Migrating to Heterogenous Computing

Lessons Learned in the Sierra & El Capitan Centers of Excellence

David Richards

October 13, 2021

LLNL-PRES-827809

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

Acknowledgements - This talk builds on the work of many

Thank you to co-authors, code teams, support staff, COE Vendors, and everyone else who has helped to make Sierra a success

Johann Dahm
Aaron Black
Adam Bertsch
Leopold Grinberg
Ian Karlin
Sara Kokkila-Schumacher
Edgar Leon
Rob Neely
Ramesh Pankajakshan
Olga Pearce
Brian Ryujin
Jason Burmark

Brian Pudliner
Adam Kunen
David Dawson
Rich Hornung
David Beckingsale
Peter Robinson
Tom Scogland
Holger Jones
David Poliakoff
Jim Glosli
Bert Still
Katie Lewis

Matt Cordery
David Appelhans
Steve Rennich
Max Katz
Noah Reddell
Nick Malaya
Judy Hill

And Many Others…

Sierra Center of Excellence: Lessons learned

J. P. Dahm
D. F. Richards
A. Black
A. D. Bertsch
L. Grinberg
I. Karlin
S. Kokkila-Schumacher
E. A. León
J. R. Neely
R. Pankajakshan
O. Pearce

The introduction of heterogeneous computing via GPUs from the Sierra architecture represented a significant shift in direction for computational science at Lawrence Livermore National Laboratory (LLNL), and therefore required significant preparation. Over the last five years, the Sierra Center of Excellence (CoE) has brought employees with specific expertise from IBM and NVIDIA together with LLNL in a concentrated effort to prepare applications, system software, and tools for the Sierra supercomputer. This article shares the process we applied for the CoE and documents lessons learned during the collaboration, with the hope that others will be able to learn from both our success and intermediate setbacks. We describe what we have found to work for the management of such a collaboration and best practices for algorithms and source code, system configuration and software stack, tools, and application performance.

1 Introduction

VOLUME 64, NUMBER 3/4, MAY/JULY 2020

IBM. Journal of Research and Development

Including IBM Systems Journal

Summit and Sierra Supercomputers

A Center of Excellence is a close partnership between the NNSA and system vendors

- Establish joint work plans, information sharing, and collaboration mechanisms
- Dedicated vendor staff work alongside lab code teams
  - Some staff assigned to work at lab sites
- Labs provide access to our codes
  - Including classified codes for those with security clearances
- Vendors provide non-public information and early access to hardware and software

Sierra

IBM

NVIDIA

El Capitan

Hewlett Packard Enterprise

AMD

NNSA National Nuclear Security Administration

Lawrence Livermore National Laboratory

Sandia National Laboratories

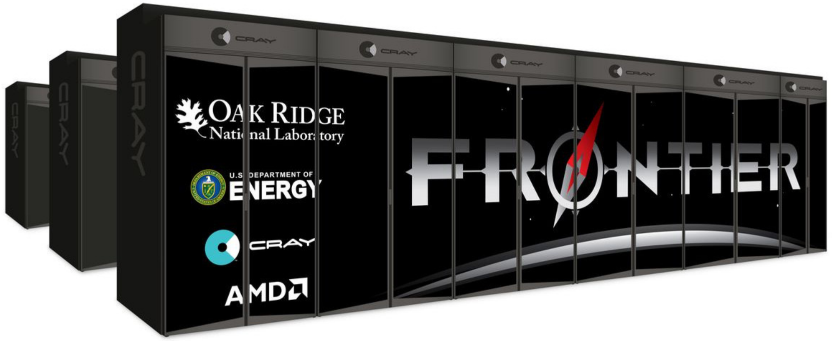
Los Alamos NATIONAL LABORATORY EST. 1943

Forming a Center of Excellence has become a recognized best practice for large DOE system procurements

DOE leadership systems clearly show we have now entered the heterogeneous era

Perlmutter NERSC, 2020
AMD CPU, Nvidia Tesla GPU

Frontier ORNL, 2021
AMD CPU, AMD GPU, 1.5 ExaFlop



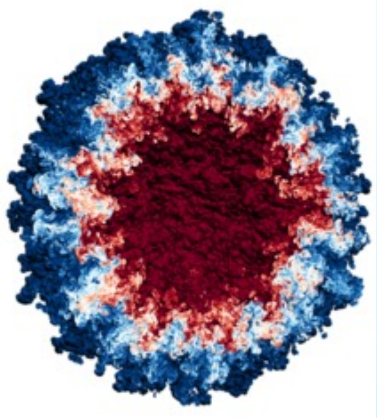
Aurora Argonne, 2022
Intel CPU, Intel Xe GPU, > 1 ExaFlop

El Capitan LLNL, 2023
AMD CPU, AMD GPU, > 1.5 ExaFlop

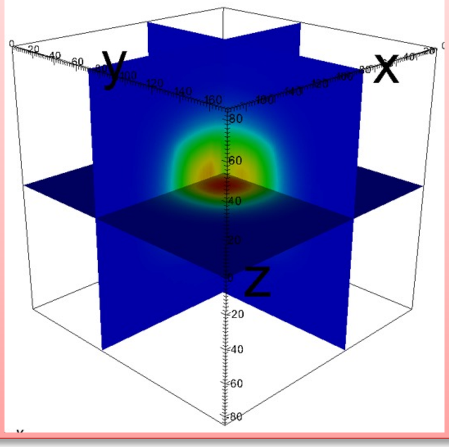
Lawrence Livermore National Laboratory
U.S. DEPARTMENT OF ENERGY
NNSA
CRAY
EL CAPITAN

Our switch to GPU-based computing is paying off with big performance increases on Sierra

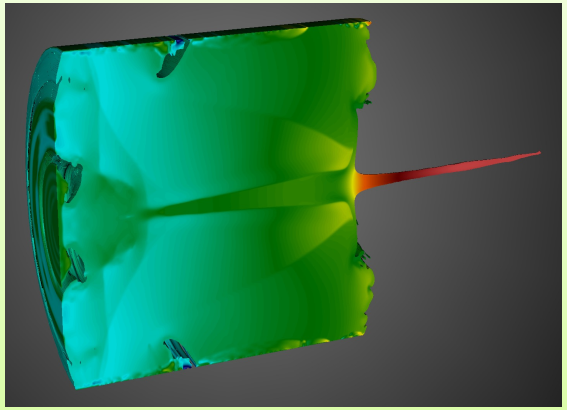
Ares, RT Mixing
13x speedup



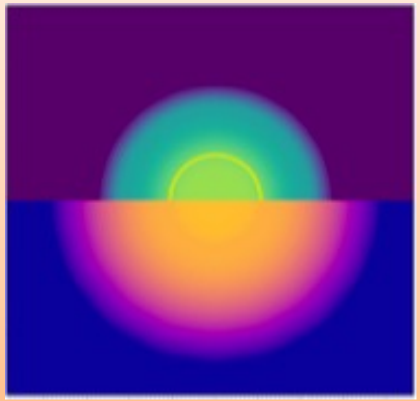
Ardra, Reactor Safety
16x speedup



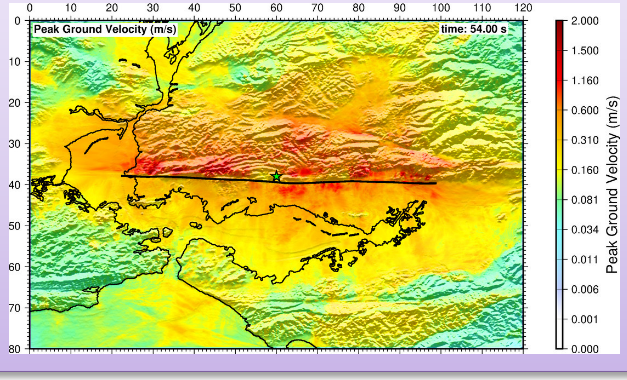
ALE3D, Shaped Charge
8x speedup



Kull/Teton
Radiating Sphere
7x speedup



SW4, Hayward Fault, 28x speedup

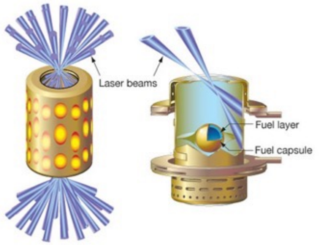


Our large, integrated multi-physics codes are tailored to mission needs and HPC resources and support a broad range of simulation capabilities

