Preparing Applications for Exascale and Beyond: Initial Lessons From the Exascale Computing Project

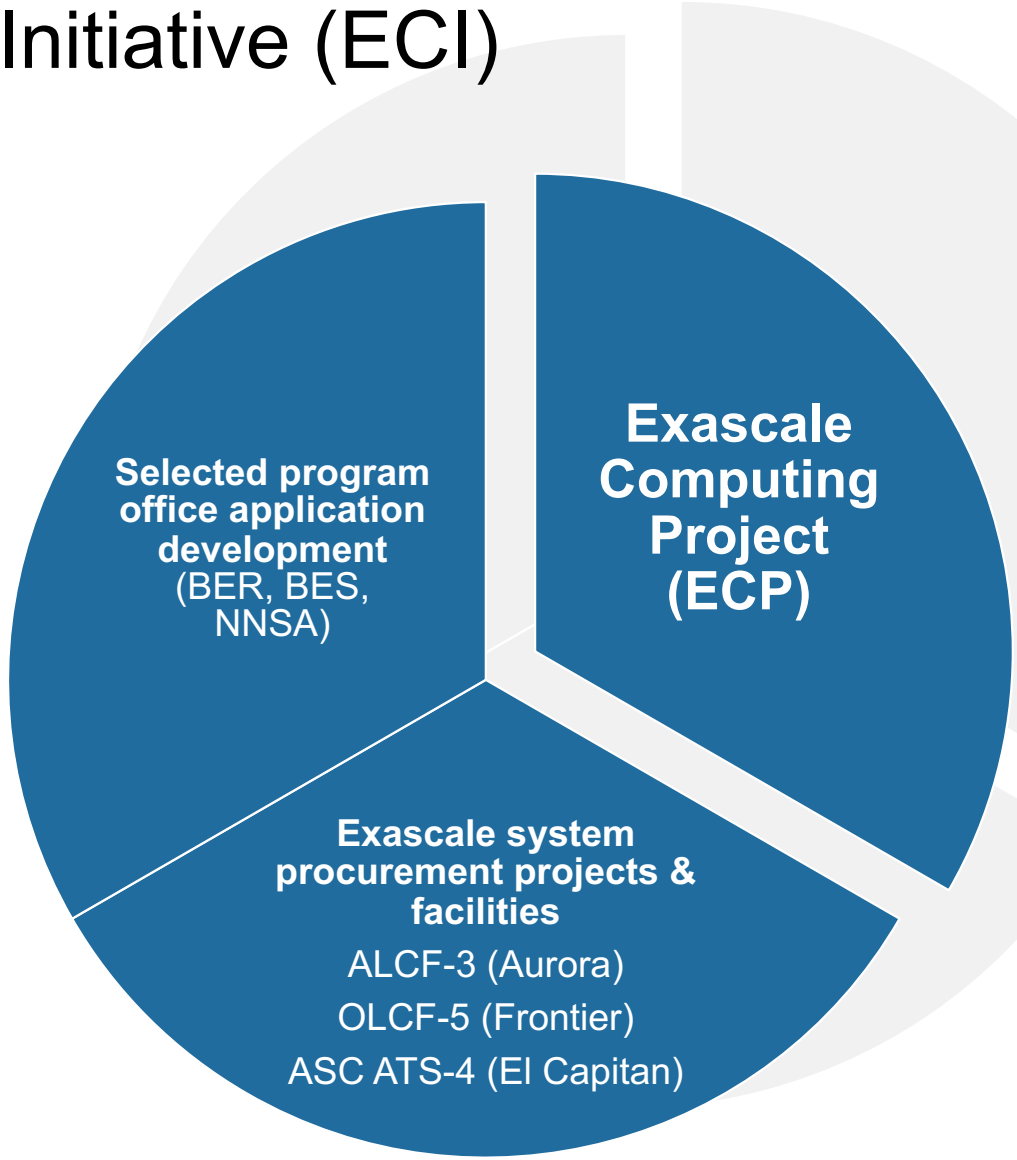
Erik Draeger (LLNL), Deputy Director of Application Development, ECP

44th ORAP Forum
November 29, 2019
Paris, France

LLNL-PRES-796967

National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY | Office of Science

The ECP is part of the broader DOE Exascale Computing Initiative (ECI)



Three Major Components of the ECI

ECI partners

US DOE Office of Science (SC) and National Nuclear Security Administration (NNSA)

ECI mission

Accelerate R&D, acquisition, and deployment to deliver exascale computing capability to DOE national labs by the early- to mid-2020s

ECI focus

Delivery of an *enduring and capable exascale computing capability for use by a wide range of applications* of importance to DOE and the US

ECP by the Numbers

A seven-year, $1.7B R&D effort that launched in 2016

Six core DOE National Laboratories: Argonne, Lawrence Berkeley, Lawrence Livermore, Los Alamos, Oak Ridge, Sandia

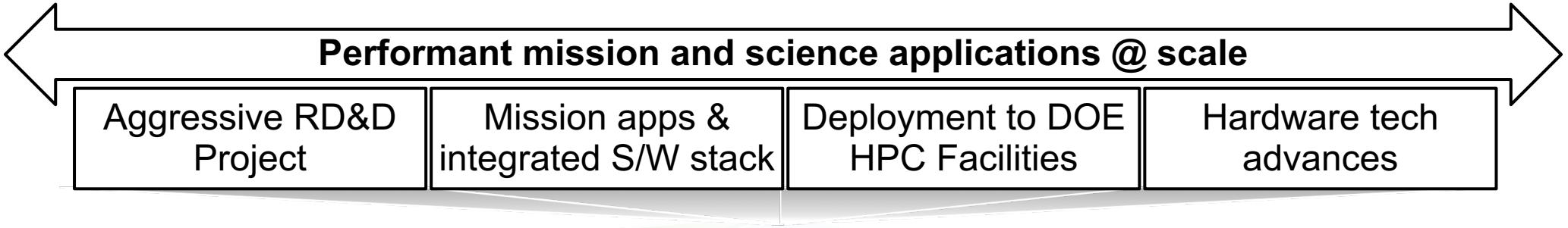
- Staff from most of the 17 DOE national laboratories take part in the project

Four focus areas: Hardware and Integration, Software Technology, Application Development, Project Management

More than 100 top-notch R&D teams

Hundreds of consequential milestones delivered on schedule and within budget since project inception

The three technical areas in ECP have the necessary components to meet national goals



Application Development (AD)

Develop and enhance the predictive capability of applications critical to the DOE

24 applications including national security, energy, earth systems, economic security, materials, and data

Software Technology (ST)

Deliver expanded and vertically integrated software stack to achieve full potential of exascale computing

66 unique software products spanning programming models and run times, math libraries, data and visualization

Hardware and Integration (HI)

Integrated delivery of ECP products on targeted systems at leading DOE HPC facilities

6 US HPC vendors focused on exascale node and system design; application integration and software deployment to facilities

ECP **Application Development (AD)**

Goal

Develop and enhance predictive capability of applications critical to DOE across science, energy, and national security mission space

- High impact science or engineering exascale *challenge problem*
- Detailed criteria for assessing successful completion of challenge problem
- A Figure of Merit (FOM) formula quantifying performance of challenge problem
- Demonstration and assessment of effective software integration

Chemistry and Materials

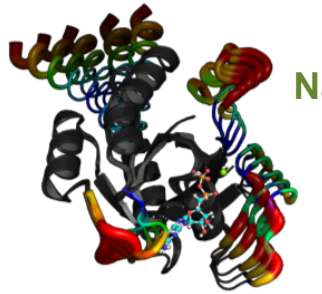
Earth and Space Science

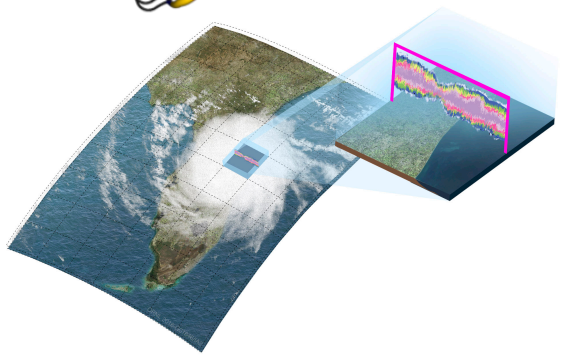
Energy

Data Analytics and Optimization

National Security

Co-Design





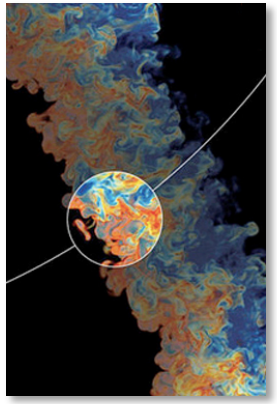
AD subprojects target national problems in DOE mission areas

National security

Next-generation, **stockpile stewardship** codes

Reentry-vehicle-environment simulation

Multi-physics science simulations **of high-energy density physics** conditions



Energy security

Turbine **wind plant** efficiency

Design and commercialization of **SMR**s

Nuclear fission and fusion reactor **materials design**

Subsurface use for **carbon capture**, petroleum extraction, waste disposal

High-efficiency, low-emission **combustion engine** and gas turbine design

Scale up of **clean fossil fuel** combustion

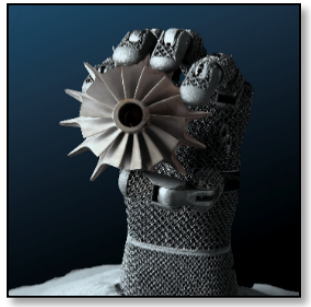
Biofuel catalyst design

Economic security

Additive manufacturing of qualifiable metal parts

Reliable and efficient planning of the **power grid**

Seismic hazard risk assessment



Scientific discovery

Cosmological probe of the standard model of particle physics

Validate fundamental laws of nature

Plasma wakefield accelerator design

Light source-enabled **analysis of protein and molecular structure** and design

Find, predict, and control materials and properties

Predict and control **magnetically confined fusion plasmas**

Demystify **origin of chemical elements**

Earth system

Accurate regional impact assessments in **Earth system models**

Stress-resistant crop analysis and catalytic conversion of **biomass-derived alcohols**

