Getting Started

- Please sign out an RSA token.
  - You will need a photo ID to do this
- The *csep###* written on the envelope will be your username.
- Keep track of the envelope the token comes in
- Once you have the token, please follow the instructions that come with it to set up a 4 to 8 digit PIN. **Remember your PIN**
- If you don't have Putty or Terminal on your laptop please use one of our computers at the back of the room to set the PIN.

- **Please give us back your token in its envelope and sign it back in at the end.**

OAK RIDGE
National Laboratory

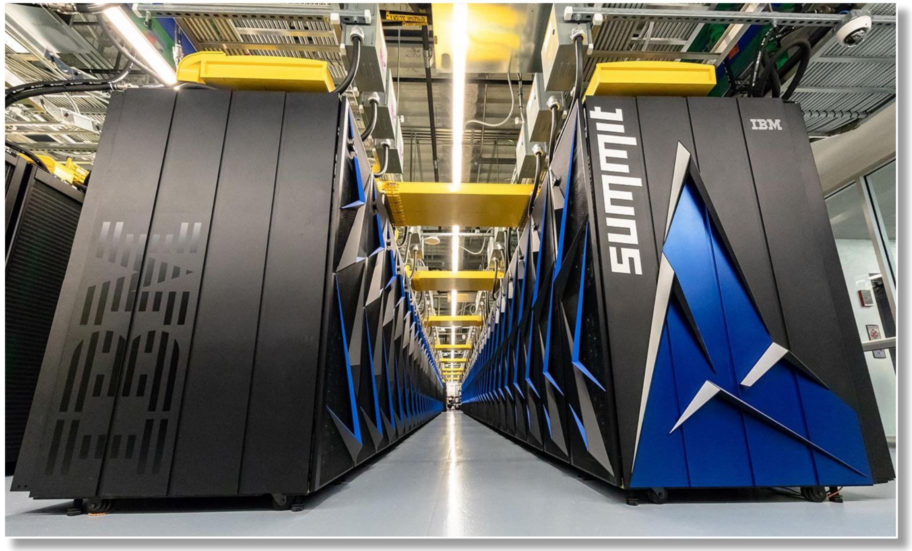
Introduction High Performance Computing

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF
ENERGY

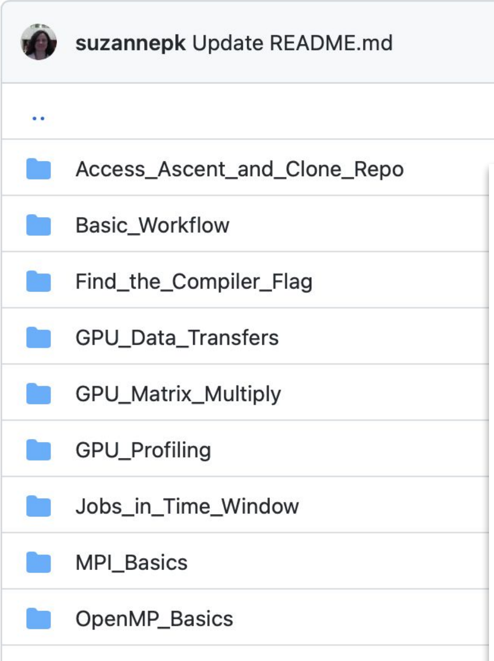
What you will learn

- You will run on the 4th most powerful supercomputer in the world.
- You will be introduced to what High Performance Computing is and why it is useful
- You'll use the Unix command line and a text editor
- You'll be introduced to techniques for distributing work over the different processors and pools of memory
  - OpenMP (Shared memory parallelism)
  - MPI (Distributed memory parallelism)
- You'll get access to a Git repo and resources that will allow you to continue learning after this course.



How you will learn it

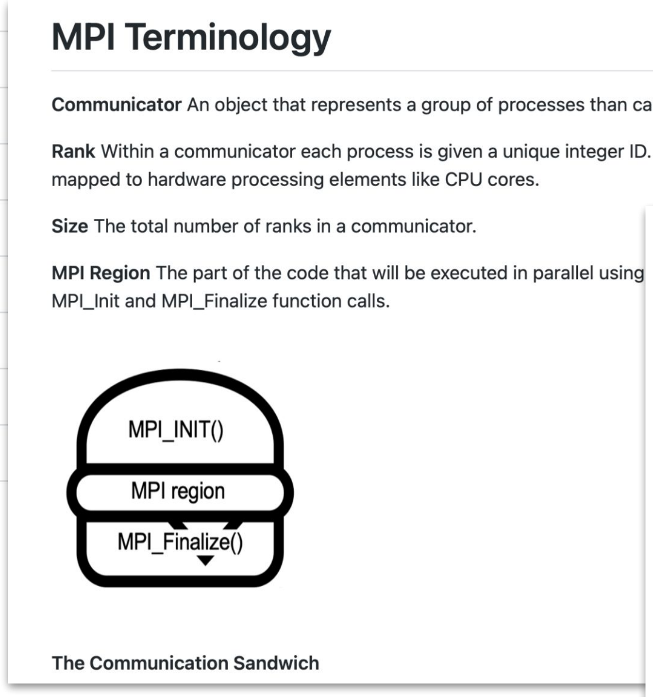
- Hands on!
  - We have a set of workbooks that guide you through each exercise and we will be here to answer questions



suzannepk Update README.md

..

- Access_Ascent_and_Clone_Repo
- Basic_Workflow
- Find_the_Compiler_Flag
- GPU_Data_Transfers
- GPU_Matrix_Multiply
- GPU_Profiling
- Jobs_in_Time_Window
- MPI_Basics
- OpenMP_Basics



MPI Terminology

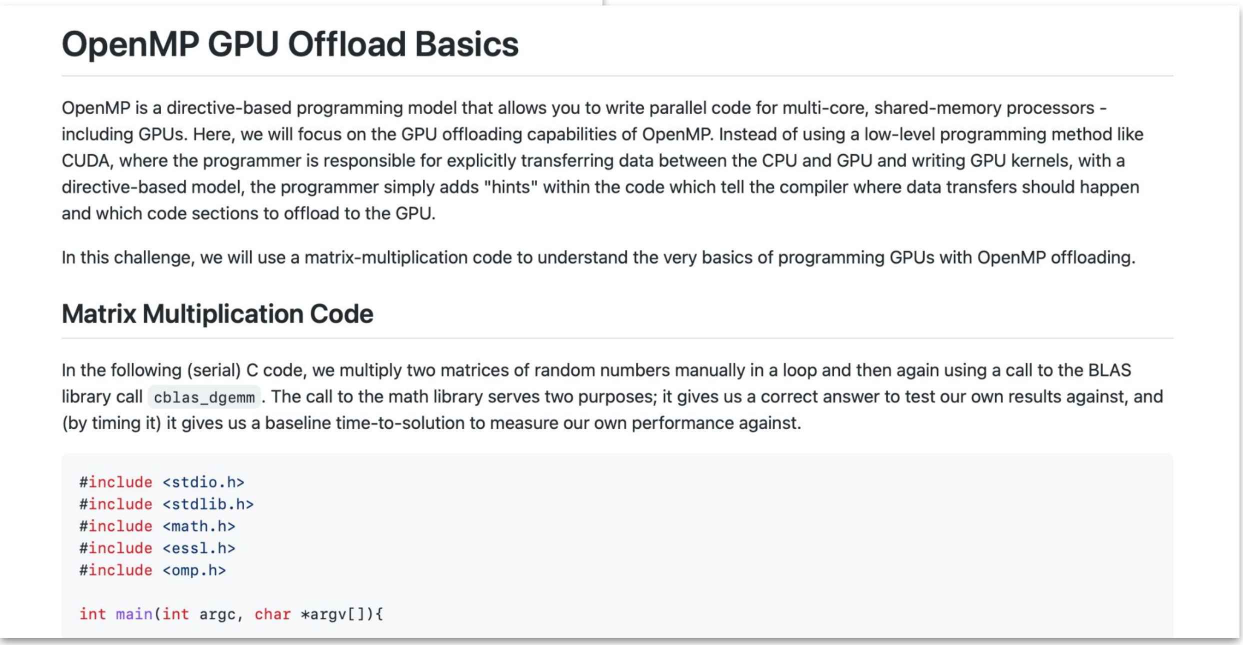
Communicator An object that represents a group of processes than can communicate with each other.

