siliu            #Another command to use: "showq -u"
     JOBID   PARTITION     NAME     USER   ST       TIME  NODES NODELIST(REASON)
    123456  development    myMPI     siliu    R       0:04      1 c401-702

Stampede2> scancel 123456
```

Modules - How to Use Them

- Environment variables make many tasks easy
  - Creating scripts, porting code, running compilers and software
  - Viewing and managing applications, tools, and libraries
- The preferred environment management package is **Modules**
- TACC systems use **LMod**, a Lua based module system

  https://www.tacc.utexas.edu/research-development/tacc-projects/lmod
- Some of the module commands
  - To see what modules have been loaded: `module list`
  - To see what modules are available: `module avail`
  - To load one specific module: `module load matlab`
  - To swap one module with another: `module swap intel gcc`
  - To get help on a module (named foo) : `module help foo`

Explore the System: login nodes

- Login nodes are Intel Xeon Skylake (Gold 6132)

  $ cat /proc/cpuinfo

  $ lscpu

- Take a look at different queues

  $ sinfo -o "%20P %5a %.10l %16F"

- Take a look at queue limits

  $ qlimits

- Queues indicate memory and cluster mode

  (development/normal: Cache-adrant)

- "*development*" or "skx-dev" queue is where you can test codes

Compile Your Programs (C/C++, Fortran)

- Intel is the compiler of choice for Stampede2
  - The default version now is intel/17.0.4 + impi/17.0.3
  - Intel/18.0.0 is available (module load intel/18.0.0)
  - Several versions of gcc are also available

- Compilers available from both login nodes and compute nodes

- Numerous math libraries available, but MKL is especially important (www.intel.com/software/products/mkl)

Fat Binaries (1)

- Login nodes are shared with other users
- Login nodes are NOT for execution!

  Do not run any productive job there!
- Why?

  Compute nodes (KNL and Skylake) are for parallel execution

  Login nodes are multi-purpose (compiling, job scheduling, etc.)
- We can compile for KNL/Skylake architecture on Login nodes
- We can make our code a “fat binary”

  Run most advanced vector instructions on compute nodes

Fat Binaries (2)

- From within lab_knl_intro folder show your present working directory

  $ pwd

- There is a hello world program (hello.c). Compile and run it

  $ icc hello.c -o hello

  $ ./hello

- Try to use the “-xCORE-AVX512” flag: compile and run

  $ icc -xCORE-AVX512 hello.c -o hello

  $ ./hello

- Try to use the “-xMIC-AVX512” flag: compiler and run (What happens?)

  $ icc -xMIC-AVX512 hello.c -o hello

  $ ./hello

Fat Binaries (3)

Intel compiler allows compilation of binaries that work on multiple architecture

- Try to create a fat binary this time
- This is possible with the flags -xCORE-AVX2 -axCORE-AVX512,MIC-AVX512

```
$ icc -xCORE-AVX2 -axCORE-AVX512,MIC-AVX512 hello.c -o hello
$ ./hello
```

- Why did this one run? Where did you run the binary?
- A fat binary generates branches of code at places where instruction sets differ
- To create a binary that works for both KNL and Skylake, use the flag:

  -xCORE-AVX2 -axCORE-AVX512,MIC-AVX512

Explore: Connect to a KNL and a Skylake

"idev" start an interactive session

- Open three terminals on your laptop for your convenience
- Type the idev command

```
$ idev –r intro_manycore_skx        # 1st terminal (SKX)
$ idev –r intro_manycore_knl        # 2nd terminal (KNL-normal)
$ idev –m 60 –p flat-quadrant       # 3rd terminal (KNL-flat-quadrant)
```

- Have a look at the number of cores in SKX and KNL terminals

```
$ cat /proc/cpuinfo
```

- Run "hostname" command if necessary (show the name of compute node)

```
$ hostname
```

- Run the previously created binary inside SKX and KNL terminals

```
$ ./hello
```

Explore: Memory Modes and NUMA Nodes

You can try this one on KNL normal queue (Cache-Quadrant Mode)

$ numactl –H

```
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146
147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262
263 264 265 266 267 268 269 270 271
node 0 size: 98207 MB
node 0 free: 88632 MB
node distances:
node   0
  0:  10
```

Explore: Memory Modes and NUMA Nodes

Try the same command now on KNL flat-quadrant queue (Flat-Quadrant Mode)

```
$ idev –m 60 –p flat-quadrant
$ numactl –H
```

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145
146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174
175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203
204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261
262 263 264 265 266 267 268 269 270 271
node 0 size: 98207 MB
node 0 free: 90304 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15856 MB

node distances:
node   0   1
  0:  10  31
  1:  31  10
```

Explore: Memory Affinity and MCDRAM (1)

DDR and MCDRAM are on separate NUMA nodes in Flat Mode nodes

- DDR is NUMA node 0
- MCDRAM is NUMA node 1
- numactl -m 0 ./executable

  this will pin executable task to NUMA node 0 (DDR)
- numactl –preferred 0 ./executable

  this will pin executable task to NUMA node 0 unless it runs out of memory, then it will fall back to other NUMA nodes
- numactl allows pinning to cores and memory

Explore: Memory Affinity and MCDRAM (2)

Run “stream” on Cache-Quadrant node and look at performance difference

- ***stream*** is the industry standard benchmark for measuring memory bandwidth
- Build stream

  $ icc -qopenmp -O2 -xMIC-AVX512 stream.c -o stream
- Within interactive sessions and set OMP threads to 1:

  $ export OMP_NUM_THREADS=1

  $ ./stream
- Set OMP threads to 68

  $ export OMP_NUM_THREADS=68

  $ ./stream
- Do results make sense to you?

