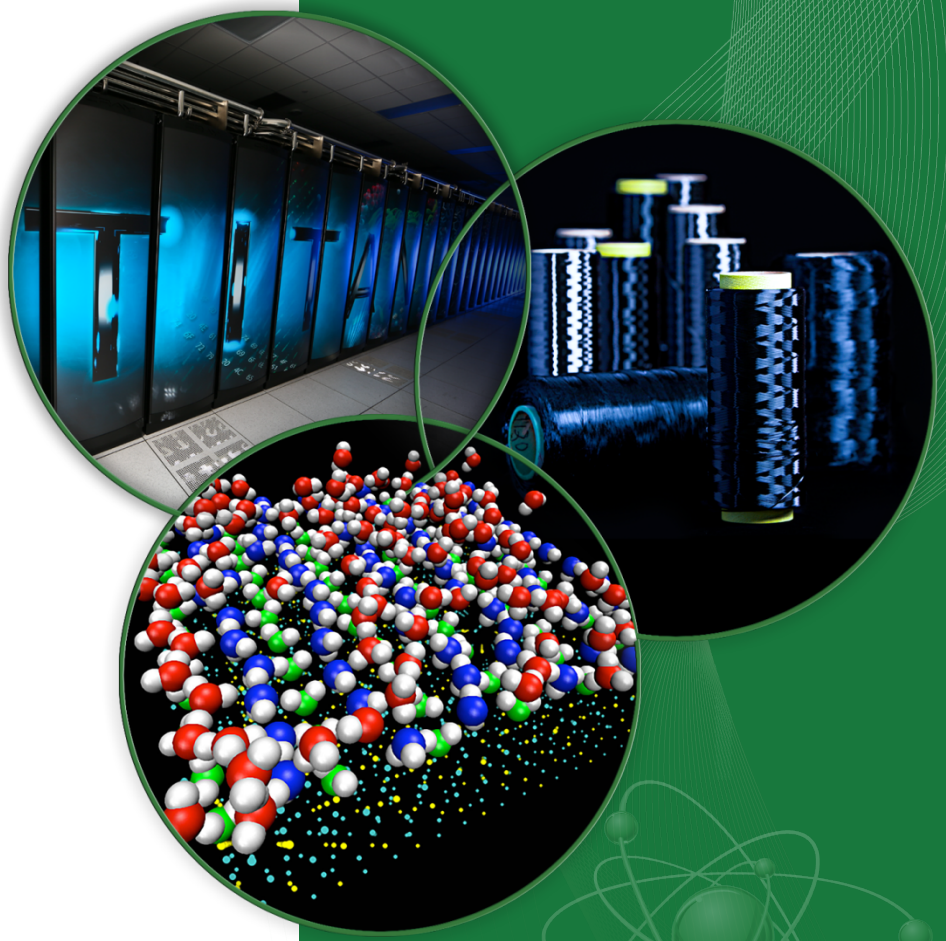
GPU accelerated computing



Verónica G. Vergara Larrea

Adam B. Simpson

Tapia 2017

September 23, 2017

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

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Outline

- What is the OLCF?
  - OLCF Compute and Data Resources
- Titan
- Computing at the OLCF
- GPU basics
- Hands-on: Vector Addition
- Exercise: SPH
- Profiling & Optimizing
- Visualizing results

What is the OLCF?

- **O**ak Ridge **L**eadership **C**omputing **F**acility
- Established in 2005 at ORNL
- DOE-SC user facility
  - Open to nearly everyone in the world
  - Free to use for non-proprietary work
  - Allocations are merit based
- **L**eadership **C**omputing **F**acility
  - Develop and use the most advanced computing systems in the world

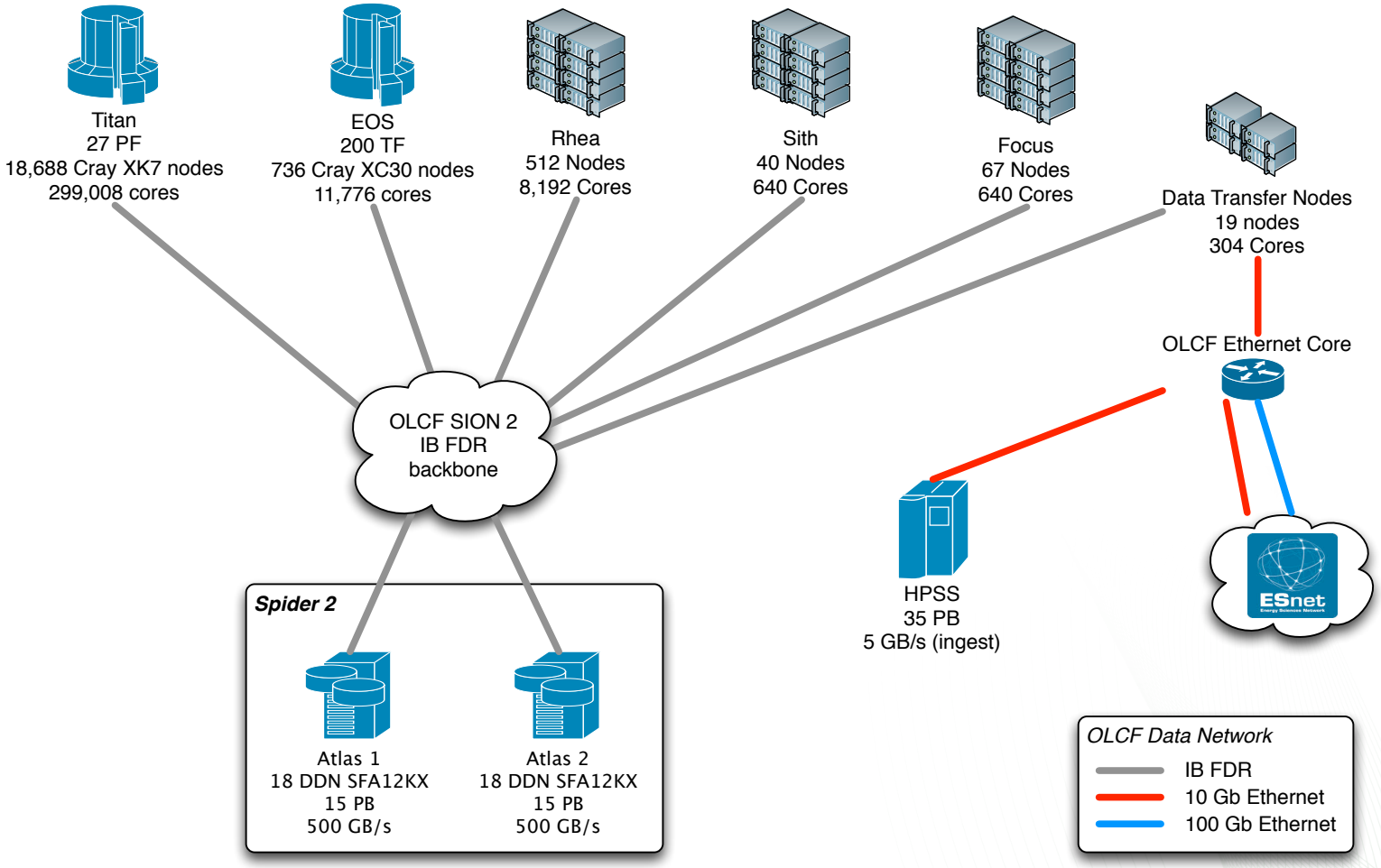
OLCF Compute Resources

- Cray XK7
- #4 in the TOP500 (June 2017)
- 27 PF
- 18,688 nodes/299,008 cores
  - AMD Opteron + K20x GPU per node

- Cray XC30
- 736 nodes/11,766 cores
  - Two 8-core Intel Xeon E5-2670
- Used for pre- and post-processing

- RHEL6 Linux cluster
- 512 nodes/8,192 cores
  - Two 8-core Intel Xeon E5-2650
- 9 nodes have two K80 GPUs
- Used for pre- and post-processing

The OLCF Compute & Data ecosystem



From "OLCF I/O best practices" (S. Oral)

The Titan supercomputer

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Size	18,688 Nodes	5,000 Sq. feet	
Peak Performance	27.1 PetaFLOPS	2.6 PF CPU	24.5 PF GPU
Power	8.2 MegaWatts	~7,000 homes	
Memory	710 TeraBytes	598 TB CPU	112 TB GPU
Scratch File system	32 PetaBytes	1 TB/s Bandwidth	

Image courtesy of ORNL

Titan Node Layout

- AMD Opteron CPU
  - 16 bulldozer “cores”
  - 8 FP units
  - 32GB DDR3
- NVIDIA K20x GPU
  - 14 SMX “cores”
  - 896 64-bit FP units
  - 6GB GDDR5
- Cray Gemini interconnect
  - 20 GB/s bandwidth

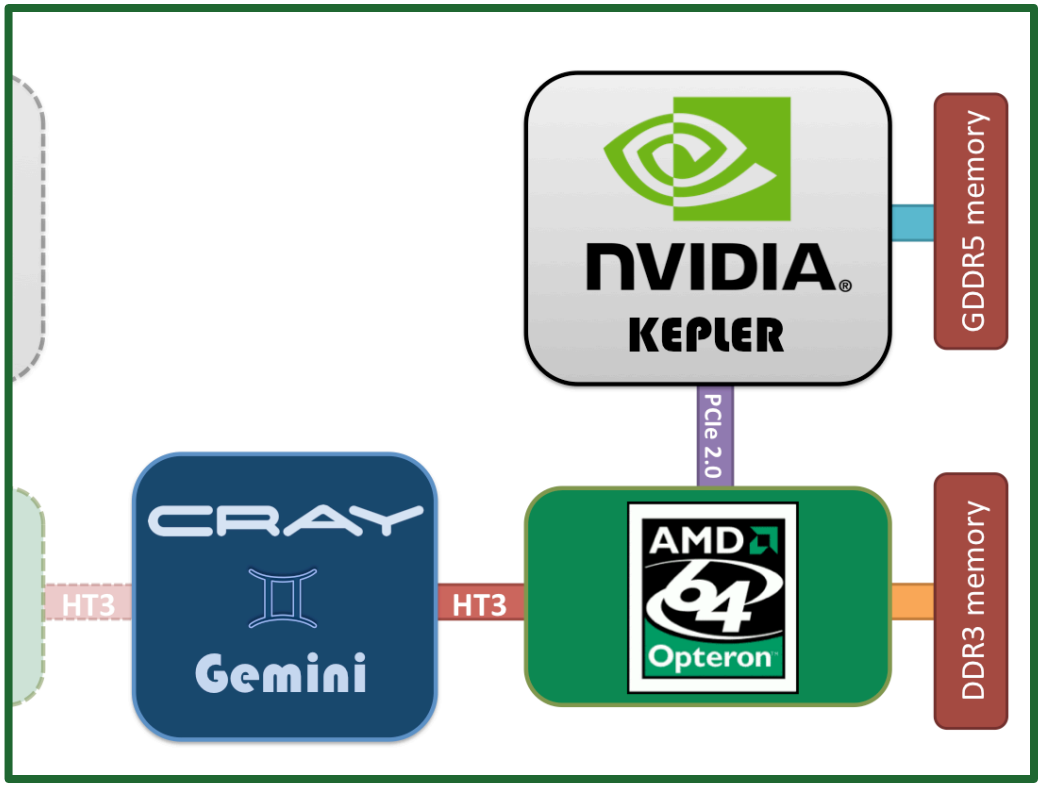
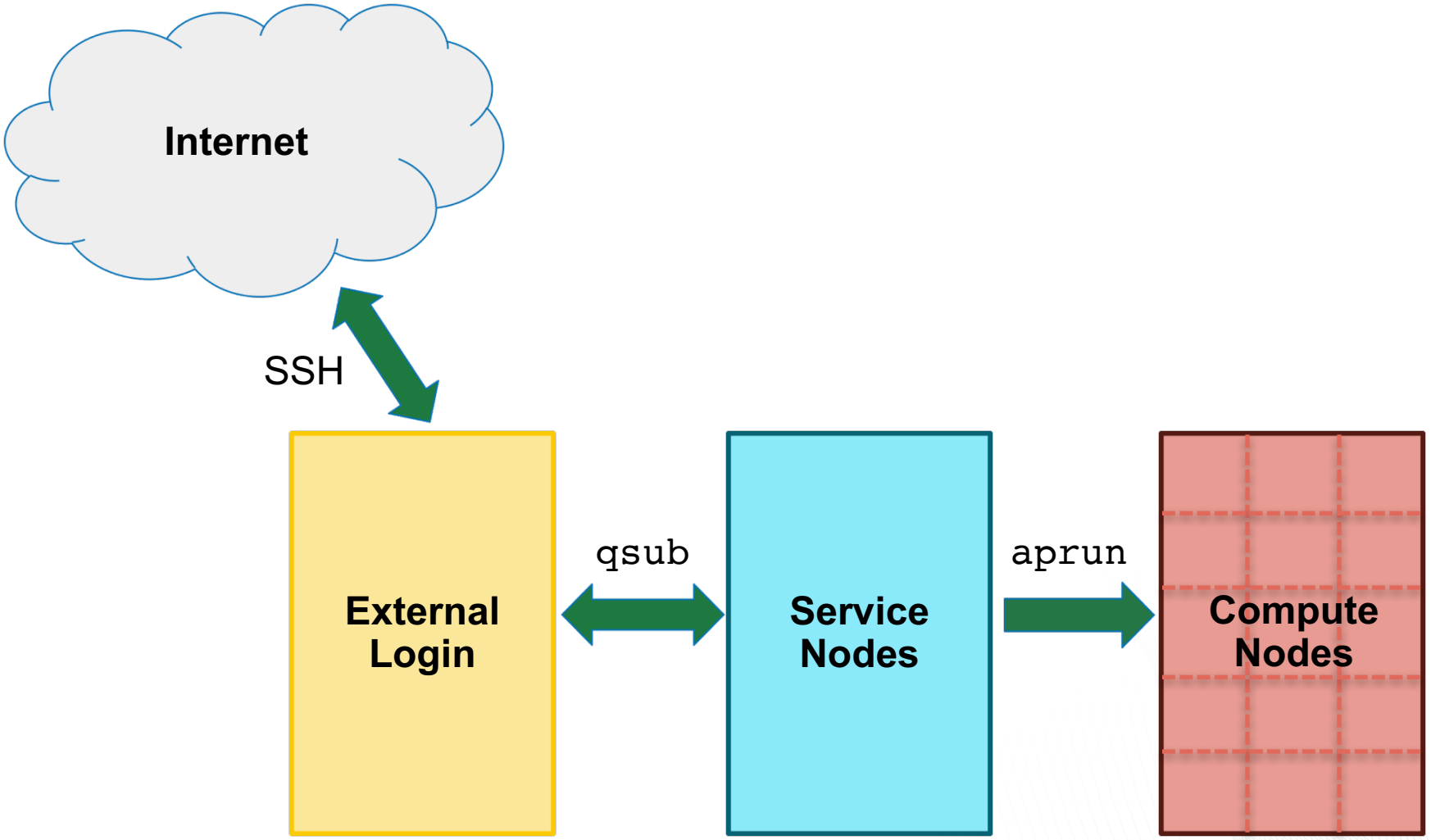