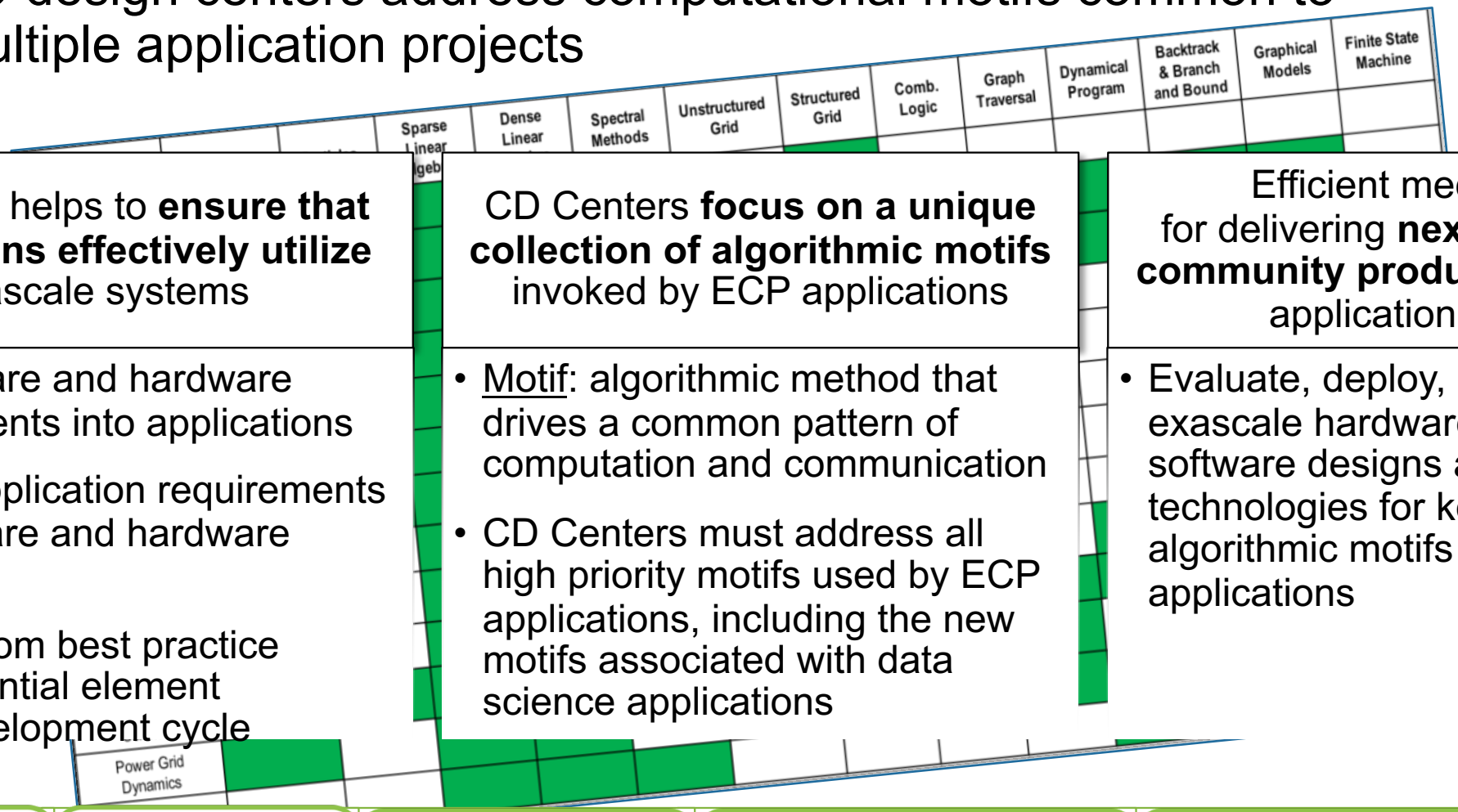
Metagenomics for analysis of biogeochemical cycles, climate change, environmental remediation

Health care

Accelerate and translate **cancer research** (partnership with NIH)

Co-design Subprojects

- Co-design centers address computational motifs common to multiple application projects



Co-design helps to **ensure that applications effectively utilize** exascale systems

- Pull software and hardware developments into applications
- Pushes application requirements into software and hardware RD&D
- Evolved from best practice to an essential element of the development cycle

CD Centers **focus on a unique collection of algorithmic motifs** invoked by ECP applications

- Motif: algorithmic method that drives a common pattern of computation and communication
- CD Centers must address all high priority motifs used by ECP applications, including the new motifs associated with data science applications

Efficient mechanism for delivering **next-generation community products** with broad application impact

- Evaluate, deploy, and integrate exascale hardware-aware software designs and technologies for key crosscutting algorithmic motifs into applications

CODAR *Data and workflows*	**COPA** *Particles/mesh methods*	**AMReX** *Block structured AMR*	**CEED** *Finite element discretization*	**ExaGraph** *Graph-based algorithms*	**ExaLearn** *Machine Learning*

Department of Energy (DOE) Roadmap to Exascale Systems

An impressive, productive lineup of *accelerated node* systems supporting DOE's mission

Pre-Exascale Systems | **First U.S. Exascale Systems***

2012	2016	2018	2020	2021-2023

ORNL
Cray/AMD/NVIDIA

ORNL
IBM/NVIDIA

ANL
IBM BG/Q

ANL
Cray/Intel KNL

LBNL
Cray/Intel Xeon/KNL

LLNL
IBM BG/Q

LANL/SNL
Cray/Intel Xeon/KNL

LLNL
IBM/NVIDIA

ORNL *
Cray/AMD

ANL *
Cray/Intel

LBNL
Cray/AMD/NVIDIA

Three different types of GPUs!

LANL/SNL
TBD

LLNL*
TBD

EXASCALE COMPUTING PROJECT

ECP applications need to be portable and resilient against future hardware changes

- Many codes are being used to fulfill current mission needs and need to remain usable during refactoring
- Rewriting large, validated codes is a major effort, redoing this every few years is not an option.
- HPC resources available to researchers can be diverse, want to be able to use with minimum of code modification

San Francisco-Oakland Bay Bridge Replacement

ECP applications are using a variety of programming models and implementation strategies

GPU-specific kernels

- Isolate the computationally-intensive parts of the code into CUDA/HIP/SYCL kernels.
- Refactoring the code to work well with the GPU is the majority of effort.

Loop pragma models

- Offload loops to GPU with OpenMP or OpenACC.
- Most common portability strategy for Fortran codes.

C++ abstractions

- Fully abstract loop execution and data management using advanced C++ features.
- Kokkos and RAJA developed by NNSA in response to increasing hardware diversity.

Co-design frameworks

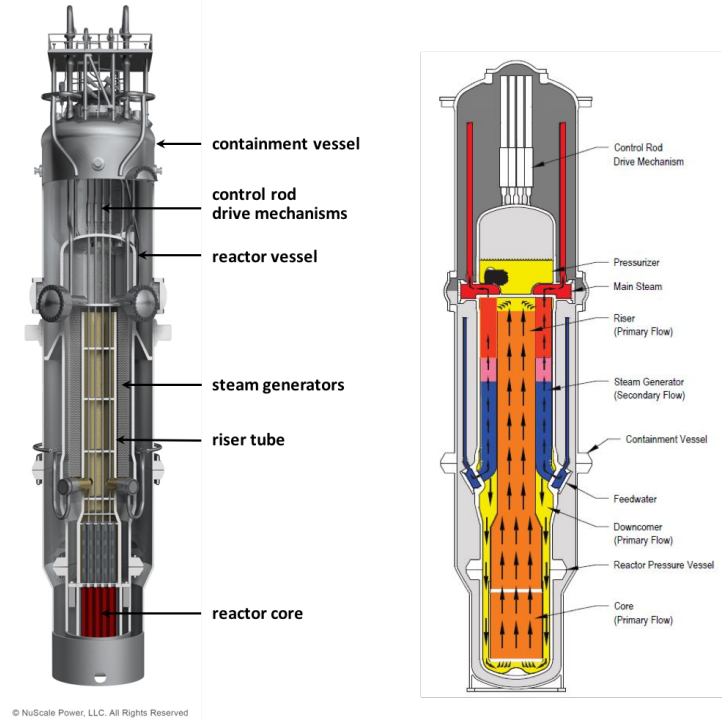
- Design application with a specific motif to use common software components
- Depend on co-design code (e.g. CEED, AMReX) to implement key functions on GPU.

