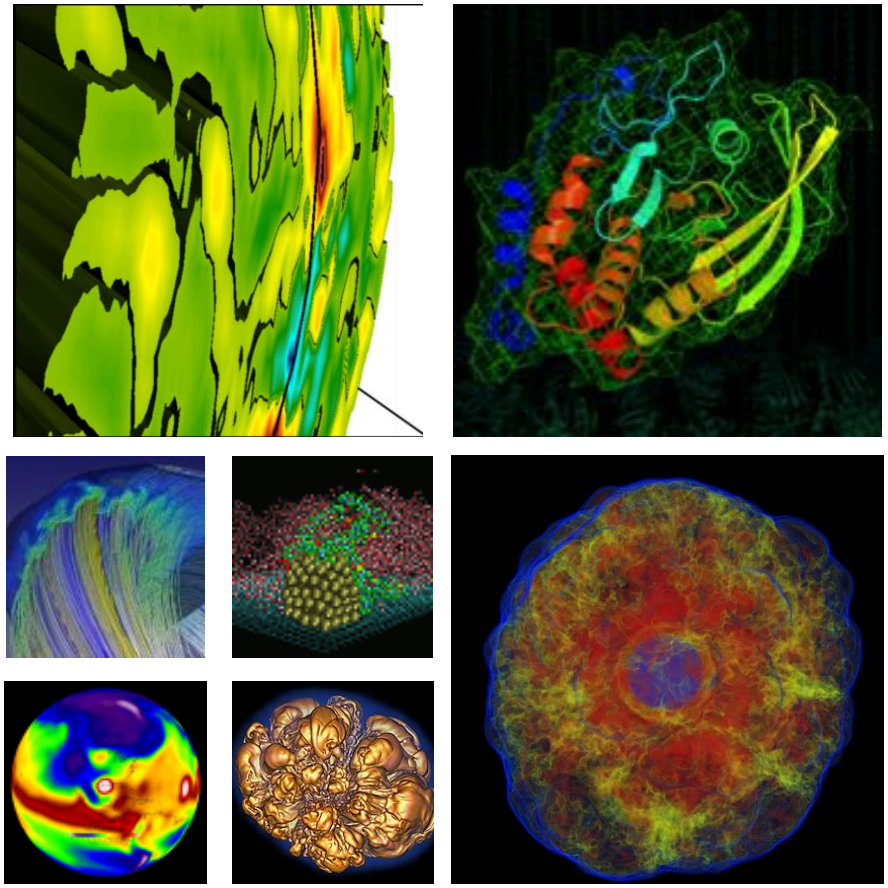
Optimizing Fusion PIC Code XGC1 Performance on Cori Phase 2



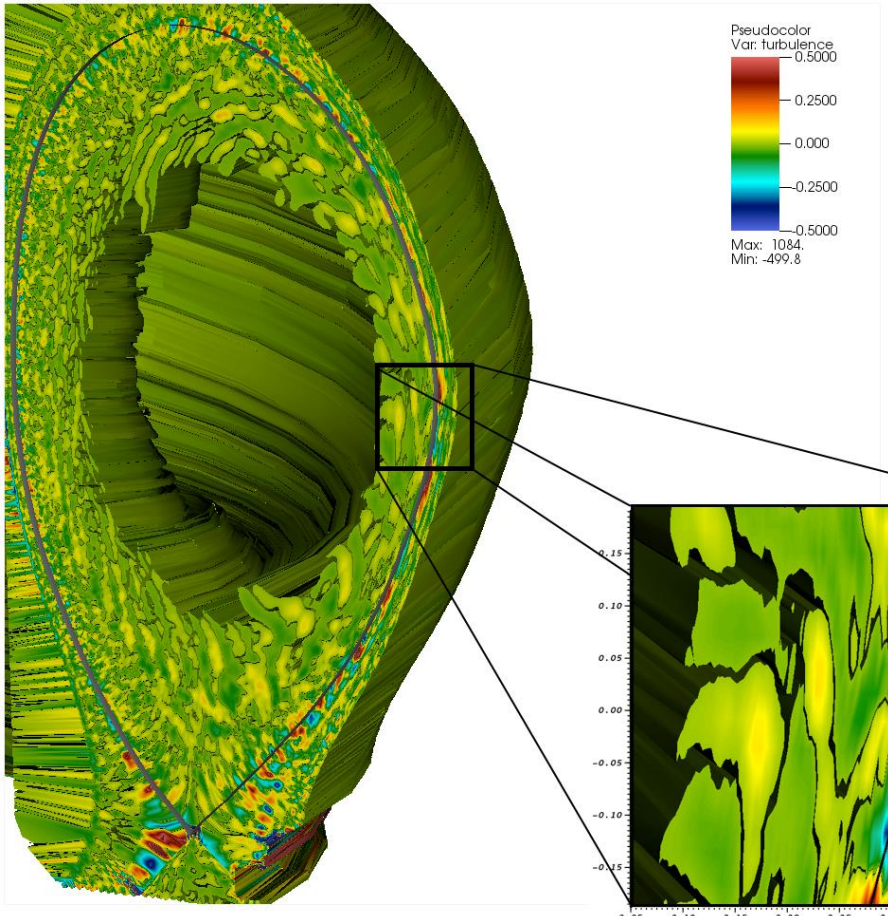
T. Koskela, J. Deslippe
NERSC / LBNL
tkoskela@lbl.gov

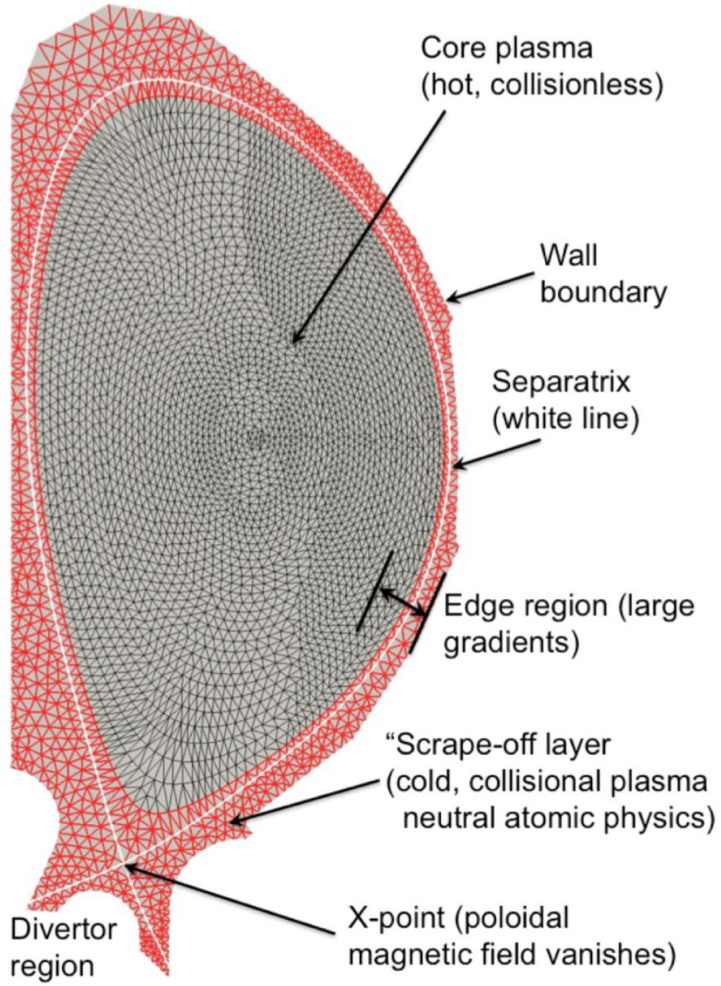
June 23, 2017

Thank you to all collaborators!

