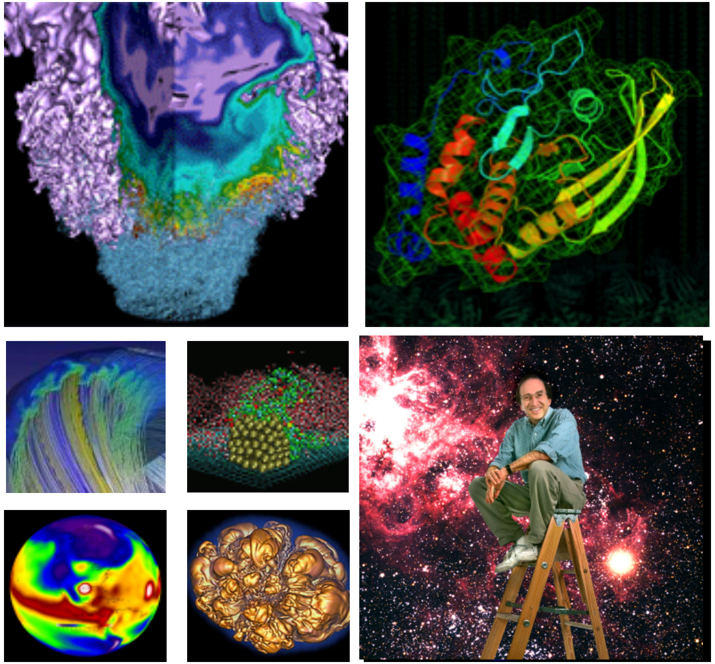
Preparing Applications for Perlmutter as an Exascale Waypoint



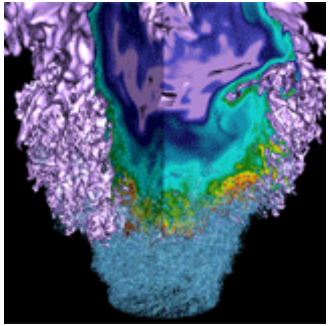
Brandon Cook, Brian Friesen, Jack Deslippe, Kevin Gott, Rahul Gayatri, Charlene Yang, Muaaz Gul Awan

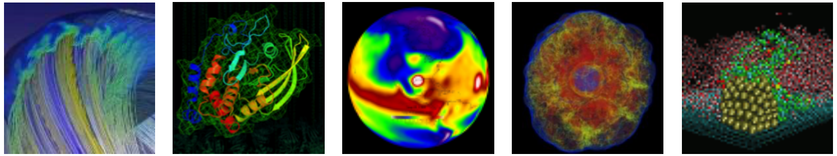
NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

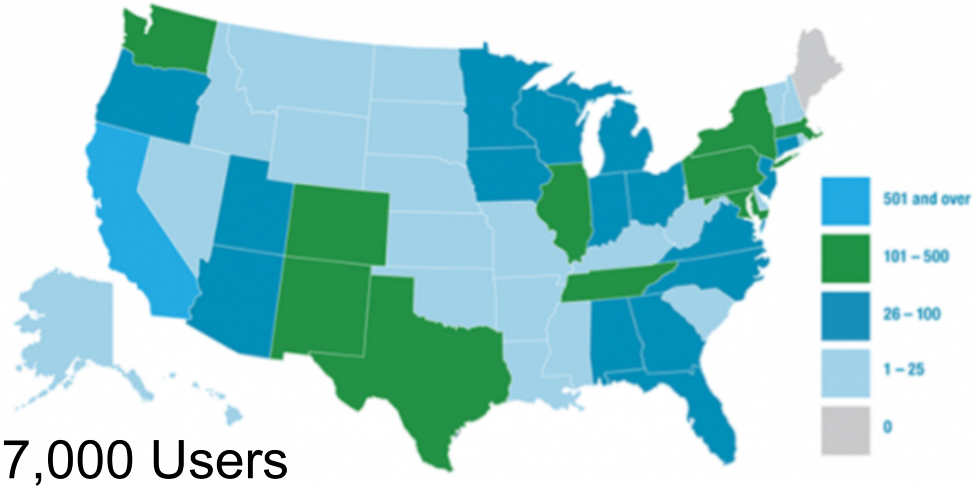
BERKELEY LAB

Perlmutter Overview

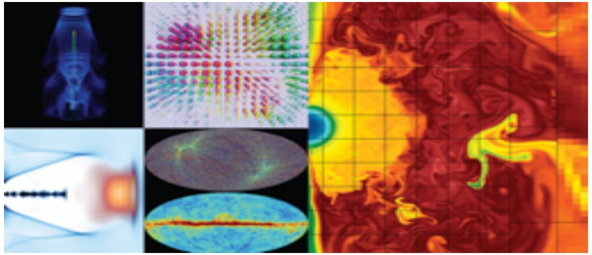




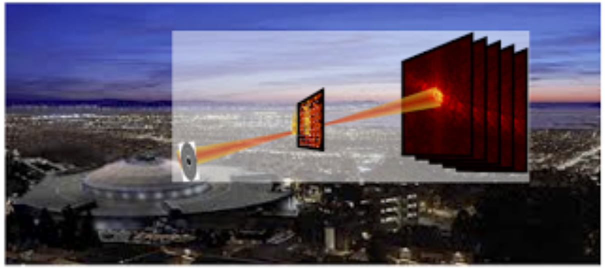
NERSC is the mission High Performance Computing facility for the DOE SC



7,000 Users
800 Projects
700 Codes
2000 NERSC citations per year

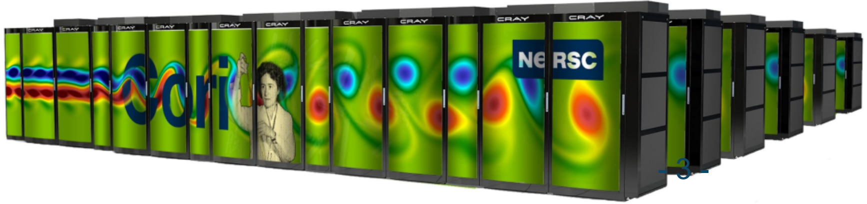


Simulations at scale



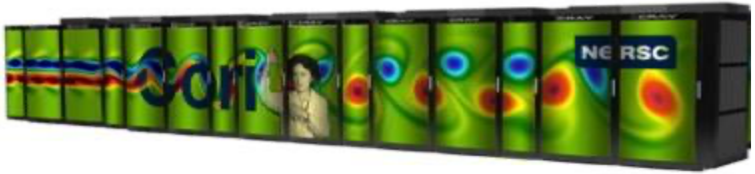
Data analysis support for DOE's experimental and observational facilities

Photo Credit: CAMERA



NERSC has a dual mission to advance science and the state-of-the-art in supercomputing

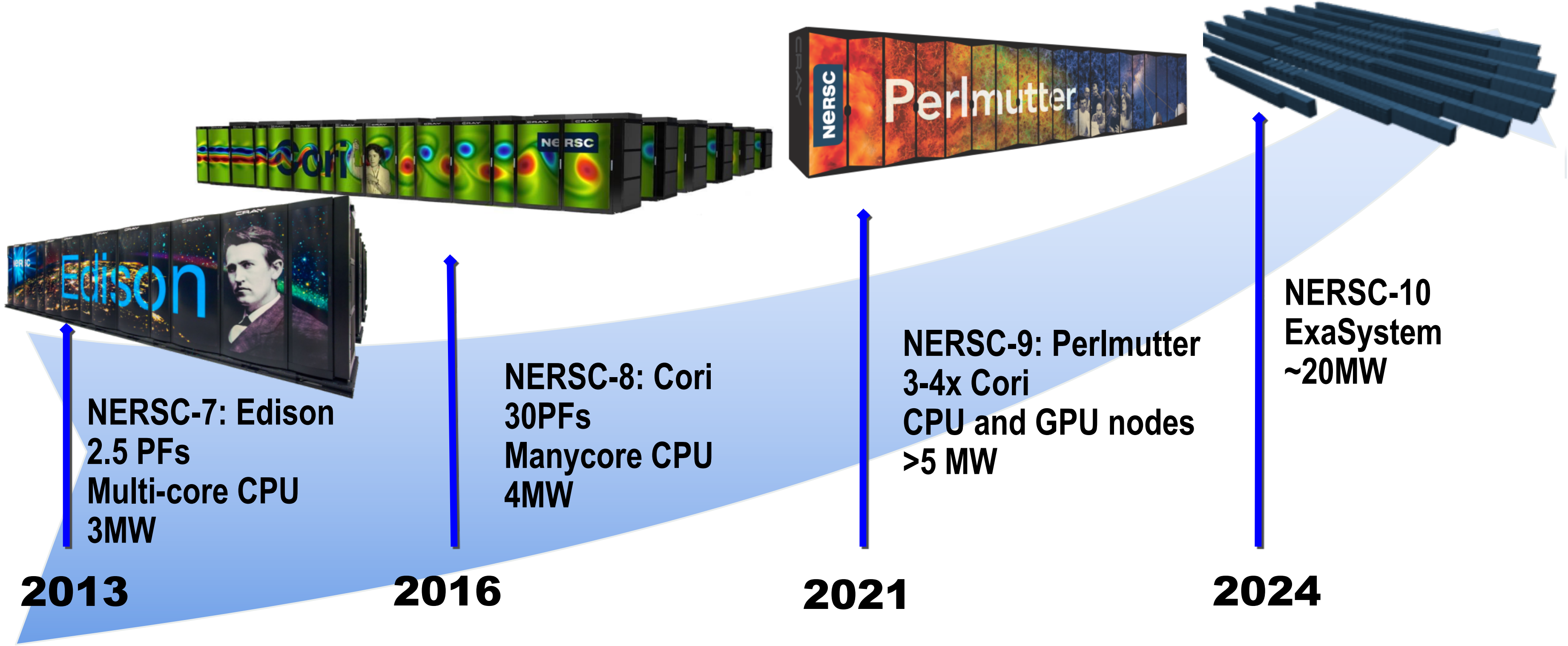
- We collaborate with computer companies years before a system’s delivery to deploy advanced systems with new capabilities at large scale
- We provide a highly customized software and programming environment for science applications
- We are tightly coupled with the workflows of DOE’s experimental and observational facilities – ingesting tens of terabytes of data each day
- Our staff provide advanced application and system performance expertise to users



NERSC-9 will be named after Saul Perlmutter

- Winner of 2011 Nobel Prize in Physics for discovery of the accelerating expansion of the universe.
- Supernova Cosmology Project, lead by Perlmutter, was a pioneer in using NERSC supercomputers to combine large scale simulations with experimental data analysis
- Login “saul.nersc.gov”

NERSC Systems Roadmap



NERSC-7: Edison
2.5 PFs
Multi-core CPU
3MW

NERSC-8: Cori
30PFs
Manycore CPU
4MW

NERSC-9: Perlmutter
3-4x Cori
CPU and GPU nodes
>5 MW

NERSC-10
ExaSystem
~20MW

2013

2016

2021

2024

Perlmutter is a Pre-Exascale System

Pre-Exascale Systems

2013 2016 2018 2020

Exascale Systems

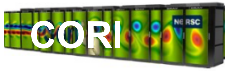
2021-2023

Argonne
IBM BG/Q

ORNL
Cray/NVidia K20

LLNL
IBM BG/Q

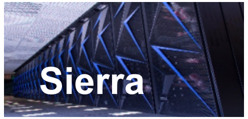
Argonne
Intel/Cray KNL



LBNL
Cray/Intel Xeon/KNL

LANL/SNL
Cray/Intel Xeon/KNL

ORNL
IBM/NVIDIA
P9/Volta



LLNL
IBM/NVIDIA
P9/Volta

LBNL
Cray/NVIDIA/AMD

Argonne
Intel/Cray

FRONTIER

ORNL
Cray/AMD

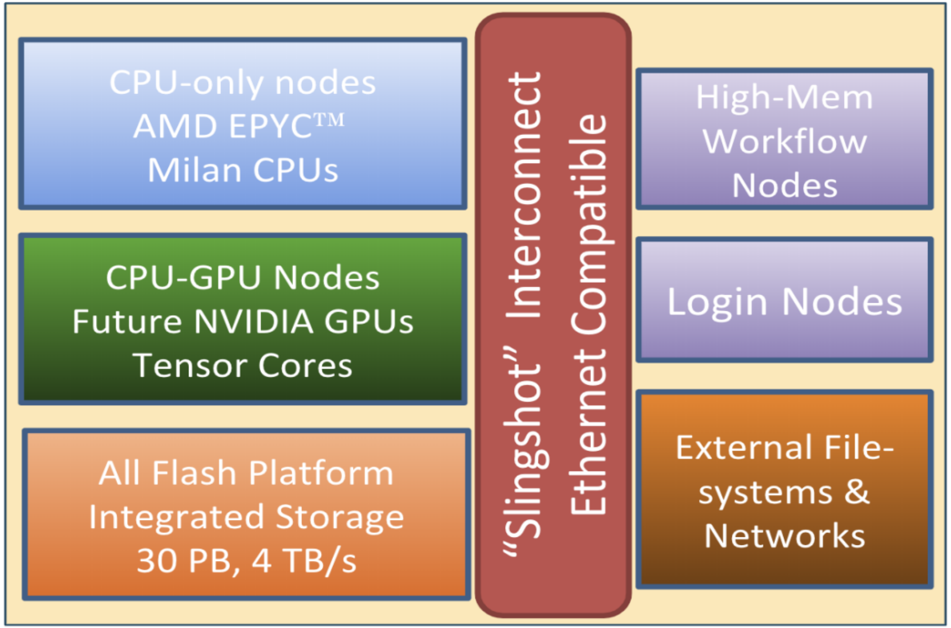
CROSSROADS

LANL/SNL
TBD

LLNL
Cray/?

Perlmutter: A System Optimized for Science

- GPU-accelerated and CPU-only nodes meet the needs of large scale simulation and data analysis from experimental facilities
- Cray "Slingshot" - High-performance, scalable, low-latency Ethernet-compatible network
- Single-tier All-Flash Lustre based HPC file system, 6x Cori's bandwidth
- Dedicated login and high memory nodes to support complex workflows



CPU-only nodes
AMD EPYC™
Milan CPUs

CPU-GPU Nodes
Future NVIDIA GPUs
Tensor Cores

All Flash Platform
Integrated Storage
30 PB, 4 TB/s

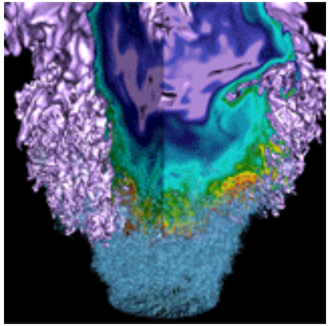
"Slingshot" Interconnect
Ethernet Compatible

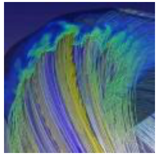
High-Mem
Workflow
Nodes

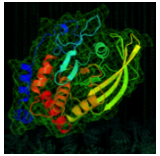
Login Nodes

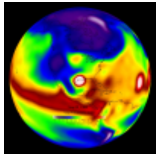
External File-systems &
Networks

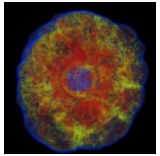
COE Activities

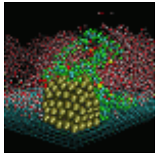












NERSC

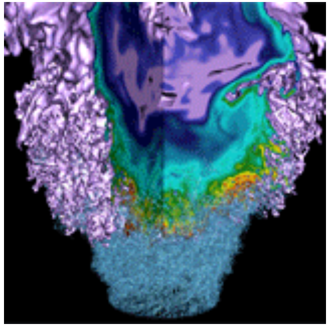
Vendor Resources Available to NESAP Teams

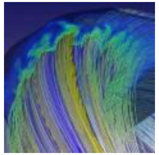
- Quarterly hackathons with NERSC, Cray, NVIDIA engineer
- General programming, performance and tools training
- Training events, such as the *CUDA Training Series*.
- Early access to Perlmutter
- Early access to Cori’s GPU testbed

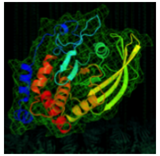
NERSC-9 Application Transition COE Hackathons

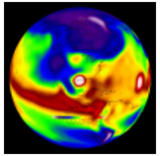
- One hackathon per quarter from 2019-2021
  - 3-4 code teams per hackathon
  - Priority given to NESAP teams
- NERSC, Cray, NVIDIA attendance
- 6-week ‘ramp-up’ period with code team+Cray/NVIDIA for ~6 weeks leading up to hackathon
  - Ensures everyone is fully prepared to work on hackathon day 1
- Tutorials/deep dives into GPU programming models, profiling tools, etc.
- Access to Cori GPU nodes

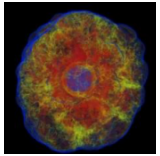
Data Analytics Stack and IO Considerations

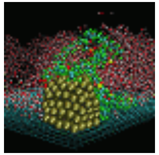






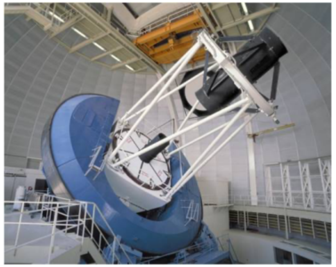


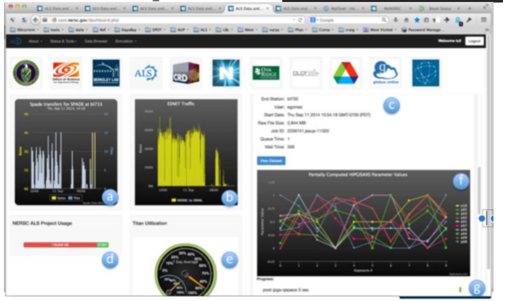




Analytics and Workflow Integration

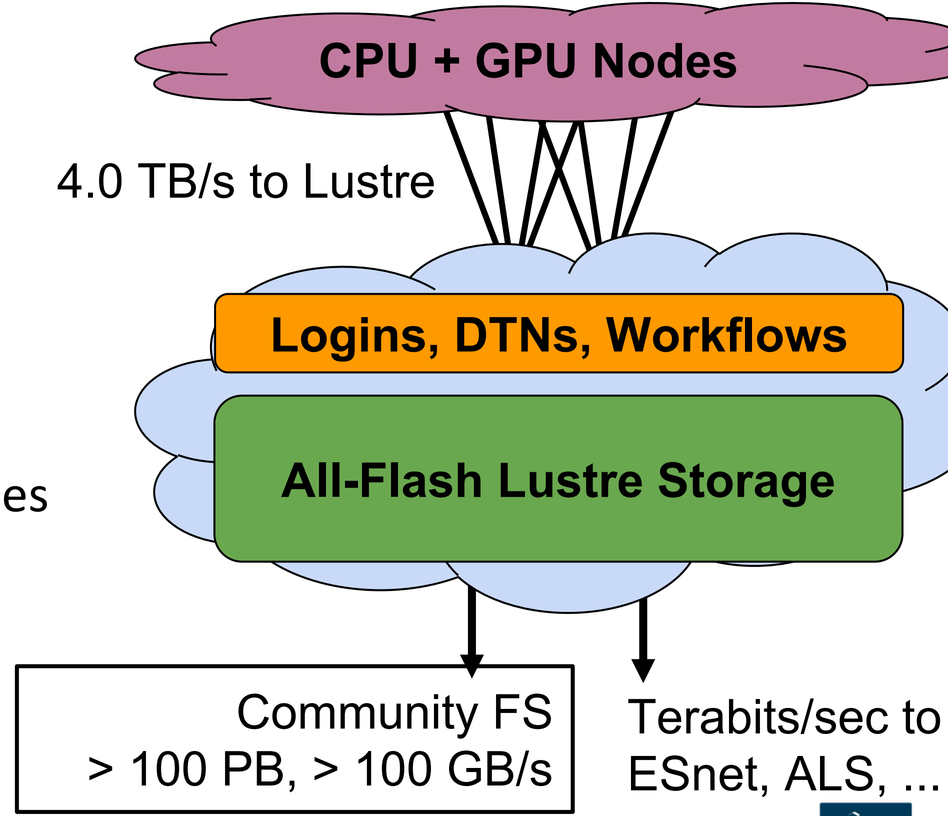
- **Software**
  - Optimized analytics libraries, includes Cray Analytics stack
  - Collaboration with NVIDIA for Python-based data analytics support
  - Support for containers
- **Perlmutter will aid complex end-to-end workflows**
  - **Slurm co-scheduling of multiple resources and real-time/deadline scheduling**
  - **Workflow nodes: container-based services**
    - Connections to scalable, user workflow pool (via Spin) with network/scheduler access
  - **High-availability workflow architecture and system resiliency for real-time use-cases**





