NESAP Project: XGC & WDMApp on Perlmutter

A. Scheinberg[1], P. Lin[2], S. Ku[3], K. Huck[4], S. Ethier[3],
D. Kulkarni[2], CS Chang[3]

NERSC GPUs for Science Day

October 25, 2022

[1]Jubilee Development
[2]Lawrence Berkeley National Laboratory
[3]Princeton Plasma Physics Laboratory
[4]University of Oregon

XGC introduction

- Tokamak plasma physics code specializing in edge physics and realistic geometry
- Gyrokinetic (i.e. 6D → 5D via analytic reduction using gyro-averaging)
- Particle-in-cell with an **unstructured 2D grid** and **structured toroidal dimension**
- Domain decomposition: toroidally sliced, then each MPI rank handles a subset of the grid

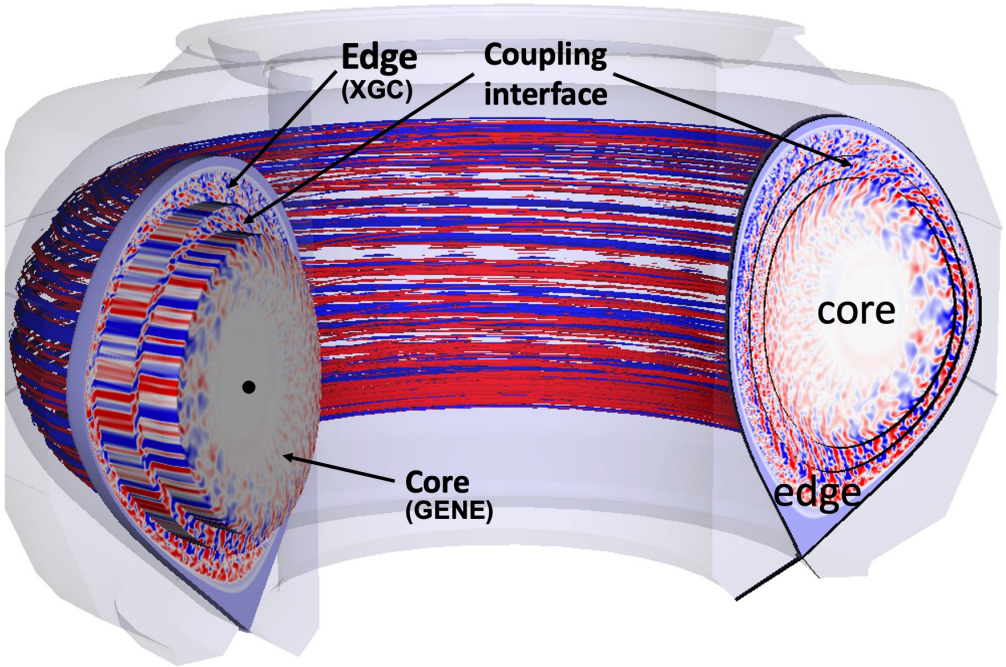
Charge scatter

Field solve

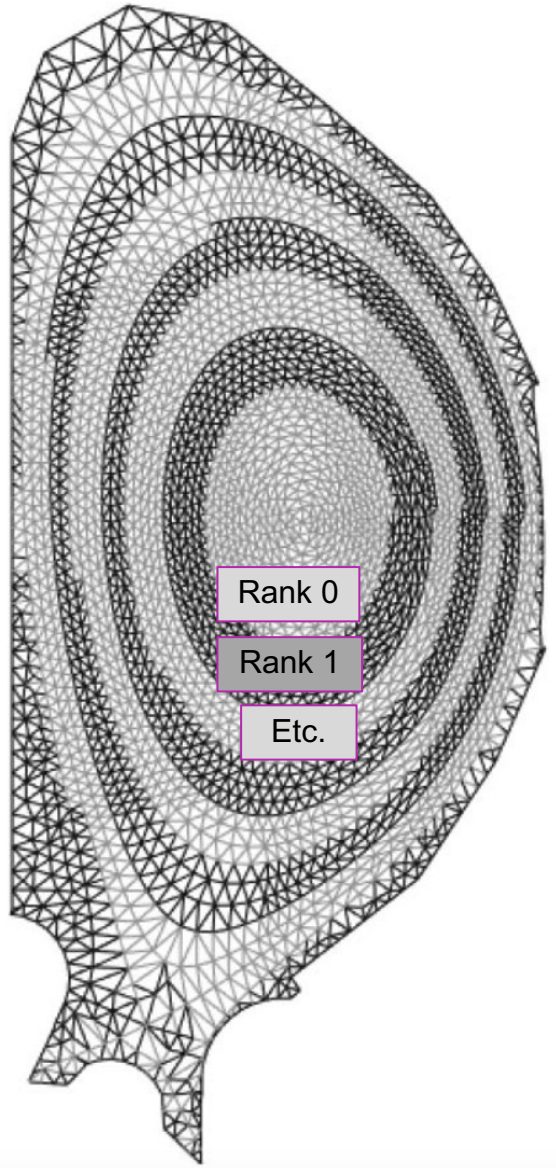
Electron push (x6-60)
Ion push

Shift particles to domain

Collisions
Sources
Diagnostics

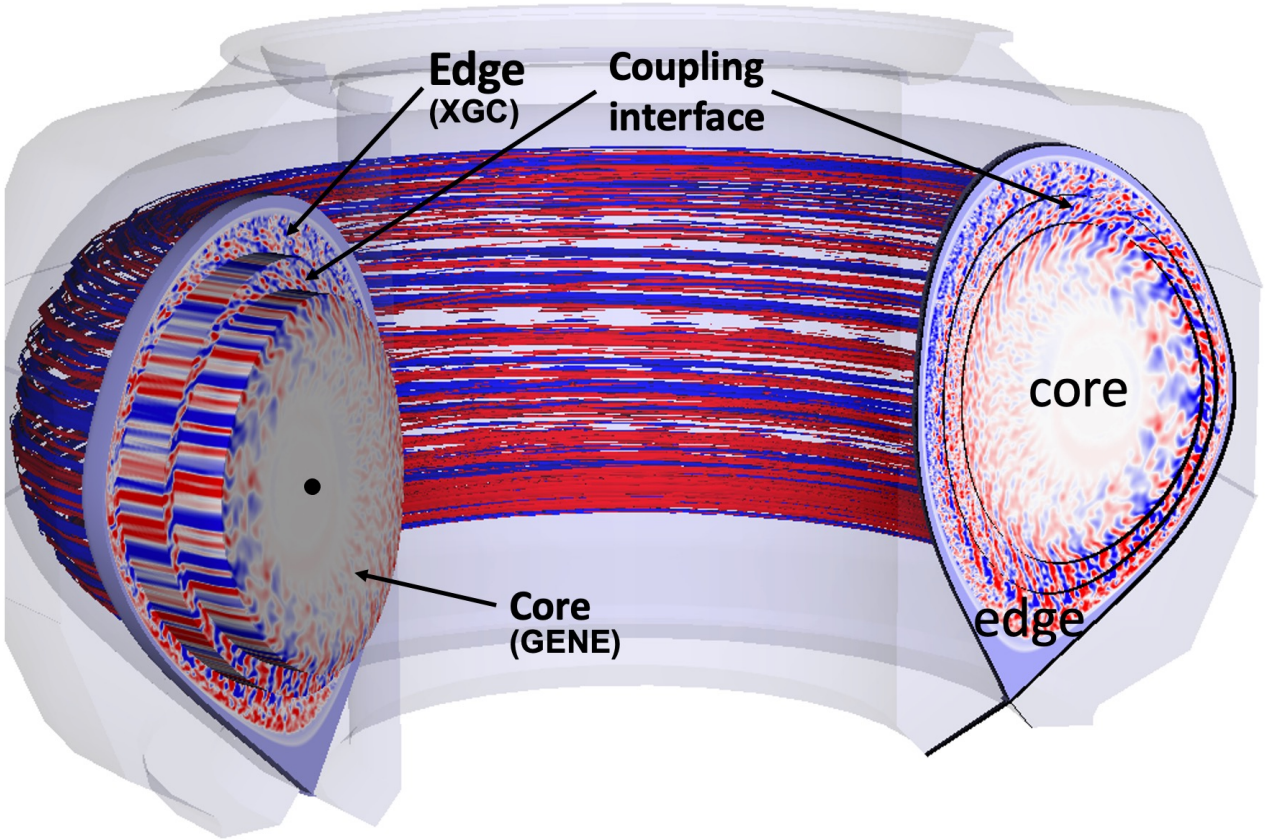


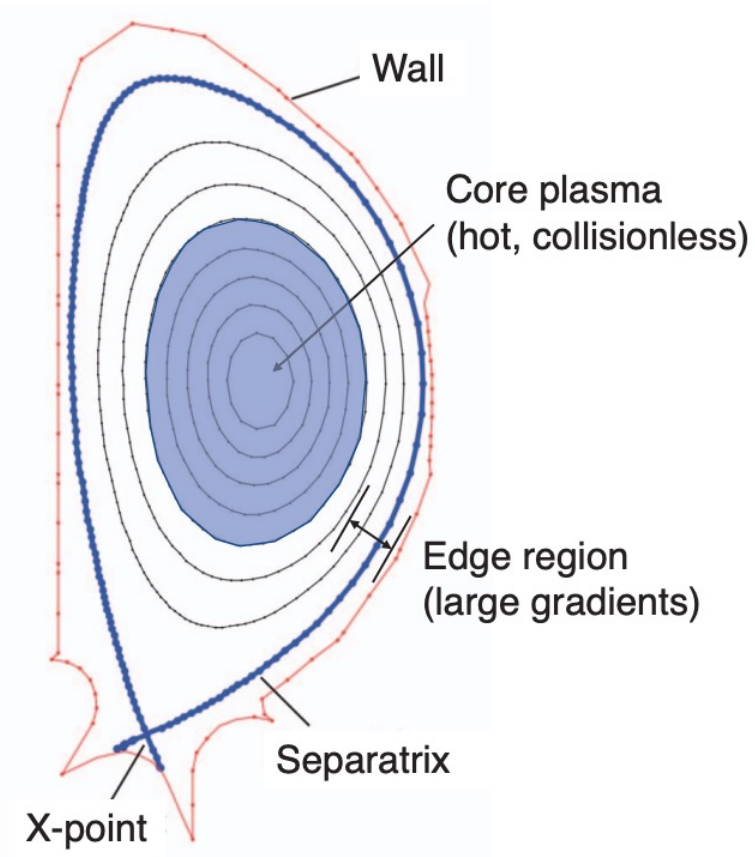
Tokamak cross-section



ECP and the Whole Device Model (WDMApp)

- The WDMApp is an Exascale Computing Project (ECP) application
- Couples XGC with a core code (GENE or GEM) for ”whole device modeling”
- The vast majority (>90%) of time spent is spent in XGC, so its optimization is most critical





XGC engineering challenges

- A wide array of physics features and modes must be supported, e.g.:
  - Delta-f (perturbation from Maxwellian) and full-f
  - Electrostatic (magnetic field perturbations due to plasma ignored) and electromagnetic
  - Axisymmetric (“XGCa”)
  - Impurities
  - Neutral particles with atomic cross-sections
  - Coupling (GENE, GEM, XGC, in-situ analysis)
