Exascale Computing at ORNL Past, Current, and Future: Opportunities for High Energy Physics

Tom Evans
HPC Methods and Applications Group
Oak Ridge National Laboratory

CERN
June 29, 2022

Outline

- HPC Computing at the OLCF
- The Exascale Computing Project
  - Overview
  - Projects and Science
- Experiences on AMD/Frontier
- Development of HEP Applications for use at the OLCF

OLCF
OAK RIDGE LEADERSHIP COMPUTING FACILITY

ECP
EXASCALE COMPUTING PROJECT

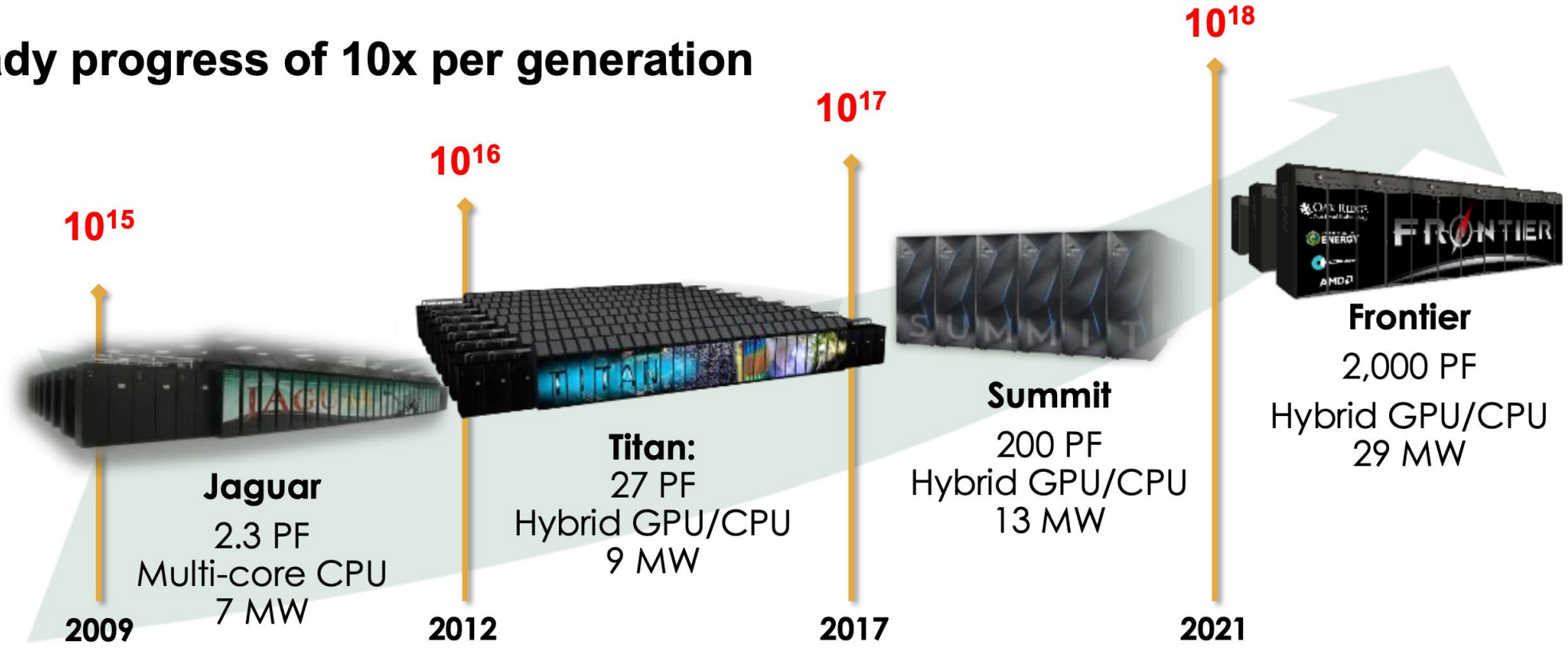
U.S. DEPARTMENT OF ENERGY | Office of Science

HPC Computing at the OLCF

OLCF pathway to Exascale

Mission: Providing world-class computational resources and specialized services for the most computationally intensive global challenges	Vision: Deliver transforming discoveries in energy technologies, materials, biology, environment, health, etc.

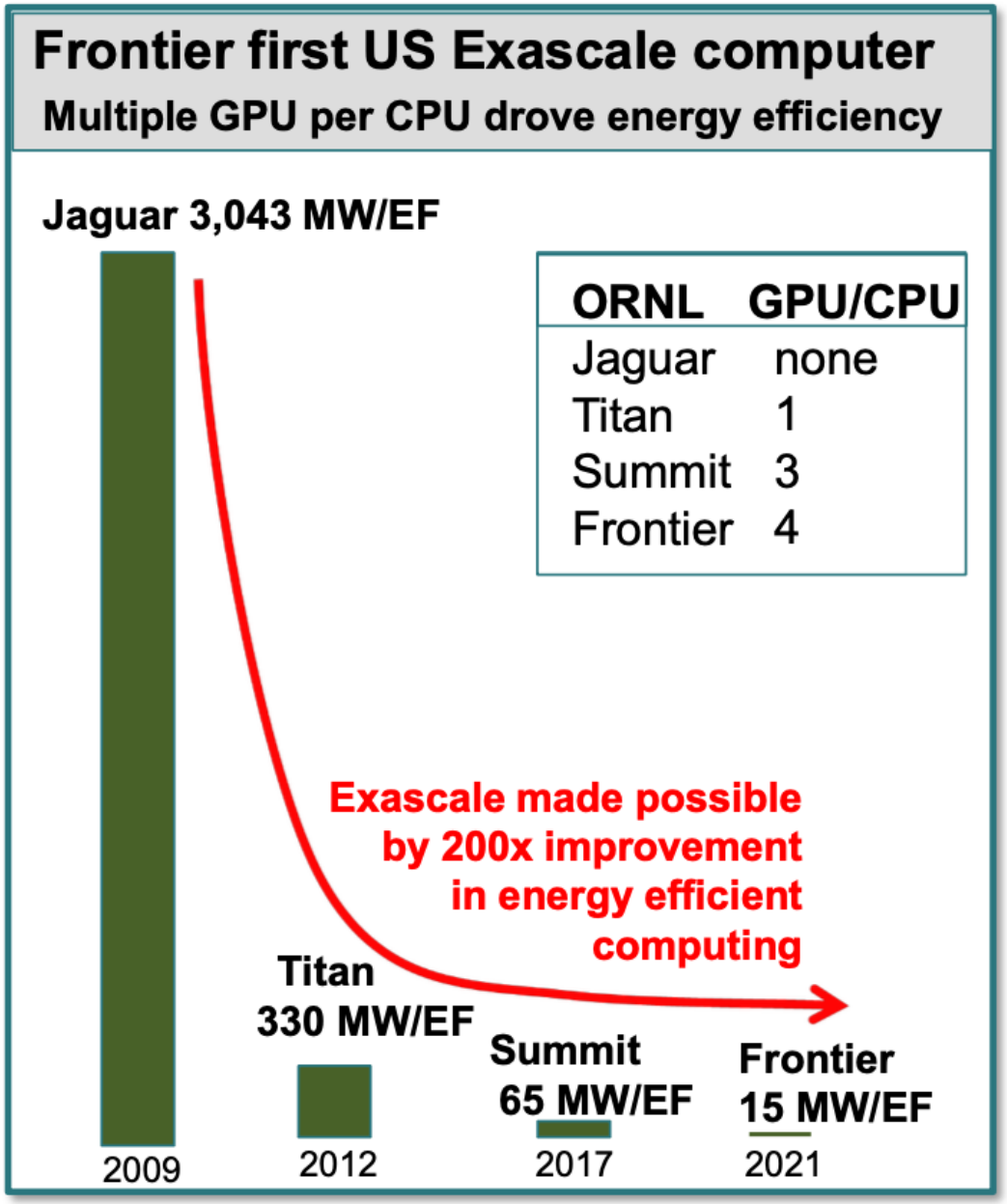
Steady progress of 10x per generation



Energy efficient computing

Since 2009 the biggest concern with reaching Exascale has been energy consumption

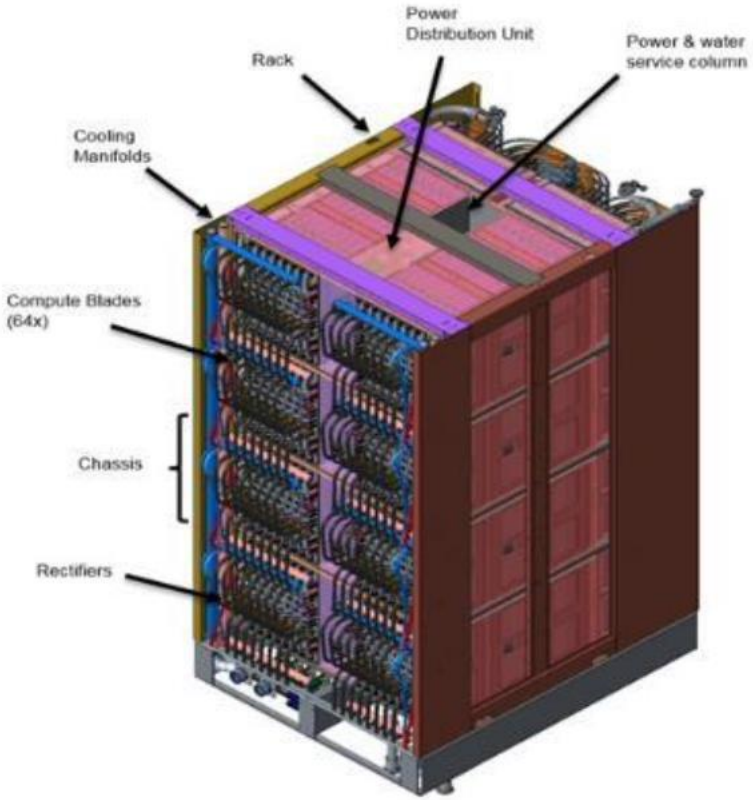
- **ORNL pioneered GPU use in supercomputing** beginning in 2012 with Titan thru today with Frontier. Significant part of energy efficiency improvements
- **ASCR [Fast, Design, Path] Forward vendor investments** in energy efficiency (2012 – 2020) further reduced the power consumption of computing chips (CPUs and GPUs)
- **200x reduction in energy per Flops** from Jaguar to Frontier at ORNL
- ORNL achieves additional energy savings from using warm water cooling in Frontier (32 C). **ORNL Data Center PUE = 1.03**



Frontier overview

System
- 2 EF peak DP Flops
- 74 compute racks
- 29 MW power consumption
- 9408 nodes
- 9.2 PB memory (4.6 PB HBM, 4.5 PB DDR4)
- Cray Slingshot network with dragonfly topology
- 37 PB node local storage
- 716 PB center-wide storage
- 4000 ft^2 footprint

Olympus rack
- 128 AMD nodes
- 8000 lbs
- Supports 400 kW



AMD node
- 1 AMD "Trento" CPU
- 4 AMD MI250X GPUs
- 512 GiB DDR4 memory on CPU
- 512 GiB HBM2e total per node (128 GiB HBM per GPU)
- Coherent memory across the node
- 4 TB NVM
- GPUs & CPU fully connected with AMD Infinity Fabric
- 4 Cassini NICs, 100 GB/s network BW

Compute blade
- 2 AMD nodes

All water cooled, even DIMMS and NICs

Frontier multi-tier storage system is designed to excel at Data Science and AI for Scientific Discovery

Capacity	Performance	
Multi-tier I/O Subsystem	**Read**	**Write**
37 PB Node Local Storage	65.9 TB/s	62.1 TB/s
	11 Billion IOPS	
11 PB Performance tier	9.4 TB/s	9.4 TB/s
695 PB Capacity tier	5.2 TB/s	4.4 TB/s
10 PB Metadata	2M Transactions per sec	

Two 2 TB SSD NVM per node local storage (flash)

Gazelle SSD storage board (Performance Tier and Metadata)

Moose HDD storage board (Capacity Tier)

Frontier first to break ExaFlop barrier

#1 Top500 list

- 1.1 exaflops on the Linpack Benchmark test
- > 7x performan

#1 Green500 list

- 62.86 gigaflops

One of our goals this week is to complete the User Agreement between CERN and OLCF to enable collaboration and use of Frontier for HEP science

https://www.olcf.ornl.gov/frontier

The Exascale Computing Project

- Overview
- Projects and Science

The Exascale Computing Project (ECP) enables US revolutions in technology development; scientific discovery; healthcare; energy, economic, and national security

ECP Mission

Develop exascale-ready applications and solutions that address currently intractable problems of strategic importance and national interest.

Create and deploy an expanded and vertically integrated software stack on DOE HPC exascale and pre-exascale systems, defining the enduring US exascale ecosystem.

Deliver **US HPC vendor technology advances and deploy ECP products** to DOE HPC pre-exascale and exascale systems.

