Lessons Learned from Porting LLNL Applications to Sierra

GTC 2019

David M. Dawson
Lawrence Livermore National Laboratory

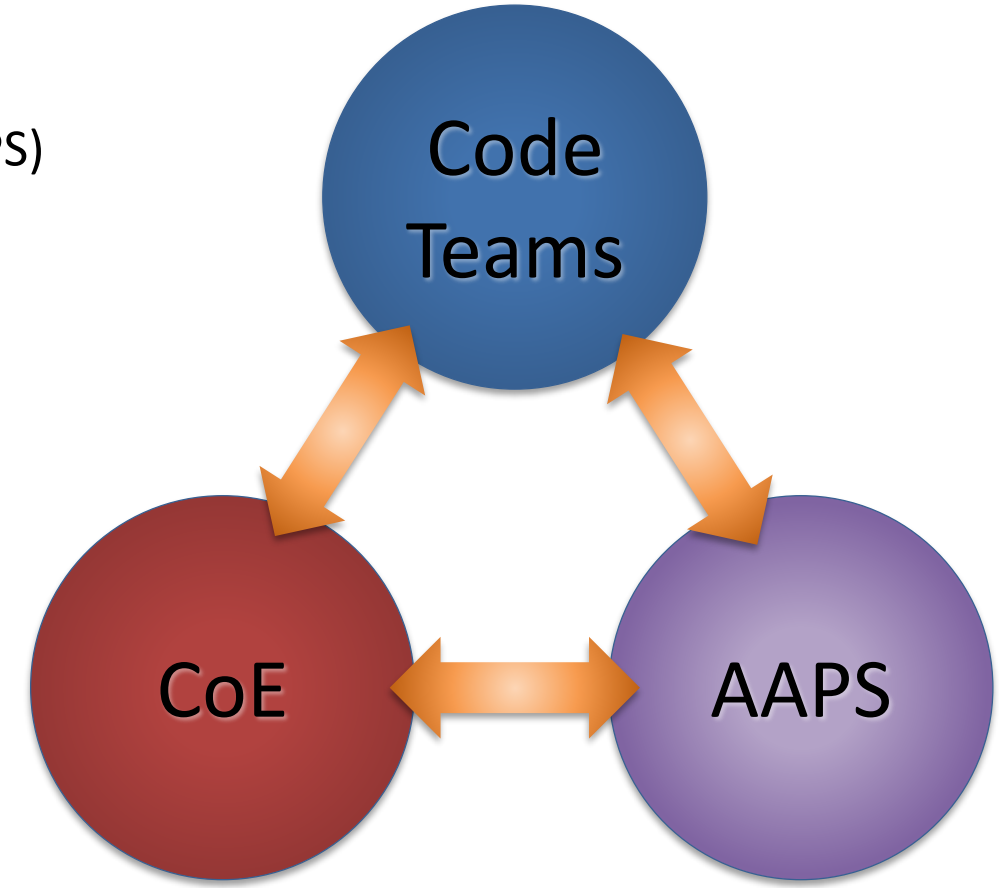
March 19, 2019

LLNL-PRES-769074
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

LLNL has been heavily investing in performance on GPUs

- 17+ code projects/teams/organizations
  - Code development teams
  - Advanced architecture and portability specialists (AAPS)
  - Tool development teams
  - Sierra Center of Excellence (CoE)
  - Livermore Computing
  - Vendors (IBM, Nvidia)
- 78 contributors… and counting!
- 4+ years preparing for Sierra



The expertise, creativity, and collaboration of our teams make technological advances like Sierra possible.

Porting strategies must address more than just performance

- Real codes – real challenges
  - Scale: millions of lines of code in multiple programming languages
  - Continue to provide new capabilities to users
  - Pedigree: maintain connection to prior V&V efforts
  - Libraries: coordinate use of limited memory resources
- Portable performance
  - Our codes must be fast, reliable, and accurate on multiple systems
- Future proof(ish)
  - Heterogeneity is likely here to stay … for a while anyway
  - We can't afford to do this with every new machine
  - Reduce time to performance on new machines
    - Greater utilization of these expensive investments
- Position ourselves for exascale success!

Laptops
Emergency Response Teams

Workstations
DOD & Industry

Commodity
Linux Clusters

Sequoia/Trinity
Advanced Architectures

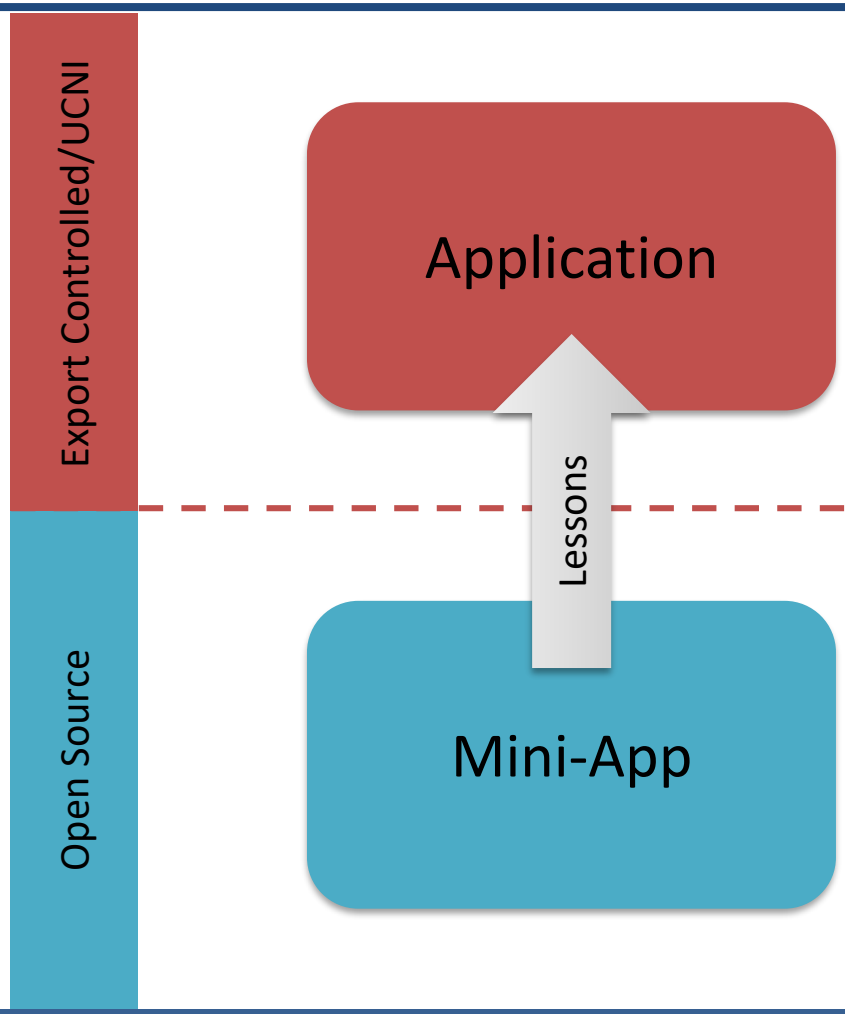
Sierra
GPU Accelerated

Exascale

CORAL-2
El Capitan

This is an opportunity to invest in future performance.

Lightweight mini-apps are used to study algorithmic behavior and facilitate collaboration with vendors and academia



- LLNL production code
  - Million+ lines of code
  - Multiple languages
  - New features added regularly
  - Multiple physical processes interacting
  - Sensitive/proprietary

- Open source research application
  - Focused and lightweight
  - Single physics (few algorithms)
  - Can be shared with vendors and academic collaborators
    - Facilitates performance optimization

Mini-apps allow us to leverage vendor and academia expertise in optimizing our full production codes

A note on measuring performance

CPU and GPU performance is difficult to measure

- All speedup numbers are node-to-node speedup as compared with CTS-1
  - What users will generally experience
- Most of our codes are primarily memory-bandwidth bound on the CPU
- To set expectations, compare relevant effective memory bandwidths of architectures

	CTS-1 (Broadwell)		Sierra EA (2× P8 CPU + 4× P100 GPU)		Sierra (2× P9 CPU + 4× V100 GPU)
DRAM bandwidth per node	130 GB/s	16.9×	2,200 GB/s	1.5×	3,400 GB/s
L2 bandwidth per node	3,870 GB/s	8.0×			
Shared memory bandwidth per node			31,052 GB/s	1.6×	48,320 GB/s

- How does performance scale with relevant memory bandwidth?
  - This is not a perfect measure, but it is a good place to start

Memory bandwidth is a first-order predictor of performance (as opposed to peak FLOPS).

The deterministic transport project is realizing significant performance gains through focused refactor and porting efforts

- Deterministic transport codes
  - Ardra: particle transport
  - Teton: thermal radiative transfer
- Porting strategy
  - Teton
    - OpenMP 4.5
    - CUDA-C
  - Ardra
    - RAJA, CHAI, Umpire

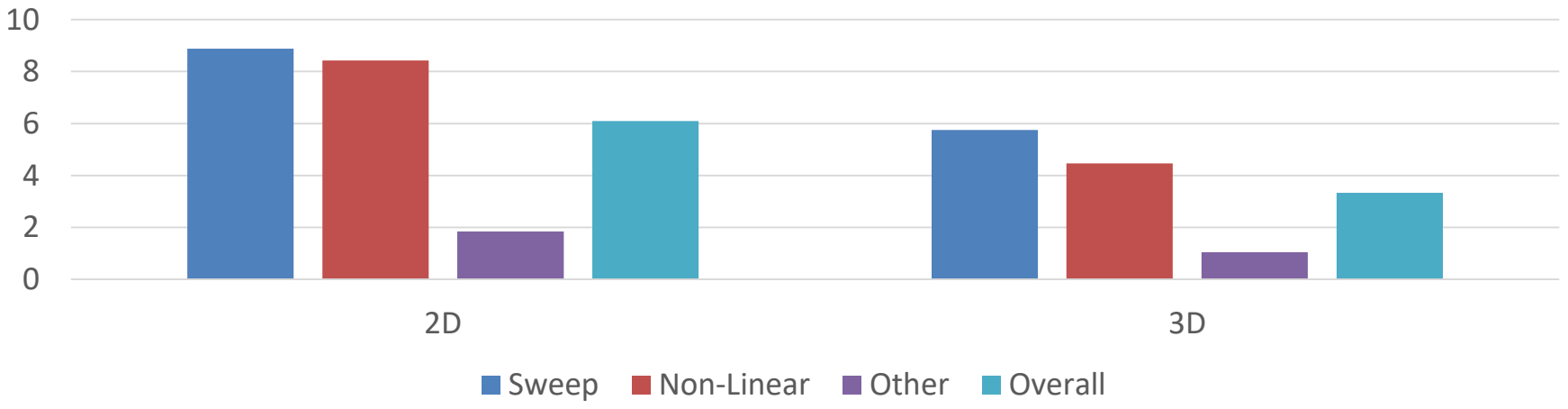
- Enabling performant sweeps on GPUs was a significant challenge that had not previously been demonstrated
  - Memory requirements and algorithmic dependencies create technical challenges on new architectures