- These different modes of operation can drastically alter landscape of performance bottlenecks
- Physics in constant state of development
  - Some changes are modular additional features
    - e.g. new sources
  - But others are (sometimes fundamental) structural modifications, e.g.:
    - Split-weight scheme
    - Stellarator
    - 6D
    - Multirate timestepping
    - Time telescoping
    - Implicit timestepping

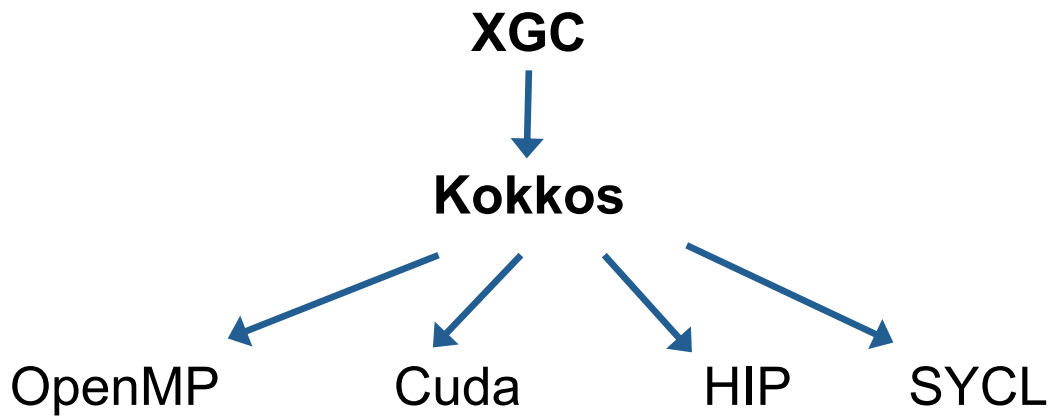
Target architectures

Machine	Cori KNL	Summit	Perlmutter	Frontier	Aurora
Testbed				Crusher	Florentia
Vendor	Intel	Nvidia	Nvidia	AMD	Intel
“Native” language		Cuda	Cuda	HIP	SYCL
GPU resources per rank		1 V100	1 A100	½ MI250X	[redacted]
Host memory per rank	96 GB	85.3 GB	64 GB	64 GB	[redacted]
Device memory per rank		16 GB	40 GB	64 GB	[redacted]

- Some calculations are better off distributed among compute nodes on Cori (less memory per rank, slower computation), but shouldn’t be distributed on Perlmutter (negligible computation time, more memory available)
- Some data is better off stored on device memory if there is a lot of available device memory, but must be moved between host and device if there is not

Exascale Preparation: Kokkos and C++

Kokkos: a portability abstraction layer that maps to OpenMP, Cuda, HIP, and SYCL



XGC Timeline

Pre 2019
Fortran code with 3 versions of dominant kernels:
- OpenACC collisions and

Cuda Fortran electron push for GPUs
- Vectorized CPU version,
- Simple reference CPU version

2019
Fortran code using wrappers and macros to offload with Kokkos
- Tedious and inflexible
- Unclear for AMD/Intel GPUs

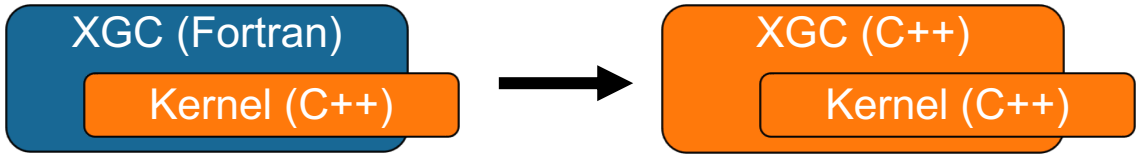
Present day
C++ code with non-critical components left in Fortran