ExaSMR: Small Modular Reactors

PI: Steve Hamilton, ORNL
Institutions: ANL, INL, MIT

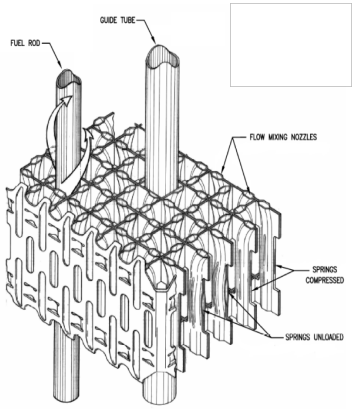
ExaSMR: modeling and simulation of small modular reactors

- Small modular nuclear reactors present significant simulation challenges
  - Small size invalidates existing low-order models
  - Natural circulation flow requires high-fidelity fluid flow simulation
- ExaSMR will couple most accurate available methods to perform “virtual experiment” simulations
  - Monte Carlo neutronics
  - CFD with turbulence models



Reproduced with permission

MC Neutronics		CFD	
Petascale	**Exascale**	**Petascale**	**Exascale**
• System-integrated responses • Single physics • Constant temperature • Isotopic depletion on assemblies • Reactor startup	• Pin-resolved (and sub-pin) responses • Coupled with T/H • Variable temperatures • Isotopic depletion on full core • Full-cycle modeling	• Single fuel assembly • RANS • Within-core flow	• Full reactor core • Hybrid LES/RANS • Entire coolant loop



Fuel assembly mixing vane

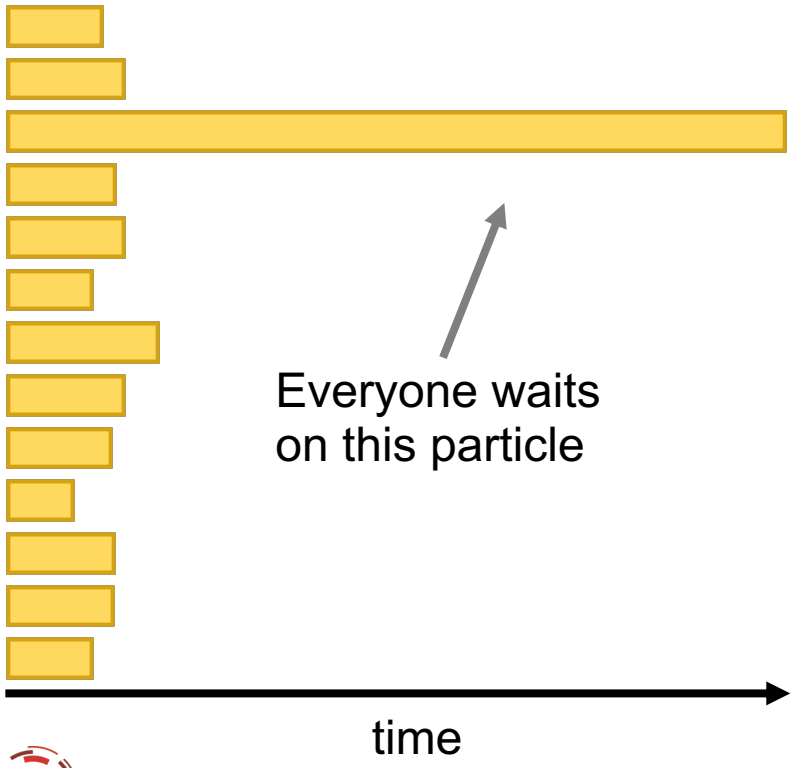
ExaSMR had to reexamine algorithms based on the properties of the hardware

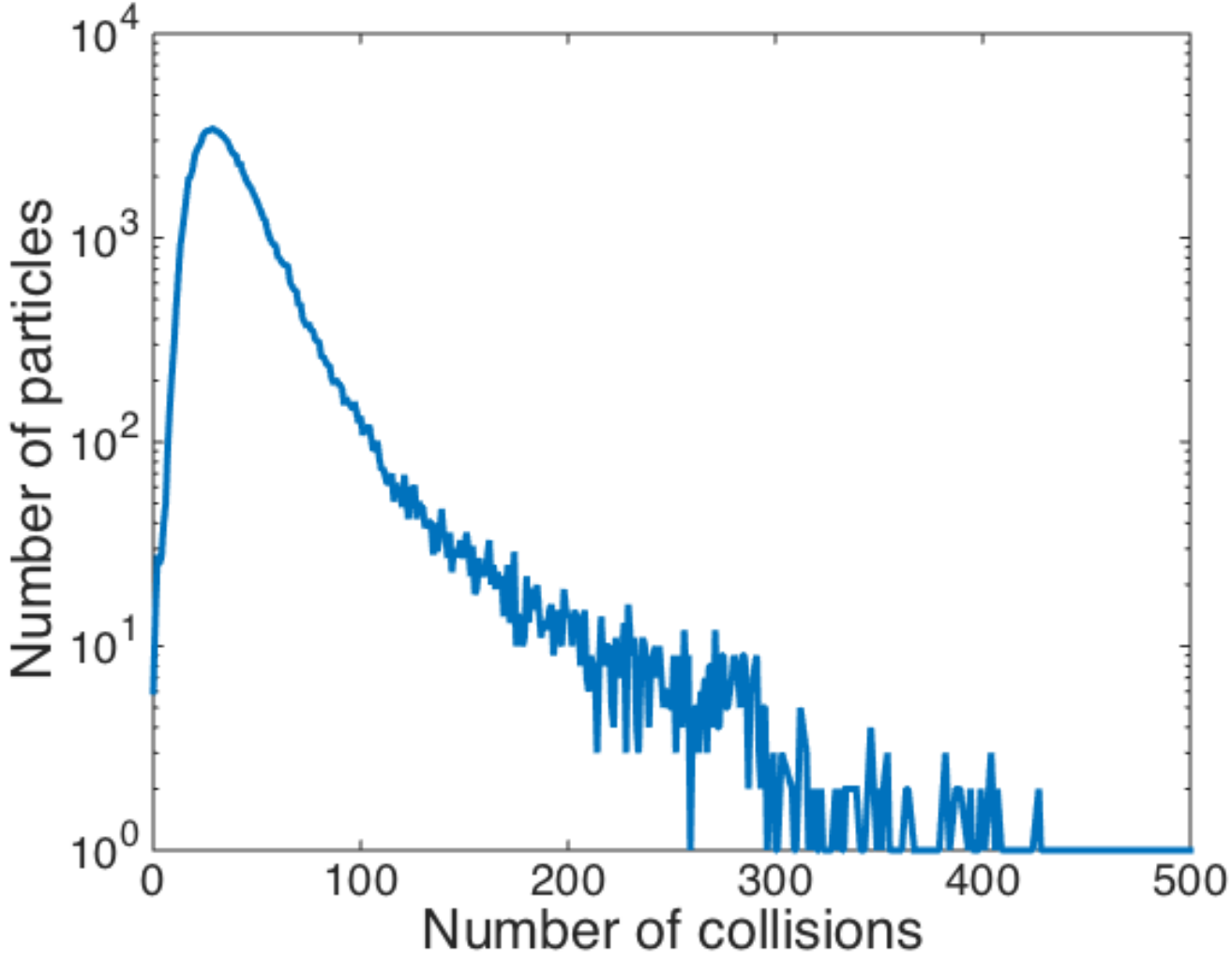
NVIDIA GPU execution model:

- Threads are grouped into *warps* of 32 threads that execute together in lockstep
- *Thread divergence* occurs when threads in same warp take different code paths
- GPU maintains more active warps than it can execute simultaneously
  - Switching between execution of different warps hides memory latency
  - Complex routines require significant resources per thread and reduce *occupancy*
- Substantial concurrency (>10^4 threads) is necessary to saturate a single GPU

Randomness of Monte Carlo is poorly suited to GPUs

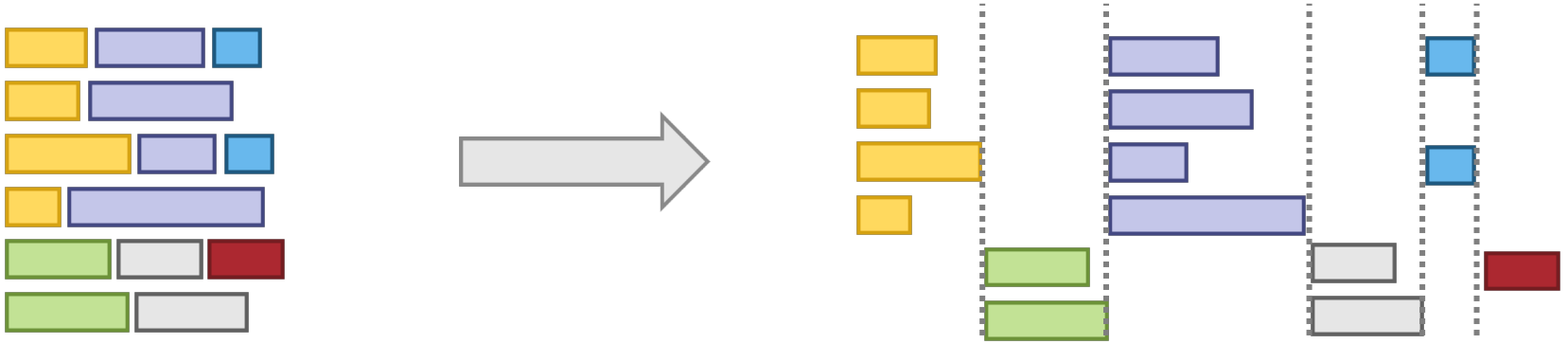
- Stochastic history-based algorithm follows particles from birth to death.
- Most particles are short-lived, a few are not.