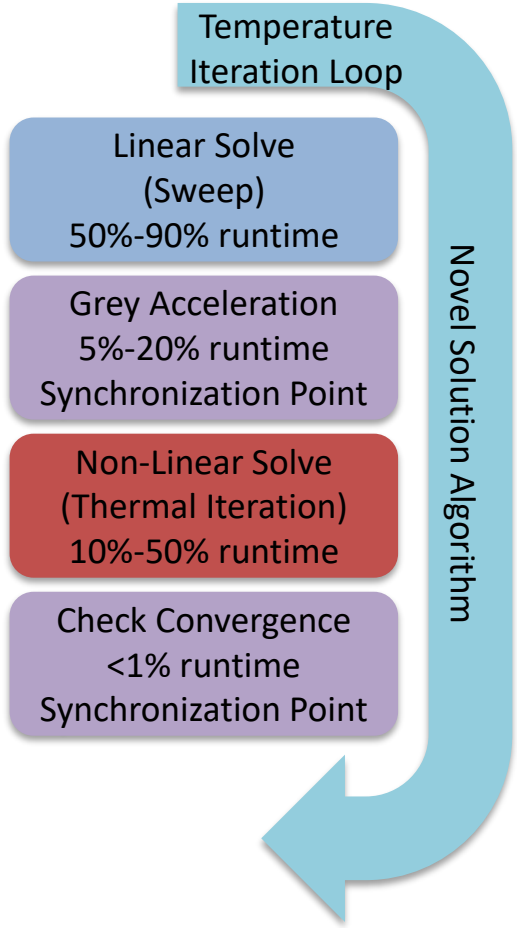
Deterministic transport pushes memory requirements to the limits of the device.

Teton's computational performance is dominated by two kernels: Linear Solve (Sweep) and Non-linear Solve

- We have ported the linear solve (Sweep) to GPUs
  - OpenMP 4.5 and CUDA-C
- We have ported the non-linear solve to GPUs
  - CUDA-C
- Teton is Fortran (cannot use RAJA)
  - Fortran tools/compilers lag those of C/C++

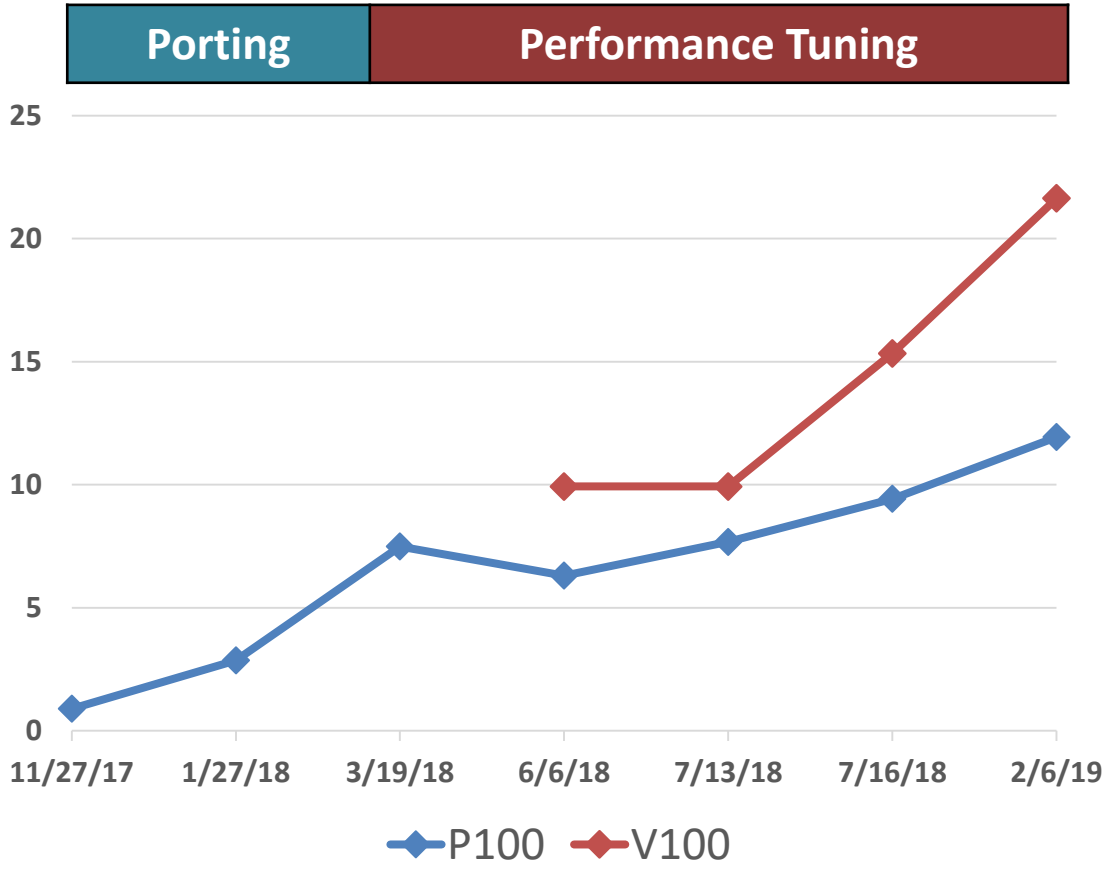
- We accept the risk (for now) of maintaining separate CPU- and GPU-specific versions of a small number of key algorithms
  - Algorithms are tailored to the hardware to maximize performance
- Can we refactor code with a clever abstraction layer and maintain only one version?





We are exploring multiple porting strategies in full production code, including tradeoffs between CUDA-C and OpenMP 4.5.

Speedup is being measured with criticality solve



Research

- Mini-app research
  - Work with Sierra CoE to optimize algorithms
- Develop RAJA nested loops
- Data structure refactor

Porting

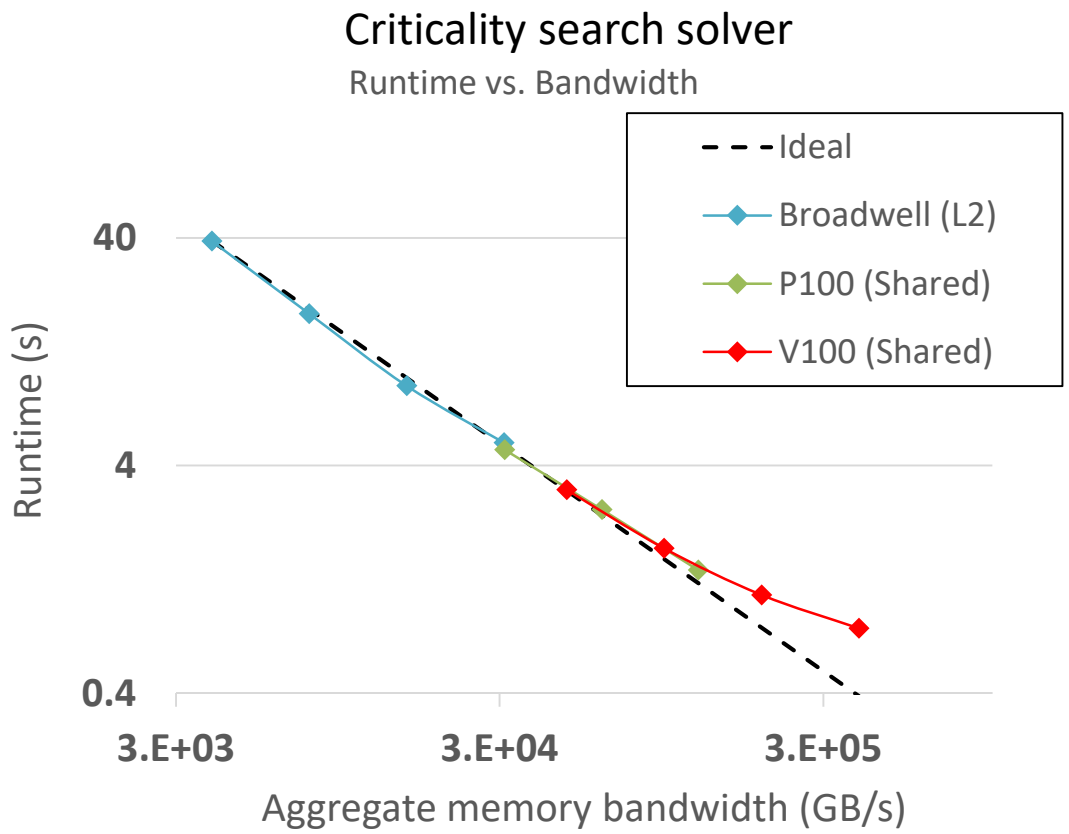
- Transition code to RAJA/CHAI/Umpire
- Performance poor because of significant data motion
- Aiming for correctness, not speed

Performance Tuning

- All kernels running on GPU
- Data stays resident on GPU (except communication)
- Algorithms take advantage of GPU shared memory

Focused and strategic porting of deterministic transport is yielding significant speedups.