Image courtesy of ORNL

Titan node structure - connectivity

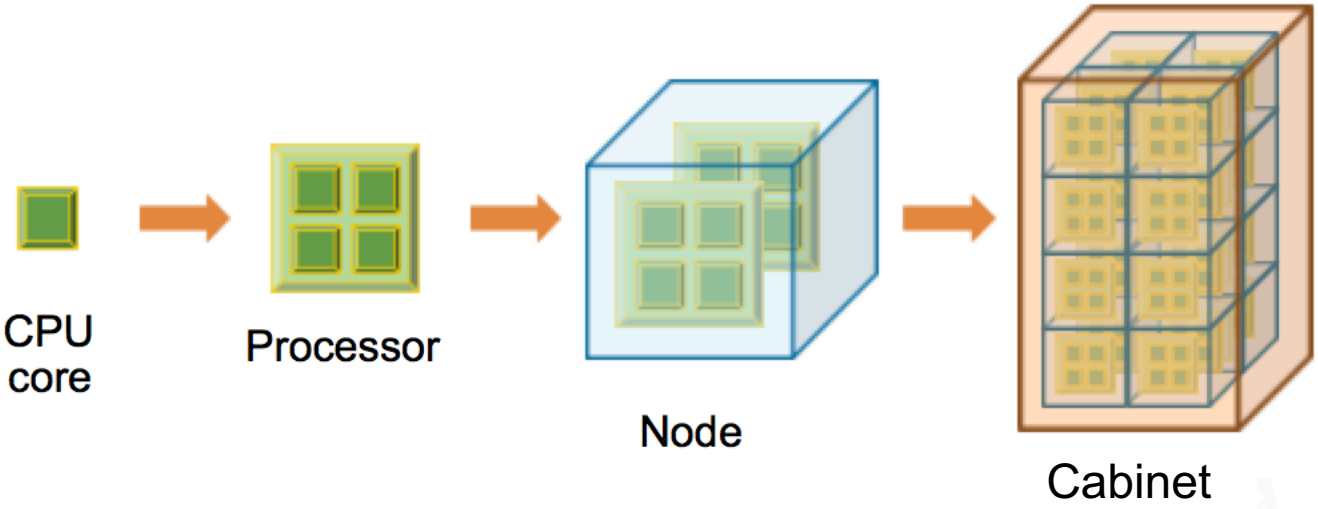


Supercomputing Basics

Supercomputer refresher

- Harnesses the power of multiple systems
- Connected by a high-speed network
- Used to solve larger and/or more complex problems in attainable time scales
- Applications need to be parallelized to take full advantage of the system
- Parallel programming models
  - Distributed: MPI
  - Shared: OpenMP, pthreads, OpenACC

Supercomputer components



Linux clusters vs. Cray supercomputers

Cluster

- Ethernet or InfiniBand
- Login nodes == compute nodes (usually)
- Computes can be accessed via `ssh` and `qsub -I`
- Uses `mpirun`

Cray

- Proprietary interconnect
- Login/service nodes != compute nodes
- Compute nodes can only be accessed via `aprun`
- Uses `aprun`

Hands-on: connecting to OLCF

- For this hands-on session we will use Titan
- To login from your laptop:
  - `ssh <username>@home.ccs.ornl.gov`
    - e.g. `ssh csep145@home.ccs.ornl.gov`
  - Follow the hand-out with instructions to set your PIN
- From `home.ccs.ornl.gov`:
  - `ssh titan.ccs.ornl.gov`

Understanding the environment

Environment Modules

- Dynamically modifies your user environment using modulefiles
- Sets and/or modifies environment variables
- Controlled via the `module` command:
  - `module list` : lists loaded modulefiles
  - `module avail` : lists all available modulefiles on a system
  - `module show` : shows changes that will be applied when the modulefile is loaded
  - `module [un]load` : (un)loads a modulefile
  - `module help` : shows information about the modulefile
  - `module swap` : can be used to swap a loaded modulefile by another

Programming environment

- Cray provides default programming environments for each compiler:
  - PrgEnv-pgi (default on Titan and Metis)
  - PrgEnv-intel
  - PrgEnv-gnu
  - PrgEnv-cray
- Compiler wrappers necessary for cross-compiling:
  - `cc` : pgcc, icc, gcc, craycc
  - `CC` : pgc++, icpc, g++, crayCC
  - `ftn` : pgf90, ifort, gfortran, crayftn

File systems available

- Home directories:
  - User home: `/ccs/home/csep###`
  - Project home: `/ccs/proj/trn001`
- Scratch directories:
  - Live on the Lustre parallel file system (Spider 2)
  - User scratch: `$MEMBERWORK/trn001`
  - Project scratch: `$PROJWORK/trn001`
  - World scratch: `$WORLDWORK/trn001`
- Only directories on Spider 2 are accessible from the compute nodes

Data Transfer

- Between OLCF file systems:
  - standard Linux tools: `cp`, `mv`, `rsync`
- From/to local system to/from OLCF:
  - `scp`
  - `bbcp`: https://www.olcf.ornl.gov/kb_articles/transferring-data-with-bbcp/
  - Globus:
    - https://www.globus.org
    - https://www.olcf.ornl.gov/computing-resources/data-management/data-management-user-guide/#10897

Visualization Tools

- Several applications available:
  - Paraview: http://www.paraview.org/
  - VisIt: https://wci.llnl.gov/simulation/computer-codes/visit/
  - and more.
- At the GPU Accelerated Computing session, we will use Paraview:
  - Client available for download at: http://www.paraview.org/download/
  - Transfer results to your local machine
- Much more powerful, can also be used in parallel to create simulations remotely

The PBS system

- **P**ortable **B**atch **S**ystem
- Allocates computational tasks among available computing resources
- Designed to manage distribution of batch jobs and interactive sessions
- Different implementations available:
  - OpenPBS, PBSPro, TORQUE
- Tightly coupled with workload managers (schedulers)
  - PBS scheduler, Moab, PBSPro
- At the OLCF: TORQUE and Moab are used

PBS commands and options

- Useful TORQUE commands:
  - `qsub` : submits a job
  - `qstat` : shows current status of a job from TORQUE's perspective
  - `qdel` : deletes a job from the queue
  - `qalter` : allows users to change job options
- Useful Moab commands:
  - `showq` : shows all jobs in the queue
  - `checkjob` : shows you the status of a job
- Common PBS options:
  https://www.olcf.ornl.gov/kb_articles/common-batch-options-to-pbs/
  - **`-l nodes=`**<# of nodes>**`,walltime=`**<HH:MM:SS>
  - **`-A`** <ProjectID>
  - **`-q`** <queue>
  - **`-I`** : interactive job

The `aprun` command

- Used to run a compiled application across one or more compute nodes
- Only way to reach compute nodes on a Cray
- Allows you to:
  - specify application resource requirements
  - request application placement
  - initiate application launch
- Similar to `mpirun` on Linux clusters
- Several options:
  - **`-n`** : total # of MPI tasks (processing elements)
  - **`-N`** : # of MPI tasks per physical compute node
  - **`-d`** : # of threads per MPI task
  - **`-S`** : # of MPI tasks per NUMA node

PBS submission file example

```
#!/bin/bash -l
#PBS -A TRN001
#PBS -l nodes=1
#PBS -l walltime=00:15:00
#PBS -N test


cd $MEMBERWORK/trn001

module list


aprun -n 2 /bin/hostname
```

Hands-on: Submitting a job

- On Chester, check the queue:

```
csep145@titan-ext4:~> qstat
```

- Then submit an interactive job:
  - This session will be used for the duration of the workshop

```
qsub -l nodes=1,walltime=02:00:00 -A TRN001 –I
```

- If you check the queue again you should see your job:

```
titan-batch1:~ vgv$ qstatme

titan-moab.ccs.ornl.gov:
                                                                  Req'd       Req'd       Elap
Job ID                  Username    Queue    Jobname         SessID  NDS   TSK   Memory      Time    S   Time
----------------------- ----------- -------- --------------- ------ ----- ------ --------- --------- - ---------
3638007                 vgv         batch    STDIN            18162     1     16       --   02:00:00 R  00:00:08
```