XGC engineering approaches

- Portability with Kokkos
- Major focus on encapsulation/modularity
- Templating
  - e.g., electron push and ion push are quite different (electrons subcycle and are drift kinetic, ions are gyrokinetic) but use the same code
  - Makes it much easier than before to experiment/swap out options
- Kernels
  - Most major code components can be run independently
  - They use the same code base - not copies. This means:
    - They are never outdated
    - They don’t require extra maintenance
    - Work on the kernels can immediately benefit the code
- Testing/CI
  - Unit tests, kernel regression tests, and run test on every pull request
  - Automated physics testing still in progress – planned for bimonthly

Transition to C++

- Original attempt was to keep computation kernels in Fortran
  - Wrappers for Kokkos/Cabana functionality
  - Kernels launched with Kokkos::parallel_for() adapted to be compiled as both Fortran and Cuda Fortran



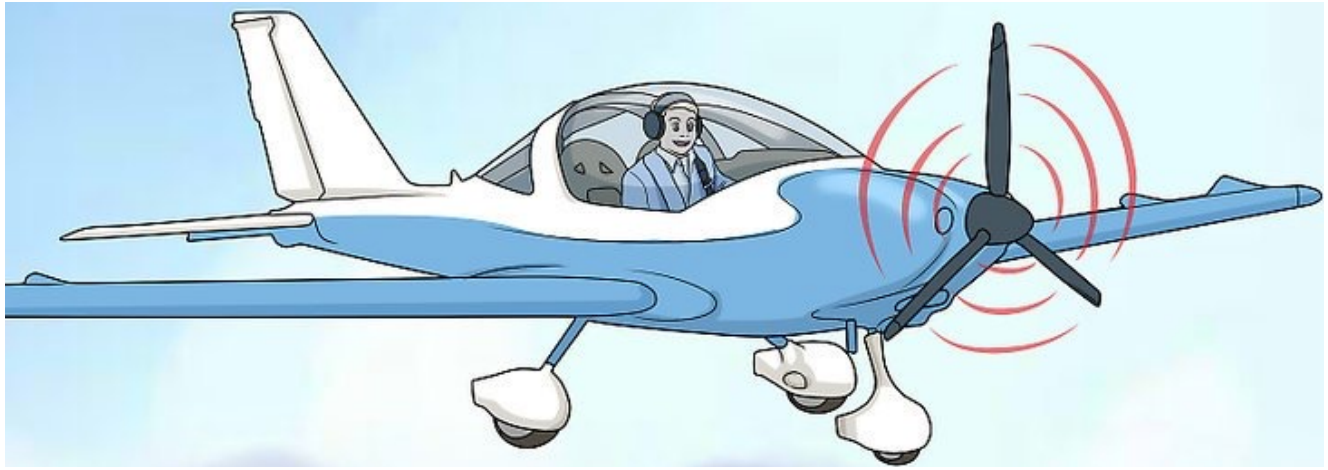
Continue with Fortran

- Feasible strategy for Cuda Fortran, but less clear for AMD/Intel GPUs
- Tedious interfaces, limited functionality
- OpenMP+ offloading instead of Kokkos?

Transition to C++

- Better usage of Kokkos/Cabana
- Earlier compiler support on ECP target architectures
- Lots of additional work to convert code
- Refactoring needed for offloading, might as well convert at the same time

“Like replacing every part of an airplane while in flight”



- Alternative in retrospect:
  - Write C++ version separately, from scratch?

New plane (C++ code from scratch)

- No time spent on Fortran interfacing
- Faster development since correctness not critical to current production

Mid-air replacement

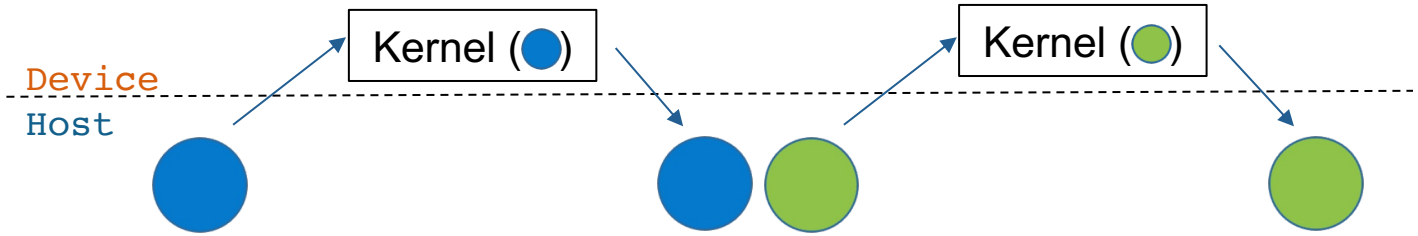
- Single code base
  - Maintenance and improvements benefit current production code
- Code kept up to date as new physics capabilities integrated
  - Already: EM physics, multispecies physics
- Code continually tested in production conditions

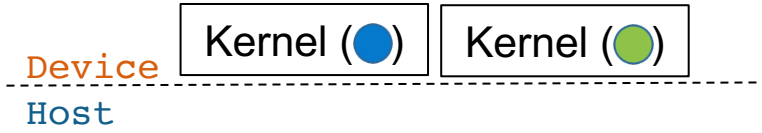
Particle memory management: Reside on CPU or GPU?

- Summit and Perlmutter have different optimal memory management for particles
  - Depends on available memory per GPU and per MPI rank, and communication rate

Particles sent to device for each kernel

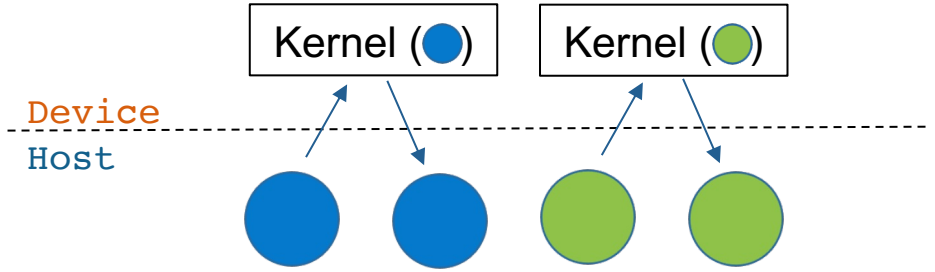
- More particles possible – only one species needs to fit on the GPU at a time
- Extra communication time





All particles reside permanently on GPU

- No time spent on communication
- Number of particles per species limited by GPU memory