Ardra performance tracks closely with cache bandwidth across architectures



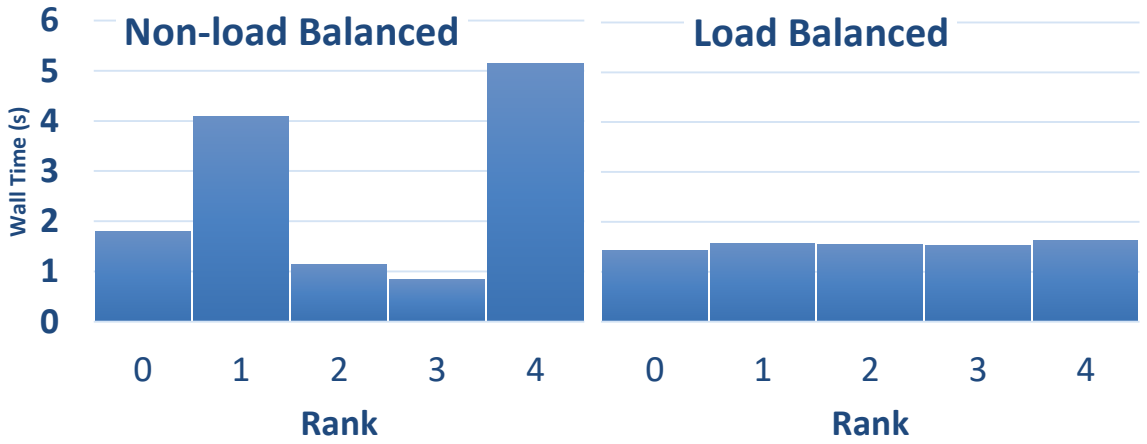
	Resources	Nodes	Runtime (s)	Speedup (×)
CTS-1 (Broadwell)	36 CPU cores	1	38.76	1.0
	72 cores	2	18.57	2.1
	144 cores	4	8.95	4.3
	288 cores	8	5.03	7.7
EA (P8+P100)	4 P100 GPUs	1	4.69	8.3
	8 GPUs	2	2.56	15.1
	16 GPUs	4	1.39	27.8
Sierra (P9+V100)	4 V100 GPUs	1	3.13	12.4
	8 GPUs	2	1.73	22.4
	16 GPUs	4	1.08	35.8
	32 GPUs	8	0.77	50.5

The Mercury particle transport and Imp IMC thermal radiative transfer capabilities have been ported to Sierra

- Particle (Mercury) and thermal photon (Imp) transport consolidated into single code base
  - Built from shared infrastructural source code
  - Facilitated GPU port
- History-based Monte Carlo transport is generally hostile to most advanced architectures
  - Particle tracking loop is thousands of lines of branchy, latency-sensitive code
- GPU porting strategy
  - CUDA "big kernel" history-based particle tracking with CUDA managed memory
  - Exploring RAJA for more typical "loops over cells" code
- Targeting 2-3× speedup on Sierra
  - Based on mini-app results

Dynamic Heterogeneous Load Balancing

- Uses speed information from previous cycle to balance the particle workload among all ranks
- Performance limited by longest running rank
- Early tests show up to 3× speedup



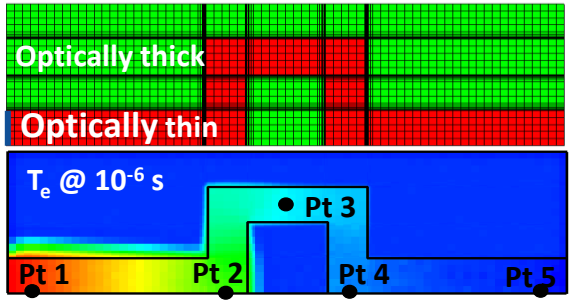
Monte Carlo transport capabilities are entering the performance tuning phase and exploring heterogeneous load balancing.

We are assessing Imp and Mercury performance on Sierra

- Crooked pipe idealized thermal radiative transfer test problem
  - 2× speedup overall
  - Particle tracking showing decent speedup

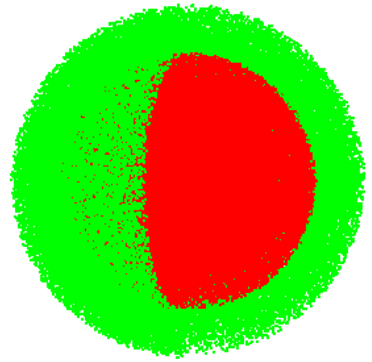
Resources	CPU / GPU	Total Time [minutes]	Particle Time [minutes]	Init/Final Time [minutes]
CTS-1	36 cores	31.67	29.61	2.05
V100+P9	4 GPUs + 36 cores	15.88 (1.99×)	11.70 (2.53×)	4.18 (0.49×)

Source



- Godiva critical sphere surrounded by water, criticality solve
  - 1.1× overall speedup

Resources	CPU / GPU	Total Time [minutes]	Particle Time [minutes]	Init/Final Time [minutes]
CTS-1	36 cores	2.53	2.27	0.26
V100+P9	4 GPUs + 36 cores	2.28 (1.11×)	1.83 (1.24×)	0.45 (0.58×)



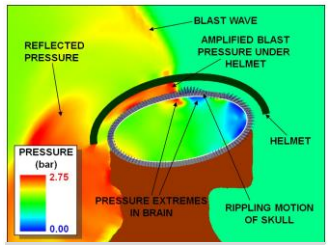
Monte Carlo transport on GPUs is hard, but progress is being made.

* D. E. Cullen, C. J. Clouse, R. Procassini, R. C. Little, "Static and Dynamic Criticality: Are They Different," UCRL-TR-201506 (2003)

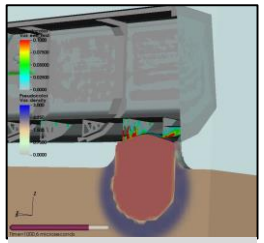
HE Performance, Lethality, Vulnerability and Safety Code

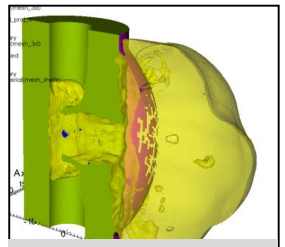
- Required physics capabilities
  - 3D/2D ALE hydrodynamics
  - 3D arbitrarily connected hexahedral mesh
  - High-explosive modeling
  - Material contact
  - Advanced material models

DoD: Munitions and rocket design performance, lethality, vulnerabilities, and safety



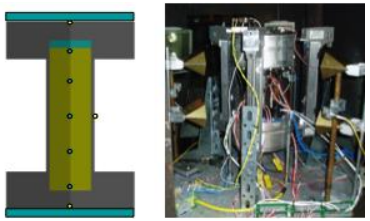
Blast/Impact for TBI


Buried Blast

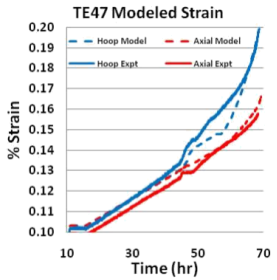

Rocket Motor


Rail Gun

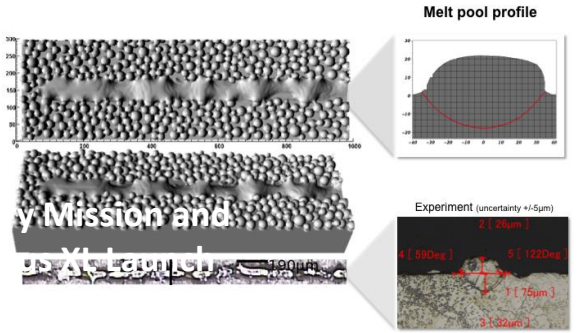
DOE: Stockpile Stewardship, other NNSA programs



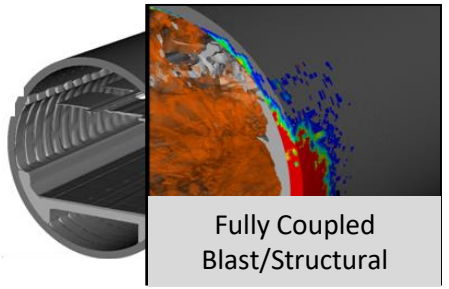
Explosive Cookoff Violence of Reaction

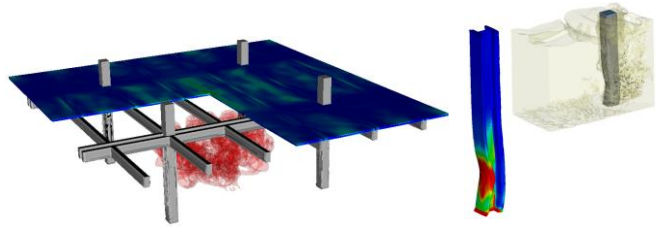


Other: Additive Manufacturing



DHS: Transit and structures vulnerabilities and safeguard designs


Fully Coupled Blast/Structural


Component and System-Level Analysis

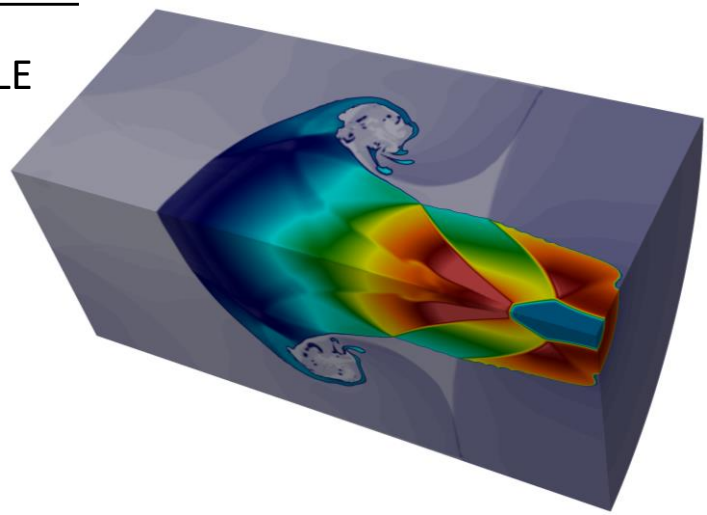
The goal is to turn current month-long complex calculations around in a weekend (10× speedup or more).