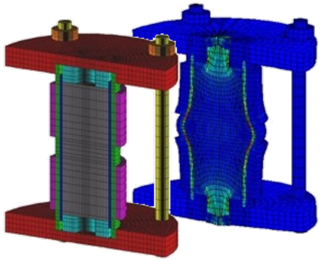
- Millions of lines of code in multiple programming languages
- Scale to O(1M) MPI ranks
- Multiple spatial/temporal scales
- Maintain connection to prior V&V efforts
- Coordinate with 10-60+ libraries

- Long life-time projects
  - 15+ years of development by large teams
  - 10–20+ people, ~50/50 CS/Physicists
- Portable performance
  - Our codes must be fast, reliable, and accurate on multiple systems
  - Laptops, Workstations, Commodity Clusters, Advanced Architectures, Heterogenous Architectures

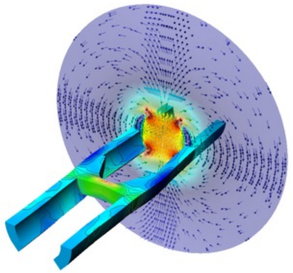
Inertial Confinement Fusion



HE Cookoff

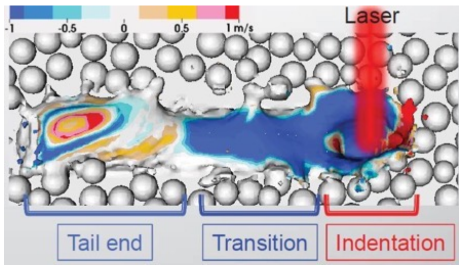


Navy Railguns



Fracture and Failure

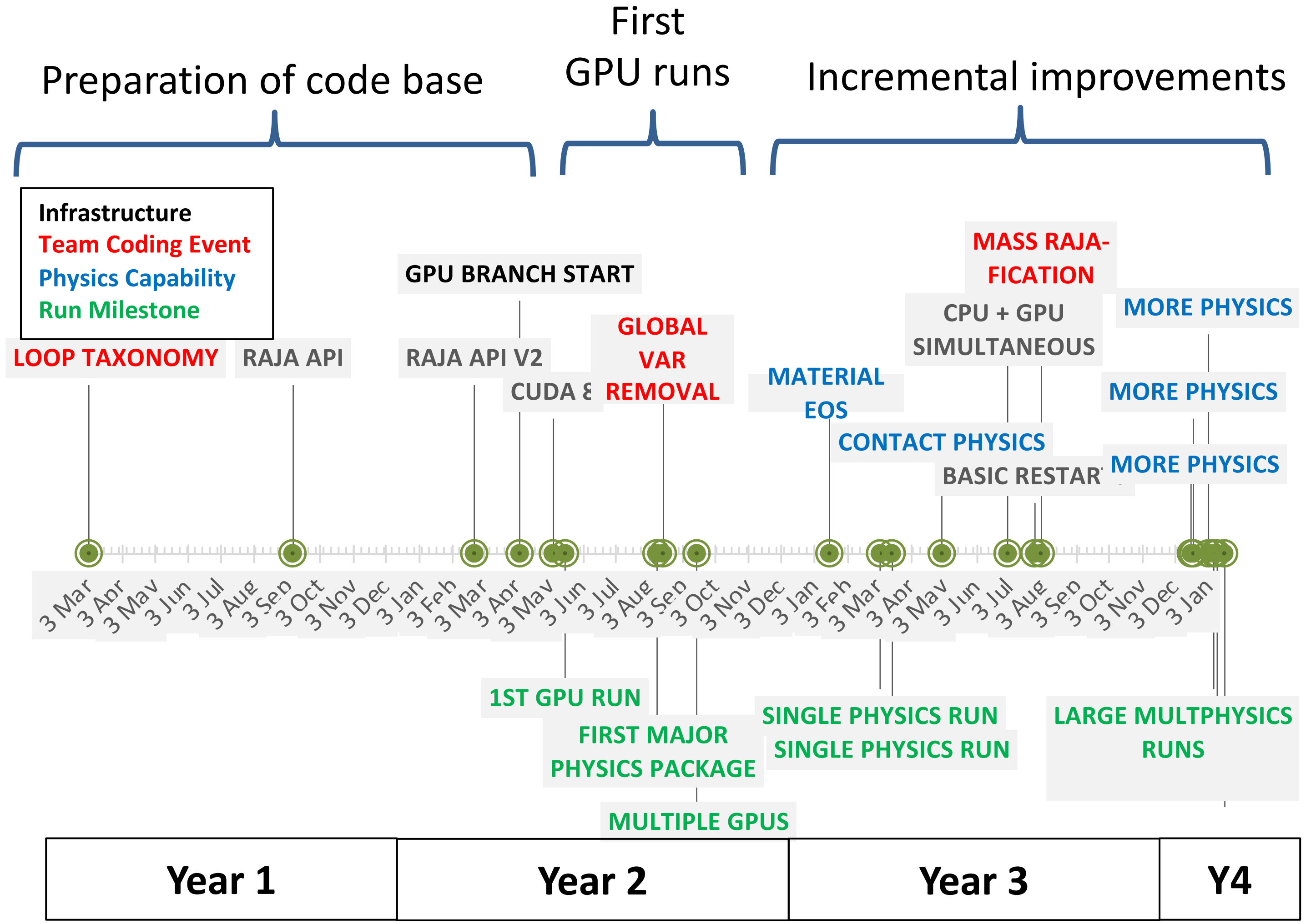
Additive Manufacturing



Migrating these codes to new architectures is a significant challenge

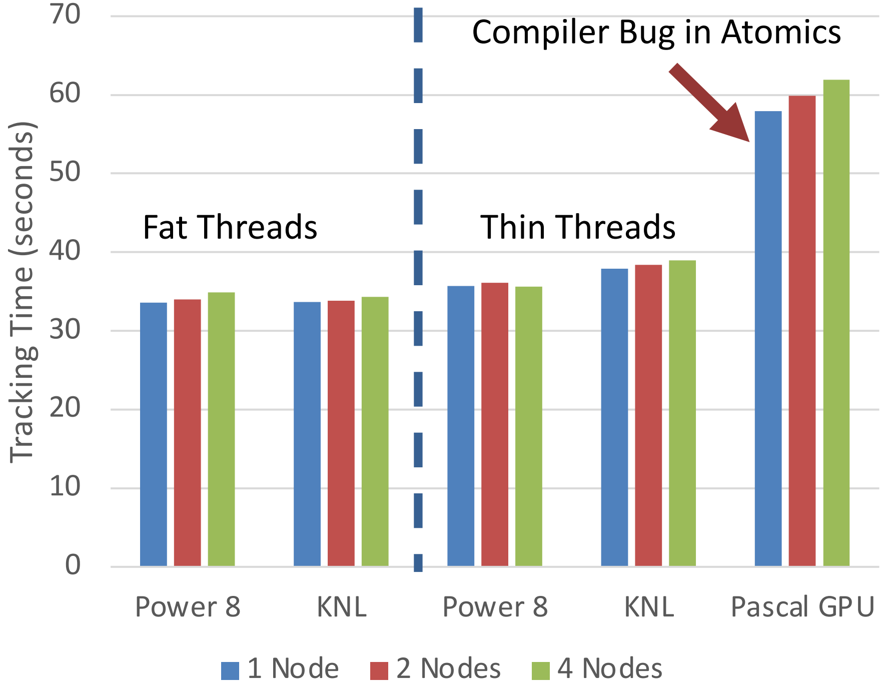
Successful application modernization follows a consistent pattern

1. Refactor and remove anti-patterns
2. Create a mini-app to explore design space
3. Use portable abstractions and frameworks
4. Focus on a specific use case
5. Search for additional parallelism
6. Manually manage memory
7. Iteratively apply the steps above