Rank Within a communicator each process is given a unique integer ID. Ranks start at 0 and are incremented contiguously. Ranks can be mapped to hardware processing elements like CPU cores.

Size The total number of ranks in a communicator.

MPI Region The part of the code that will be executed in parallel using MPI_Init and MPI_Finalize function calls.

The Communication Sandwich



OpenMP GPU Offload Basics

OpenMP is a directive-based programming model that allows you to write parallel code for multi-core, shared-memory processors - including GPUs. Here, we will focus on the GPU offloading capabilities of OpenMP. Instead of using a low-level programming method like CUDA, where the programmer is responsible for explicitly transferring data between the CPU and GPU and writing GPU kernels, with a directive-based model, the programmer simply adds "hints" within the code which tell the compiler where data transfers should happen and which code sections to offload to the GPU.

In this challenge, we will use a matrix-multiplication code to understand the very basics of programming GPUs with OpenMP offloading.

Matrix Multiplication Code

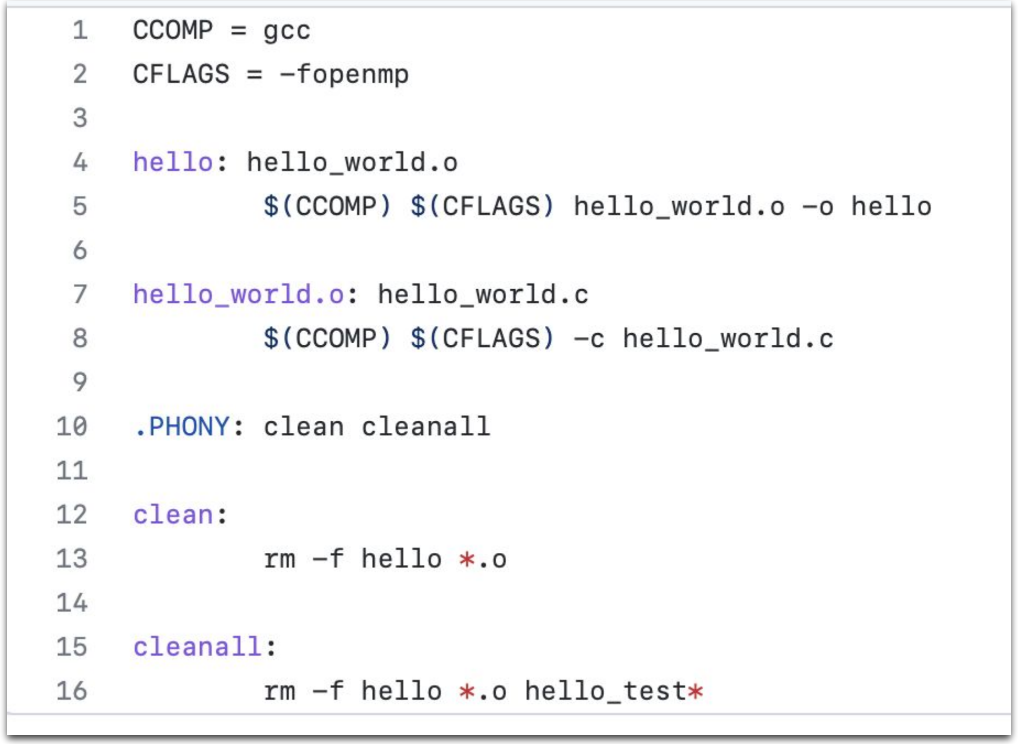
In the following (serial) C code, we multiply two matrices of random numbers manually in a loop and then again using a call to the BLAS library call `cblas_dgemm`. The call to the math library serves two purposes; it gives us a correct answer to test our own results against, and (by timing it) it gives us a baseline time-to-solution to measure our own performance against.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <essl.h>
#include <omp.h>

int main(int argc, char *argv[]){
```

How you will learn it

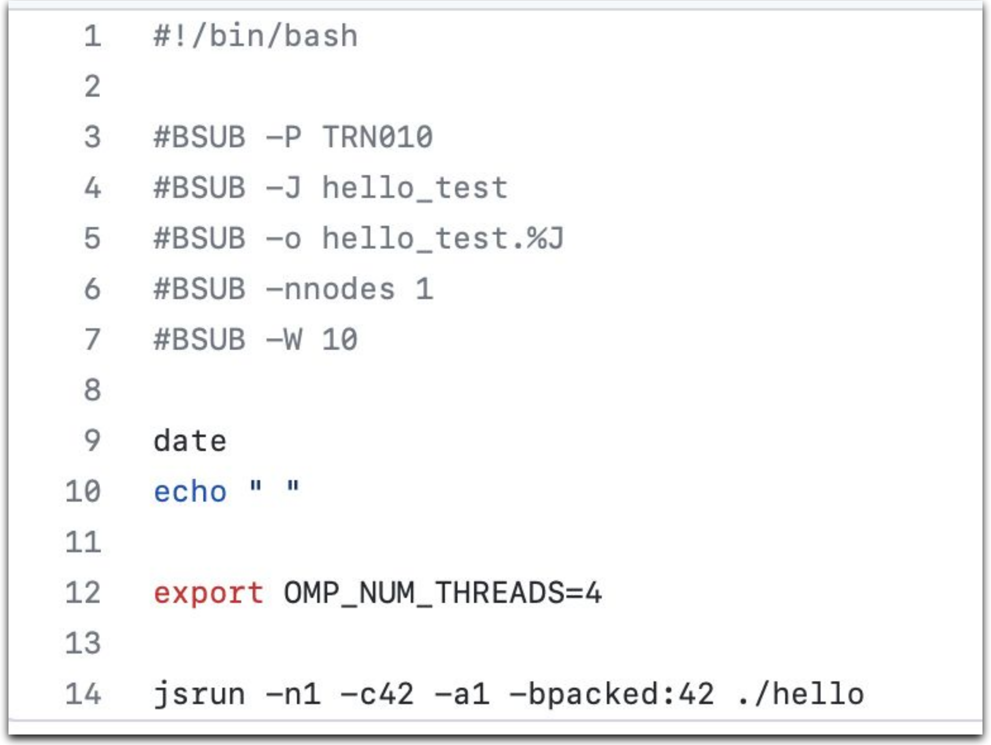
- We have handled for you
  - The makefiles that compile the codes
  - The submissions scripts that tell the computer how to run the jobs

Makefile