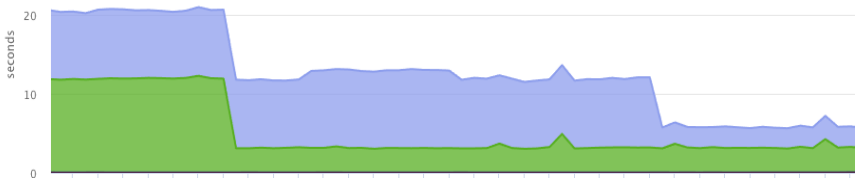
We are making great progress enabling a wide range of physics capabilities

- Main hydro packages have been ported and are being optimized
- Strategy: RAJA/CHAI/Umpire
- Current focus
  - Porting reactive flow models for high fidelity HE modeling
  - Slides for material contact
  - Addressing performance bottlenecks
- Successfully run 3D high-resolution problems on GPUs

Triple-Point Shock Problem

- 3D multi-material ALE hydrodynamics
- 17B zones
- 512 nodes
  - 2,048 GPUs
- Only 12% of Sierra
  - Or roughly Sequoia





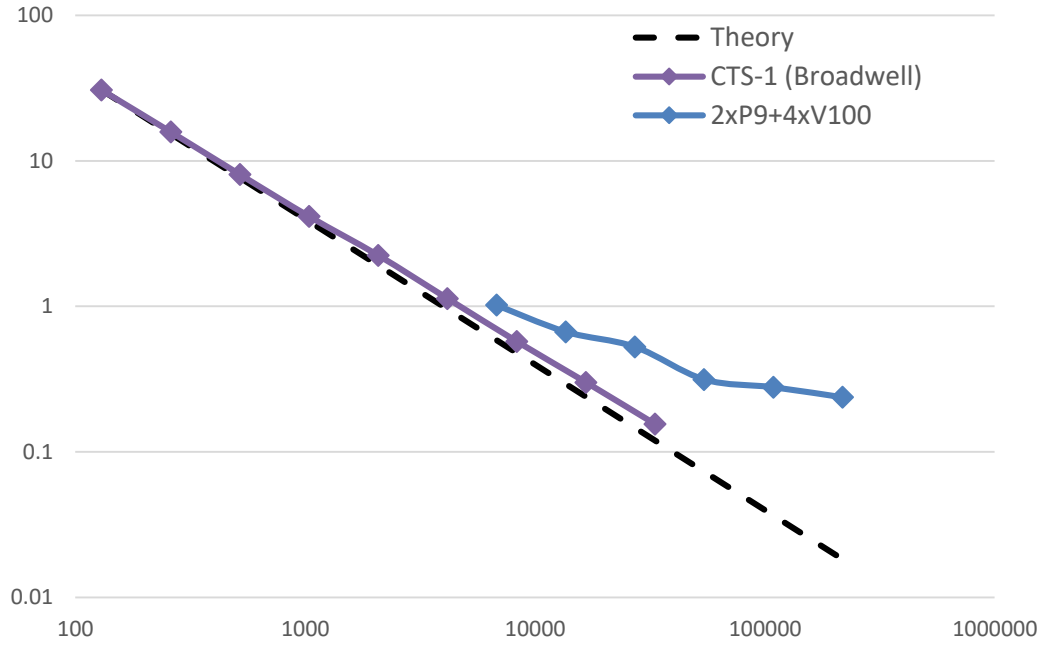
Tracking performance improvements over time

The ability to run problems like this on a small fraction of Sierra makes high-resolution 3D UQ feasible.

A number of optimization opportunities have been identified

	Resources	Nodes	Runtime (hours)
CTS-1	36 cores	1	30.7
	72 cores	2	15.8
	144 cores	4	8.1
	288 cores	8	4.2
	576 cores	16	2.2
	1152 cores	32	1.1
	2304 cores	64	0.6
	4608 cores	128	0.3
	9216 cores	256	0.2
Sierra	8 GPUs	2	1.1
	16 GPUs	4	0.7
	32 GPUs	8	0.5
	64 GPUs	16	0.3
	128 GPUs	32	0.3
	256 GPUs	64	0.2

Node-Node Speedup 14.5×

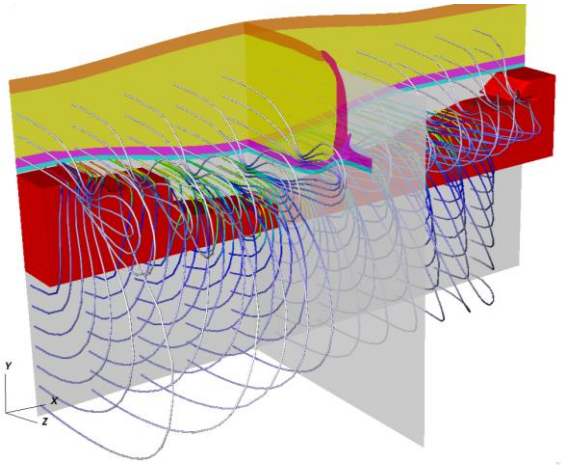


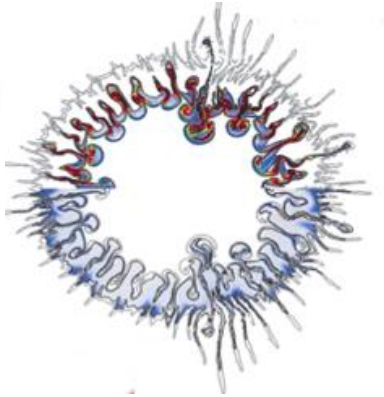
- Up to 8M zones per GPU
- Identified major bottlenecks
  - Kernel launch overhead
  - GPU register pressure

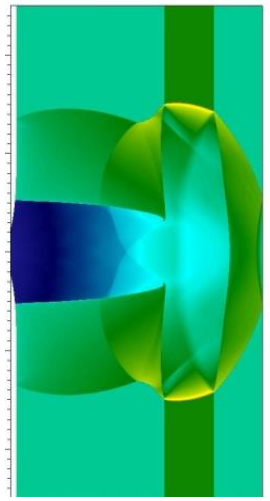
Performance bottlenecks must be overcome to achieve memory bandwidth scaling.

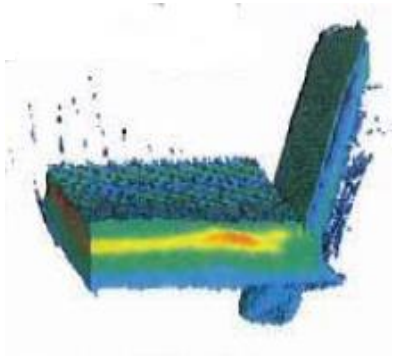
Ares: NIF debris, pulsed power, ICF, and HE simulation code

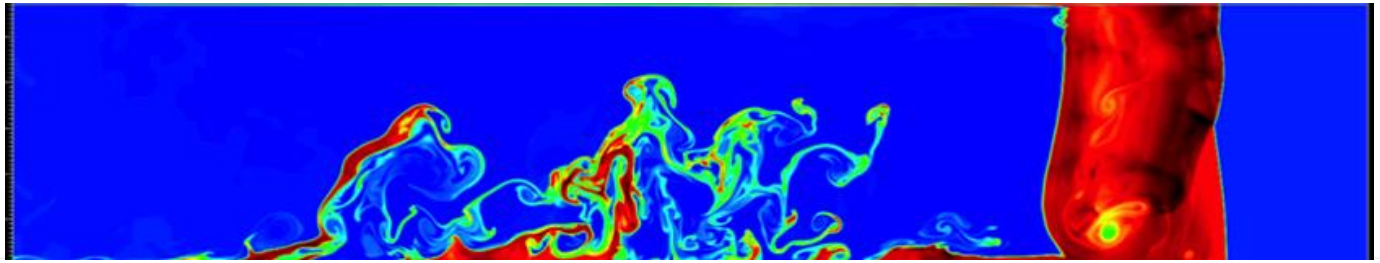
- Physics capabilities:
  - ALE-AMR hydrodynamics
  - High-order Eulerian hydrodynamics
  - Elastic-plastic flow
  - 3T plasma physics
  - High-explosive (HE) modeling
  - Diffusion and deterministic thermal radiative transfer
  - Multiphase particle flow
  - Laser ray-tracing
  - MHD
  - Dynamic mixing
  - Non-LTE opacities







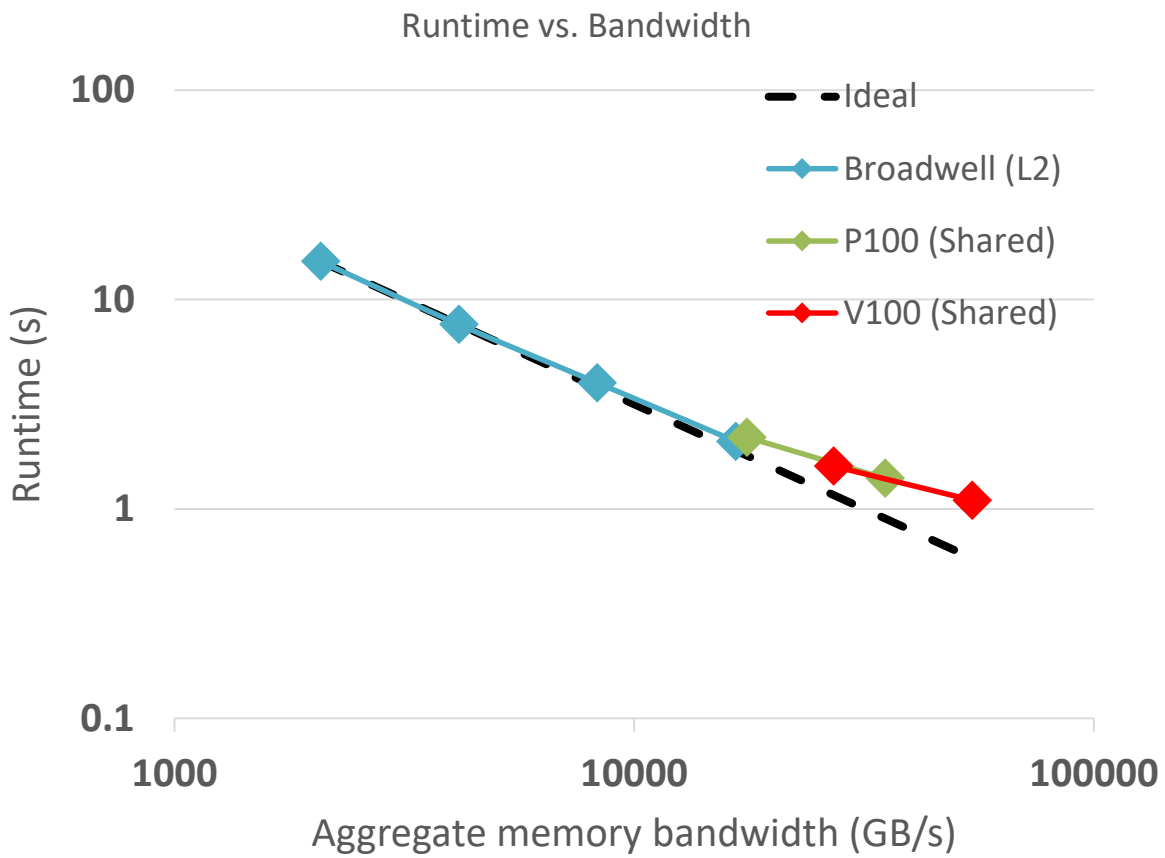




Applications:
- Inertial Confinement Fusion (ICF)
- Pulsed power
- National Ignition Facility debris
- HE experiments

GPUs show great promise for increasing throughput for Ares applications

	Resources	Nodes	Runtime (hours)	Speedup (×)
CTS-1 Broadwell	576 cores	16	15.2	1.00
	1152 cores	32	7.6	2.00
	2304 cores	64	4.0	3.80
	4608 cores	128	2.1	7.24
Sierra EA P8 + P100	32 P100 GPUs	8	2.2	6.91
	64 P100 GPUs	16	1.4	10.86
Sierra P9 + V100	32 V100 GPUs	8	1.6	9.5
	64 V100 GPUs	16	1.1	13.82



14× speedup has been achieved, but further optimizations are being explored.