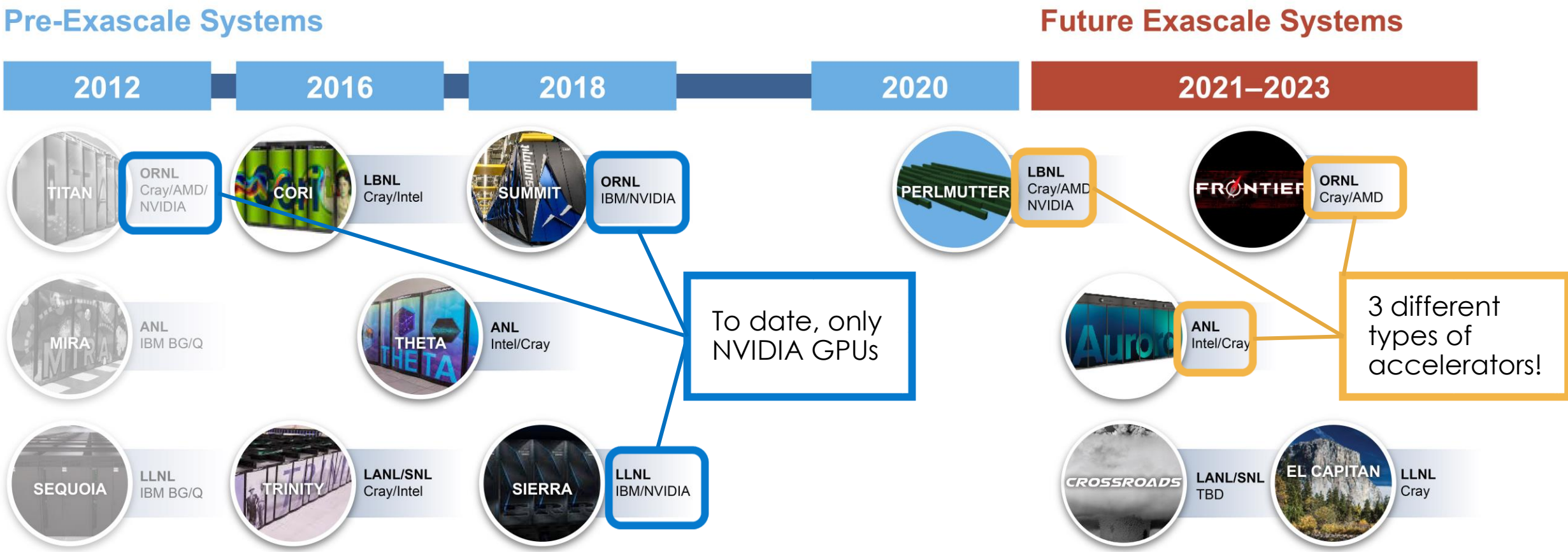
ECP Vision

Deliver **exascale simulation and data science innovations and solutions to national problems** that enhance US economic competitiveness, change our quality of life, and strengthen our national security.

- Funded by Office of Science, Advanced Scientific Computing Research (ASCR)
- 7 year project – $1.7B
- 6 core labs: ORNL, ANL, LBNL, LLNL, SNL, LANL
- More than 100 research teams
  - >1000 researchers
  - Drawn heavily from 17 DOE labs plus national universities

Where we are going

Department of Energy (DOE) Roadmap to Exascale Systems



ECP Application Development

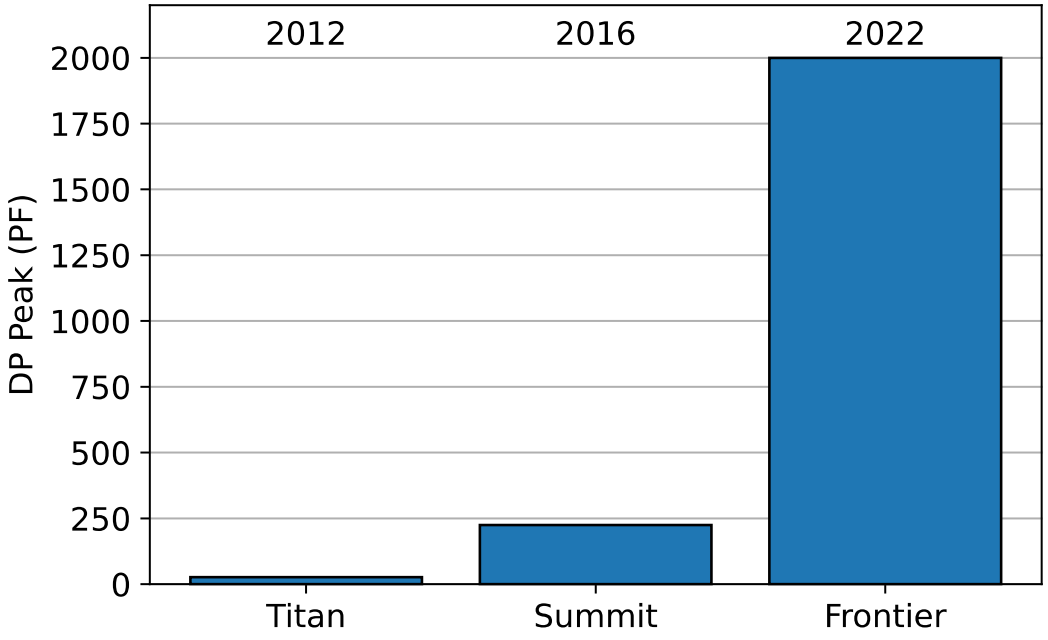
Goal: Ensure that exascale hardware impacts DOE science/engineering mission

Approach: Significant investment in scientific applications well in advance of exascale machines

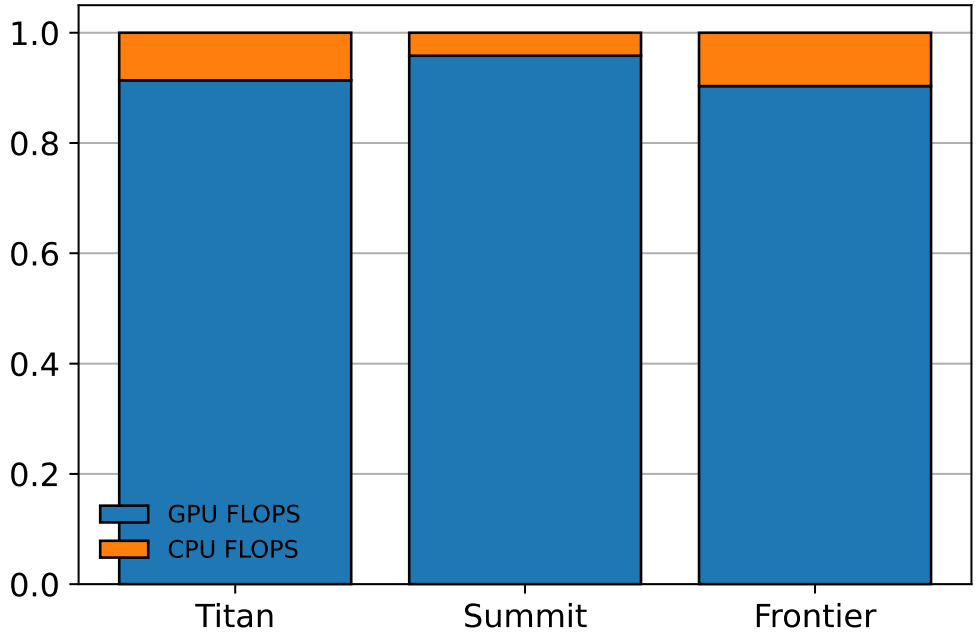
- ~~Code Porting~~
- Algorithmic Restructuring
- New Numerical Approaches
- Alternate choice of Physical Models

Hardware has significant impact on all aspects of simulation strategy

Performance on current and next-gen HPC architectures requires effective use of GPUs

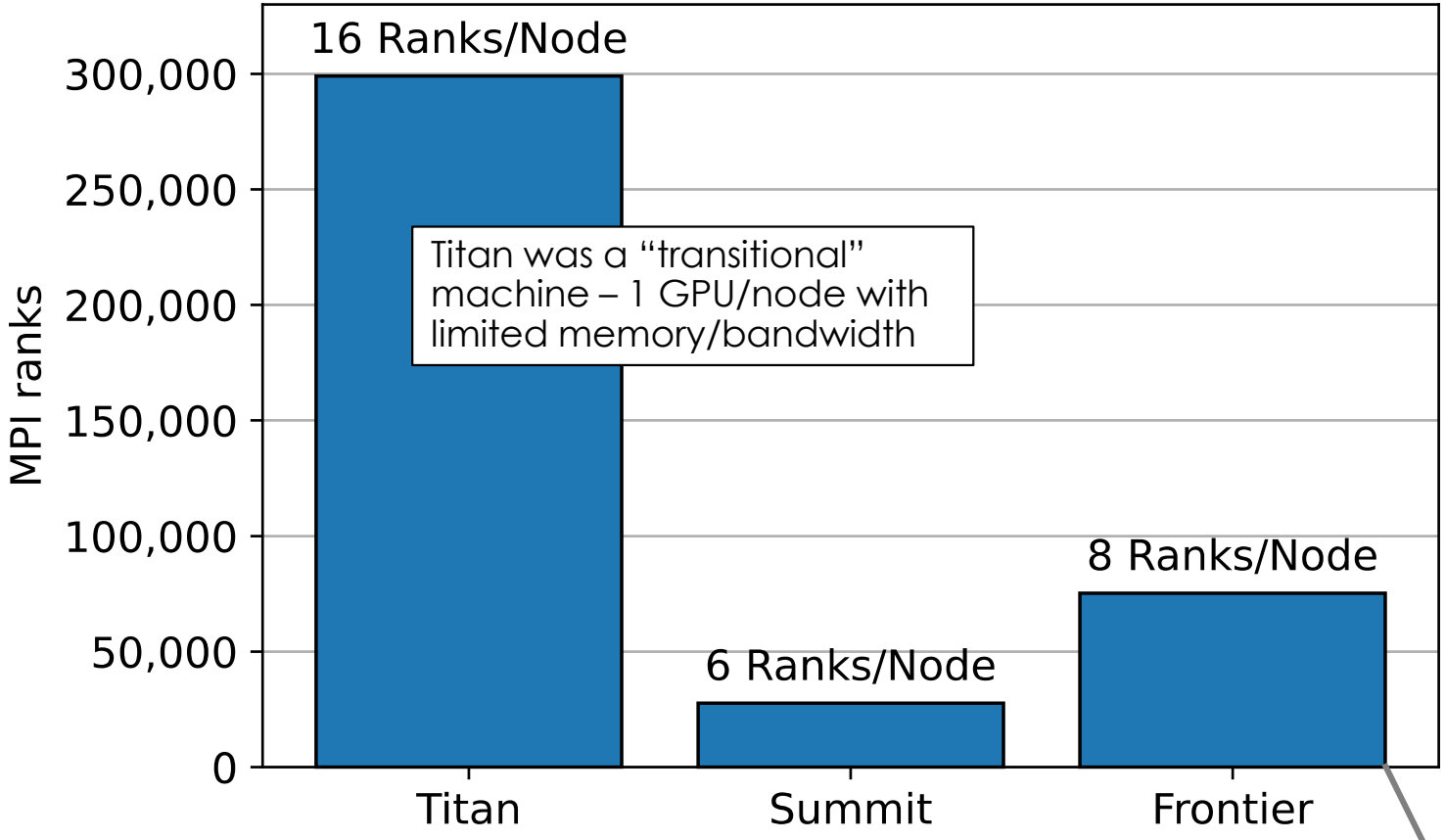


Peak performance



FLOPS by device

Getting performance on-node is the real challenge



Typical MPI runtime configurations

- We used to think of scaling as running O(100k) – O(1M) MPI ranks
- Starting in 2016 (Summit) the FLOPS per node has risen dramatically (48 TF)
- This focuses effort on "scaling in" instead of "scaling out"
- **Bottom line: we need to do more work per node on fewer MPI ranks**

AMD supports multiple ranks per node by launching each rank on its own stream – some applications have seen performance gains using this mode of operation

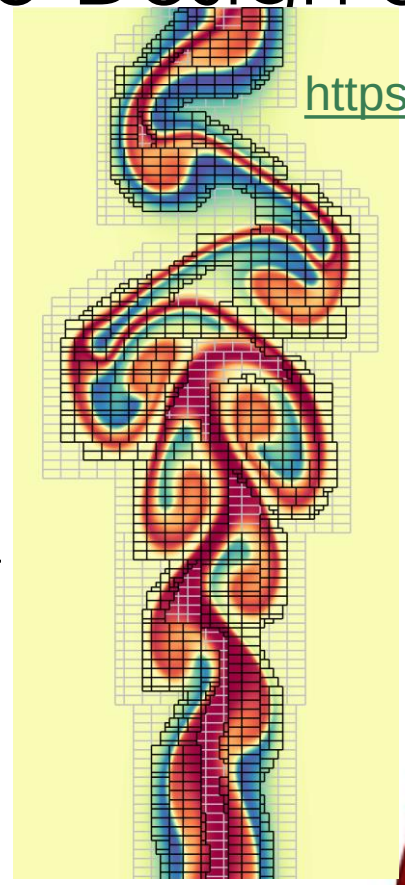
- NVIDIA supports this as well, but not by default