```
CCOMP = gcc
CFLAGS = -fopenmp

hello: hello_world.o
	$(CCOMP) $(CFLAGS) hello_world.o -o hello

hello_world.o: hello_world.c
	$(CCOMP) $(CFLAGS) -c hello_world.c

.PHONY: clean cleanall

clean:
	rm -f hello *.o

cleanall:
	rm -f hello *.o hello_test*
```

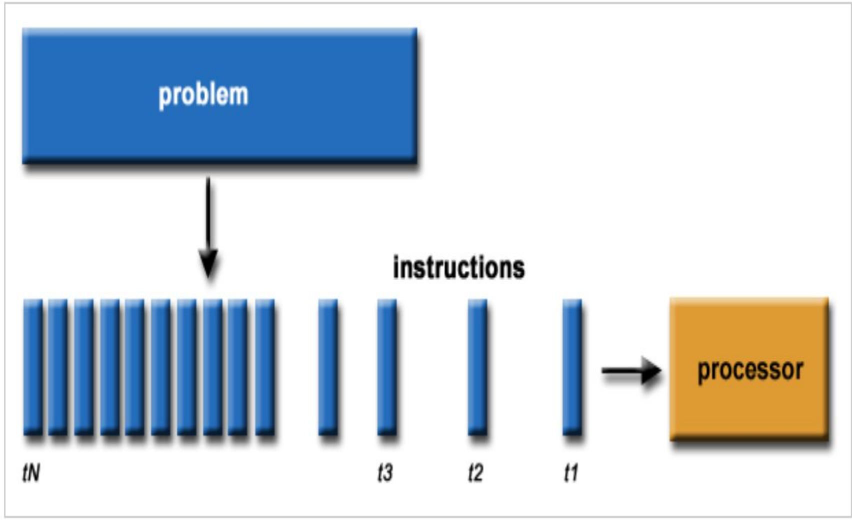
Submit.lsf

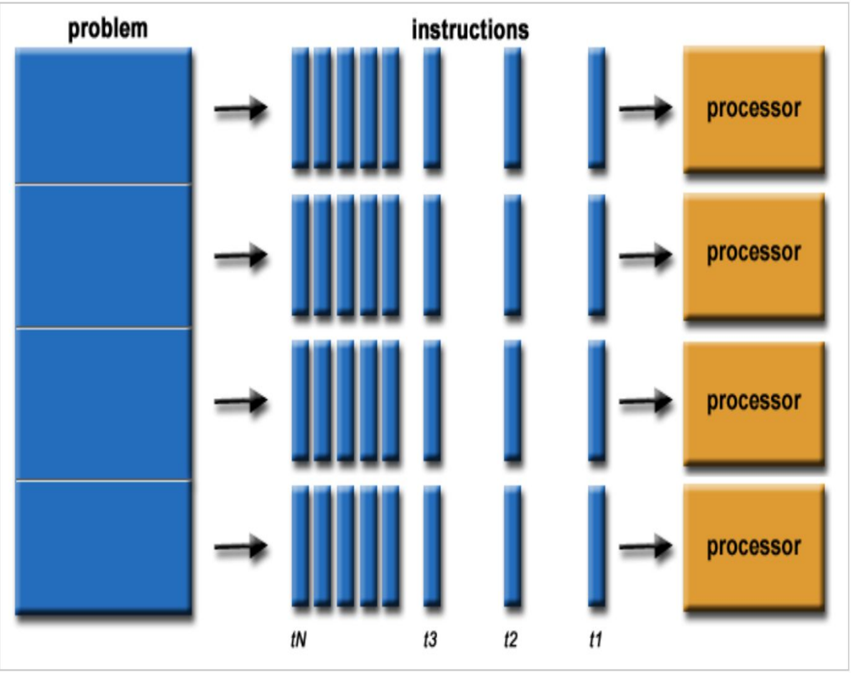


```
#!/bin/bash

#BSUB -P TRN010
#BSUB -J hello_test
#BSUB -o hello_test.%J
#BSUB -nnodes 1
#BSUB -W 10

date
echo " "