All-flash file system

- **Fast across many dimensions**
  - > 4 TB/s sustained bandwidth
  - > 7,000,000 IOPS
  - > 3,200,000 file creates/sec
- **Usable for NERSC users**
  - > 30 PB usable capacity
  - Familiar Lustre interfaces
  - New data movement capabilities
- **Optimized for data workloads**
  - NEW small-file I/O improvements
  - NEW features for high IOPS, non-sequential I/O

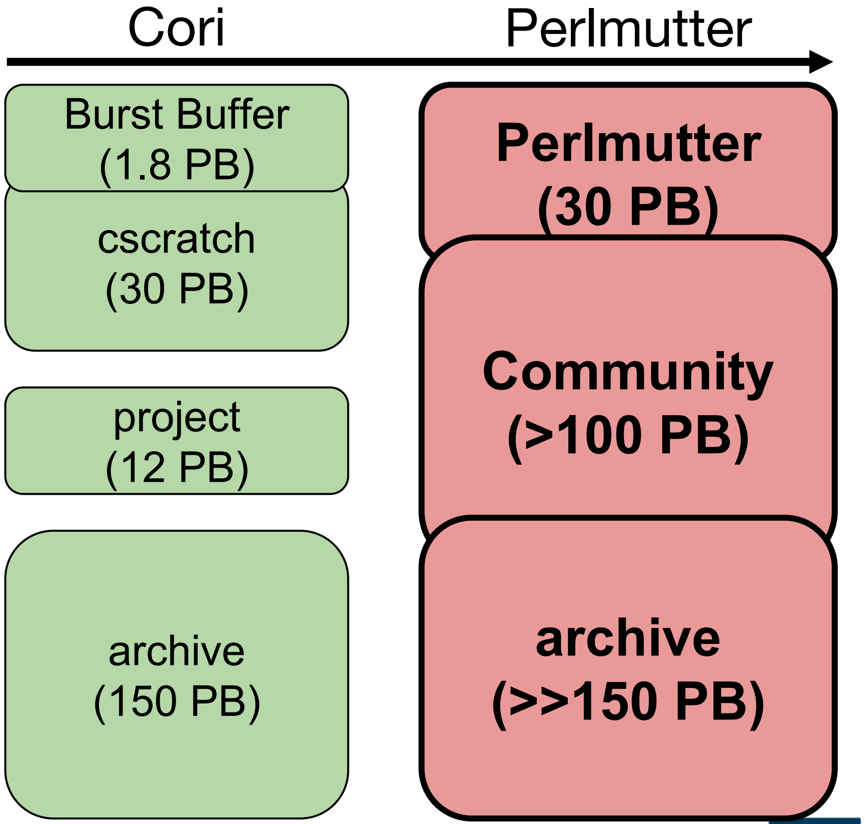


Maximizing I/O Performance

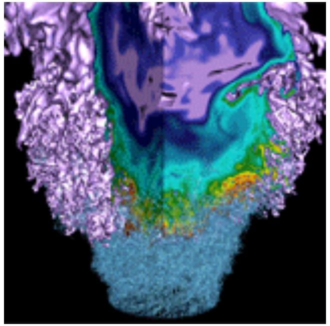
- **Usual best practices apply for best performance**
  - Use Lustre file striping
  - Avoid opening many files at once
  - Avoid using small files and small reads/writes
- **...but Perlmutter will be more forgiving**
  - Forget to stripe? Lustre **Progressive File Layouts** will do it for you automatically
  - Have many files? Lustre **Distributed Namespace** adds more metadata processing muscle
  - Must do small I/O? Lustre **Data-on-MDT** stores smaller files on IOPS-optimized storage

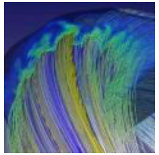
Data Movement

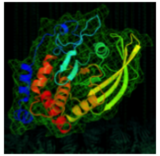
- Project file system replaced with Community file system
- NERSC-Cray collaboration will simplify data motion between Perlmutter & Community FS
- Feedback on how your workflow moves data between tiers will help define this data movement API

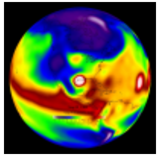


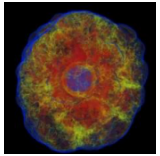
Programming & Performance Portability

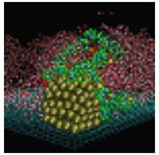




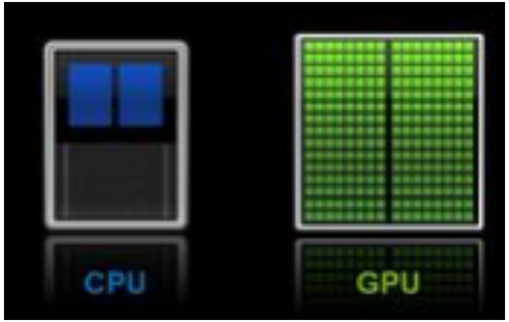








Exposing Parallelism



CPU (KNL)

- 68 cores
- 4 threads each
- 512-bit vectors
- pipelined instructions
- double precision
  - ~2000 way parallelism (68*4*8)

GPU (V100)

- 80 SM
- 64 warps per SM
- 32 threads per warp
- double precision
  - ~150,000+ way parallelism (80*64*32)

Data Locality

GPU Bus has low bandwidth compared to HBM.

Need to carefully manage data locality to avoid moving data back and forth often.

UVM can “potentially” help, but still need to think!

Performance Portability Strategy

Threads and Vectors (SMT, SIMT, SIMD).

1. SIMT ≅ SMT : What you tend to get when taking a GPU code and attempting a first pass portable version. This leaves SIMD on the GPU un-expressed. Leads to concept of coalescing.

1. SIMT ≅ SIMD : What you tend to get by default with OpenMP (!$OMP SIMD). Limits what you can vectorize on GPU to code with which the CPU can vectorize.

1. Use nested parallelism to map GPU SMs/Warps to CPU Cores/Threads and threads within Warps to Vector lanes. Still lose flexibility on the GPU.

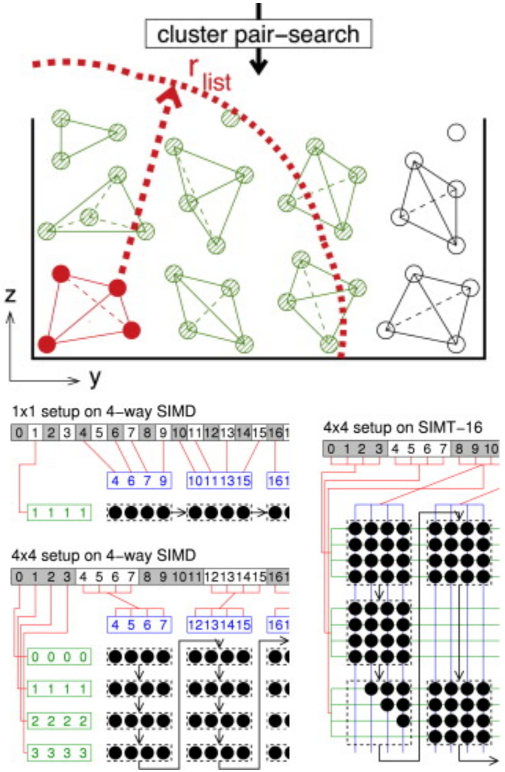
Abstractions and parallelism

Abstract operations for variable-width vectors

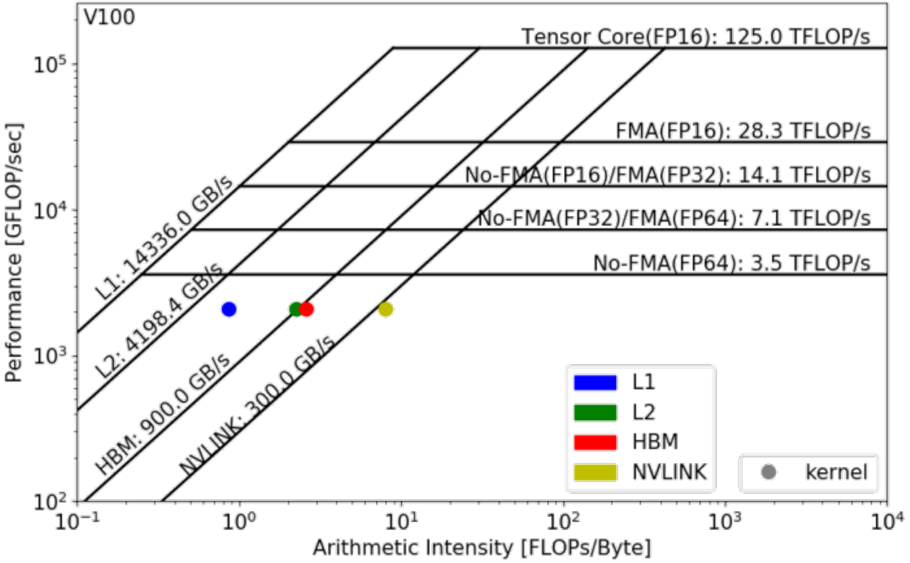
Example: Gromacs “cluster” pair-list adapts to 128, 256, 512-bit simd and 32 way SIMT by resizing the cluster.

Porting to new arch = implement abstract interface with intrinsics

*Effectiveness of this strategy depends on number of performance critical kernels



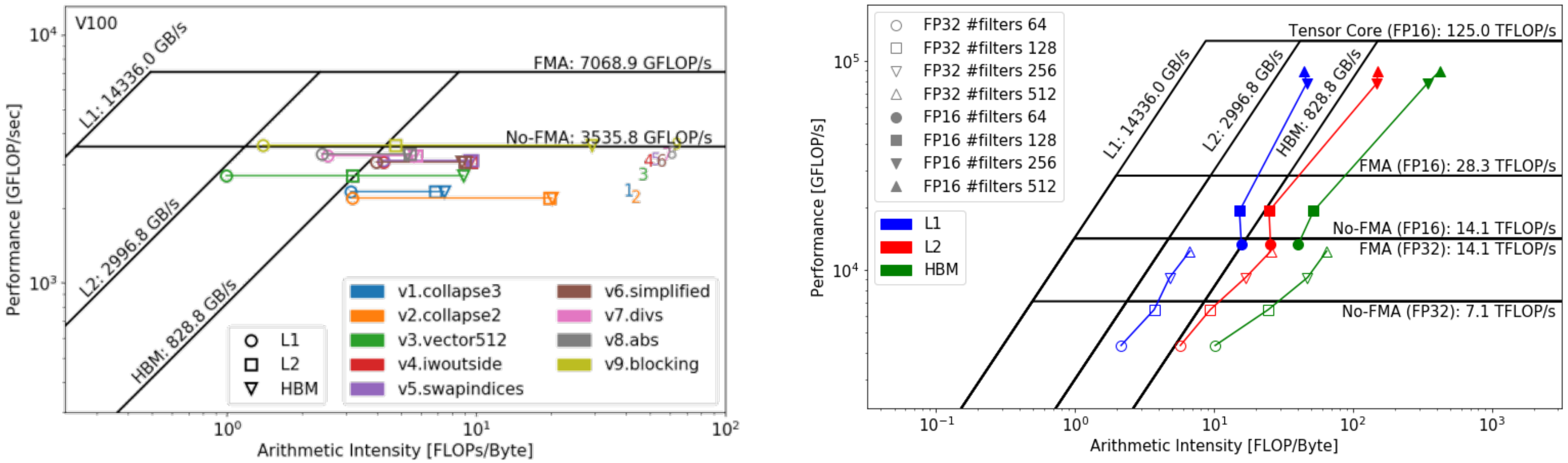
Roofline on NVIDIA GPUs



We have proposed a methodology to construct a hierarchical Roofline

- **that incorporates the full memory hierarchy**
  - L1, L2, HBM, System Memory (NVLink/PCIe)
- **and instruction types, data types…**
  - FMA/no-FMA/IntOps/…
  - FP64, FP32, FP16, …
  - CUDA core/Tensor core

Roofline on NVIDIA GPUs



Analyze performance and track optimization on both traditional HPC and Machine Learning applications. Left: Sigma-GPP from BerkeleyGW. Right: 2D convolution kernel from ResNet50 using TensorFlow.

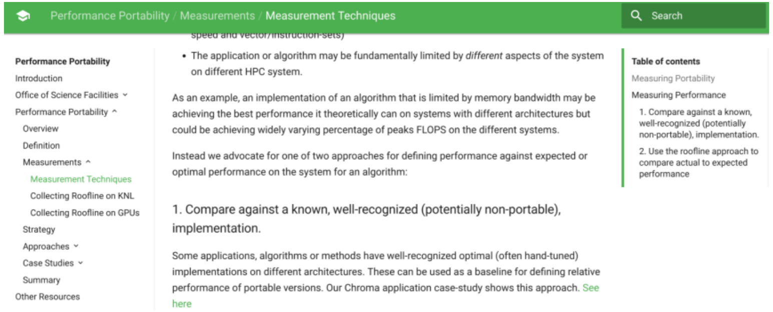
Performance Portability Options

- **Abstractions**
  - identify and use appropriate abstractions to flexibly expose the parallelism in a problem
  - account for potential switch in algorithm
- **Use a library when possible**
- **Programming model support**
  - C++ templates with CUDA/ CPU intrinsics, Kokkos, Raja, OpenMP, OpenACC, CUDA Fortran, and more

Engaging around Performance Portability

NERSC is working with PGI to enable OpenMP GPU acceleration with PGI compilers

- Ensures continuity of OpenMP added to NERSC apps for N8
- Co-design with PGI to prioritize OpenMP features for GPU
- Use lessons learned to influence future versions of OpenMP
- Monitoring SOLLVE efforts


Performance Portability / Measurements / Measurement Techniques
Search
Performance Portability
Introduction
Office of Science Facilities
Performance Portability
Overview
Definition
Measurements
Measurement Techniques
Collecting Roofline on KNL
Collecting Roofline on GPUs
Strategy
Approaches
Case Studies
Summary
Other Resources
Table of contents
Measuring Portability
Measuring Performance
1. Compare against a known, well-recognized (potentially non-portable), implementation.
2. Use the roofline approach to compare actual to expected performance

speed and vector/instruction-sets)

- The application or algorithm may be fundamentally limited by *different* aspects of the system on different HPC system.

As an example, an implementation of an algorithm that is limited by memory bandwidth may be achieving the best performance it theoretically can on systems with different architectures but could be achieving widely varying percentage of peaks FLOPS on the different systems.

Instead we advocate for one of two approaches for defining performance against expected or optimal performance on the system for an algorithm:

1. Compare against a known, well-recognized (potentially non-portable), implementation.

Some applications, algorithms or methods have well-recognized optimal (often hand-tuned) implementations on different architectures. These can be used as a baseline for defining relative performance of portable versions. Our Chroma application case-study shows this approach. See here

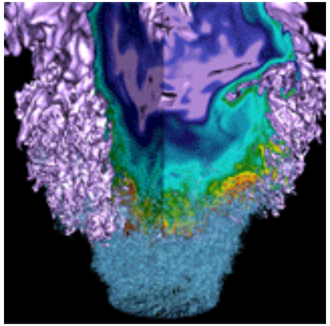
NERSC collaboarting with OLCF and ALCF on development of performanceportability.org

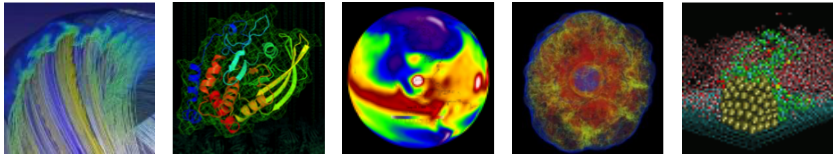
- Are you part of an ECP ST project? Interested in contributing a NERSC hosted training?
- kokkos, flang, SLATE, CI/Gitlab, spack

OpenMP for GPUs

- **OpenMP 5.0 improvements for accelerators**
  - Unified memory support
  - Implicit declare target
- **NERSC is collaborating with NVIDIA and OpenMP committee to enable OpenMP GPU acceleration in PGI compilers**
  - Co-design with application requirements
- **Tell us your experience!**
  - Techniques that work? Failures?

Application readiness



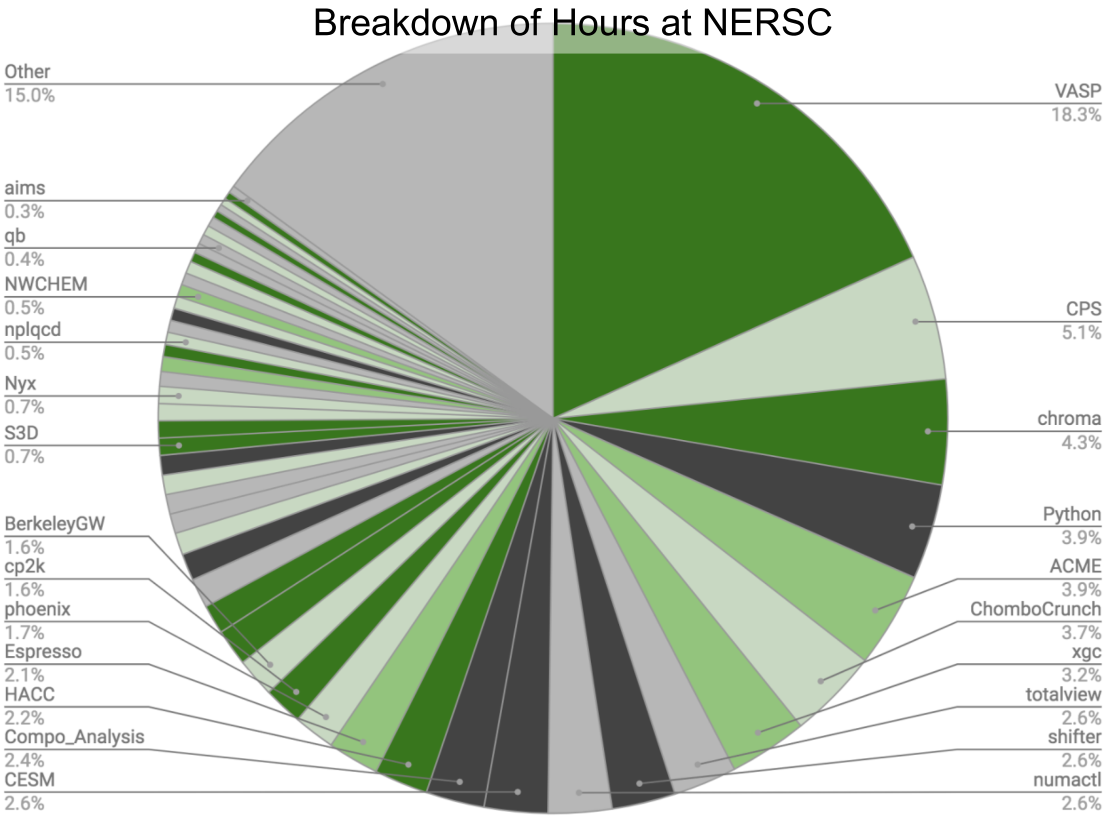


Application Readiness Strategy for Perlmutter

NERSC’s Challenge

How to enable NERSC’s diverse community of 7,000 users, 750 projects, and 700 codes to run on advanced architectures like Perlmutter and beyond?