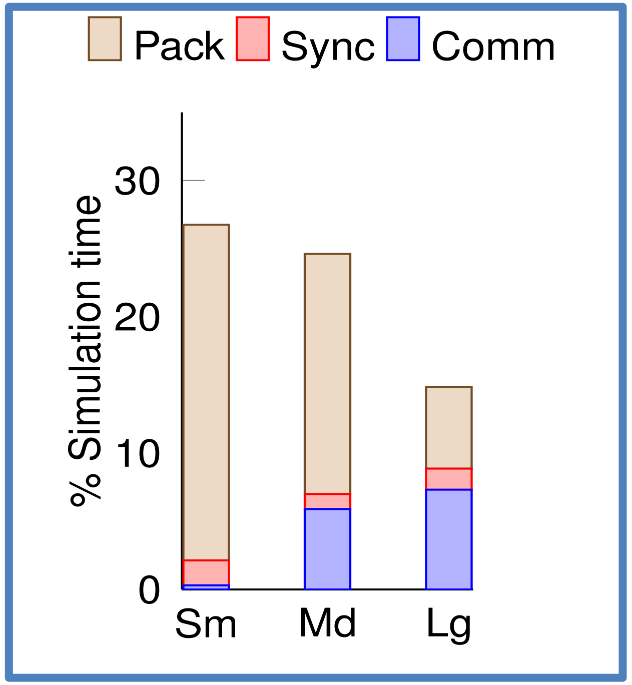
Proxy apps are extremely useful to explore design and refactoring choices as well as performance bottlenecks

Quicksilver tracking loop times (weak scaling, lower is better)



Testing thread strategies for Mercury

Comb represents the data packing and communication for the halo exchange in Ares



Messaging time breakdown for various job sizes on Sierra.

Proxy apps are also useful for benchmarking and vendor co-design

RAJA is our performance-portability solution that uses standard C++ idioms to target multiple back-end programming models

- Decouple loop traversal and iterate (body)
- An iterate is a “task” (aggregate, (re)order, …)
- IndexSet and execution policy abstractions simplify exploration of implementation/tuning options without disrupting source code

Pattern
(forall, reduction, scan, etc.)

Execution Policy
(how loop runs: PM backend, etc.)

Index
(index sets, segments to partition, order, …. iterations)

C-style for-loop

```
double* x ; double* y ;
double a, tsum = 0, tmin = MYMAX;

for ( int i = begin; i < end; ++i )  {
     y[i] += a * x[i] ;
     tsum += y[i] ;
     if ( y[i] < tmin ) tmin = y[i];
}
```

RAJA-style loop

```
double* x ; double* y ;
double a ;
RAJA::SumReduction<reduce_policy, double> tsum(0);
RAJA::MinReduction<reduce_policy, double> tmin(MYMAX);

RAJA::forall< exec_policy > ( index_set , [=] (int i) {
     y[i] += a * x[i] ;
     tsum += y[i];
     tmin.min( y[i] );
}  );
```

High-level parallelization abstractions target multiple back-ends and unlock cross platform portability

Unified (coherent) memory is helpful, but is not a panacea

No single strategy: multiple paths to success have emerged

- SW4: Allow managed memory to handle transfers. Overhead amortized by much re-use between transfers.
- Ares: Data transfers are explicit for performance. Managed memory pointers are helpful for libraries and code simplicity.
- Teton: All data transfers are explicit.

Abstractions improve code performance and developer productivity

- CHAI: Smart pointers automate explicit data transfers (Ardra, ALE3D)
- Umpire:
  - Unified, portable API to 3rd party memory capabilities
  - Coordinates memory use/introspection among multiple packages
  - Provides memory pools etc. to improve performance

Host-device data transfers must be treated as first class concerns

Umpire is being developed to coordinate complex memory allocations and movement

Assume three packages/libraries A,B,C – each with their own view of the GPU memory resources

Phase 0

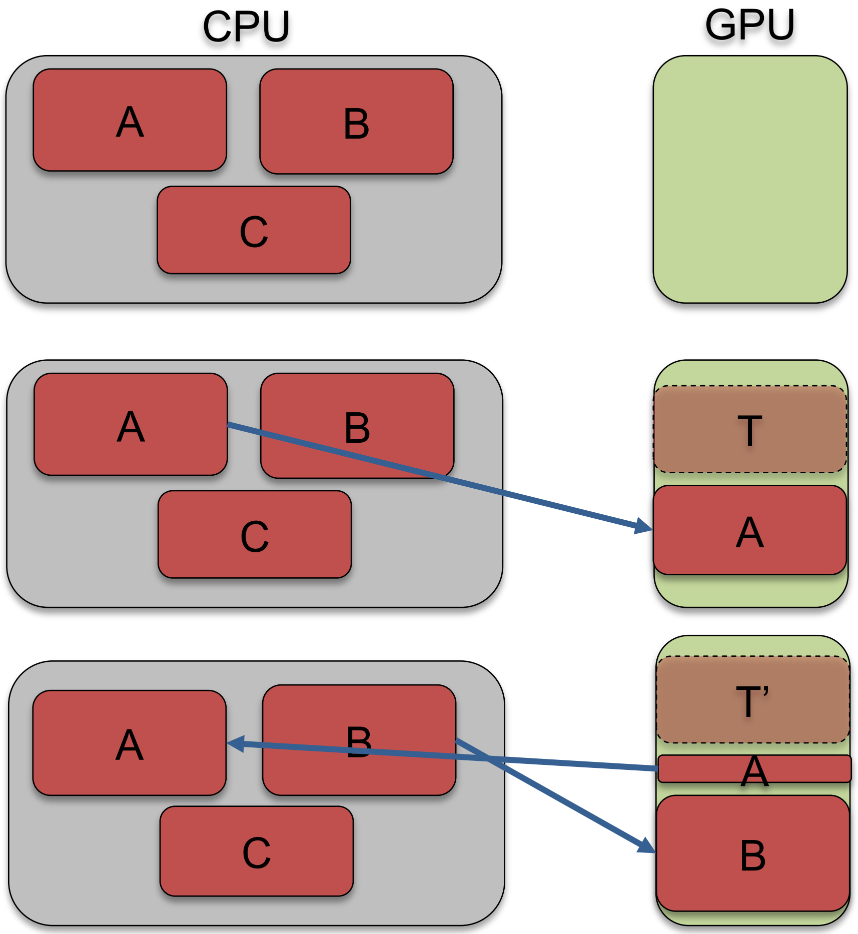
- Initial state of problem staged in CPU memory

Phase 1

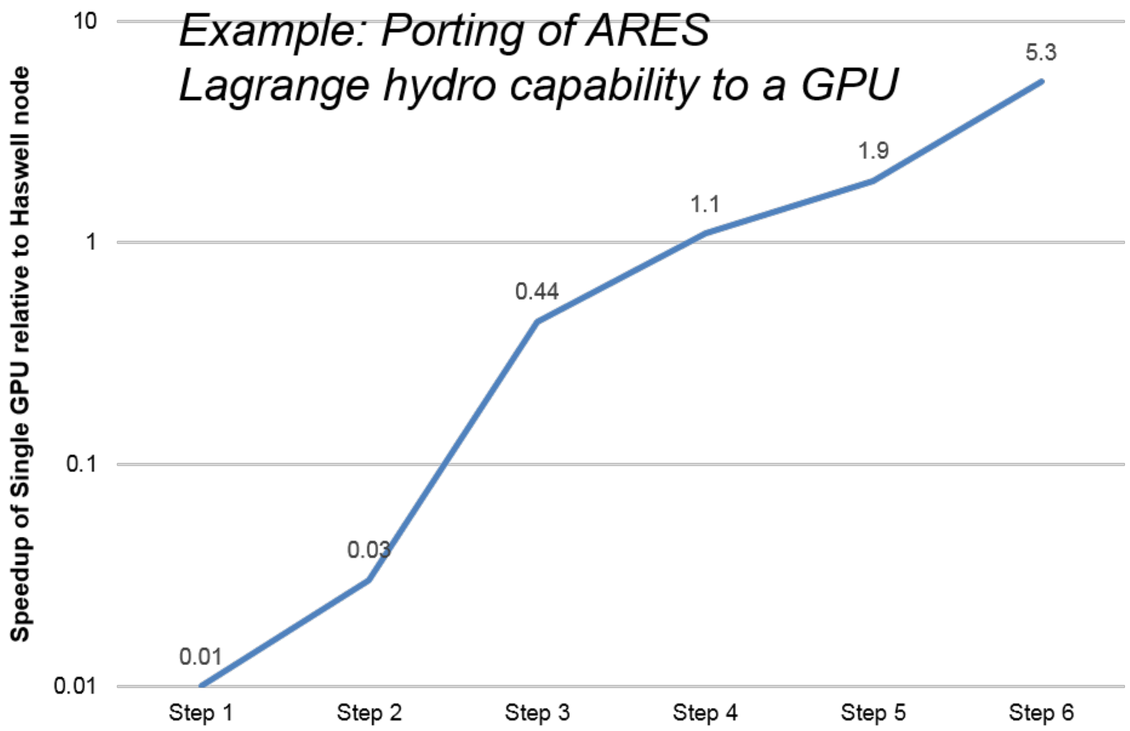
- A executes first
- A allocates temporary data in a memory pool (T)
- A's data is copied to GPU