export OMP_NUM_THREADS=4

jsrun -n1 -c42 -a1 -bpacked:42 ./hello
```

What is High Performance Computing?

- High Performance Computing (HPC) is about solving the world’s largest engineering and science problems with supercomputers.
- That means doing work efficiently in parallel

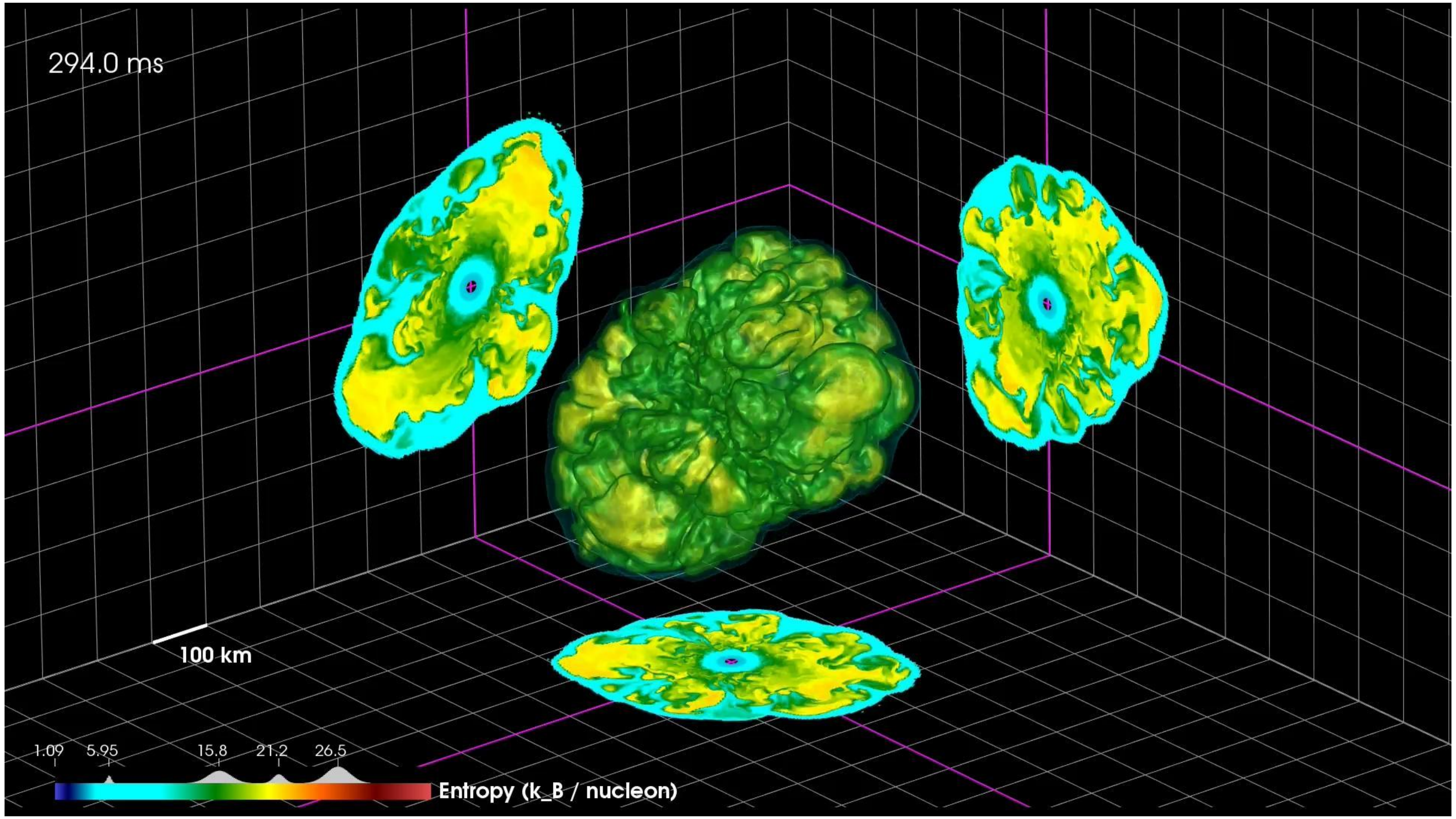




Images courtesy of : https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutoria

Why do HPC?

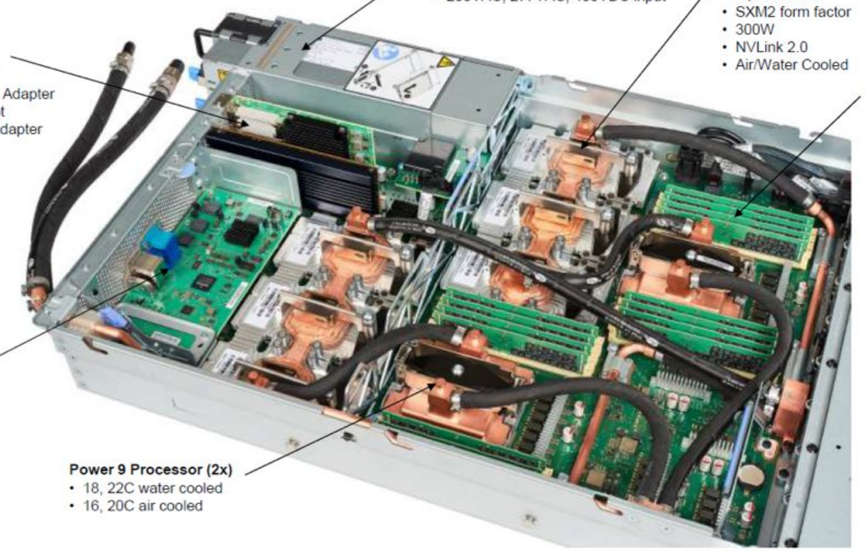
HPC allows simulations of nature to be built with enough detail that they can make accurate predictions of natural phenomena, so you can test things that are impossible to test physically or perhaps to expensive:



Movie Courtesy of Michael Sandoval

Node Overview

summit



Power Supplies (2x)
- 2200W
- 200VAC, 277VAC, 400VDC input

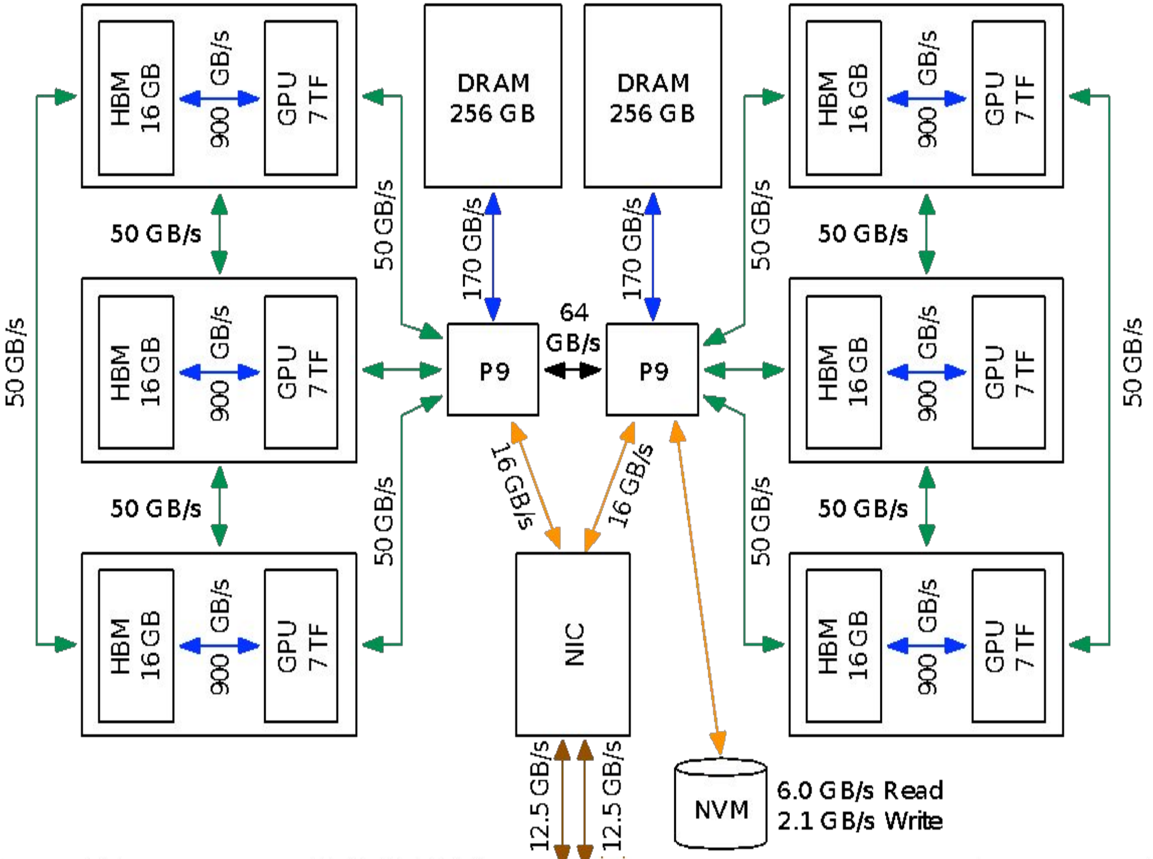
NVidia Volta GPU
- 3 per socket
- SXM2 form factor
- 300W
- NVLink 2.0
- Air/Water Cooled

PCIe slot (4x)
- Gen4 PCIe
- 2, x16 HHHL Adapter
- 1, Shared slot
- 1 x8 HHHL Adapter

Memory DIMM's (16x)
- 8 DDR4 IS DIMMs per socket
- 8, 16, 32,64, 128GB DIMMs

BMC Card
- IPMI
- 1 Gb Ethernet
- VGA
- 1 USB 3.0

Power 9 Processor (2x)
- 18, 22C water cooled
- 16, 20C air cooled



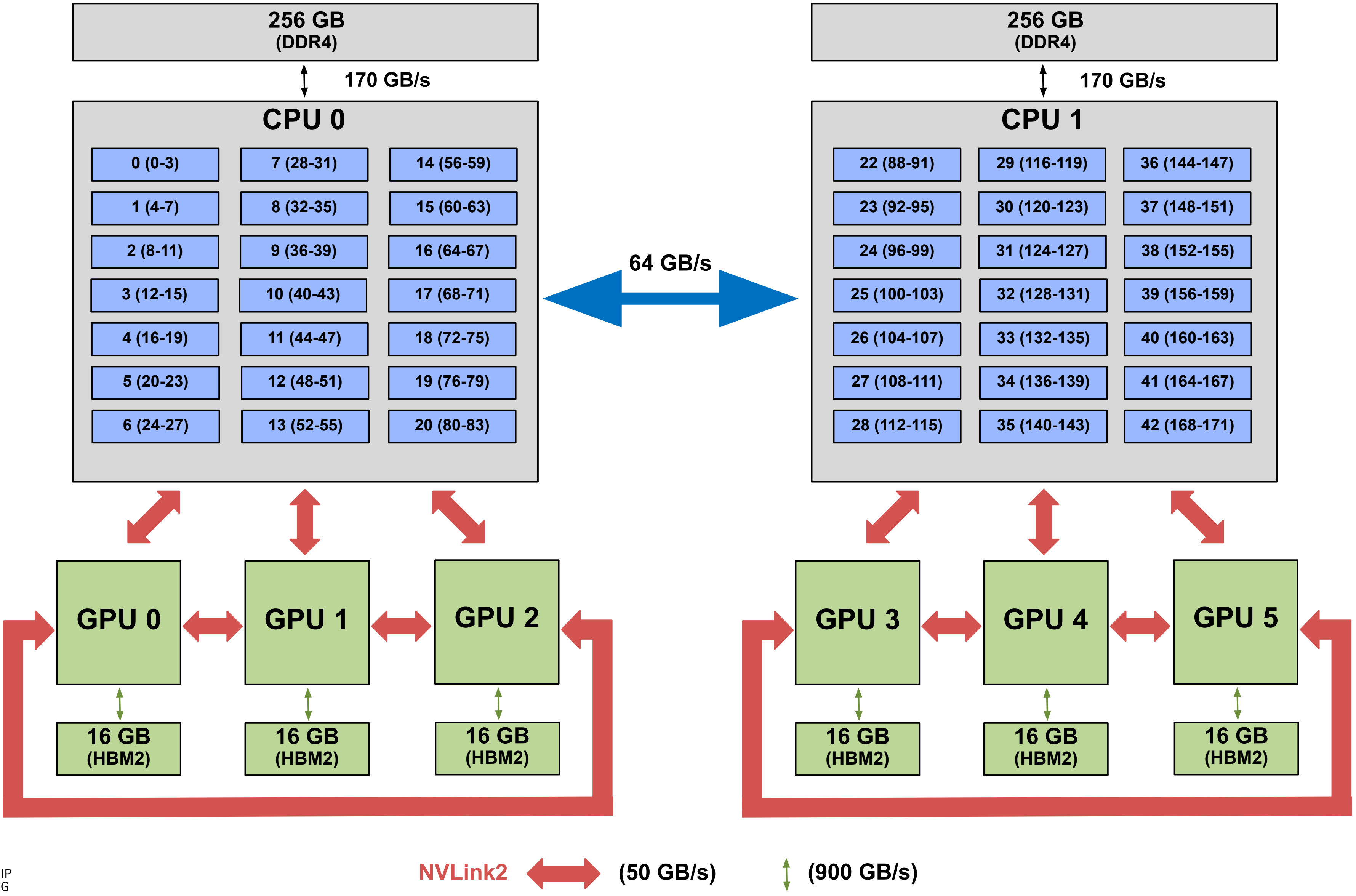
TF	42 TF (6x7 TF)
HBM	96 GB (6x16 GB)
DRAM	512 GB (2x16x16 GB)
NET	25 GB/s (2x12.5 GB/s)
MMsg/s	83

- ⟷ HBM/DRAM Bus (aggregate B/W)
- ⟷ NVLINK
- ⟷ X-Bus (SMP)
- ⟷ PCIe Gen4
- ⟷ EDR IB

HBM & DRAM speeds are aggregate (Read+Write).
All other speeds (X-Bus, NVLink, PCIe, IB) are bi-directional.

Summit Node

(2) IBM Power9 + (6) NVIDIA Volta V100



256 GB
(DDR4)

170 GB/s

CPU 0

0 (0-3)	7 (28-31)	14 (56-59)
1 (4-7)	8 (32-35)	15 (60-63)
2 (8-11)	9 (36-39)	16 (64-67)
3 (12-15)	10 (40-43)	17 (68-71)
4 (16-19)	11 (44-47)	18 (72-75)
5 (20-23)	12 (48-51)	19 (76-79)
6 (24-27)	13 (52-55)	20 (80-83)

256 GB
(DDR4)

170 GB/s

CPU 1

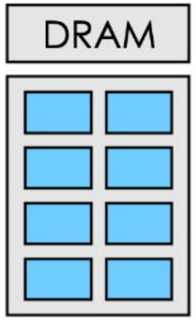
22 (88-91)	29 (116-119)	36 (144-147)
23 (92-95)	30 (120-123)	37 (148-151)
24 (96-99)	31 (124-127)	38 (152-155)
25 (100-103)	32 (128-131)	39 (156-159)
26 (104-107)	33 (132-135)	40 (160-163)
27 (108-111)	34 (136-139)	41 (164-167)
28 (112-115)	35 (140-143)	42 (168-171)

64 GB/s

Parallel Programming Models

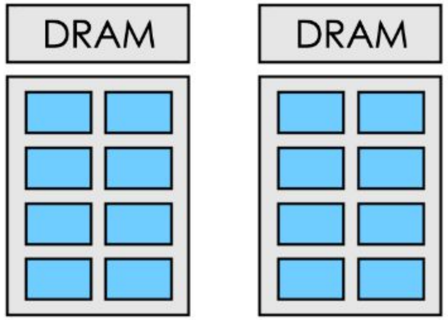
A =

