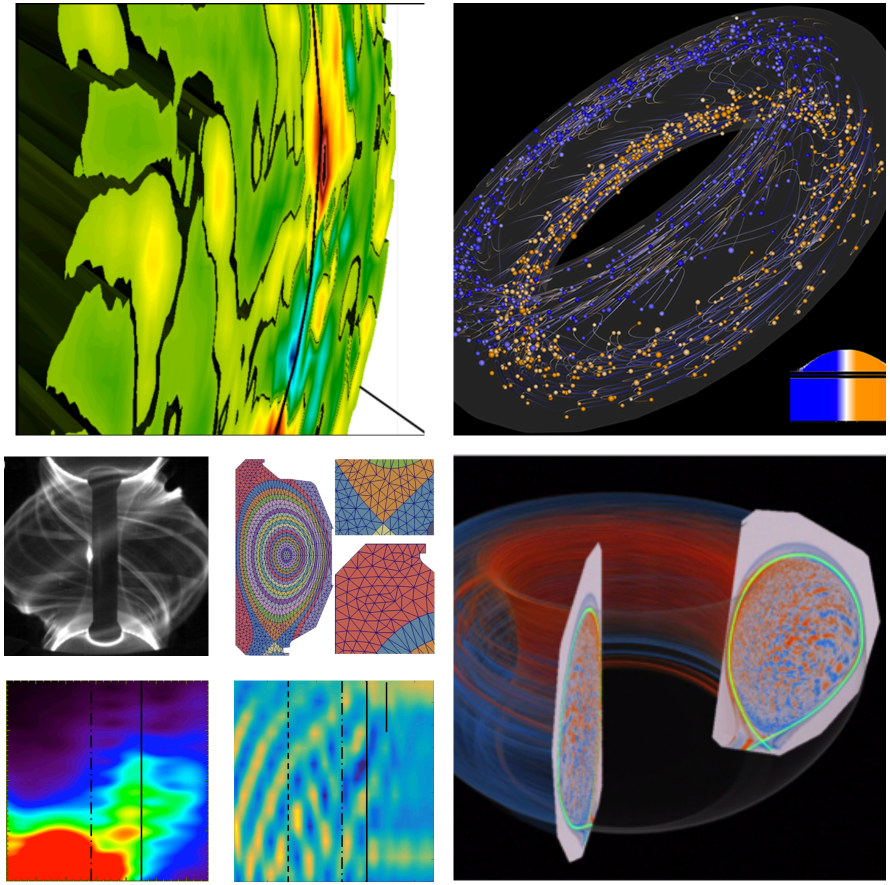
Fusion PIC Code Performance Analysis on the Cori KNL System



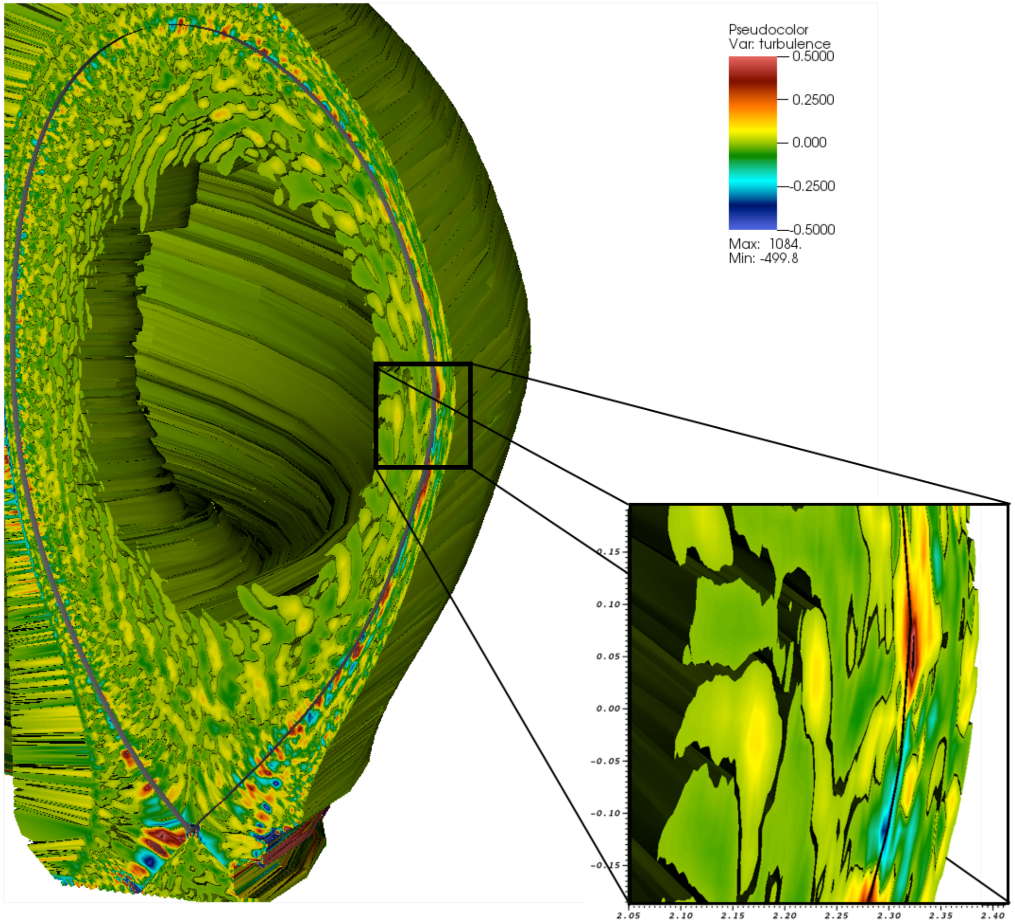
T. Koskela*, J. Deslippe*,
K. Raman, B. Friesen***
*NERSC
** Intel
tkoskela@lbl.gov

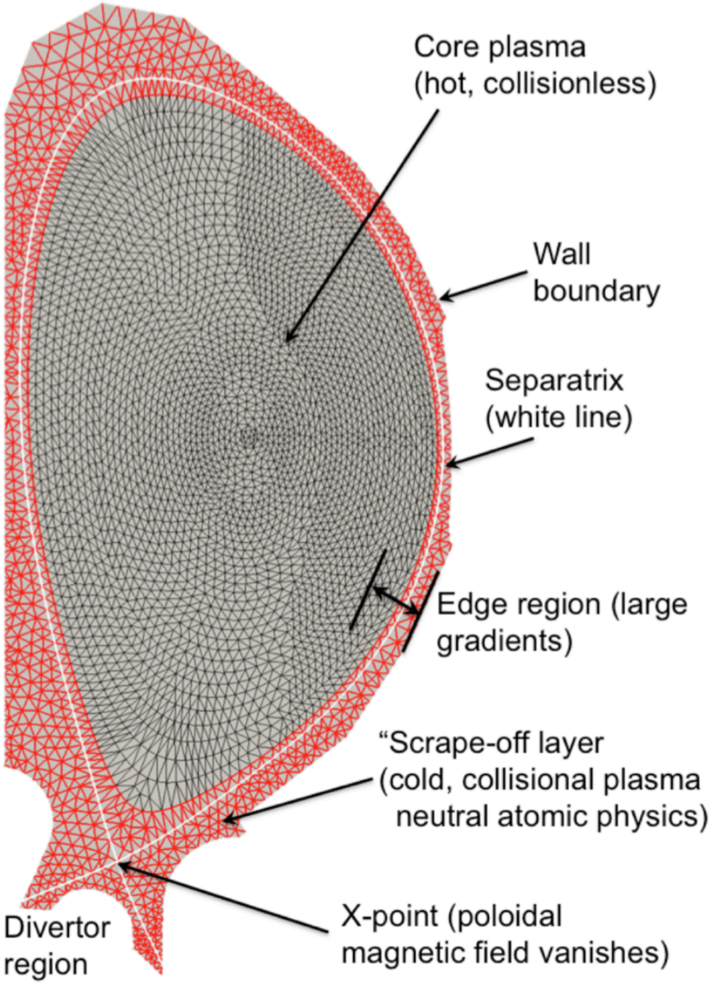
May 18, 2017

Outline

- Introduction to magnetic fusion plasma simulation and XGC1
- Motivation for the electron push mini-app Toypush
- Roofline performance analysis
- Optimization lessons learned
- Summary of obtained speedups

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





Core plasma (hot, collisionless)

Wall boundary

Separatrix (white line)

Edge region (large gradients)

"Scrape-off layer (cold, collisional plasma neutral atomic physics)