Phase 2

- A's data copied back to CPU
  - Some shared data remains on the GPU
- B's data copied to GPU
- Temporary data T deallocated
- B allocates temporary data T' using the same memory pool

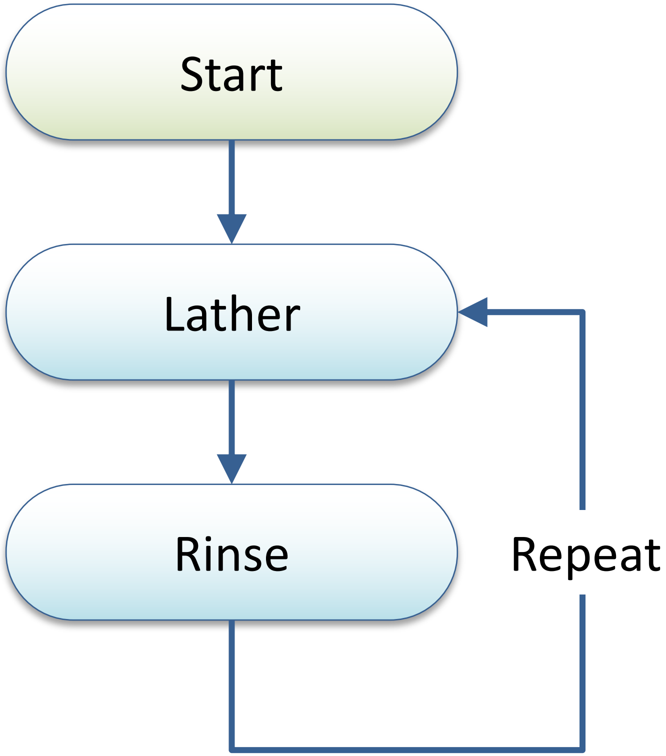


Performance improvement is an iterative process. Each step improves performance but also uncovers the next problem



Example: Porting of ARES Lagrange hydro capability to a GPU

Compile and run on a GPU	Minimize data motion	Minimze dynamic memory allocations	Asynchro-nous execution	Optimize individual kernels	Sierra EA hardware



This result required sustained effort over long time by many people.
Vendor partners and COE were critical to this success.

Fortran/OpenMP is not as well supported as C/C++ on GPUs

- Flang/F18 is likely to help with compiler availability
- OpenMP is the only real choice for portable GPU off-load in Fortran. No mechanism for abstraction layers.
- Modern Fortran and features not shared with C/C++ such as shaped arrays or array notation are especially problematic
- Write your Fortran code as much like C as possible if you want it to perform well
- Up-to-date proxies and tests are critical to ensuring compilers will function as desired

The Fortran community is relatively small compared to C++. We should pool effort and spread the overhead/effort.

For El Capitan, we use a 3-stage readiness process to prepare both applications and the system software environment

Level 1: Basic Build & Run Testing:
Expose show-stopping problems in the application code or the software stack

- Uncover any major problems as early as possible. Some work-arounds might be needed
- Build and run on CPU & GPU
- Test basic operation of debuggers and performance tools

Level 2: Performance Analysis and Comprehensive Tool Tests:
Establish performance baselines, expand tool testing

- Establish performance baselines on production and early-availability systems
- Characterize application performance and identify bottlenecks
- Utilize additional compilers, debugging, and performance tools

Level 3: Optimization and Testing at Scale:
Improve code performance and test tools at scale

- Optimization is almost always an iterative process. Train application team to continue code development and optimization independently of COE staff
- Report/resolve performance problems due to abstraction layers
- Test performance and debugging tools at scales typically used by developers
- Verify that changes do not adversely impact performance on other important platforms

Both applications and system software require testing and preparation on new hardware

Feature matrices help communicate the status of complex application codes with many build options and/or capabilities

Feature/Capability	Priority	CPU Build	GPU Build	GPU Performance
Feature 1	2	🟩	🟥	
Library 2	4	🟩	🟩	🟩
Capability 3	1	🟩	🟩	🟨
Use Case 4	5	🟩	🟥	
Feature 5	5	🟩	🟥	
Capability 6	5	🟥	🟥	

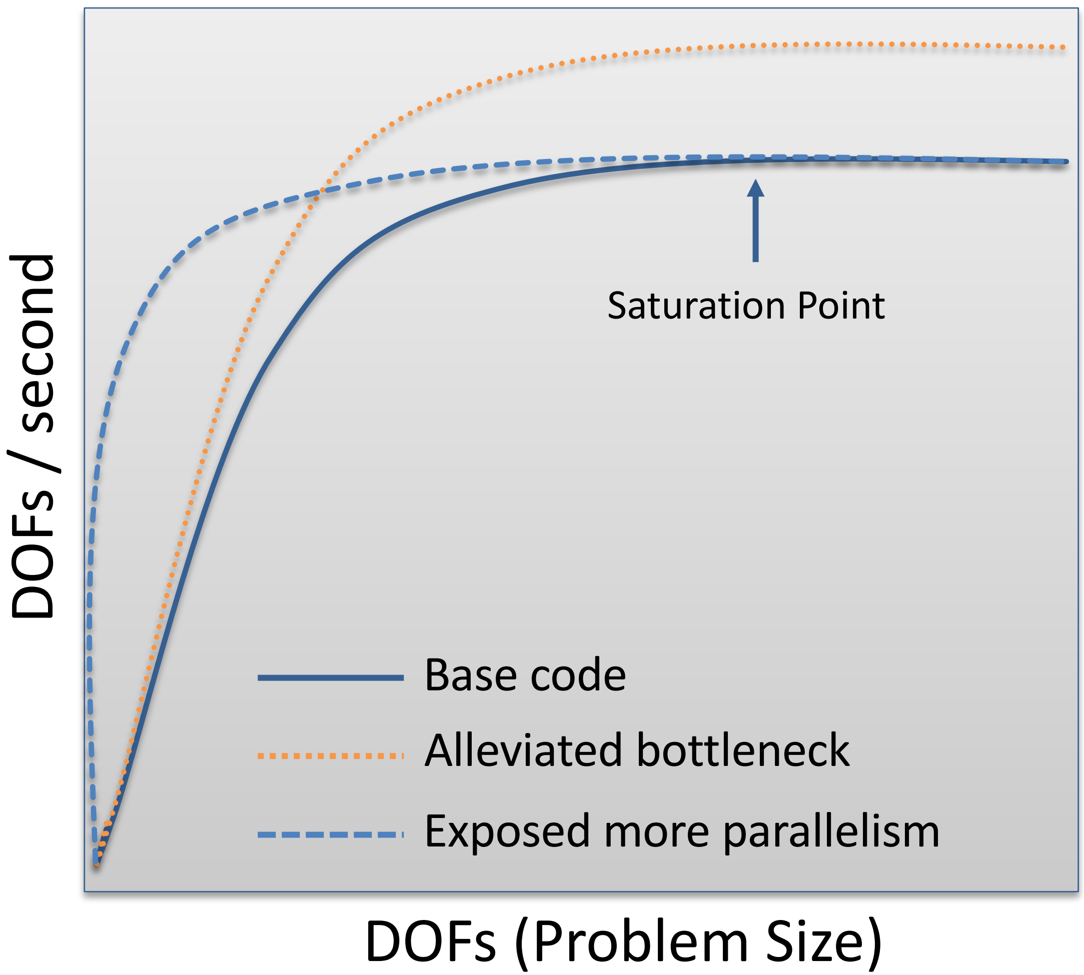
We also track work-arounds and known issues that apply to specific applications