```
for(int i=0; i<N; i++){

    B[i] = A[i]*A[i]

}
```



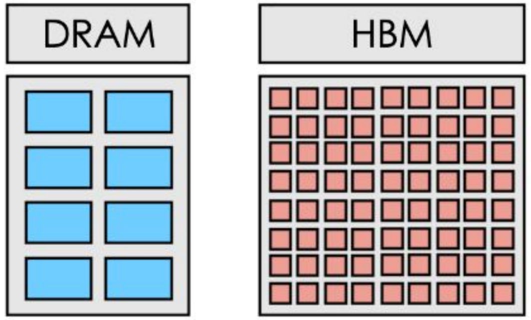
Shared-memory models

- E.g., OpenMP
  - All process threads can access same memory (single-node)



Distributed-memory models

- E.g., Message Passing Interface (MPI)
  - All processes (i.e., MPI ranks) have access to their own memory (multi-node)



GPU

- CUDA, HIP
- OpenACC, OpenMP offload (directive-based models)
- Kokkos (portability)

Ascent Account

If you want to keep working these modules after the event,

give me your name, Institution and email on the sheet and take a paper that tells you how to apply for time on Ascent, our training cluster.

If you complete these . . . . and one of these

1. Access_Summit_and_Clone_Repo
2. Basic_Workflow
3. OpenMP_Basics
4. MPI_Basics
5. Find_the_Compiler_Flag
6. GPU_Data_Transfers

1. Password_in_a_Haystack
2. Python_Conda_Basics
3. jsruAccess_Summit_and_Clone_Repo
4. LSF_Job_Launcher
5. Parallel_Scaling_Performance
6. Jobs_in_Time_Window
7. OpenMP_Offload
8. Python_Parallel_HDF5
9. Python_Cupy_Basics
10. Python_Pytorch_Basics
11. GPU_Matrix_Multiply
12. GPU_Profiling

we will give you a certificate that says you have been introduced to HPC.

Login

Remotely ssh access clusters

- Connect to a cluster through secure shell SSH

Jupyter or Mac terminal

MAC

- Applications> Utilities > Terminal

Jupyter

- Open browser
- Go to https://jupyter.olcf.ornl.gov/
- Login with cesp### user name and PIN followed by
- Choose Tapia training series

Both Mac and Jupyter Terminal

- Click Open Terminal

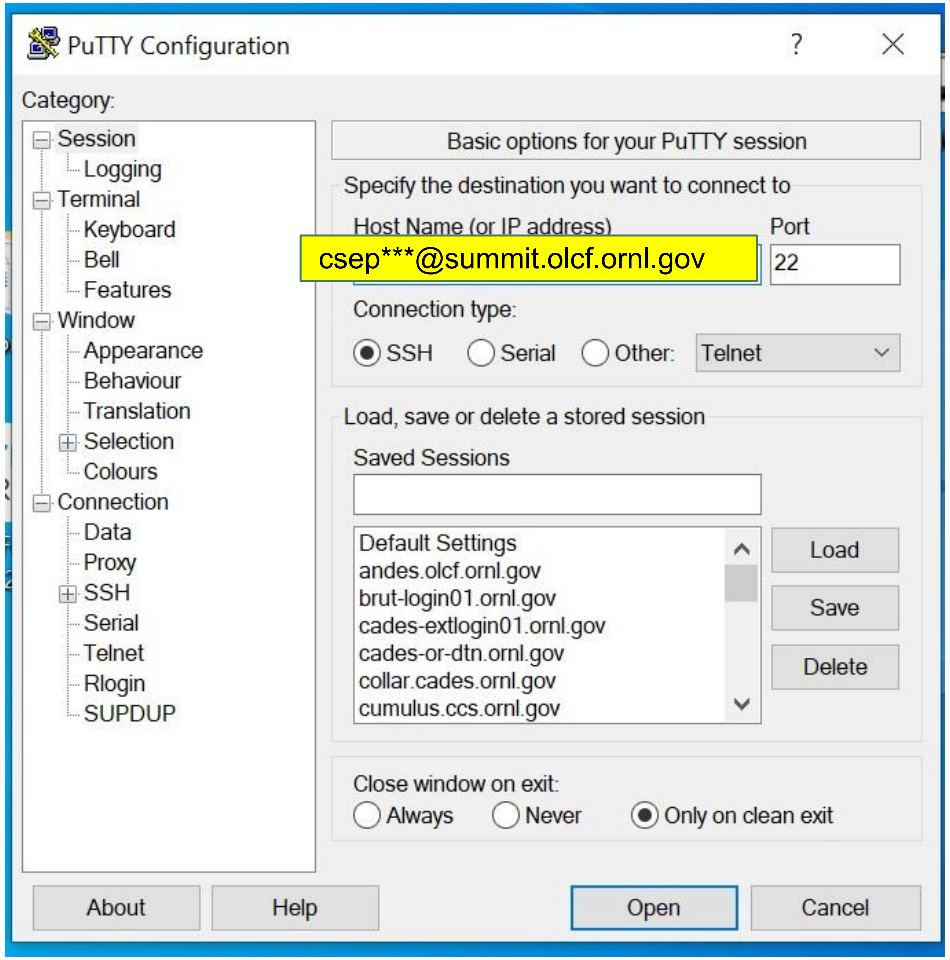
  Then type