- **LBNL**
  - Brian Friesen, Ankit Bhagatwala, Mark Adams, Mathieu Lobet, Tareq Malas, Andrey Ovsyannikov, Kevin Gott
- **PPPL**
  - CS Chang, Robert Hager, Seung-Hoe Ku, Stephane Ethier
- **ORNL**
  - Ed D'Azevedo, Stephen Abbott
- **Intel**
  - Thanh Phung, Zakhar Matveev, John Pennycook, Martyn Corden, Karthik Raman
- **RPI**
  - Eisung Yoon

XGC1 is a Particle-In-Cell Simulation Code for Tokamak (Edge) Plasmas



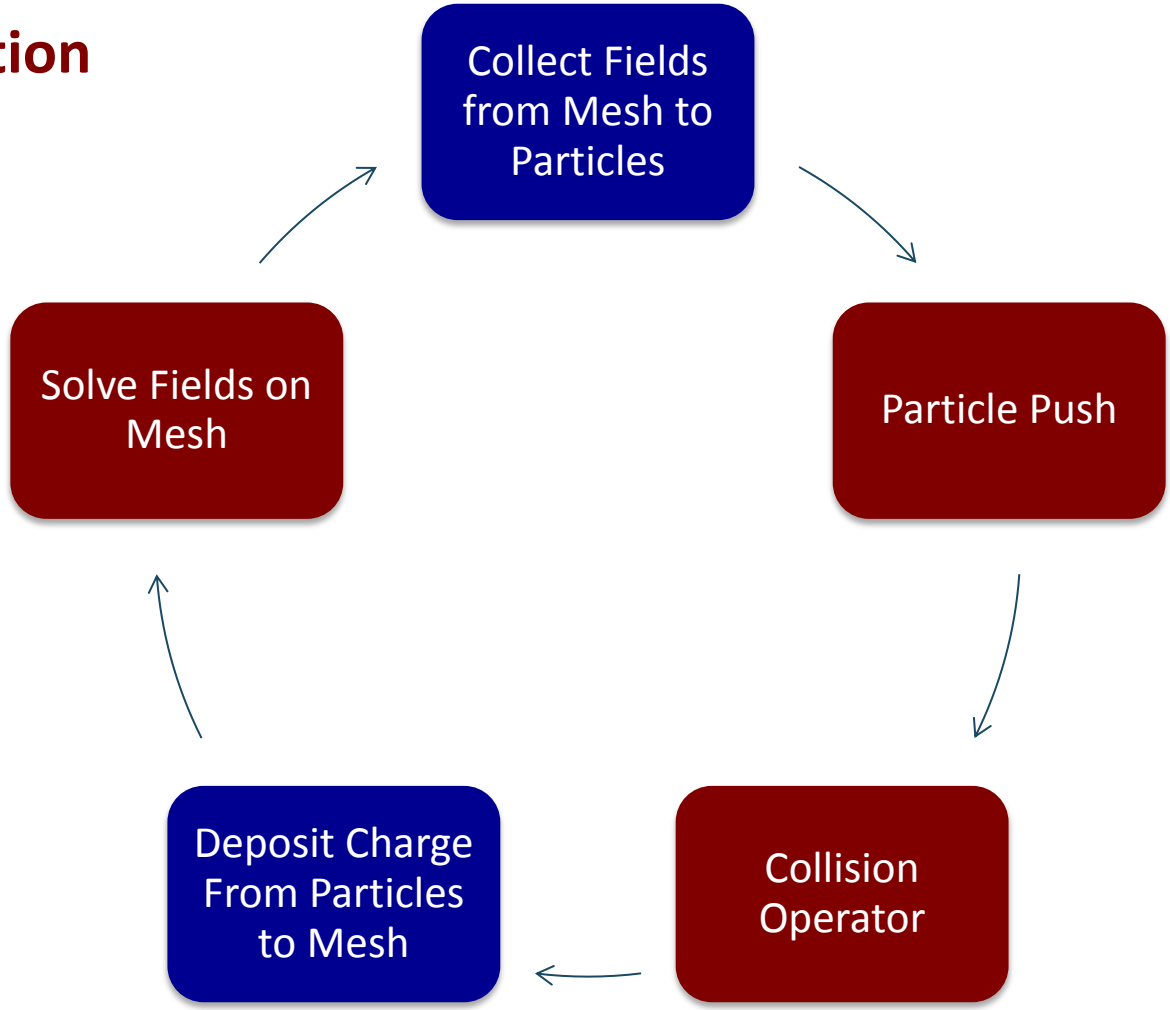


PI: CS Chang (PPPL) | ECP: High-Fidelity Whole Device Modeling of Magnetically Confined Fusion Plasma