Recent work on Sierra demonstrates that we will be able to study mix phenomena in ICF at unprecedented resolutions

Resources (V100s)	Nodes	Zones	Runtime (hours)
32	8	191M	1.6
256	64	1.52B	3.5
2048	512	12.2B	7.8
16384	4096	97.8B	13.05

- Performance scales with memory bandwidth
- Opportunities remain for further optimization

Rayleigh-Taylor Mixing Layer in a Convergent Geometry

- 4π
- ALE hydrodynamics
- Dynamic species
- Idealized ICF capsule

Heroic calculations like these can be turned around in a matter of days.

Kull: HED experiments simulation code

- Computational runtime is dominated by transport in ICF calculations
  - Thermal radiative transfer can account for an overwhelming majority of the runtime
- Kull strategy
  - Refactor code for compatibility with RAJA (in progress)
  - Initially, provide an environment that facilitates maximizing transport performance
  - Enable flexibility for choices made by transport algorithms
    - Multiple levels of parallelism
    - Provide an ecosystem that supports C++/Fortran/OpenMP/CUDA-C/CUDA-Fortran all in one code

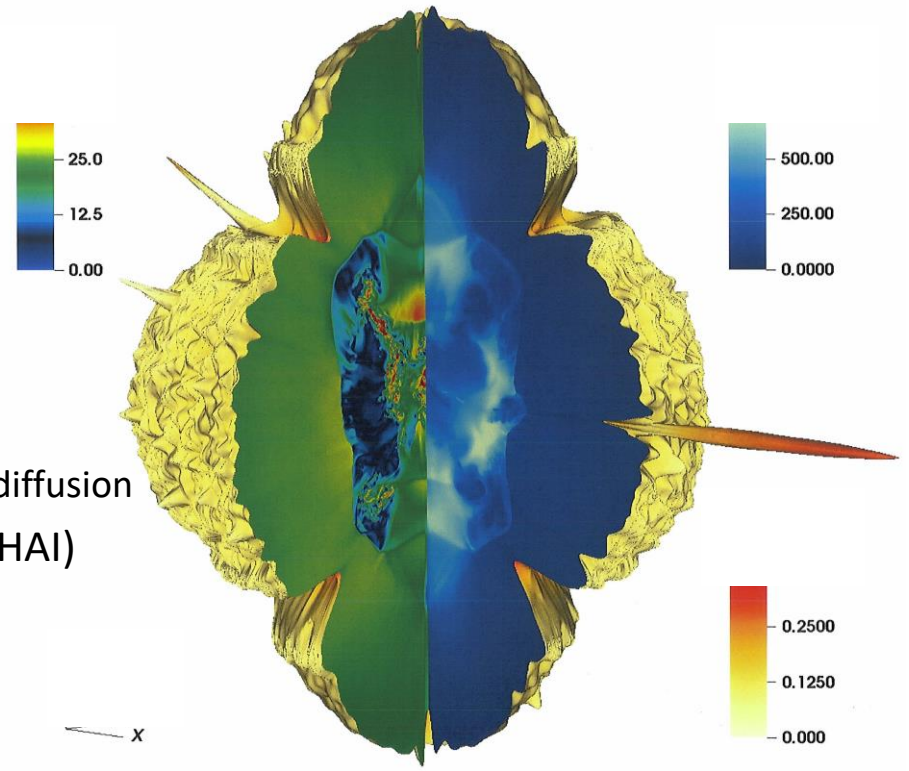
	Total Runtime/Speedup	Teton Sweep	Teton NL Solve	Teton GTA/Init/Finalize
CPU (CTS-1)	52.45 / 1.0x	21.9	10.35	9.99
GPU Sweep + NL Solver	26.67 / **1.97×**	2.75×	2.03×	0.73×

*Radiating Sphere Test Problem

Early performance gains will come from thermal radiative transfer gains, with hydro gains expected as refactor progresses.

HYDRA physics packages being ported to run on Sierra GPUs using a staged approach

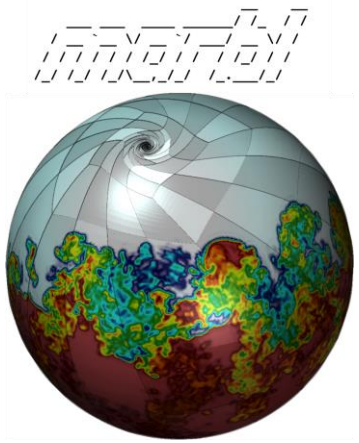
- Initial focus is on porting the most expensive physics packages to GPUs
  - Implicit Monte Carlo Photonics (IMC)
    - Evaluated porting options in mini-app
    - IMC package now running on CPUs and GPUs simultaneously
  - Non-Local Thermodynamic Equilibrium (NLTE)
    - Currently evaluating mini-app performance on GPUs
  - MHD package has been modified to support GPU parallelism
  - GPU parallel version of *hypre* solvers undergoing testing
    - Employed in multigroup diffusion, thermal transport, and charged particle diffusion
  - Exploring multiple approaches (OpenMP 4.5, CUDA, RAJA/Umpire/CHAI)
- Sierra will enable…
  - Higher throughput of high-resolution 3D simulations
  - More accurate NLTE models (100× increase in configurations)

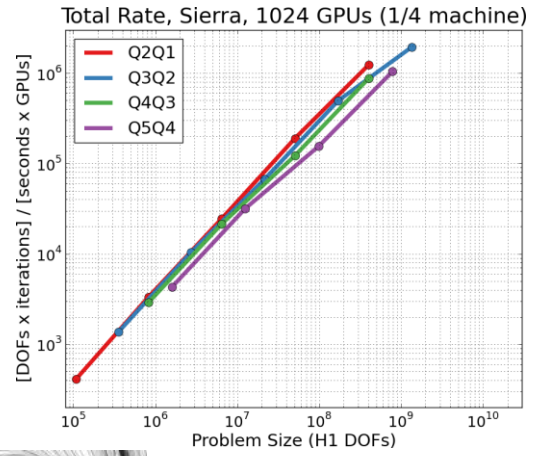


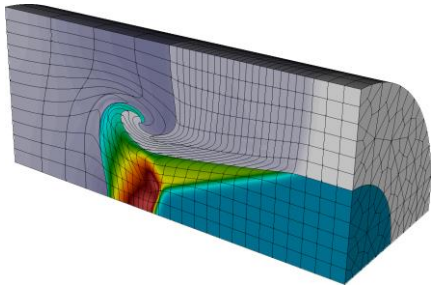
Staged porting allows for focused performance tuning on each physics package before putting it all together.

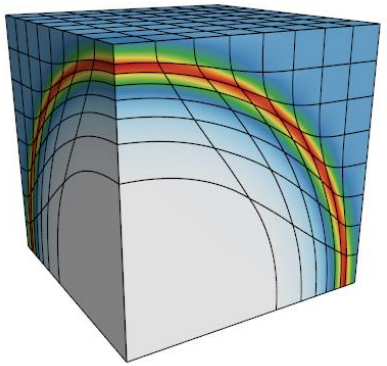
MARBL, our next-generation high-order ICF code, shows great promise on Sierra

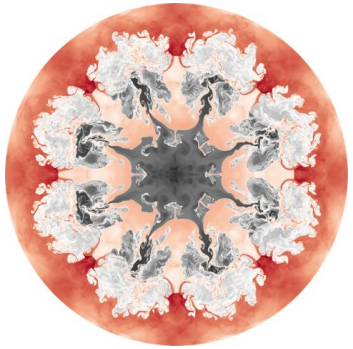
- MARBL has two hydro modules
  - BLAST: High-order unstructured ALE
    - Lessons learned from high-order Lagrangian mini-app currently being transferred to BLAST
  - Miranda: High-order structured Eulerian
    - Mini-app helping to understand best practices for using OpenMP 4.5 in Fortran code
    - Parallelizing Fortran array operations over thread teams requires addition to OpenMP standard (pending)









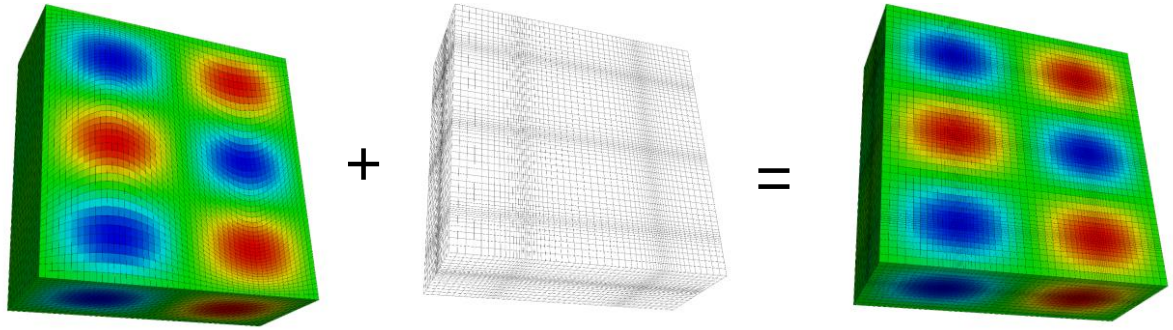


High-order methods in MARBL look to be particularly well suited for performance on Sierra.