Hands-on: putting it all together

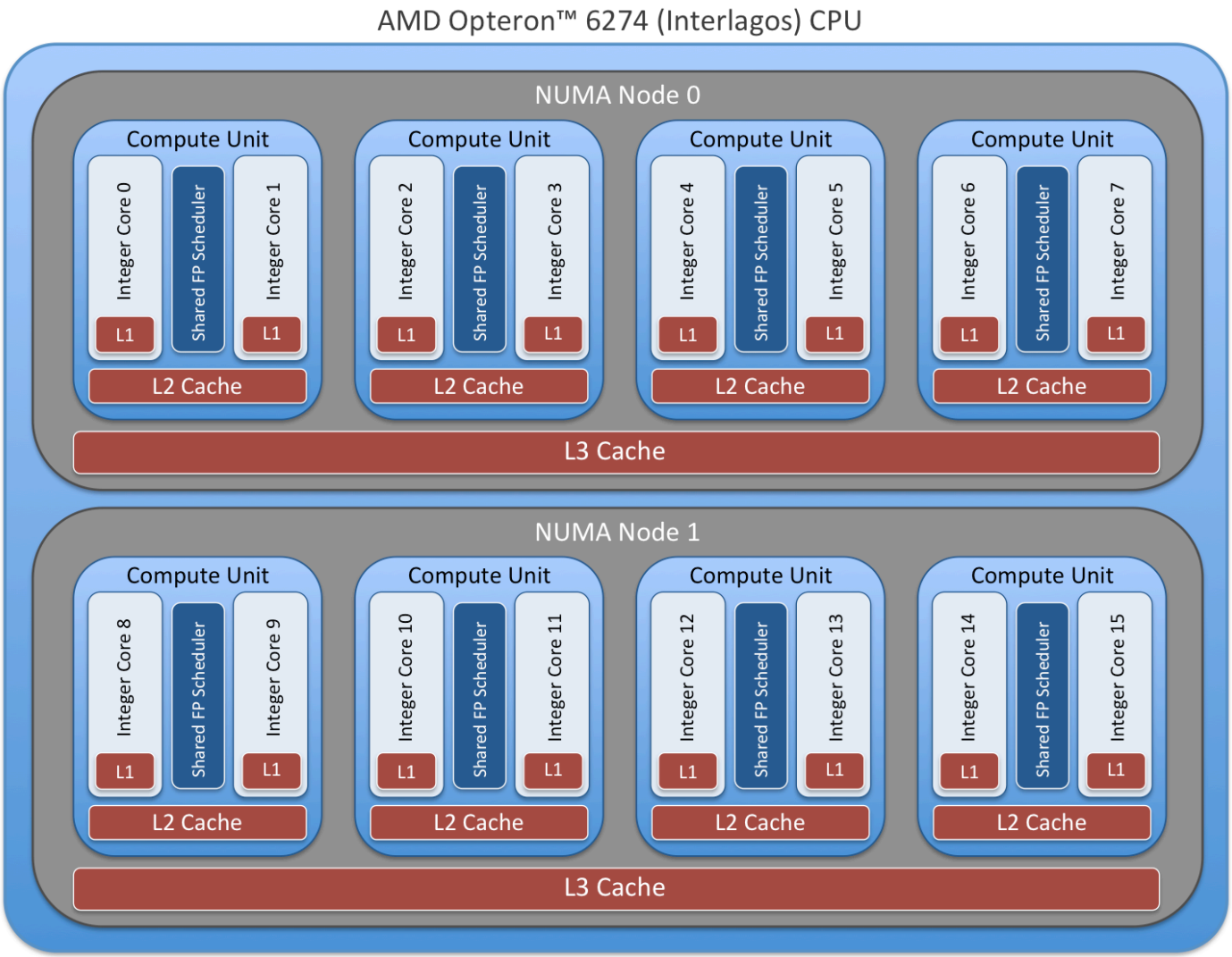
```
$ module avail
$ module list
$ cd $MEMBERWORK/trn001
$ cp $PROJWORK/trn001/tapia2017/hello-mpi.c .
$ cc hello-mpi.c -o hello.titan
$ aprun -n 2 ./hello.titan
Rank:    0  NID:          15  Total:     2
Rank:    1  NID:          15  Total:     2
```

Things to remember

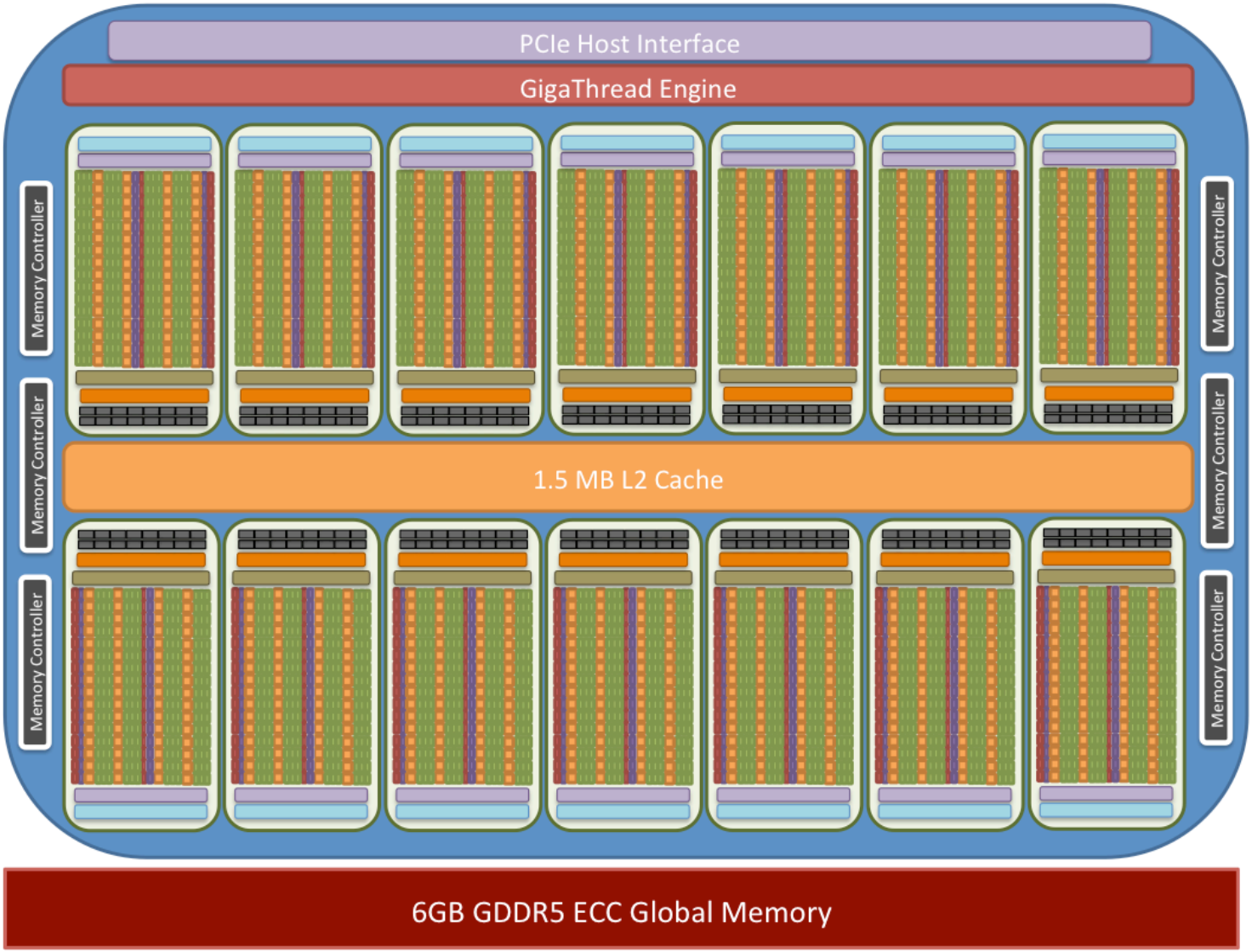
- Compute nodes can only be reached from via the `aprun` command
- Login and batch/service nodes do not have the same architecture as compute nodes
- Cross-compiling possible via the compiler wrappers: `cc, CC, ftn`
- Computes can only see the Lustre parallel file system
- Limit editing and compiling to home directories
- Archive data you want to keep long term

What about the GPUs?

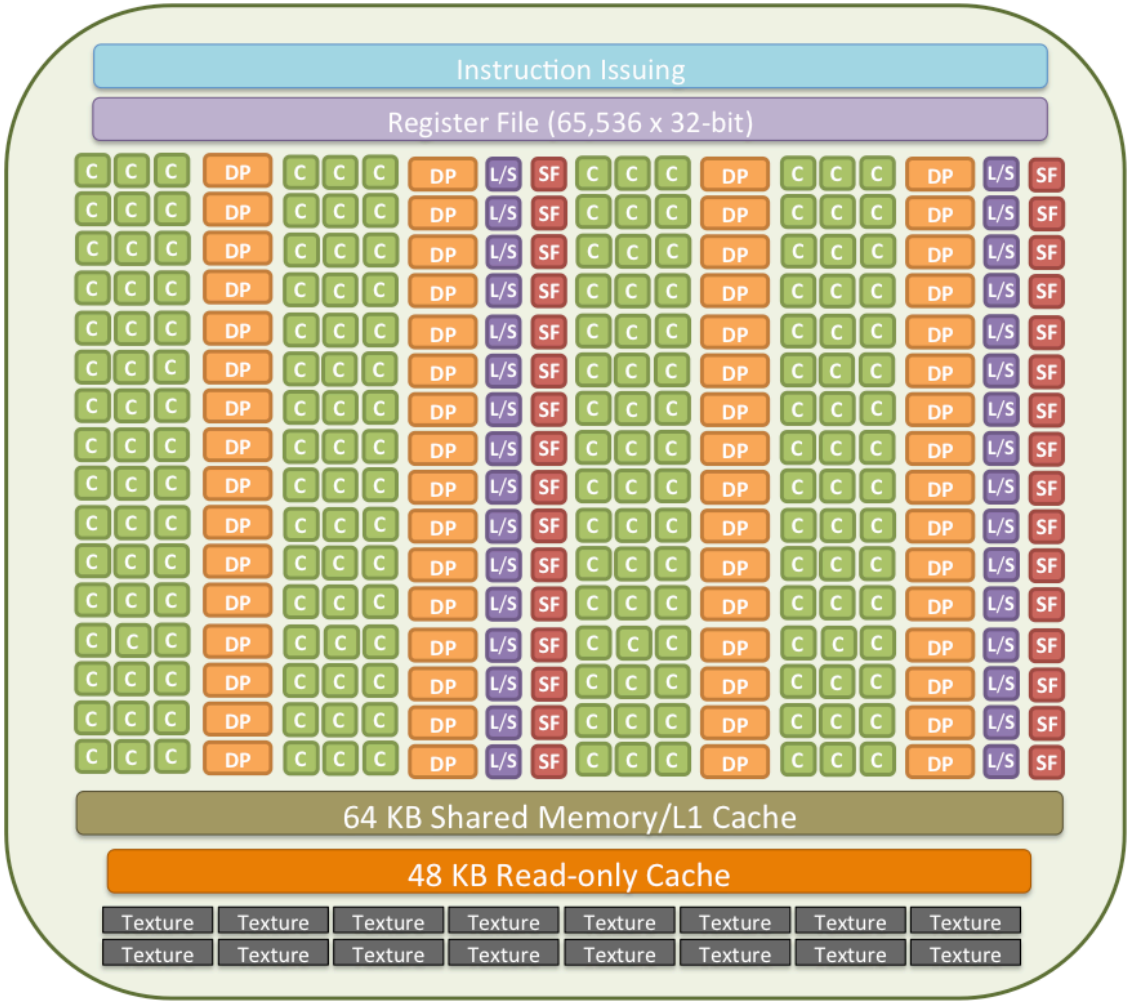
What is a CPU?



What is a GPU?



What is a GPU?