Divertor region

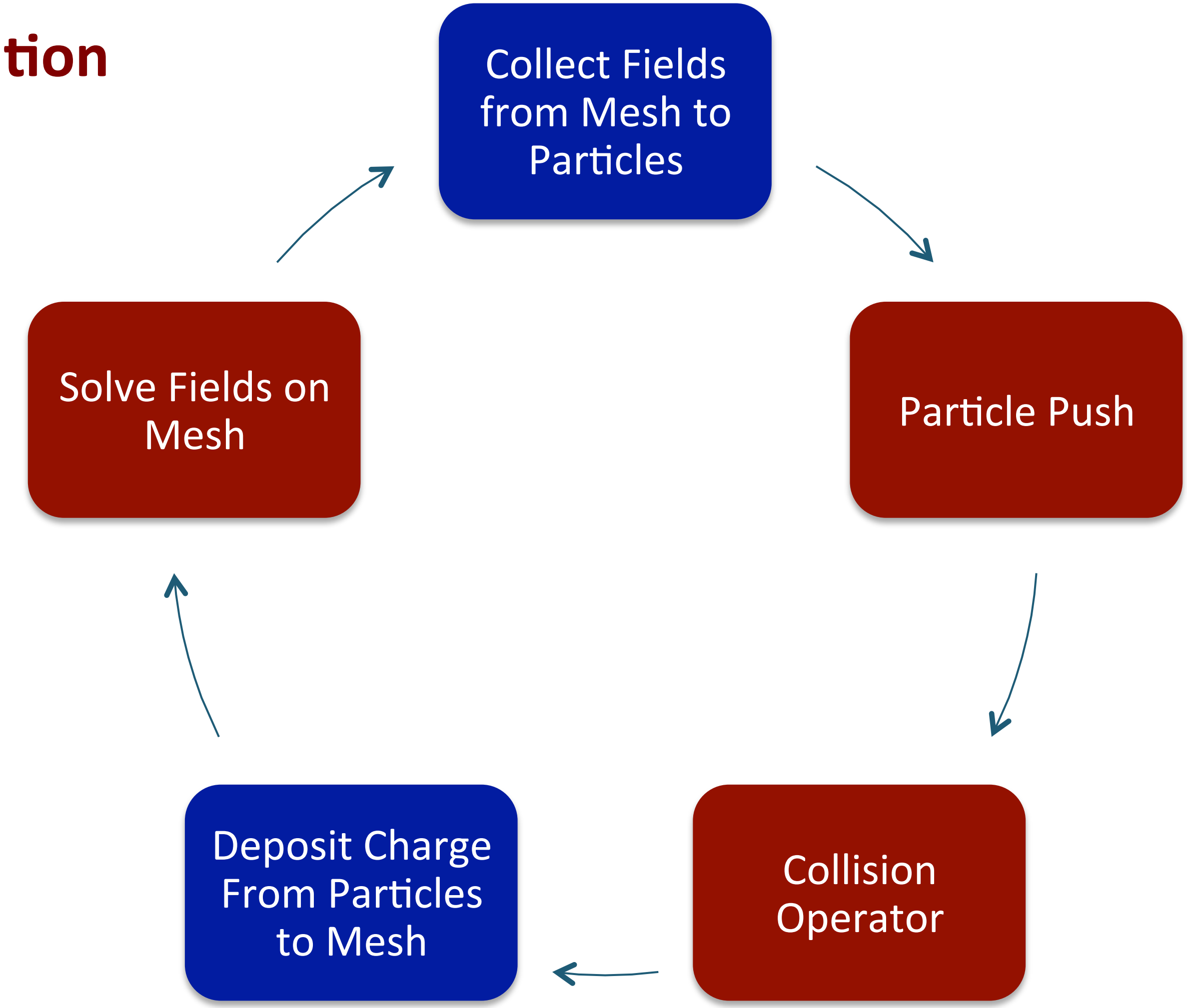
X-point (poloidal magnetic field vanishes)

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

Basic PIC Code Flowchart

Computation

Mapping



Collect Fields from Mesh to Particles

Particle Push

Collision Operator

Deposit Charge From Particles to Mesh

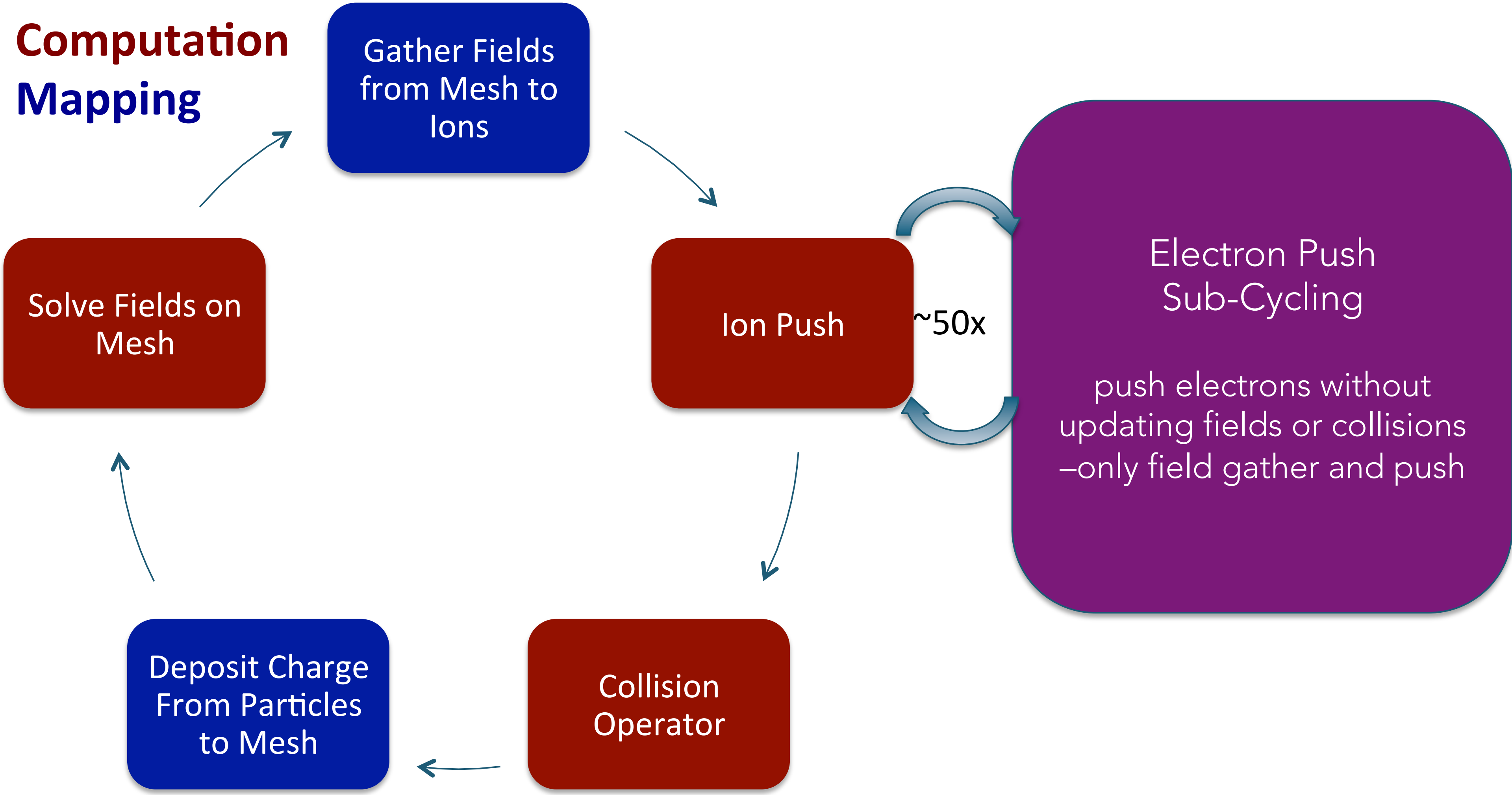
Solve Fields on Mesh

Unique Optimization Challenges in XGC1

- **Complicated Toroidal Geometry**
  - Unstructured mesh in 2D (poloidal) plane(s)
  - Nontrivial field-following (toroidal) mapping between meshes
  - Typical exascale full-f 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 field derivatives in EoM
  - -- Gyro-averaging scheme in field gather
- **Electron Sub-Cycling**