```
ssh csep###@summit.olcf.ornl.gov
```

enter PIN followed by

Windows Putty

- Click Open Putty
- Enter csep###@summit.olcf.ornl.gov
- Click Open



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): csep***@summit.olcf.ornl.gov

Port: 22

Connection type: ● SSH ○ Serial ○ Other: Telnet

Load, save or delete a stored session

Saved Sessions

Default Settings
andes.olcf.ornl.gov
brut-login01.ornl.gov
cades-extlogin01.ornl.gov
cades-or-dtn.ornl.gov
collar.cades.ornl.gov
cumulus.ccs.ornl.gov

Close window on exit: ○ Always ○ Never ● Only on clean exit

Direct your browser here:

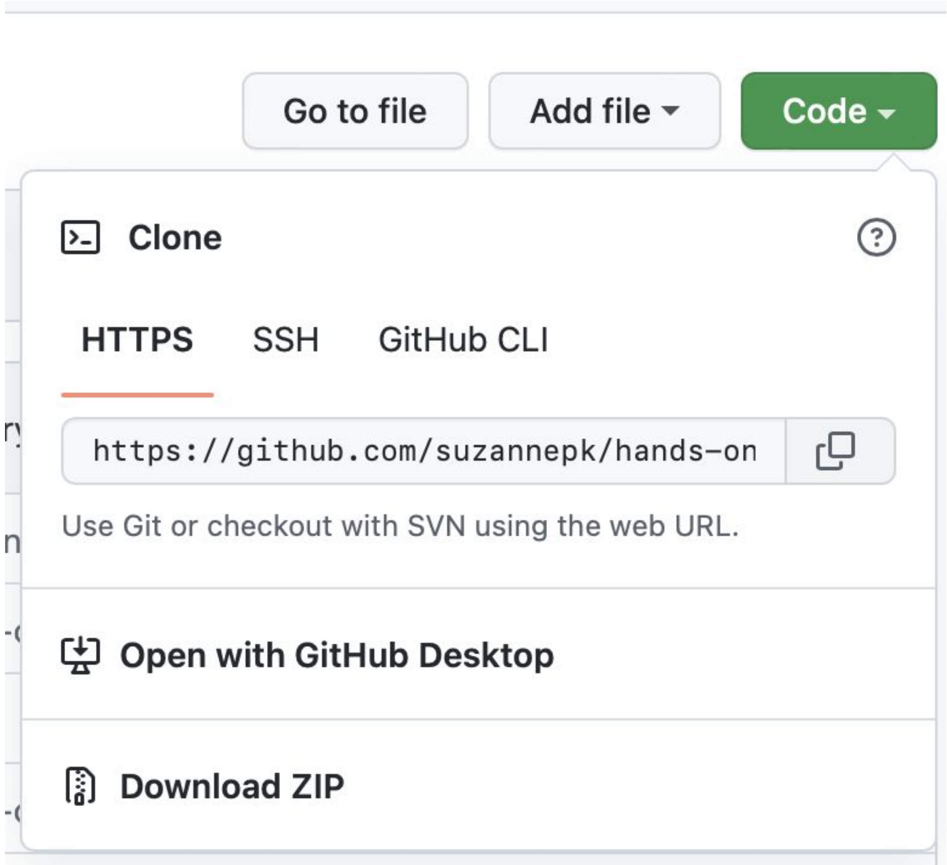
https://bit.ly/tapia_hpc

How to get the repo

Go to https://bit.ly/tapia_hpc

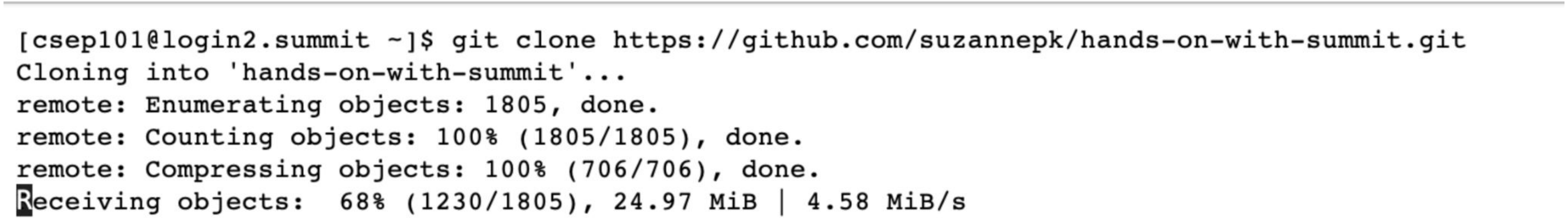
Click on the ⧉ to copy the address.

Go to your command line and type
`“git clone”`
then paste in the address.



https://github.com/suzannepk/hands-on

Use Git or checkout with SVN using the web URL.