GPU Readiness Among NERSC Codes (Aug’17 - Jul’18)



GPU Status & Description	Fraction
Enabled: Most features are ported and performant	32%
Kernels: Ports of some kernels have been documented.	10%
Proxy: Kernels in related codes have been ported	19%
Unlikely: A GPU port would require major effort.	14%
Unknown: GPU readiness cannot be assessed at this time.	25%

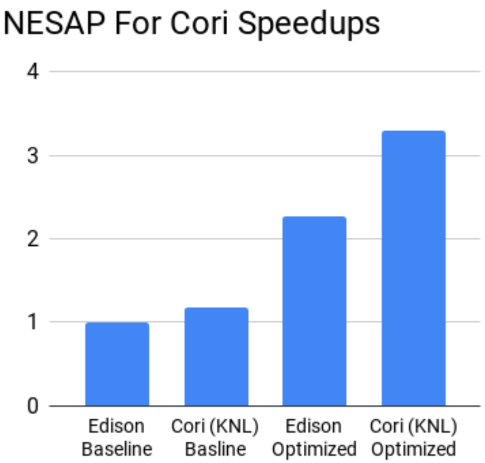
A number of applications in NERSC workload are GPU enabled already.

We will leverage existing GPU codes from CAAR + Community

Application Readiness Strategy for Perlmutter

How to transition a workload with 700 Apps? NESAP

- ~25 Projects selected from competitive application process with reviews
- ~15 postdoctoral fellows
- Deep partnerships with every SC Office area
- Leverage vendor expertise and hack-a-thons
- **Knowledge transfer through documentation and training for all users**
- **Optimize codes with improvements relevant to multiple architectures**



NERSC Exascale Science Application Program (NESAP)

Simulation
~12 Apps

Data Analysis
~8 Apps

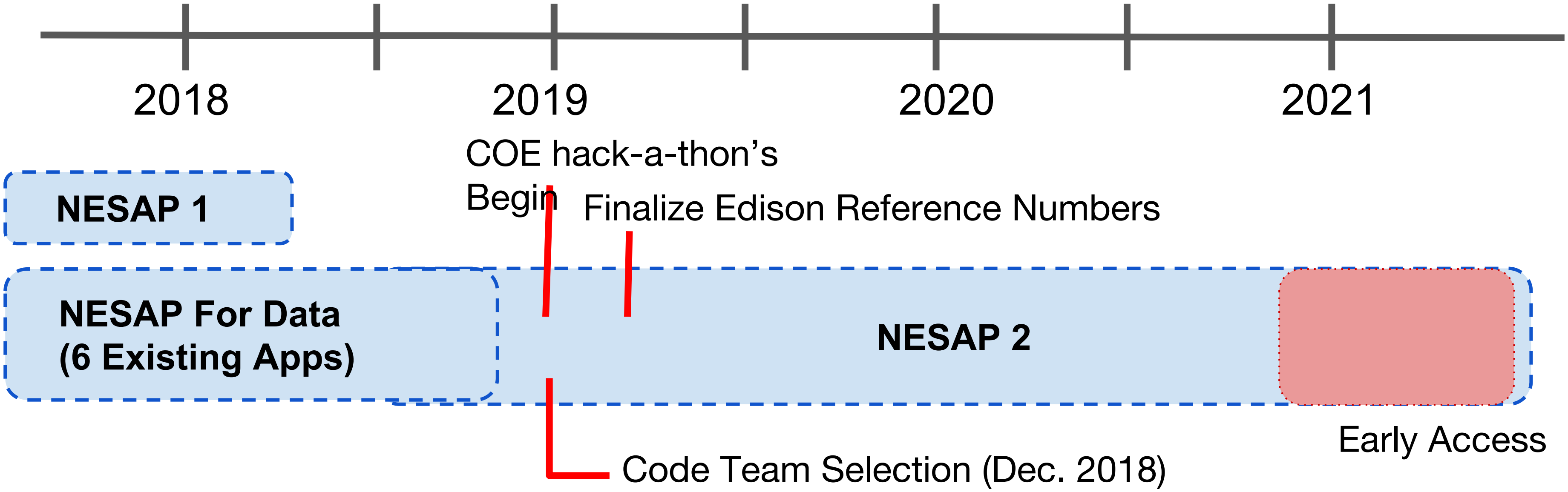
Learning
~5 Apps

- Based on successful NESAP for Cori program, similar to CAAR and ESP
- Details: https://www.nersc.gov/users/application-performance/nesap/

Selected ECP NESAP engagements

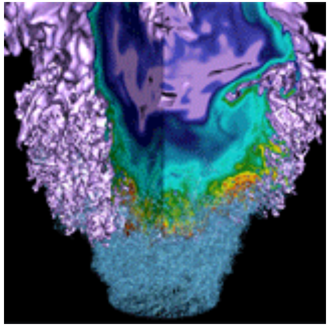
WDMAPP	Subsurface	EXAALT
NWChemEx	ExaBiome	ExaFEL
WarpX (AMReX)	ExaLearn	

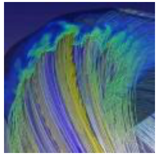
NESAP 2 Timeline

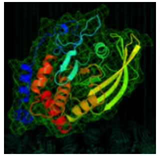


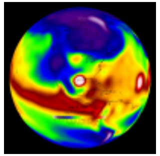
2018
2019
2020
2021
COE hack-a-thon's Begin
Finalize Edison Reference Numbers
NESAP 1
NESAP For Data (6 Existing Apps)
NESAP 2
Early Access
Code Team Selection (Dec. 2018)

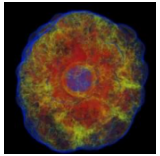
Application readiness: case studies

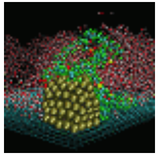






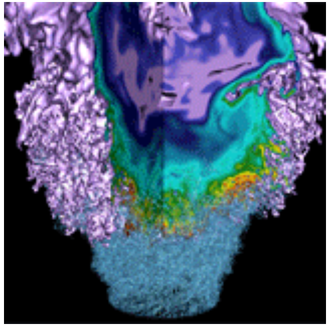


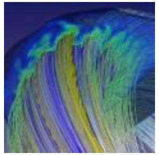


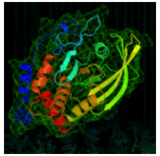


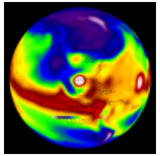
NERSC

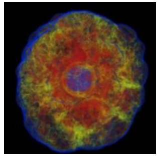
ExaBiome

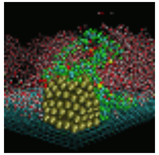








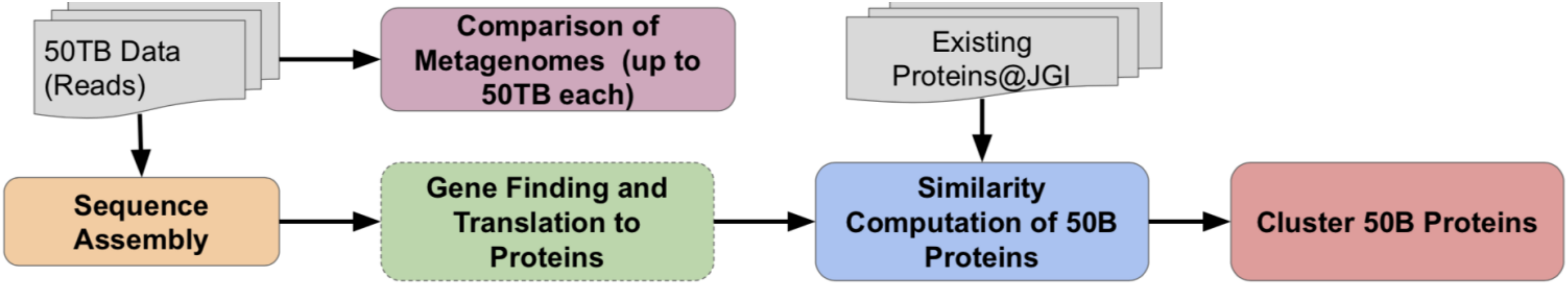




Microbiomes

- **Microbes:** these are single cell organism, e.g. viruses, bacteria
- **Microbiomes:** communities of microbial species living in our environment.
- **Metagenomics:** genome sequencing of these communities (growing exponentially)

ExaBiome software stack

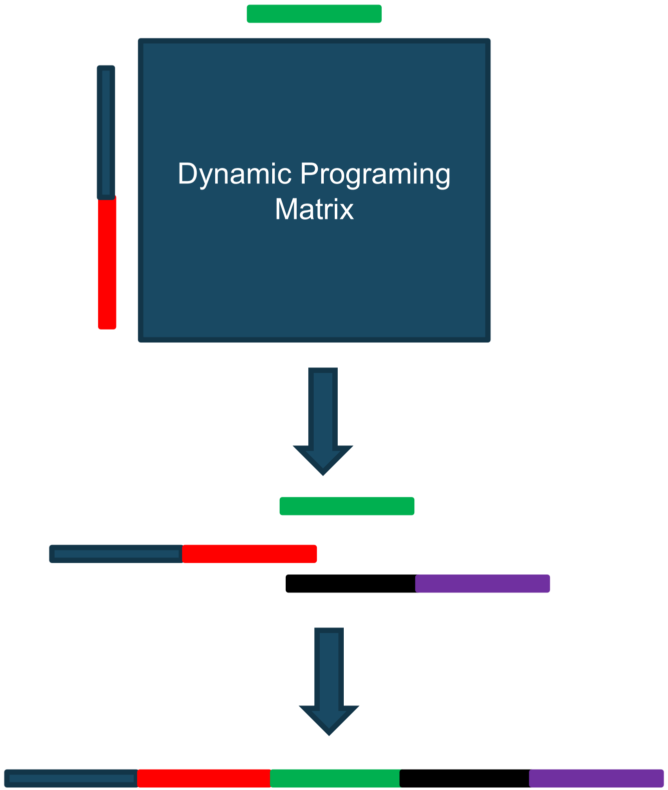

50TB Data (Reads)
Comparison of Metagenomes (up to 50TB each)
Existing Proteins@JGI
Sequence Assembly
Gene Finding and Translation to Proteins
Similarity Computation of 50B Proteins
Cluster 50B Proteins

- MetaHipMer: optimized for assembling metagenomes.
- diBELLA: Long read aligner.
- PISA: protein clustering

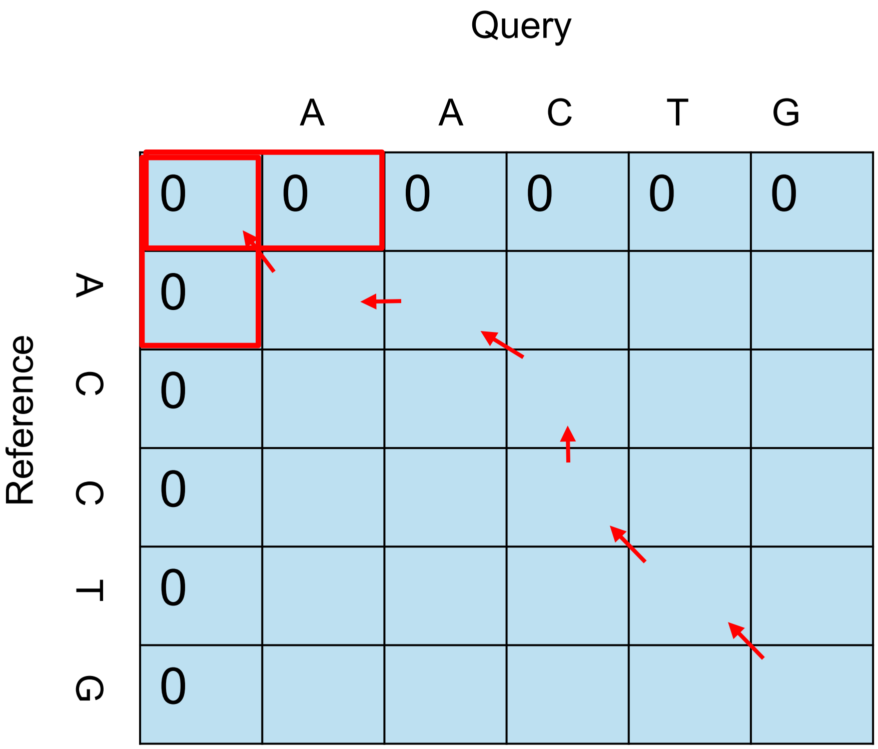
Smith-Waterman, the core of all the assembly

Majority of the ExaBiome tools make use of Smith-Waterman algorithm at their core.

This alignment information is used to stitch together different overlapping parts of the genome or determine similarity among proteins.



Smith-Waterman Algorithm



		A	A	C	T	G
	0	0	0	0	0	0
A	0					
C	0					
C	0					
T	0					
G	0					

$$S = \mathrm{Max}\,(H_{i-1,j-1} + M,\; H_{i-1,j} + M,\; H_{i,j-1} + M,\; 0)$$

$M = 5, -3$

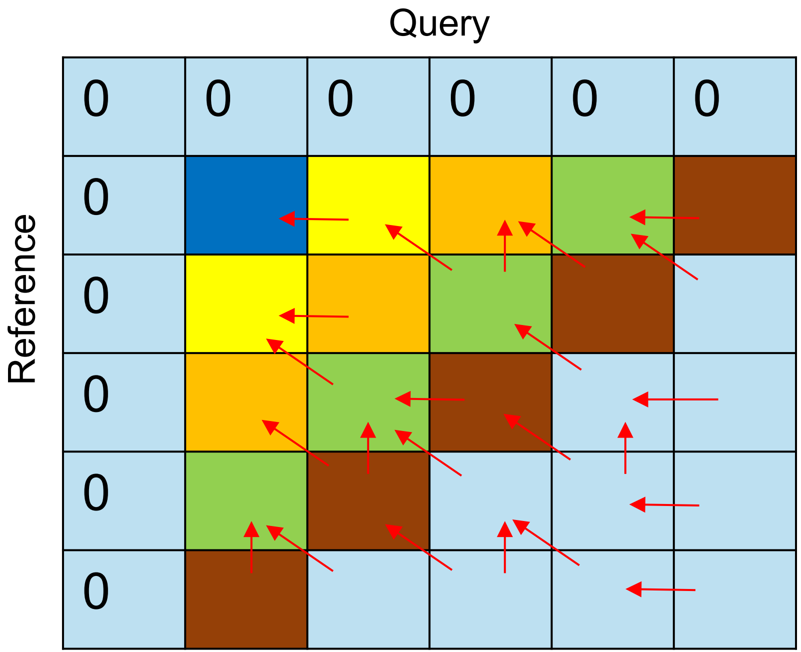
G T - C A A
G T C C - A

Smith Waterman: Challenges

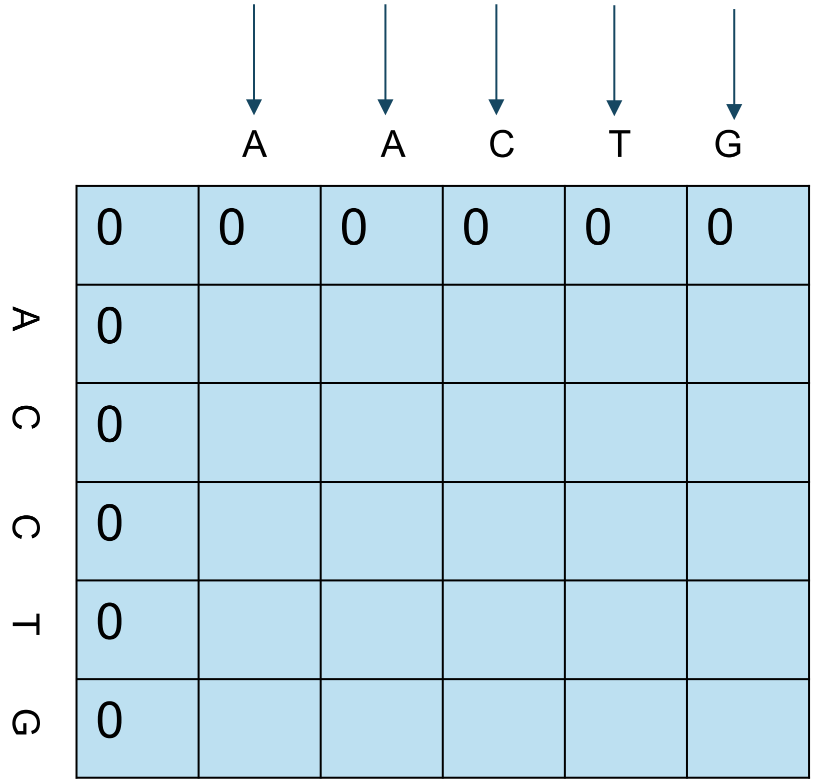
- Because of convoluted dependencies, parallelism exists only along the minor-diagonal.

- Amount of parallelism varies as the algorithm progresses.

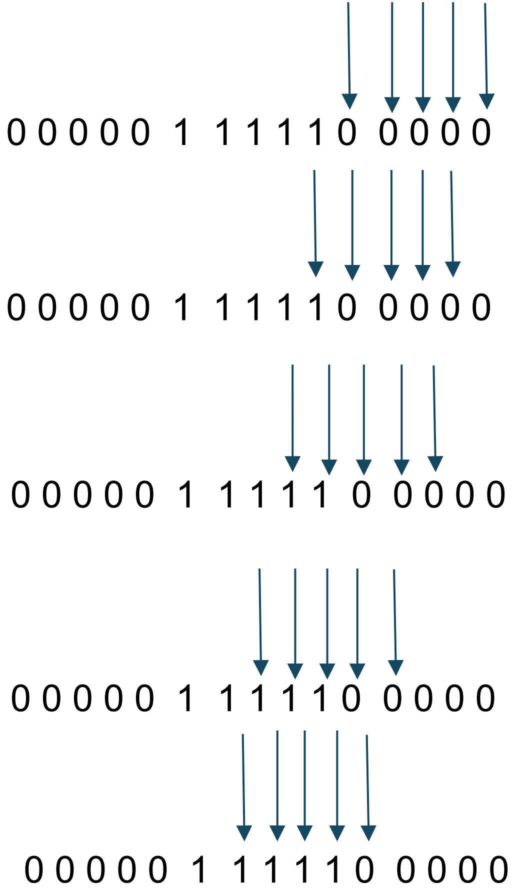
- Cell dependencies make the memory accesses a challenge on GPU.



How to handle dependencies?



