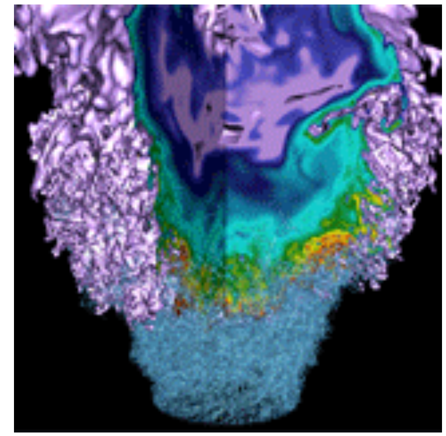
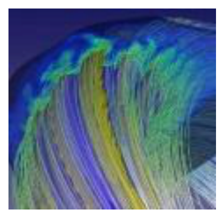
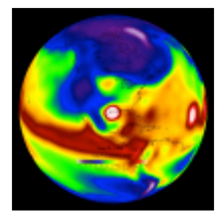
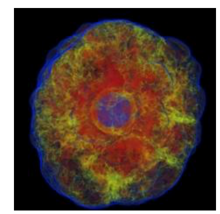
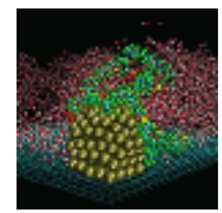
Optimization of the Particle-In-Cell code WARP











NERSC

Mathieu Lobet, Henri vincenti, Remi Lehe, Jean-Luc Vay, Jack Deslippe

mlobet@lbl.gov
NERSC
November 3 2016

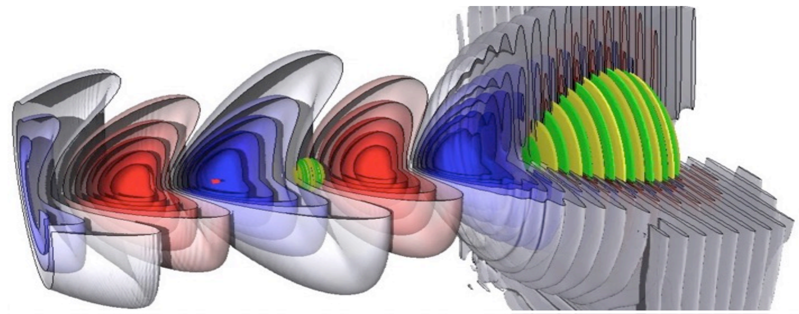
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

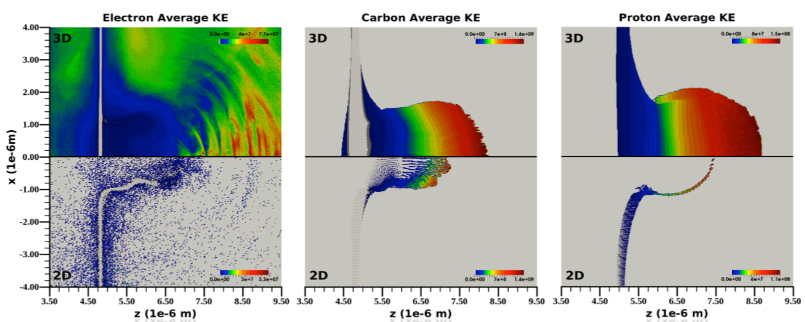
Particle-In-Cell code applications

Application domain: plasma physics, laser-matter interaction, particle accelerators

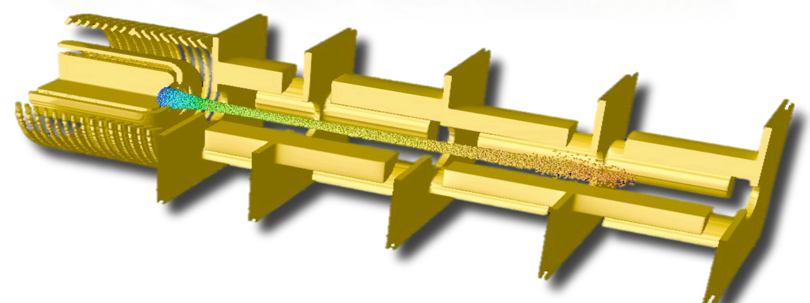
Particle-In-Cell (PIC) methods: solve kinetically collective interactions between the matter (plasmas) seen as charged particles and electromagnetic fields