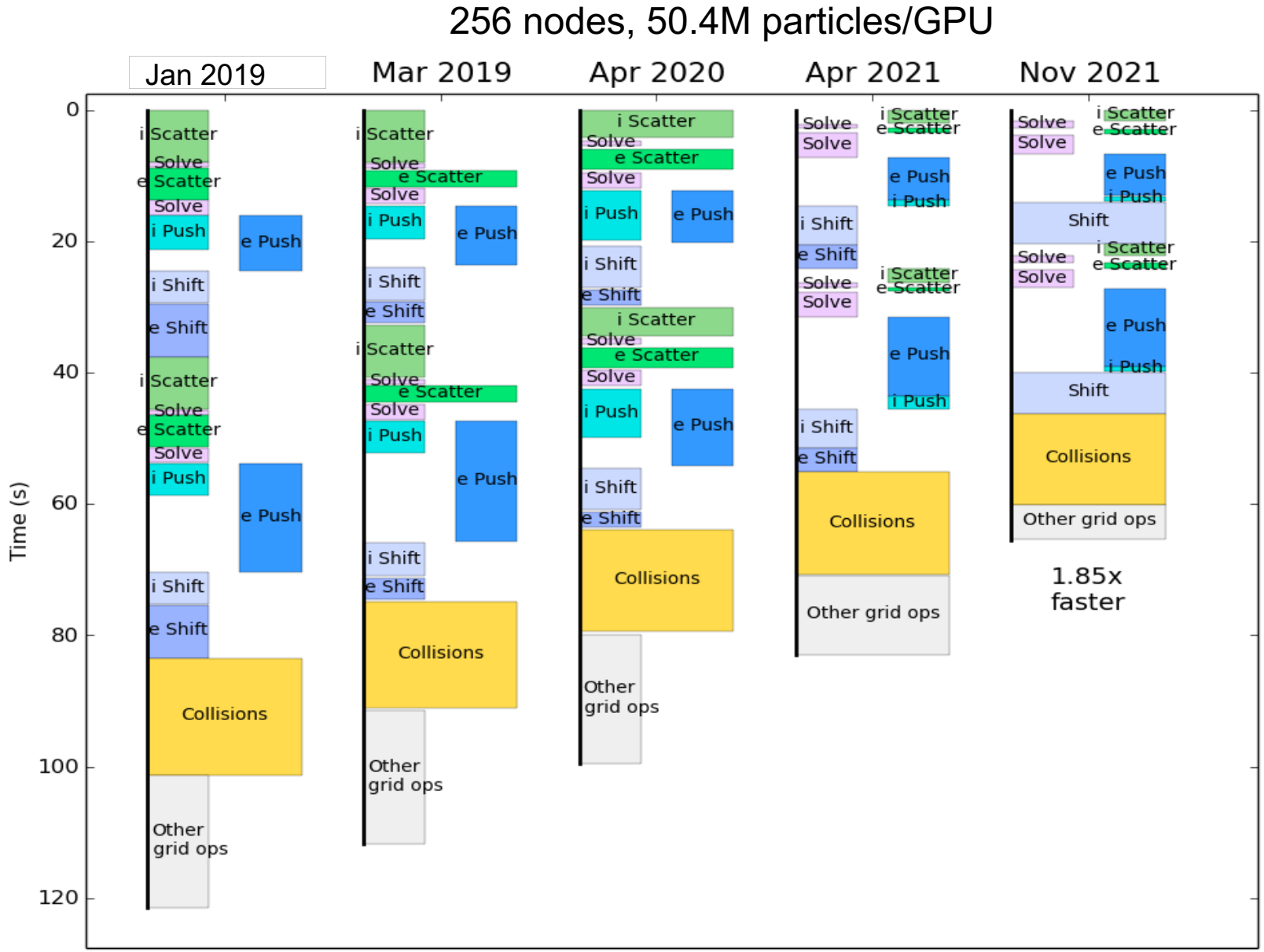


Asynchronous streaming to device and back

- Communication costs hidden to some extent
- Particles not limited by GPU memory
- Multiple streams may complicate portability

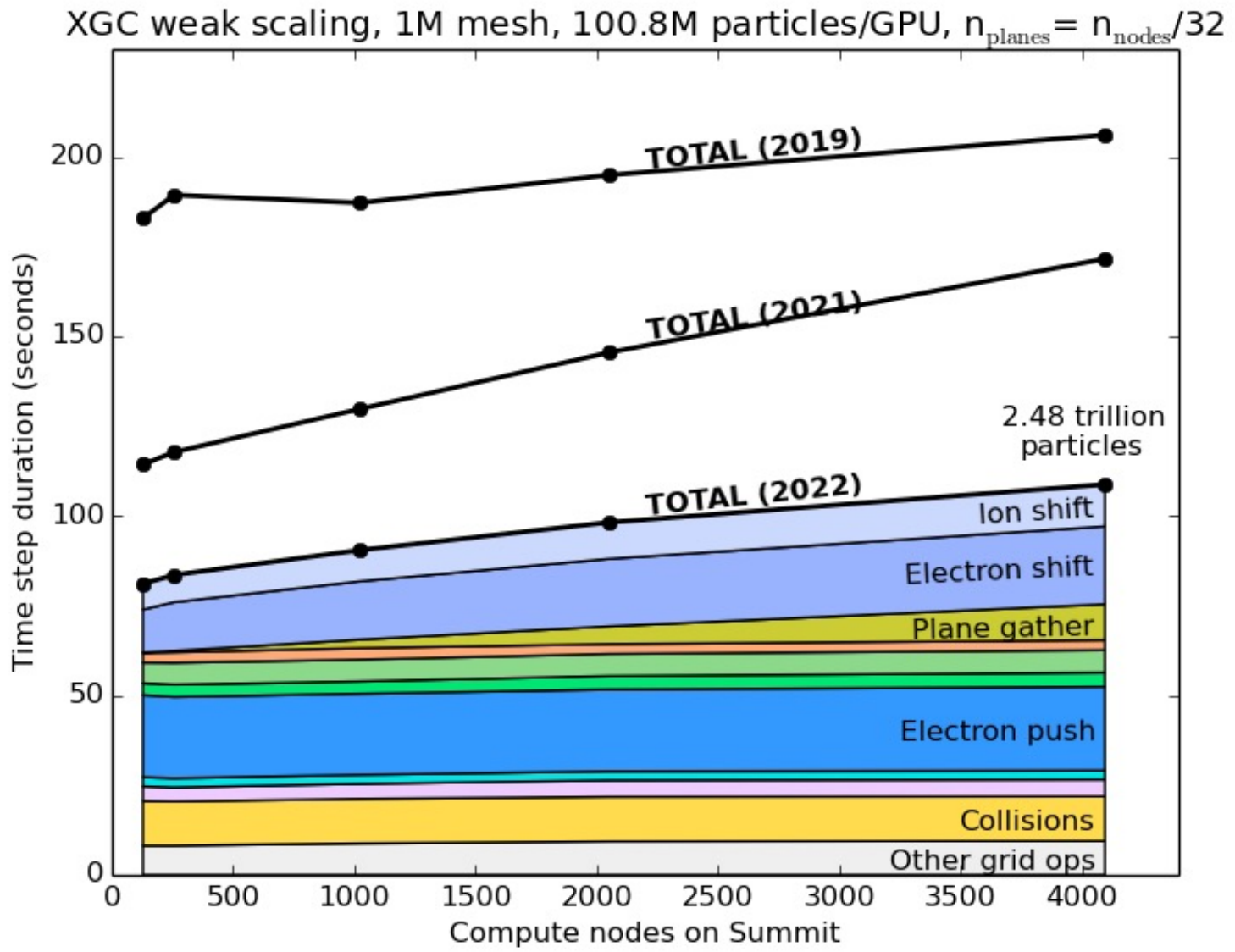
Summit performance

- Electrostatic simulation
- Steady improvement over time as more kernels moved from CPU (left) to GPU (right)