Collisional Plasma PIC Code Flowchart

Computation

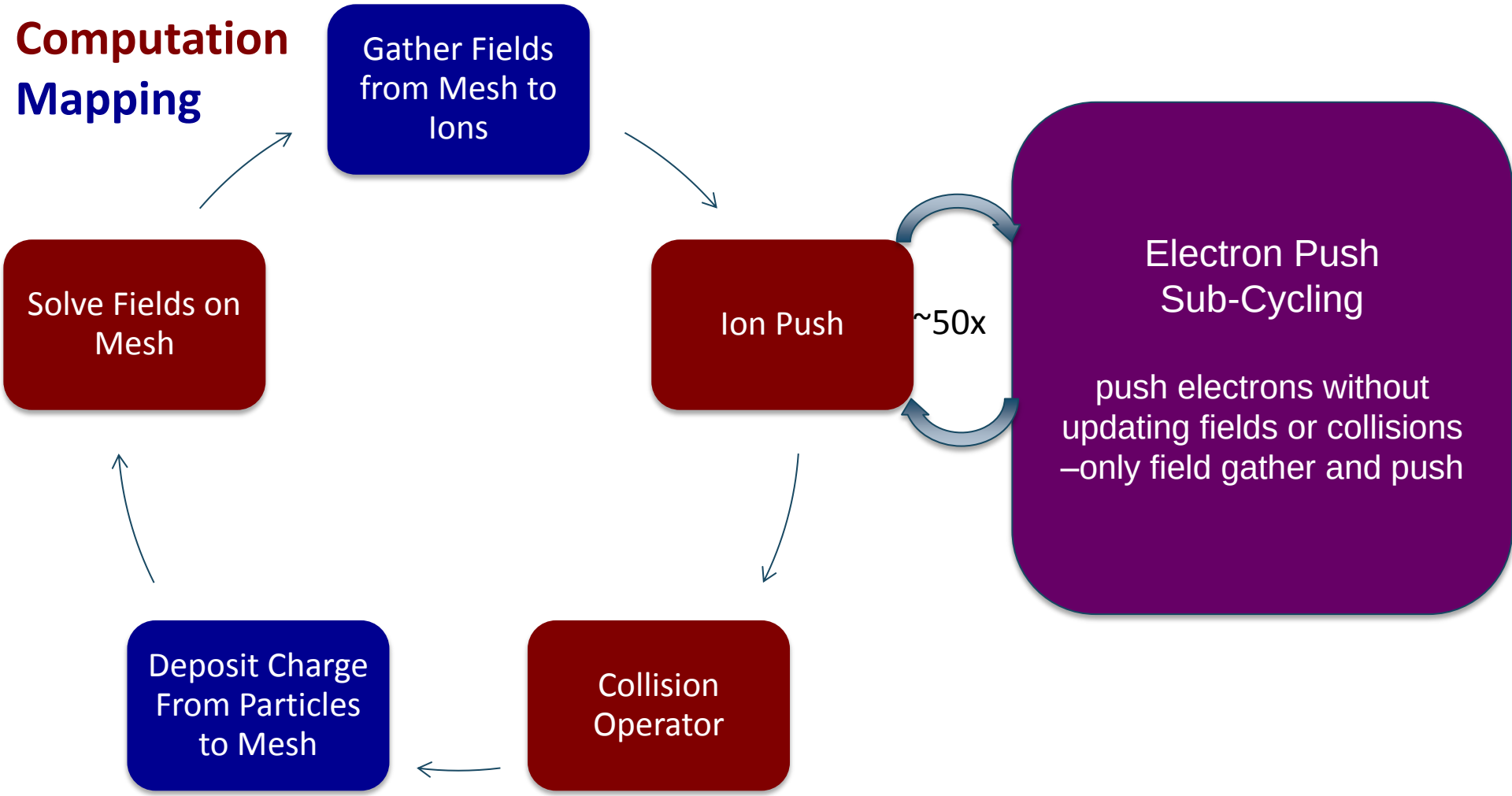
Mapping



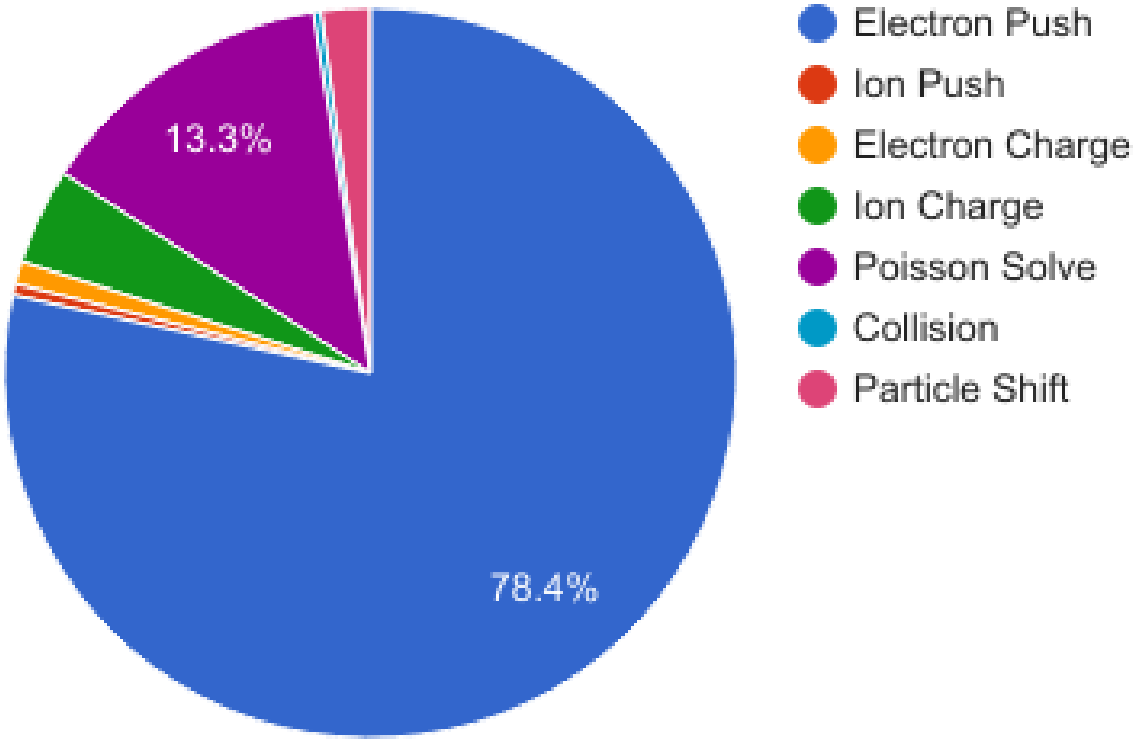
XGC1 Unique Optimization Challenges

- **Complicated Toroidal Geometry**
  - Unstructured mesh in 2D (poloidal) plane(s)
  - Nontrivial field-following (toroidal) mapping between meshes
  - Typical exascale simulation has 10 000 particles per cell, 1 000 000 cells per domain, 64 toroidal domains.
- **Gyrokinetic Equation of Motion in Cylindrical Coordinates**
  - + 6D to 5D problem
  - + O(100) longer time steps
  - -- Higher (2nd) order derivative terms in EoM
  - -- Averaging scheme in field gather
- **Electron Sub-Cycling**

Separated From the Ion Push in a Sub-Cycling Loop

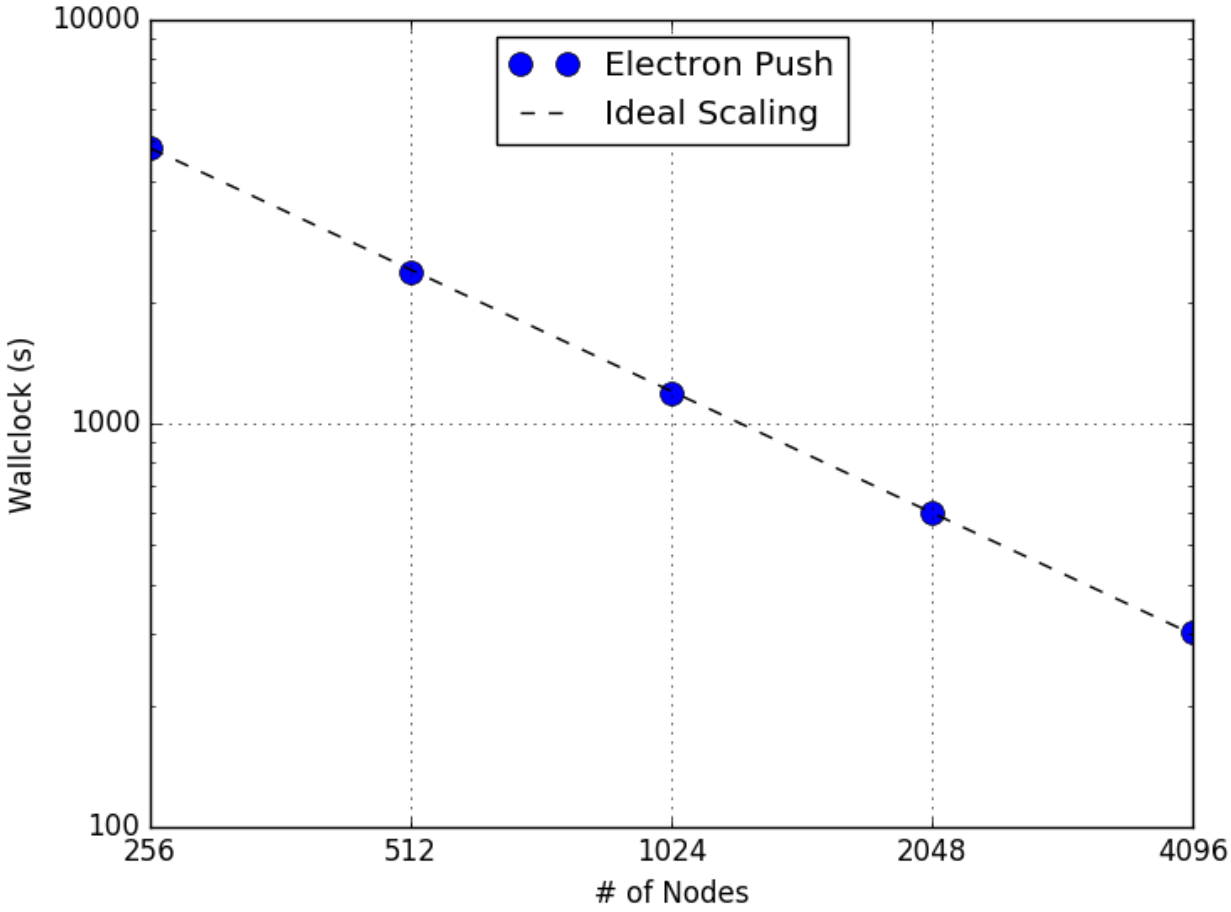


Motivation: XGC1 CPU time is dominated by electron push sub-cycle



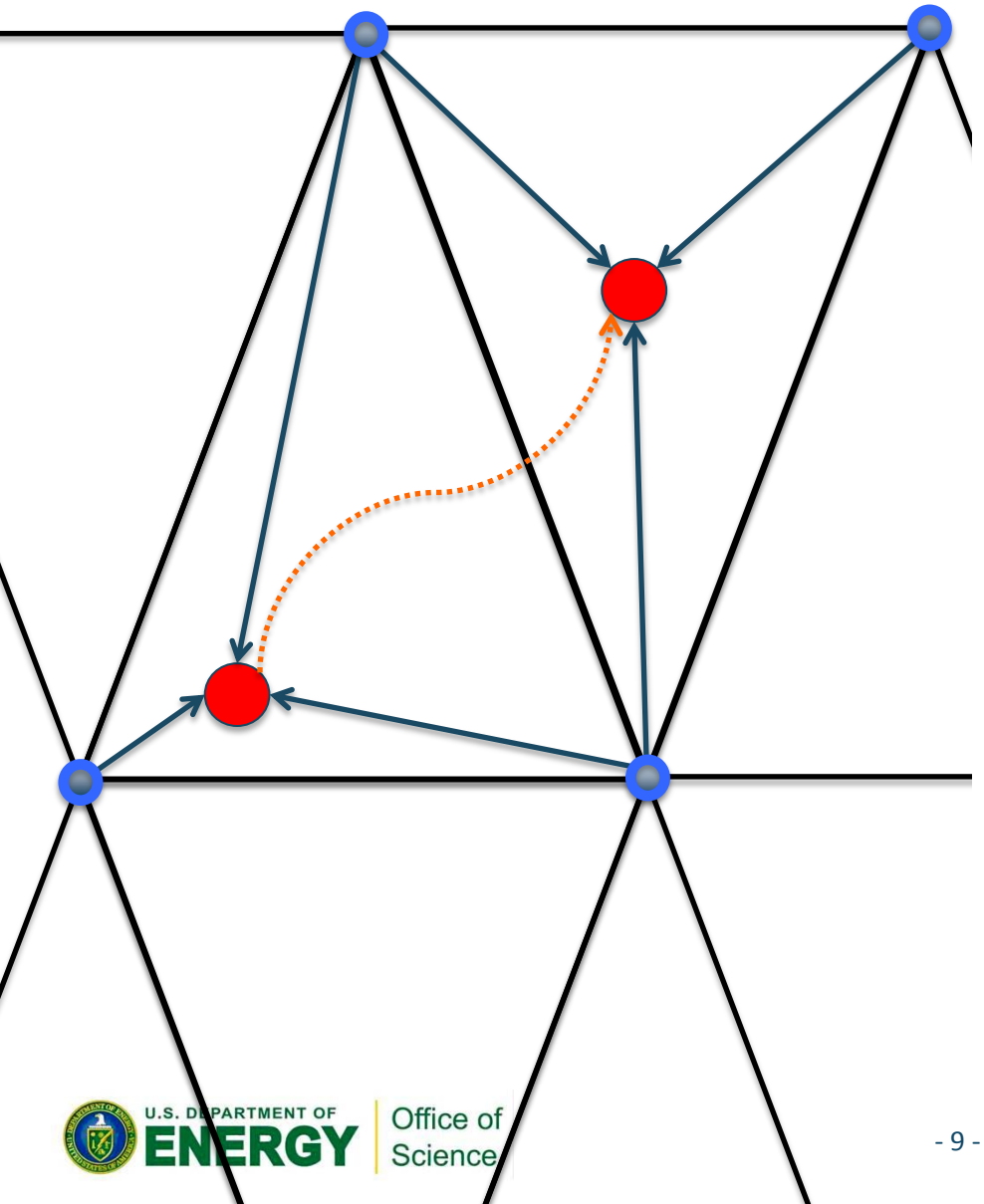
Baseline XGC1 Timing on 1024 Cori KNL nodes in quadrant flat mode.