In XGC1 Electron Time Scale is Separated From the Ions in a Sub-Cycling Loop

Computation Mapping



Gather Fields from Mesh to Ions

Ion Push

~50x

Electron Push Sub-Cycling

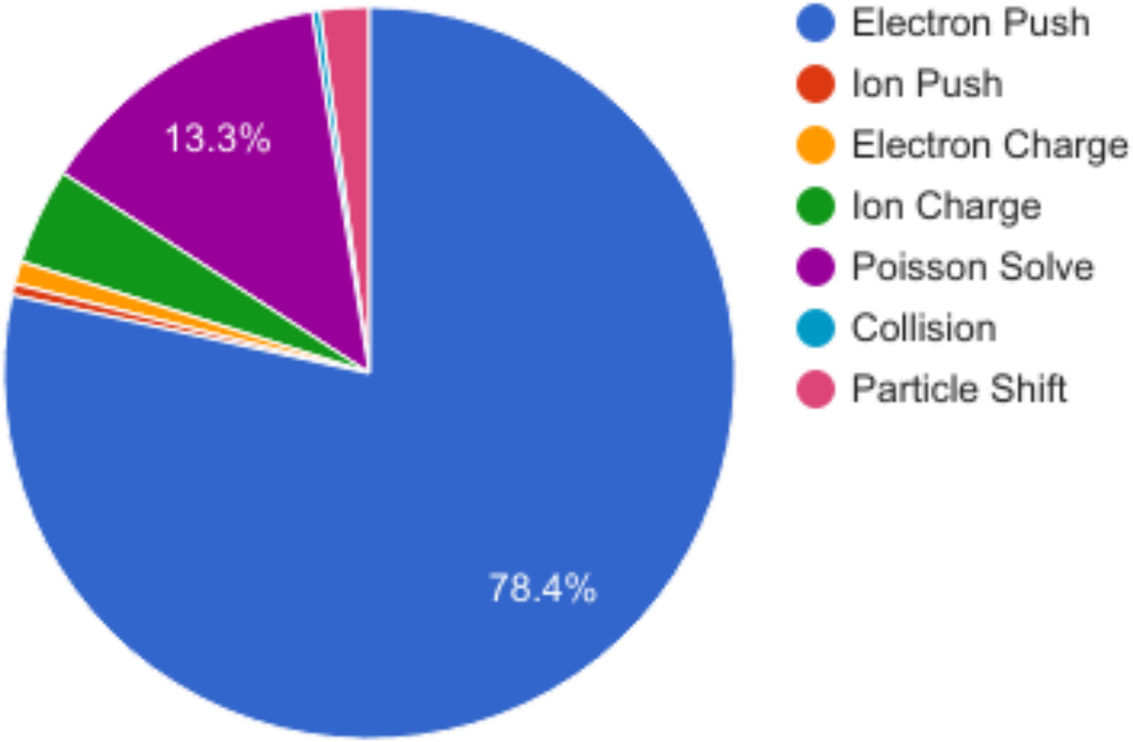
push electrons without updating fields or collisions –only field gather and push

Collision Operator

Deposit Charge From Particles to Mesh

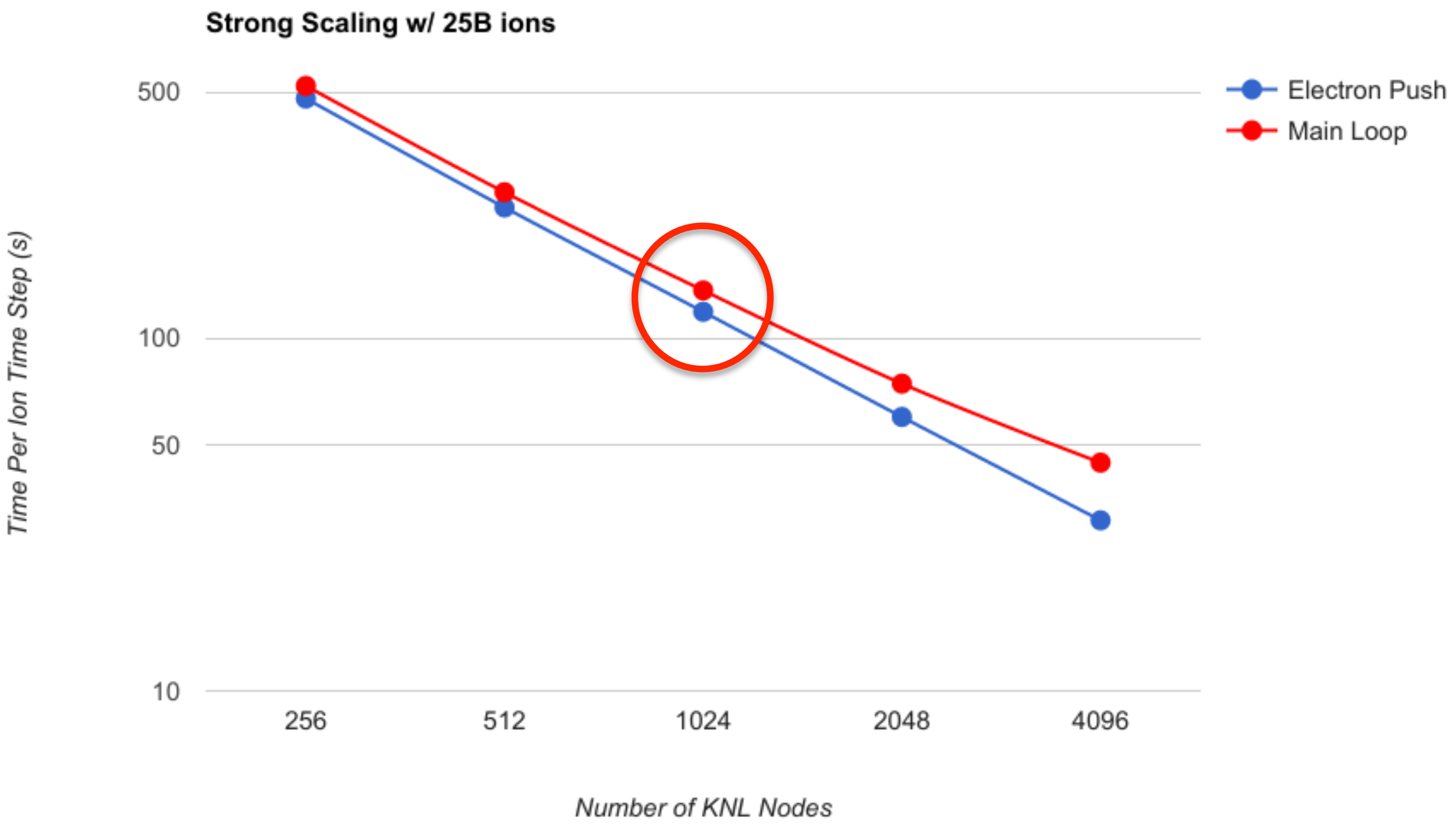
Solve Fields on Mesh

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



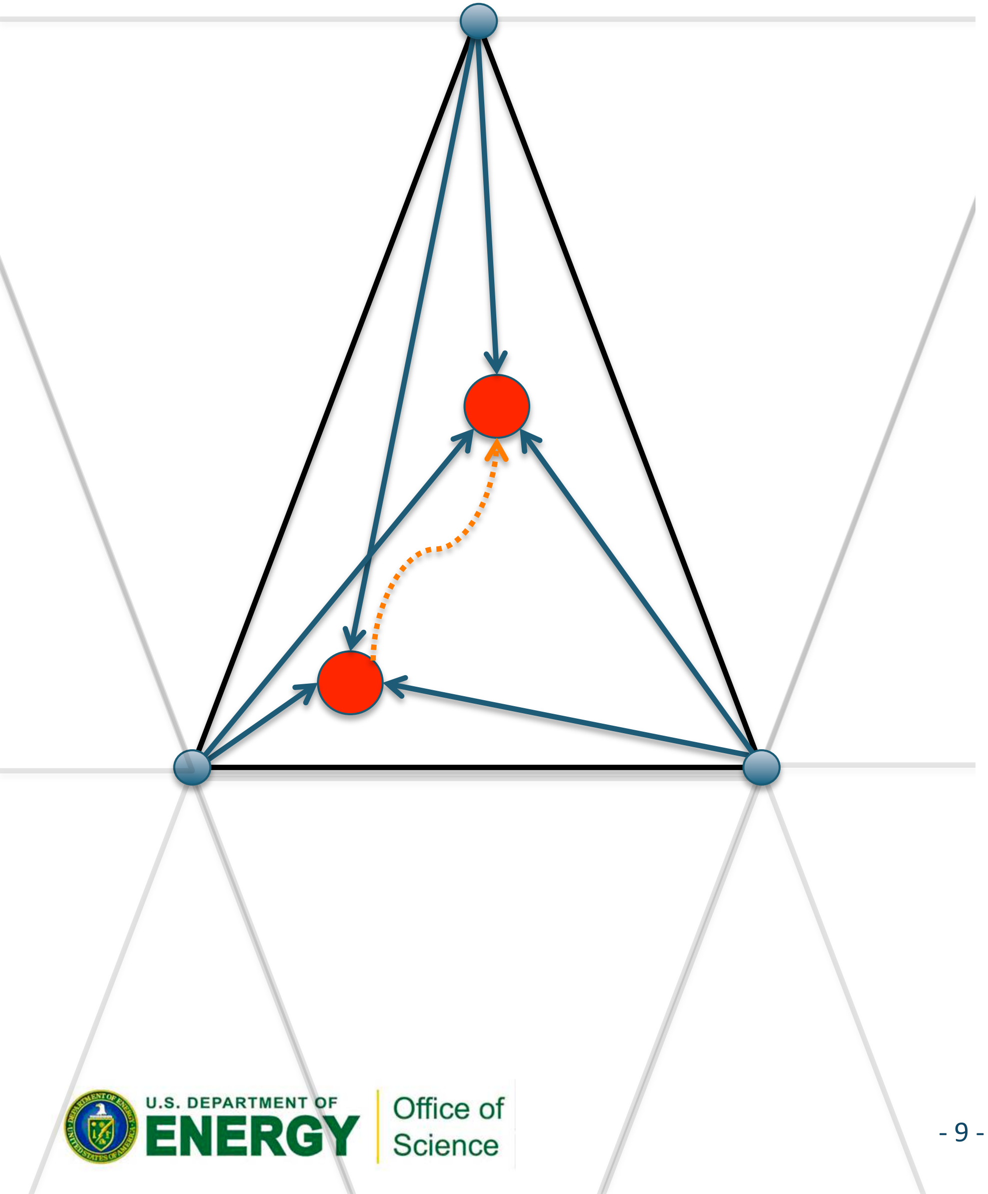
Unoptimized XGC1 Timing on 1024 Cori KNL nodes in quadrant flat mode.