We are exploring geometric and sampling intersection evaluation methods for solution mapping on the GPU

- Geometric (standard Overlink)
  - Based on Material Interface Reconstruction (MIR)
  - Near machine accuracy
  - Initial studies showed not well suited to GPUs
- Geometric Lite (suited to GPU)
  - Mixed zones are homogenized (slight loss of accuracy)
- Sampling
  - 8000 samples per zone = 0.25% statistical error
  - Backward map is slower for large numbers of mixed zones

Transfers mesh-based data from an original donor mesh to Cartesian target mesh

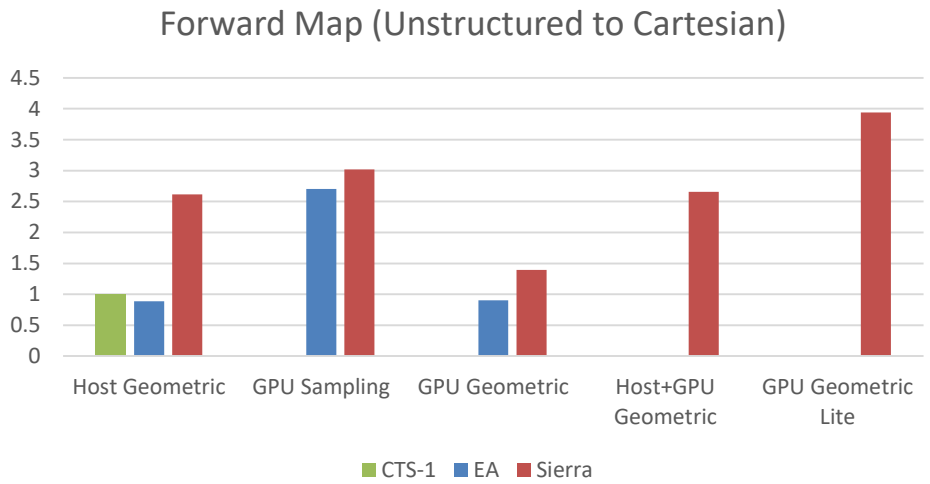


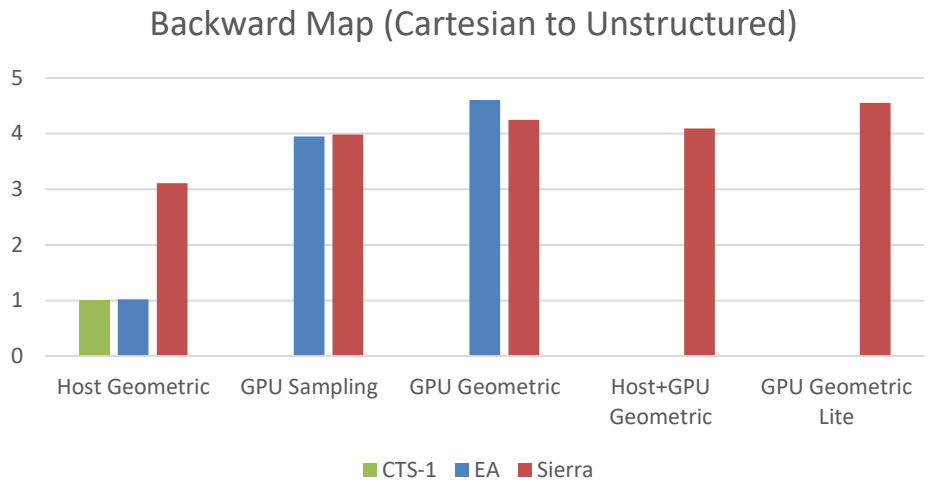
Input:
Variable on
"Donor" Mesh

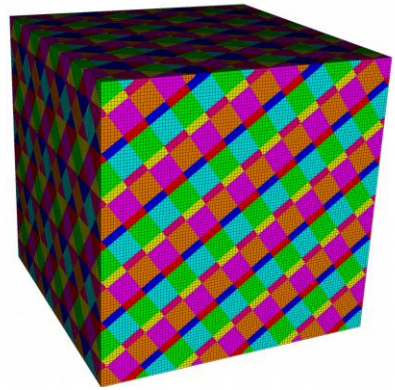
Input:
Cartesian
"Target" Mesh

Output:
"Donor" Variable
on "Target" Mesh

Geometric methods are proving to be superior to sampling methods on Sierra







- On EA, GPU Sampling was fastest
  - But suffered from non-trivial error
- Host+GPU Geometric becomes competitive
  - Extremely low error
- GPU Geometric Lite was fastest on Sierra
  - Small error at mixed zones

- On EA, GPU Geometric was fastest
  - Extremely low error
- On Sierra, GPU Geometric Lite was fastest
  - Small error at mixed zones
- All methods benefit significantly from MPI improvements on Sierra (vs. CTS-1)
- Sampling was no faster than Geometric

Algorithm choice indicated by EA machine turned out NOT to be the best method on Sierra.

Team structure and plan for porting is important

- Upfront research and scoping with mini-apps is invaluable to developing a deep understanding of algorithmic behavior and selecting an appropriate porting strategy
- Challenges continue in production codes and multi-physics contexts
- Having computer scientists co-located between tools teams and applications teams has been vital
  - Co-development of tools
  - Feature requirements and feature development often by the same personnel
  - Facilitates implementation and adoption
  - Easy access to RAJA/CHAI/Umpire expertise

Rapid dissemination of experience and best practices between teams have been essential.

Abstraction layers can provide excellent performance and portability

LLNL Abstractions – RAJA/CHAI/Umpire

- RAJA provides excellent portability with little performance loss
  - Performance gap continues to close with help from vendors
- CHAI provides a hardware-agnostic automated data transfer solution
- Umpire across host/library codes allows for cohesive memory management with pools
- Architecture-specific implementations are suitable for complex kernels or if a small number of kernels dominate runtime

OpenMP 4.5+

- Useful abstraction tool for Fortran codes
- OpenMP support in Nvidia tools is problematic
  - Hampers debugging and performance profiling
- No way to catch runtime errors with OpenMP
  - Makes debugging painful

Our assumption that there would be a significant tradeoff between performance and portability was wrong.

Memory management and migration is a significant performance factor

- CHAI
  - Hardware-agnostic automatic data migration
  - Good performance but large upfront investment
  - Compile time correctness checking
- Unified Memory (UM)
  - Essentially no upfront cost
  - Automatic memory migration is almost always slow
  - Automatic eviction when GPU runs out of memory
  - UM allows memory management to be treated as a performance optimization
    - Facilitating porting
- Umpire
  - Hardware-agnostic memory management abstraction
  - Provides memory pools
  - Memory introspection for better decision making
    - Where is this pointer?
    - How big is the allocation?
    - What allocator is used?
    - How much memory is being used on this resource?

You will have to put in work, at one end or the other, to make memory management performant.

GPUs have performance overheads that we don't see on CPUs that must be managed

Performance

- Kernel launch overhead
  - Can be hidden using asynchronous kernel launches
- Data transfer between memory spaces
  - Needs to be avoided or hidden behind other kernels
- Memory allocation is significantly more expensive on the GPU
  - Necessitates the usage of memory pools

Library Coordination

- Different porting strategies/timelines
- Un-ported libraries can result in costly CPU/GPU data transfer
- The GPU can be considered a communal resource with multiple competing stakeholders
  - Memory pools can help

Tools

- Debugging: CUDA memcheck, CUDA GDB, Totalview, good ol' print statements
- Performance: NVProf, Archer: Thread Sanitizer
- Common source (via abstractions) provides access to wider range of tools

Achieving performance requires a deeper understanding of our codes/algorithms, which will yield dividends in the future.

Summary

- Our code teams have undertaken careful and detailed evaluations of porting and execution strategies to optimize our codes for Sierra and maintain portable performance.
- A wide range of physics capabilities have been ported to GPUs and are either in the process of exploring optimization strategies or have achieved game-changing speedup.
- Maintaining speedup in complex calculations is challenging.
- We are exploring multiple porting strategies where appropriate.
- Abstraction layers provide excellent performance and portability with minimal tradeoff between the two.

We are well positioned to use Sierra for high-fidelity 3D studies in what were previously considered “heroic” calculations

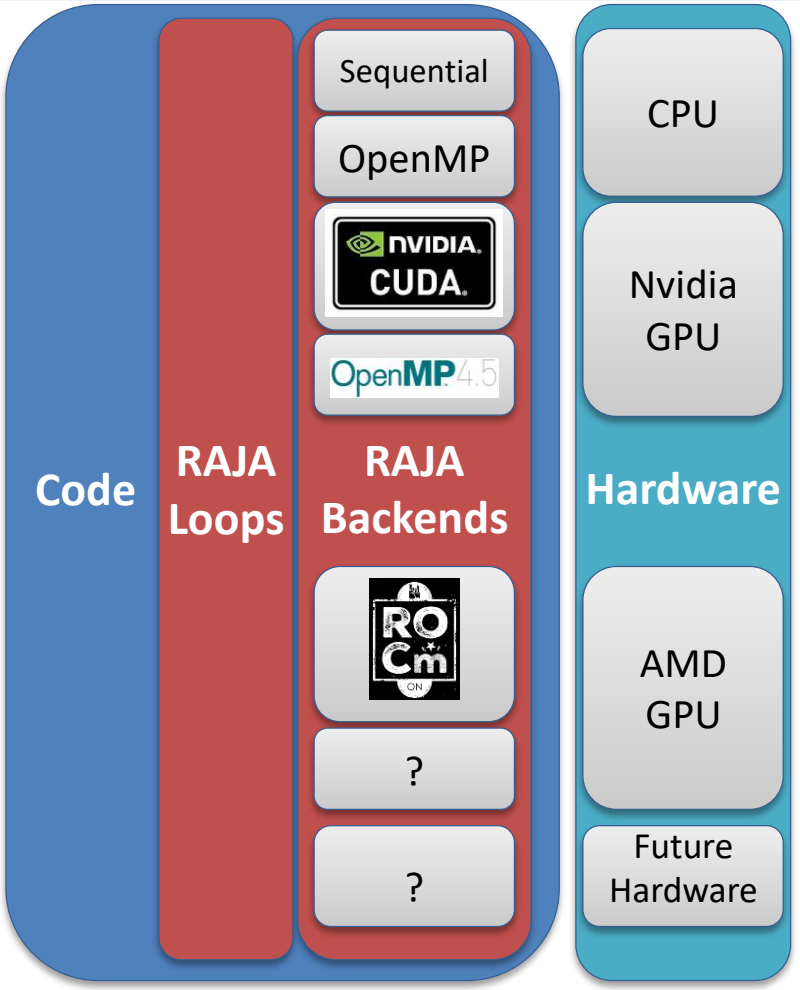
- Increased physics and geometric fidelity
  - Increased resolution
  - Fewer compromises on physics models
- Pose questions on the scale of hours instead of days or weeks (or even months)
  - This fundamentally changes what questions you ask and how you ask them
- Improved turnaround of large 3D calculations
  - 3D Uncertainty Quantification becomes feasible
- Hero calculations become practical
  - Extremely high-resolution calculations in 2 days instead of 45

What will “heroic” look like on Sierra ?