Where we started

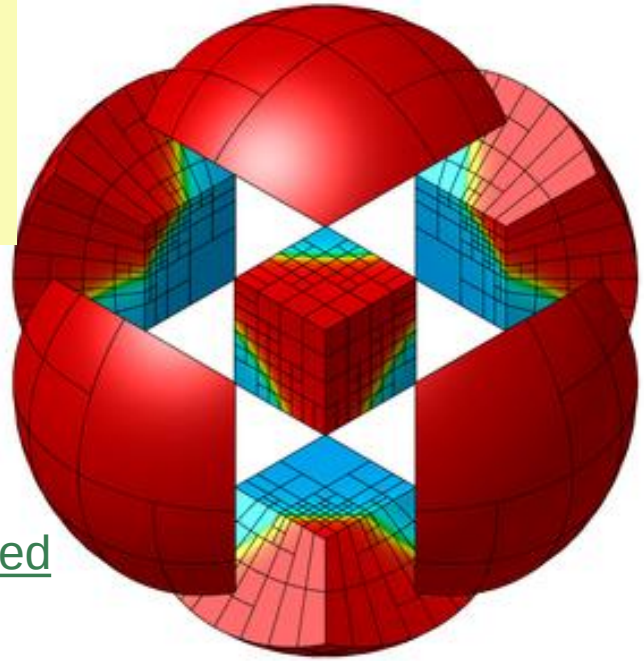
- **24 applications** and **6 co-design** projects
  - Including **62 separate codes**
  - Representing over **10 million lines of code**
  - Many supporting large user communities
  - Covering broad range of mission critical S&E domains
  - Mostly all MPI or MPI+OpenMP on CPUs at beginning of ECP
- Each project defines a domain-specific challenge problem for final benchmark
- Applications are evaluated in one of two categories
  - Performance – achieve a **50x** performance increase
  - Capability – utilize new architectures for expanded S&E

ECP Supporting Technologies : Co-Design and Software Technology

- Co-Design
  - Co-Design projects provide middleware support for ECP Applications projects
    - Much of the platform portability provided through co-design frameworks
  - **CEED**: Unstructured mesh + finite element methods (FEM)
  - **AMReX**: Block structured adaptive mesh refinement (AMR) with particle support
  - **CoPA**: particle mechanics + fast Fourier transforms (FFT)
- Software Technology
  - Libraries for encompassing wide-array of scientific computing services (E4S: e4s-project.github.io)
  - Linear solvers (Trilinos, PETSc, hypre, xSDK)
  - FFTs (FFTx, heFFTe)
  - Programming models (**Kokkos**)



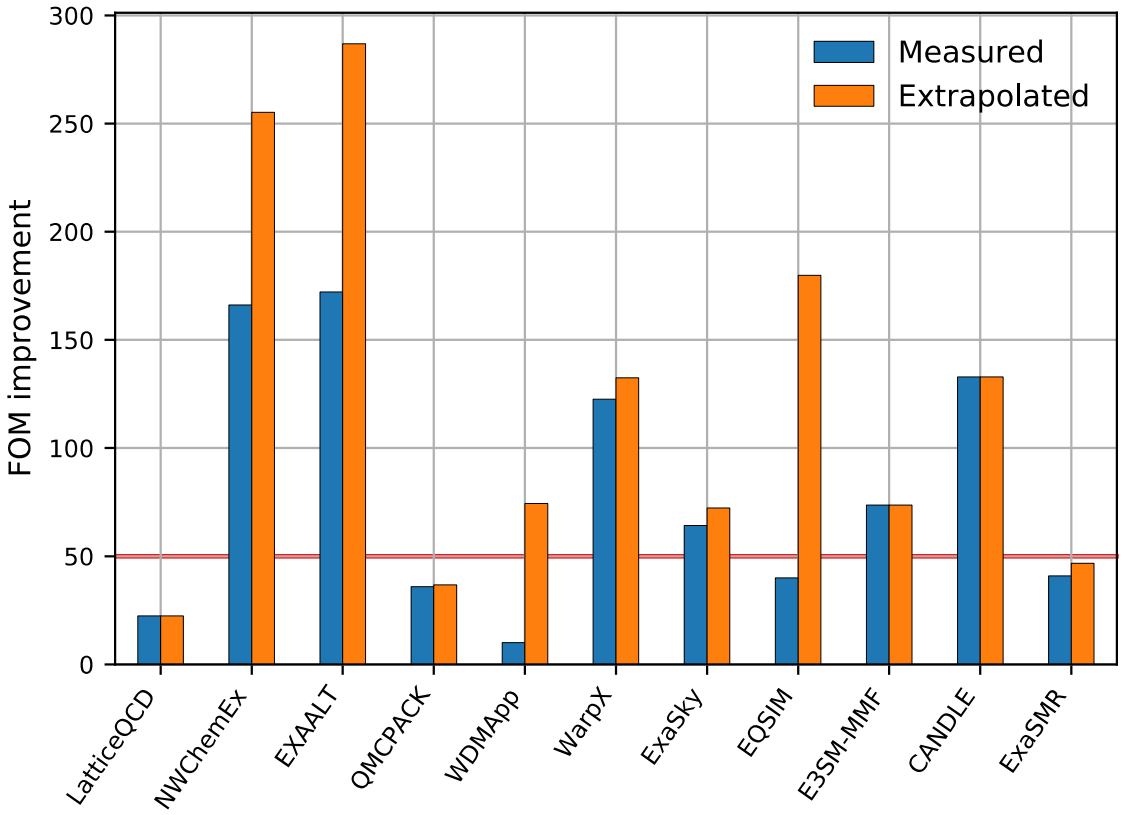
AMReX
https://amrex-codes.github.io/amrex



CEED
https://github.com/ceed

Where we are now

- All AD teams have all demonstrated progress on multi-GPU machines
  - many have already shown substantial performance on **Summit** or **Sierra**
- **Co-design Centers** have developed into **best practice** for large simulation projects.
- Refactoring code to run well on a heterogeneous machine has required **fundamental changes to data structures**, **data movement and algorithms** that could be made independently of specific accelerator features.
- AD projects are guinea pigs in exercising **performance portable programming** models



A. Siegel *et al.*, "Map Applications to Target Exascale Architecture with Machine-Specific Performance Analysis, Including Challenges and Projections," Exascale Computing Project, WBS 2.2, Milestone PM-AD-1110, ECP-U-AD-RPT_2021_00208, Mar. 2021. [Online]. Available: https://www.exascaleproject.org/reports.

T. Evans *et al.*, "A survey of software implementations used by application codes in the Exascale Computing Project," *Int. J. HPC Appl.*, DOI: 10.1177/10943420211028940, 2021. [Online]. Available: https://journals.sagepub.com/home/hpc.

“Porting” has included hierarchy of adaptations

Porting: Same high-level algorithm mapped to GPUs

- Rewrite, profile, and optimize
  - Generally preserve the exact answer
- Data Layout for memory coalescing
- Loop ordering
- Kernel flattening
- Increased locality
- Recomputing vs. storing
- Reduced branching
- Eliminating copies

General numerical strategies for GPUs

- Reduced communication
- Reduced synchronization
- Increased parallelism
- Reduced precision
- Optimized linear algebra

Domain-driven Adaptations

- Mathematical representation
- “On the fly” recomputing vs. lookup tables
- Prioritization of new physical models
- Alternate discretizations (high AI)
- Localized subgrid models
- Sparse → dense systems
- Defining weak scaling target
- Initial condition from ROM

Platform portability: challenges and characteristics

Positive characteristics

- Hot spot performance profile
- Simple loops and tasks
- SIMD(T) parallel paradigms
- Coalesced/ordered memory

Negative characteristics

- Flat performance profiles
- Deep functions
- Recursive algorithms
- Random access memory

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.

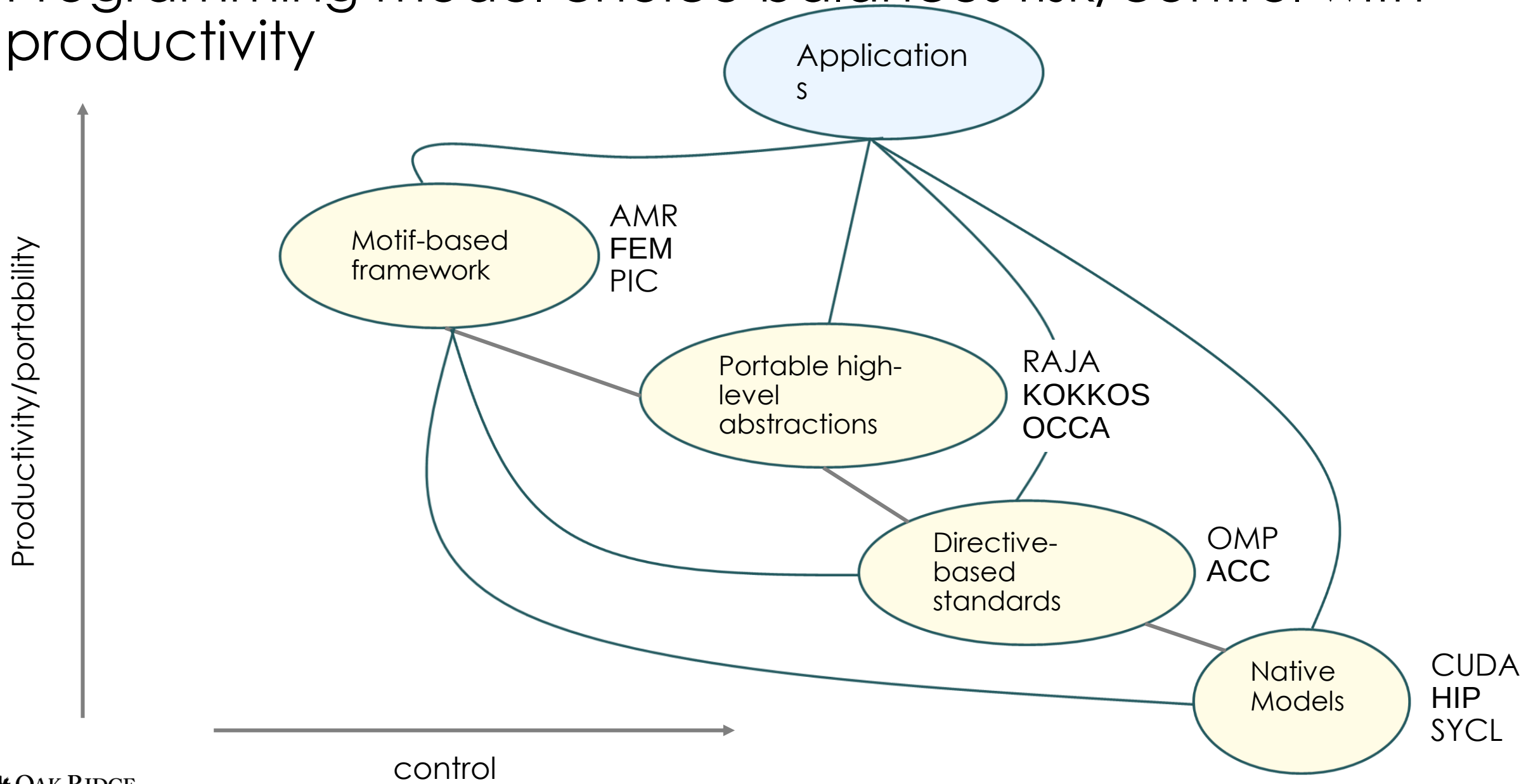
C++ abstractions

- 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

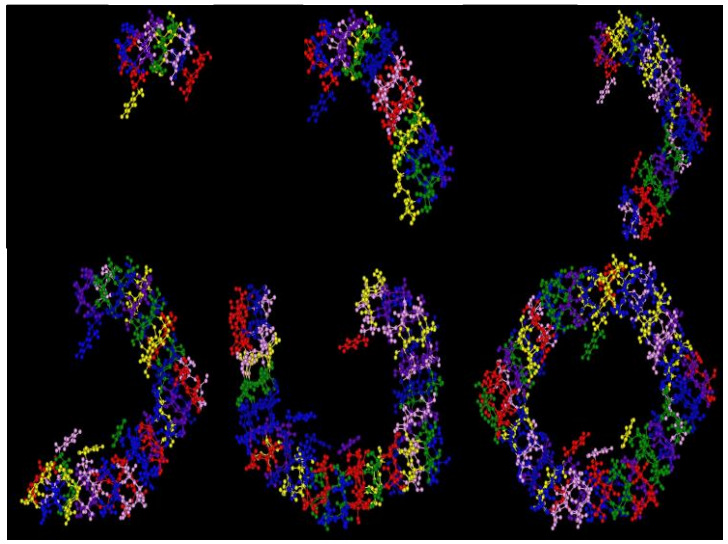
- Design application with a specific motif to use common software components
- Depend on co-design code (e.g. CEED, AMReX) to implement key functions on GPU.