		A	A	C	T	G
	0	0	0	0	0	0
A	0					
C	0					
C	0					
T	0					
G	0					



0 0 0 0 0 1 1 1 1 1 0 0 0 0 0

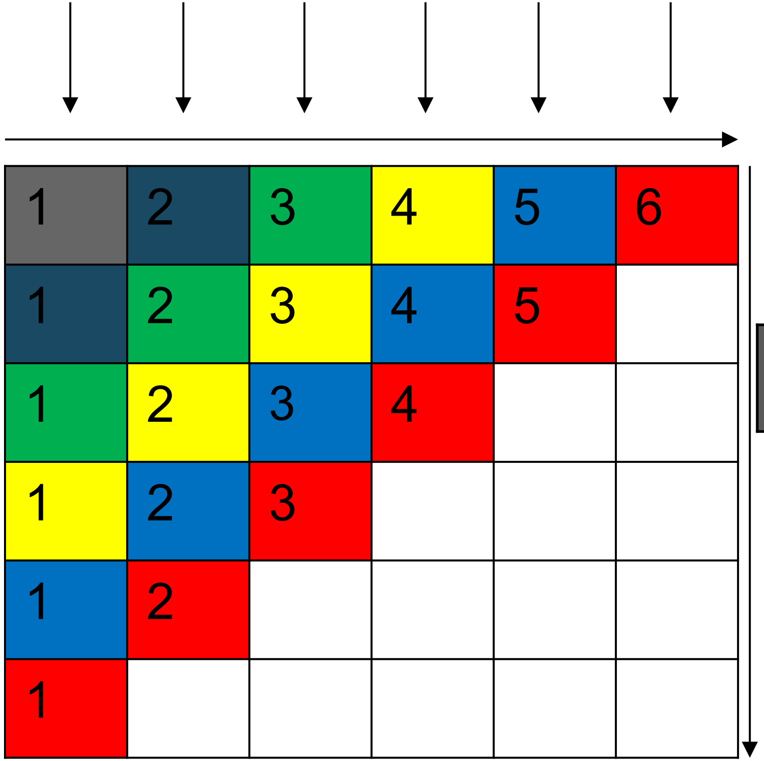
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 0 0 0 0 0

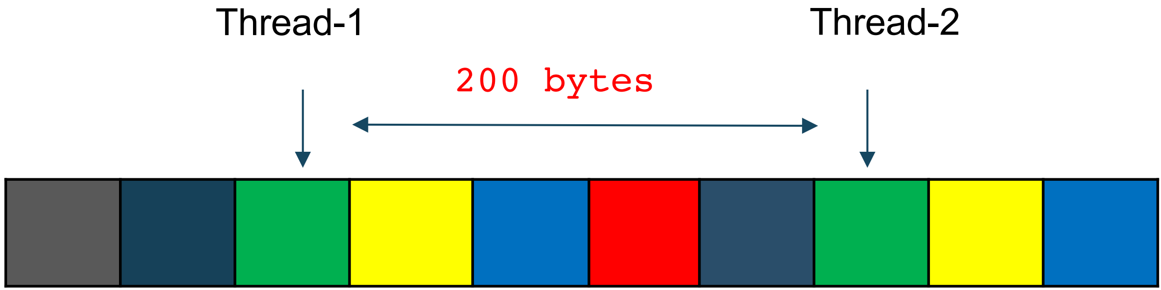
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0

The problem of non-coalesced memory accesses



1	2	3	4	5	6
1	2	3	4	5	
1	2	3	4		
1	2	3			
1	2				
1					

Row Major Indexing



- Threads access locations `length(query)*2` bytes apart while cache line is 128 bytes long.

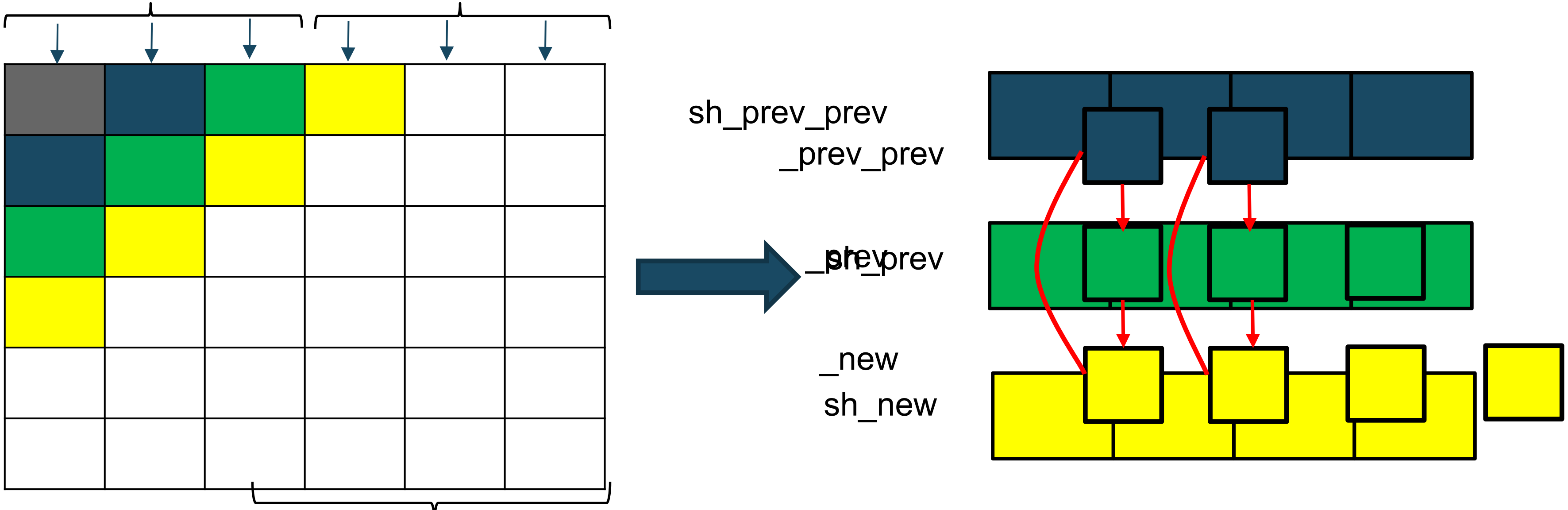
- This leads to non-coalesced memory accesses.

When only a portion of matrix is required

- Using the valid array to identify the active threads, helps correctly identifying the dependencies and enables using shuffle synch in scoring phase.

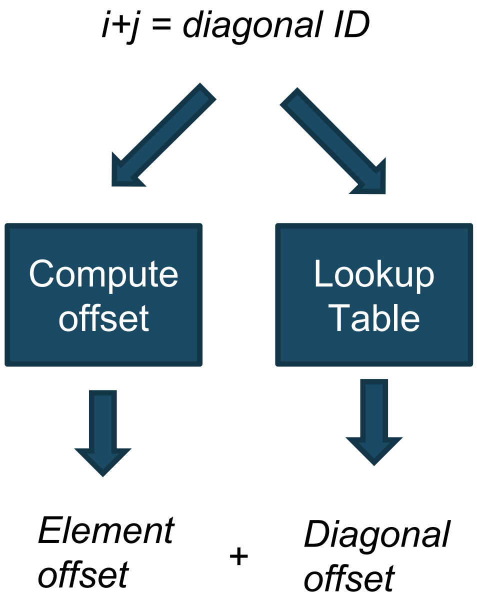
- Effectively storing the DP table-arrays in registers instead of shared memory

Inter-warp values are shared using shared memory



Phasing-out threads need to spill their registers.