Questions ?

RAJA provides portable performance and was co-designed with our application requirements

- A typical code has O(10k) execution loops, but only 0(10) loop types
  - Provides optimized backend for common loop types
- Portability and maintainability
  - Algorithms don't change with backend
  - Leverage vendor optimizations in all codes
- Future proofing
  - RAJA PerfSuite and Kripke are part of the CORAL-2 benchmark
    - Reduce "time-to-performance" on future hardware
- RAJA is not a universal solution
  - Does not support Fortran
  - Requires C++ll and Lambdas
  - Does not yet concisely address some of our application use cases



Backend can change as technologies wax and wane without modifying algorithms

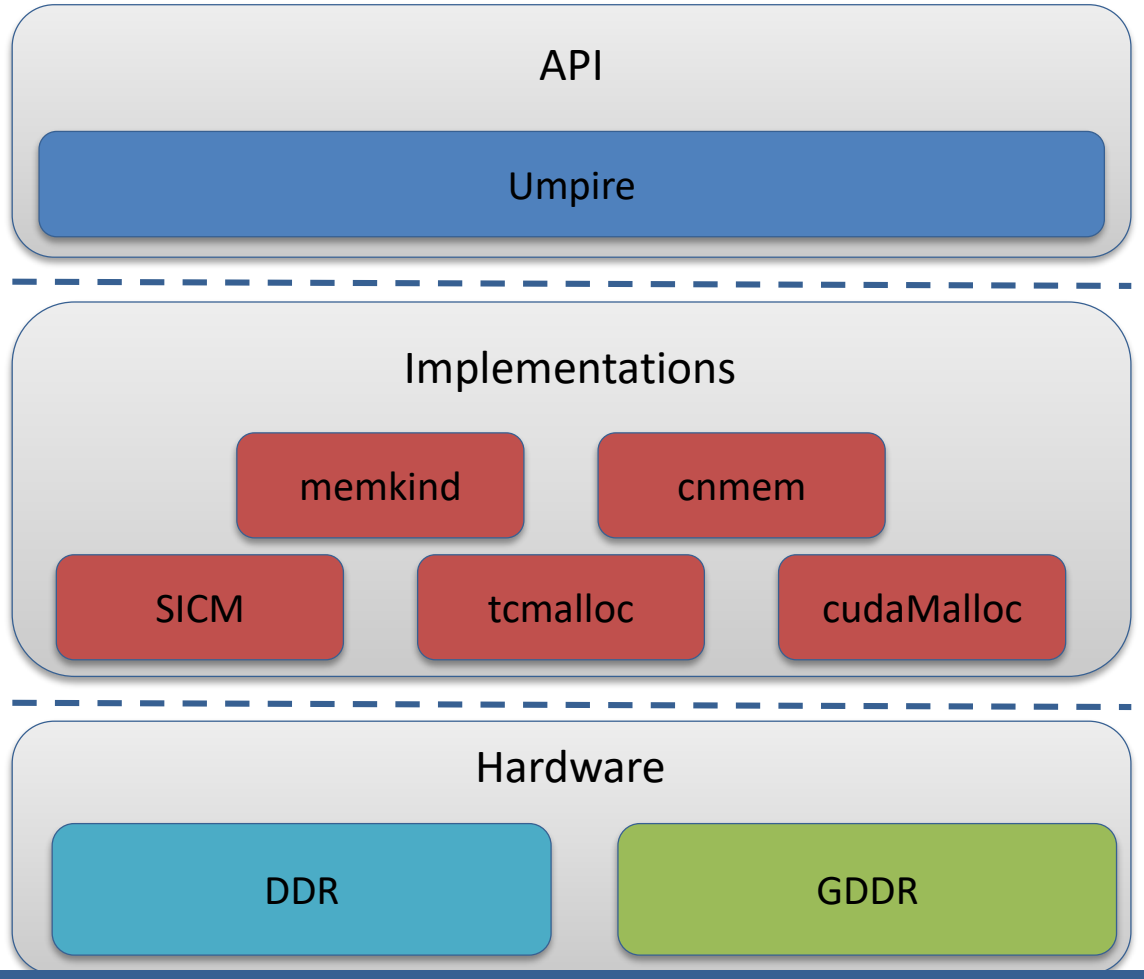
CHAI automatically handles runtime data transfers

- CHAI (Copy Hiding Abstraction Interface)
  - Smart pointer type detects execution location and ensures data locality
- Simplifies porting
  - No explicit memory copying needed
  - Errors caught at compile-time
  - Umpire backend
- Portability and future proofing
  - Portable to machines that lack UM
- Disadvantages
  - Additional changes necessary relative to UM
  - Eviction policies and/or asynchronous data transfers needed for additional performance optimizations (in progress)

Provides robust portability and performance but requires additional initial porting investment relative to UM.

Umpire provides a unified memory management API

- Portability, backend based
- Easily manage memory throughout complex memory hierarchy
  - Allocate/deallocate/copy /move
- Memory pools
  - More efficient allocation/deallocation of memory
  - Facilitates sharing memory pool between code components
    - More efficient use of memory (larger problems)
- Memory introspection for better decision making
  - Where is this pointer?
  - How big is the allocation?
  - What allocator was used?
  - How much memory is being used on this resource?



Provides portable memory management and convenient memory pools

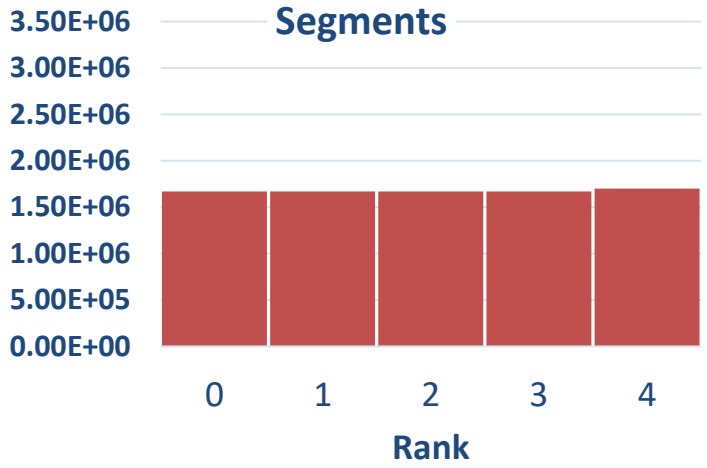
Mini-app research has been key to the planning and design of Armus and the success of Ardra on GPUs

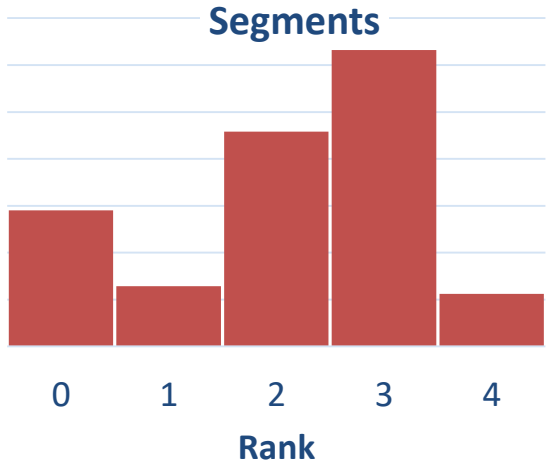
Initial Investigations -> Armus Framework	Armus Framework -> 1st GPU Run	1st GPU Run -> 15x Speedup
21 Months	8 Months	13 Months
Research Mini-app research, initial Armus development, RAJA nested loops, early Ardra refactor	**Porting** Adopt Armus data structures, transition to RAJA, first GPU run	**Performance Tuning** Performance analysis, tuning, use GPU shared memory
• Developed Kripke mini-app to explore data structures and programming models • Worked with CORAL CoE to develop CUDA version of Kripke • Started development of nested loop abstractions in RAJA • Developed requirements for a deterministic transport framework, and created Armus • Started refactoring Ardra to accommodate GPU compatible data structures	• Focus porting activities on 3D static criticality solver • Ported code to Arm us data structures • Transitioned code to RAJA • Continued development of RAJA based on issues encountered in Kripke and Ardra • First GPU run "worked" but had significant robustness issues	• Converted vector kernels to use CUDA • Ported remaining kernels to RAJA • Fixed correctness and robustness issues • Started performance analysis and tuning of major kernels • Started to take advantage of GPU shared memory

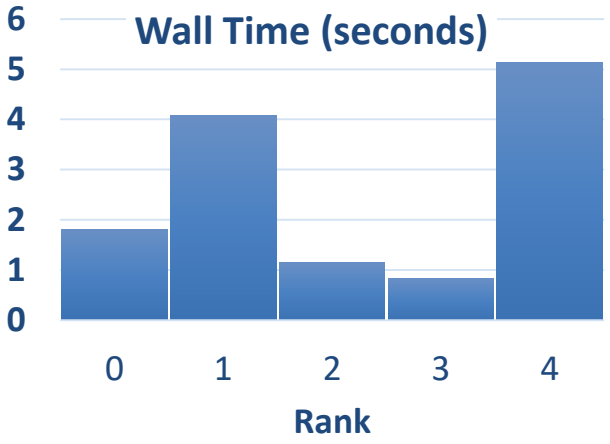
Ardra took an ambitious multi-pronged approach to investing in current and future performance, yielding significant speedup.