```
[csep101@login2.summit ~]$ git clone https://github.com/suzannepk/hands-on-with-summit.git
Cloning into 'hands-on-with-summit'...
remote: Enumerating objects: 1805, done.
remote: Counting objects: 100% (1805/1805), done.
remote: Compressing objects: 100% (706/706), done.
Receiving objects:  68% (1230/1805), 24.97 MiB | 4.58 MiB/s
```

The exercise

Do these first

1. Access_Summit_and_Clone_Repo

   1.5. Basic_Unix_Vim

2. Basic_Workflow
3. OpenMP_Basics (will be great if you finish here)
4. MPI_Basics
5. Find_the_Compiler_Flag
6. GPU_Data_Transfers

After completing these challenges, feel free to "choose your own adventure" by completing the rest of the challenges in any order you prefer.

- Password_in_a_Haystack
- Python_Conda_Basics
- LSF_Job_Launcher
- Parallel_Scaling_Performance
- Jobs_in_Time_Window
- OpenMP_Offload
- Python_Parallel_HDF5
- Python_Cupy_Basics
- Python_Pytorch_Basics
- GPU_Matrix_Multiply
- GPU_Profiling

Questions

Join the slack channel for this course:

https://bit.ly/hpc-slack

Instructions for Applying for Ascent

https://bit.ly/ascent-access

Extra slides

The OLCF has Successfully Delivered Six Systems Since 2004

- Frontier will be system number seven and will provide an increased capability that is 5-10 X more powerful than Summit

- Partnering with vendors and users has been essential to delivering science in a rapidly changing computational environment

Phoenix X1

- X1e

2004

Jaguar XT3

- Dual core upgrade

2005

Jaguar XT4

- Quad core upgrade

2007

Jaguar XT5

- 6 core upgrade
- 16 core upgrade

2008

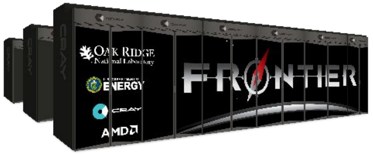
Titan XK7

- GPU upgrade

2012

Summit

2018

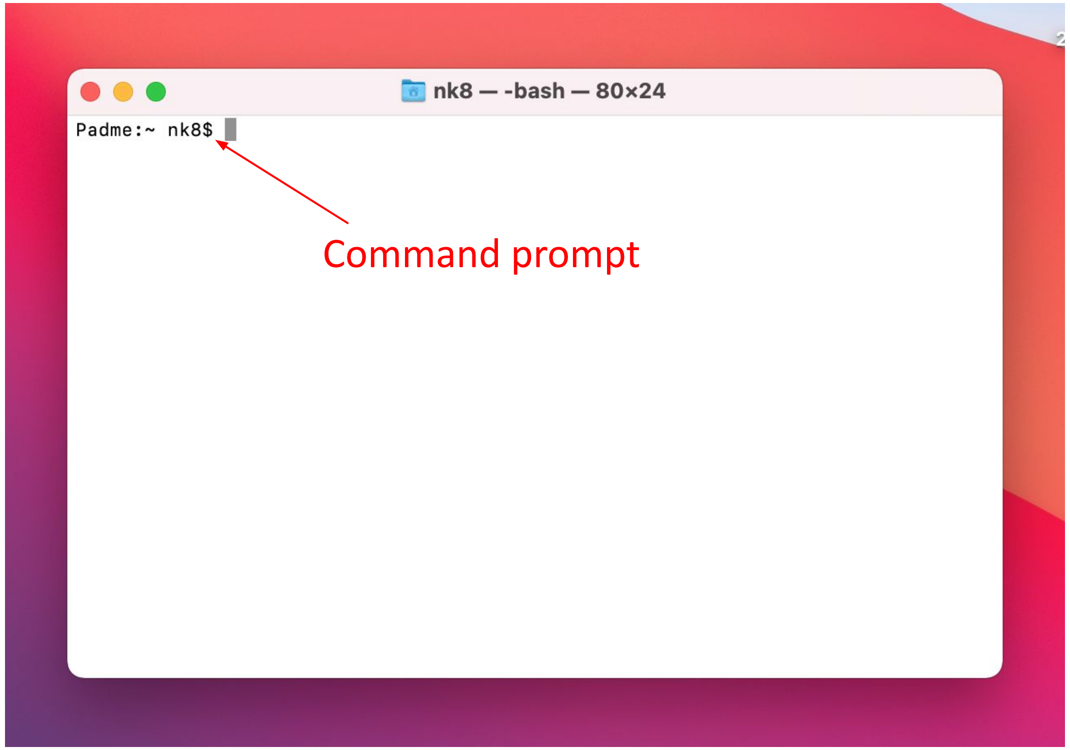


Frontier

2021

Terminal on MAC or our Jupyter GUI

- Terminal allows you to control your Mac from a command line.
- Usually under Applications> Utilities > Terminal
- Example shows bash shell, which has “$” for a command prompt.



```
Padme:~ nk8$ 
```

Command prompt

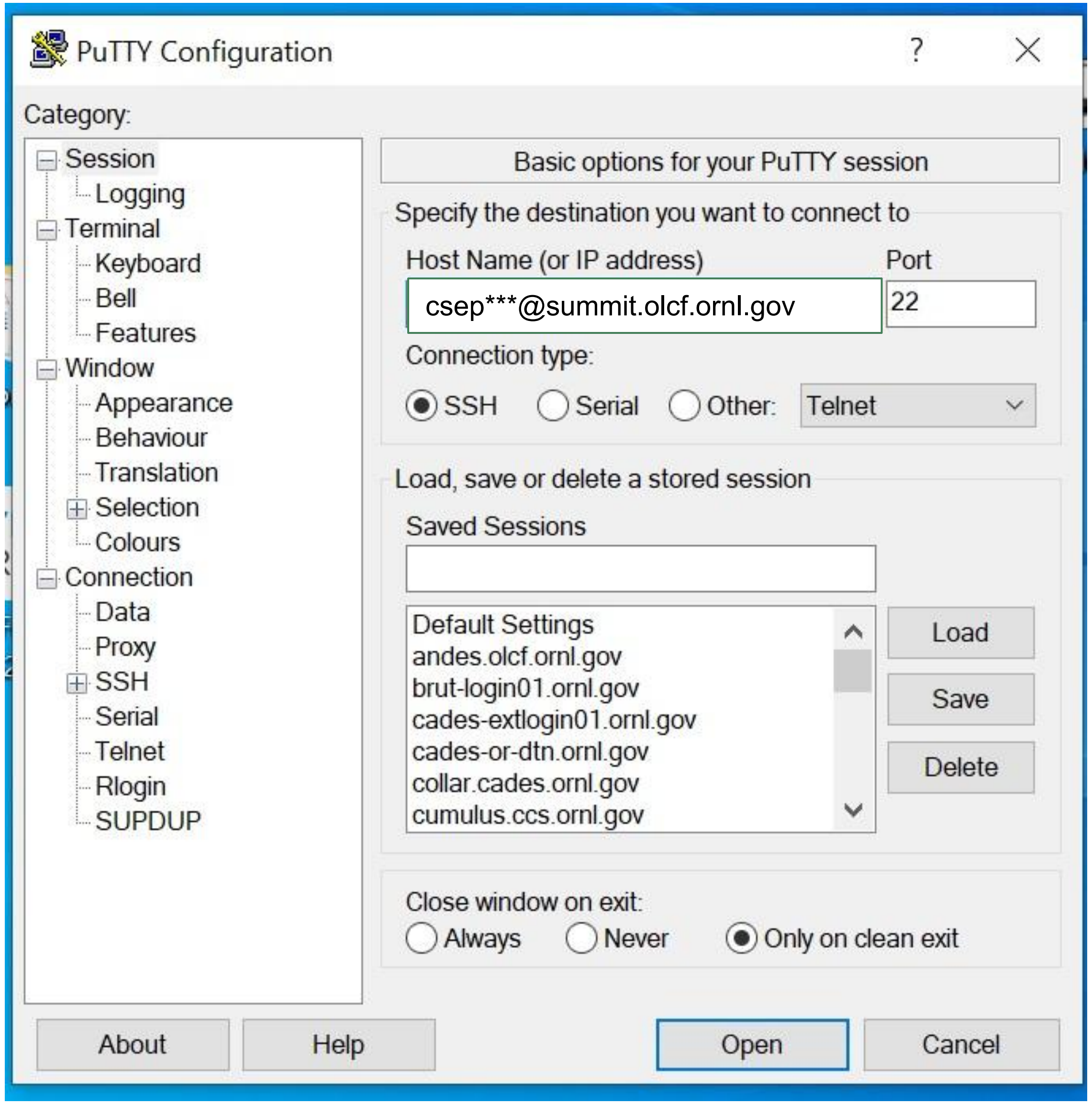
To Login to a remote computer:

- Open the terminal application
- Type command: `ssh csep***@summit.olcf.ornl.gov`
- Type PIN followed by the passcode on your RSA token

Putty

- Install and run PuTTY
- Type remote hostname or IP
- Click Open

For PuTTY help, see https://www.ssh.com/ssh/putty/windows



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): csep***@summit.olcf.ornl.gov

Port: 22

Connection type: ● SSH ○ Serial ○ Other: Telnet

Load, save or delete a stored session

Saved Sessions

Default Settings
andes.olcf.ornl.gov
brut-login01.ornl.gov
cades-extlogin01.ornl.gov
cades-or-dtn.ornl.gov
collar.cades.ornl.gov
cumulus.ccs.ornl.gov

Close window on exit: ○ Always ○ Never ● Only on clean exit

OpenMP overview

- OpenMP is programming model that allows you to write parallel code for multi-core, shared-memory processors. Your laptop/desktop likely has a multi-core processor (e.g., 4-core). This is also true of each individual compute node on Summit.
- Uses directives, (instructions to the compiler) to designate a parallel region

Serial