Randomness of Monte Carlo is poorly suited to GPUs

Even when each particle has roughly the same amount of work, thread divergence is still a big problem when random sampling sends them down different code paths



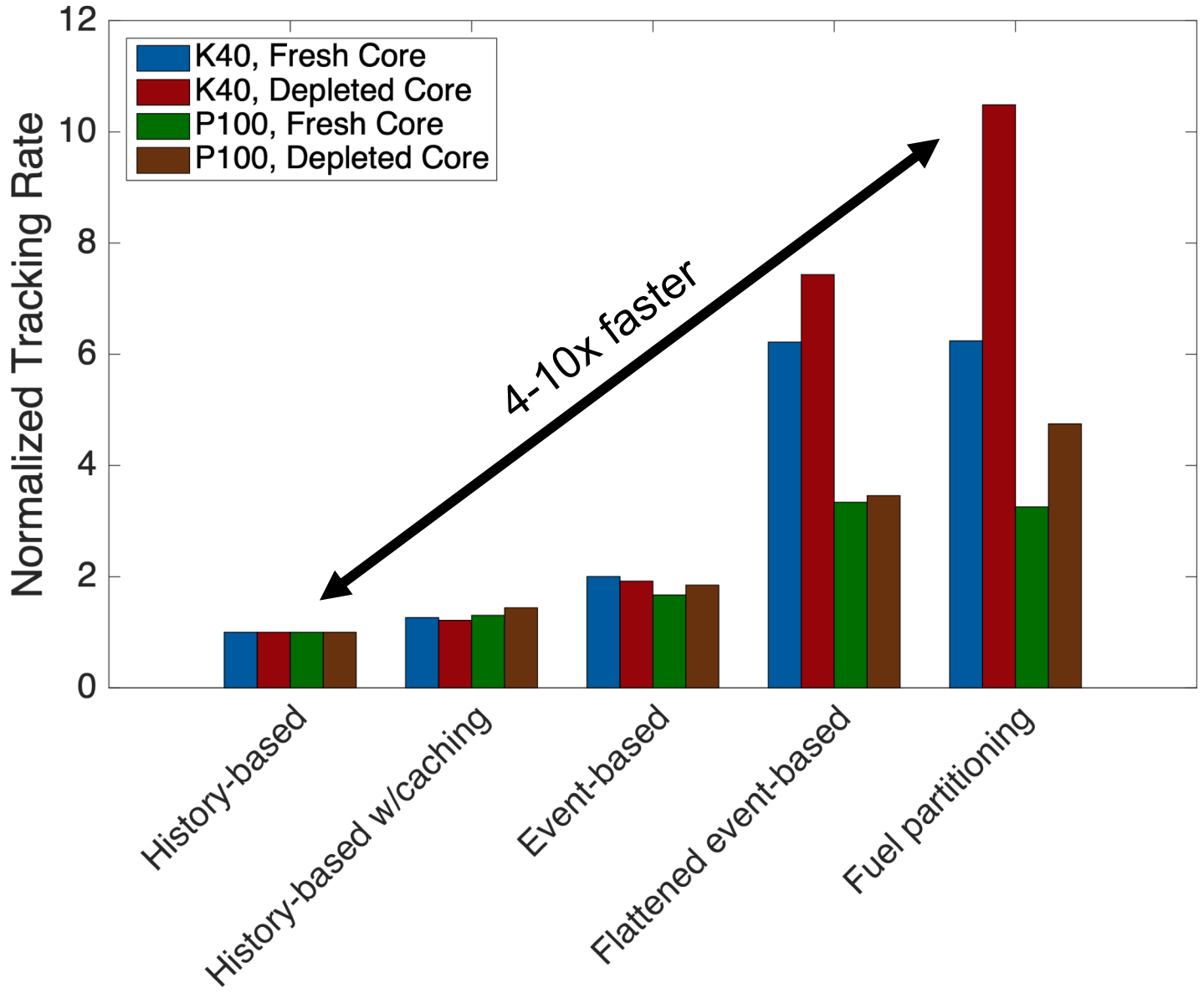
parallel work

GPU execution

Instead of parallelizing over particles, parallelize over **events** (i.e. common code paths)

New event-based algorithm shows significant speedup

- Parallelizing over events is a much better match for a SIMD/SIMT architecture than parallelizing over particles.
- Flattening: smaller, focused kernels allow for better occupancy, i.e. more efficient use of the hardware
- Further improvements gained by identifying parts of the system that have significantly different behavior and separating them out.



ECP applications are using a variety of programming models and implementation strategies

GPU-specific kernels

- Isolate the computationally-intensive parts of the code into CUDA/HIP/SYCL kernels.
- Refactoring the code to work well with the GPU is the majority of effort.

Loop pragma models

- Offload loops to GPU with OpenMP or OpenACC.
- Most common portability strategy for Fortran codes.

C++ abstractions

- Fully abstract loop execution and data management using advanced C++ features.
- Kokkos and RAJA developed by NNSA in response to increasing hardware diversity.

Co-design frameworks

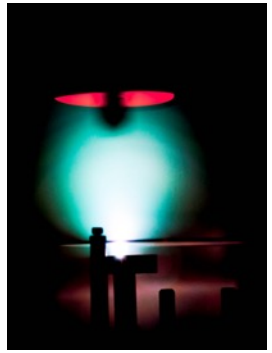
- Design application with a specific motif to use common software components
- Depend on co-design code (e.g. CEED, AMReX) to implement key functions on GPU.

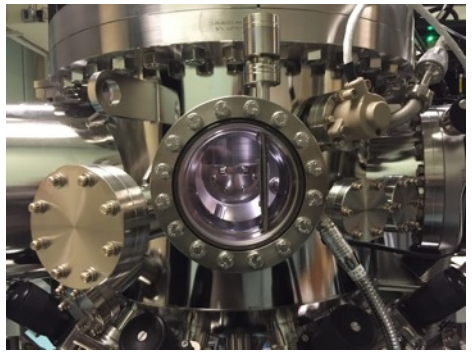
QMCPACK: Predictive and Improvable Quantum-mechanics Based Simulations

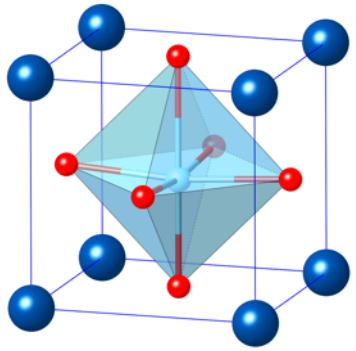
PI: Paul Kent, ORNL
Institutions: ANL, SNL, LLNL

Quantum Monte Carlo Methods for Materials

- We require reliable, non-empirical, atomistic quantum mechanics based methods to predict, understand, and design next generation materials
- QMCPACK implements multiple Quantum Monte Carlo (QMC) algorithms to achieve this. We focus on the **real-space diffusion Monte Carlo** (DMC) and orbital space **auxiliary field QMC** (AFQMC) algorithms to enable cross-validation.
- QMC methods have very few approximations – and they can potentially be systematically reduced – yielding predictive and reliable QM for materials where we currently lack these methods.
- Formal numerical scaling of each Monte Carlo step is $O(N^3)$, parallel scaling is high.