Tracking features & priorities helps focus effort where needed

Large architecture differences between systems greatly complicates performance comparisons

- What does “10x speedup” mean?
  - Is it useful to say, “my code is 10x faster on Sierra than CTS-1”
- Can we develop ways to characterize the behavior of a code on different platforms in a way that
  - Makes sense
  - Allows natural comparisons
- Idea: use “throughput curves” to characterize performance
  - DOFs vs DOFs/second
  - Curves represent:
    - One specific code and one specific problem of interest
    - One “dimension” in changing DOFs (varying zones)
    - Using all of the resources on a single node
  - Saturation Point
    - Resources are bottlenecked
    - 2x more DOFs = 2x more time

Single-node throughput curve

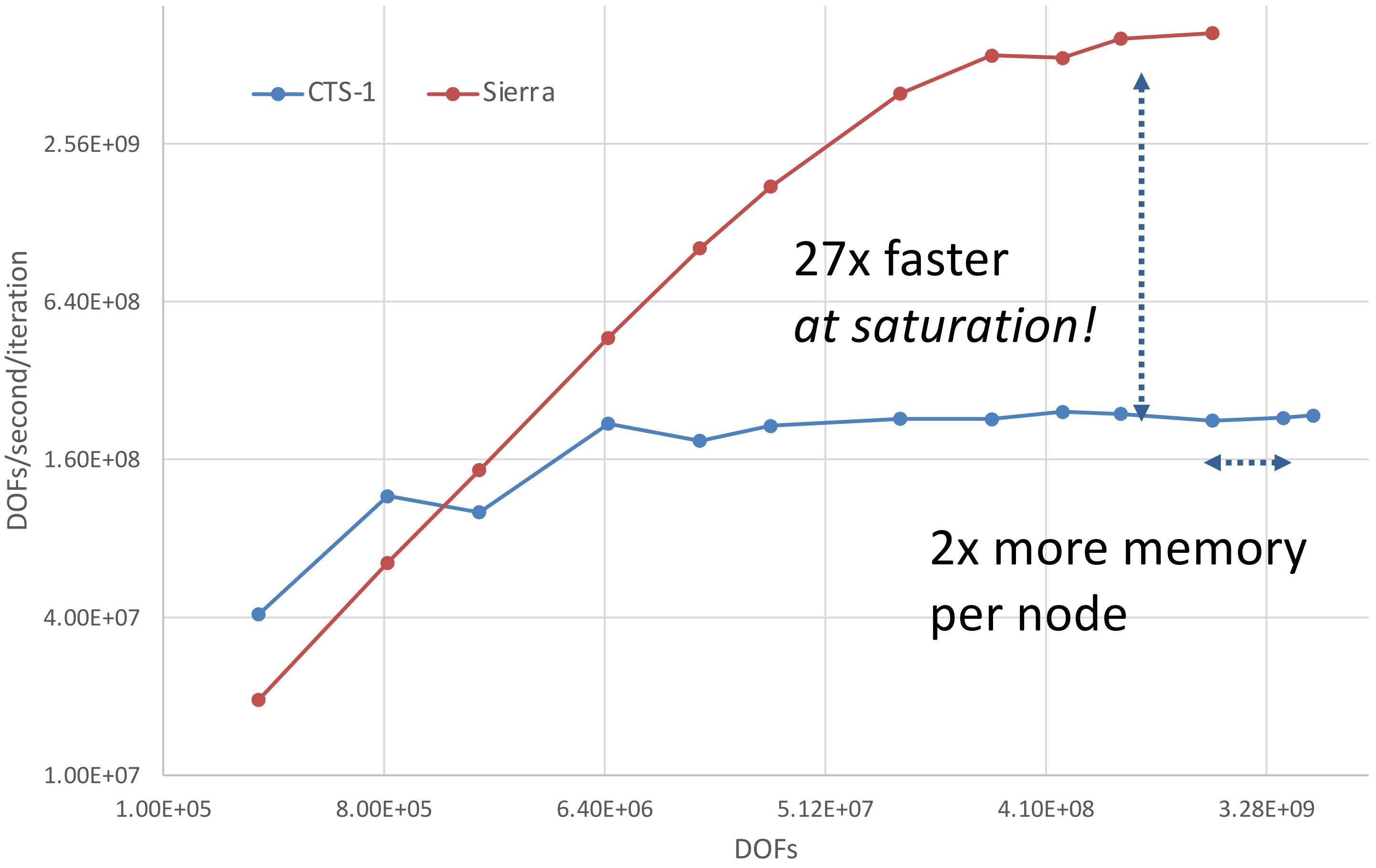


Based on slides by Adam Kunen

Throughput curves help characterize machine performance, and demonstrate the balance of parallelism and memory capacity

- Throughput curve:
  - Single node performance
  - DOFs/second
  - From very small problem to OOM
- Throughput is related to:
  - Available parallelism
  - Work required to saturate memory bandwidth
  - Work required to saturate compute
- CPUs (CTS-1)
  - Modest memory bandwidth and compute
  - Relatively small amount of parallelism
  - Easy to saturate, but lower overall performance
- GPUs (Sierra)
  - Very high memory bandwidth and compute
  - Large amount of parallelism
  - Need large amounts of work to saturate, but you get high performance in return

Ardra Single-Node Throughput on Sierra (GPU) vs CTS-1 (CPU)
3D, 48 groups, S8, P4 scattering, varying number of zones



27x faster
at saturation!

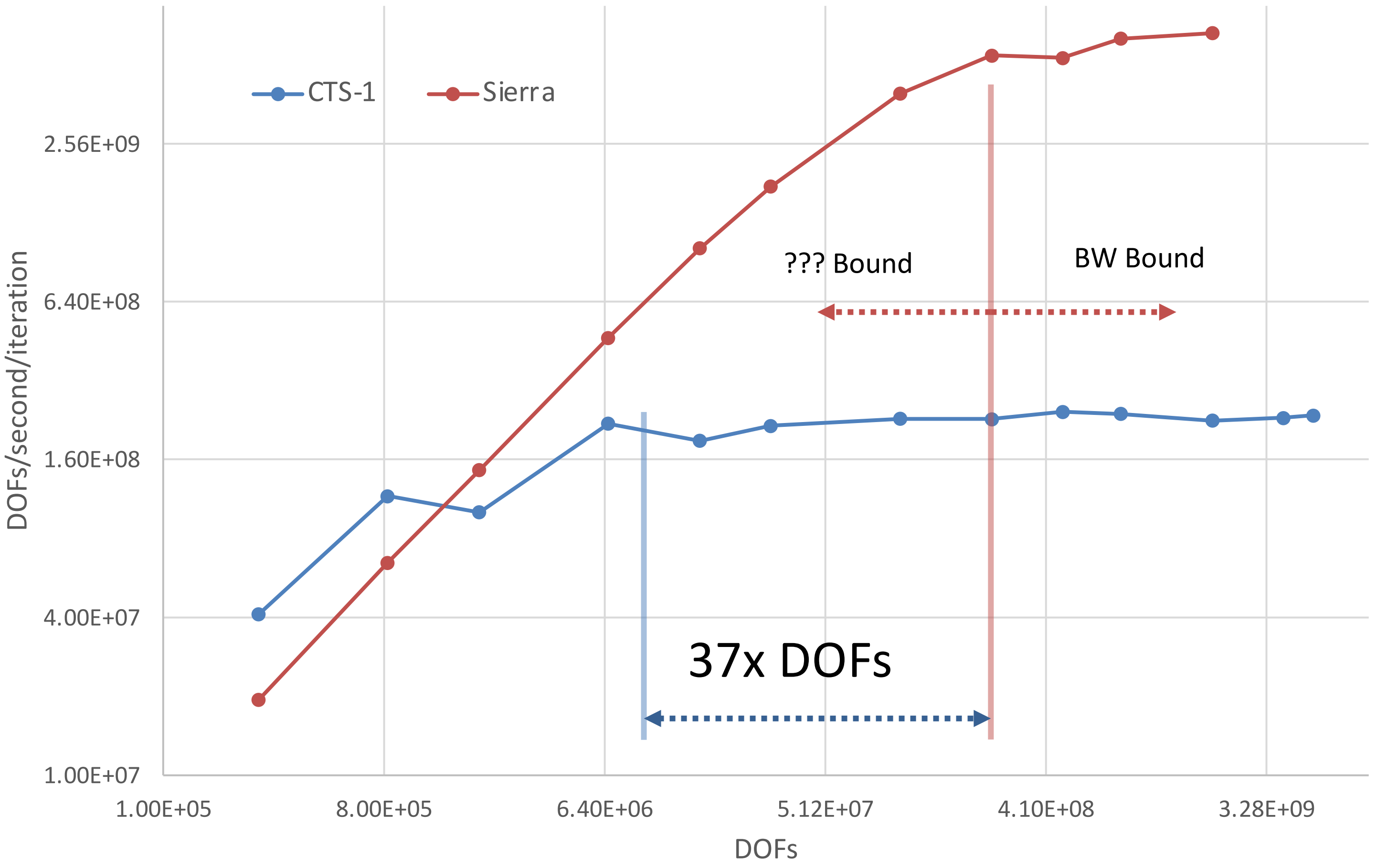
2x more memory per node

Slide from Adam Kunen

Performance bottlenecks only become clear at saturation, analysis below that point is murky