Motivation: Ideal Strong Scaling of Electron Sub-Cycling On Cori



Cori KNL quadrant cache nodes, 16 MPI ranks per node/16 OpenMP threads per rank

(Simplified) Particle Push Algorithm



1. Search for nearest 3 mesh nodes to the particle position
2. Interpolate fields from 3 mesh points to particle position
3. Calculate force on particle from fields
4. Push particle for time step dt

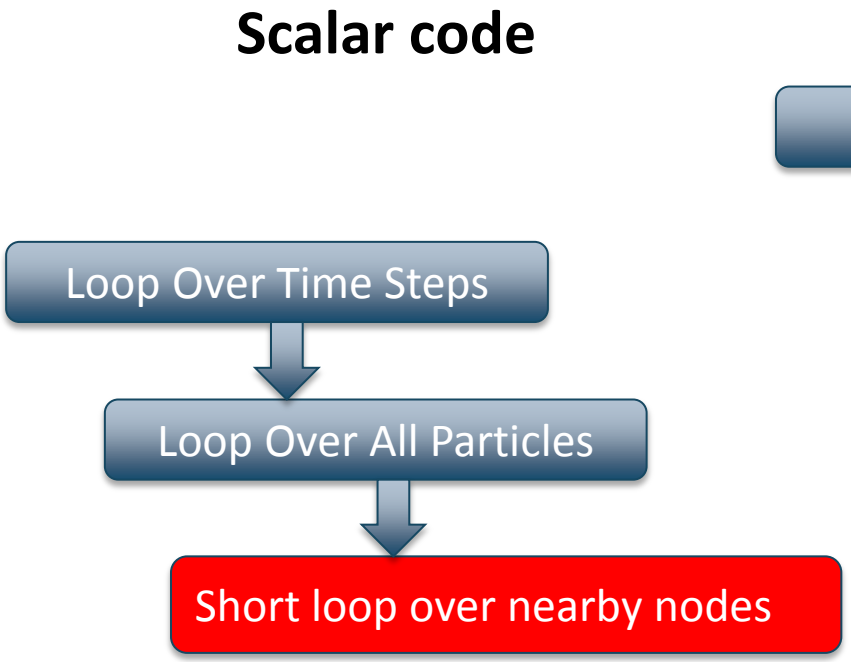
Main Bottlenecks in Electron Push

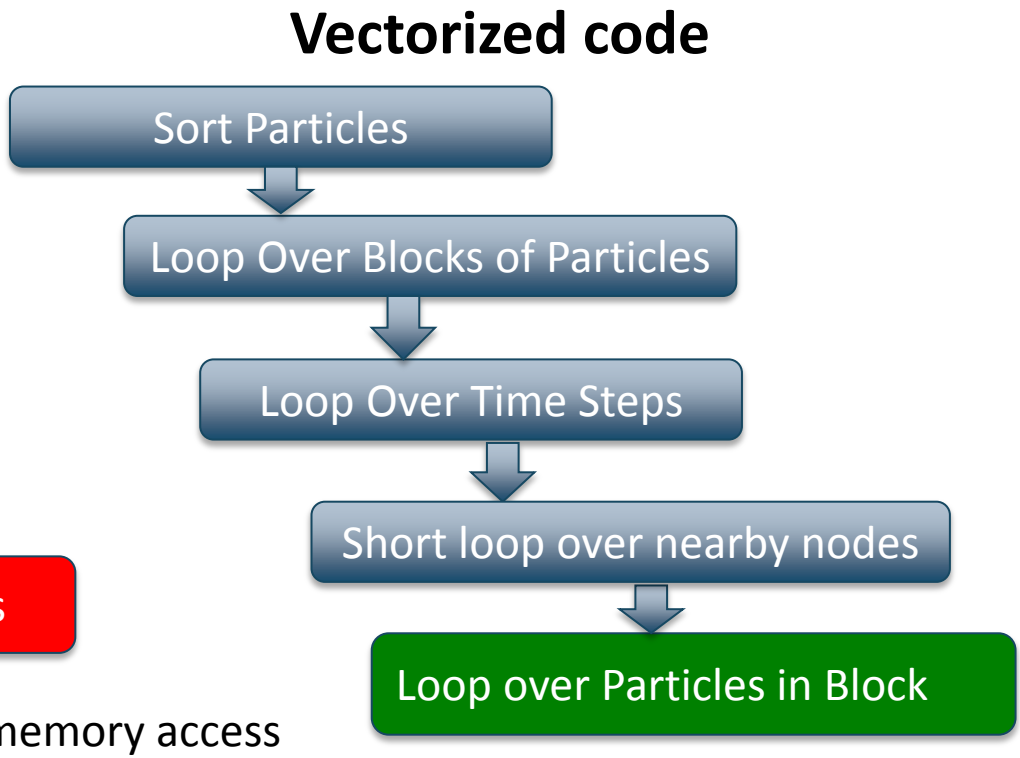
- **E and B Field Interpolation**
  - Inner loops over nearby grid nodes with short trip counts make auto-vectorization ineffective
  - Indirect grid access produces gather/scatter instructions
- **Search on Unstructured Mesh**
  - Multiple exit conditions
- **Force Calculation**
  - Strided memory access in complicated data types
  - Cache unfriendly

Main Optimizations in Electron Push

- **Enabling Vectorization**
  - Insert loops over blocks of particles inside short trip count loops
  - Sort particles to reduce random memory accesses
- **Data Structure Reordering**
  - Store field and particle data in SoA format.
  - SoA best when accessing multiple components with a gather instruction
- **Algorithmic Improvements**
  - Sort particles by the mesh element index instead of local coordinates
  - Reduce number of unnecessary calls to the search routine

Re-Ordering Loops to Enable Vectorization





- Sort particles to reduce random memory access
- Swap the order of time step and particle loops to improve cache reuse
- Insert vectorizeable loop over blocks of particles inside short trip count loop
- Near-ideal vectorization in compute-heavy loops
  → Indirect memory access becomes the bottleneck

Reorder Particle Data Structures

AoS

```
type field
  real :: Bx,By,Bz
  real :: Ex,Ey,Ez

  ...
end type

type(field) :: fld(number_of_particles)
```

SoA

```
type field_vec
  real :: Bx(number_of_particles)
  real :: By(number_of_particles)
  real :: Bz(number_of_particles)
 ...
end type

type(field_vec) :: fld
```

- Stores field data at particle location between field gather and particle push
- AoS → Strided access when accessing one data type of multiple particles
- SoA → Unit Stride when accessing one data type of multiple particles

Reorder Particle Data Structures

AoS

```
type field
  real :: Bx,By,Bz
  real :: Ex,Ey,Ez

  ...
end type

type(field) :: fld(number_of_particles)
```

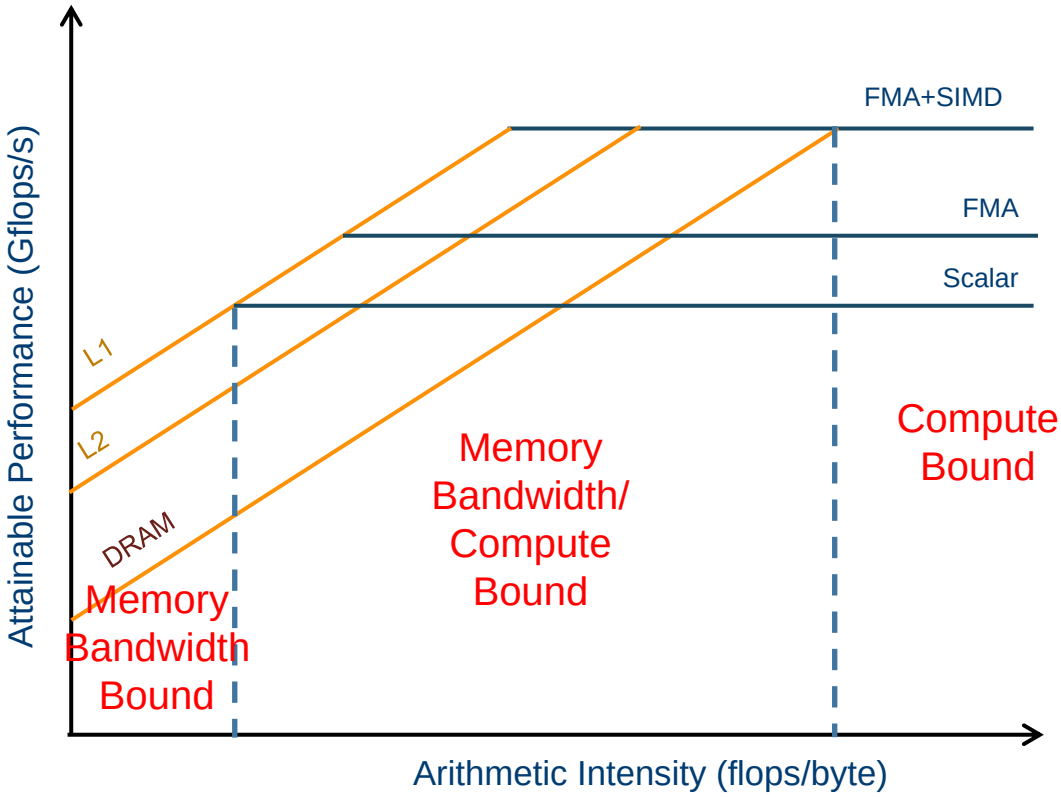
SoAoS

```
type field_vec
  real :: B(3,number_of_particles)
  real :: E(3,number_of_particles)

  ...
end type

type(field_vec) :: fld
```