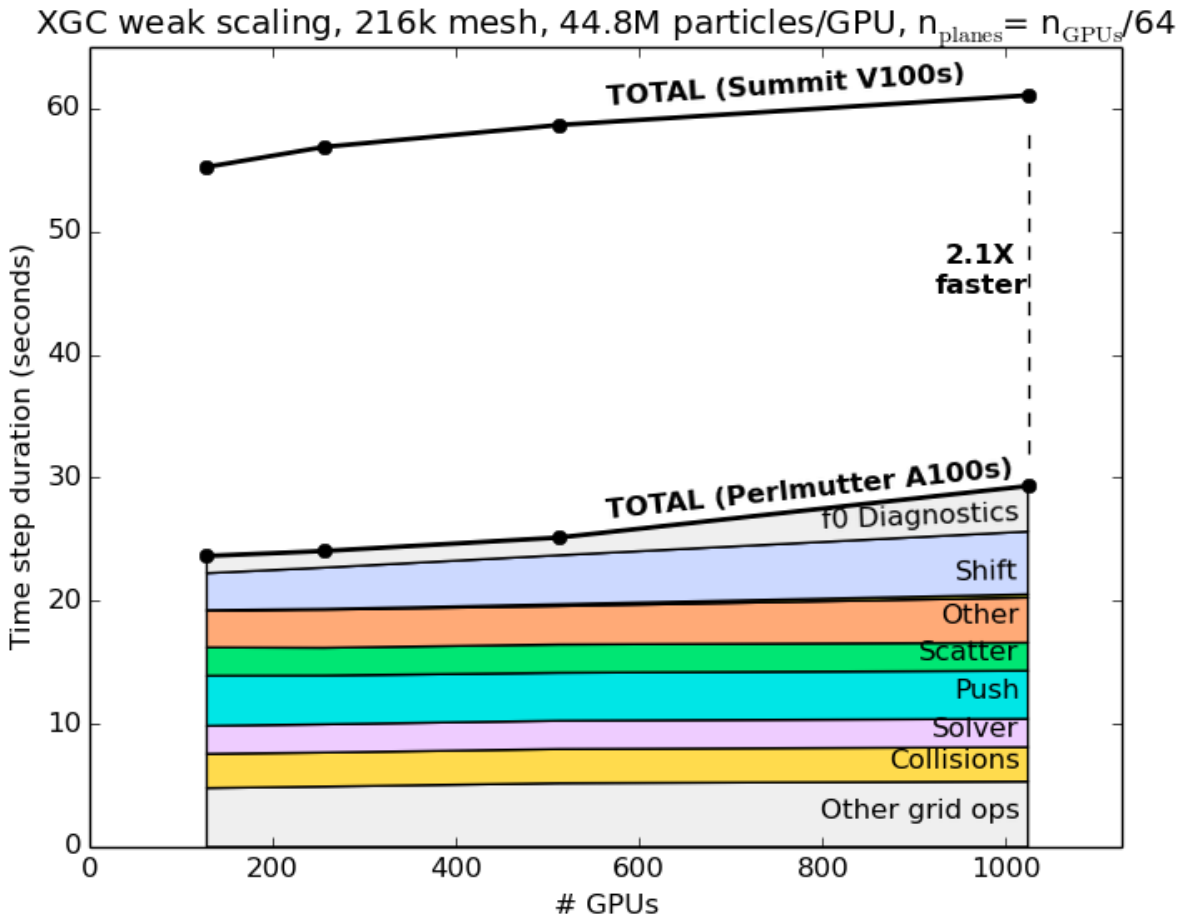
Summit performance

- Faster computation -> worse relative weak scaling
- Deviation from weak scaling due to:
  - Particle shift
  - Electric field interplanar gather (MPI_Allgather)



Perlmutter vs Summit

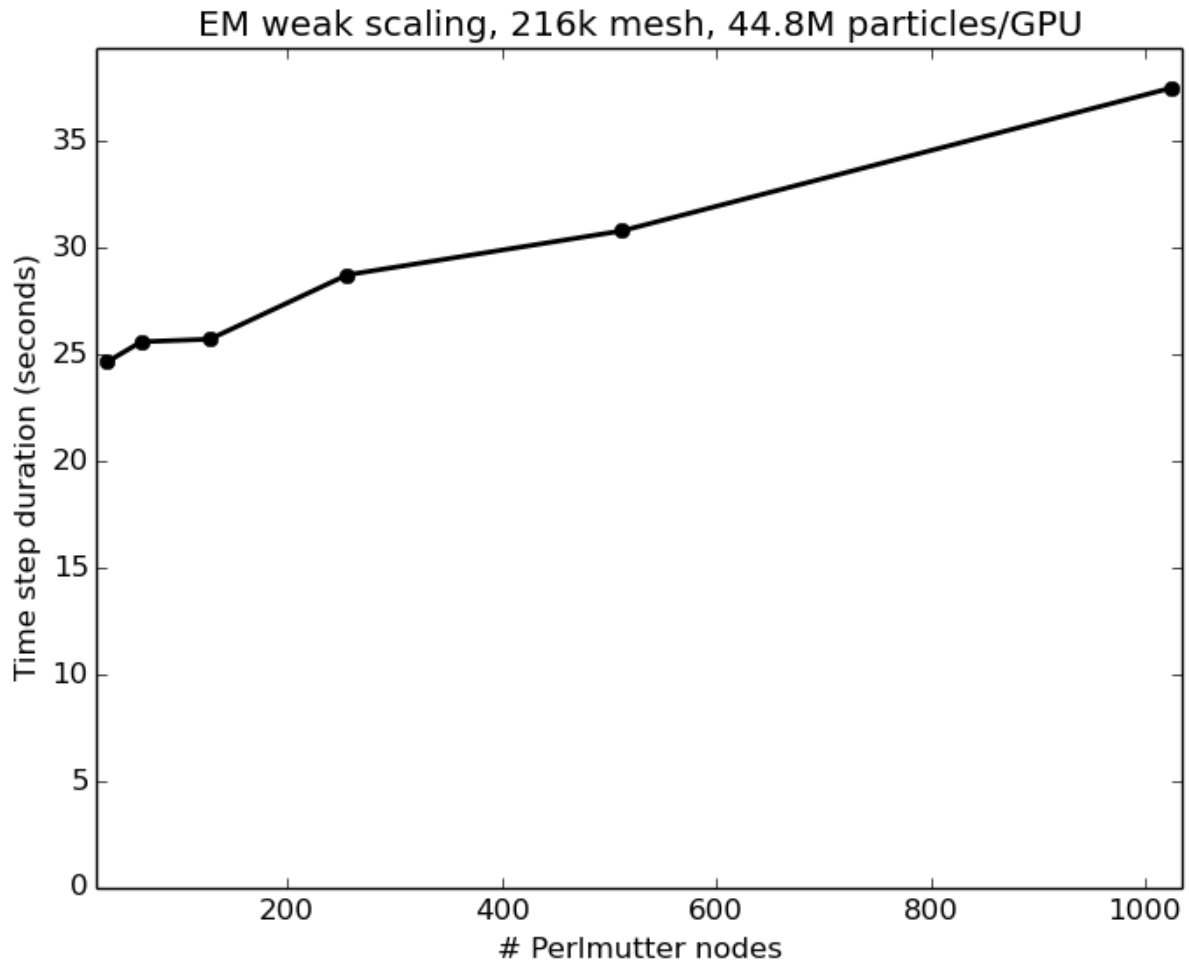
- Note: *Electromagnetic* simulation
  - Electron push kernel is less important since it is subcycled fewer times
  - Additional grid operations (due to extra fields)
- **~2.1X** faster than Summit (same # GPUs)



EXASCALE COMPUTING PROJECT

Perlmutter full-machine weak scaling

- GPU-aware MPI may improve this
- Simulation size too small – Perlmutter can fit more per node
  - Particle operations well-suited for GPU
  - Will improve weak scaling by increasing computation vs MPI communication