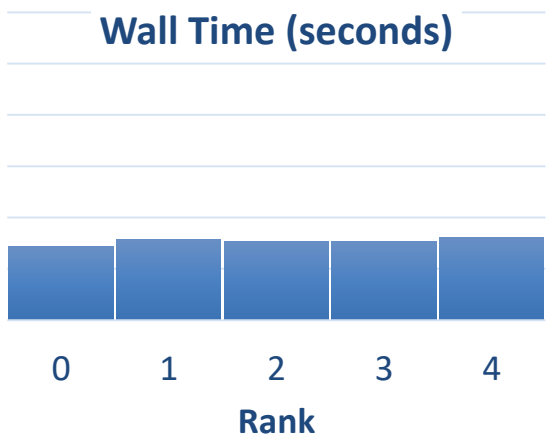
Monte Carlo Transport project has implemented heterogeneous CPU/GPU load balancing

- Mercury/Imp have heterogeneous CPU/GPU load balancing
  - Assumes MPI only for now
  - libQuo or thread-based balancing can be explored
- Uses speed information from the previous cycle to balance the particle workload
- Performance limited by longest running rank
- 3.2X Speedup
  - 1 zone thermal emission test problem



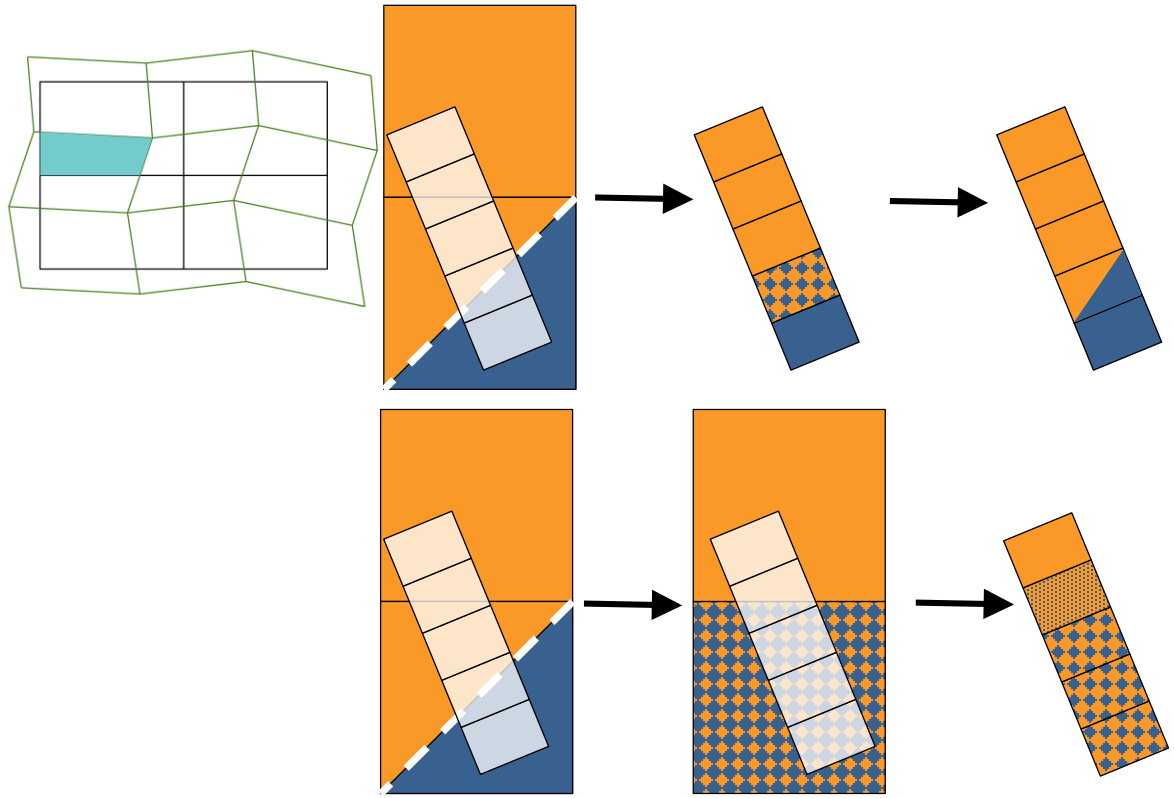






We are exploring geometric and sampling intersection Evaluation methods for solution mapping on the GPU

- Geometric (standard Overlink)
  - Based on Material Interface Reconstruction (MIR)
  - Near machine accuracy
  - Initial studies showed not well suited to GPUs*
- Geometric Lite (Suited to GPU)
  - Mixed zones are homogenized (slight loss of accuracy)
- Sampling
  - 8000 samples per zone= 0.25% statistical error
  - Backward map is slower for large numbers of mixed zones

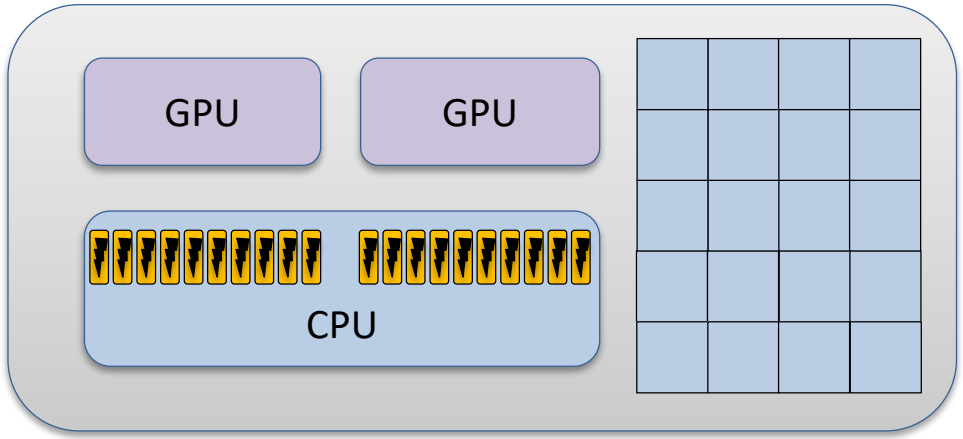


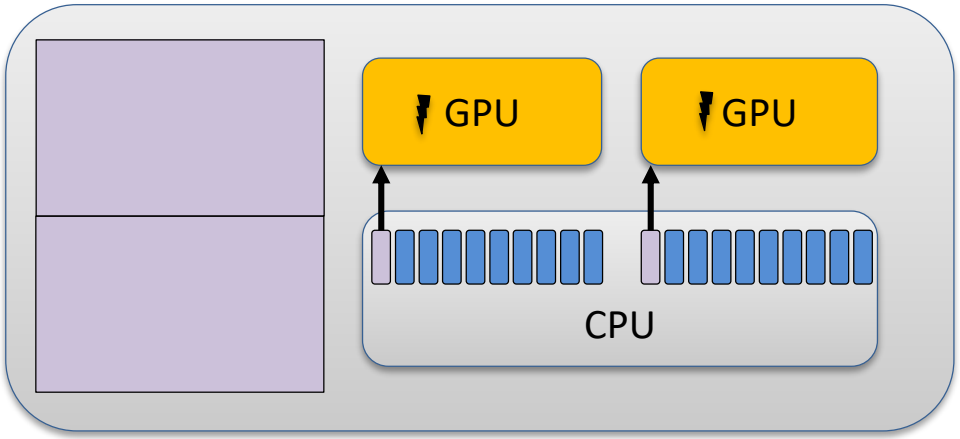
There are a lot of questions around how to best utilize the resources at our disposal

- What modes of execution are best for performance?
  - CPU vs. GPU
  - How many MPI processes
- What about multiphysics?
  - If some phases use one MPI process per GPU, can we productively use remaining CPU cores?
  - If some phases use one MPI process per CPU core, can we use multiple MPI process per GPU for the accelerated phases?

We are beginning to investigate some of these questions, but there are many opportunities to explore.

Accommodating different modes within a single simulation



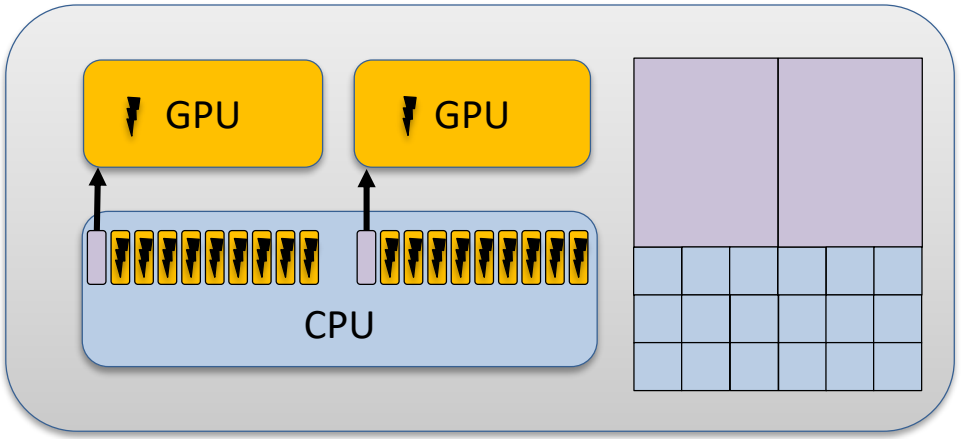


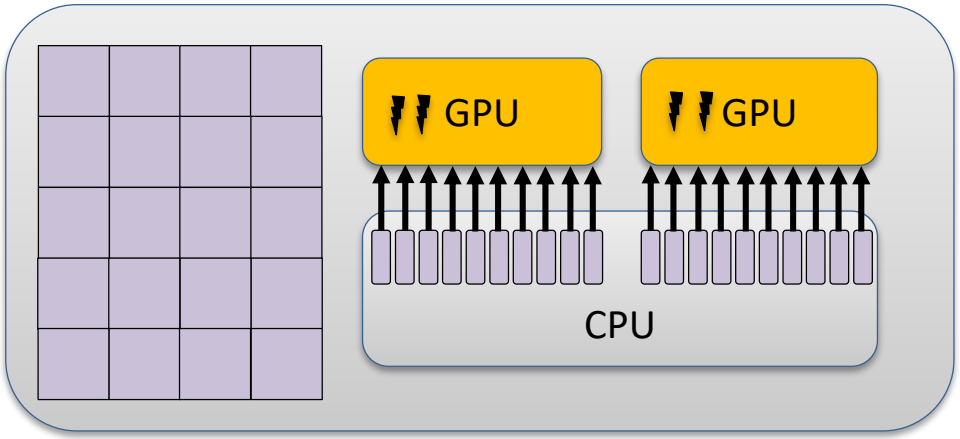
- Perform initialization on all CPUs
- Re-decompose (costly)
- Compute on GPUs

- Example: 196M zone problem on 8 nodes of EA system
  - 2.58x speedup for generation including redistribution cost
  - Saved an hour of runtime (~13% total speedup)

Modest speedup for generation phase of problem. This gets even better when oversubscribing the GPU.

Heterogeneous execution or oversubscribing the GPU may yield valuable performance gains





- Divide work via uneven domain decomposition
  - Very difficult to get the load balancing right
- Proof of concept implemented in Ares
  - RAJA provides same source code for CPU and GPU
- 10% performance improvement over GPU only

- More CPU cores leads to better CPU memory bandwidth utilization
- More MPI processes = more communication
- Multi-Process Service (MPS) allows kernels launched from different MPI processes to be processed concurrently on the same GPU
  - Can result in better utilization of SMs

Oversubscribing the GPU may prove beneficial if improvements in load balance outweigh additional MPI cost.