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



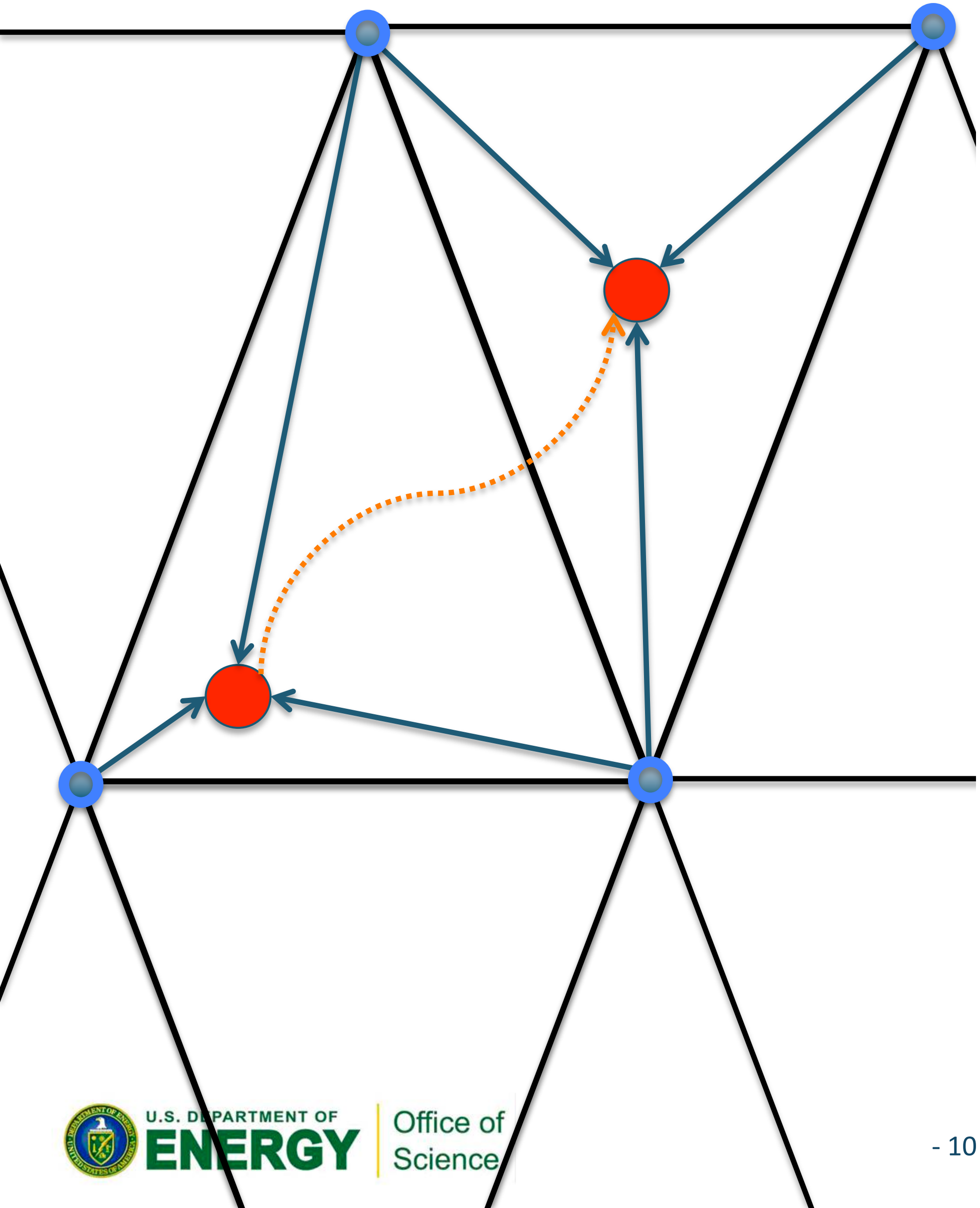
Cori KNL quadrant cache nodes, 16 MPI ranks per node/16 OpenMP threads per rank

Toypush Mini-App Algorithm 1: Single Mesh Element



1. **Interpolate** fields from 3 mesh points to particle position
2. **Calculate force** on particle from fields
3. **Push** particle for time step dt

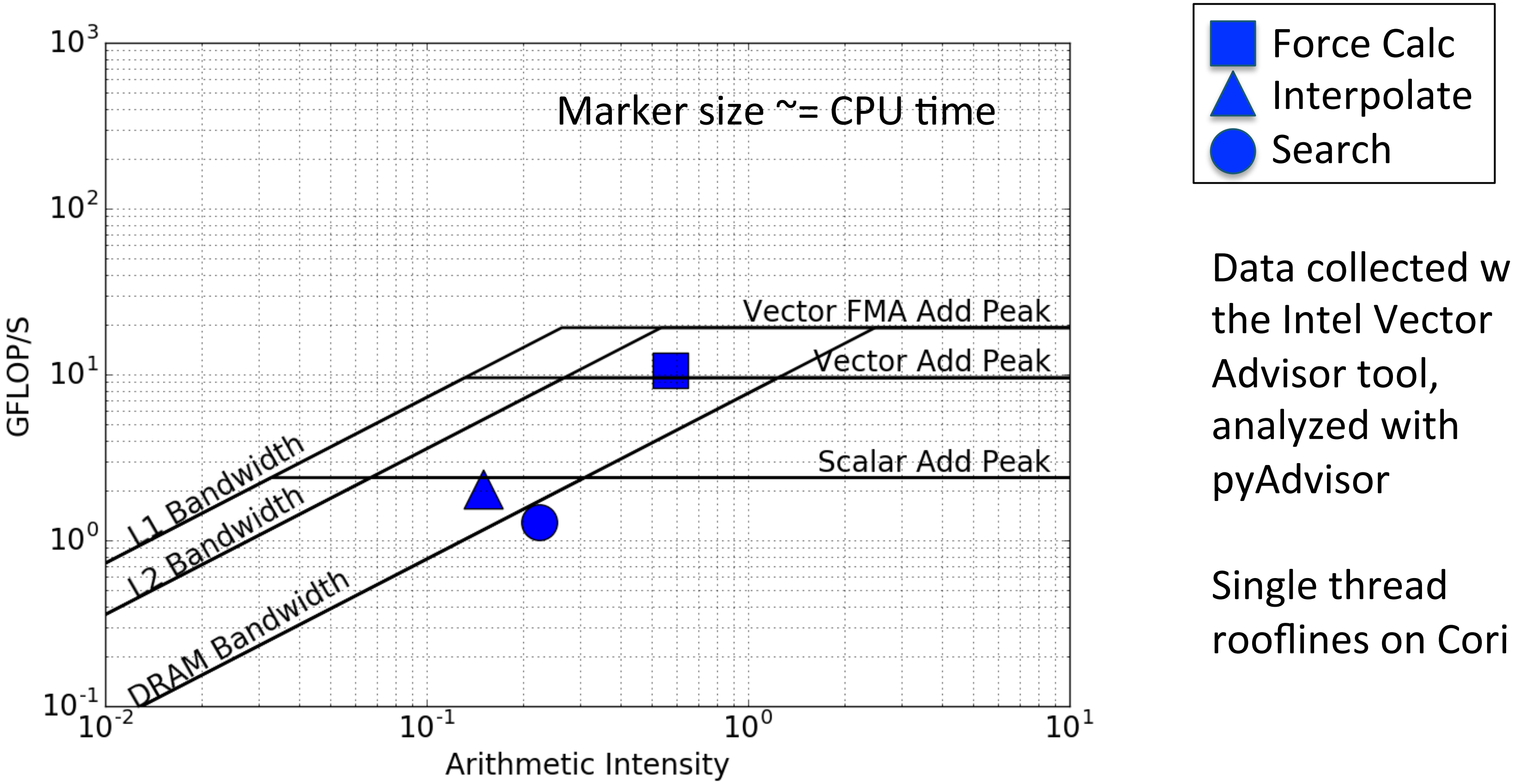
Toypush Mini-App Algorithm 2: Multiple Mesh Elements