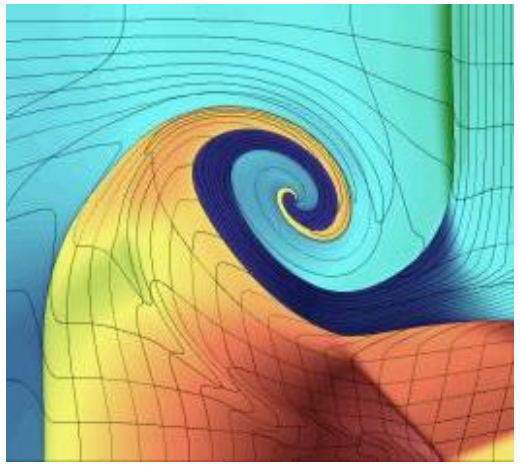
Programming model choice balances risk/control with productivity

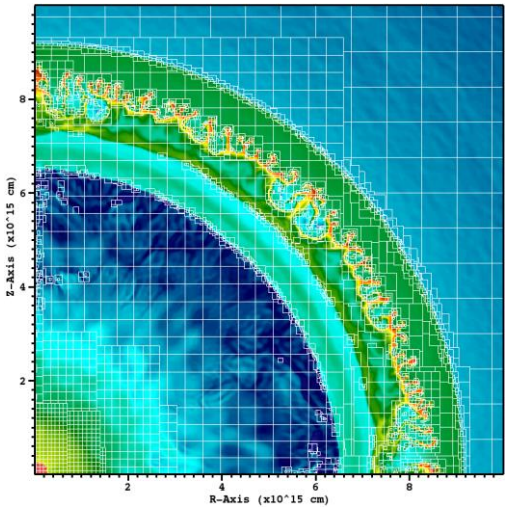


Domain Driven Adaptations critical for making efficient use of exascale systems in ECP

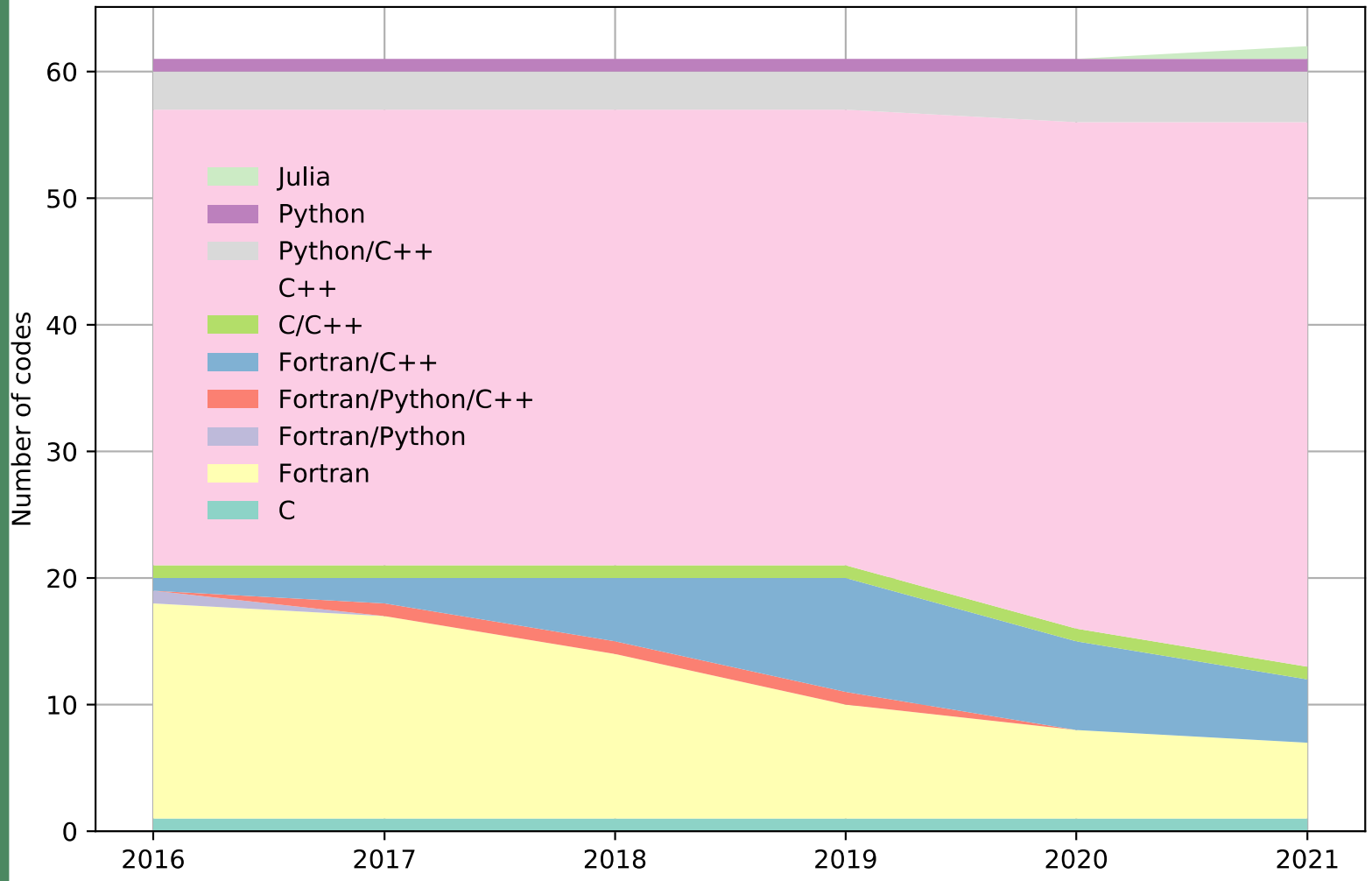
- Inherent strong scaling challenges on GPU-based systems →
  - Ensembles vs. time averaging
  - Fluid dynamics, seismology, molecular dynamics, time-stepping
- Increased dimensions of (fine-grained) parallelism to feed GPUs
  - Ray tracing, Markov Chain Monte Carlo, fragmentation methods
- Localized physics models to maximize "free flops"
  - MMF, electron subcycling, enhanced subgrid models, high-order discretizations
- Alternatives to sparse linear systems
  - Higher order methods, Monte Carlo
- Reduced branching
  - Event-based models

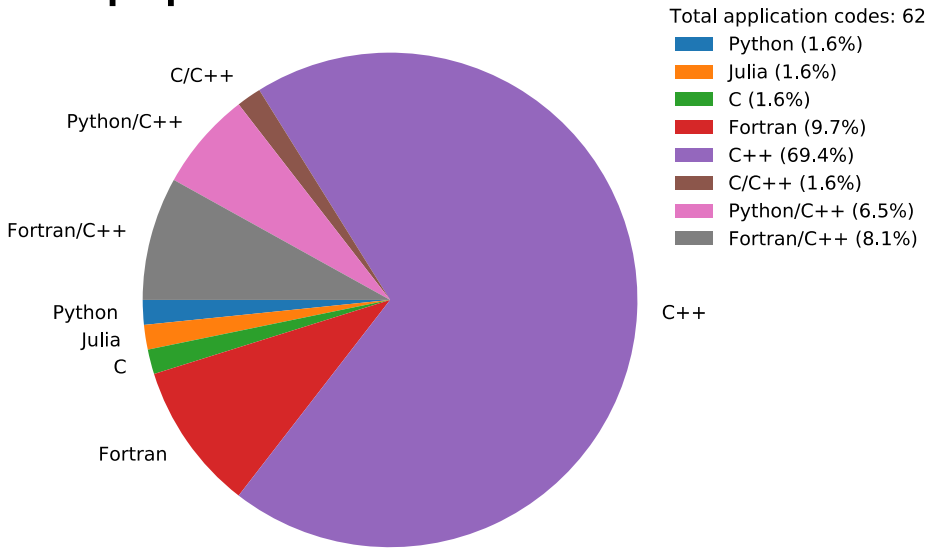






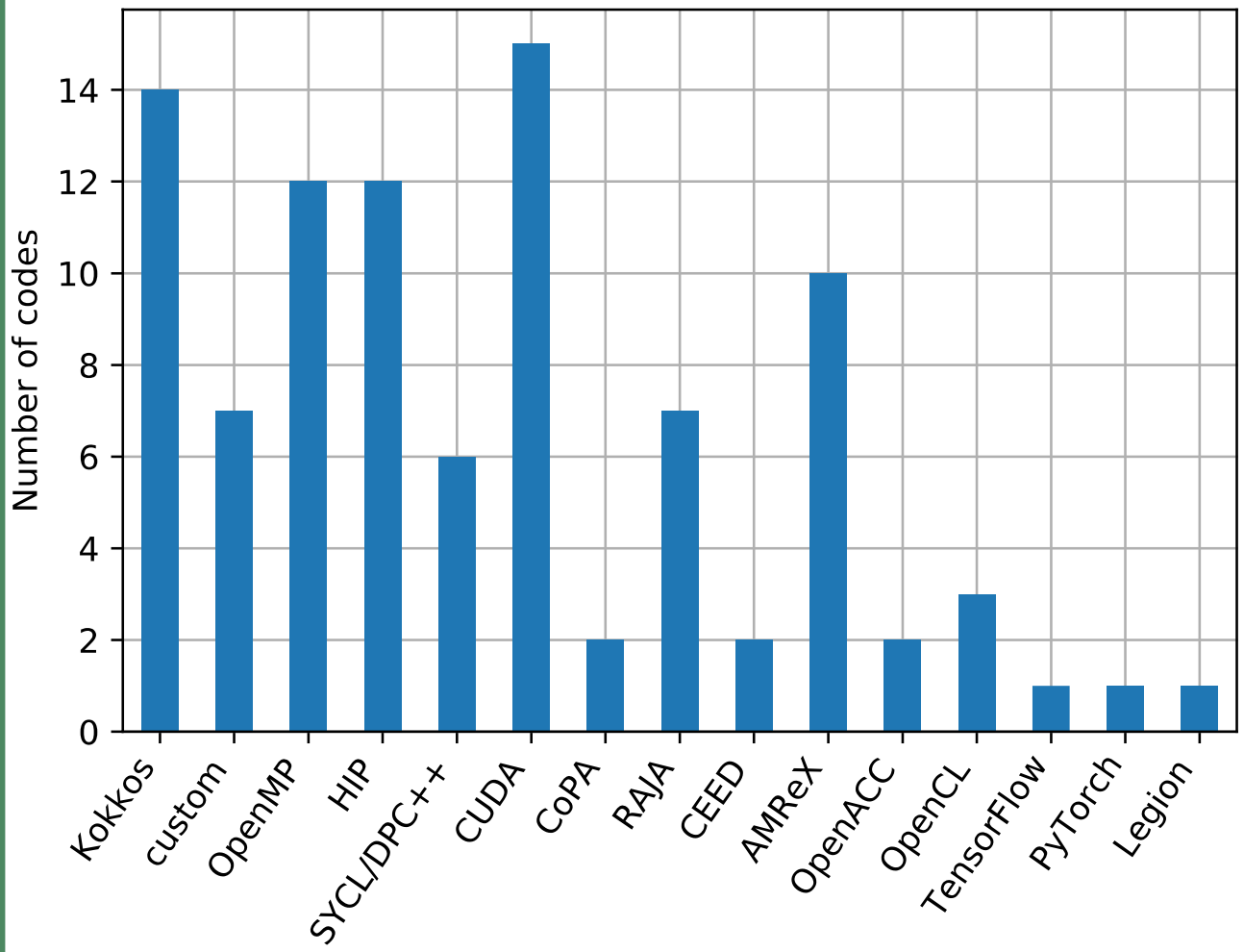
Programming languages used in ECP applications





- C++ has become dominant implementation language for HPC codes
- Fortran support for device code is currently lagging
  - 31% → 18%
  - Very little new Fortran written except in ~10% of codes
  - Directive-based programming models are only support for now (OpenACC)

Programming models used in ECP applications



Platform portability provided by co-design projects (CoPA, CEED, AMReX)	23%
ST programming models (Kokkos, RAJA, Legion)	35%
Directive-based programming models: (OpenMP, OpenACC)	23%
Native (CUDA/HIP/SYCL) or custom implementations	**55%**