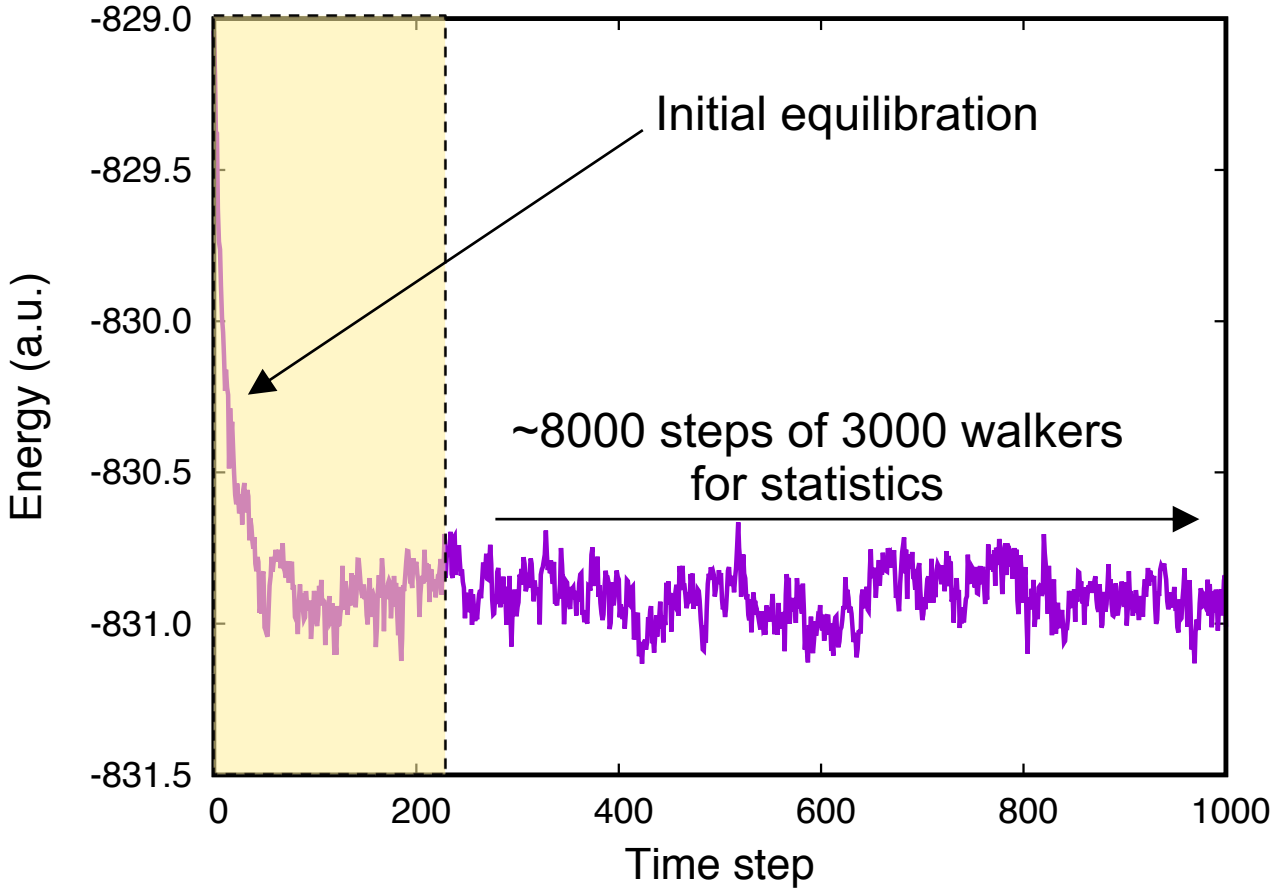




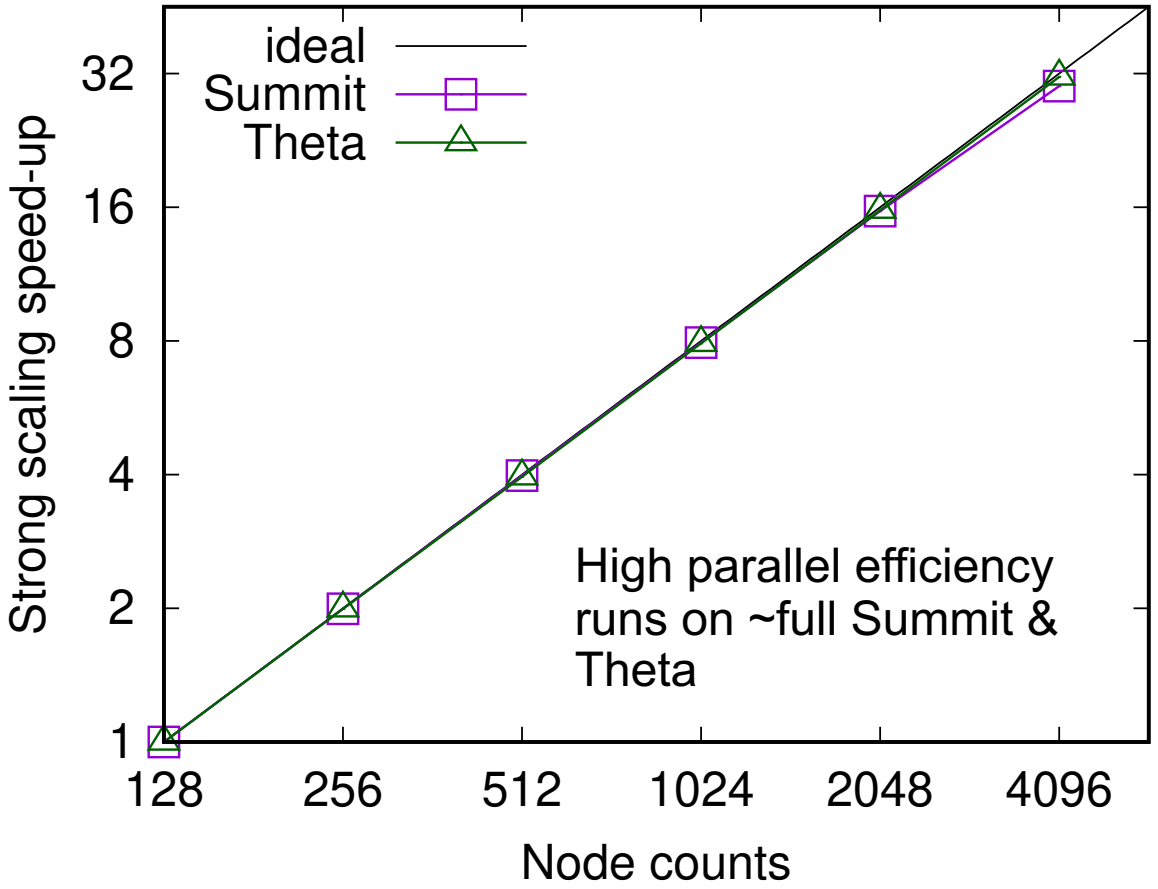
Word cloud of “popular” DFTs

Reducing time to science

Monte Carlo Markov chains (“walkers”) must be equilibrated. Simply adding more walkers to fill a large machine will eventually stop reducing time to reach a desired error bar. Improving time to science for large or complex problems will advance the state of the art: elements of strong & weak scaling required.



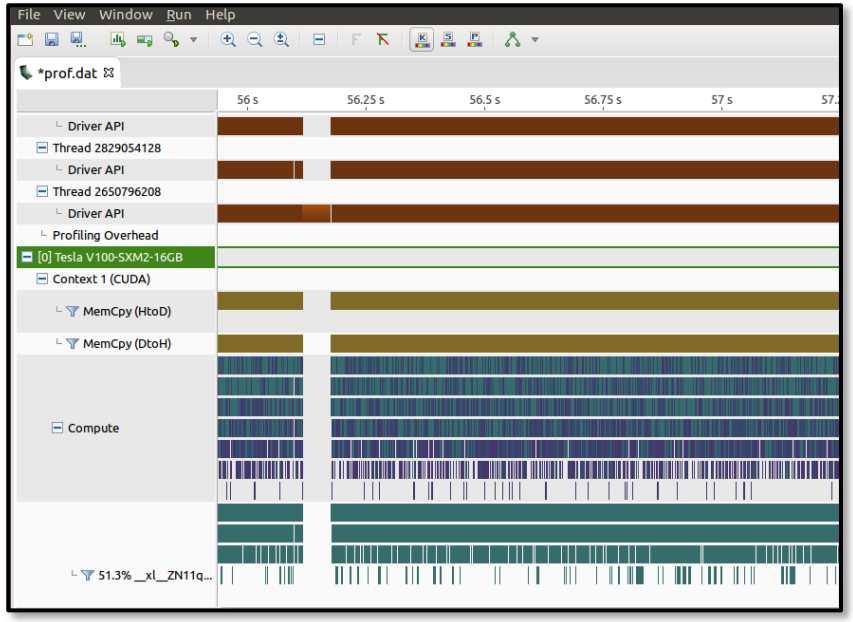
Production DMC calculation for bulk VO_2 from Kylanpaa PRM **1** 065408 (2017). 200 electrons, 0.01 a.u. timesteps



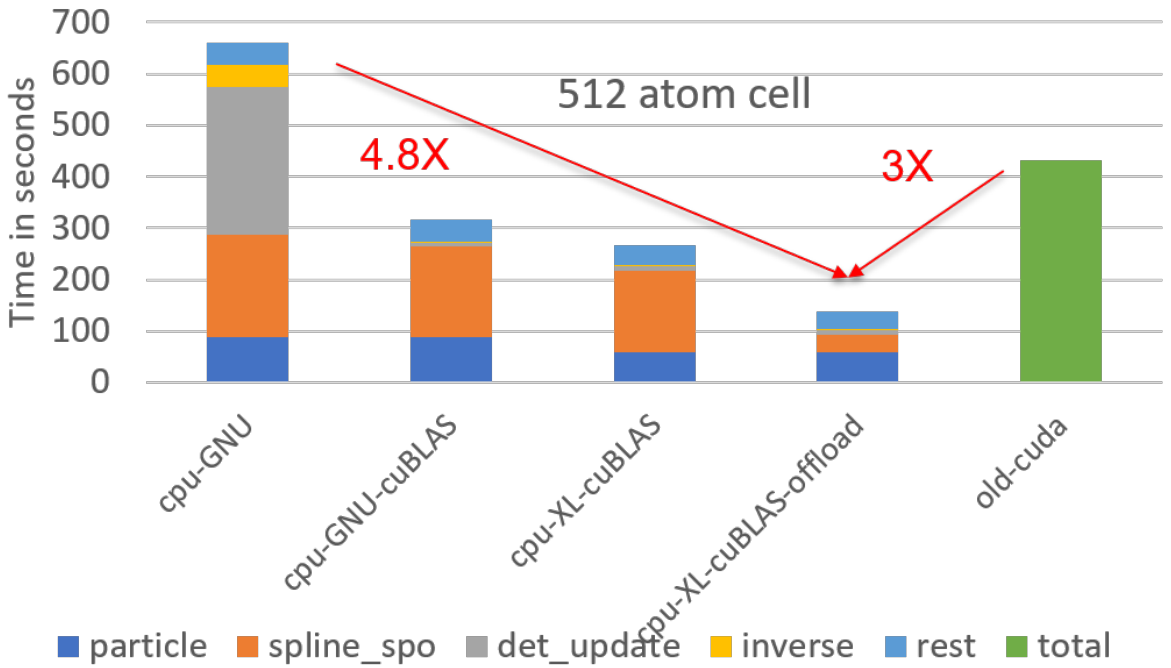
See J. Kim et al, JPCM **30** 195901 (2018) for discussion, plus Titan & Mira data