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

How Good is the Performance on KNL? Roofline Analysis

- Force Calculation kernel close to vector peak performanc
- Less than scalar peak performance from Interpolate and Search kernels

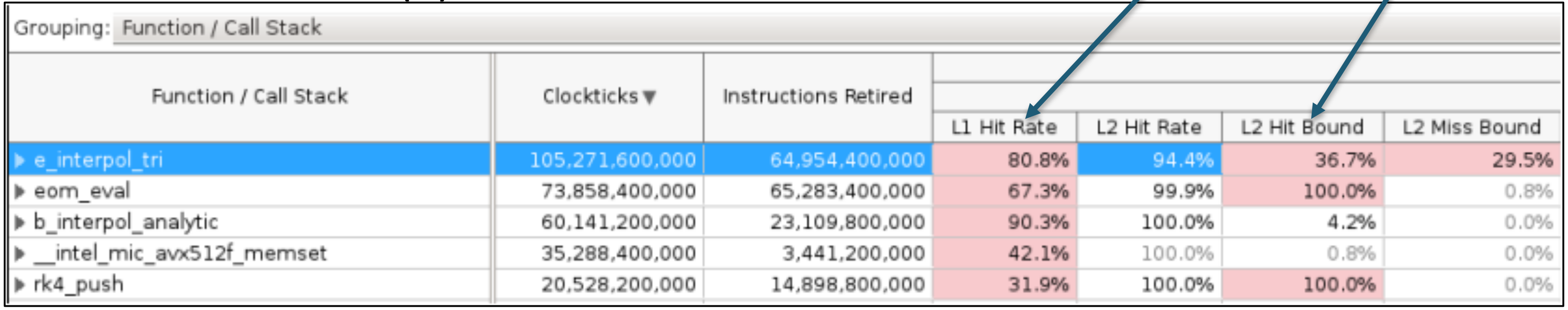


Data collected with the Intel Vector Advisor tool, analyzed with pyAdvisor

Single thread rooflines on Cori KNL

Optimization: L1 Blocking

- **Veclength optimizations**
  - Baseline: 2^(9)

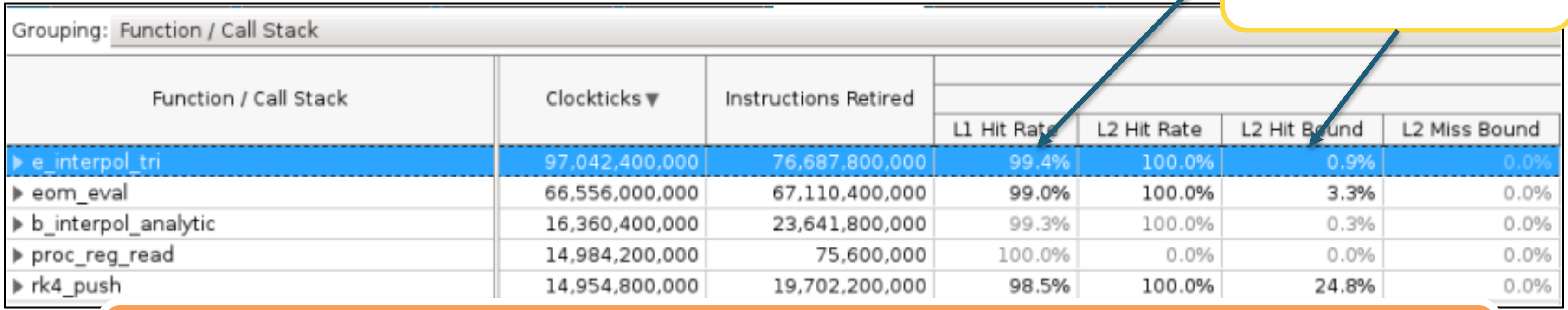


Low L1 Hit Rate, L2 Hit Bound

Grouping: Function / Call Stack

Function / Call Stack	Clockticks ▼	Instructions Retired	L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound
▸ e_interpol_tri	105,271,600,000	64,954,400,000	80.8%	94.4%	36.7%	29.5%
▸ eom_eval	73,858,400,000	65,283,400,000	67.3%	99.9%	100.0%	0.8%
▸ b_interpol_analytic	60,141,200,000	23,109,800,000	90.3%	100.0%	4.2%	0.0%
▸ __intel_mic_avx512f_memset	35,288,400,000	3,441,200,000	42.1%	100.0%	0.8%	0.0%
▸ rk4_push	20,528,200,000	14,898,800,000	31.9%	100.0%	100.0%	0.0%

  - Optimized: 2^(6)



High L1 Hit Rate

Grouping: Function / Call Stack

Function / Call Stack	Clockticks ▼	Instructions Retired	L1 Hit Rate	L2 Hit Rate	L2 Hit Bound	L2 Miss Bound
▸ e_interpol_tri	97,042,400,000	76,687,800,000	99.4%	100.0%	0.9%	0.0%
▸ eom_eval	66,556,000,000	67,110,400,000	99.0%	100.0%	3.3%	0.0%
▸ b_interpol_analytic	16,360,400,000	23,641,800,000	99.3%	100.0%	0.3%	0.0%
▸ proc_reg_read	14,984,200,000	75,600,000	100.0%	0.0%	0.0%	0.0%
▸ rk4_push	14,954,800,000	19,702,200,000	98.5%	100.0%	24.8%	0.0%

~1.5x improvement (MCDRAM Flat); ~2x improvement (DDR Flat)

Indirect Access to Grid Data

Field data is stored on grid nodes, particles access nearest 3 grid nodes indirectly via triangle index.

Interpolation loop is vectorized but not efficiently because of gather loads

18 Gathers per loop iteration
(3 nodes x 3 components x 2)

Intel Compiler Vectorization Report

```
LOOP BEGIN at interpolate_aos.F90(67,48)
   reference itri(iv) has unaligned access
   reference y(iv,1) has unaligned access
   reference y(iv,3) has unaligned access
   reference evec(iv,icomp) has unaligned access
   reference evec(iv,icomp) has unaligned access
…..
irregularly indexed load was generated for the
variable <grid_mapping_(1,3,itri(iv))>, 64-bit
indexed, part of index is read from memory
…..

LOOP WAS VECTORIZED
unmasked unaligned unit stride loads: 6
unmasked unaligned unit stride stores: 3
unmasked indexed (or gather) loads: 18
…..
```

Optimization: Direct Access to Grid Data

Optimization: Group particles that access the same triangle together, access grid nodes directly with a scalar index.

Single element: trivial

Multiple element: Feasible for number of particles >> number of grid nodes

Align arrays during compile time.

Intel Compiler Vectorization Report

```
LOOP BEGIN at
interpolate_aos.F90(72,51)
reference y(iv,1) has aligned access
reference y(iv,3) has aligned access
reference evec(iv, icomp) has aligned
access
…..
SIMD LOOP WAS VECTORIZED
…..
unmasked aligned unit stride loads: 5
unmasked aligned unit stride stores: 3
….
```

~1.6x improvement

Initialization of large arrays with memset

Initialization of large arrays with avx512_memset at every time step before entering vector loop becomes memory bandwidth bound.