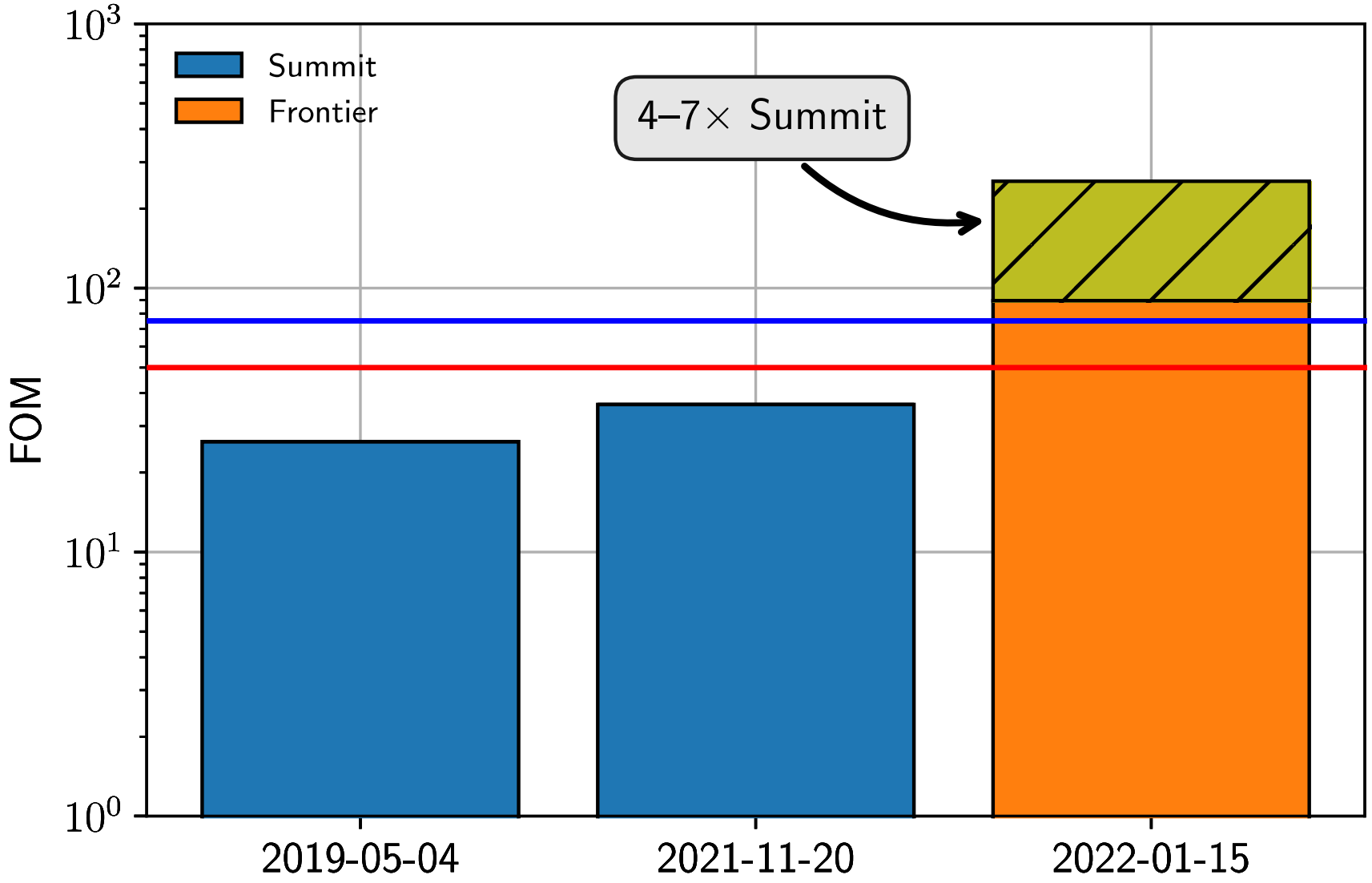
Applications use multiple models – these are not normalized to one

Some observations based on development in the ECP

- C++ is the dominant language on advanced architectures
  - Most applications are doing new development in C++
  - Legacy Fortran still in use
- Use of co-design/ST technologies provide significant benefit
  - Fine-scale architectural details provided by co-design frameworks
- **Large percent of custom implementations reflects difficulty of universal platform-portable programming models that span diverse scientific applications**

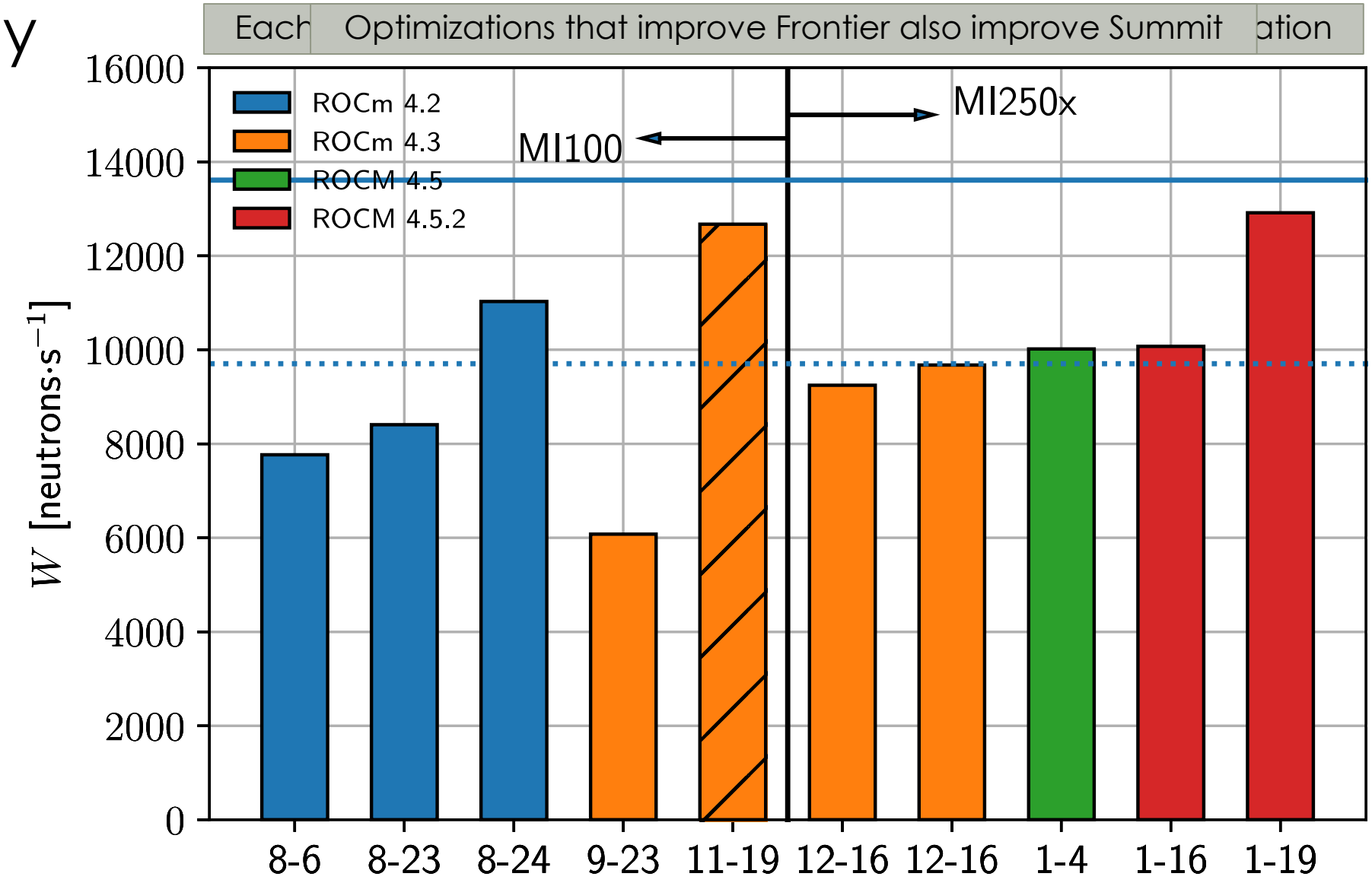
Experiences on AMD/Frontier

The projected end-state for ECP on Frontier looks good



ECP ExaSMR Shift Monte Carlo neutron transport simulation

Getting performance on AMD has not been a linear journey

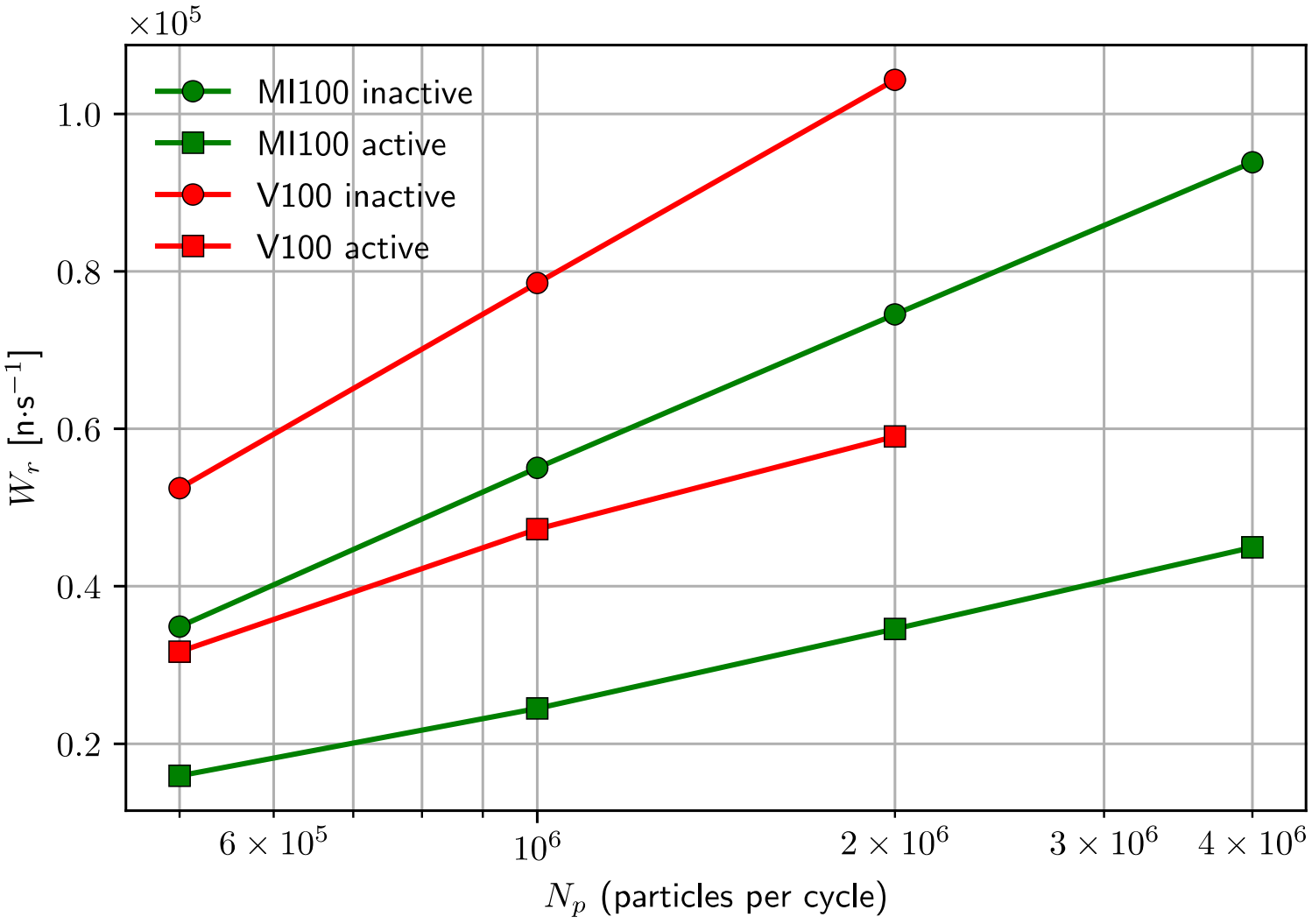


ECP ExaSMR Shift Monte Carlo neutron transport simulation

The state of the AMD software/hardware stack is under constant development

- Hardware
  - MI250x has 2x (per GCD) performance at compute bound (FP64) relative to M100
  - MI250x has 1.3x (per GCD) performance at bandwidth bound relative to MI100
  - MI100 has 120 compute units, the MI250x has 110 (per GCD)
  - MI250x does not have thread independent program counters – Nvidia V100/A100 does
  - The AMD L1 Caches are 16kB (2 – 4x smaller) and ~5x slower than Nvidia V100 L1 cache
  - **These features will have disproportionate impact on *latency* bound applications**
    - e.g., Sparse PDE solvers, upwind schemes and hyperbolic systems, implicit time-discretizations, etc.
- Compilers
  - Register/spill/stack size/occupancy performance varies significantly between versions
  - Manual interrogation and tuning has been required

So, back to our initial runs on the MI100



$$W_r = \frac{\text{neutrons}}{\text{wall clock time}}$$

Higher is better

Single GPU Comparison of Shift on Spock (AMD) vs Summit (Nvidia V100)

Optimizing register usage

Examine ISA (.s) using --save-temps

Kernel	Vgpr	Spills	Occupancy
dist_to_collision	64	4734	4
sample_fission	64	8967	4
create_fission_sites	64	4171	4
process_collision	64	41087	4
calc_macro_rxn	64	32870	4
calc_macro_mt	64	2390	4
calc_partial_macro_rxn	64	14200	4
calc_partial_macro_mt	64	1894	4
calc_micro_rxn	64	14141	4
calc_micro_mt	64	1898	4
calc_micro_stateless	64	1823	4

→

Kernel	Vgpr	Spills	Occupancy
dist_to_collision	133	0	1
sample_fission	132	0	1
create_fission_sites	133	0	1
process_collision	146	0	1
calc_macro_rxn	133	0	1
calc_macro_mt	130	0	1
calc_partial_macro_rxn	131	0	1
calc_partial_macro_mt	128	0	2
calc_micro_rxn	131	0	1
calc_micro_mt	127	0	2
calc_micro_stateless	127	0	2