Perlmutter vs Cori KNL

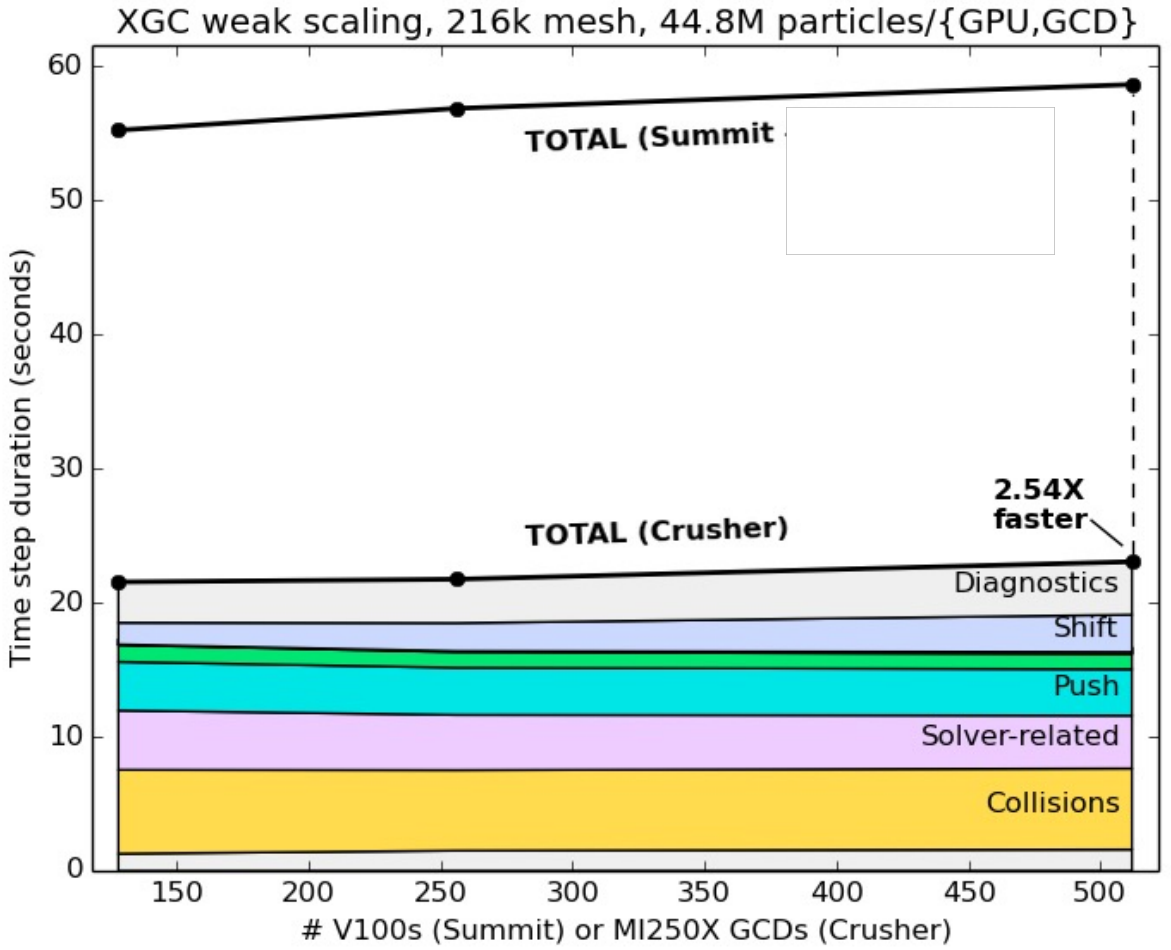
- How best to compare two very different machines? (Different memory layout, FLOPS, GPUs, etc.)
- One option: Run the same simulation ”packed” into as few nodes as possible

	64 Cori KNL nodes	**16 Perlmutter nodes**	**Difference**
Memory (host + device)	6.14 TB	6.66 TB	8%
Theoretical peak FLOPS (DP)	192 TFLOPS	660 TFLOPs	3.4X
Time per step	369.2 s	41.5 s	**8.9X faster**

- For this electrostatic simulation, we are utilizing Perlmutter resources are 2.6X better than on Cori KNL
- Varies depending on simulation type: electromagnetic simulations are not so GPU-optimized yet

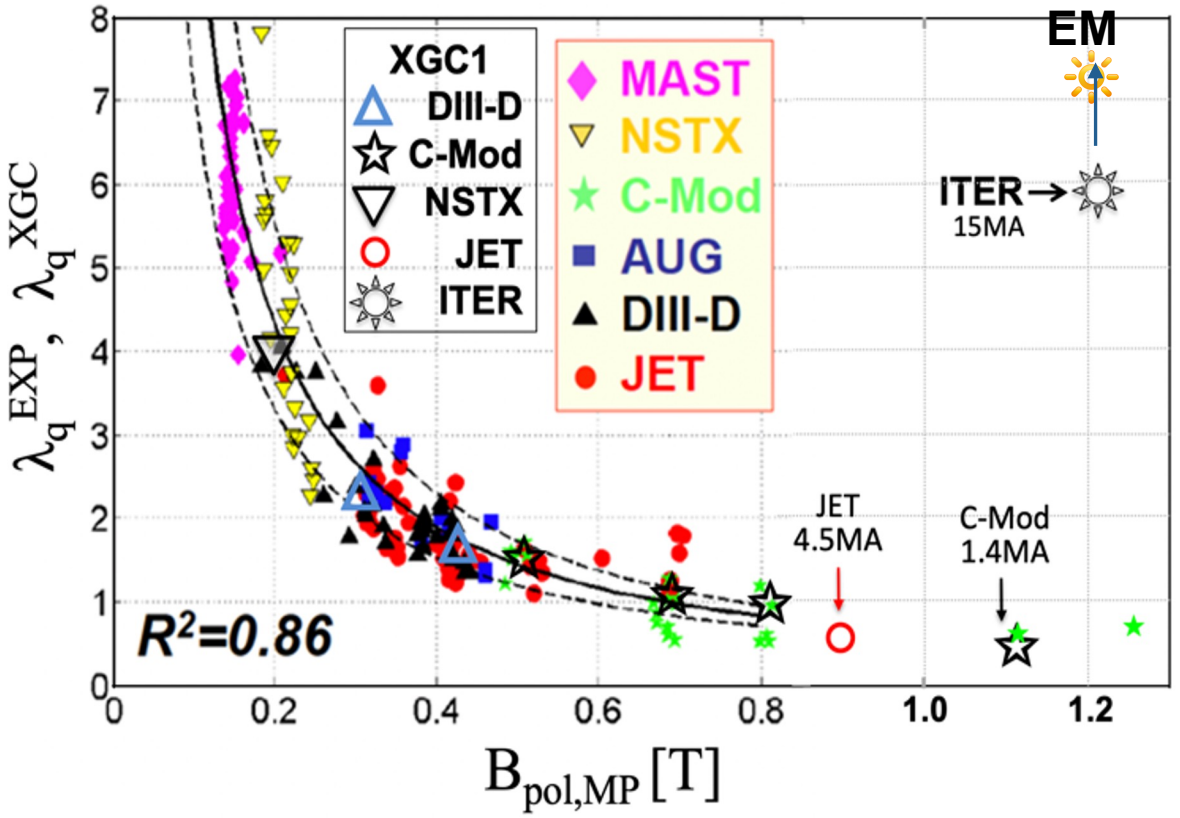
ECP testbed performance

- Comparing V100 vs GCD (i.e. ½ MI250X), EM simulation is **~2.54X** faster on Crusher (Frontier testbed) than Summit
- Performance on Aurora testbed machine also looking promising (not public yet)
- Our portability strategy seems to be working!