When complete matrix needs to be stored

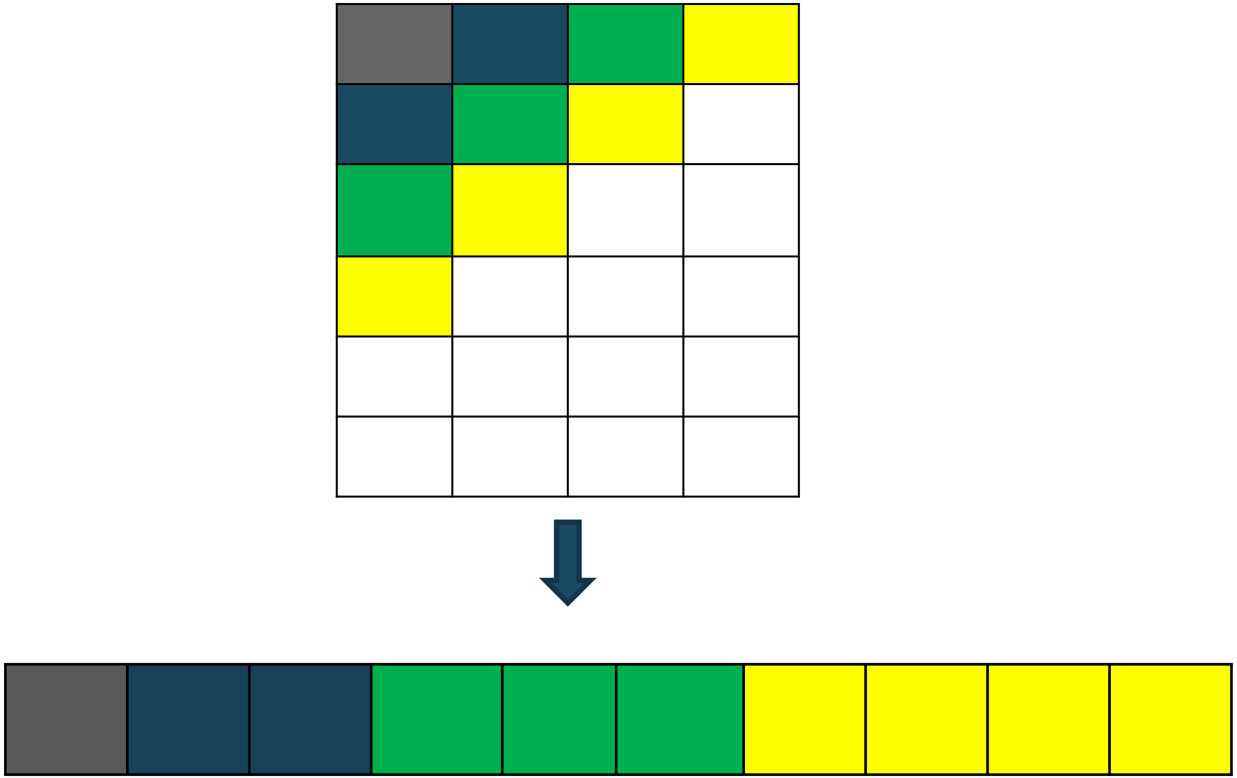


i+j = diagonal ID

Compute offset

Lookup Table

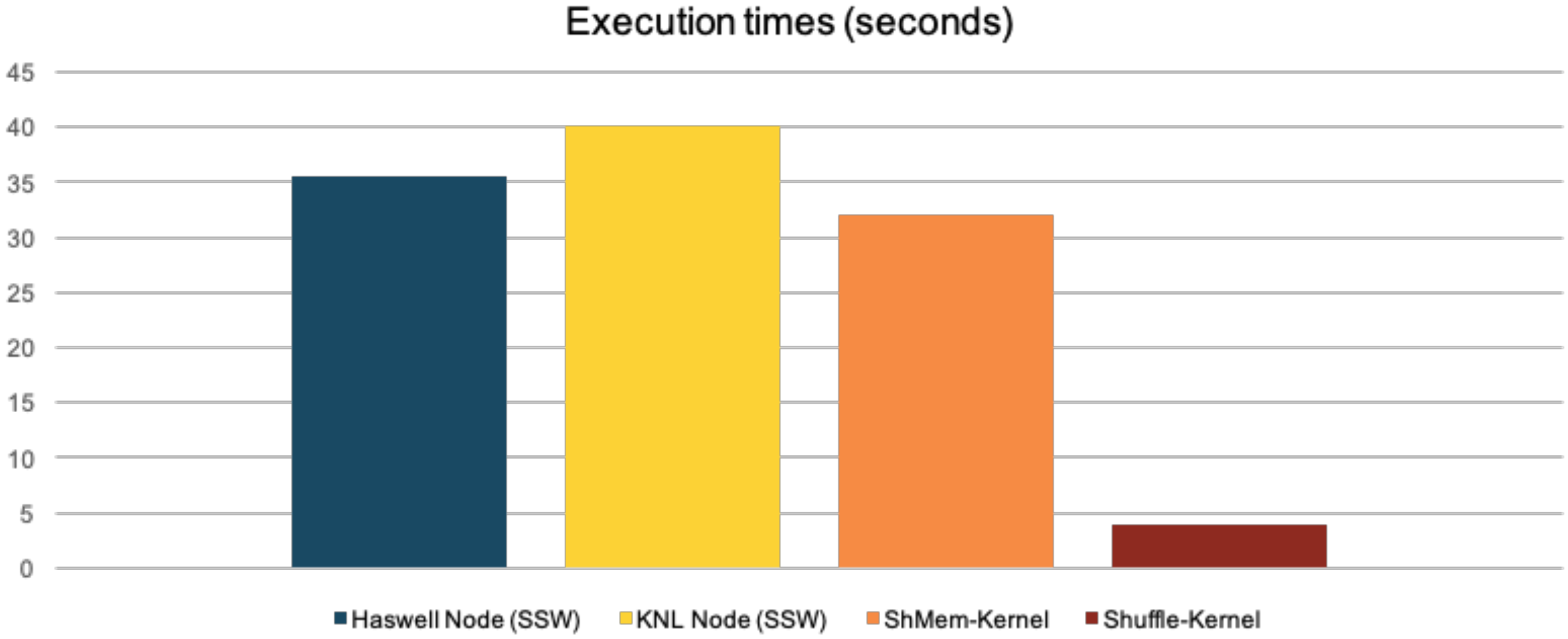
Element offset + *Diagonal offset*



Diagonal Major Indexing

Comparison with Shared Memory Approach

Total Alignments: 1 million Contig Size: 1024 Query Size: 128

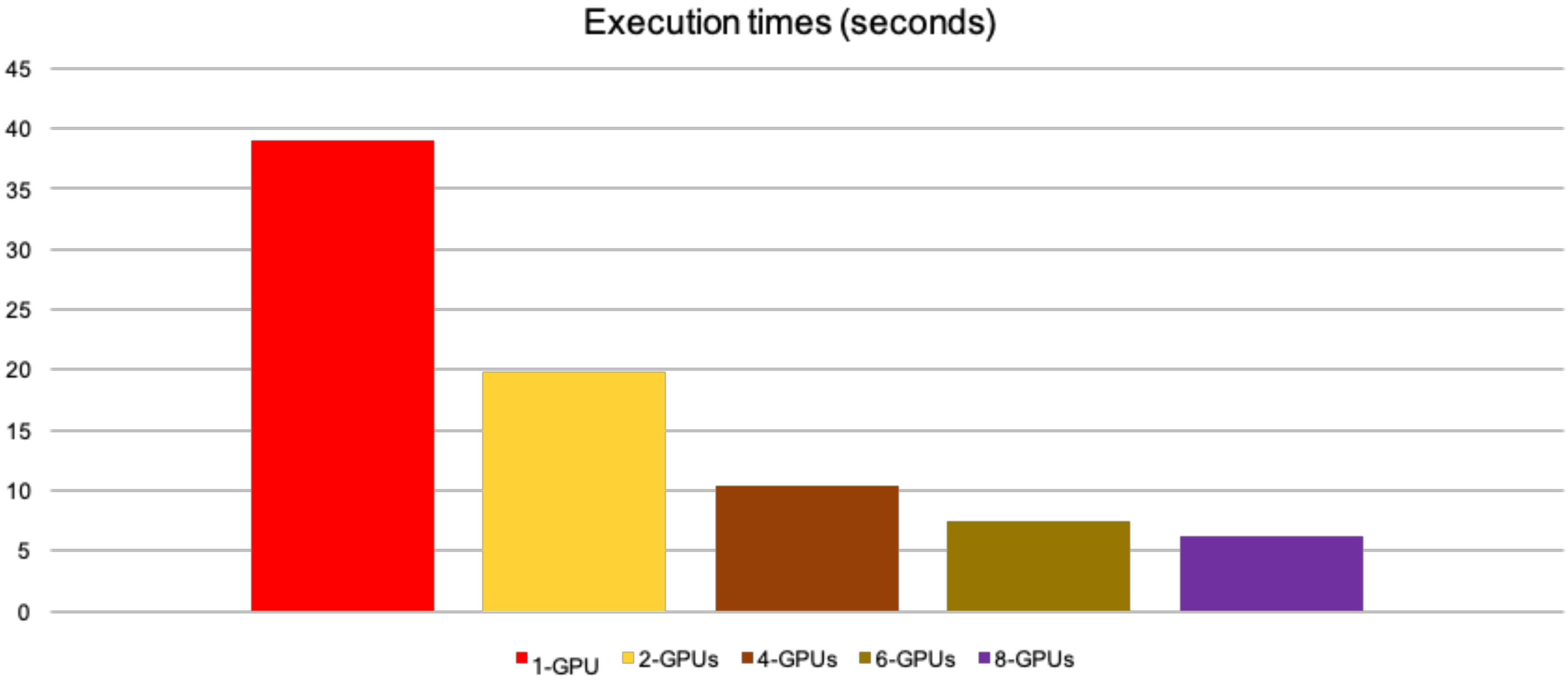


Scaling across multiple GPUs

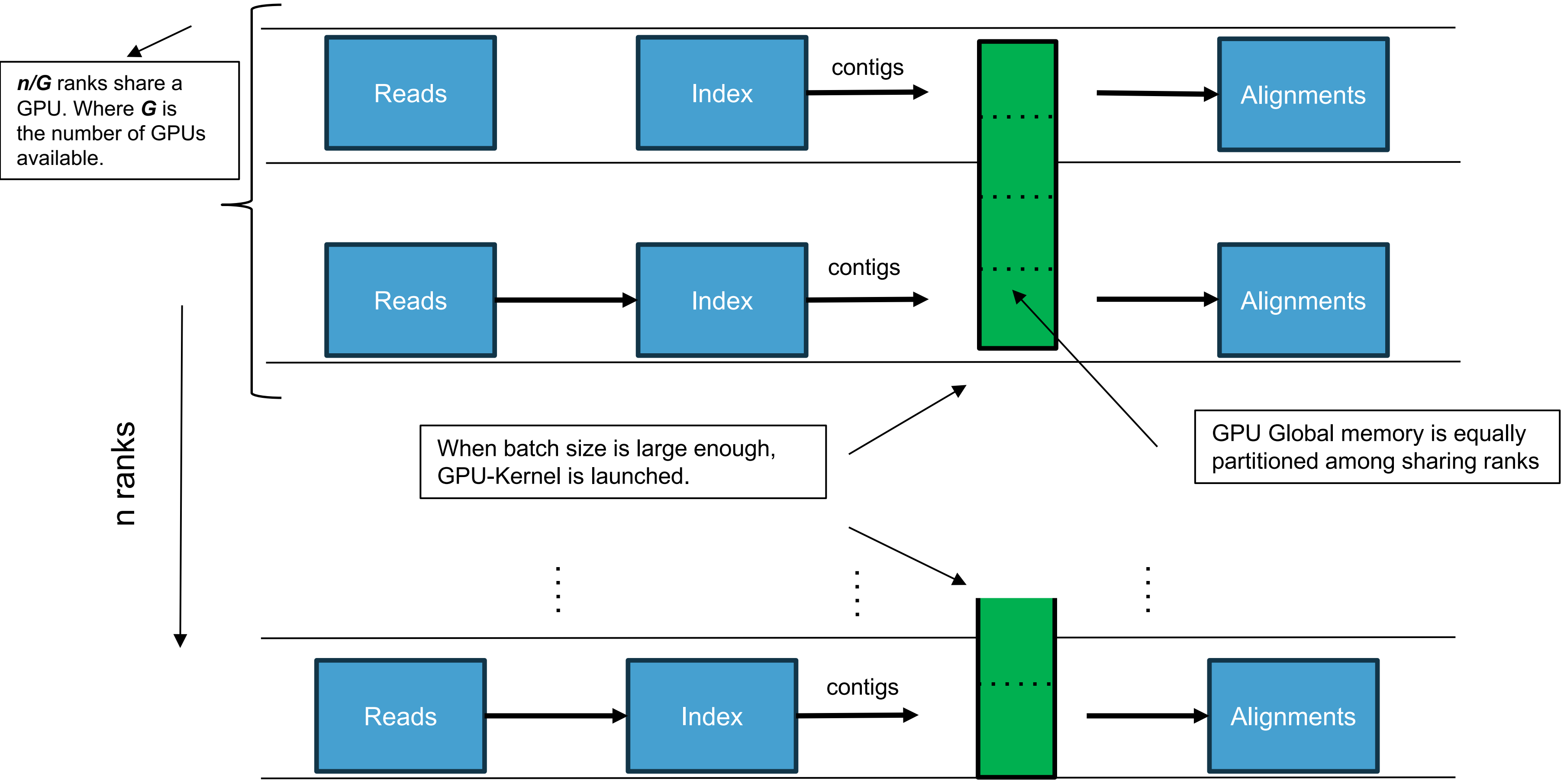
Total Alignments: 10 million

Contig Size: 1024

Query Size: 128



GPU-Klign Workflow

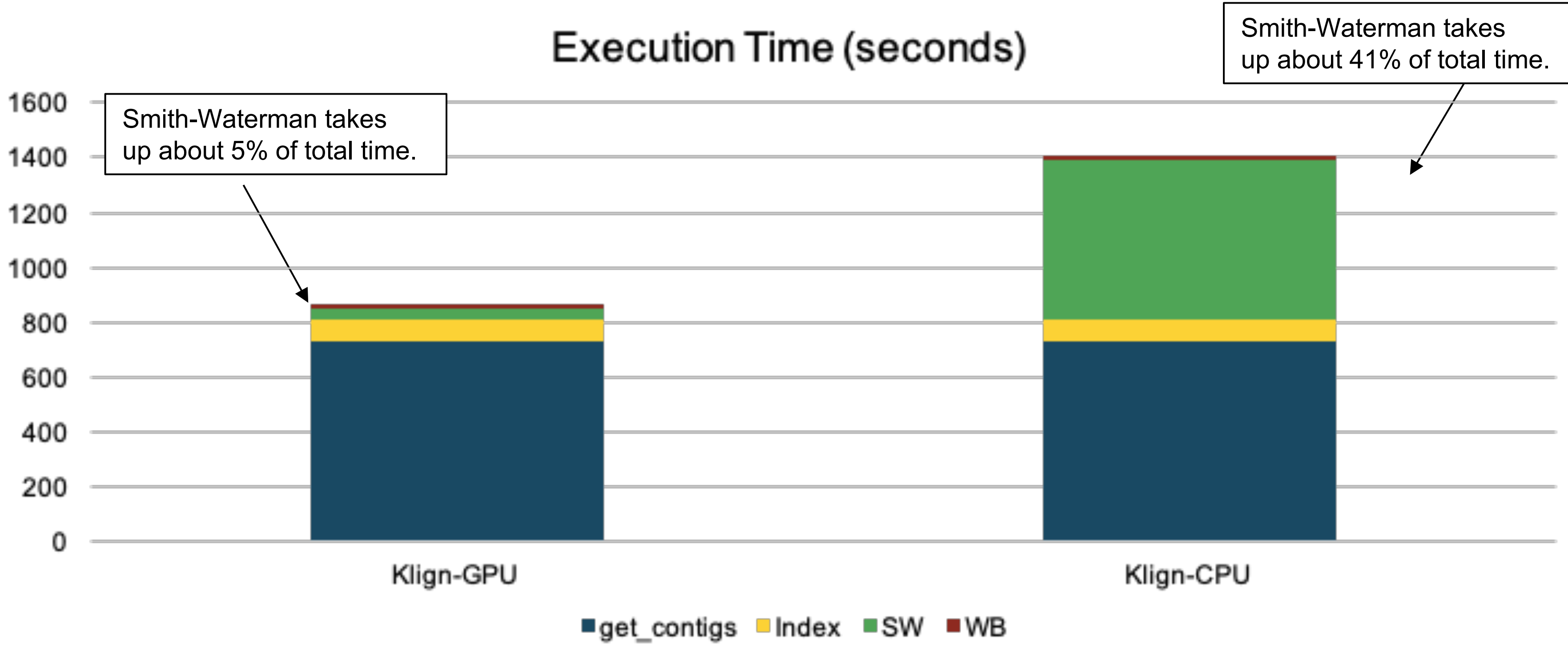


n/G ranks share a GPU. Where ***G*** is the number of GPUs available.

When batch size is large enough, GPU-Kernel is launched.

GPU Global memory is equally partitioned among sharing ranks

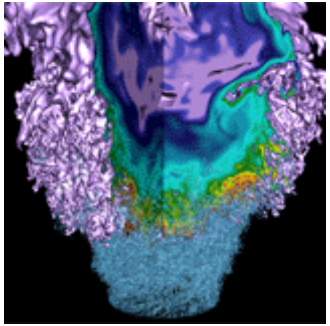
Klign vs GPU-Klign

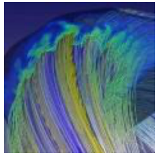


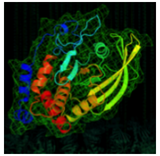
Smith-Waterman takes up about 5% of total time.

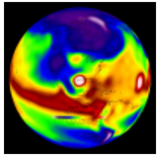
Smith-Waterman takes up about 41% of total time.

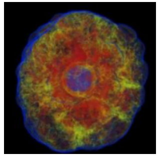
EXAALT

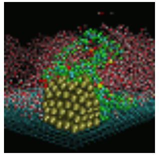












EXAALT ECP APP

- ECP EXAALT project seeks to extend the accuracy, length and time scales of material science simulations for fission/fusion reactors using LAMMPS MD
- Primary KPP target is MD of nuclear fusion material that uses the SNAP interatomic potential in LAMMPS
  - Performance directly depends on a single node performance of SNAP

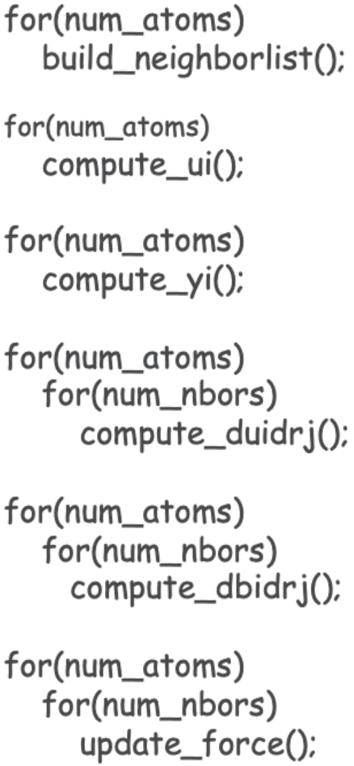
TestSNAP

- TestSNAP - an independent standalone app for SNAP module in LAMMPS
- Testbed for various parallelization and optimization strategies
- Successful optimizations are merged into LAMMPS

```
for(num_atoms) // loop over atoms
{
    build_neighborlist(); //build neighborlist for each atom
    compute_ui();
    compute_yi();
    for(num_nbors) //loop over neighbors
    {
        compute_duidrj();
        compute_dbidrj();
        update_force(); //update force for (atom,nbor) pair
    }
}
```

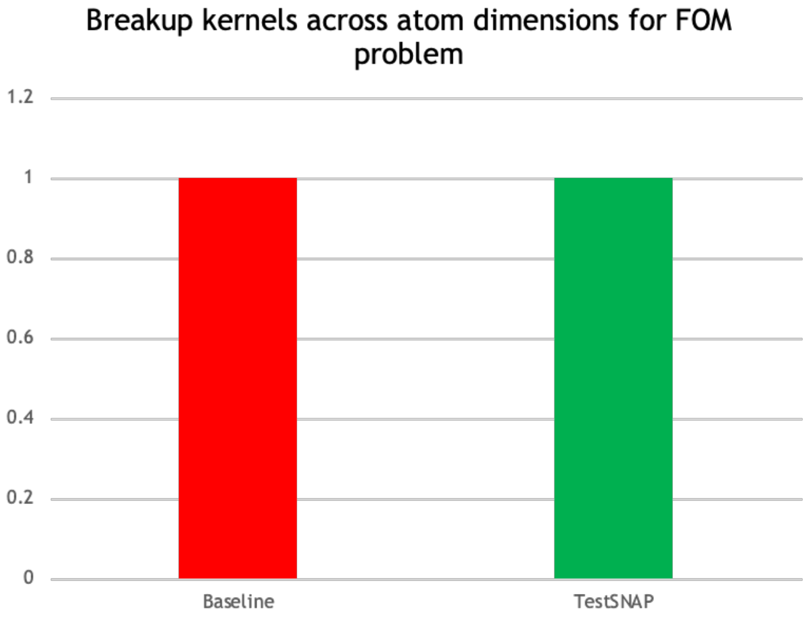
TestSNAP refactored

```
for(num_atoms)
{
    build_neighborlist();
    compute_ui();
    compute_yi();
    for(num_nbors)
    {
        compute_duidrj();
        compute_dbidrj();
        update_force();
    }
}
```



```
for(num_atoms)
   build_neighborlist();

for(num_atoms)
   compute_ui();

for(num_atoms)
   compute_yi();

for(num_atoms)
   for(num_nbors)
      compute_duidrj();

for(num_atoms)
   for(num_nbors)
     compute_dbidrj();

for(num_atoms)
   for(num_nbors)
      update_force();
```

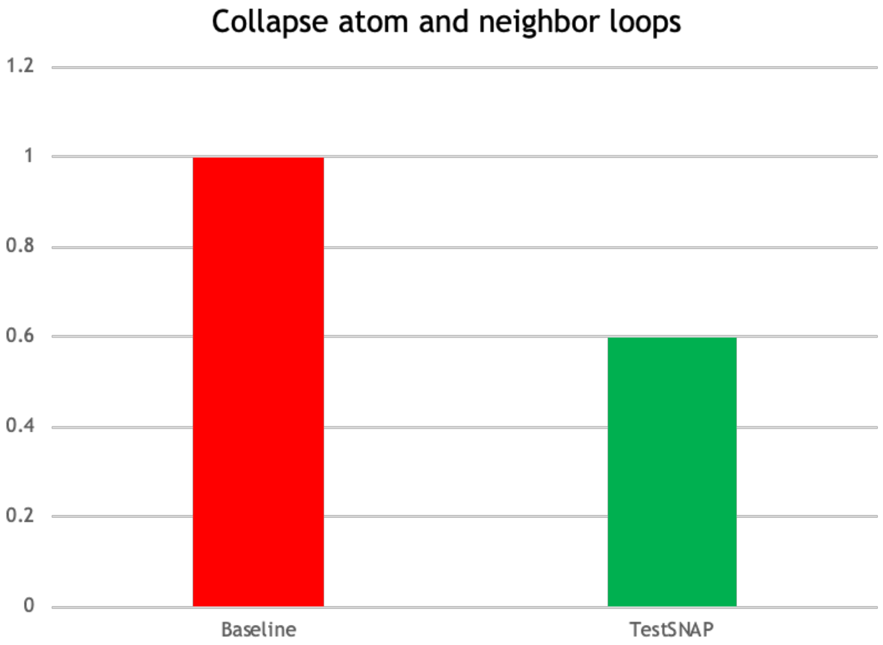
Distribute work across atom dimension



(Results for V100 GPU)

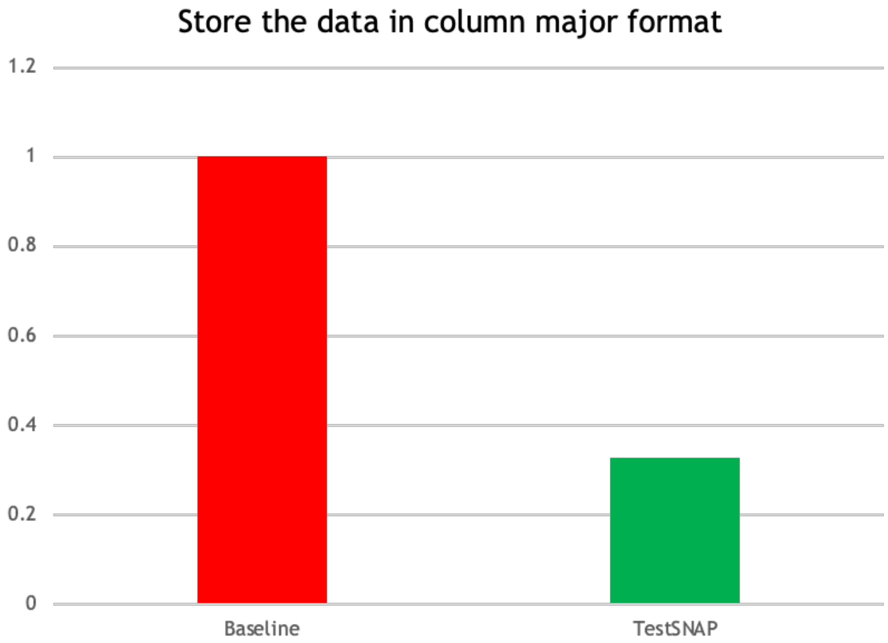
- Break up the compute kernels
- Store atom specific information across kernels
- Increases memory footprint
- Distribute the atom specific work in each kernel over the threadblocks and threads of a threadblock

Collapse atom and neighbor loops



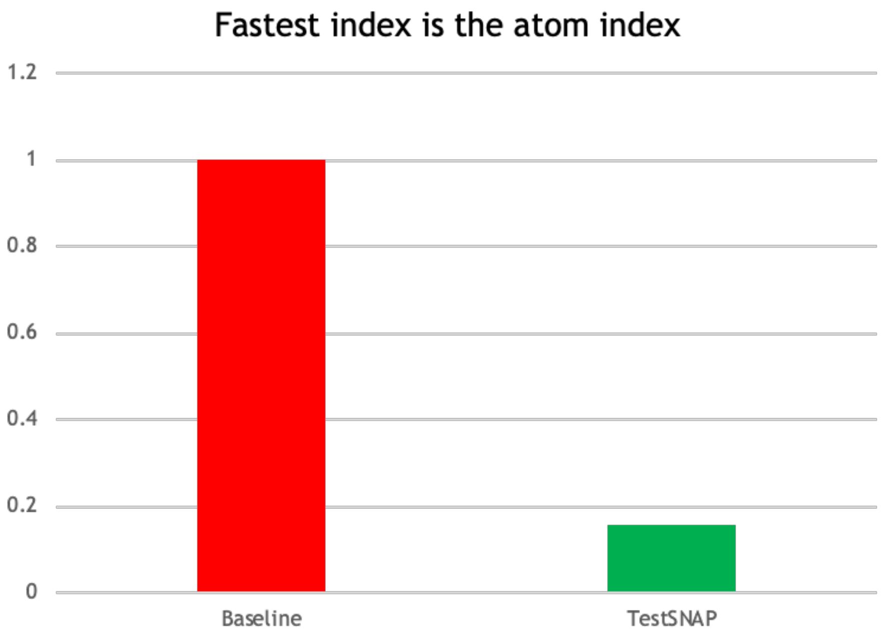
Distribute the works across atom and neighbor loops

Column major data access



Accessing the data in a column major fashion gave us a ~2X performance boost

Reverse loop order

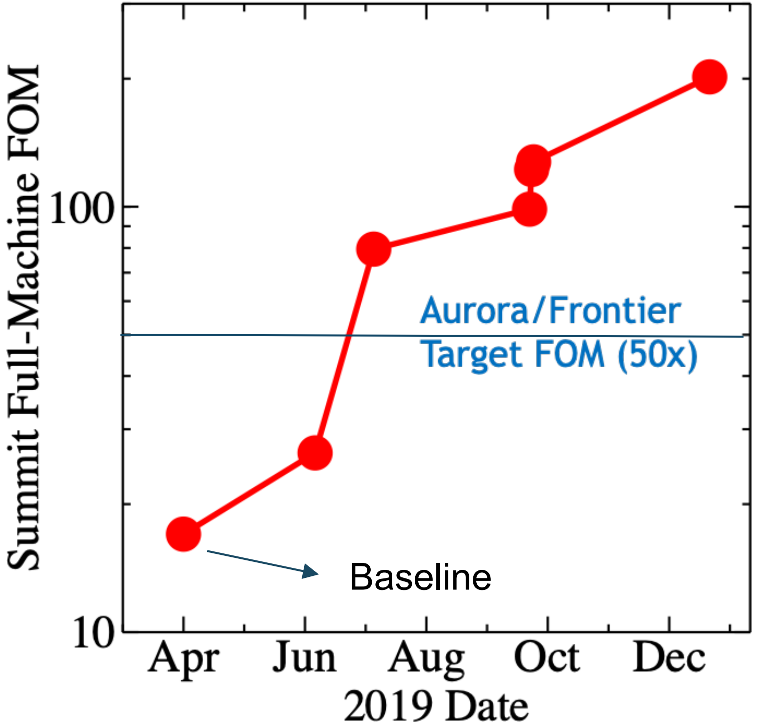


```
for neighbor j {
    for atom i {
        ...
    }
}
```

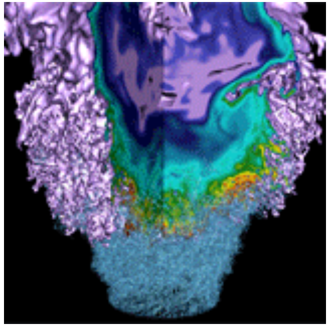
- Reverse the loops to make atom index as the fastest moving index
    - Gave a 2x performance boost

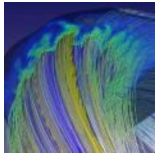
TestSNAP updates in LAMMPS/SNAP

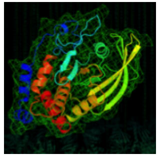
- **All the updates from TestSNAP have been successfully included in LAMMPS/SNAP.**

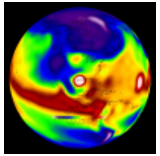


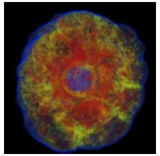
AMReX

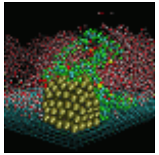










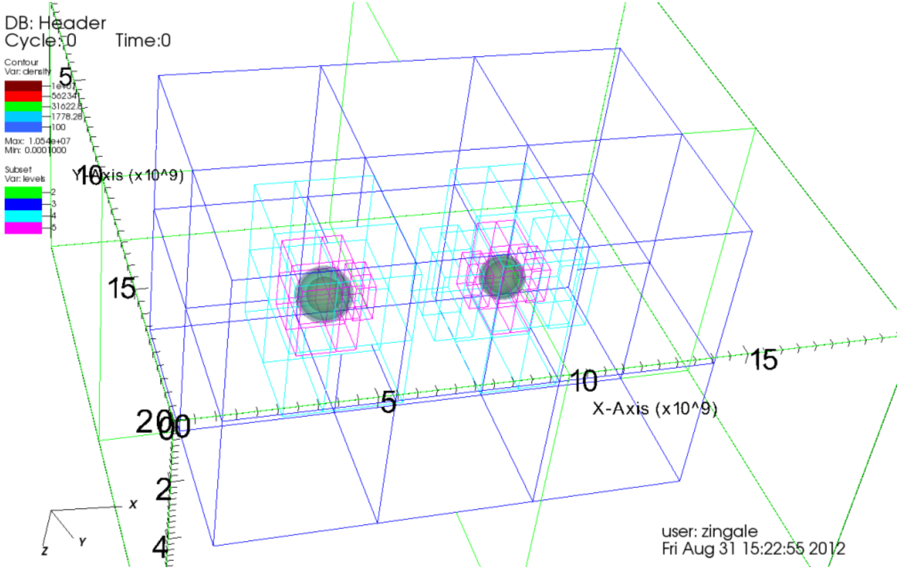


NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

AMReX: Block-Structured AMR Co-Design Center

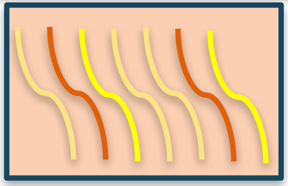


- Mesh, Particle, AMR, Linear Solvers, Cut-Cell Embedded Boundary
- Written in C++ (also an option for using Fortran interfaces)
- MPI + X
  - OpenMP on CPU
  - CUDA, HIP, DPC++ internally on GPU
  - Support for OpenACC, OpenMP on GPU
- Solution of parabolic and elliptic systems using geometric multigrid solvers
- Support for multiple load balancing strategies
- Native I/O format – supported by Visit, Paraview, yt

AMReX: Implementing on GPUs

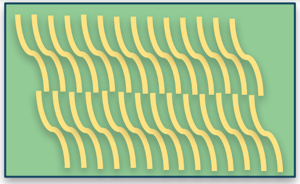
Overall Strategy: Put **floating point data** (mesh values, particle data) **on the accelerator** and leave it there. Move as little as possible throughout.