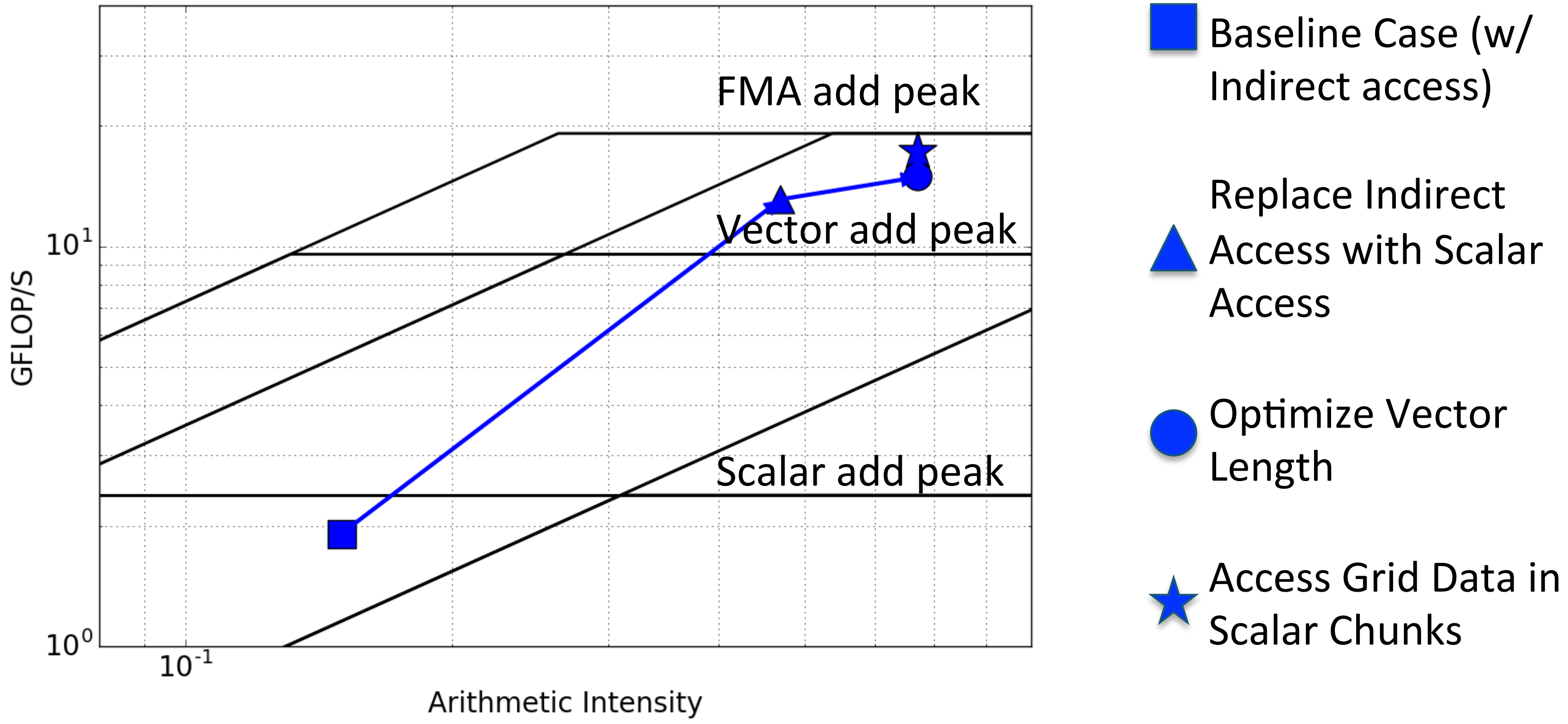
Intel Compiler Vectorization Report

```
LOOP BEGIN at interpolate_aos.F90(57,5)
        memset generated
        loop was not vectorized:
        loop was transformed to memset or memcpy
 LOOP END
```

1. **Initialize array inside the vector loop (if you can)**
2. **Use threads for initialization**

~5% improvement
Higher if no. of particle increases

Interpolation Kernel Performance on Roofline



Optimizations move the kernel to compute bound regime, AI increases with contiguous memory access. Peak compute performance is nearly reached.

Search Routine

Optimization: Remove cycle + OMP SIMD Private

Multiple exits and assumed read after write dependency prevent vectorization

Intel Compiler Vectorization Report

```
LOOP BEGIN at search.F90(62,8)
loop was not vectorized: loop with multiple exits cannot be
vectorized unless it meets search loop idiom criteria
```

Optimization: Replace exit condition with a logical mask

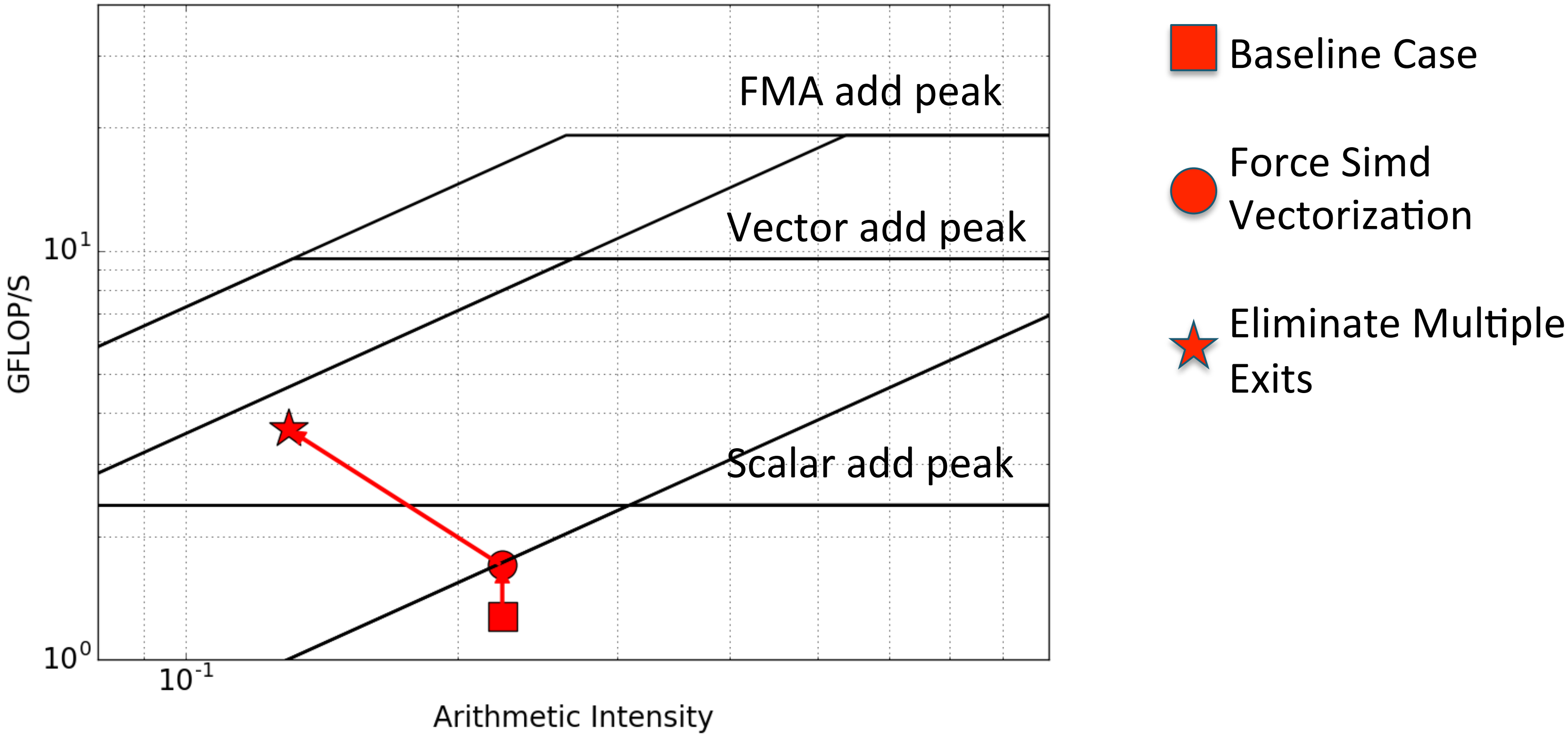
Vectorize with omp simd directive, declare private arrays simd private

Intel Compiler Vectorization Report

```
 LOOP BEGIN at search.F90(66,8)
    reference y(iv,1) has aligned access
    reference y(iv,3) has aligned access
    reference id(iv) has aligned access
    reference continue_search(iv) has aligned access
    data layout of a private variable bc_coords was
optimized, converted to SoA
    OpenMP SIMD LOOP WAS VECTORIZED
    unmasked aligned unit stride loads: 4
                                      le stores: 1
```