- Profiling beyond saturation
  - Mostly results in same results
  - Shows bottlenecks
- Profiling under saturation
  - Produces different results
  - This can be really misleading
- Efficiency improvements can move the curve higher
- Algorithmic changes that expose more parallelism can move the curve to the left

Ardra Single-Node Throughput on Sierra (GPU) vs CTS-1 (CPU)
3D, 48 groups, S8, P4 scattering, varying number of zones

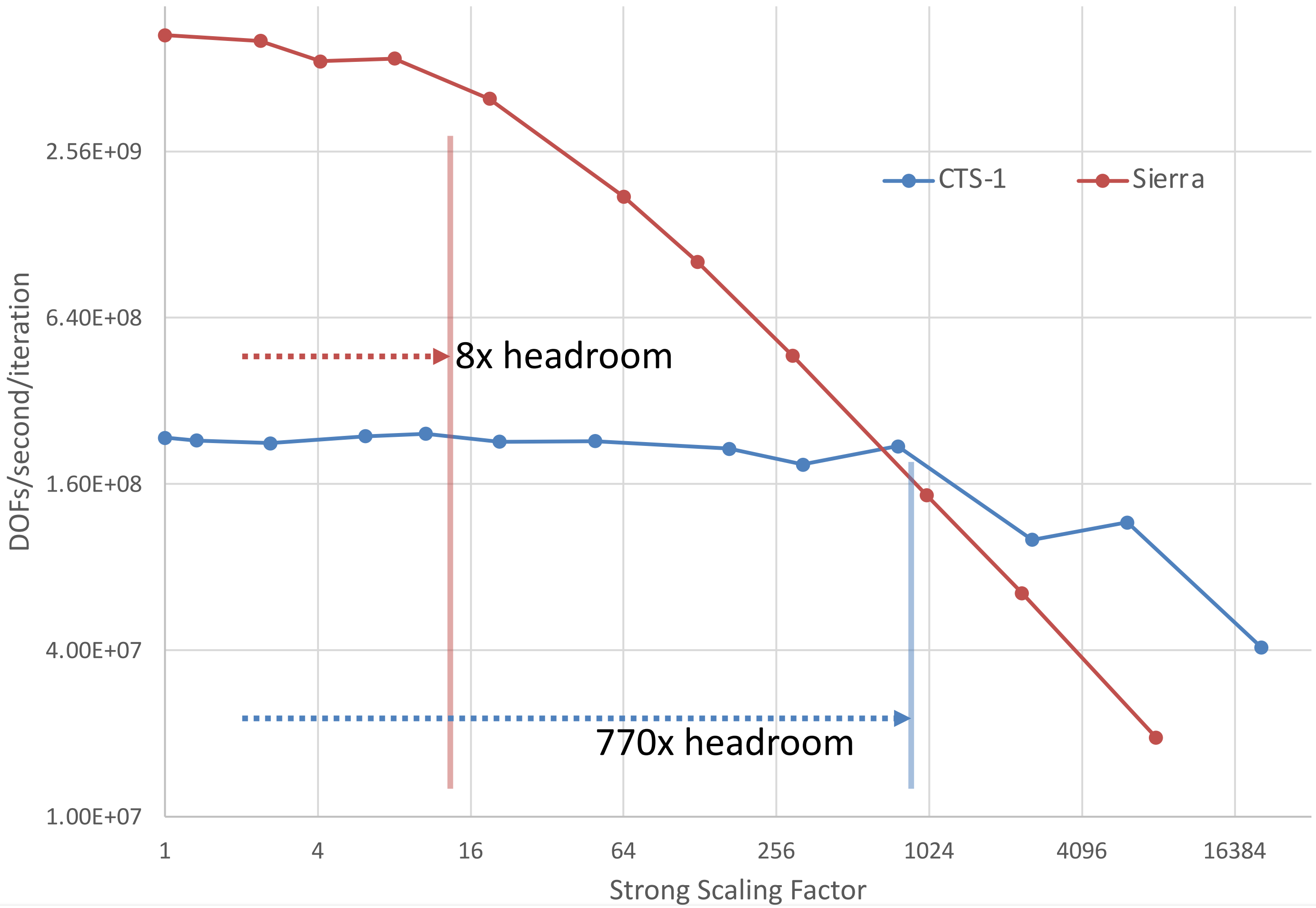


Slide from Adam Kunen

A smaller fraction of memory needed to achieve peak throughput provides more flexibility

- Headroom:
  - Inverse fraction of memory left over once you hit peak throughput
  - How much strong scaling can happen before *on-node performance* suffers
- Example:
  - 8x headroom = 1/8 memory being used
  - Could fit an 8x larger problem in same resource
  - Could strong scale an 8x larger problem by 8x
- A larger headroom is more flexible:
  - More strong scaling potential
  - Smaller problems can be run efficiently

Ardra Single-Node Throughput on Sierra (GPU) and CTS-1 (CPU)
3D, 48 groups, S8, P4 scattering, varying number of zones