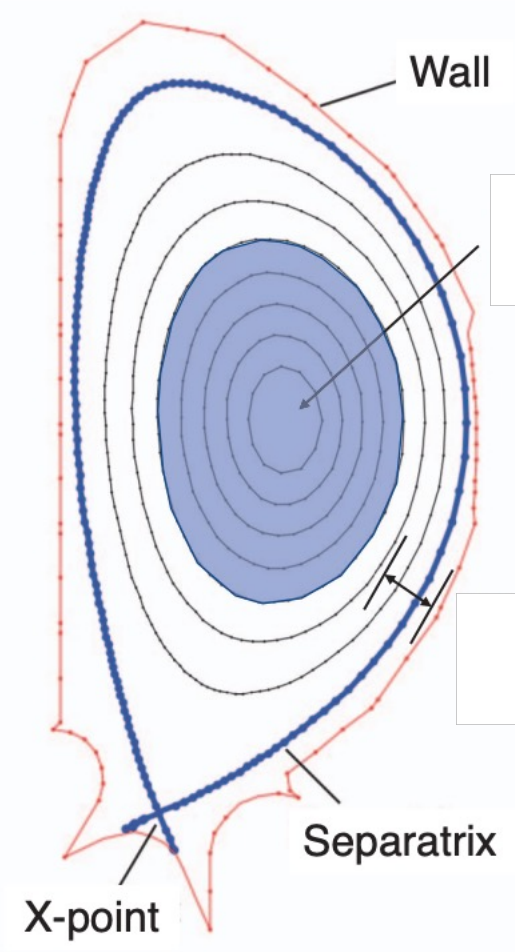


New Physics on Perlmutter

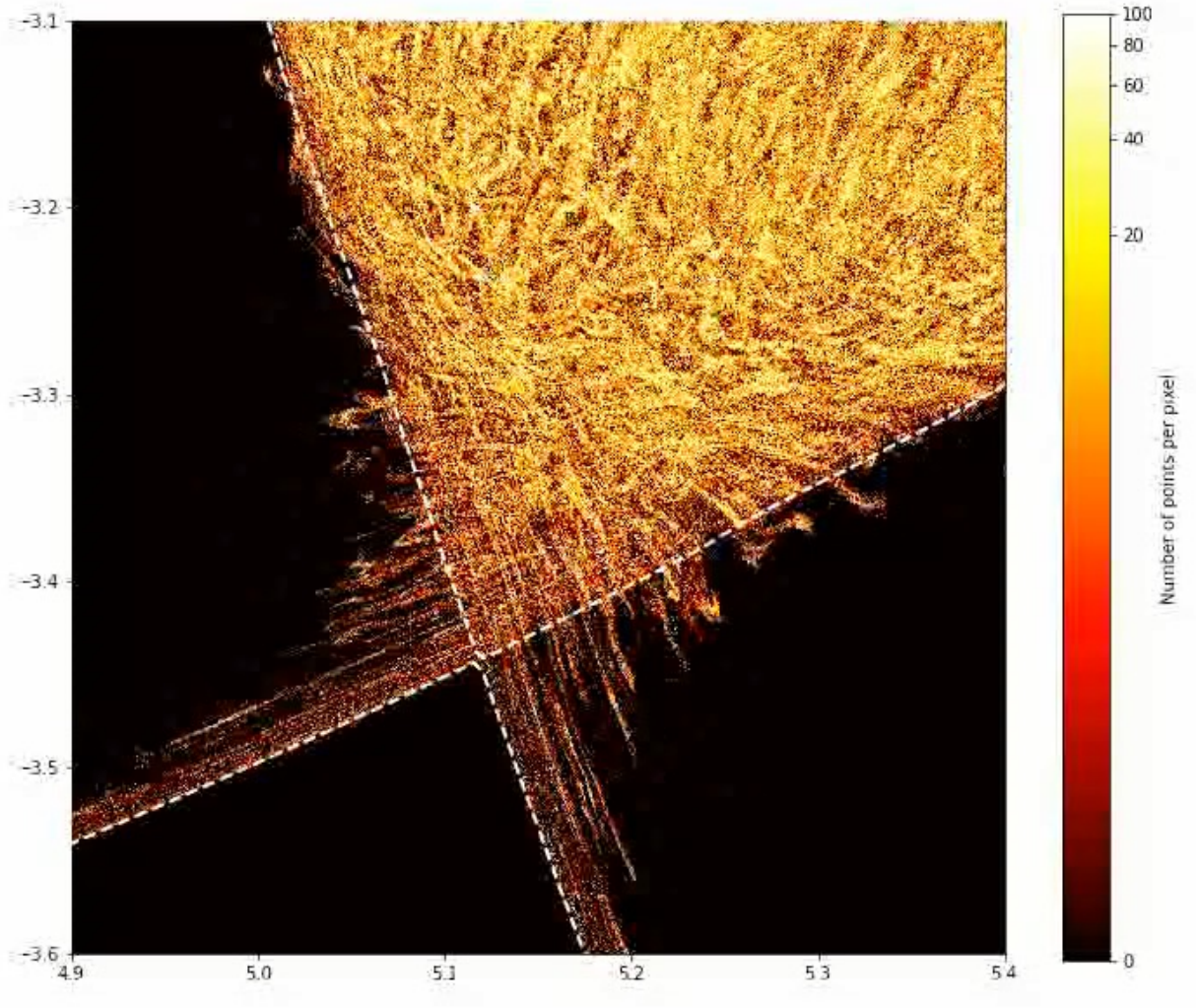
- In many tokamaks, exhaust from plasma is directed along the **separatrix** toward **divertor** plates
- The divertor must be prepared to handle the heat of this exhaust – a wider impact area is better, so **divertor heat load width** λ_q is an important parameter to study



- Using electrostatic simulations, XGC has matched observed λ_q for other tokamaks, but its predictions for ITER suggest it will be in a new regime with a higher-than-expected λ_q (good news)
- Our fully electromagnetic simulations on **Perlmutter** suggest that EM effects will increase λ_q even further