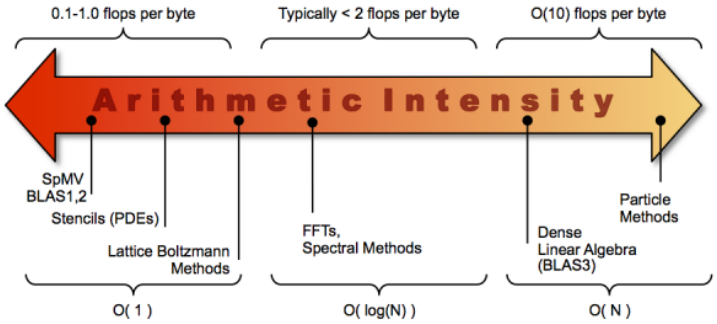
- Stores field data at particle location between field gather and particle push
- AoS → Strided when accessing one data type of multiple particles
- SoAoS → Retrieve all components of a vector field on the same cache line

Roofline Performance Model

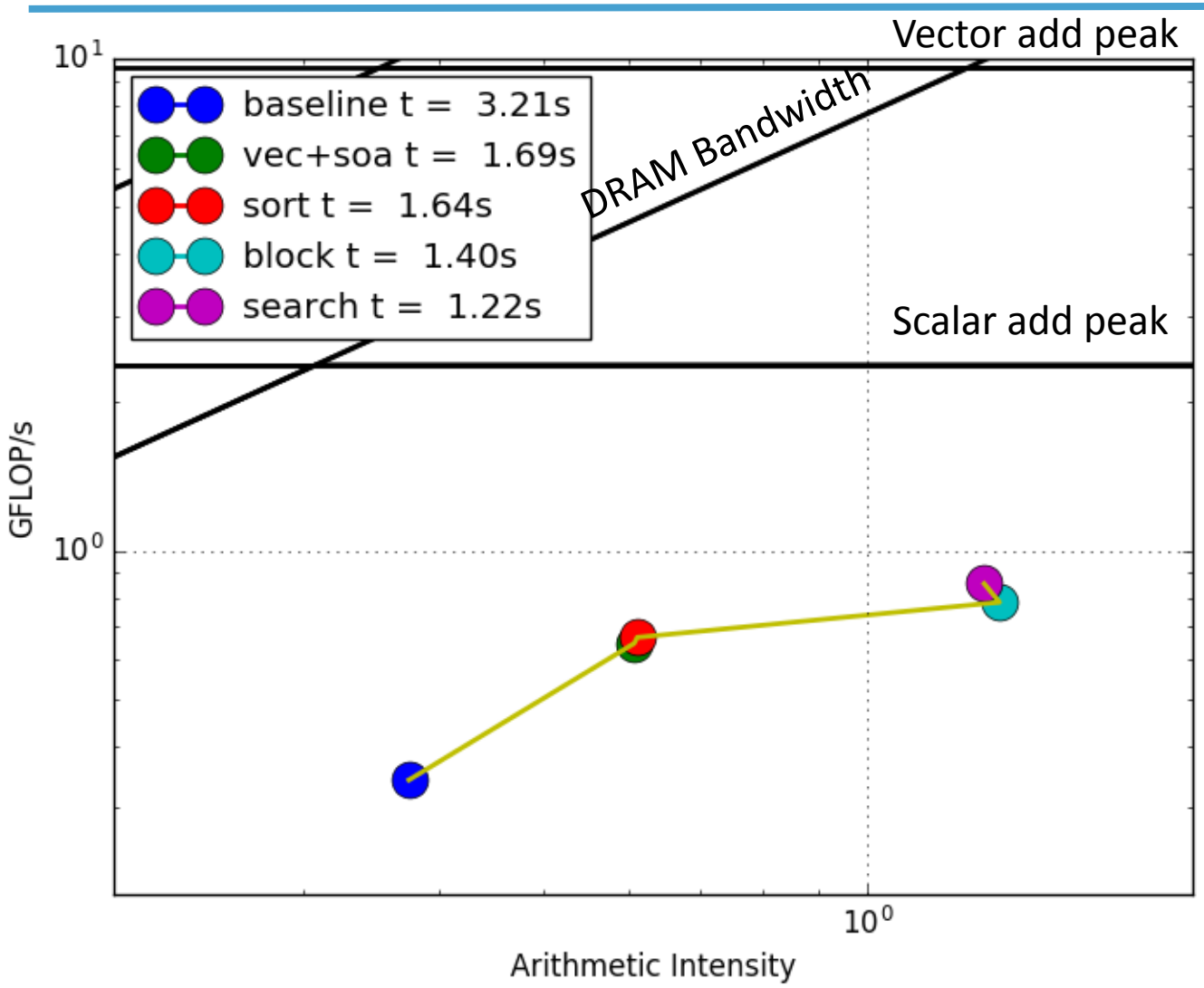
Roofline reflects an absolute performance bound (Gflops/s) of the system as a function of Arithmetic Intensity (flops/byte) of the application.



$$\text{Arithmetic Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred from DRAM}}$$



Roofline Analysis for Electron Push Kernel, KNL quadrant cache node



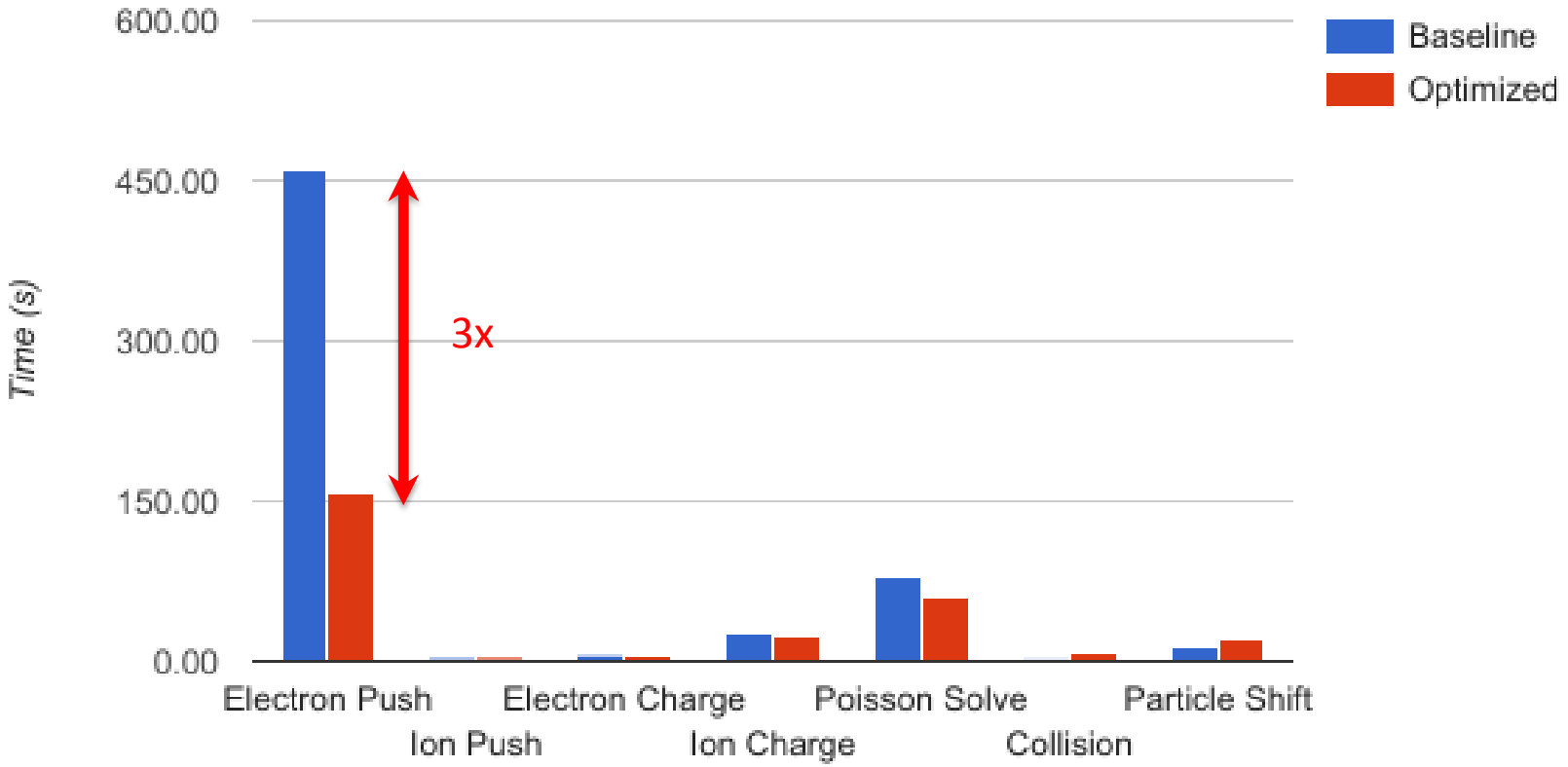
Single thread performance on KNL

3x Speedup achieved

Large increase in AI from blocking/sorting

Optimized performance still 10x below vector peak, AI is high enough to reach it.