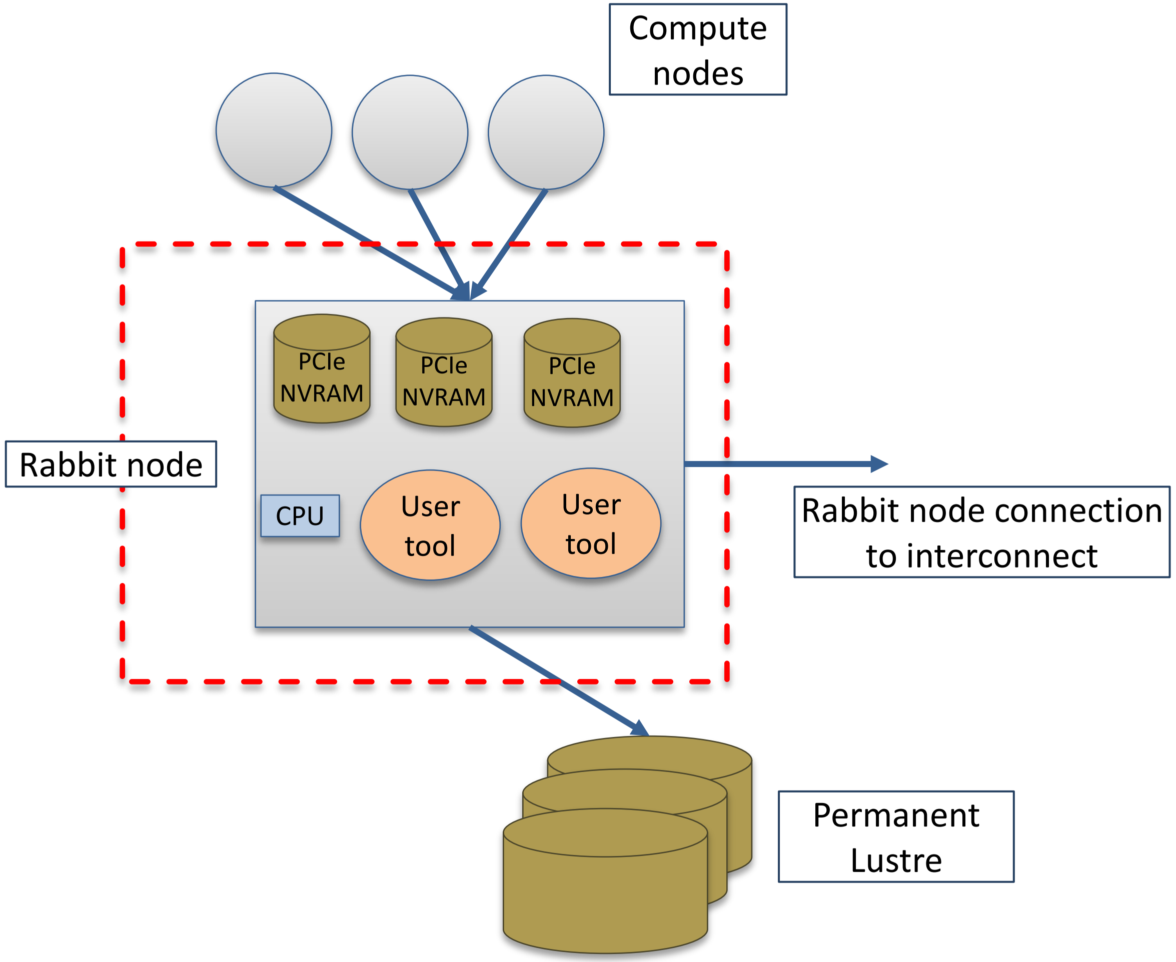
Slide from Adam Kunen

Complex workflows including machine learning, and real-time analytics or visualization are placing new demands on facilities

Four key lessons learned from large scale workflows

- Optimize resource allocations at the workflow level
  - Match workflow elements to hardware
  - Allocate data generators close to corresponding data consumers
- Use workflow management tools
  - Matching the available resources to ready tasks requires dedicated management software
  - Checkpointing a workflow can be harder than you think
- Consider the memory hierarchy and data sharing tools when designing a workflow
  - File I/O is not adequate to coordinate complex workflows
- Package managers and continuous integration can help ensure the reproducibility of a workflow

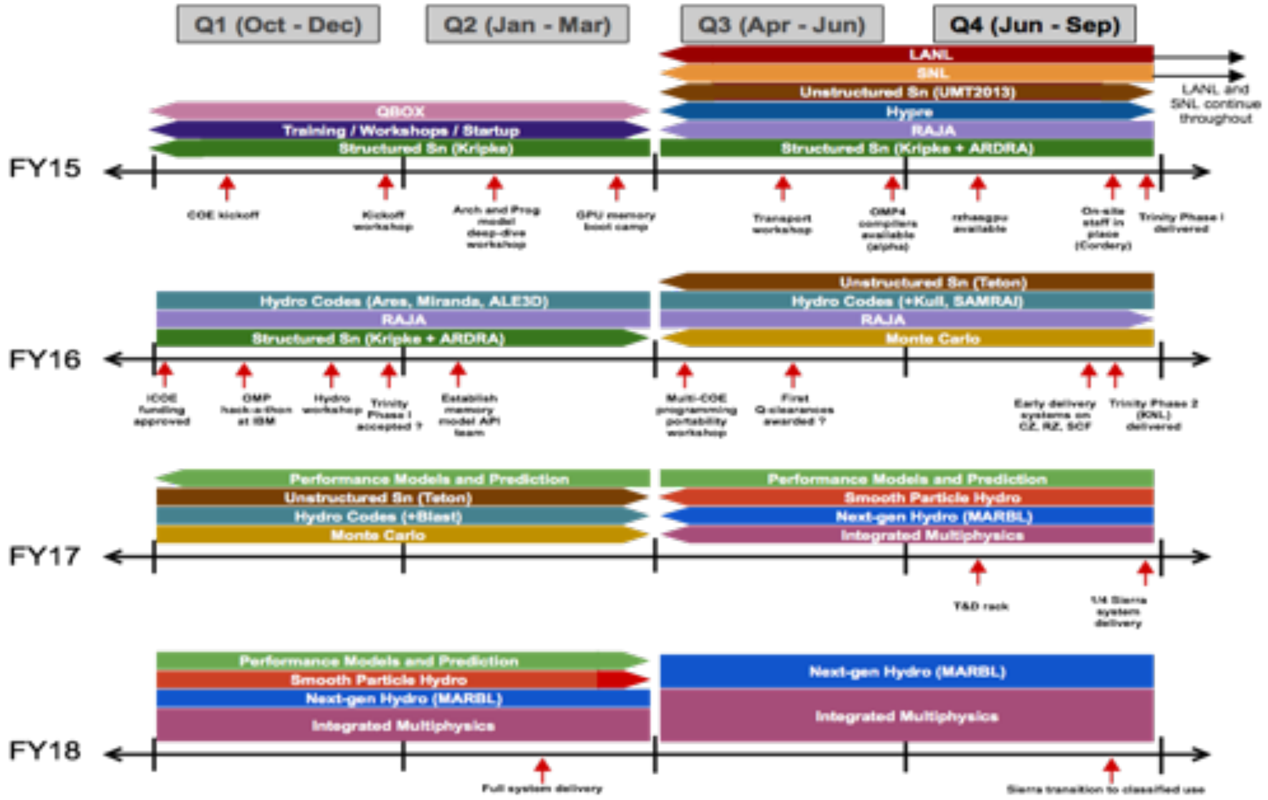
The El Capitan I/O system will support modern workflows with near-node local storage called Rabbits



- Each Rabbit will have 18 SSDs (16+2 spare) and 1 AMD EPYC CPU
- User tools can run in containers on the Rabbit CPU, e.g., SCR, UnifyFS, analytics tools, etc.
- Rabbits are connected via the system interconnect and PCIe connection to local compute nodes
  - User tools can communicate between Rabbits
  - User tools can also communicate between compute nodes and Rabbits

Sierra Center of Excellence work plans featured built-in flexibility

- Codes and libraries "phased in" over time
  - — Multi-year plan overlaid with hardware and compiler availability to guide work plans
- Work plans were intentionally overcommitted to allow agility
- Earliest work focused on training and proxy apps
- After year one – pivoted to real applications and greater team engagement with vendor help


Q1 (Oct - Dec)
Q2 (Jan - Mar)
Q3 (Apr - Jun)
Q4 (Jun - Sep)
FY15
QBOX
Training / Workshops / Startup
Structured Sn (Kripke)
LANL
SNL
Unstructured Sn (UMT2013)
Hypre
RAJA
Structured Sn (Kripke + ARDRA)
LANL and SNL continue throughout
COE kickoff
Kickoff workshop
Arch and Prog model deep-dive workshop
GPU memory boot camp
Transport workshop
OMP4 compilers available (alpha)
rzhasgpu available
On-site staff in place (Cordery)
Trinity Phase I delivered
FY16
Hydro Codes (Ares, Miranda, ALE3D)
RAJA
Structured Sn (Kripke + ARDRA)
Unstructured Sn (Teton)
Hydro Codes (+Kull, SAMRAI)
RAJA
Monte Carlo
iCOE funding approved
OMP hack-a-thon at IBM
Hydro workshop
Trinity Phase I accepted ?
Establish memory model API team
Multi-COE programming portability workshop
First Q-clearances awarded ?
Early delivery systems on CZ, RZ, SCF
Trinity Phase 2 (KNL) delivered
FY17
Performance Models and Prediction
Unstructured Sn (Teton)
Hydro Codes (+Blast)
Monte Carlo
Performance Models and Prediction
Smooth Particle Hydro
Next-gen Hydro (MARBL)
Integrated Multiphysics
T&D rack
1/4 Sierra system delivery
FY18
Performance Models and Prediction
Smooth Particle Hydro
Next-gen Hydro (MARBL)
Integrated Multiphysics
Next-gen Hydro (MARBL)
Integrated Multiphysics
Full system delivery
Sierra transition to classified use