WARP: Simulation of laser wakefield acceleration of electrons



WARP: Laser-thin foil interaction and ion acceleration



WARP: Conventional beam accelerators

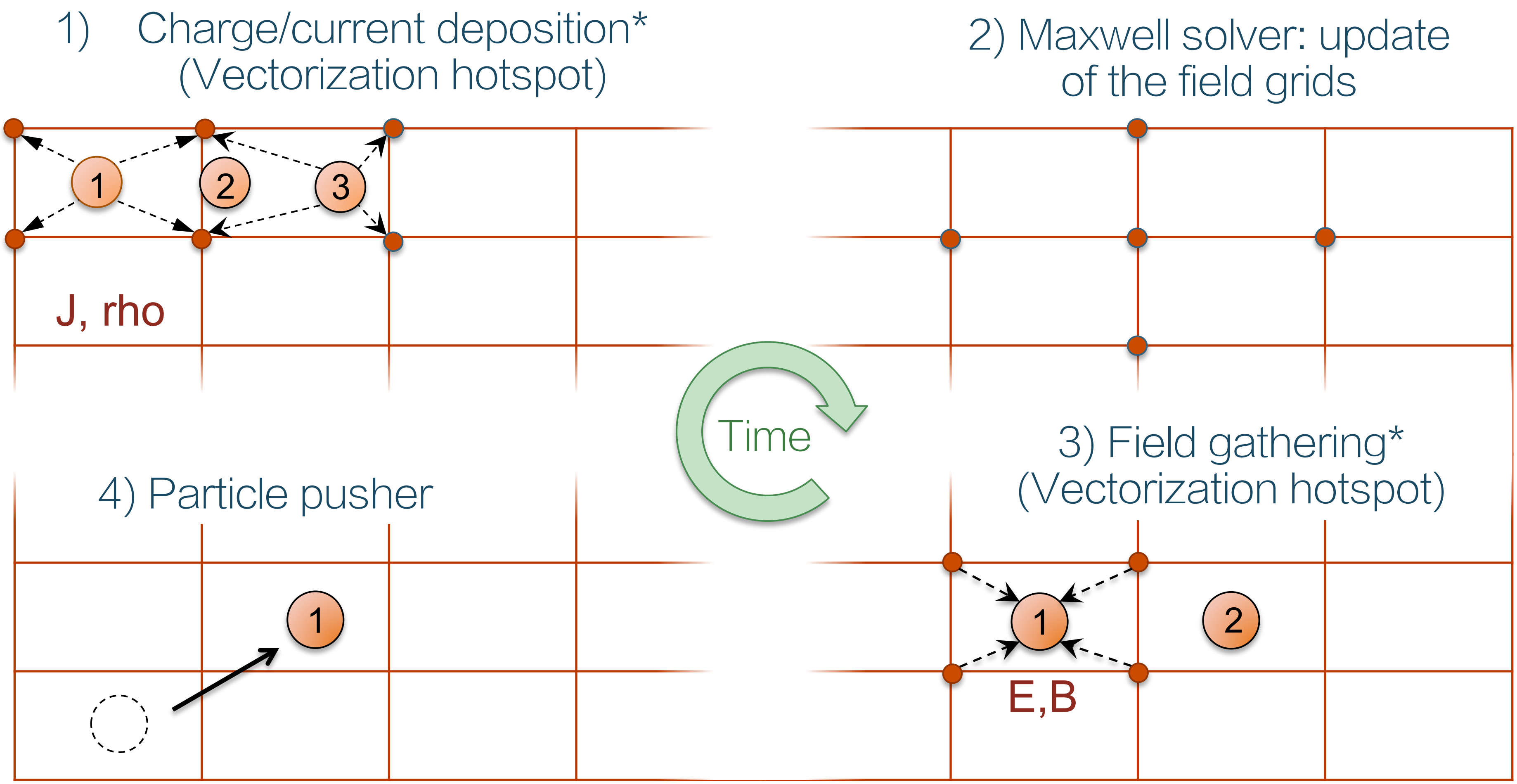
The high-performance library PICSAR

PICSAR (Particle-In-Cell Scalable Application Resource): a high-performance Fortran/Python Particle-In-Cell library targeting MIC architectures.