__launch_bounds__(256)

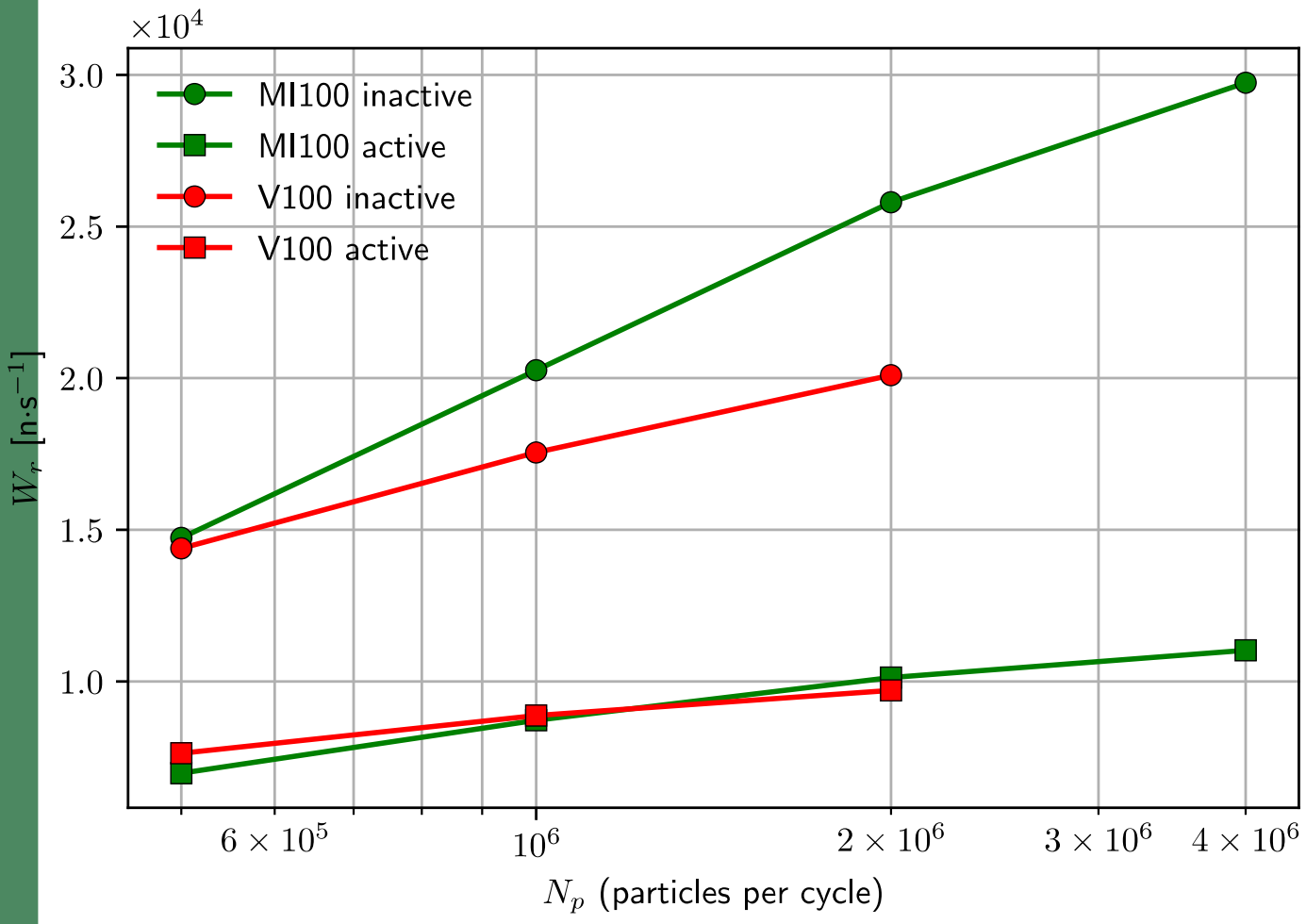
→

Kernel	Vgpr	Spills	Occupancy
dist_to_collision	122	0	2
sample_fission	128	3577	2
create_fission_sites	128	379	2
process_collision	128	39253	2
calc_macro_rxn	128	316	2
calc_macro_mt	110	0	2
calc_partial_macro_rxn	102	0	2
calc_partial_macro_mt	128	0	2
calc_micro_rxn	101	0	2
calc_micro_mt	127	0	2
calc_micro_stateless	127	0	2

__launch_bounds__(256,2)

- Revert 4 spilling kernels to __launch_bounds__(256,2) and accept lower occupancy
- Need to experiment with calc_macro_rxn parameters
  - This is a performance–impacting kernel
  - Eating spills may be worth improvement in occupancy

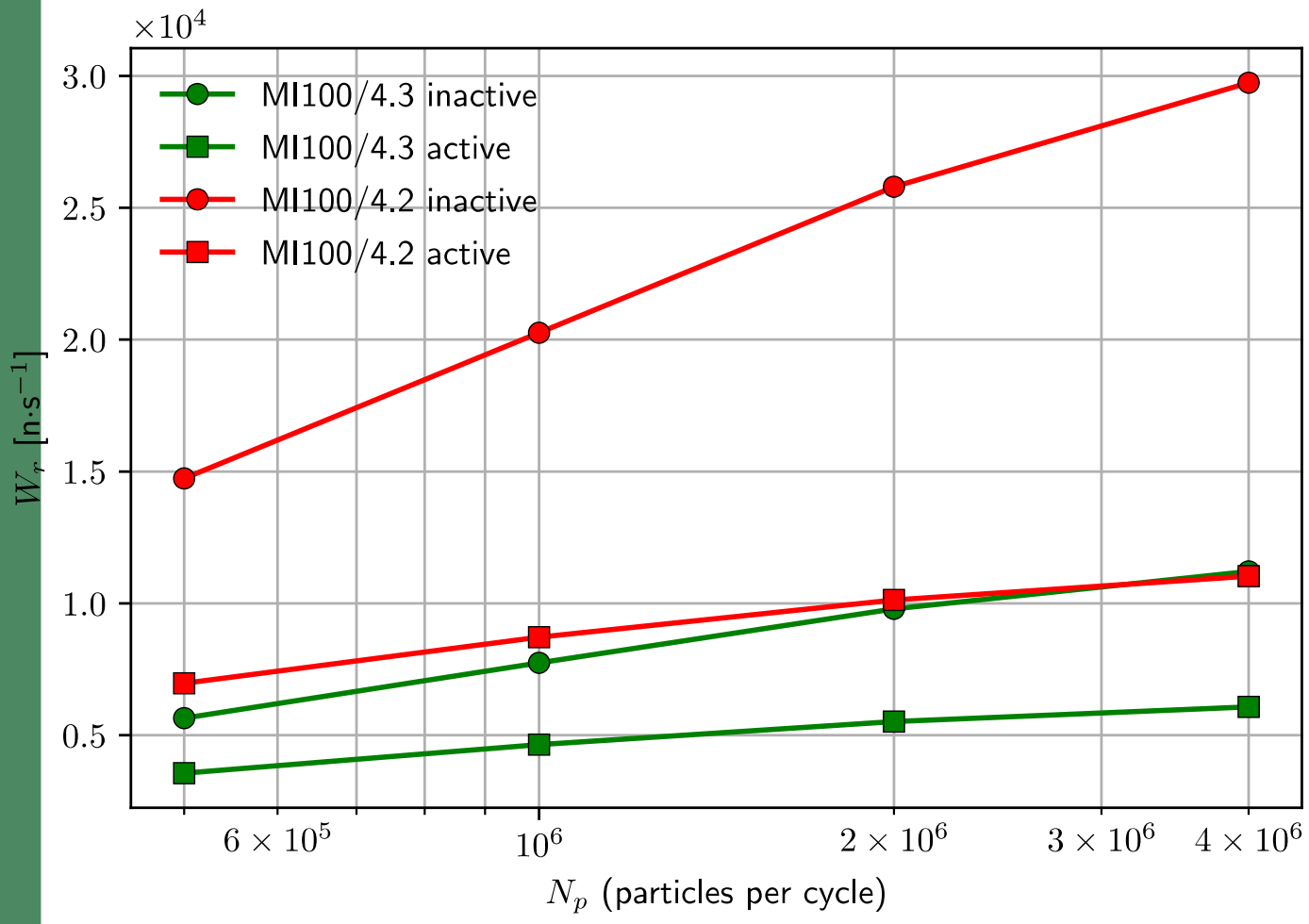
We achieve performance on the MI100 – ROCm 4.2



Kernel	threads/block	warps/EU	Vgpr	Spills	Occupancy
dist_to_collision	256	2	122	**0**	**2**
sample_fission	256	1	132	0	1
create_fission_sites	256	1	133	0	1
process_collision	256	1	146	0	1
calc_macro_rxn	256	2	128	**316**	**2**
calc_macro_mt	256	2	110	0	2
calc_partial_macro_rxn	256	2	102	0	2
calc_partial_macro_mt	256	1	128	0	2
calc_micro_rxn	256	2	101	0	2
calc_micro_mt	256	1	127	0	2
calc_micro_stateless	256	1	127	0	2

We accept spills in calc_macro_rxn because the increased occupancy gives an overall benefit

ROCm 4.3 drops – and its back to the drawing board



Kernel	threads/block	warps/EU	Vgpr	Spills	Occupancy
dist_to_collision	256	2	128	**4341**	**2**
sample_fission	256	1	142	0	1
create_fission_sites	256	1	139	0	1
process_collision	256	1	168	0	1
calc_macro_rxn	256	2	128	**34516**	**2**
calc_macro_mt	256	2	125	0	2
calc_partial_macro_rxn	256	2	113	0	2
calc_partial_macro_mt	256	1	128	0	2
calc_micro_rxn	256	2	112	0	2
calc_micro_mt	256	1	127	0	2
calc_micro_stateless	256	1	127	0	2

ROCm 4.3 vs ROCm 4.2 with same launch bound settings

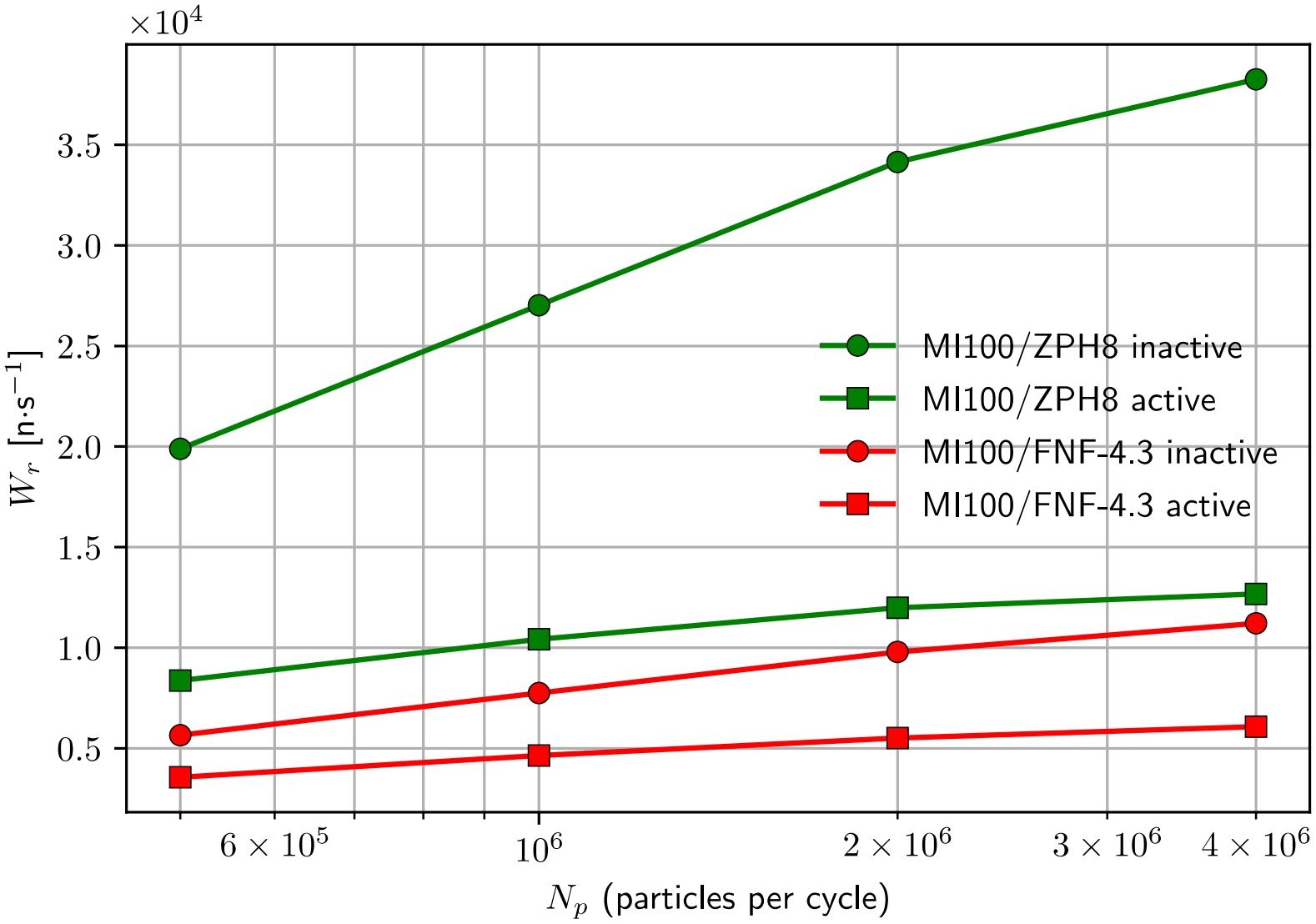
Algorithmic changes were required to address this issue

```
__device__ Complex_Dbl operator()(Complex_Dbl z) const
{
   // . . .
   if (thrust::abs(z) <= 6.0)
   {
      Complex_Dbl prefactor = 8.124330e+01 * d_i;
      double an[24]; // . . .
      double neg_1n[24] // . . .
      double denominator_left[24]; // . . .
      Complex_Dbl eterm  = thrust::exp(d_i * 12. * z);
      Complex_Dbl factor = 144.0 * z * z;
      Complex_Dbl w      = d_i * (1.0 - eterm) / (12. * z);
      Complex_Dbl sum = 0;
      for (int n = 0; n < 24; n++)
      {
         Complex_Dbl top = neg_1n[n] * eterm - 1.;
         Complex_Dbl bot = denominator_left[n] - factor;
         sum += an[n] * (top / bot);
      }
      w += prefactor * z * sum;
      return w;
   }
   // . . .
}
```

```
__device__ Complex_Dbl operator()(Complex_Dbl z) const
{
   z += Complex_Dbl(0.9 * d_i);
   const auto zz = z * z;
   const Complex_Dbl aa8[8] = {+11.7559071436993,
      -32.310199761603 * d_i,
      -21.9357456686406,
      31.490536152863 * d_i,
      6.75847413957232,
      -8.07354660639634 * d_i,
      -0.507771291744591,
      0.564189504758109 * d_i};
   const Complex_Dbl bb8[5] = {6.5625, -52.5, 52.5, -14.0, 1.0};
   return (((((((aa8[7] * z + aa8[6]) * z + aa8[5]) * z + aa8[4]) * z
      + aa8[3]) * z
      + aa8[2]) * z
      + aa8[1]) * z
      + aa8[0])
      / ((((bb8[4] * zz + bb8[3]) * zz + bb8[2]) * zz + bb8[1]) * zz
      + bb8[0]);
}
```