CPU: Few slower, generalized threads.



- Solution Control
- Communication
- Load Balancing
- I/O

And other serial or metadata calculations.



GPU: Many faster, specialized threads.

- Particle Calculations
- Stencil Operations
- Linear Solvers

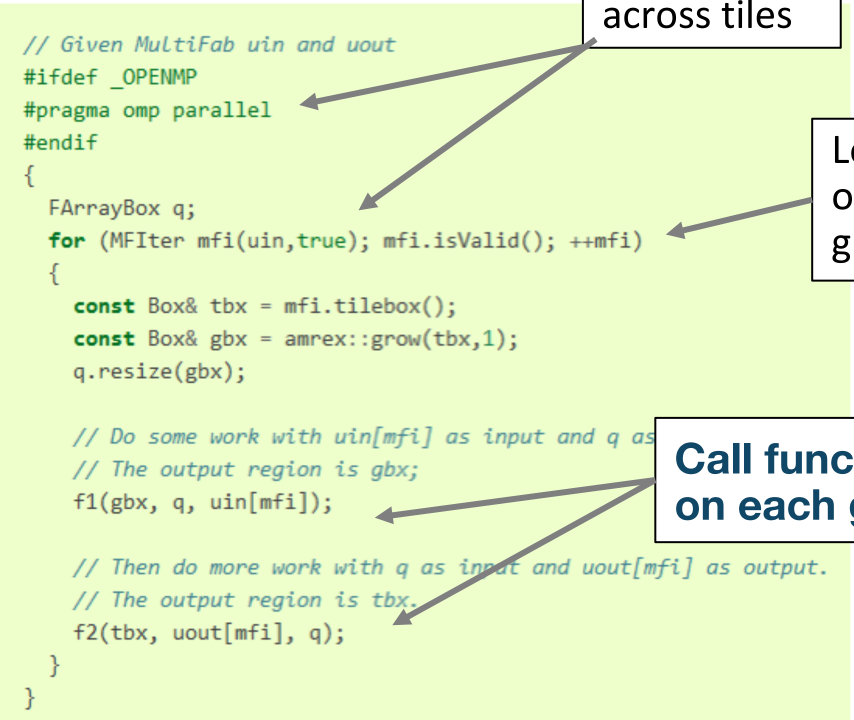
And other highly parallelizable algorithms.

- **Eliminate dependencies** (e.g. Thrust, compiler without any Fortran compiler).
- **User-proof API**. (Can’t do it wrong).

- Optimizing **communication** (currently the biggest single bottleneck).
- **Simultaneous CPU & GPU work** w/ C++ threads (e.g. I/O).

A Porting Example: Before and After

AMReX in 2018, early 2019:



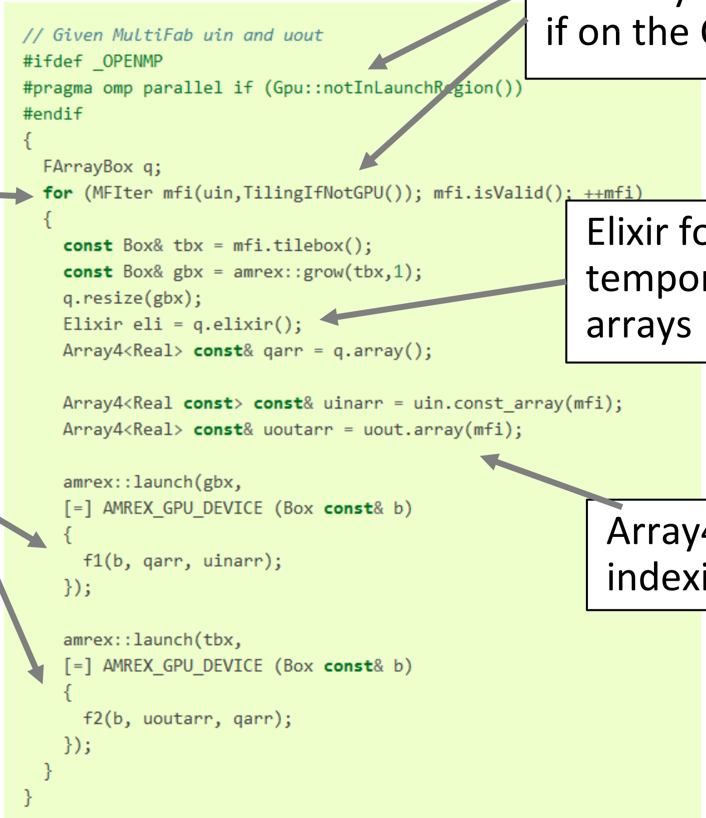
```
// Given MultiFab uin and uout
#ifdef _OPENMP
#pragma omp parallel
#endif
{
  FArrayBox q;
  for (MFIter mfi(uin,true); mfi.isValid(); ++mfi)
  {
    const Box& tbx = mfi.tilebox();
    const Box& gbx = amrex::grow(tbx,1);
    q.resize(gbx);

    // Do some work with uin[mfi] as input and q as
    // The output region is gbx;
    f1(gbx, q, uin[mfi]);

    // Then do more work with q as input and uout[mfi] as output.
    // The output region is tbx.
    f2(tbx, uout[mfi], q);
  }
}
```

OpenMP across tiles

Loop over grids

Call functions on each grid.

AMReX in 2020:



```
// Given MultiFab uin and uout
#ifdef _OPENMP
#pragma omp parallel if (Gpu::notInLaunchRegion())
#endif
{
  FArrayBox q;
  for (MFIter mfi(uin,TilingIfNotGPU()); mfi.isValid(); ++mfi)
  {
    const Box& tbx = mfi.tilebox();
    const Box& gbx = amrex::grow(tbx,1);
    q.resize(gbx);
    Elixir eli = q.elixir();
    Array4<Real> const& qarr = q.array();

    Array4<Real const> const& uinarr = uin.const_array(mfi);
    Array4<Real> const& uoutarr = uout.array(mfi);

    amrex::launch(gbx,
    [=] AMREX_GPU_DEVICE (Box const& b)
    {
      f1(b, qarr, uinarr);
    });

    amrex::launch(tbx,
    [=] AMREX_GPU_DEVICE (Box const& b)
    {
      f2(b, uoutarr, qarr);
    });
  }
}
```

Tile only if on the CPU

Elixir for temporary arrays

Array4s for indexing.

Performance Example: setBndry

```
for (MFIter fai(*this); fai.isValid(); ++fai)
{
   const Box& gbx = fai.fabbox();
   const Box& vbx = fai.validbox();
   BoxList blst = amrex::boxDiff(gbx,vbx);
   const int nboxes = blst.size();
   if (nboxes > 0)
   {
      AsyncArray<Box> async_boxes(blst.data().data(), nboxes);
      Box const* pboxes = async_boxes.data();

      long ncells = 0;
      for (const auto& b : blst) {
         ncells += b.numPts();
      }

      auto fab = this->array(fai);
      AMREX_FOR_1D ( ncells, icell,
      {
         const Dim3 cell = amrex::getCell(pboxes, nboxes, icell).dim3();
         for (int n = strt_comp; n < strt_comp+ncomp; ++n) {
            fab(cell.x,cell.y,cell.z,n) = val;
         }
      });
   }
}
```

Old GPU Version:

1) CPU: calculate a list of boundary boxes,
2) GPU: launch and set the value on only those boxes.

```
for (MFIter fai(*this); fai.isValid(); ++fai)
{
   const Box& gbx = fai.fabbox();
   const Box& vbx = fai.validbox();
   auto fab = this->array(fai);

   AMREX_PARALLEL_FOR_4D(gbx, ncomp, i, j, k, n,
   {
      if (!(vbx.contains({i, j, k})))
      {
         fab(i,j,k,n) = val;
      }
   });
}
```

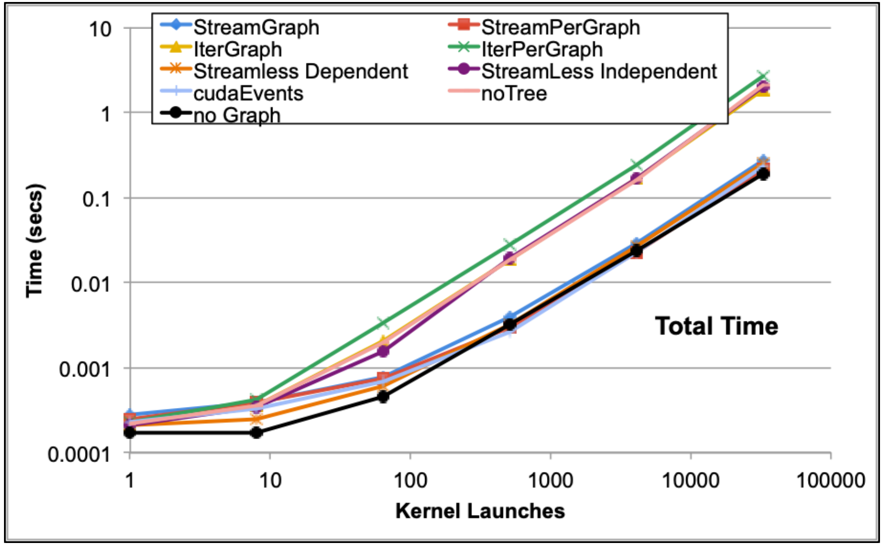
- 50% - 150% faster on the GPU.
- Considerably slower on the CPU.
- Merging kernel does NOT improve performance of either.

New GPU Version:

1) Immediately launch over entire FAB’s box.
2) If thread’s cell is outside the valid box (so, it’s a ghost cell) set the value

AMReX is a platform for testing advanced features on production-scale simulations.

Testing CUDA Graphs for Halo Distribution algorithm:

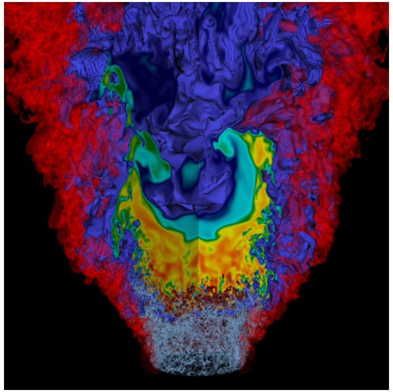


- Comparison of **CUDA graph build methods** vs. using a fused kernel launch methodology.
- Recording with **dependencies** and **well defined simultaneous work** gives better performance in all aspects.
- AMReX fused kernels are currently better, but only barely. Keeping an eye on further developments to ensure optimal communication performance.

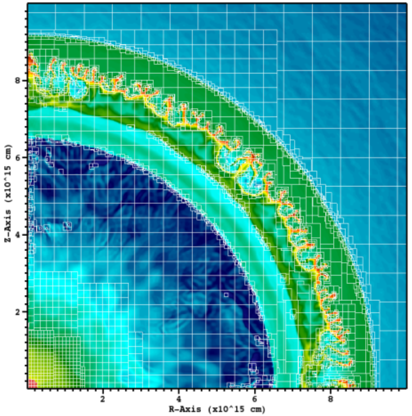
❖ AMReX is also a platform to test (CUDA vs. HIP vs. DPC++) & C++ portability.

❖ Additional advanced NVIDIA libraries we want to test: NVSHMEM, Optix.

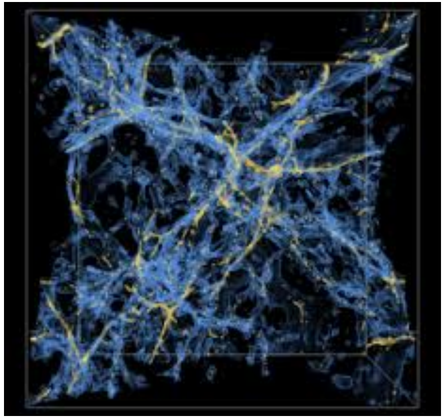
AMReX used by six ECP applications



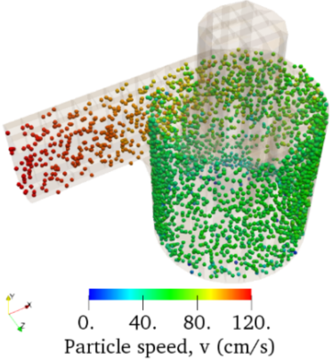
Combustion (Pele)



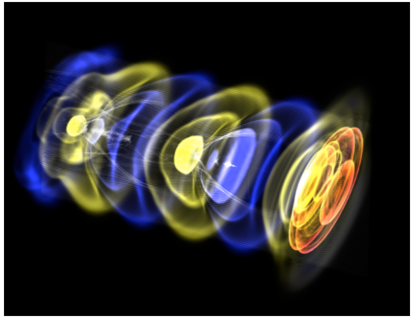
Astrophysics (Castro)



Cosmology (Nyx)



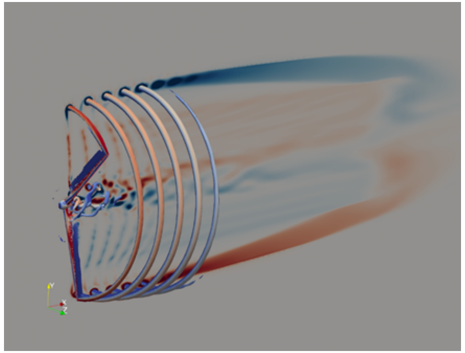
Multiphase flow (MFIX-Exa)



Accelerators (WarpX)

Non-ECP applications

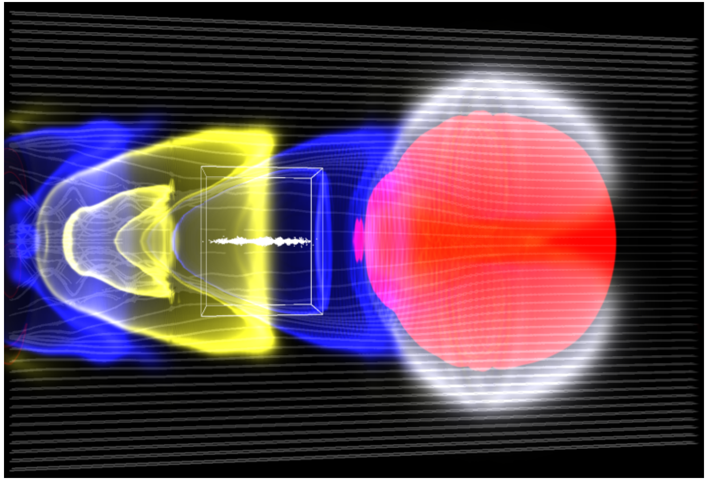
- Phase field models
- Microfluids
- Ionic liquids
- Non-Newtonian flow
- Fluid-structure interaction
- Shock physics
- Cellular automata
- Low Mach number astrophysics
- Defense science

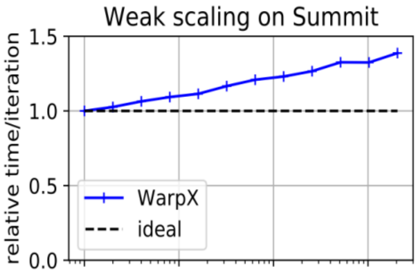


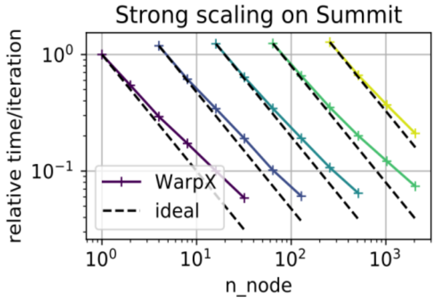
Exawind

WarpX

- **Original GPU strategy was using OpenACC in Fortran functions.**
- **Converted to AMReX's C++ lambda based approach.**
  - Thrust vectors as particle containers used too much memory
  - AMReX's PODVector class mitigates memory usage issue allowing for runs with more particles. The latest
- **AMReX has added more features for random numbers and bin data structure to support binary collision of particles.**
- **KPP measurement on 2048 Summit nodes was over 47x compared to baseline.**



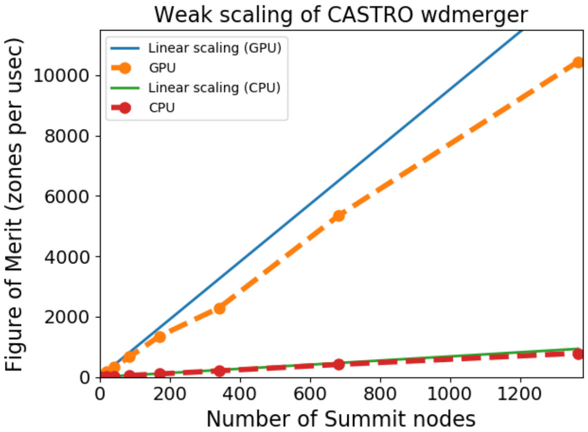

Weak scaling on Summit

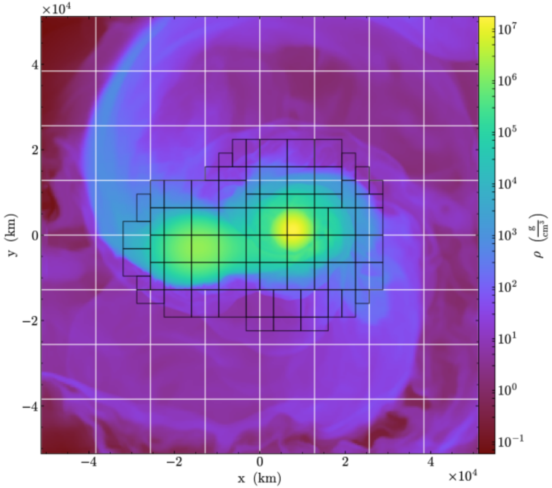

Strong scaling on Summit

Castro: Open-Source Astrophysical Radiation Hydrodynamics

- **Castro functionality on GPUs:**
  - Hydrodynamics (2nd order unsplit CTU)
  - Strang-split or Simple SDC reactions (VODE)
  - Explicit thermal diffusion
  - Poisson self-gravity with geometric multigrid
  - Stellar equations of state
- **Ongoing/Future GPU ports:**
  - Flux-limited diffusion radiation
  - 4th-order SDC for hydro + reactions