- designed to be interfaced with the PIC code WARP [2] used at Berkeley Lab
- soon released as an open-source project (already available upon demand)
- selected code for the NERSC Exascale Science Applications Program [1] (NESAP) that aims at preparing the arrival of the super-computer CORI phase II equipped of Intel Xeon Phi KNL.

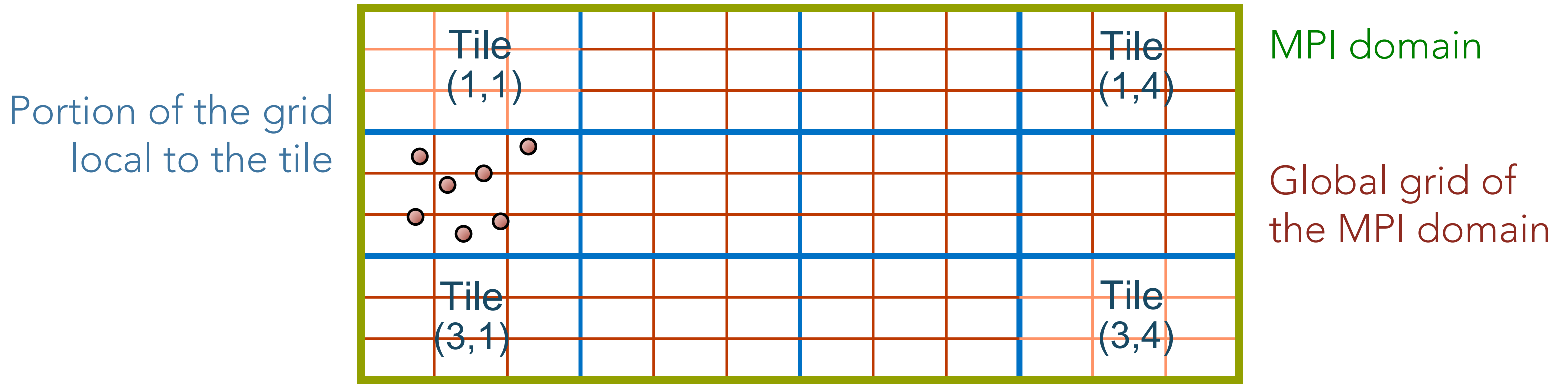
[1] http://www.nersc.gov/users/computational-systems/cori/nesap/
[2] http://warp.lbl.gov/

The 4 main steps of the Particle−In−Cell loop



* Interpolations steps between particle and the grids

Optimization: Tiling (cache blocking) + OpenMP (shared memory) [1]



- Tiling: new subdivision into tile inside MPI domains: local field grids + guard cells from the global grids, local particle property arrays
- Tile size:
  - field grids can fit in L2 cache (main constraint)
  - Particle arrays can partially or entirely fit in L3 on Haswell
- Tiles are handled by OpenMP
  - Number of tiles >> number of threads = load balancing between the tiles

Better memory locality and cache reuse.

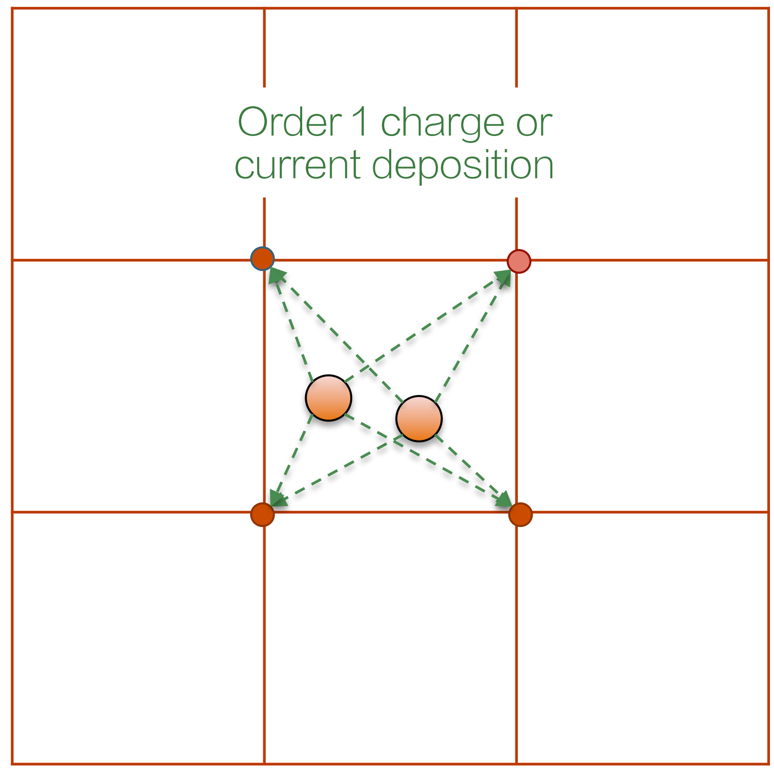
[1] H. Vincenti et al, ArXiv 1061.02056 (2016)