C single precision/integer CUDA core

DP double precision FP unit

L/S memory load/store unit

SF special function unit

Using the GPU

- Low level languages
  - CUDA, OpenCL
- Accelerated Libraries
  - cuBLAS, cuFFT, cuRAND, thrust, magma
- Compiler directives
  - OpenACC, OpenMP 4.x

Introduction to OpenACC

- Directive based acceleration specification
  - Programmer provides compiler annotations
  - Uses #pragma acc in C/C++, !$ acc in Fortran
- Several implementations available
  - PGI(NVIDIA), GCC, Cray, several others
- Programmer expresses parallelism in code
  - Implementation maps this to underlying hardware

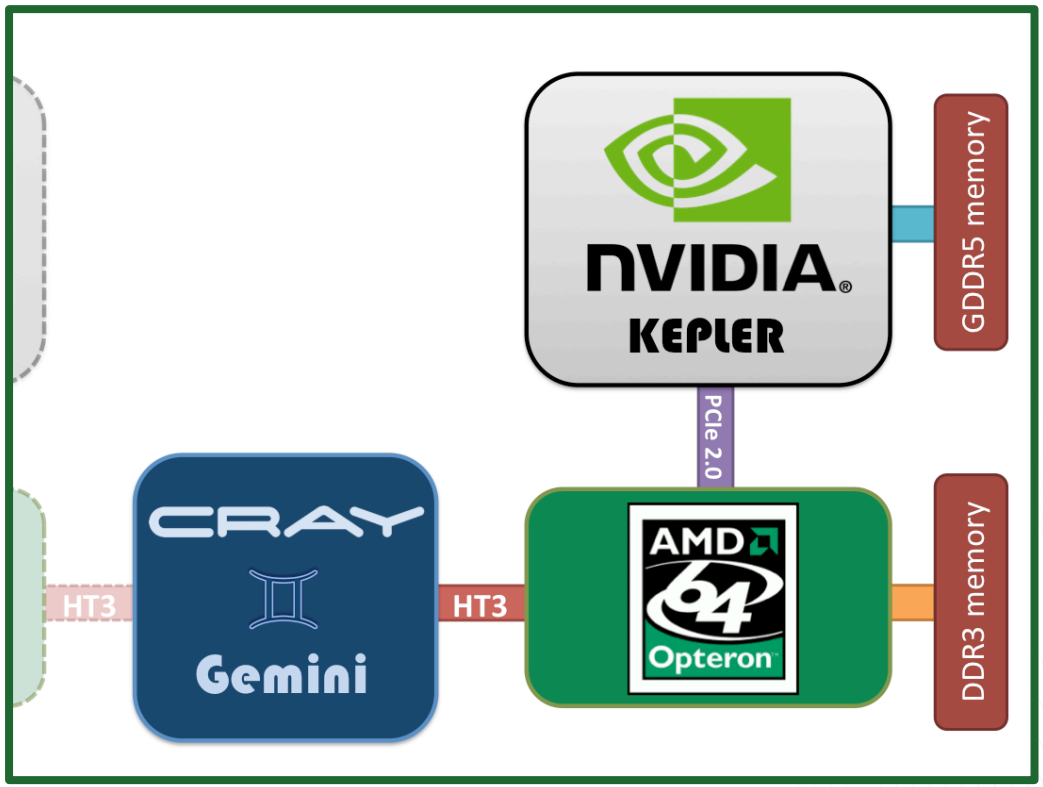
An example: Vector Addition

How to get the code?

```
$ qsub -l nodes=1,walltime=02:00:00 -A trn001 -I
$ module load cudatoolkit
$ cd ~
$ git clone https://github.com/olcf/ACC-intro.git
$ cd ~/ACC-intro/Vec
```

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit



Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

```
float *arr = (float*)malloc(n*sizeof(float));
```

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

```
float *arr = (float*)malloc(n*sizeof(float));
```

Allocate array on **GPU**

```
#pragma acc enter data create(arr[0:n], …)
```

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

```
float *arr = (float*)malloc(n*sizeof(float));
```

Allocate array on **GPU**

```
#pragma acc enter data create(arr[0:n], …)
```

Allocate array on **GPU** and copy data from **CPU**

```
#pragma acc enter data copyin(arr[0:n], …)
```

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

```
free(arr);
```

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

free(arr);

Remove array from **GPU**

#pragma acc exit data delete(arr[0:n], …)

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

free(arr);

Remove array from **GPU**
#pragma acc exit data delete(arr[0:n], …)

Remove array from **GPU** and copy data to **CPU**
#pragma acc exit data copyout(arr[0:n], …)

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

```
float *arr = (float*)malloc(n*sizeof(float));
```

Copy data from **CPU** to **GPU**

```
#pragma acc update device(arr[0:n], …)
```

Accelerating the code: Data

- CPU and GPU have distinct memory spaces and data operations are generally explicit

```
float *arr = (float*)malloc(n*sizeof(float));

Copy data from CPU to GPU
#pragma acc update device(arr[0:n], …)

Copy data from GPU to CPU
#pragma acc update host(arr[0:n], …)
```

Accelerating the code: Data

- Apply data operations to code
- Compile code with OpenACC enabled

```
cc -acc vec.c -o vec.out
```

- Check compiler generated info

```
cc -acc -Minfo vec.c -o vec.out
```

Accelerating the code: Offload

Parallelism is derived from **for** loops

```
for ( i=0; i<n; i++ ) {
  arr[i] = i;
}
```

Accelerating the code: Offload

Parallelism is derived from **for** loops

```
#pragma acc parallel loop default(present)
for ( i=0; i<n; i++ ) {
  arr[i] = i;
}
```

Accelerating the code: Offload

Parallelism is derived from **for** loops

```
float scalar = 0.0;

#pragma acc parallel loop default(present)
for ( i=0; i<n; i++ ) {
  scalar = scalar + arr[i];
}
```

Accelerating the code: Offload

The programmer is responsible for ensuring no race condition is created through parallelization.

```
float scalar = 0.0;

#pragma acc parallel loop default(present) reduction(+:scalar)
for ( i=0; i<n; i++ ) {
  scalar += arr[i];
}
```

Accelerating the code: Offload

Functions must be compiled for the accelerator

```
float my_func(int i) {
  return i + 1.0;
}

for ( i=0; i<n; i++ ) {
  arr[i] = my_func(i);
}
```

Accelerating the code: Offload

Functions must be compiled for the accelerator

```
#pragma acc routine
float my_func(int i) {
  return i + 1.0;
}

#pragma acc parallel loop default(present)
for ( i=0; i<n; i++ ) {
  arr[i] = my_func(i);
}
```

Accelerating the code

- Parallelize the three for loops, taking care of any reductions and function calls
- Compile code with OpenACC enabled

```
cc -acc vec.c -o ~/vec.out
```

- Check compiler generated info

```
cc -acc -Minfo vec.c -o ~/vec.out
```

- Run

```
cd $MEMBERWORK/trn001
aprun -n 1 ~/ACC-intro/vec.out
```

An example: SPH

Compiling and running the CPU code

```
$ cd ~/ACC-intro/SPH_Simple
$ make omp
$ cd /lustre/pfs1/cades-olcf-training/world-shared/$USER
$ export OMP_NUM_THREADS=16
$ aprun -n 1 -d 16 ~/ACC-intro/SPH_Simple/sph-omp.out
```

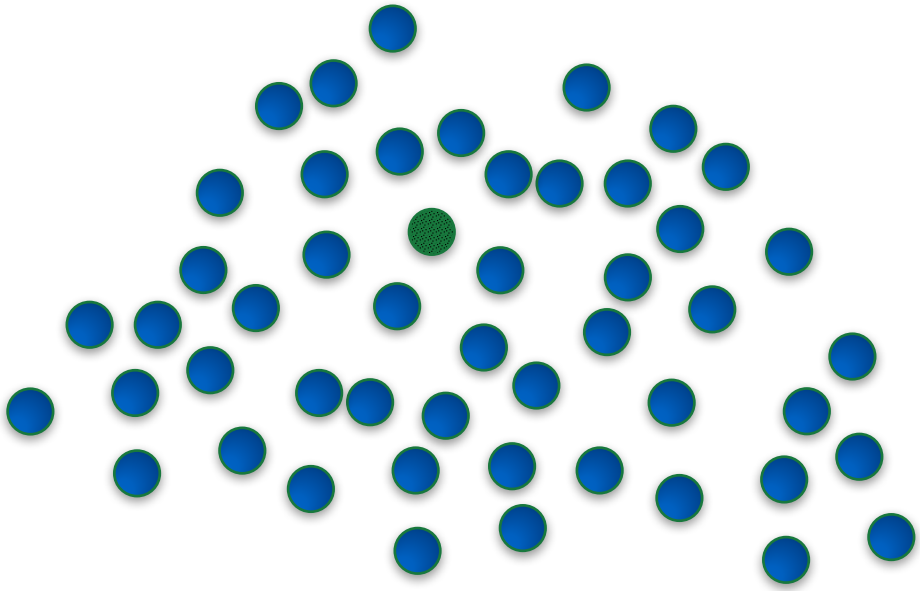
SPH

- **S**moothed **P**article **H**ydrodynamics
- Lagrangian formulation of fluid dynamics
- Fluid simulated by discrete elements(particles)
- Developed in the 70's to model galaxy formation
  - Later modified for incompressible fluids(liquids)

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{r} - \mathbf{r}_j|)$$

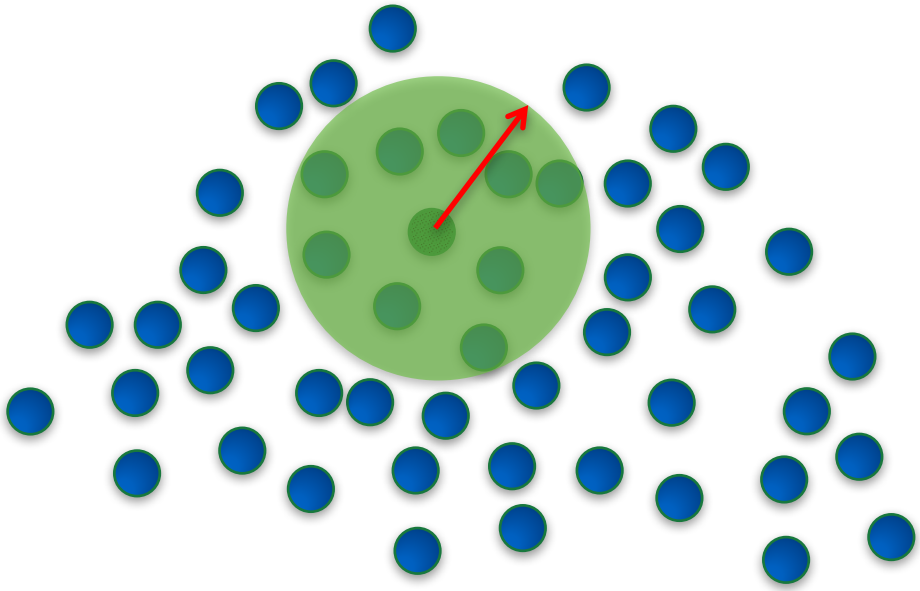
SPH

- Particles carry all information
  - position, velocity, density, pressure, …



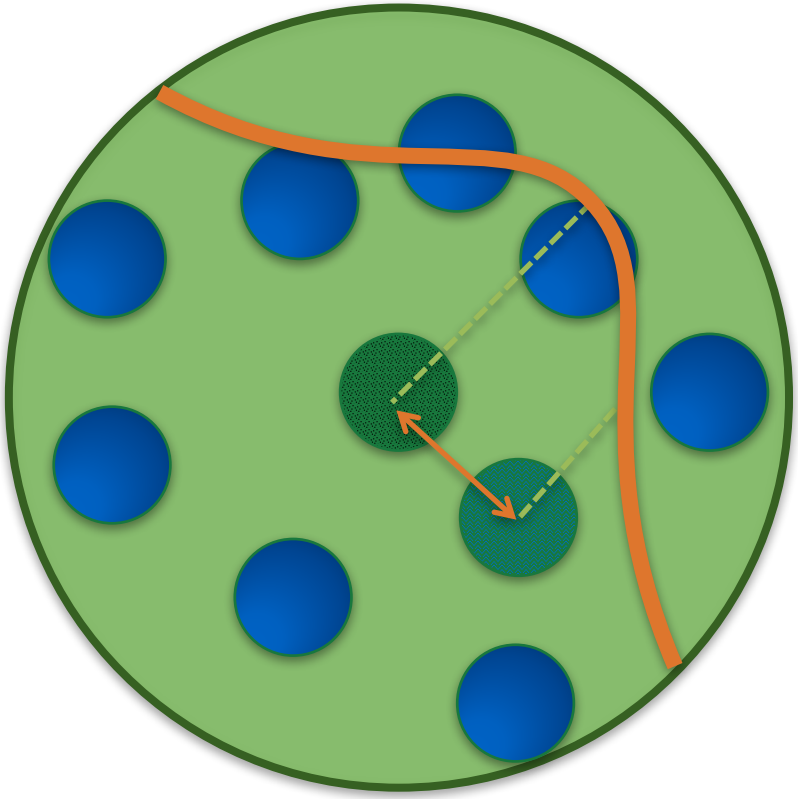
SPH

- Particles carry all information
  - position, velocity, density, pressure, …
- Only neighboring particles influence these values