New Physics on Perlmutter



Poincare puncture-density plot of magnetic field in electromagnetic turbulence

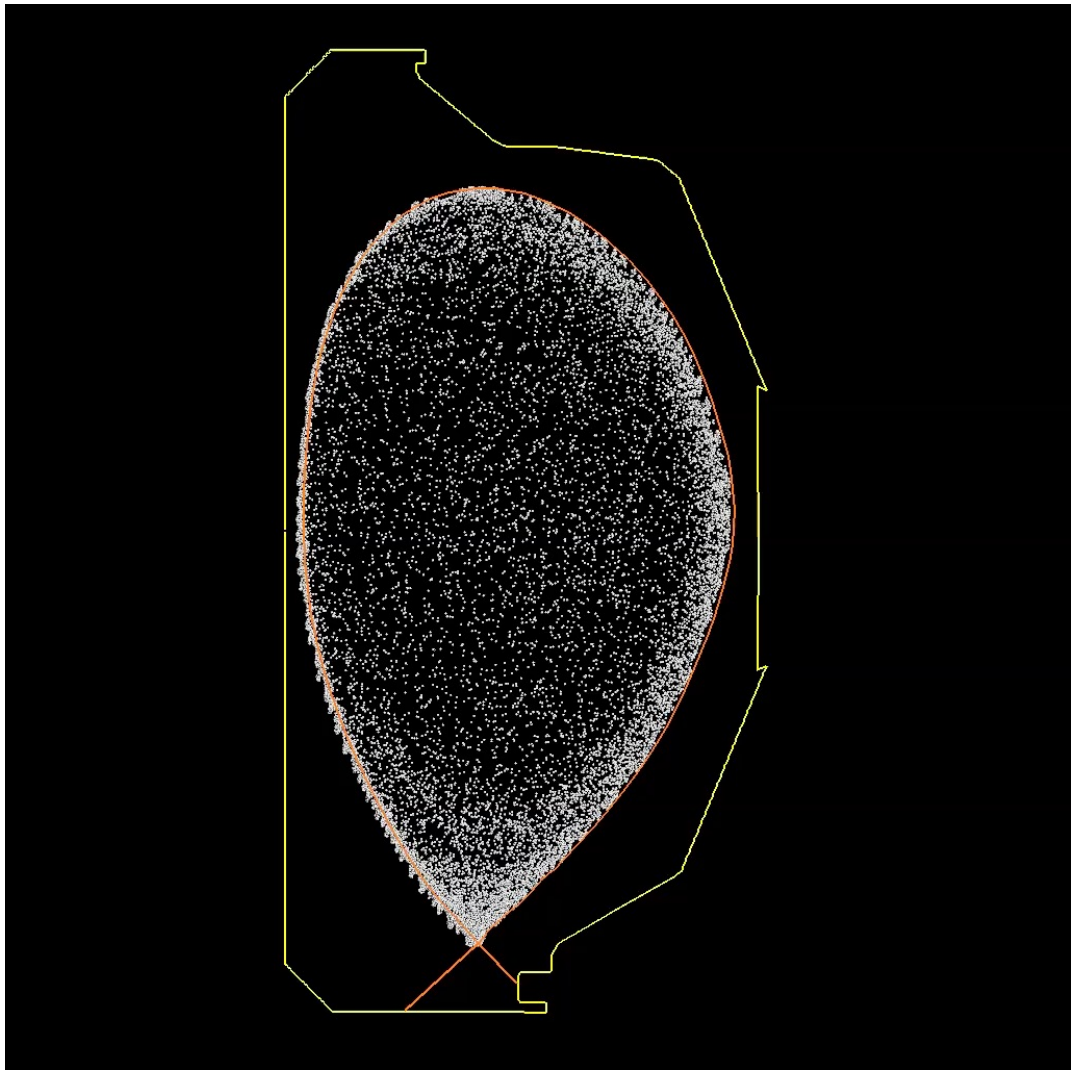
- **Homoclinic tangles** have been observed in the simulation, resulting in a more diffuse stream of plasma from the X-point to the divertor than previously thought

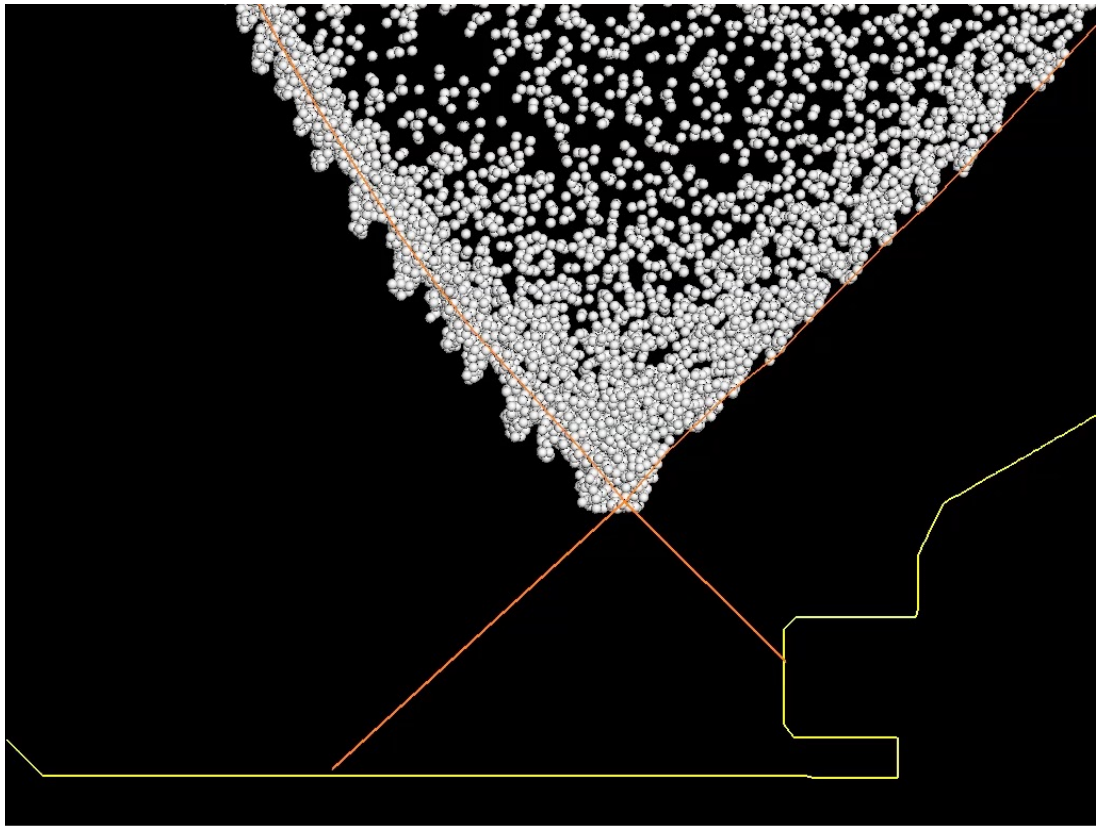
  (These have been observed before but only under special circumstances like ELMS)

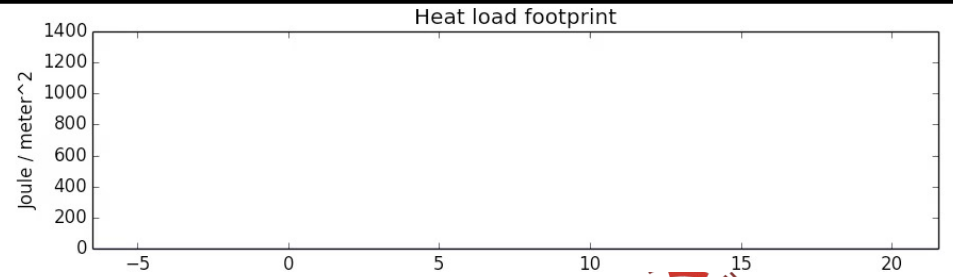
- Interventions may be possible to strengthen these tangles and further increase divertor heat load width

XGC simulation by S. Ku (PPPL) on Summit;
Visualization by D. Pugmire (ORNL)

New Physics on Perlmutter







[Particles starting from SOL are not visualized for clarity of physics.
DIII-D simulation by S. Ku, PPPL; Visualization by D. Pugmire, ORNL]

Conclusion

- XGC is running on Perlmutter and generally performing well
  - Still plenty of work to be done to optimize XGC (especially EM mode): offloading kernels to GPU via Kokkos, improving MPI communication and load balancing, and keeping up with new physics additions
- Perlmutter is already enabling XGC simulations that are providing insight on electromagnetic fusion plasma behavior and making predictions for ITER

Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.