```
// Perform element-wise addition of vectors
    for(int i=0; i<N; i++){
        C[i] = A[i] + B[i];
    }
```

- One thread/process will execute each iterations sequentially
- Total time = Time_for_1_itteration * N

Parallel

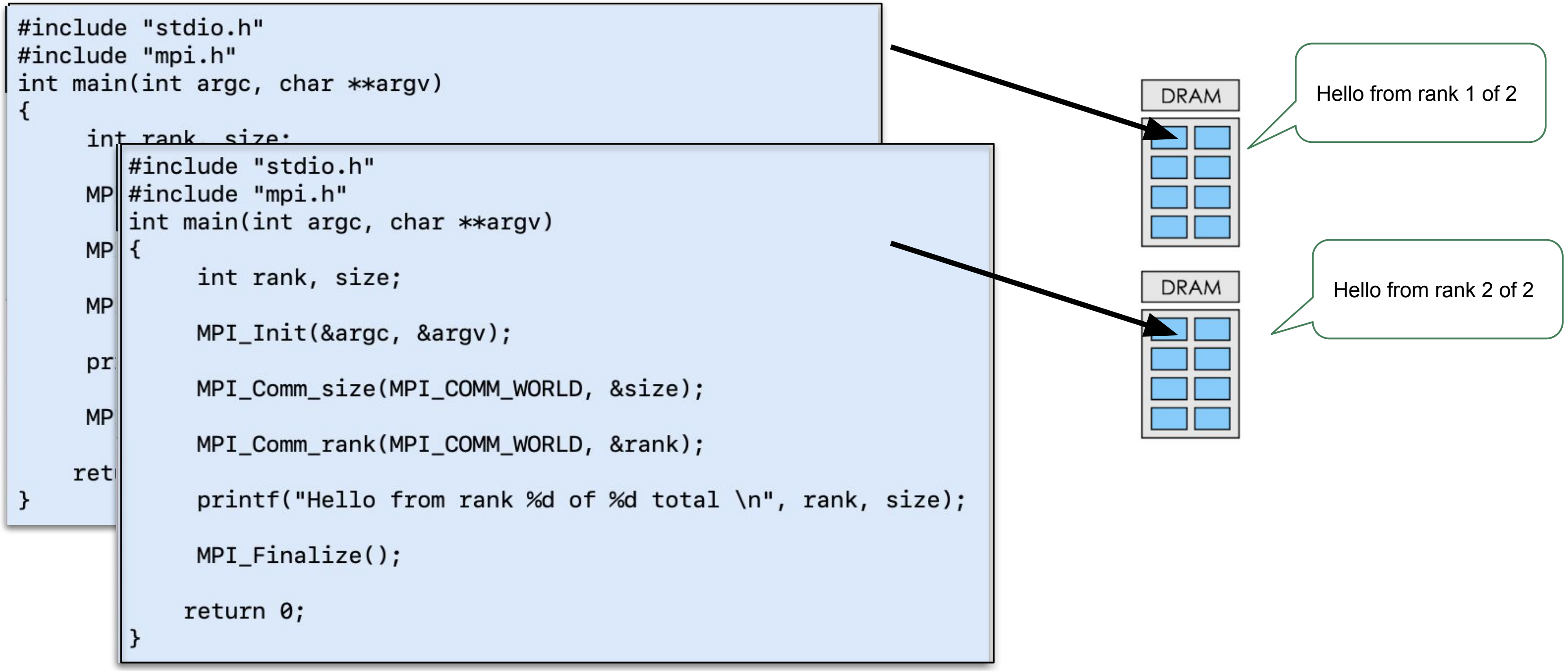
```
// Perform element-wise addition of vectors
        #pragma omp parallel default(none) shared(A, B, C)
        {
        #pragma omp for
        for(int i=0; i<N; i++){
        C[i] = A[i] + B[i];
        }
        }
```

- Say we tell it to use 4 threads (OMP_NUM_THREADS=4)
- Iterations will be distributed over 4 threads.

Total time = Time_for_1_itteration * N/4 (Or about 4 times faster)

MPI overview

- An MPI parallel computation consists of a number of processes, each working on some local data. Each process has purely local variables, and there is no mechanism for any process to directly access the memory of another.
  - Sharing of data between processes takes place by message passing, that is, by explicitly sending and receiving data between processes.
  - This encapsulation of data and instructions, means that the processes can run on nodes that have different pools of memory.



```
#include "stdio.h"
#include "mpi.h"
int main(int argc, char **argv)
{
      int rank, size;
      MP
      MP
      MP
      pr
      MP
    ret
}
```

```
#include "stdio.h"
#include "mpi.h"
int main(int argc, char **argv)
{
      int rank, size;

      MPI_Init(&argc, &argv);

      MPI_Comm_size(MPI_COMM_WORLD, &size);

      MPI_Comm_rank(MPI_COMM_WORLD, &rank);

      printf("Hello from rank %d of %d total \n", rank, size);

      MPI_Finalize();

     return 0;
}
```

MPI overview

Message passing programs consist of multiple instances of a serial program that communicate by library calls.

These calls may be roughly divided into four classes:

- Calls used to initialize,manage,and finally terminate communications.
- Calls used to communicate between pairs of processors.
- Calls that perform communications operations among groups of processors.
- Calls used to create arbitrary data types.

During your MPI challenge you will explore the first three kinds of library calls by doing exercises with an MPI hello-world and simple codes that use two kinds of MPI communication patterns. Since there are many of us we will using MPI to to allow work to be shared on one node.