SPH

- Particles carry all information
  - position, velocity, density, pressure, …
- Only neighboring particles influence these values



Example: density calculation

$$\rho_i = \sum_j m_j W(|\mathbf{r}_i - \mathbf{r}_j|)$$

Source code overview

- **fluid.c**
  - main() loop and all SPH related functions
- geometry.c
  - Creates initial bounding box
  - Only used for problem initialization
- fileio.c
  - Write particle positions in CSV format
  - Writes 30 files per simulation second to current directory

fluid.c

- Relies on C structures:
  - fluid_particle
    - position, velocity, pressure, density, acceleration
    - **fluid_particles** is array of **fluid_particle** structs
  - boundary_particle
    - position and surface normal
    - **boundary_particles** is array of **boundary_particle** structs
  - param
    - Global simulation parameters
    - **params** is instance of **param** struct

fluid.c functions

- main()
  - problem initialization
    - Set problem geometry
    - Initialize parameters
  - simulation time step loop (line 24)
    - updatePressures()
    - updateAcceleartions()
    - updatePositions()
    - writeFile()

fluid.c :: updatePressures()

- fluid.c line 150
- Updates the pressure for each particle
- Each particle must calculate change in density due to all other particles
- Density is used to compute pressure
- The pressure is used in acceleration calculation

fluid.c :: updateAccelerations()

- fluid.c line 221
- Updates acceleration for each particle
- Forces acting on particle:
  - Pressure due to surrounding particles
  - Viscosity due to surrounding particles
  - Surface tension modeled using surrounding particles
  - Boundary force due to boundary particles

fluid.c :: updatePositions()

- fluid.c line 287
- Updates particle velocity and positions
- Leap frog integration

Hands on: Data movement

- Move to GPU before simulation loop
  - fluid_particles[0:params.number_boundary_particles]
  - boundary_particles[0:params.number_fluid_particles]
  - params[0:1]
- Update host data before writing output files
- Don’t forget to cleanup before exiting

Compiling and running the GPU code

```
$ module load cudatoolkit
$ cd ~/ACC-intro/SPH_Simple
$ make acc
```

Hands on: Accelerate code

- Add loop parallelization to the following functions:
  - updatePressures()
  - updateAcceleartions()
  - updatePositions()
- Make sure any user defined functions called in the parallelized loops are compiled for the accelerator

Compiling and running the GPU code

```
$ make acc
$ cd $MEMBERWORK/trn001
$ aprun -n 1 ~/ACC-intro/Simple_SPH/sph-acc.out
```

Profiling & Optimizing

Profile: NVPROF

- Set up the environment:

```
$ module load cudatoolkit
$ export PMI_NO_FORK=1
```

- Launch nvprof passing your executable:

```
$ aprun -b -n 1 nvprof ./vec.out
```

  - A summary should be printed to your terminal on application exit

- To view a graphical timeline of your application save the output to a file:

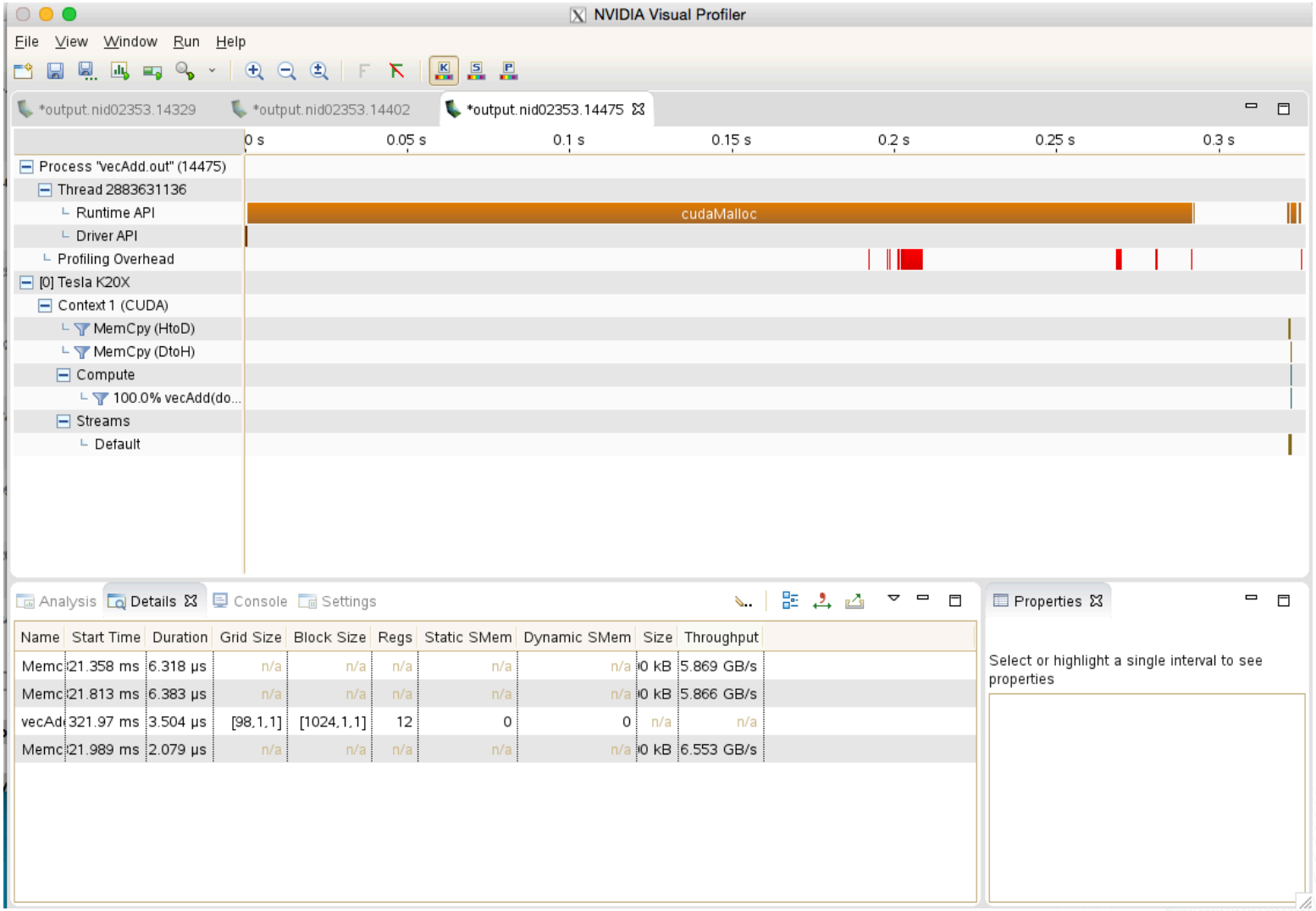
```
$ aprun -b -n 1 nvprof -o output.%h.%p ./vec.out
```

- Open NVIDIA Visual Profiler to view the result:

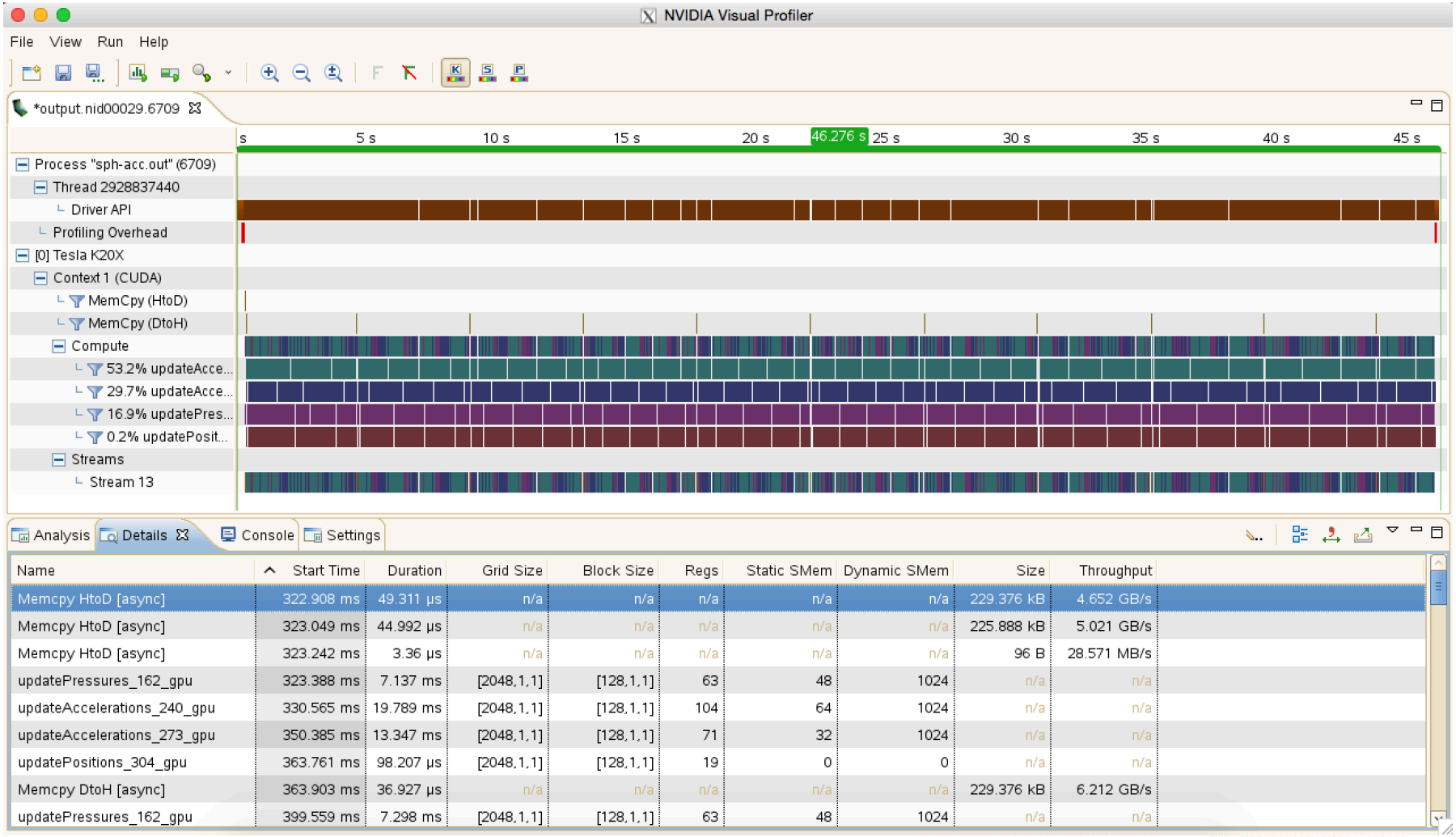
```
$ ssh -Y <username>@metis-login1.ornl.gov
$ nvvp &
```

  - In the menu: File > Open...

Profiling VecAdd



Profiling SPH



Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Size	Throughput
Memcpy HtoD [async]	322.908 ms	49.311 µs	n/a	n/a	n/a	n/a	n/a	229.376 kB	4.652 GB/s
Memcpy HtoD [async]	323.049 ms	44.992 µs	n/a	n/a	n/a	n/a	n/a	225.888 kB	5.021 GB/s
Memcpy HtoD [async]	323.242 ms	3.36 µs	n/a	n/a	n/a	n/a	n/a	96 B	28.571 MB/s
updatePressures_162_gpu	323.388 ms	7.137 ms	[2048,1,1]	[128,1,1]	63	48	1024	n/a	n/a
updateAccelerations_240_gpu	330.565 ms	19.789 ms	[2048,1,1]	[128,1,1]	104	64	1024	n/a	n/a
updateAccelerations_273_gpu	350.385 ms	13.347 ms	[2048,1,1]	[128,1,1]	71	32	1024	n/a	n/a
updatePositions_304_gpu	363.761 ms	98.207 µs	[2048,1,1]	[128,1,1]	19	0	0	n/a	n/a
Memcpy DtoH [async]	363.903 ms	36.927 µs	n/a	n/a	n/a	n/a	n/a	229.376 kB	6.212 GB/s
updatePressures_162_gpu	399.559 ms	7.298 ms	[2048,1,1]	[128,1,1]	63	48	1024	n/a	n/a

Accelerating the code: Optimize

OpenACC allows fine grain control over thread layout on the underlying hardware if desired.

```
#pragma acc parallel loop ... vector_length(32)
for ( i=1; i<n; i++ ) {
  arr[i] = 1;
}
```

Accelerating the code: Optimize

OpenACC allows fine grain control over thread layout on the underlying hardware if desired.