Effective collaboration doesn’t happen by accident

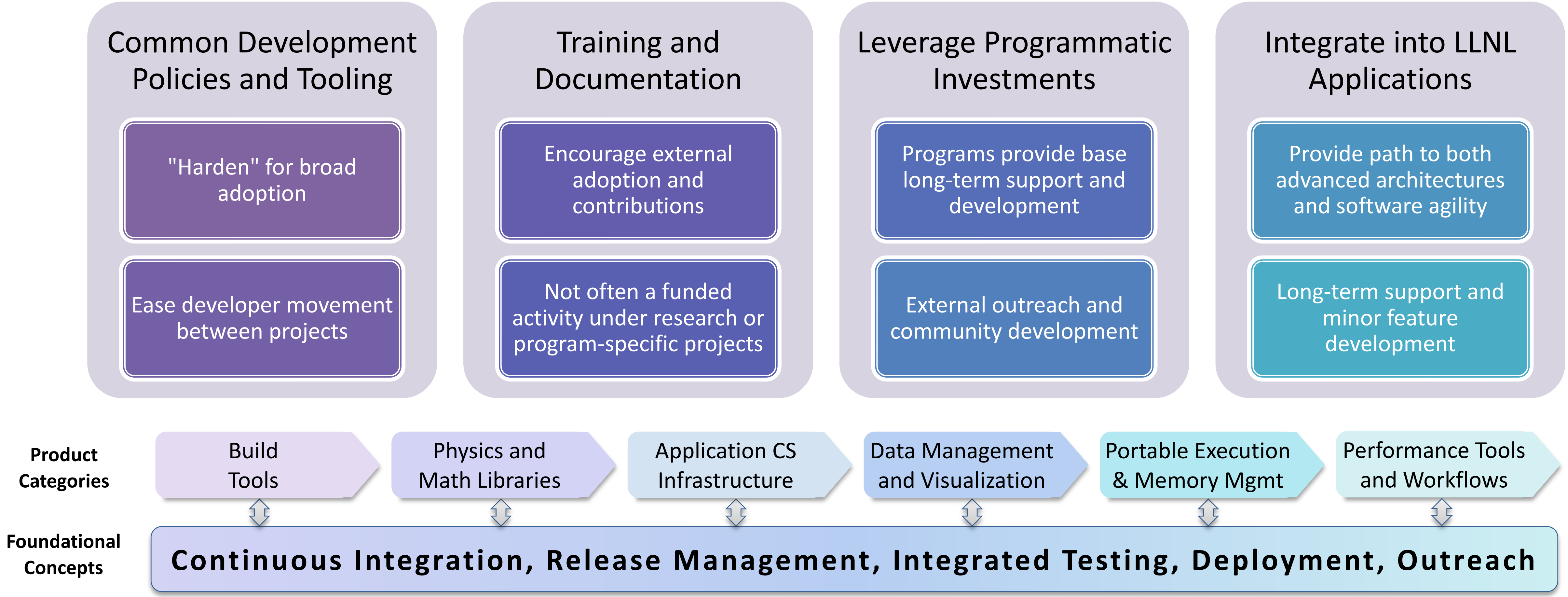
- Schedule activities to ensure two-way engagement
  - Trying to force interactions when priorities don’t align is a recipe for failure
- Be prepared to deviate from your plan
  - Things will go wrong
  - Opportunities will arise
- Invest in collaboration and software engineering tools
  - A common set of repos and communication tools will enhance productivity
  - Multi-site, secure tools can be hard to find
  - Avoid fragmentation of information
- Build multidisciplinary teams
  - Co-locate teams as much as possible.

Don’t expect sunshine, lollipops, and rainbows

- Usual new system pains: MPI, scheduler, compilers. These take time to resolve
- There is constant tension between system stability and bleeding-edge system software
- Some apps/algorithms aren’t there yet
  - Monte Carlo, Multi-grid setup phase
- Proprietary tool suites don’t handle all of our use cases
  - Vendor tools don’t always play nicely with HPC, scaling, MPI, etc.
  - Open source tools provide choices for debuggers, performance tools, etc.
- Interoperability of parallel models (OMP, CUDA, …), compliers, memory handling, across libraries are still difficult

RADIUSS supports an LLNL-developed open source software stack for rapid and enduring HPC application development

radiuss



Common Development Policies and Tooling

"Harden" for broad adoption

Ease developer movement between projects

Training and Documentation

Encourage external adoption and contributions

Not often a funded activity under research or program-specific projects

Leverage Programmatic Investments

Programs provide base long-term support and development

External outreach and community development

Integrate into LLNL Applications

Provide path to both advanced architectures and software agility

Long-term support and minor feature development

Product Categories

- Build Tools
- Physics and Math Libraries
- Application CS Infrastructure
- Data Management and Visualization
- Portable Execution & Memory Mgmt
- Performance Tools and Workflows

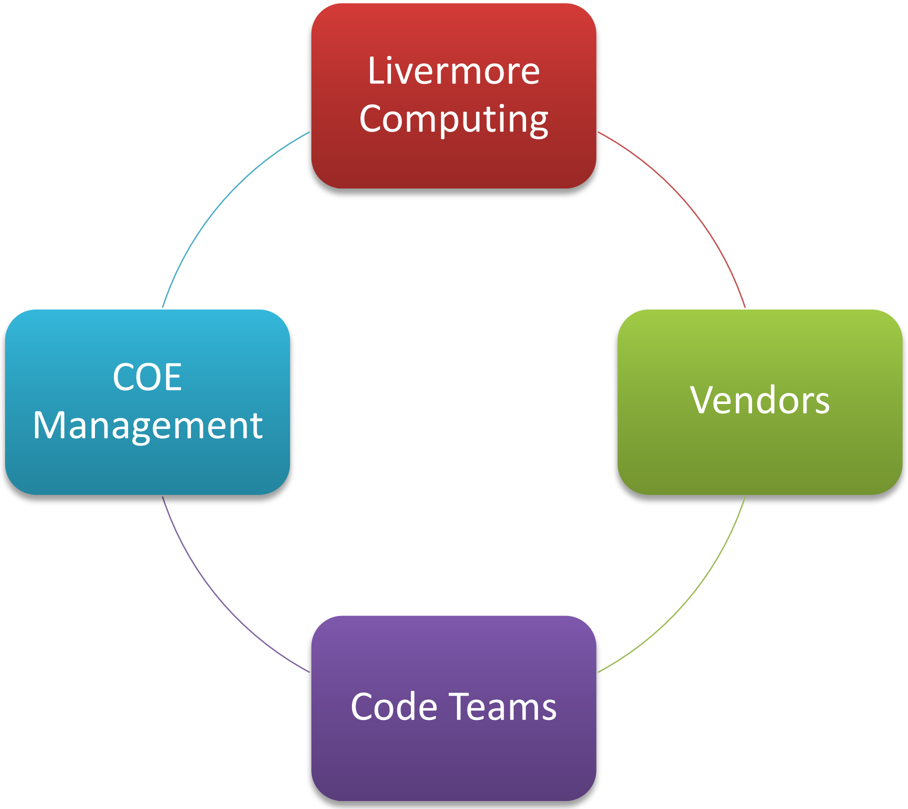
Foundational Concepts

Continuous Integration, Release Management, Integrated Testing, Deployment, Outreach

Software is core infrastructure to the laboratory, similar to institutional HPC platforms and laboratory space.
Sustained software investments are core to the mission of LLNL and our continued HPC leadership.

Migrating to heterogeneous architectures has taken years of hard work, but the results are worth it

- Many codes are seeing speedups of 10x or more
  - It *is* possible to incrementally refactor a large production code
- Code refactoring has reduced technical debt
  - But this takes commitment
- Increased performance is opening doors to previously impossible science
- Lessons learned and ecosystem improvements blaze a trail for others to follow
  - Future efforts should be easier/faster due to improvements in supporting software



Multi-discipline, multi-talent teams are a key ingredient for success

References

- Sierra Center of Excellence: Lessons learned, IBMJRD vol 64, issue 3/4, 2:1-2:14
  - DOI: 10.1147/JRD.2019.2961069
- The Importance of Kernels for Performance Portability, Tom Scogland, 2020 P3HPC Forum
  - https://www.youtube.com/watch?v=6OypNNucCrY

Previous HPC Best Practices Webinars

https://ideas-productivity.org/events/hpc-best-practices-webinars/

- An Overview of the RAJA Portability Suite (#50)
- Best Practices for Using Proxy Applications as Benchmarks (#39)
- Introduction to Kokkos (#37)
- Quantitatively Assessing Performance Portability with Roofline (#25)

Lawrence Livermore
National Laboratory

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.