Electron Push Speedup



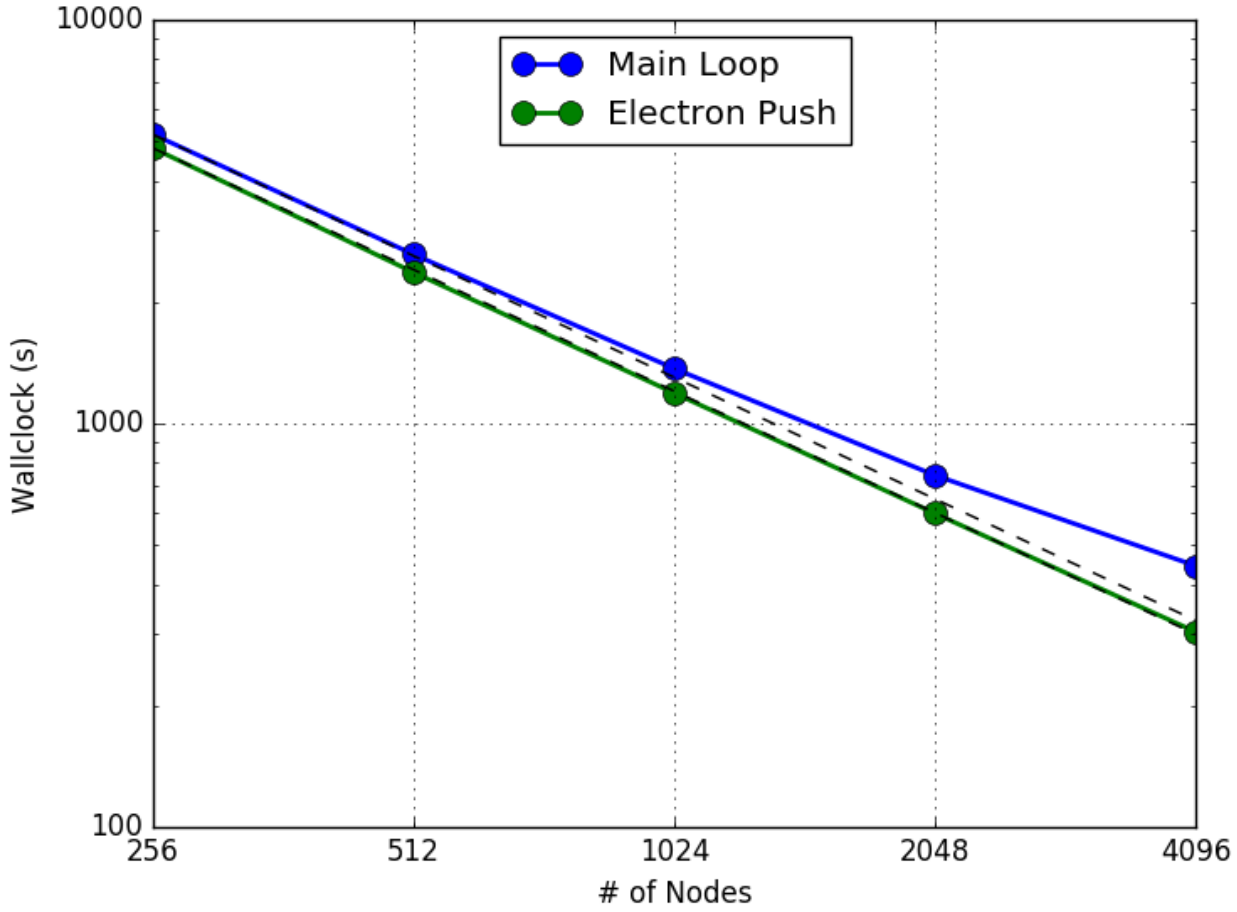
XGC1 Timing on 1024 Cori KNL nodes in quadrant flat mode.

Strong Scaling Parameters

Compute Nodes	Grid Nodes Per Rank	Particles Per Rank
256	448	12.2 M
512	224	6.1 M
1024	112	3.1 M
2048	56	1.5 M
4096	28	0.75 M

- 16 MPI ranks per Node, 16 OpenMP Threads per rank
- 5 Bn total particles
- 57 000 total grid nodes per plane, 32 planes
- Quadrant Cache mode

XGC1 Strong Scaling up to 4096 KNL Nodes



16 MPI ranks per node, 16 OpenMP threads per rank.

Strong scaling for problem size of 25 Bn particles, grid representative of present production runs (DIII-D tokamak)

Ideal Scaling in electron push

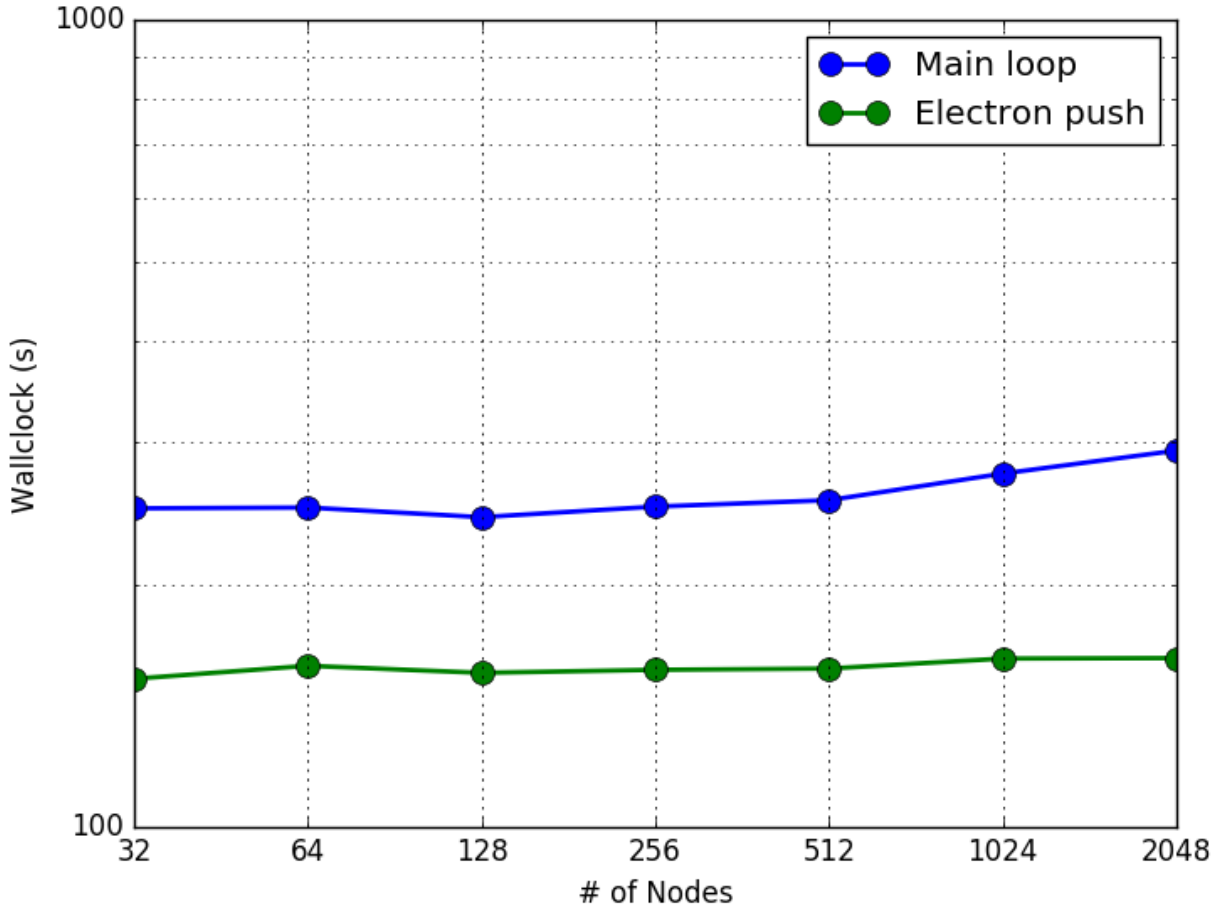
30% scaling deficit in main loop at 4096 nodes (half machine size)