CUDA block size

```
#pragma acc parallel loop ... vector_length(32)
for ( i=1; i<n; i++ ) {
  arr[i] = 1;
}
```

Hands on: thread layout

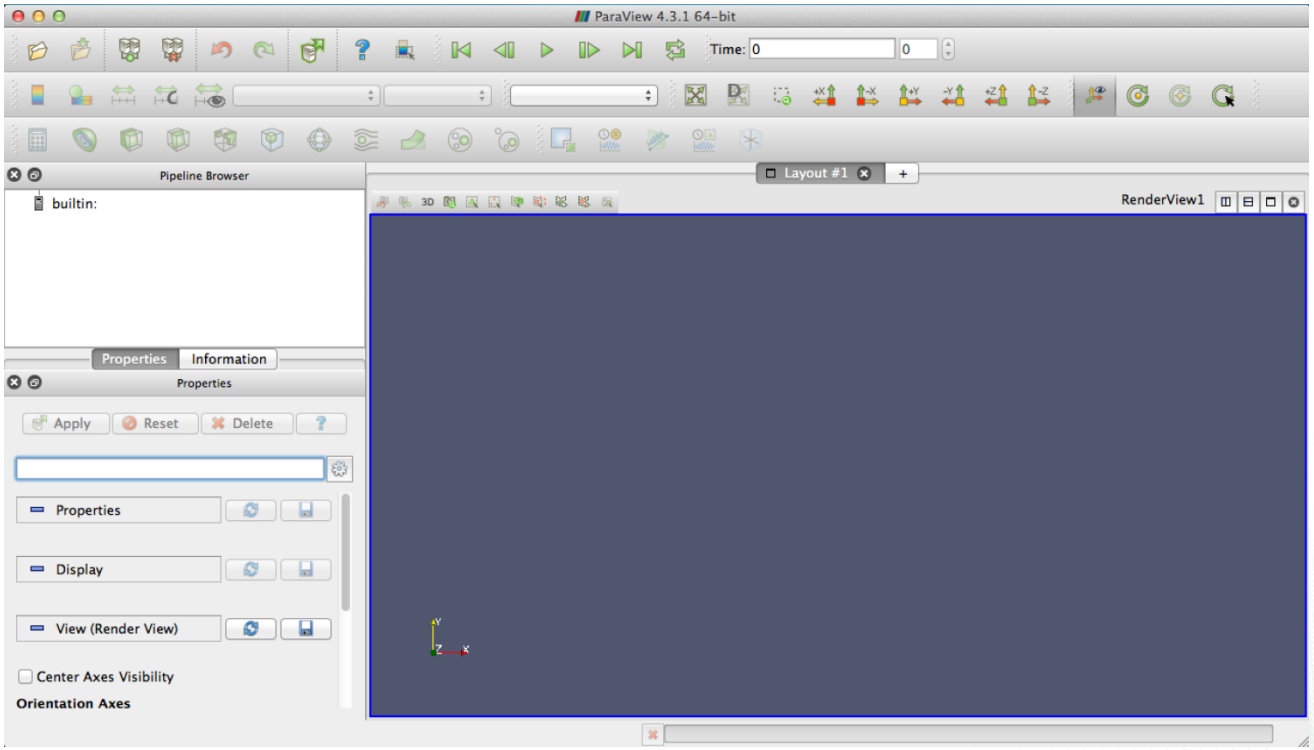
- Modify the vector_length in the following functions
  - updateAccelerations()
  - updatePressures()
- Use `nvprof` to determine which length is optimal
  - Try first with multiples of 32

Visualization of SPH

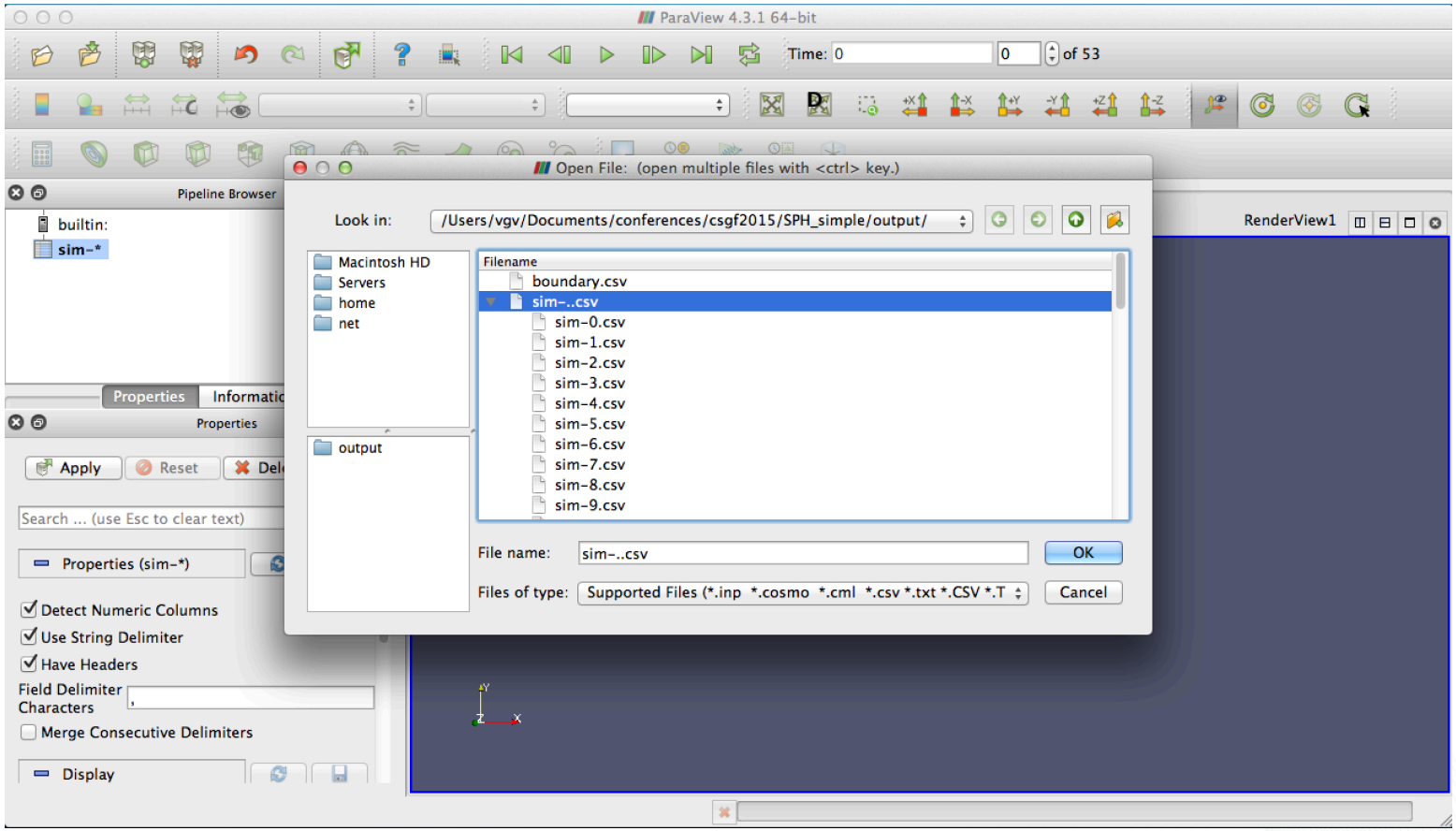
- Several applications available:
  - Paraview: http://www.paraview.org/
  - VisIt: https://wci.llnl.gov/simulation/computer-codes/visit/
  - and more.
- For simplicity, we will use the Paraview client
  - Download at: http://www.paraview.org/download/
- Transfer results to your local machine using your favorite data transfer tool:
  - `scp, bbcp, Globus` (see Supercomputing in a Nutshell slides for useful links)
- Much more powerful, can also be used in parallel to create simulations remotely

Visualization of SPH

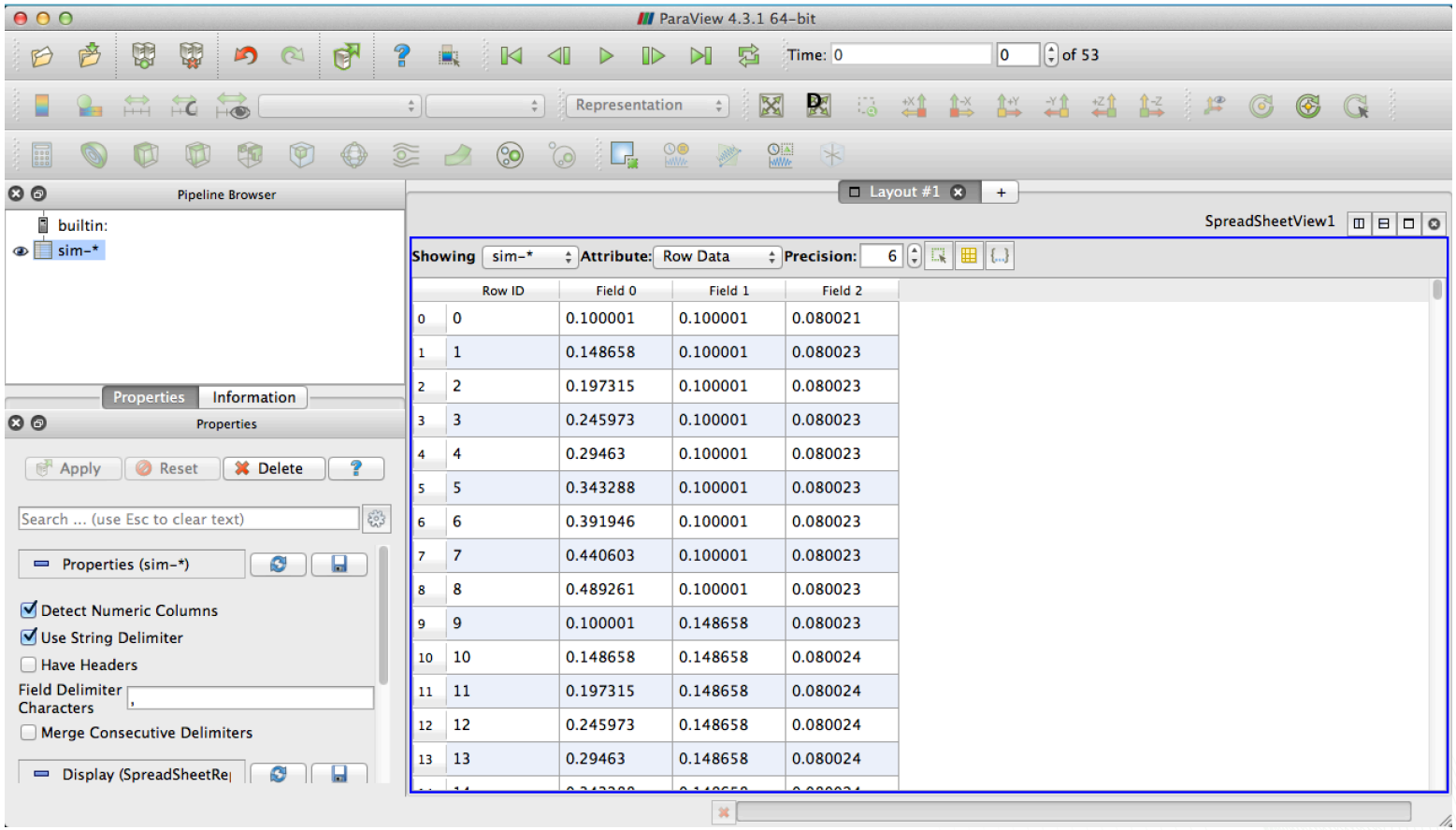
- Copy *.csv files to your local system:
  ```
  localhost:~$ scp <username>@metis-login1.ornl.gov:~/ACC-intro/SPH_Simple/*.csv .
  ```
- Open the Paraview client:



Visualization of SPH

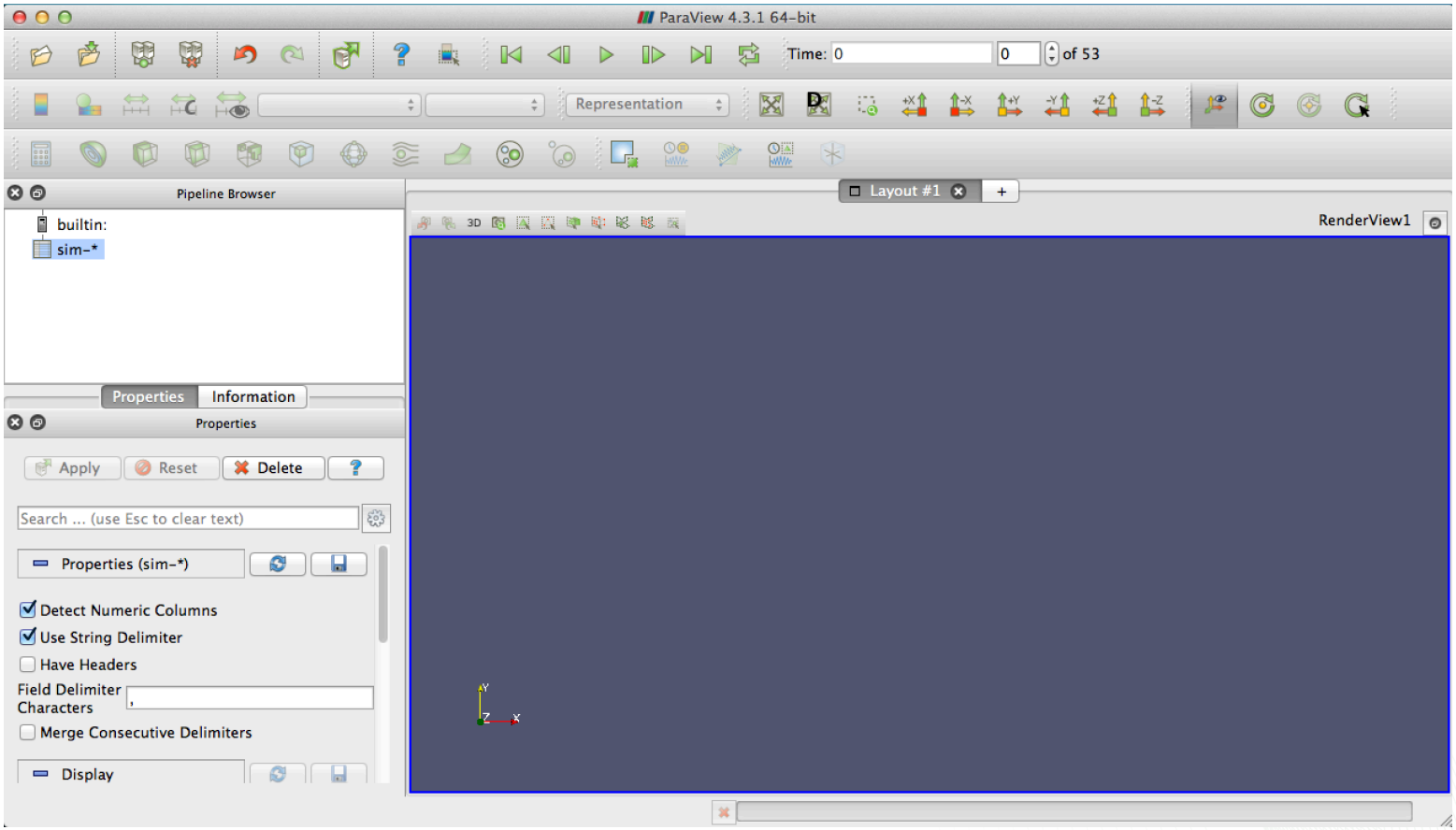


Visualization of SPH

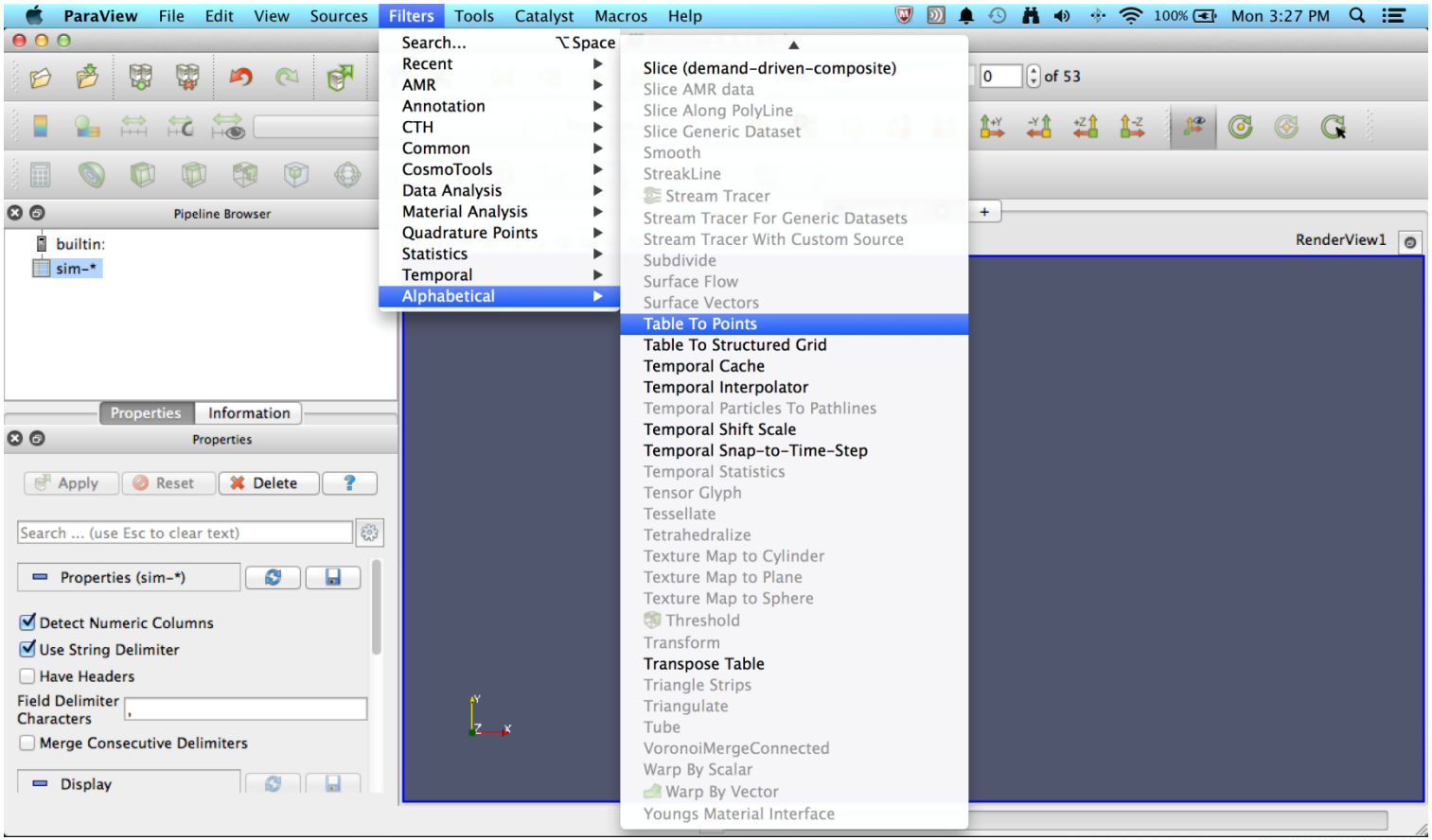


	Row ID	Field 0	Field 1	Field 2
0	0	0.100001	0.100001	0.080021
1	1	0.148658	0.100001	0.080023
2	2	0.197315	0.100001	0.080023
3	3	0.245973	0.100001	0.080023
4	4	0.29463	0.100001	0.080023
5	5	0.343288	0.100001	0.080023
6	6	0.391946	0.100001	0.080023
7	7	0.440603	0.100001	0.080023
8	8	0.489261	0.100001	0.080023
9	9	0.100001	0.148658	0.080023
10	10	0.148658	0.148658	0.080024
11	11	0.197315	0.148658	0.080024
12	12	0.245973	0.148658	0.080024
13	13	0.29463	0.148658	0.080024