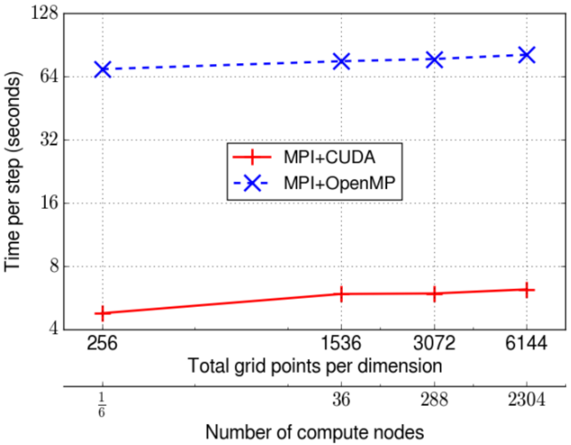
- **Castro GPU strategy:**
  - CUDA Fortran kernels loop through cells in boxes
  - Python preprocessor script inserts GPU kernels
  - Future migration to AMReX C++ lambda launches
- **ECP-funded developments: (Exastar Collaboration)**
  - Coupled to Thornado (ORNL) for two-moment neutrino radiation transport for core-collapse supernovae
  - Thornado accelerated with OpenACC & OpenMP





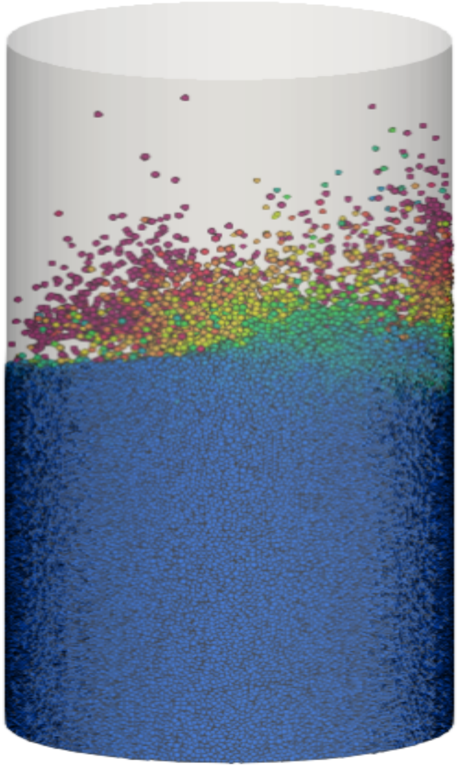
Nyx

- **GPU capabilities**
  - Dark matter particles (AMReX NeighborParticleContainer)
  - Hydrodynamics (AMReX GPU memory management (prefetching/organizing) and kernel launch)
  - Heating-cooling reactions (AMReX Arena alloc and free, linking against Sundials for time integration)
- **GPU challenges**
  - Optimizing memory access/use when overflowing high-bandwidth GPU memory
  - Investigating appropriate cost functions for load-balancing simulations where particles cluster (single grid vs dual-grid)
  - Extending different coupling strategies between advective and reactive terms to the GPU
- **Physics modules in active development**
  - AGN feedback
  - Accounting for halos effect on reionization on-the-fly
  - Non-relativistic neutrinos

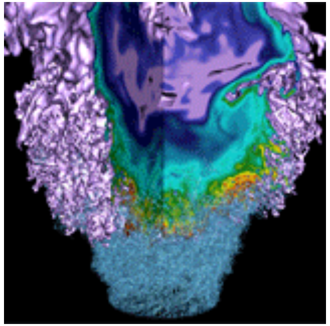


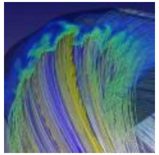
MFIX-Exa

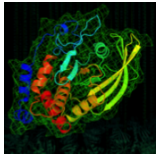
- **GPU computation for both the fluid and solid (particles) phases**
  - Solvers for the fluid-phase update scheme are AMReX solvers
  - Tests with 6 MPI tasks and 6 GPUs (1 GPU per MPI task) on a single Summit node
  - Maximum speedup of about 53.9x for a prototypical CLR with respect to a simulation with 36 MPI tasks.
- **Current focus**
  - Embedded boundary treatment of particles
  - Multiscale models for improved efficiency in dense particle regions
  - New projection based algorithm

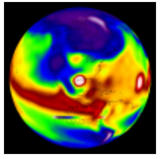


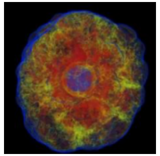
BerkeleyGW

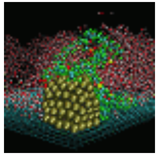










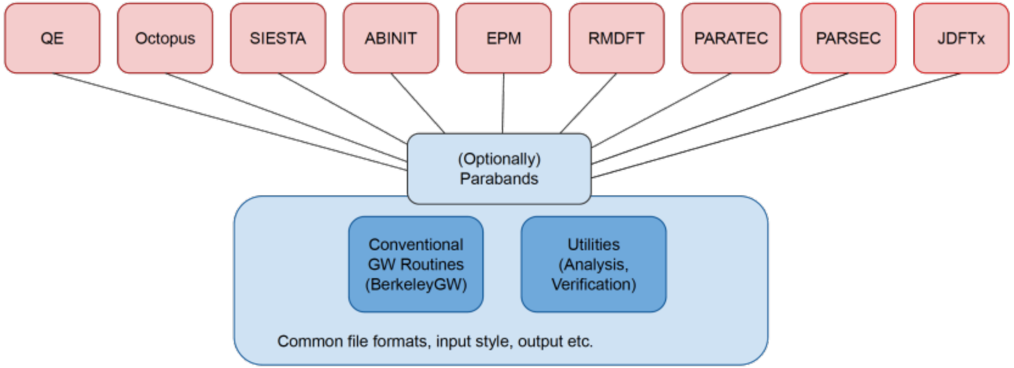


NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

BerkeleyGW



QE | Octopus | SIESTA | ABINIT | EPM | RMDFT | PARATEC | PARSEC | JDFTx

(Optionally) Parabands

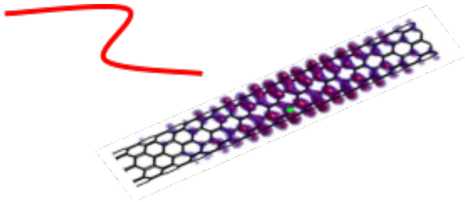
Conventional GW Routines (BerkeleyGW)

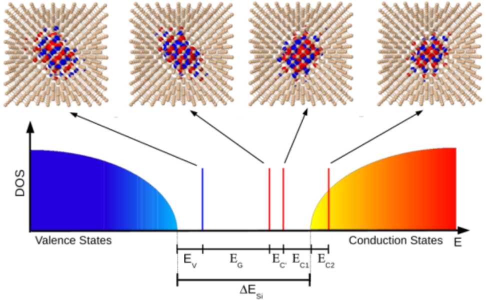
Utilities (Analysis, Verification)

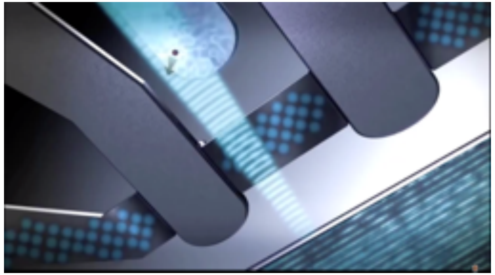
Common file formats, input style, output etc.

Many-body effects in Excited-State properties of complex materials

- Photovoltaics
- LEDs
- Quantum Computers
- Junctions / Interfaces
- Defect Energy Levels



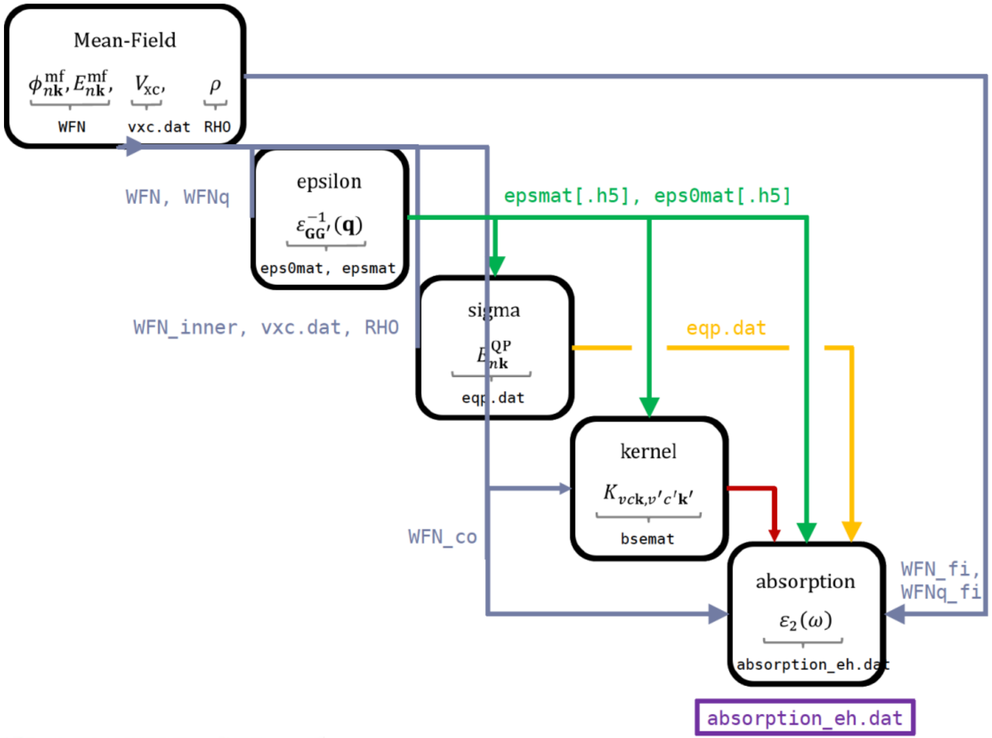




BerkeleyGW

- Material Science: http://www.berkeleygw.org
- ~100,000 lines of code, mainly Fortran 90
- MPI, OpenMP(on CPU), CUDA/OpenACC(on GPU)
- Computational motifs:
  - Large distributed matrix multiplication (tall and skinny matrices)
  - Large distributed eigenvalue problems
  - Fast Fourier Transformations (FFT)
  - Dimensionality reduction and low-rank approximations
- Libraries required:
  - BLAS, LAPACK, ScaLAPACK, FFTW, ELPA, PRIMME, HDF5
  - cuBLAS, cuFFT

BerkeleyGW Workflow



Mean-Field
$\underbrace{\phi^{\mathrm{mf}}_{n\mathbf{k}}, E^{\mathrm{mf}}_{n\mathbf{k}}}_{\texttt{WFN}}, \underbrace{V_{\mathrm{xc}}}_{\texttt{vxc.dat}}, \underbrace{\rho}_{\texttt{RHO}}$
WFN, WFNq
epsilon
$\underbrace{\varepsilon^{-1}_{\mathbf{GG}'}(\mathbf{q})}_{\texttt{eps0mat, epsmat}}$
epsmat[.h5], eps0mat[.h5]
WFN_inner, vxc.dat, RHO
sigma
$\underbrace{E^{\mathrm{QP}}_{n\mathbf{k}}}_{\texttt{eqp.dat}}$
eqp.dat
kernel
$\underbrace{K_{vc\mathbf{k},v'c'\mathbf{k}'}}_{\texttt{bsemat}}$
WFN_co
absorption
$\underbrace{\varepsilon_2(\omega)}_{\texttt{absorption\_eh.dat}}$
WFN_fi,
WFNq_fi
absorption_eh.dat

DFT Kohn-Sham (SCF and NSCF)
$\{\varphi^{\mathrm{DFT}}_{\mathrm{nk}}(\mathbf{r}), E^{\mathrm{DFT}}_{\mathrm{nk}}\}$

Compute Dielectric Function
$\{\ \varepsilon_{\mathrm{qGG'}}\ \}$ **epsilon.flavor.x** ✓

GW: Quasiparticle Properties
$\{\varphi^{\mathrm{QP}}_{\mathrm{nk}}(\mathbf{r}), E^{\mathrm{QP}}_{\mathrm{nk}}\}$ **sigma.flavor.x** ✓

BSE: Construct Kernel (coarse grid)
K(k,c,v,k',c',v') **kernel.flavor.x**

Interpolate Kernel to Fine Grid / Diagonalize BSE Hamiltonian
$\{A^{s}_{\mathrm{cvk}}, E^{s}_{\mathrm{cvk}}\}$ **absorption.flavor.x**

Porting and Optimization Strategies

Implementations

- CUDA (cuBLAS, cuFFT, self-written kernels), Fortran interface
- OpenACC directives, cuBLAS and cuFFT Fortran interface from PGI
- Better control of kernel execution with CUDA

v.s. Easier to program/portability with OpenACC

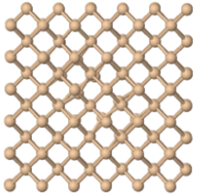
Strategies/Techniques

- Use streams for asynchronous data transfers and to increase concurrency
- Use a hybrid scheme for large reductions (100s-1000s of billions)
  - shared memory on GPU and OpenMP on CPU
- Overlap MPI communication with GPU computation
- Use batched operation for more flexible parallelism and to save memory

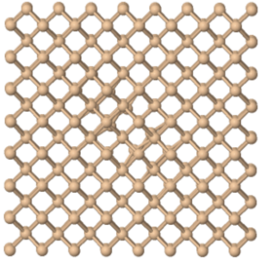
Benchmark Systems

Three benchmarks:

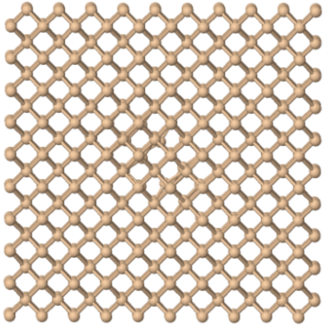
- Si214, Si510, Si998
- To study a divacancy defect in Silicon, a prototype of a solid state qbit



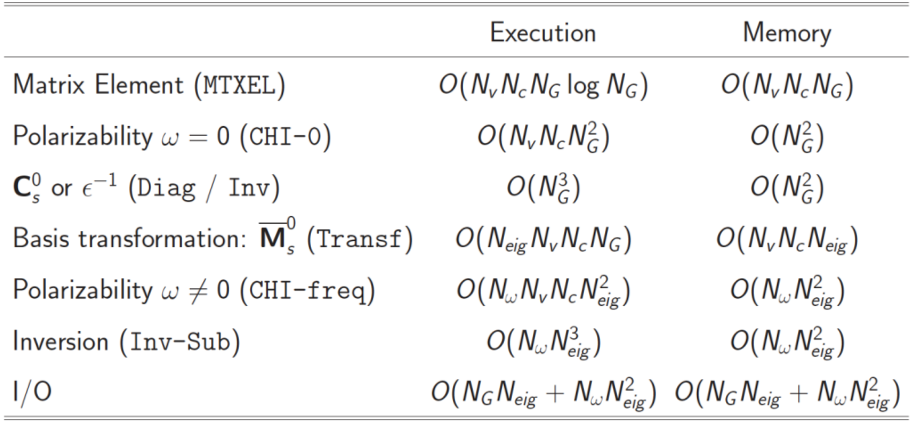
Si214



Si510



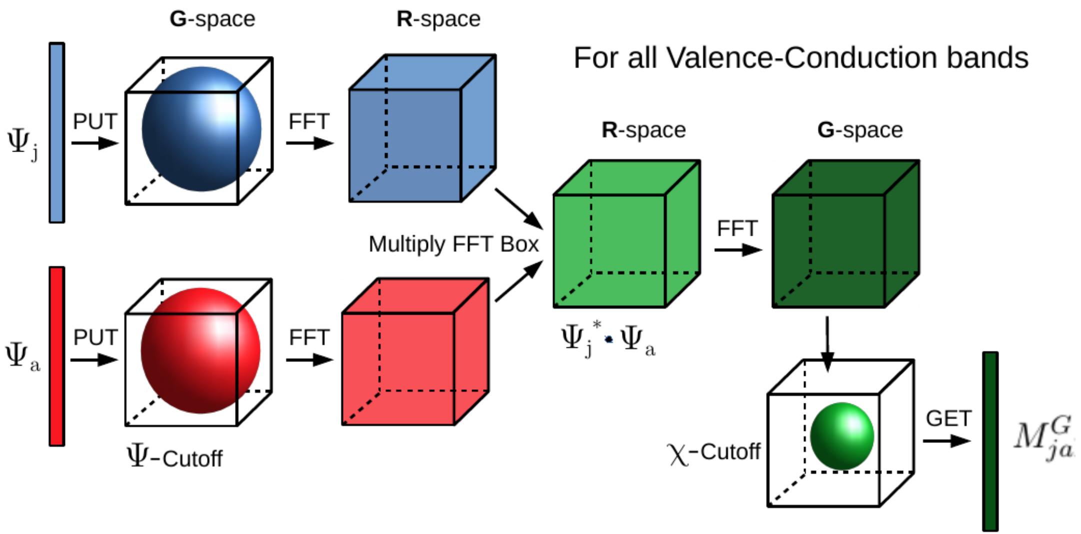
Si998



	Execution	Memory
Matrix Element (MTXEL)	$O(N_v N_c N_G \log N_G)$	$O(N_v N_c N_G)$
Polarizability $\omega = 0$ (CHI-0)	$O(N_v N_c N_G^2)$	$O(N_G^2)$
$\mathbf{C}_s^0$ or ϵ^{-1} (Diag / Inv)	$O(N_G^3)$	$O(N_G^2)$
Basis transformation: $\overline{\mathbf{M}}_s^0$ (Transf)	$O(N_{eig} N_v N_c N_G)$	$O(N_v N_c N_{eig})$
Polarizability $\omega \neq 0$ (CHI-freq)	$O(N_\omega N_v N_c N_{eig}^2)$	$O(N_\omega N_{eig}^2)$
Inversion (Inv-Sub)	$O(N_\omega N_{eig}^3)$	$O(N_\omega N_{eig}^2)$
I/O	$O(N_G N_{eig} + N_\omega N_{eig}^2)$	$O(N_G N_{eig} + N_\omega N_{eig}^2)$

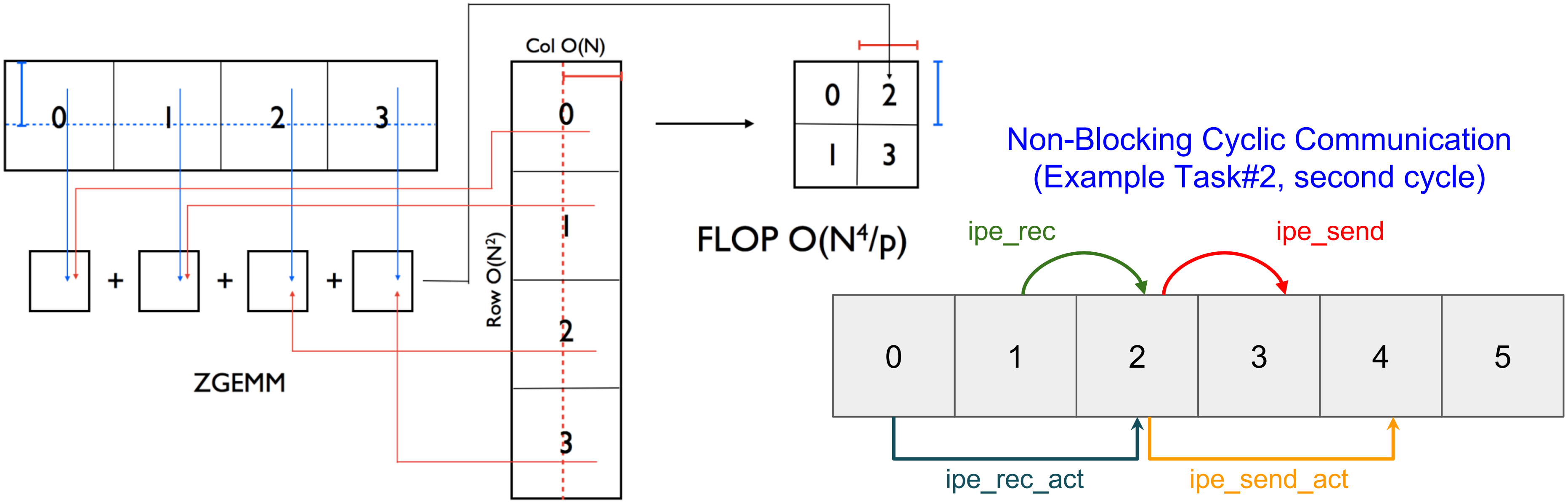
Computational Cost

Epsilon Module (MTXEL Kernel)