Particle Weak Scaling Parameters

Compute Nodes	Grid Nodes Per Rank	Particles Per Rank
32	3584	0.4 M
64	1792	0.4 M
128	896	0.4 M
256	448	0.4 M
512	224	0.4 M
1024	112	0.4 M
2048	56	0.4 M

- 16 MPI ranks per Node, 16 OpenMP Threads per rank
- 57 000 total grid nodes per plane, 32 planes
- Quadrant Cache mode

XGC1 "Weak Scaling" Up to 2048 KNL Nodes



Weak Scaling in particle structure size for fixed grid size

Grid representative of present production runs (DIII-D tokamak)

60-70% of time in electron push

Slowdown from 32 to 2048 nodes: 20%

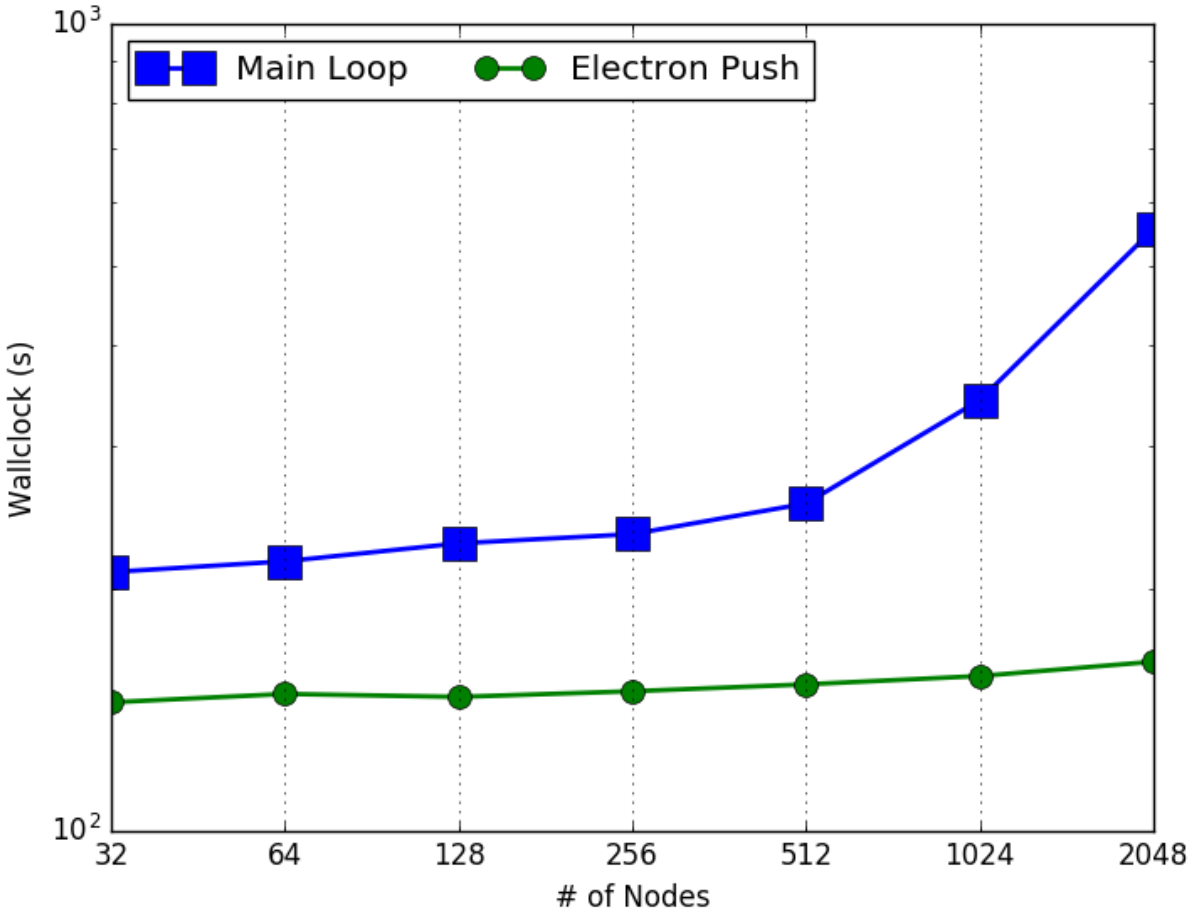
~50% slowdown at full machine size (9600 nodes) by extrapolation

Particle Weak Scaling Parameters

Compute Nodes	Grid Nodes Per Rank	Total Grid Nodes	Particles Per Rank	Total Particles
32	470	7 500	0.4 M	200 M
64	470	15 000	0.4 M	400 M
128	470	30 000	0.4 M	800 M
256	470	60 000	0.4 M	1600 M
512	470	120 000	0.4 M	3200 M
1024	470	240 000	0.4 M	6400 M
2048	470	15 M	0.4 M	12800 M