1.5x improvement

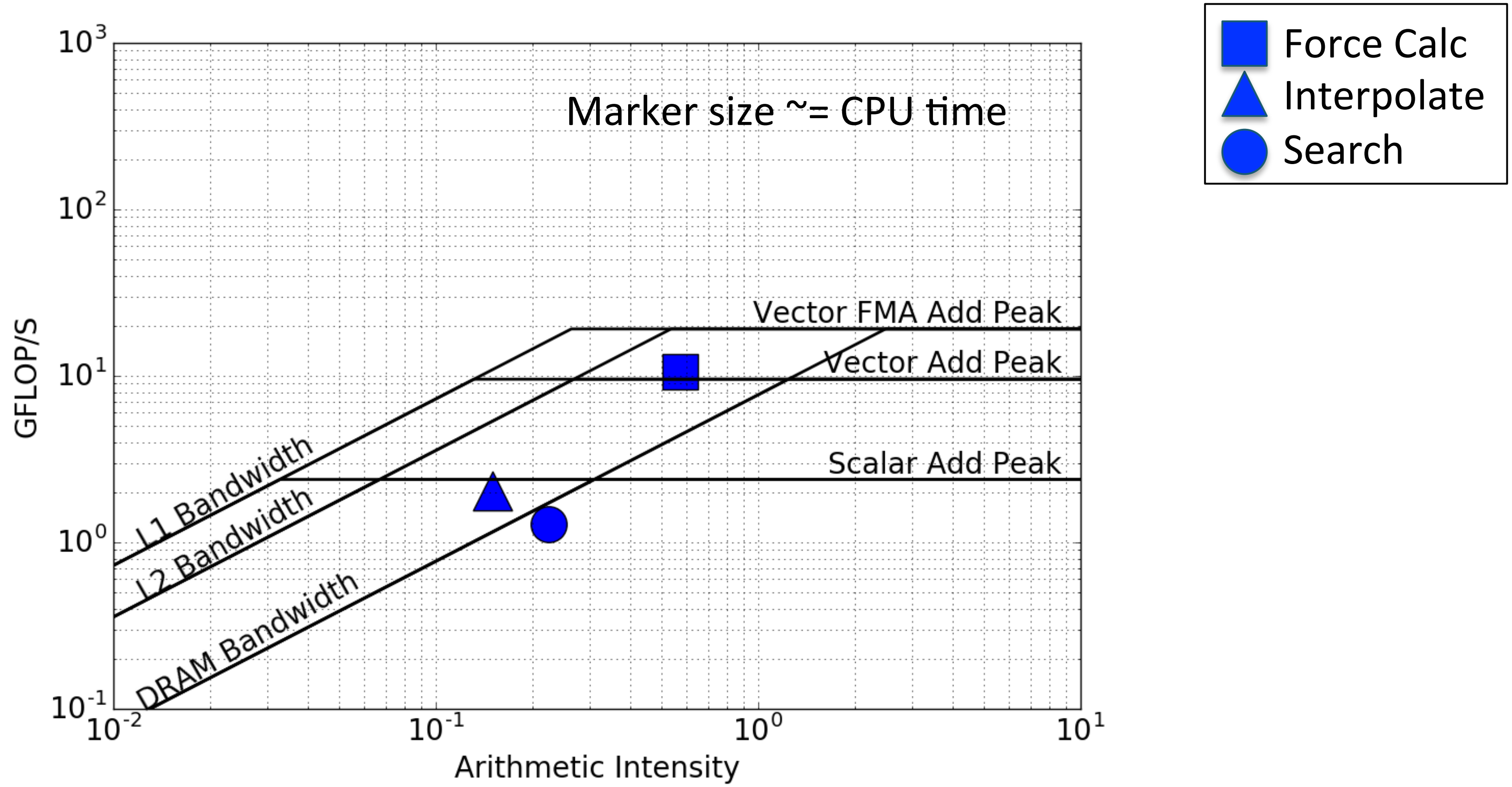
Search Kernel Performance on Roofline



Forced simd vectorization doesn’t work because of multiple exits. Once exits are eliminated the code vectorizes.

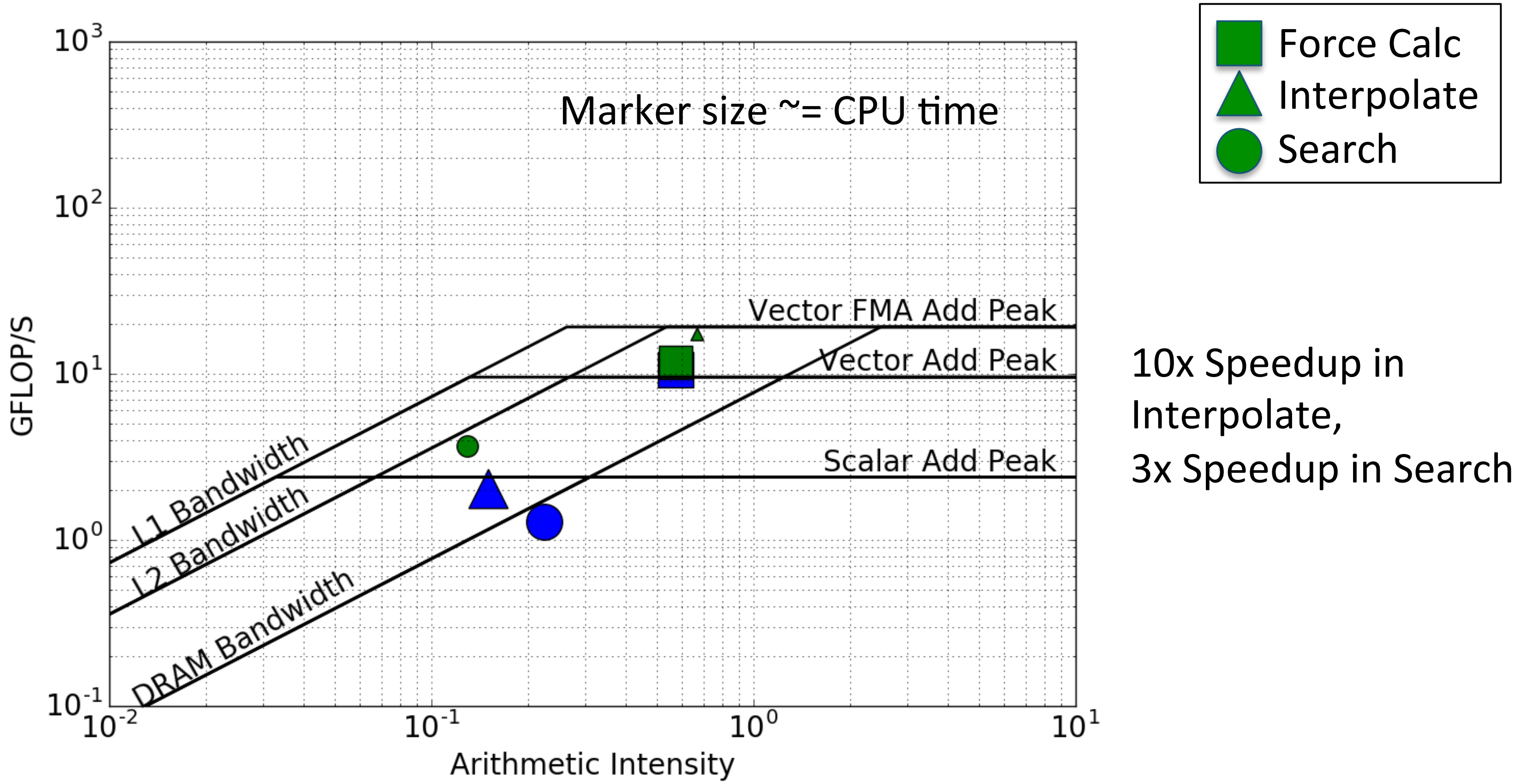
Starting Point On The Roofline

- Good vector performance from the Force Calculation kernel
- Poor performance from Interpolate and Search kernels



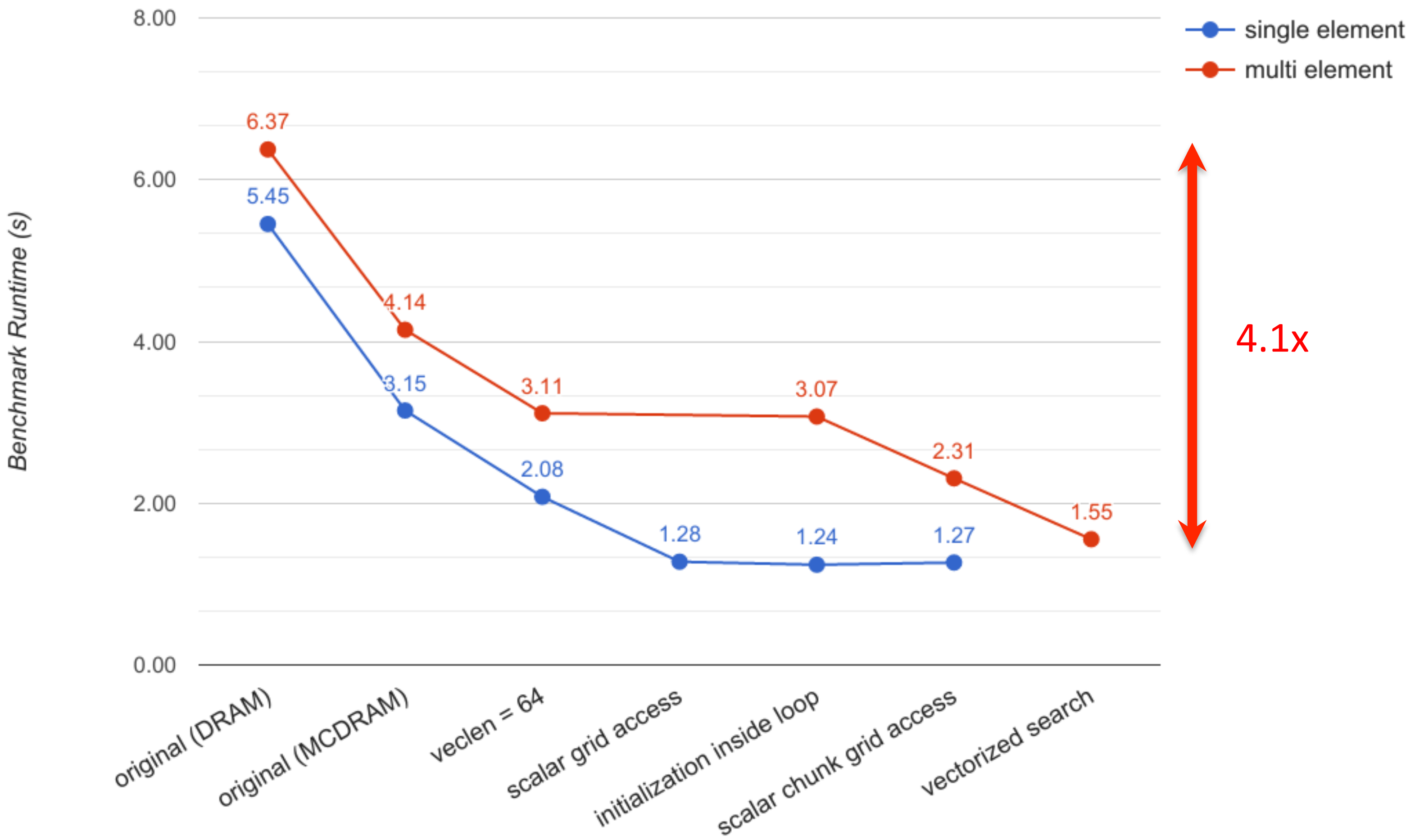
Kernel Improvements on Roofline

- Good vector performance from the Force Calculation kernel
- Interpolate kernel close to theoretical peak, Search close to by L2 bandwidth