OpenMP offload gave good performance for large systems

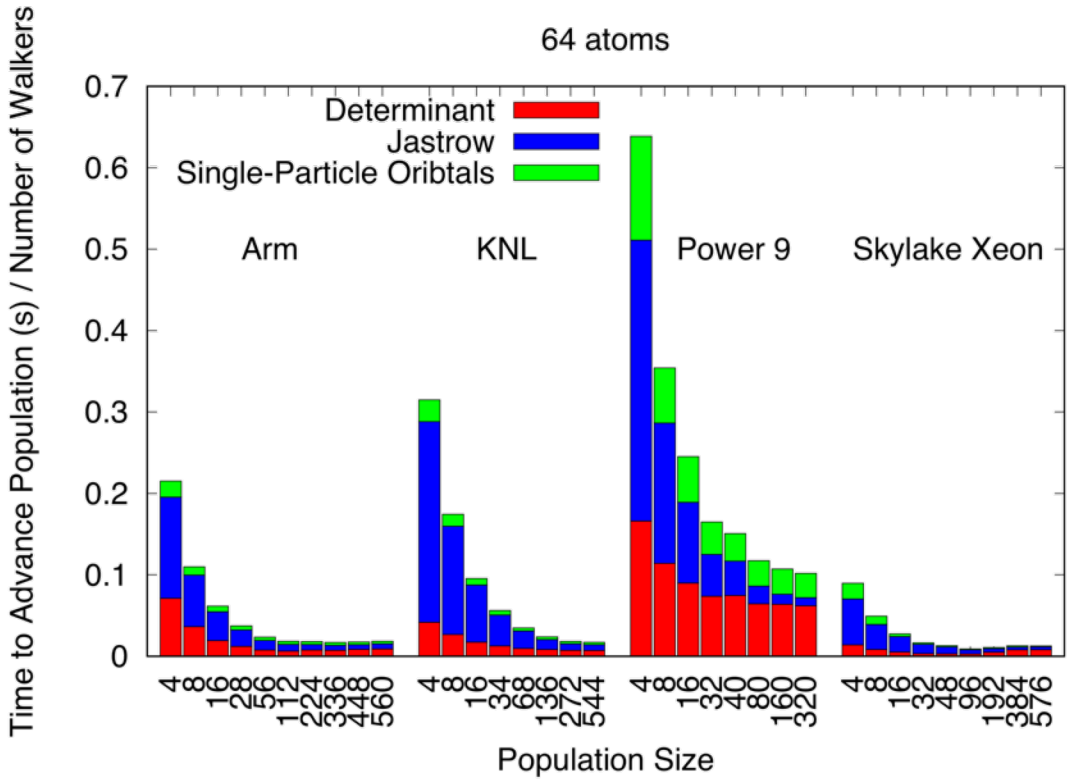
- OpenMP target/offload also implemented in mainline QMCPACK “one walker propagation at a time” (unbatched) SoA CPU code.
- For 512 atom cell, 3x speedup over legacy CUDA on Summit. Worse at 256 atoms.
- Projected 37x increase in FOM on Summit using IBM offload compiler.

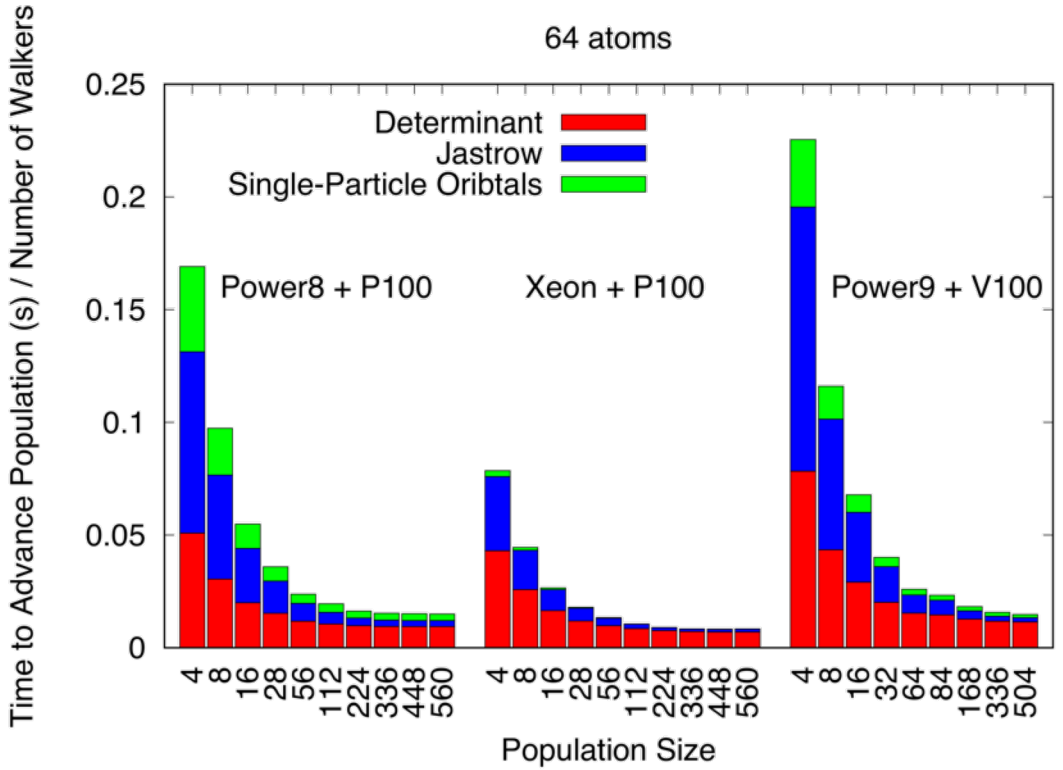


Reasonable GPU utilization for 512 atoms



For smaller systems, batched operations needed for performance





- Batching (multiple walker simultaneous propagation) needs to be included in performance portable design.
- This was implemented in miniapp to confirm it would significantly impact performance, implementation in mainline QMCPACK now underway.
- OpenMP gives best balance of performance, portability and maintainability.

ECP applications are using a variety of programming models and implementation strategies

GPU-specific kernels

- Isolate the computationally-intensive parts of the code into CUDA/HIP/SYCL kernels.
- Refactoring the code to work well with the GPU is the majority of effort.

Loop pragma models

- Offload loops to GPU with OpenMP or OpenACC.
- Most common portability strategy for Fortran codes.

C++ abstractions

- Fully abstract loop execution and data management using advanced C++ features.
- Kokkos and RAJA developed by NNSA in response to increasing hardware diversity.

Co-design frameworks

- Design application with a specific motif to use common software components
- Depend on co-design code (e.g. CEED, AMReX) to implement key functions on GPU.

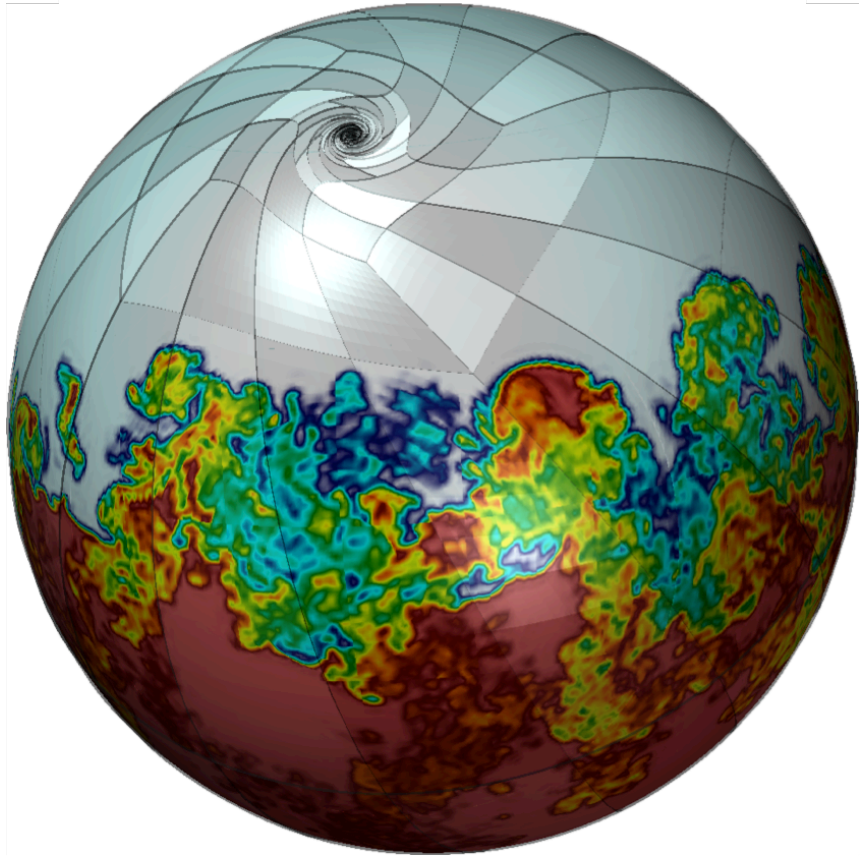
LLNL MAPP Multiphysics

PI: Rob Rieben, LLNL

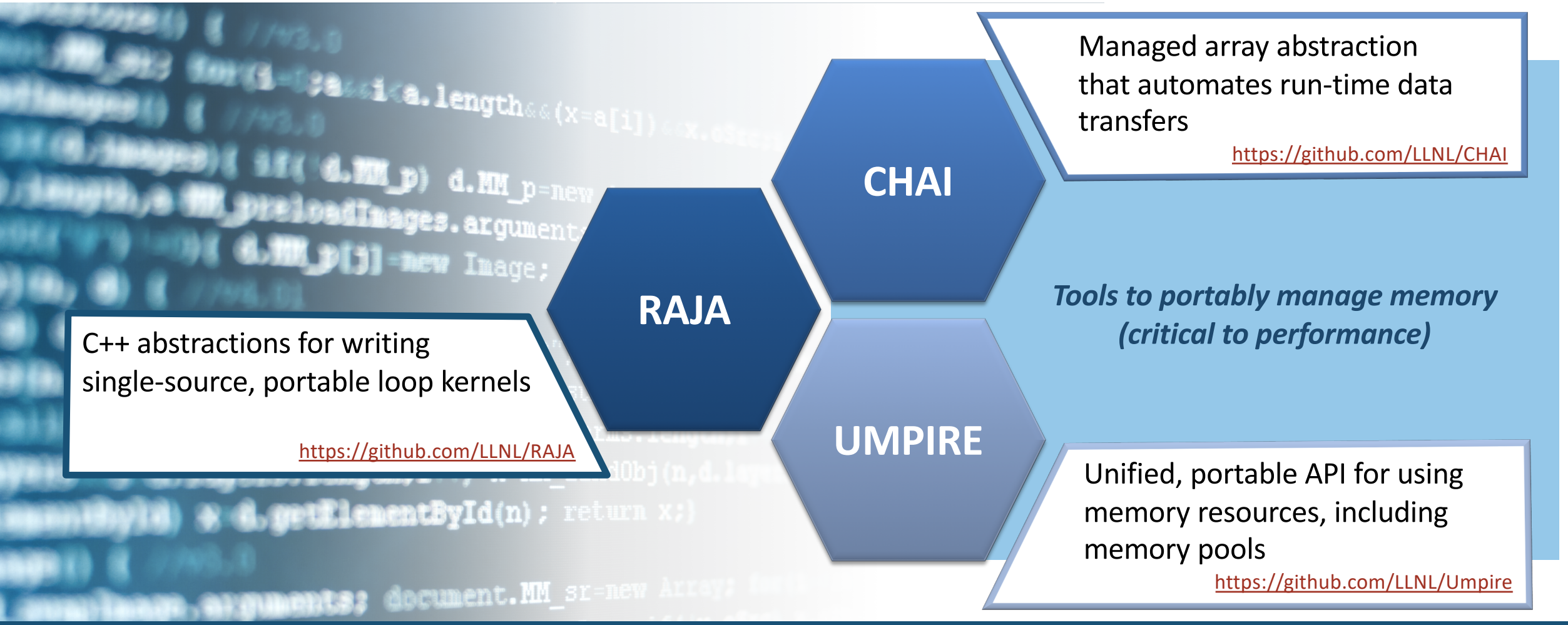
MARBL is a new code based on high-order numerical algorithms and modular CS infrastructure