Explore: Memory Affinity and MCDRAM (3)

Now, let us run stream on DDR4 and MCDRAM and look at performance difference

- Within the interactive session of Flat-Quadrant and set OMP threads to 68

  $ export OMP_NUM_THREADS=68

  $ numactl -m 0 ./stream

  $ numactl -m 1 ./stream

- Do results make sense to you?

- Notice DDR4 is slower than cache, but MCDRAM is faster
- Flat mode is better if you know exactly what to put in fast and slow memory

Ex1: Multiprocessing (1)

- idev to a Cache-Quadrant node if you are not already on one

  $ idev -r intro_manycore_knl

  note name of compute node after "idev" starts

  on prompt see: c561-041 [knl] or type "hostname"

- open another terminal on Stampede2

  ssh to stampede2.tacc.utexas.edu then ssh to compute node:

  $ ssh c561-041

- Now both terminals are on same compute node
- Run top in one terminal (you can quit top with q)

  $ top

- top shows running processes ordered by CPU Usage

  Provides memory info as well

Ex1: Multiprocessing (2)

Measuring load: “**Yes**” program

- Let us try the yes program
  - it just prints ’y’ to the screen
  - it uses 100% of one core
- Ensure two terminals open on same compute node as explained in previous slide, make sure top is running in one
- run yes in the background as:

  $ yes > /dev/null &
- What do you see in top?

  quit top (press q)
- Try it on Skylake nodes if time allows

Ex1: Multiprocessing (3)

Measuring load: “Yes” program

- in the “top” terminal, quit “top and run “core_usage”

  $ ./core_usage
- core_usage shows

  every physical core (rows labeled as CORE [0-68])

  every hardware thread on each core (columns labeled as T{1,2,3,4})
- now run another “yes” in the background

  $ yes > /dev/null &
- What do you see?
- Try it on Skylake nodes as well if time allows

Ex1: Multiprocessing (4)

Start some more “yes” runs now…

- Which cores & hardware threads do they run on?

  take a look at core_usage
- Linux load balancer is spreading them out
- KNL might be getting a little hot at this point

  Actually there is hardware that prevents processor from overheating
- How do we stop all of these background processes!?

  $ pkill -f yes

  kills any process that matches the pattern
- observe “top” and “core_usage” now

Ex2: Core affinity

Linux Load Balancer is not perfect, safer to specify by hand

- Ensure “top” is running in one terminal and all “yes” programs are killed

  $ pkill -f yes

- Press 1 (within top) to get into cpu mode
- Run $ numactl -C 0 yes > /dev/null &

  -C core argument lets you specify the core to pin your task to

- Press 1 (within top) to go back to normal mode
- Run another $ numactl -C 0 yes > /dev/null &
- What is the CPU % for each “yes” program?
- Kill “yes” program and try

  $ numactl -C 0 yes > /dev/null &

  $ numactl -C 1 yes > /dev/null &

Ex3: Python Multiprocessing

- Python has a multiprocessing module ($ module load python)
- No time for details but easy way to parallelize some Python code
- Let's try a real code multiproc_mc.py: Simple Monte Carlo Integration

  Integrate x^2 from [0,1] using MC

  Compare serial to parallel runtimes
- python multiproc_mc.py N

  vary N - this is the # of processes to use

  What's the speedup from 1 to 2, 2 to 4, 4 to 16 etc.?

  At some point, granularity is too fine - overhead dominates
- Try with core_usage running in second terminal

  $ numactl -C 0,1,2,3 python multiproc_mc.py 4

Ex4: MKL Multithreading

- MKL will use optimal threading/vectorization for problems automatically!
  - Free parallelization for many “numpy” functions!
- This is true for C and Fortan code that uses MKL as well!
- Set OMP_NUM_THREADS (default is 1)
  - MKL will use up to that number of threads (may use less)
  - export OMP_NUM_THREADS=N
- Try running ./svd.py I(with different numbers of threads
  - SVD is a Singular Value Decomposition implemented in numpy
  - $ python ./svd.py
  - run core_usage concurrently
- MKL will handle affinity and granularity for you
  - Larger matrices can efficiently use more cores

Ex5: Multithreading with OpenMP

- OpenMP has a variety of binding policies. Let's try them out!
- Run core_usage in second terminal
- Compile omp_load (just generates load on cores)

  $ icc omp_load.c -o omp_load -qopenmp
- First lets do default with 34 threads

  $ export OMP_NUM_THREADS=34

  $ ./omp_load
- Try default with 68, 136, 204, 272 threads on KNL
- Repeat with

  OMP_PROC_BIND=spread

  OMP_PROC_BIND=close
- Recall OpenMP Affinity in the early session

License