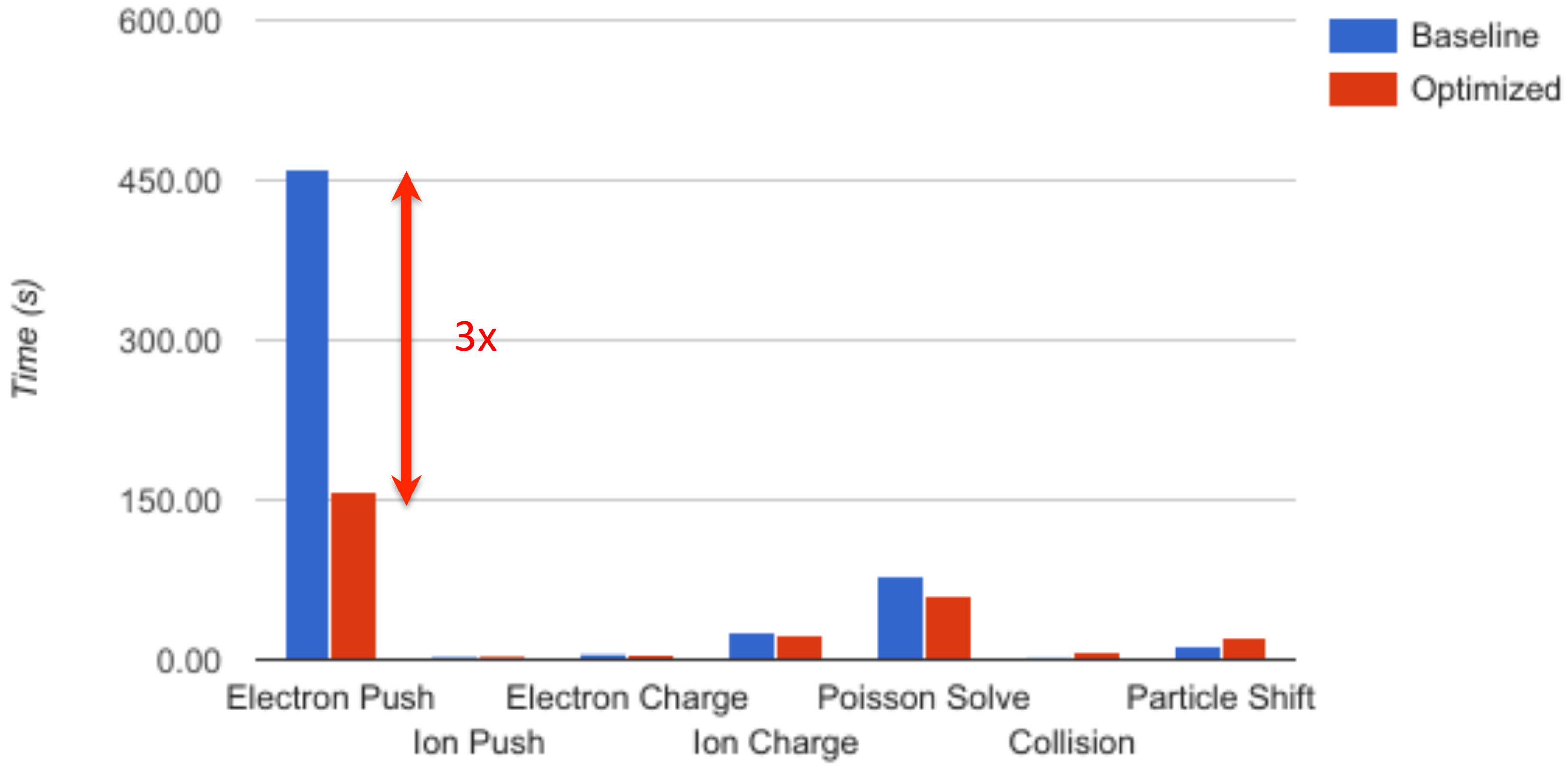
10x Speedup in Interpolate,
3x Speedup in Search

Summary of Mini-app Speedups on Cori KNL



4.1x

Applying Optimizations Back to Electron Push in XGC1 (Work in Progress)



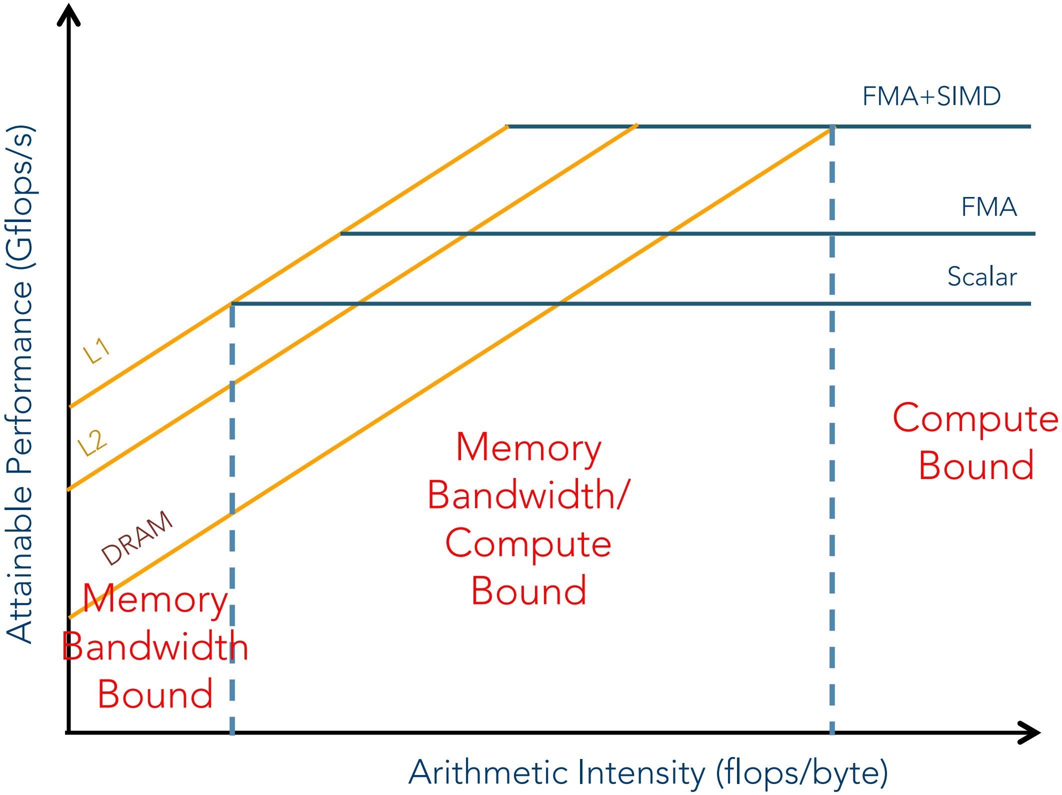
XGC1 Timing on 1024 Cori KNL nodes in quadrant flat mode.

Summary and Conclusions

- **XGC1 is an extreme example of a fusion PIC code due to unstructured mesh, real-space coordinates, and large number of particles per cell.**
- **Electron sub-cycling is used to speed up simulations by sacrificing information at electron time-scale**
  → Most CPU time spent in electron push
  - Almost no communication → On-node performance dominates
- **We optimized a mini-app to attain peak on-node performance in the electron push algorithm on KNL.**
  - Main bottlenecks are search and interpolation
  - We were successful in vectorizing and pushing them close to maximum attainable performance on the roofline chart
- **Porting and developing optimizations to XGC1 is a work in progress, 3x speedup in electron push has been achieved**
  - Electron push remains the most expensive kernel, followed by Poisson solver (PETSc linear algebra)

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.



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