- 16 MPI ranks per Node, 16 OpenMP Threads per rank
- Quadrant Cache mode

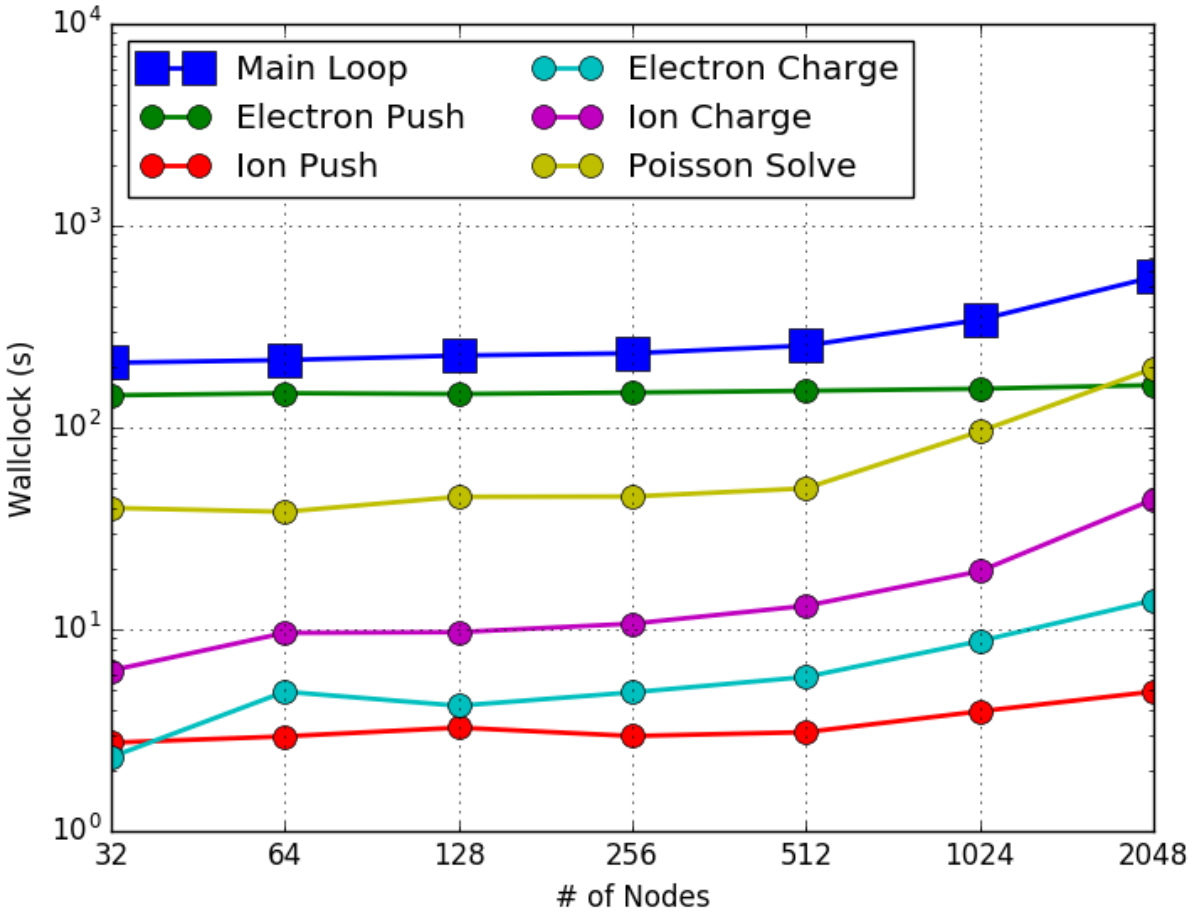
XGC1 Weak Scaling



Weak Scaling in particle structure and grid size

Slowdown from 32 to 2048 nodes: 150%

XGC1 Weak Scaling



Weak Scaling in particle structure and grid size

Slowdown from 32 to 2048 nodes: 150%

Poisson solver mostly responsible for poor scaling

Combination of PETSc library solver and MPI scatter calls

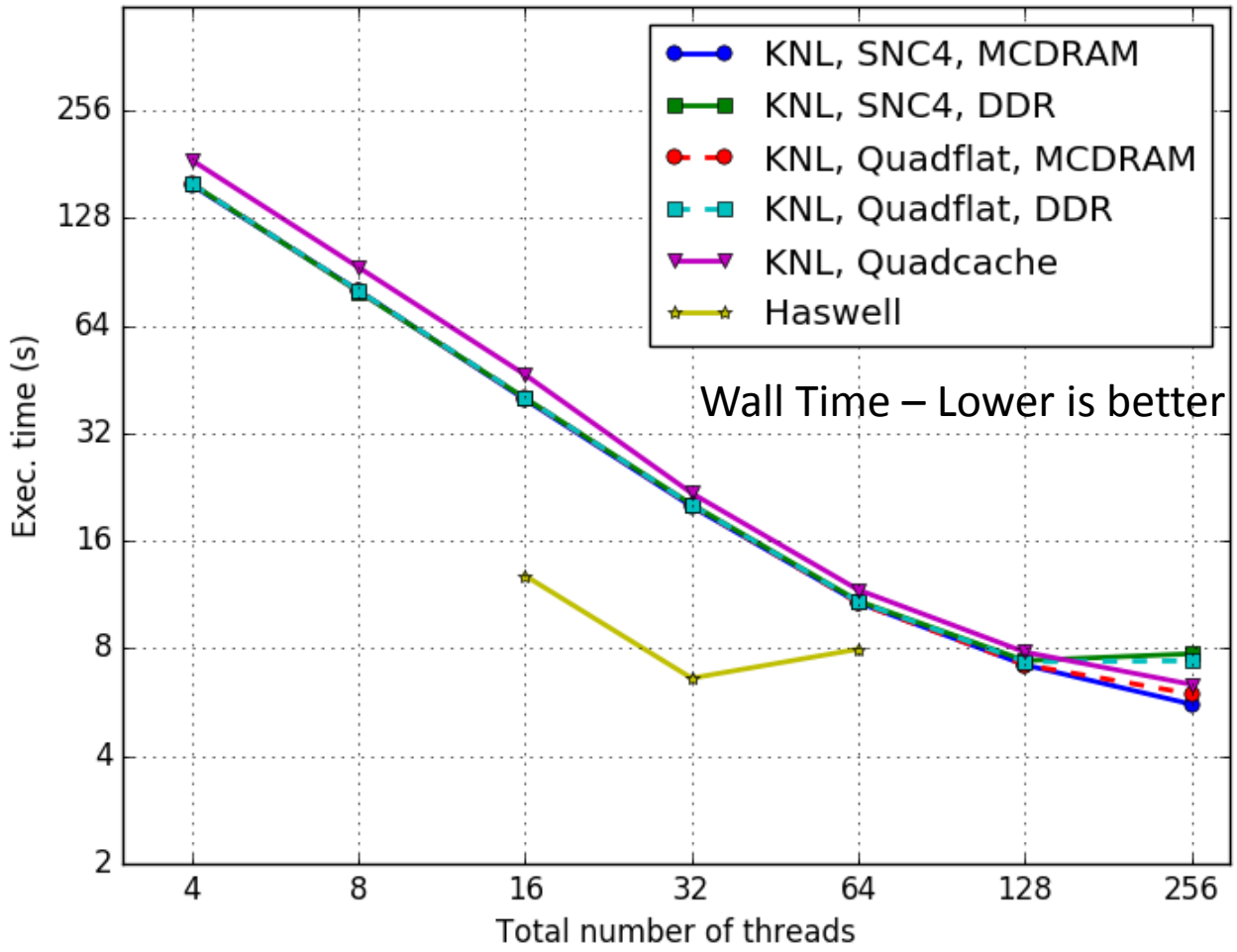
Summary And Conclusions

- **Optimizations have improved vectorization and memory access patterns in XGC1 electron push kernel**
  - Approximately 3x gained in total performance
  - Optimized electron push kernel has roughly equal per-node performance on KNL and Haswell
- **Optimization efforts have been focused using roofline analysis**
  - Focus on enabling vectorization, do not worry about memory bandwidth
  - Theoretically still room for ~10x improvement, what is limiting performance?
    - Memory latency, Memory alignment, Integer operations, Type conversions, ...
- **XGC1 strong scaling is satisfactory for moderate problem size.**
- **The large number of slow cores on KNL has exposed poor weak scaling when pushing towards a large problem size.**
  - Poisson solver has been identified as the main bottleneck.
  - Work is ongoing to resolve the issue.

Backup Slides

Single node thread scaling of electron push kernel

1 Node, 4 MPI ranks per node

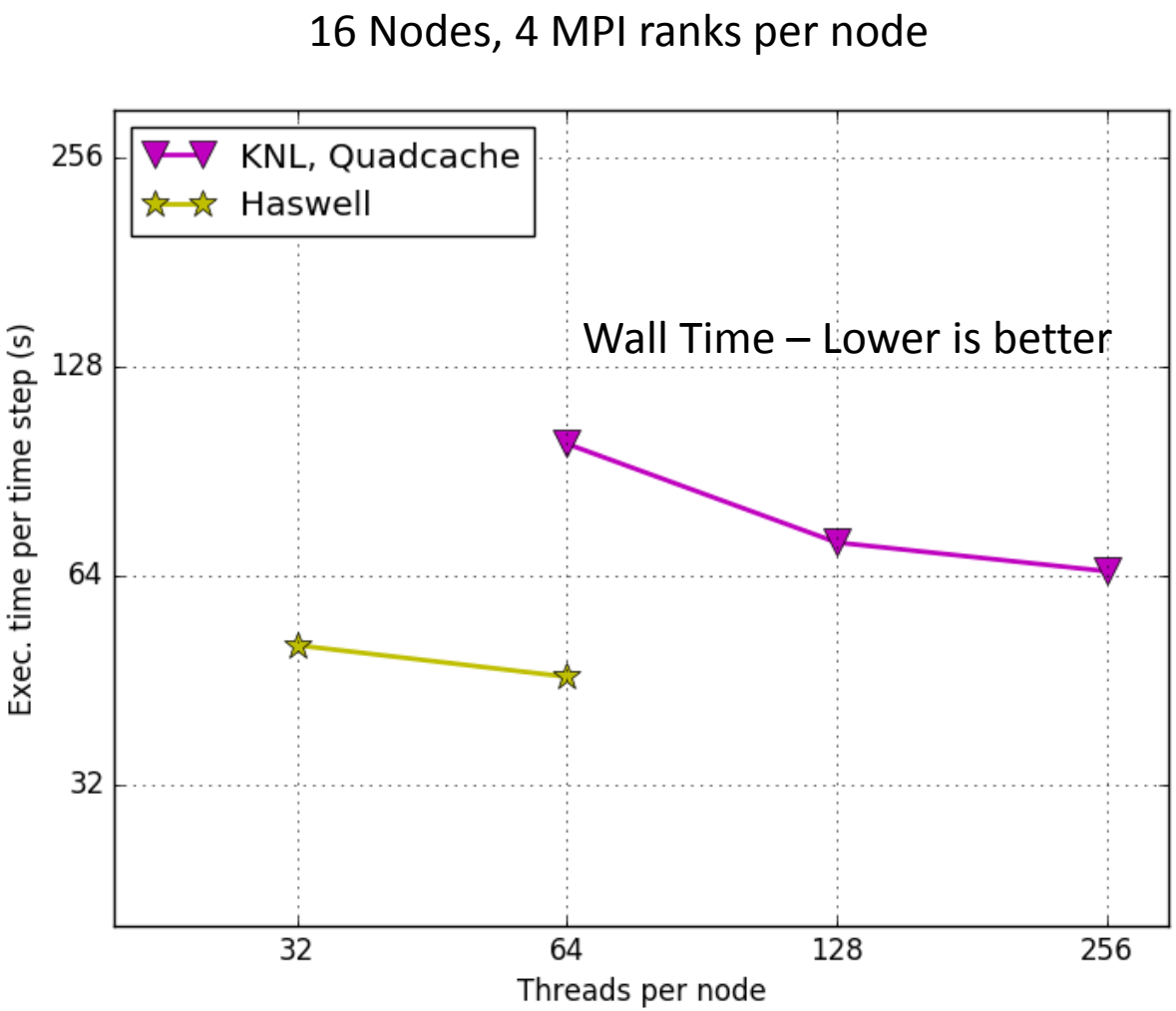


Performance gain from MCDRAM only when using more than 2 threads/core → KNL outperforms Haswell node when all logical threads are used

KNL: 64 physical cores/4 hyper threads
Haswell: 32 physical cores/2 hyper threads

KMP_AFFINITY=compact
KMP_PLACE_THREADS=1
T (N <= 64)
2T (N == 128)
4T (N == 256)
OMP_NUM_THREADS=N

Full Production Code Preliminary Performance Results



- **Ratio of KNL to Haswell performance with 1 thread/core (64 to 32) is in line with kernel results (~2x in favor of Haswell)**
- **Gain from hyper-threading is less significant than in kernel results**
- **Compiler and system bugs encountered at large scale, ongoing work to resolve**