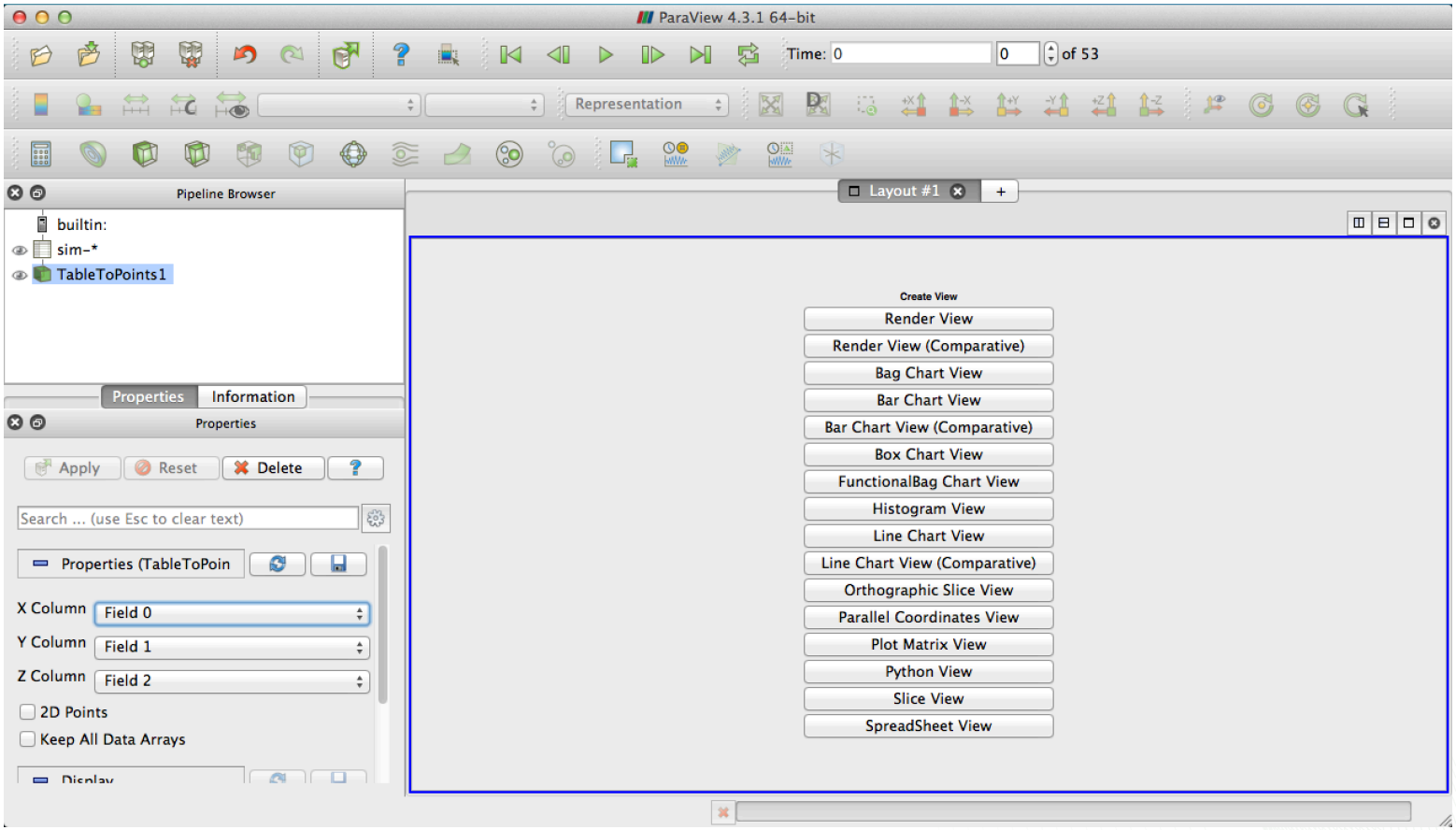
Visualization of SPH



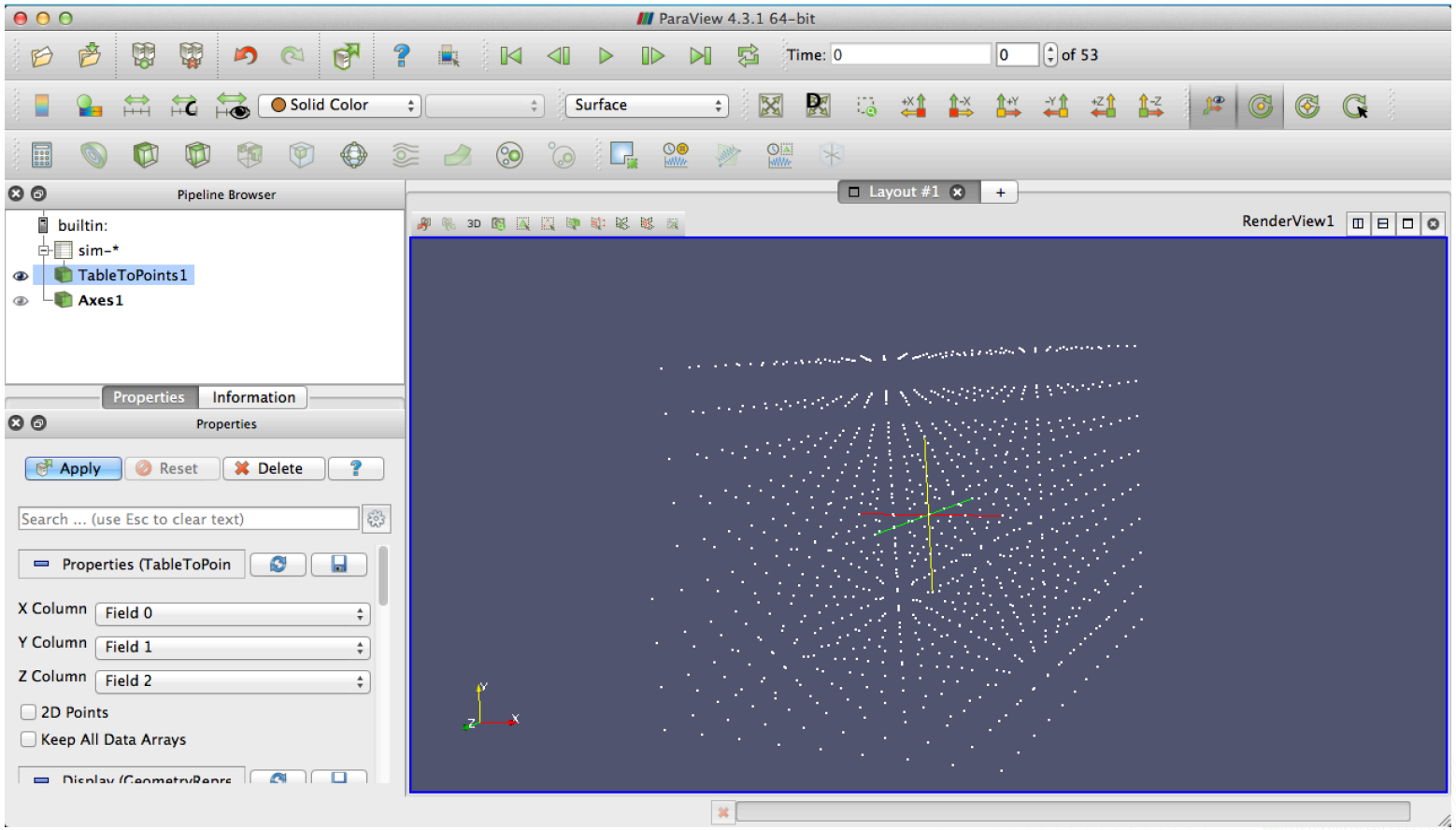
Visualization of SPH



Visualization of SPH

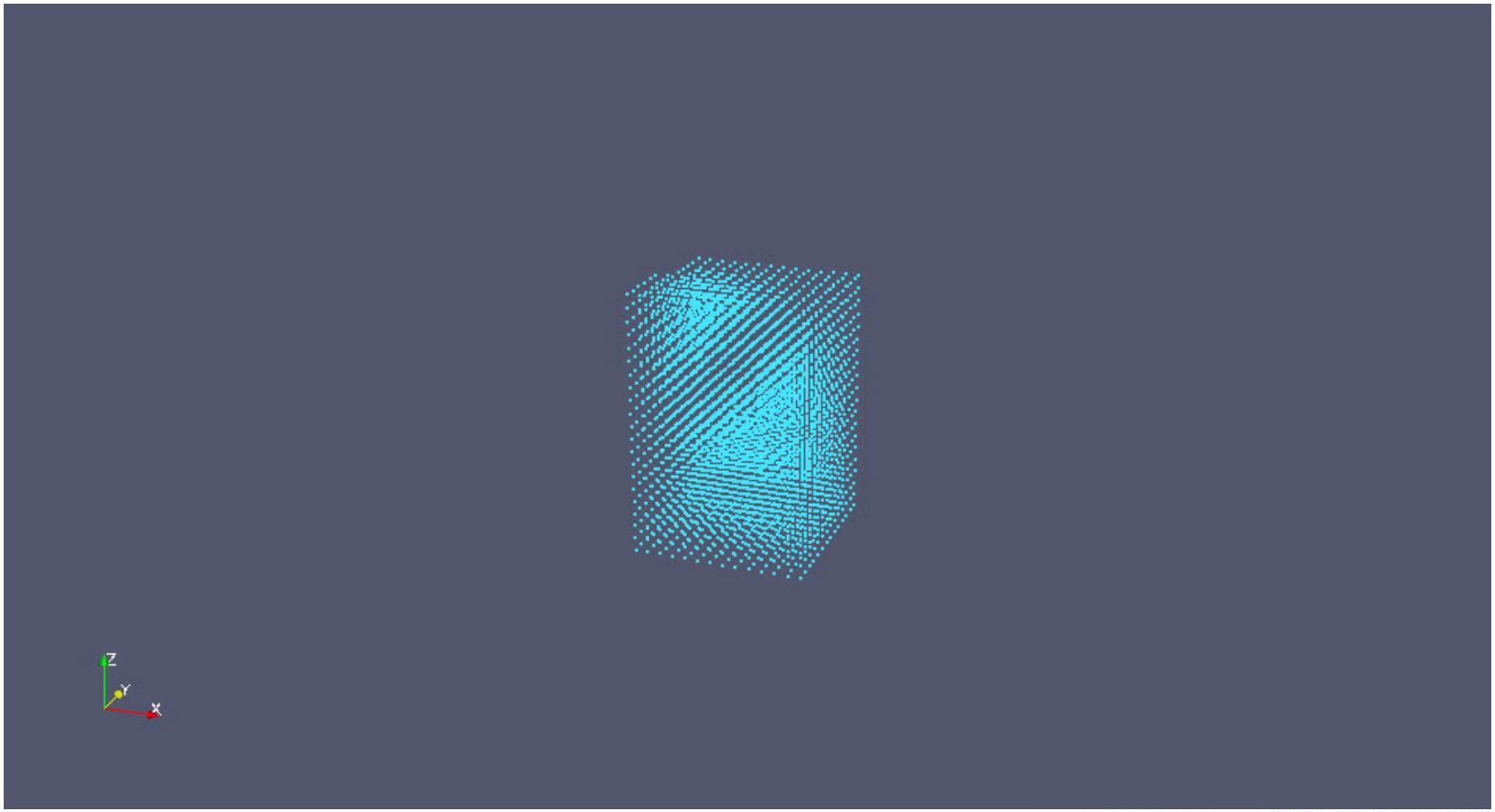


Visualization of SPH



Visualization of SPH

SPH with 4,000 particles



Thank you!

Questions?

Contact the OLCF at:
help@olcf.ornl.gov
(865) 241 - 6536

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Thank you!

Questions?

Contact the OLCF at:
help@olcf.ornl.gov
(865) 241 - 6536

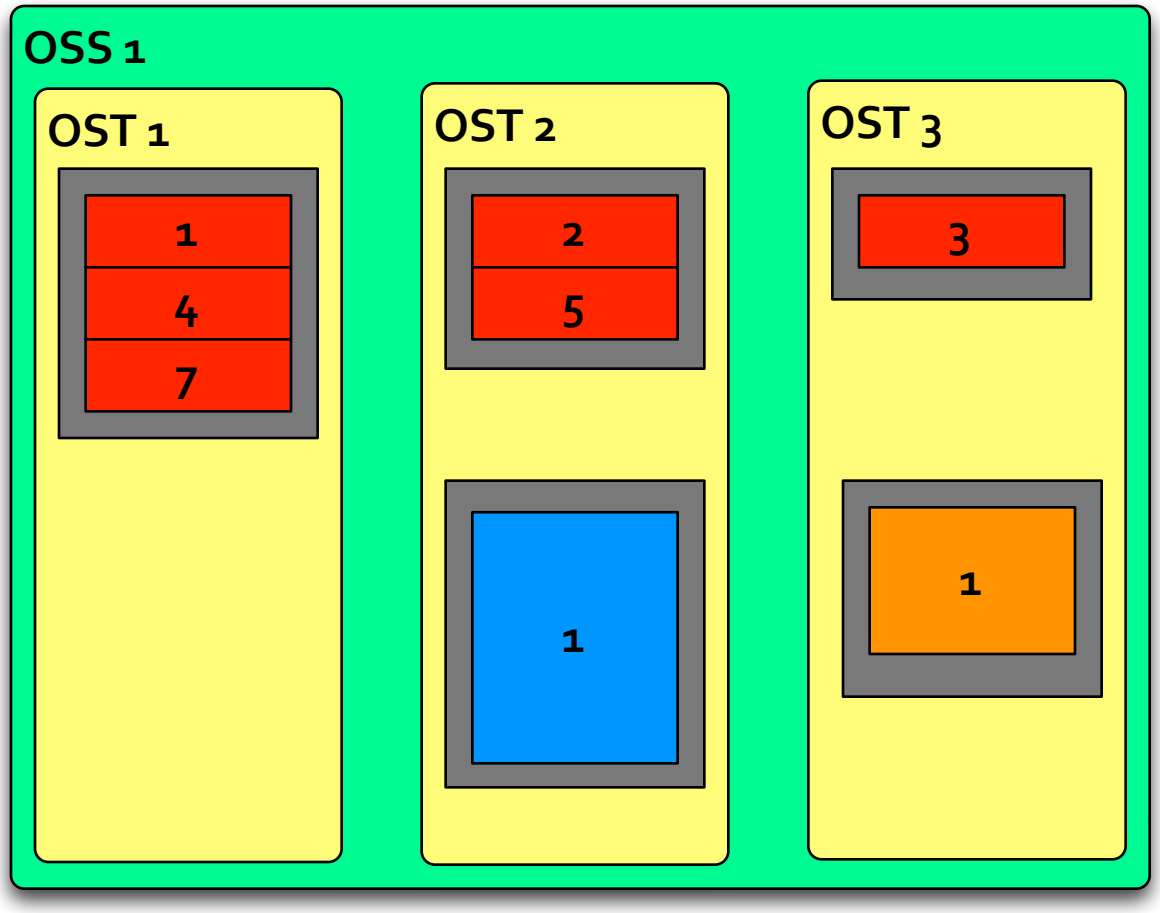
This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Extra Slides

Lustre at OLCF

- Spider 2 provides two partitions: atlas1 and atlas2
  - Each with 500 GB/s peak performance
  - 32 PB combined
  - 1,008 OSTs per partition
- Spaces on Spider 2 are scratch directories – *not backed up!*
- OLCF default striping:
  - Stripe count of 4
  - Stripe size of 1MB
- Recommendations:
  - Avoid striping across all OSTs, and in general, above 512 stripes
  - Do not change the default striping offset
  - When possible compile on home directories

Lustre Striping Basics



File A data

File B data

File C data

Lustre Object