MARBL: *A multi-physics application code for simulating HEDP and focused experiments driven by high-explosive, magnetic or laser based energy sources*

- **Based on modular physics packages**
  - Coupled with a shared Computer Science Toolkit
  - Emphasizing performance portability and flexibility
- **Distinguishing feature: high-order methods**
  - High-order finite-element ALE rad-hydro based on **Blast/mfem**
  - High-order finite-difference Direct Eulerian rad-hydro based on **Miranda**
- **Improved numerics and hardware utilization:**
  - Higher resolution/accuracy per unknown
  - Improved robustness
  - Higher FLOP/byte
  - Improved strong scaling



RAJA (and friends) provide applications with execution portability



Projects are co-designed and co-developed with WSC application teams

Basic RAJA features enable portable execution of simple loops

C-style for-loop

```
double* x ; double* y ;
double tdot = 0.0;

for ( int i = 0; i < N; ++i ) {
     tdot += x[i] * y[i] ;
}
```

RAJA-style loop

```
double* x ; double* y ;
RAJA::SumReduction<reduce_policy, double> tdot(0.0);

RAJA::forall< EXEC_POL > ( RAJA::RangeSegment(0, N), [=] (int i) {
     tdot += x[i] * y[i];
   }
);
```

Core RAJA concepts

- **Loop traversal template** (e.g., 'forall')
- **Execution policy** (seq, simd, openmp, cuda, etc.)
- **Iteration space** (range, index list, index set, etc.)
- **Loop body** (C++ lambda expression)

Built on standard C++11 features

RAJA extends the common "parallel-for" idiom to encapsulate loop execution details

RAJA supports a variety of parallel constructs and loop patterns

Simple & complex loop patterns

- Non-perfectly nested loops
- Loop tiling

Multiple execution back-ends

- Sequential
- SIMD
- OpenMP (CPU, target offload)
- CUDA
- AMD HIP (in progress)
- Intel Threading Building Blocks (experimental)

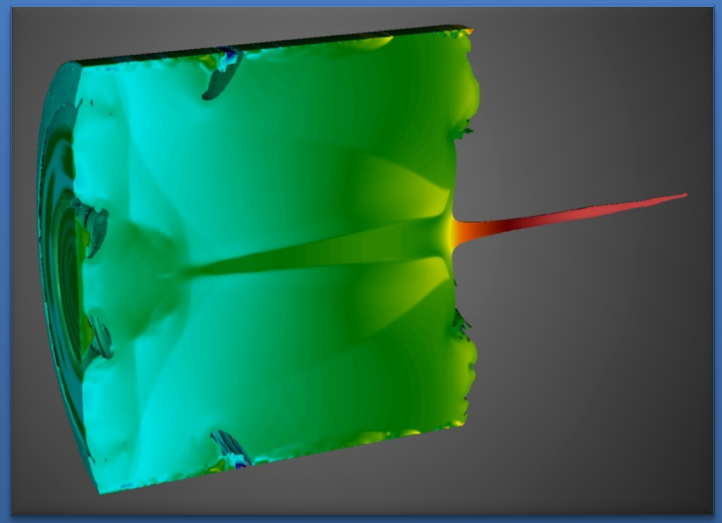
Kernel transformations
(via *execution policy* changes)

- Change order of loop iterations
- Permute loop nest ordering
- Multi-dimensional data views with offsets and index permutations
- Direct CUDA thread-block mapping control
- CPU/GPU shared and thread local memory

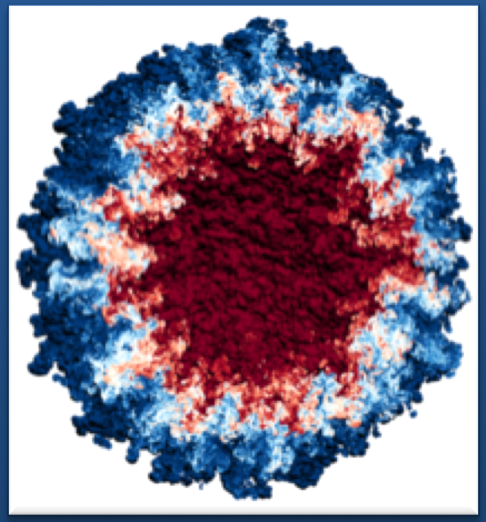
Portable reductions, scans, & atomic operations

LLNL investment in GPU-based computing (Sierra) is paying off with performance gains aligned with our expectations

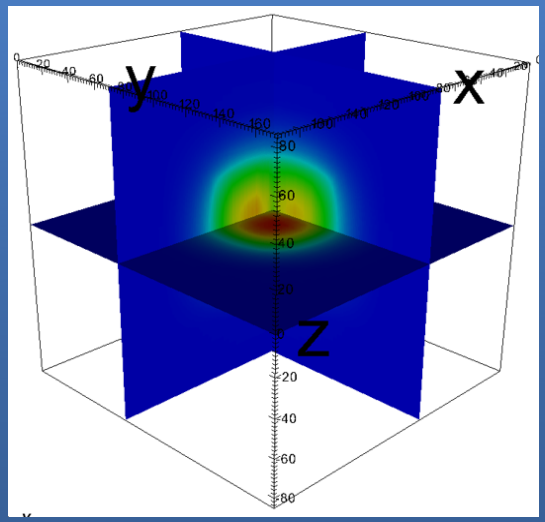
ALE3D Shaped Charge
8x speedup



Ares RT Mixing
13x speedup



Ardra Reactor Safety
16x speedup



Sierra speedups over Intel Xeon, node-for-node

Each optimization step improves performance *and* reveals the next problem to solve

GPUs have performance overheads not seen on CPUs

- **Kernel launch overhead:** Can be hidden with asynchronous kernel launches
- **Data transfer between memory spaces:** Must be avoided or hidden behind other kernels
- **GPU memory allocation is significantly more expensive:** Memory pools are a must!

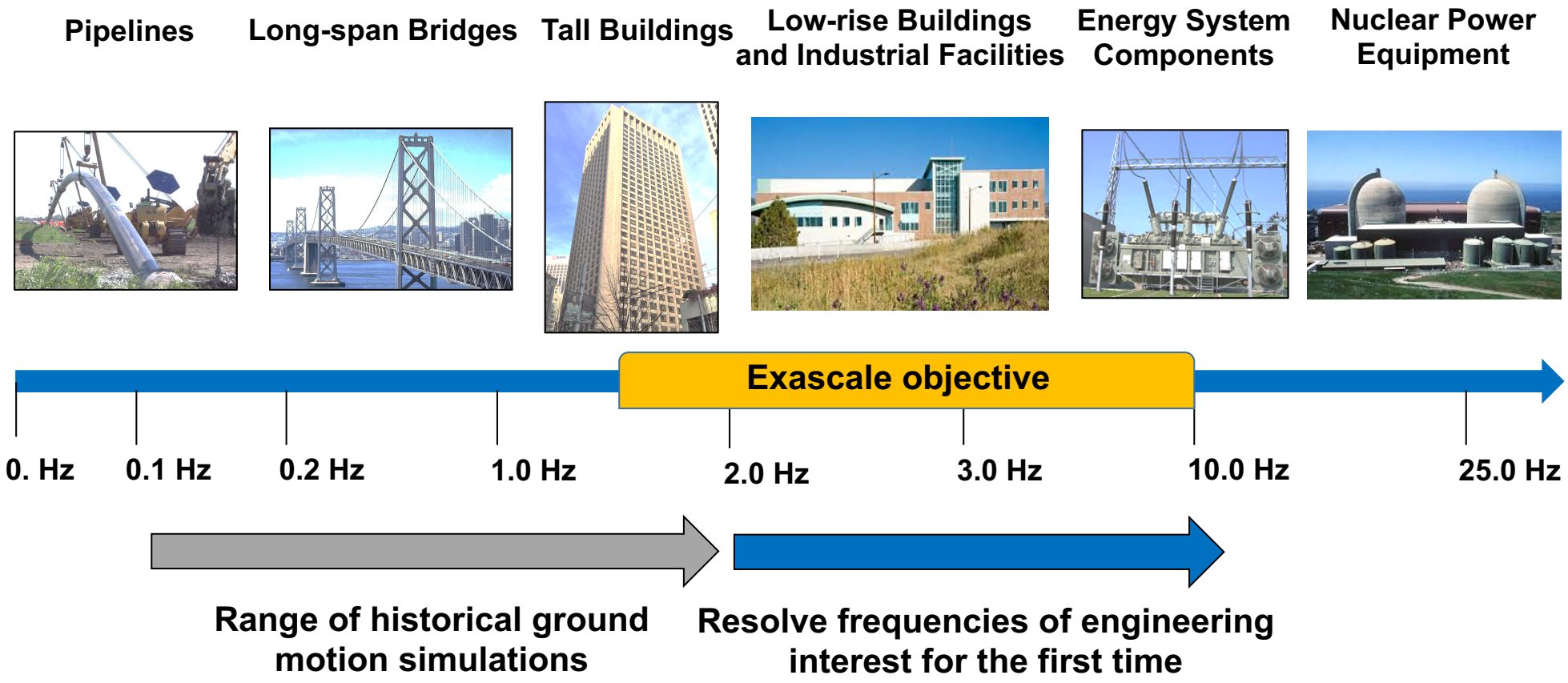
Optimization requires coordinating many parts