Replace Faddeeva function evaluation with complex exp function calls with polynomial approximation

And we have recovered the performance we were seeing in ROCm4.2



Moral(e) of story

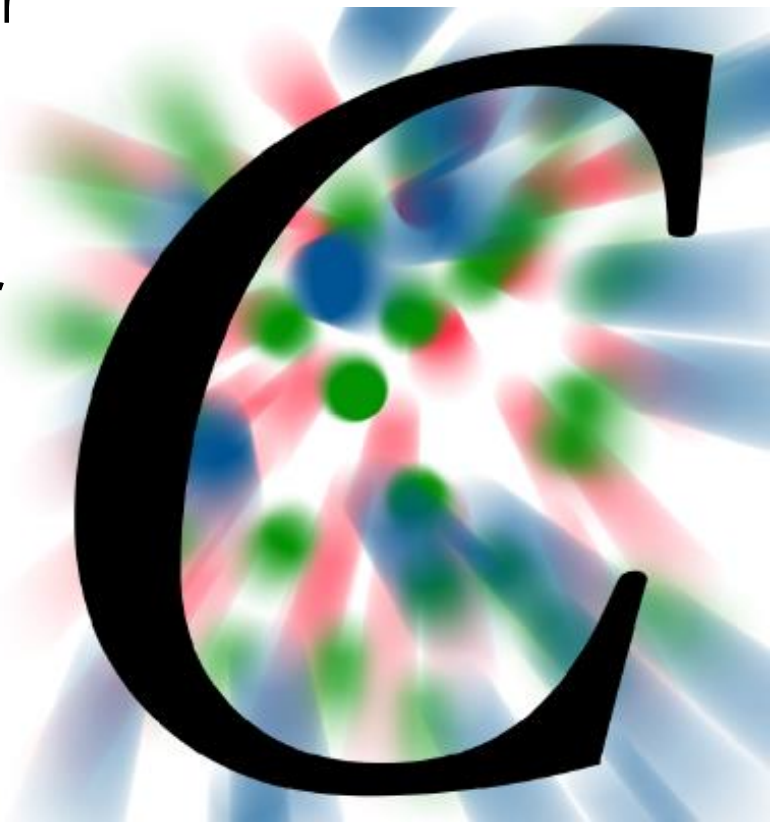
- The AMD hardware/software stack is considerably less mature than NVIDIA
- Each new compiler and hardware upgrade has required looking deeply at performance behavior
- New algorithmic changes will generally increase performance on NVIDIA hardware as well (in science terms – this is a good thing!)
- We expect the situation to improve
  - As the ROCm stack becomes more mature, it will do a better job of optimizing on their newest hardware (MI250x)
  - This is probably the only way most applications will "separate" from NVIDIA levels of performance
  - Many applications will probably see something in the range of 2-4x improvement over Summit instead of 4-7x.

Frontier will provide significant performance beyond Summit - ECP applications have worked through many issues that should make it easier for follow-on applications to achieve performance

Development of HEP Applications for use at the OLCF

Motivation and Scope

- US DOE effort to accelerate full-fidelity MC detector simulation for LHC on GPU
  - Insufficient CPU resources for LHC requirements
  - **Motivated by successful GPU refactor of nuclear reactor MC transport (ExaSMR) via the Exascale Computing Project**
- Goal: full standard physics in time for run 4
  - Standard EM physics (e-, gamma, e+) currently
  - VecGeom for CUDA detector navigation



Celeritas team represents cross-discipline group from ECP (ExaSMR) and HEP

Overview

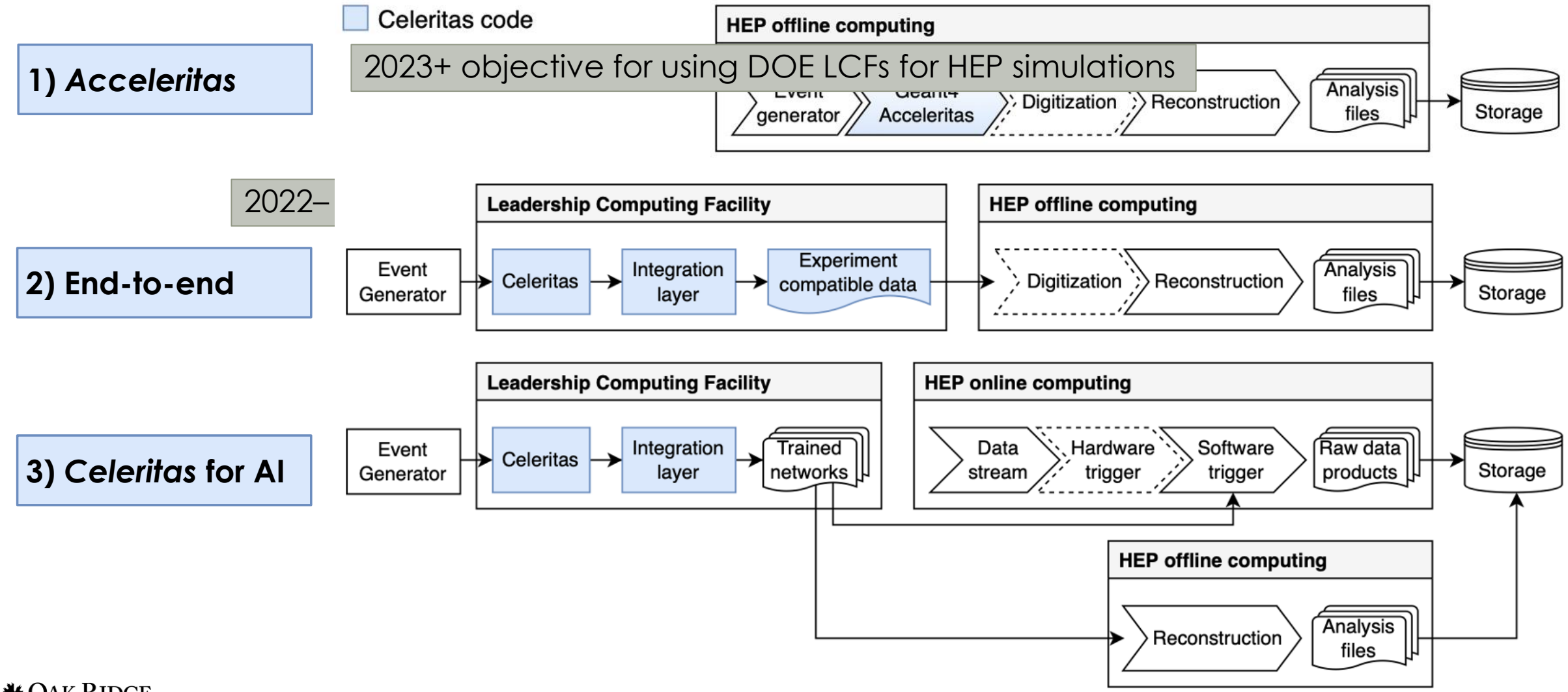
- **GPU-targeted** re-implementation of a subset of Geant4 physics leveraging both HEP physics community and HPC/GPU particle transport domain knowledge
  - Grew out of ECP Pilot project with cost-sharing from HEP
  - Currently funded through ASCR/HEP collaboration
  - 2022 DOE SciDAC
- First code committed June 2020
- Short-term application: offloading EM tracks from Geant4 to GPU (***Acceleritas*** bridge library)
- Long-term application: direct access library for higher performance integration into LHC analysis workflows

Motivation for ORANGE GPU geometry engine

- *Shift* GPU Monte Carlo: need more complex models for ex-core dosimetry and advanced reactors
- Celeritas: current VecGeom meets long-term goals (support for Geant4 detector models) but not some short-term needs
  - Portability to run on different DOE HPC machines
  - Developer and CI friendliness (CUDA RDC requirement: warnings, build times, build size)
- Good practice to try multiple approaches to cutting-edge research problems

This is now a collaborative effort between Celeritas and AdePT

Integration with experiments



Physics – current DOE emphasis on EM physics through Acceleritas

Implemented

Particle	Process	Model(s)
γ	photon conversion	Bethe–Heitler
	Compton scattering	Klein–Nishina
	photoelectric effect	Livermore
	Rayleigh scattering	Livermore
$e^{\pm}$	ionization	Møller–Bhabha
	bremsstrahlung	Seltzer–Berger, relativistic
	pair annihilation	EPlusGG
	multiple scattering	Urban, WentzelVI
$\mu^{\pm}$	muon bremsstrahlung	Muon Bremsstrahlung

Complete validations are still ongoing

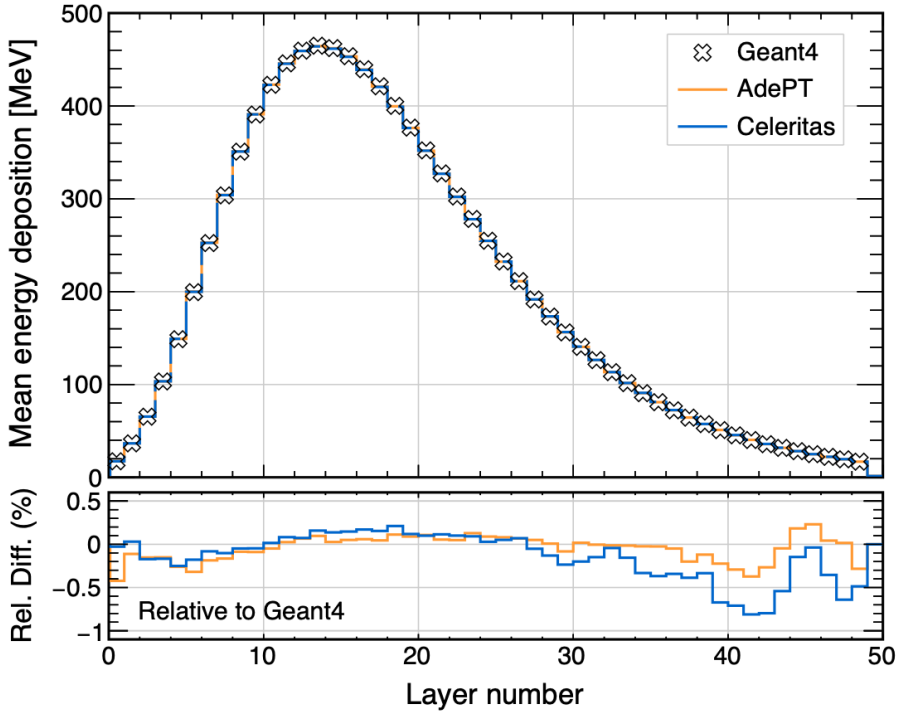
Planned

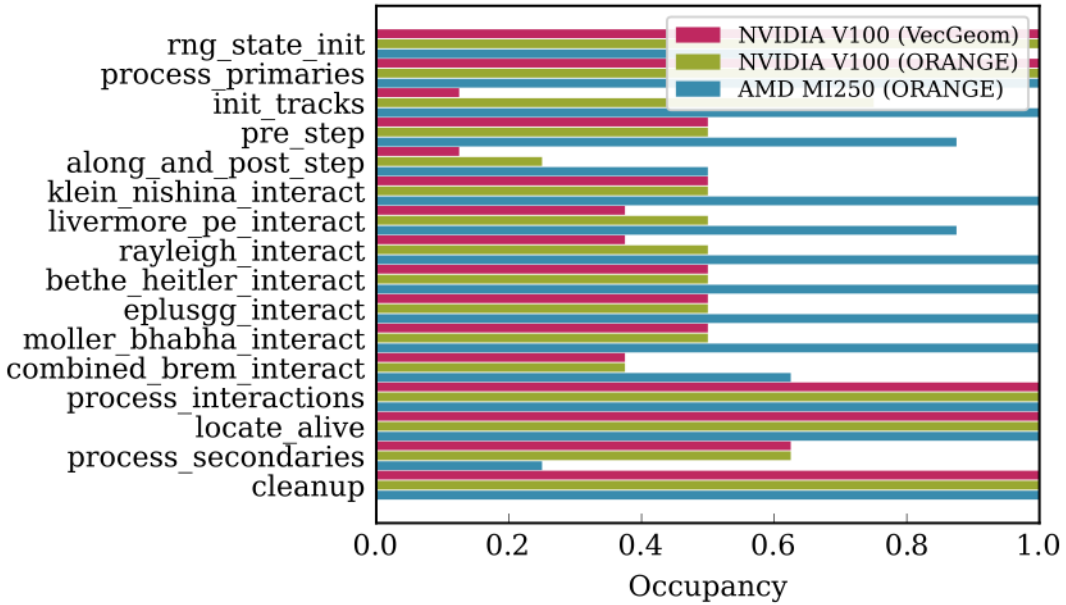
Physics	Process	Particle(s)
EM	photon conversion	γ
	pair annihilation	$e^{\pm}$
	photoelectric effect	γ
	ionization	charged leptons, hadrons, and ions
	bremsstrahlung	charged leptons and hadrons
	Rayleigh scattering	γ
	Compton scattering	γ
	Coulomb scattering	charged leptons, hadrons
	multiple scattering	charged leptons, hadrons
	continuous energy loss	charged leptons, hadrons, and ions
Decay	two body decay	$\mu^{\pm}$, $\tau^{\pm}$, hadrons
	three body decay	$\mu^{\pm}$, $\tau^{\pm}$, hadrons
	n-body decay	$\mu^{\pm}$, $\tau^{\pm}$, hadrons
Hadronic	photon-nucleus	γ
	lepton-nucleus	leptons
	nucleon-nucleon	p, n
	hadron-nucleon	hadrons
	hadron-nucleus	hadrons
	nucleus-nucleus	hadrons