- cuFFT, pinned memory, CUDA streams
- Asynchronous memory transfers, high concurrency
- Batched to avoid OOM
- CUDA kernels for element-multiply and box-vector conversion

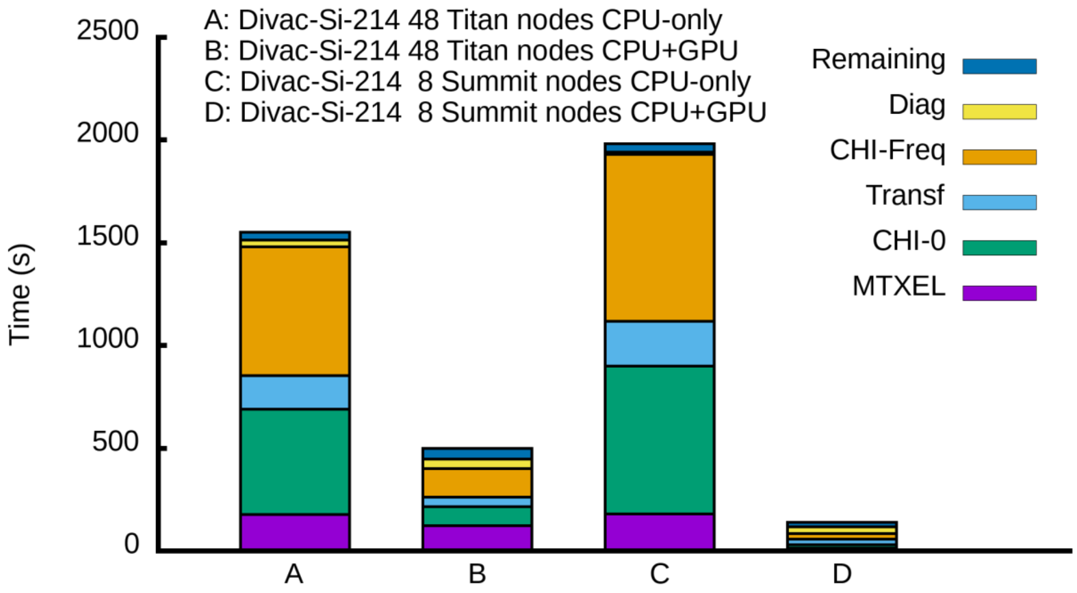
Epsilon Module (CHI-0 Kernel)



Non-Blocking Cyclic Communication
(Example Task#2, second cycle)

- cuBLAS, pinned memory, CUDA streams, async copy
- Non-Blocking cyclic communication, overlap MPI comm. with GPU compute
- Batched to avoid OOM

Epsilon Module

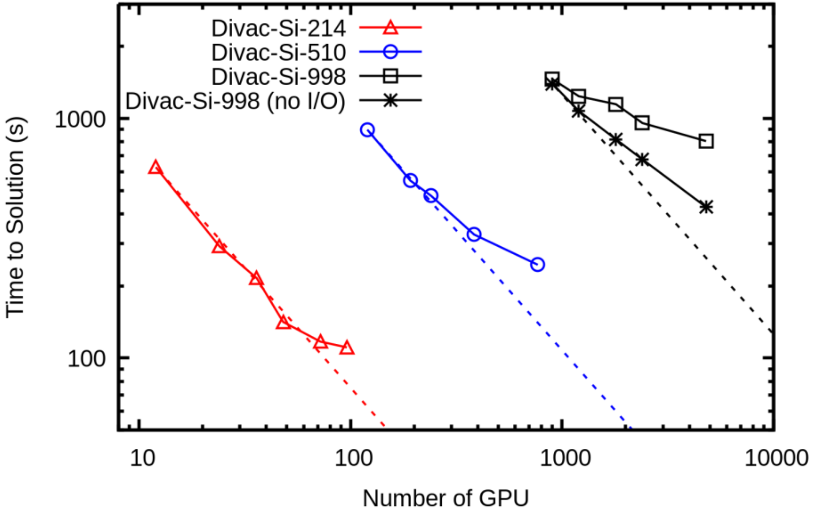


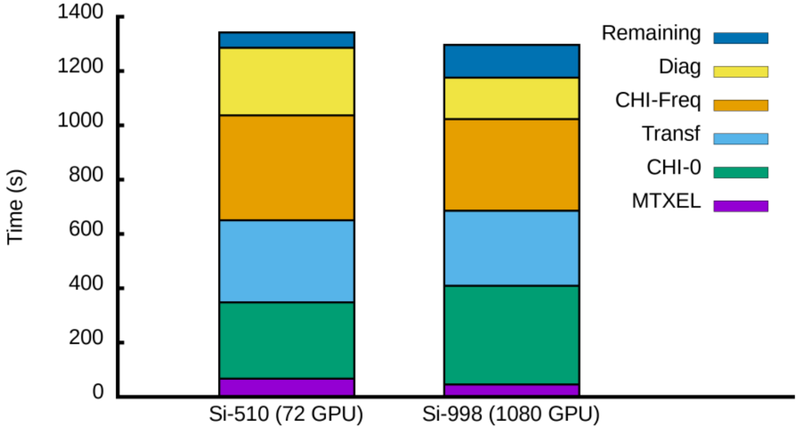
CPU+GPU vs CPU-only

- MTXEL: 12x speed-up
- CHI-0: 16x speed-up

Overall 14x!

Epsilon Module

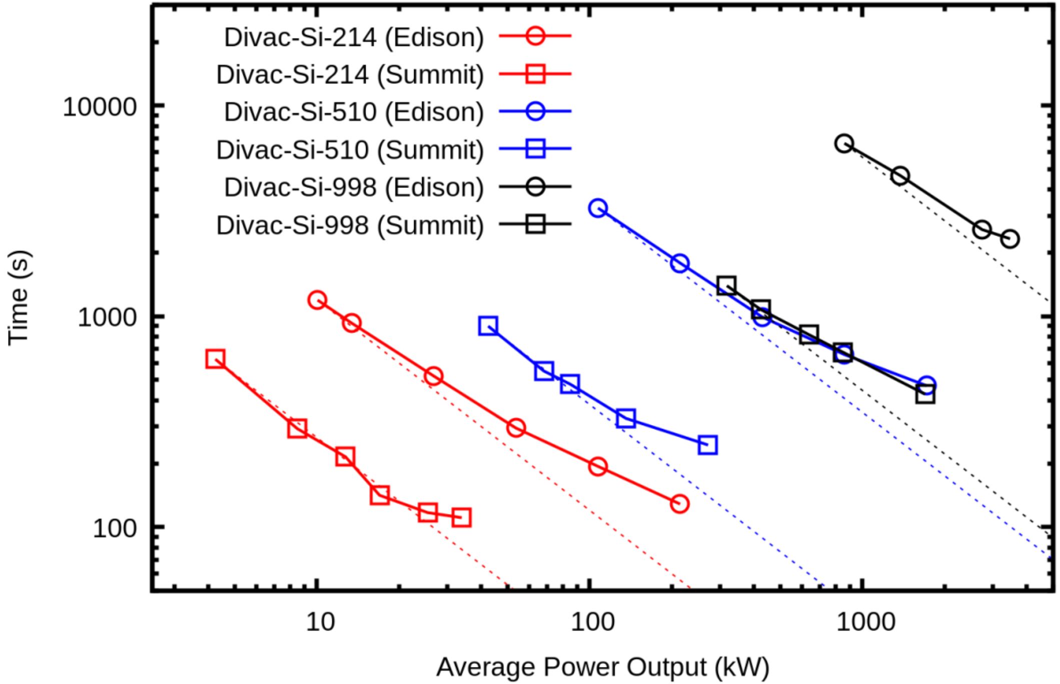




Strong scaling and weak scaling on Summit@OLCF

Left: Good parallel efficiency; still some parallel I/O issue for large scale calculations. Right: Good weak scaling; as problem size increases, memory grows to O(N^3) and FLOPs to O(N^4).

Epsilon Module



- **Comparison of power efficiency between Summit (V100 GPUs) and Edison (Xeon CPUs)**
- **GPUs are 16x more power efficient than CPUs consistently through three benchmarks!**

Sigma Module (GPP Kernel)

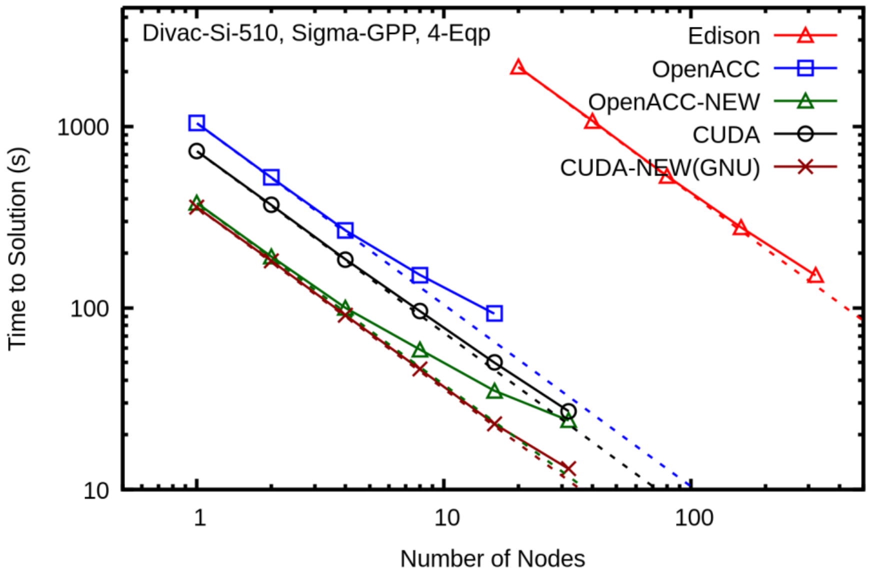
Implementations

- CUDA, more complete than the OpenACC version as of Sept 2019

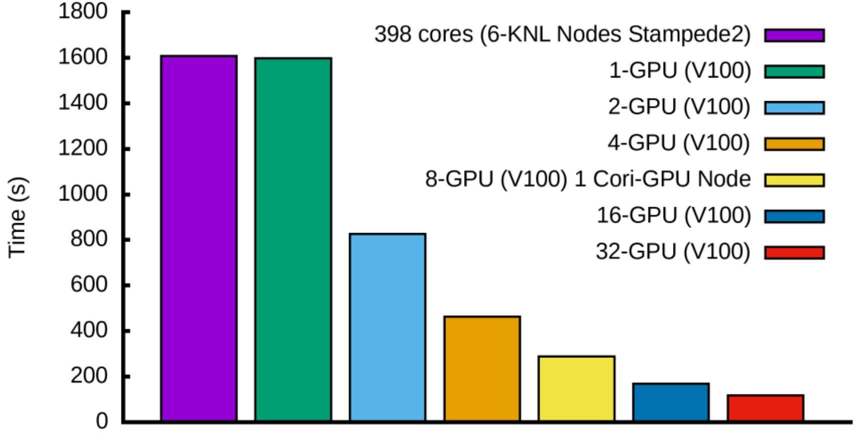
Strategies/Techniques

- Use streams for asynchronous data transfers and to increase concurrency
- Use a hybrid scheme for large reductions (100s-1000s of billions)
  - shared memory on GPU and OpenMP on CPU
- Overlap MPI communication with GPU computation
- Use batched operation for more flexible parallelism and to save memory

Sigma Module (GPP Kernel)

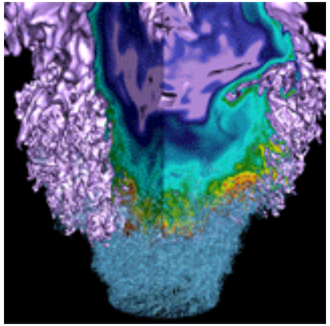


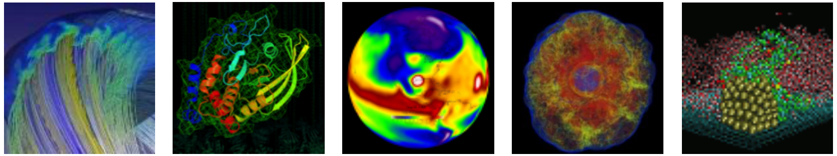
CUDA and OpenACC competing for best performance.



A 1:1 node comparison give a 33x speed-up of a Cori-GPU node vs a Stampede2-KNL node. (Timings are for a single k-point).

Summary





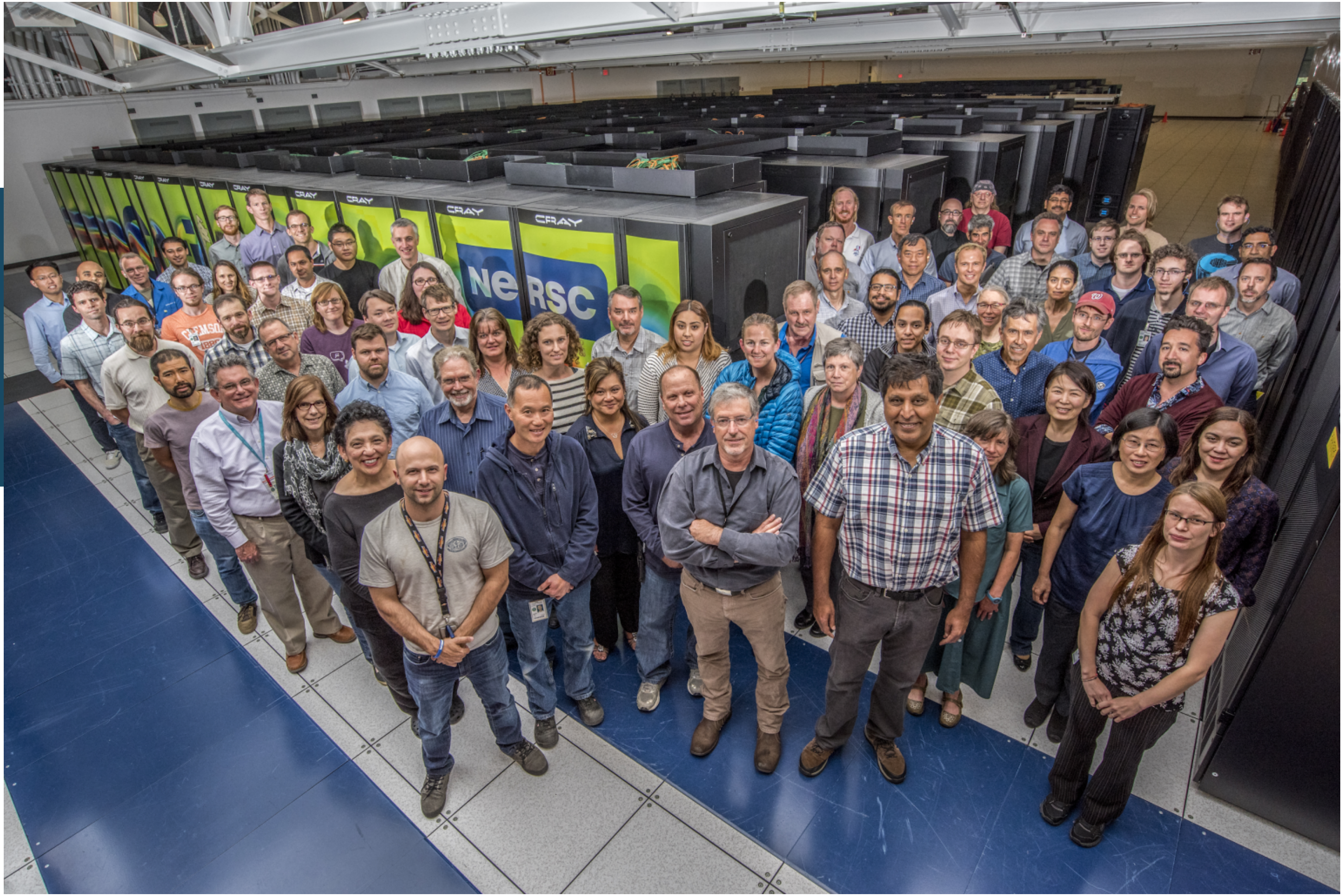
NERSC-9: A System Optimized for Science

- **Cray Shasta System providing 3-4x capability of Cori system**
- **First NERSC system designed to meet needs of both large scale simulation and data analysis from experimental facilities**
  - Includes both NVIDIA GPU-accelerated and AMD CPU-only nodes
  - Cray Slingshot high-performance network will support Terabit rate connections to system
  - Optimized data software stack enabling analytics and ML at scale
  - All-Flash filesystem for I/O acceleration
- **Robust readiness program for simulation, data and learning applications and complex workflows**

NERSC is hiring!

- Postdoctoral fellows
  - including Grace Hopper fellowship
- Application performance specialists

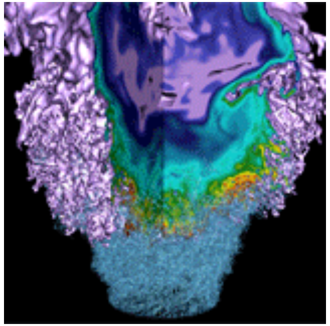
Thank You !

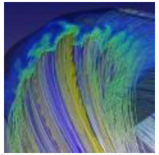


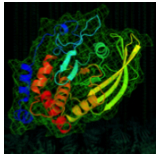
U.S. DEPARTMENT OF ENERGY | Office of Science

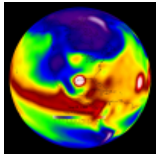
BERKELEY LAB

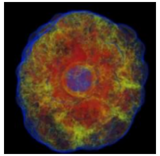
The end

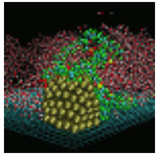












NERSC

Perlmutter was announced 30 Oct 2018

“Continued leadership in high performance computing is vital to America’s competitiveness, prosperity, and national security,” said U.S. Secretary of Energy Rick Perry. “This advanced new system, created in close partnership with U.S. industry, will give American scientists a powerful new tool of discovery and innovation and **will be an important milestone on the road to the coming era of exascale computing.**”

"We are very excited about the Perlmutter system," said NERSC Director Sudip Dosanjh. “It will provide a significant increase in capability for our users and **a platform to continue transitioning our very broad workload to energy efficient architectures.** The system is optimized for science, and we will collaborate with Cray, NVIDIA and AMD to ensure that Perlmutter meets the **computational and data needs of our users.** We are also launching a major power and cooling upgrade in Berkeley Lab’s Shyh Wang Hall, home to NERSC, to prepare the facility for Perlmutter.”