- **Libraries have different porting strategies/timelines:** Un-ported parts lead to costly CPU/GPU data transfers
- **GPU memory is a scarce resource to share:** Memory pools can help

EQSIM: Earthquake Hazard and Risk Assessment

PI: David McCallen, LBNL
Institutions: LLNL

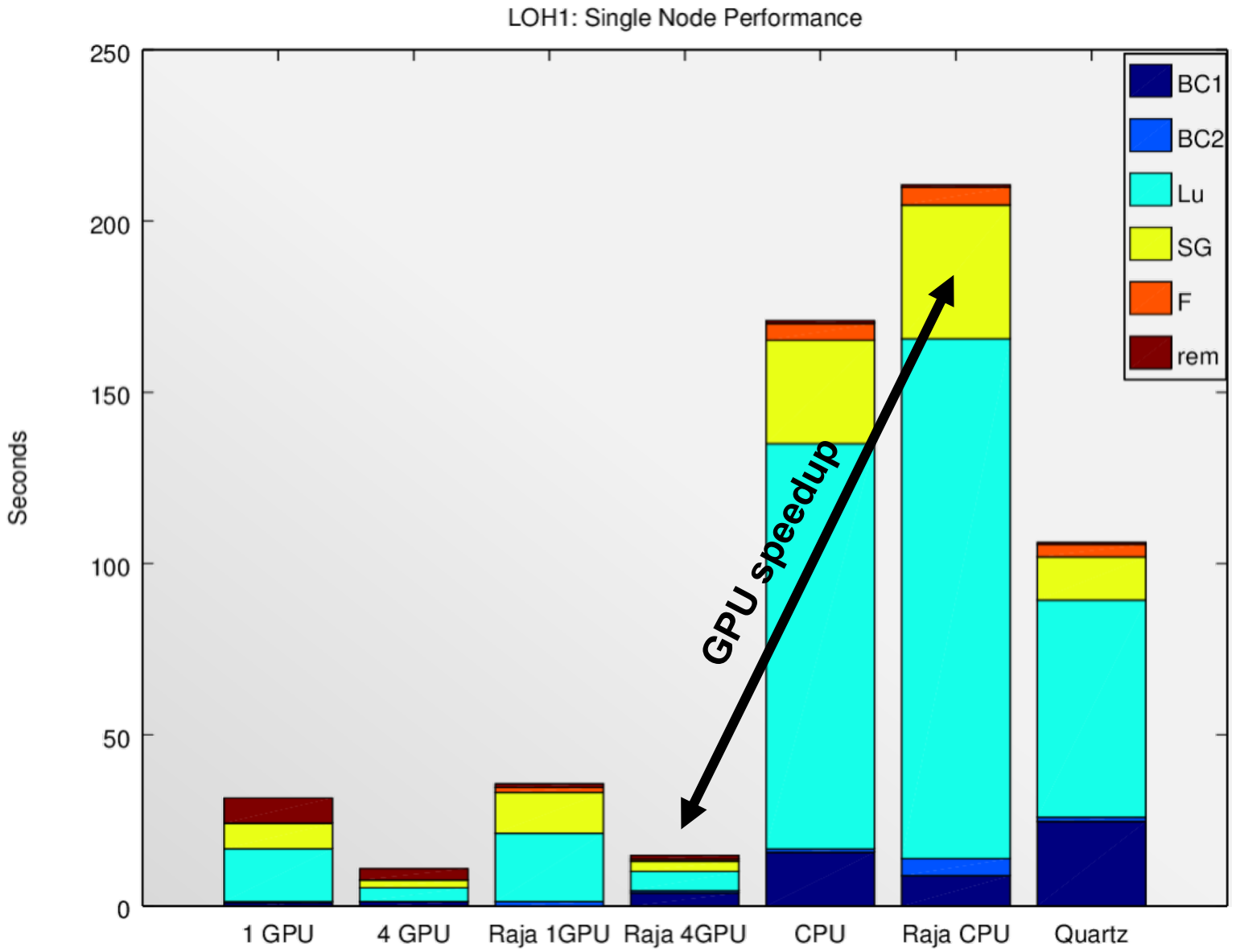
The Exascale challenge - ground motion simulations at “engineering” frequencies



Remove computation as a barrier to scientific discovery and practical earthquake hazard and risk analysis

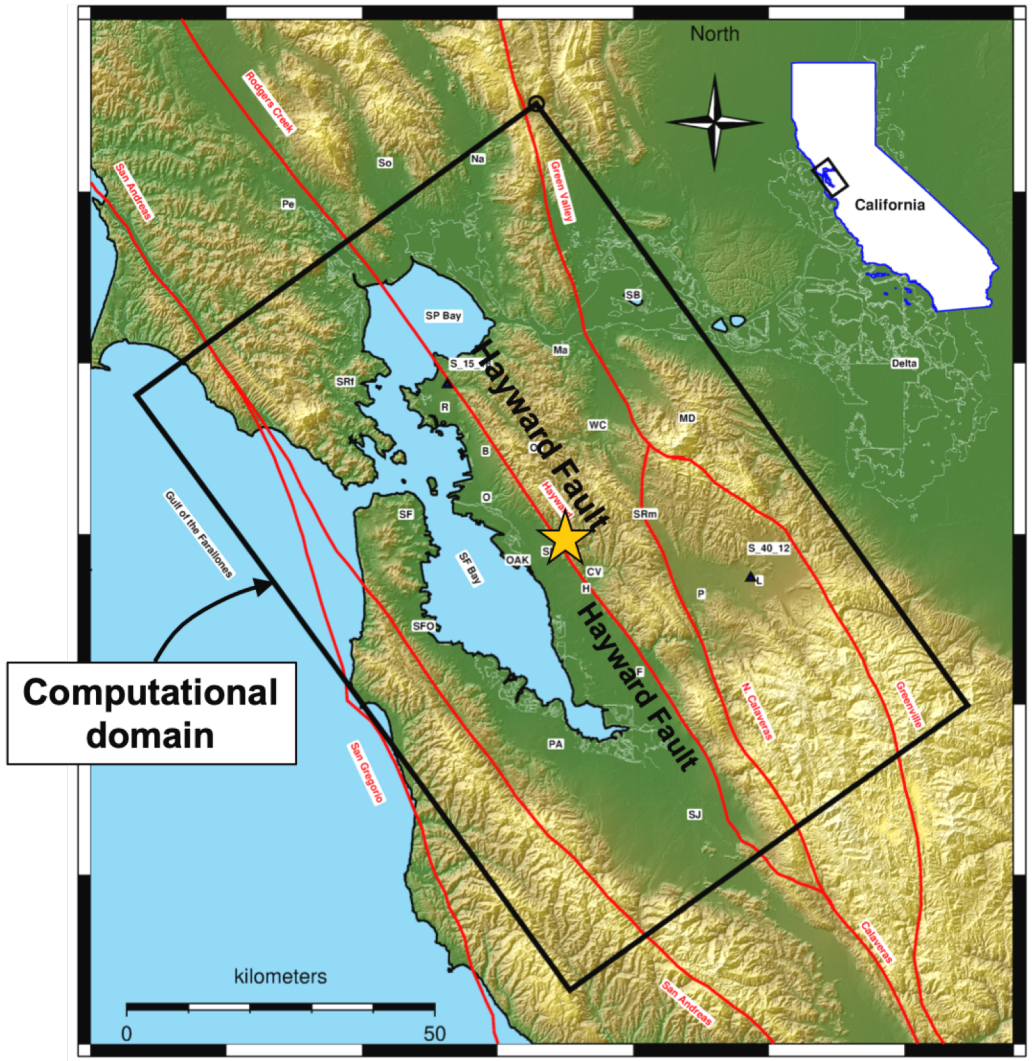
RAJA provided EQSIM with excellent performance portability

- EQSIM saw a large speedup on multi-GPU runs.
- RAJA is not a magic bullet:
  - Some performance overhead compared with direct CUDA
  - Abstractions can prevent compilers from optimizing
  - Loop execution still requires optimization and tuning
- Porting to future hardware will be substantially easier



Largest earthquake simulation ever performed run on Summit

San Francisco Bay Area Regional Scale Model



M=7.0 Hayward Fault Event Simulation	
Frequency Resolved	10 Hz
Vs_{min}	500 m/s
Number of Grid Points	203 Billion
Smallest Cell Size	6.25 m
Time Step Size	7.119e-4
Total Time Steps	126430
Platform	SUMMIT (ORNL)
Number of Nodes	1200
Wall Clock Time	19 hours, 52 minutes, one check point file created

Lessons learned

- Refactoring large established codes is a major effort. Learning from the lessons of others can help make sure you start down the best path for your application.
- Reuse shared components and software whenever you can to avoid duplication of effort.
- We are already seeing that GPU performance requires designing the code with awareness of the hardware, which is far more important than choosing the "right" programming model.

Questions?