Current results

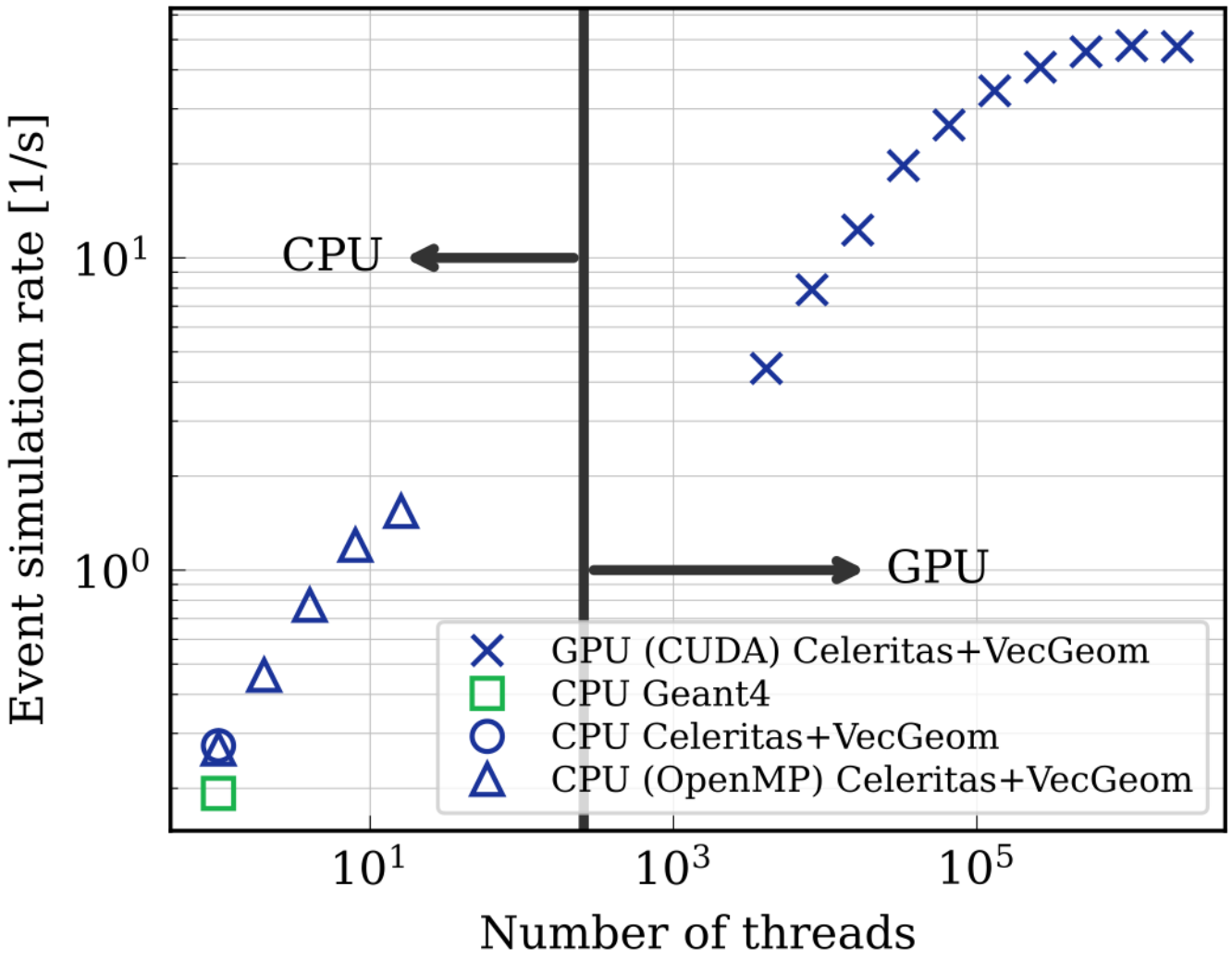
- TestEm3 (100k particles; 1GeV e^-; 50×0.8 cm lAr/Pb slabs)
- Summit: GPU to CPU-only speedup per node: **69.5×**
  - 6 NVIDIA V100 GPUs and 42 Power9 CPU cores
  - Celeritas with ORANGE: CUDA vs OpenMP
- Agreement with Geant4 to ~0.5% for this test problem
  - Physics validation in progress
- Platform-portable geometry (ORANGE)
  - Verified on AMD GPUs (MI250x, ROCm 4.5)
  - Performance testing in progress

We are working with our AdePT colleagues this week to standardize performance and benchmark standards





Preliminary results



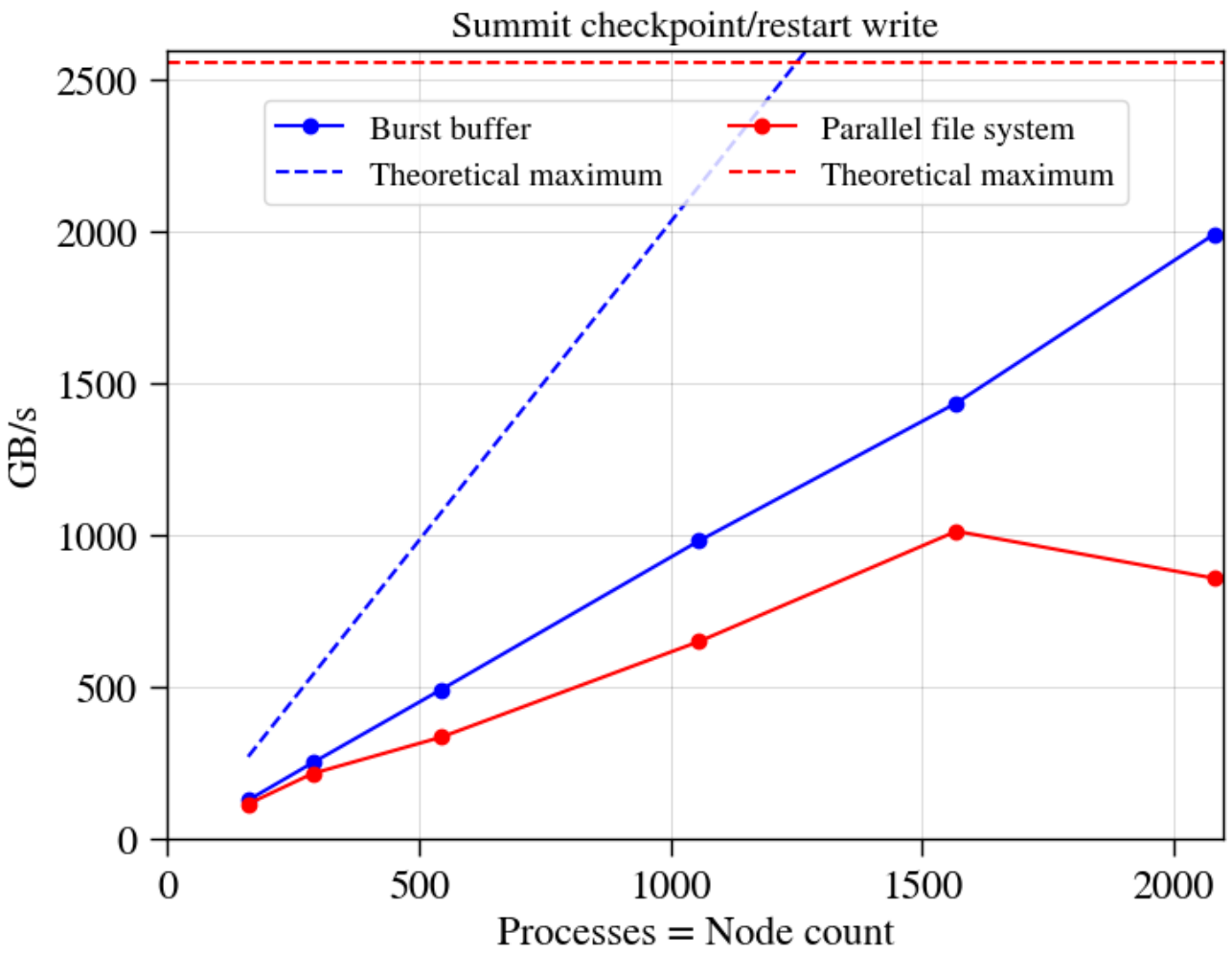
Application	Execution	Speedup
Geant4 (v10.7)	CPU (serial)	1
Celeritas	CPU (serial)	1.4
Celeritas	CPU (OpenMP)	40
Celeritas	GPU	280

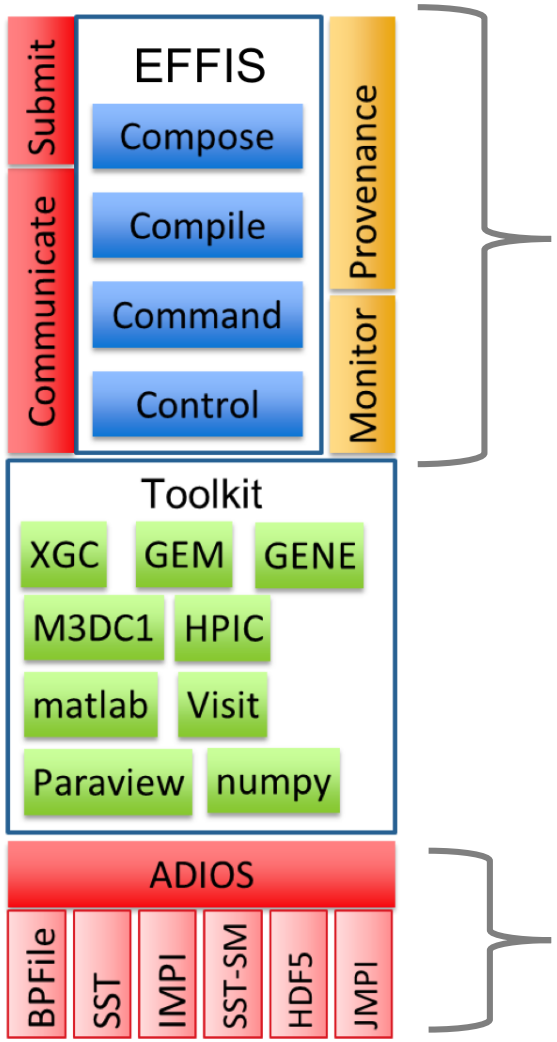
- **Geant4 scales linearly**
- 30 cores ≈ 30× serial execution
- Single NVIDIA V100 ≈ 280 cores

CPU Intel Xeon Gold 5218 @ 2.3 GHz (Cascade Lake)
GPU NVIDIA V100 @ 1.53 GHz (80 symmetric multiproc. 64 cores each 16 GB of memory)

CUDA 11.5 -O3 --use_fast_math
GCC 8.5 -O3 -march=skylake-avx512 -mtune=skylake-avx512

ECP-generated technologies support runtime and high-speed I/O on LCF computers





Runtime workflows

- Partition jobs across nodes
- Fault tolerance
- I/O management and accumulation

Interface to ROOT

Initial integration/testing of EFFIS/ADIOS part of Celeritas 2023-2024 SciDAC

Final observations

- Frontier will be a viable platform for HEP detector simulations at scale
  - Other applications will benefit as well (HEP-CCE is preparing event generators and other applications to run at DOE LCFs)
- Partnership of Celeritas and AdePT has many potential benefits
  - Collaboration on ORANGE Monte Carlo geometry engine
  - Standards for LHC Monte Carlo benchmarks for physics and performance analysis

Next Step: demonstrate advantages of using GPUs – and OLCF – for HEP detector simulations
Future: Enable full use of OLCF for end-to-end experimental workflows

Questions?