Vectorization bottleneck of the classical charge/current deposition

```
! Charge deposition simplified algorithm

For each particle in a tile:

    1) Determine nearby nodes on the charge grid

    2) Compute current/charge of the particle

    3) Deposit contributions to the charge grid
```

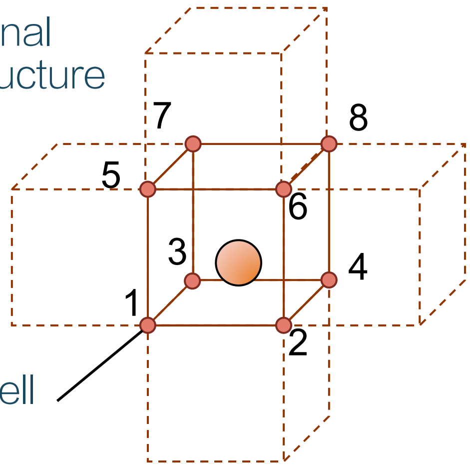
- Conditions (if) removed from the inner loop: order-specific functions
- Step 1) contains type conversions and roundings (not good but can be vectorized)
- Step 2) can be vectorized
- Step 3) prevents vectorization due to memory races when 2 particles are in the same cell
- Grid nodes not aligned in memory: gather/scatter

- Current grids Jx(NCELLS), Jy(NCELLS), Jz(NCELLS)
- Charge rho(NCELLS)



Vectorization bottleneck of the classical charge/current deposition

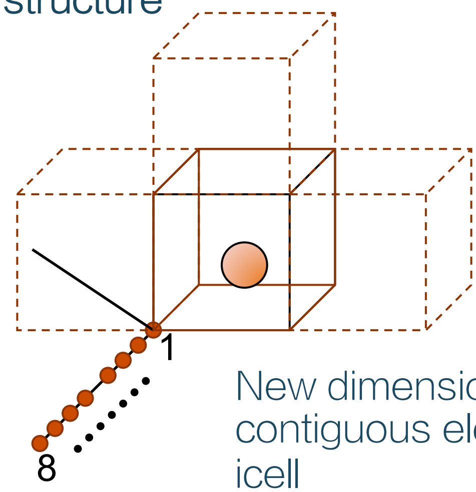
Original
Rho structure



Cell icell

New
Rhocells structure

Cell icell



New dimension for 8 contiguous elements at icell

```
! Charge deposition optimized algorithm

DO i=1,NUMBER_OF_PARTICLES,SIZE_VECT:

    !$OMP SIMD
    DO ip=1,SIZE_VECT:

       1) Determine nearby nodes on current grids
       and store them for the SIZE_VECT particles
       2) Compute contributions for each node

    DO ip=1,SIZE_VECT

        !$OMP SIMD
        DO k=1,8

           3) Add contributions in the temporary
           array structure Rhocells

Do ic=1,NUMBER_OF_CELLS
    4) Reduction of rhocells in rho
```

- New dimension in the current and charge array to access vertices of a cell in a contiguous way
- Enable vectorization of the deposition with no memory races, no gather/scatter
- Reduction at the end in the original structure: Non-efficient vectorization but in O(Ncells) with Ncells << Nparticles

[1] H. Vincenti et al, ArXiv 1061.02056 (2016)

Benchmarking and profiling test case

Test case: homogeneous thermalized plasma

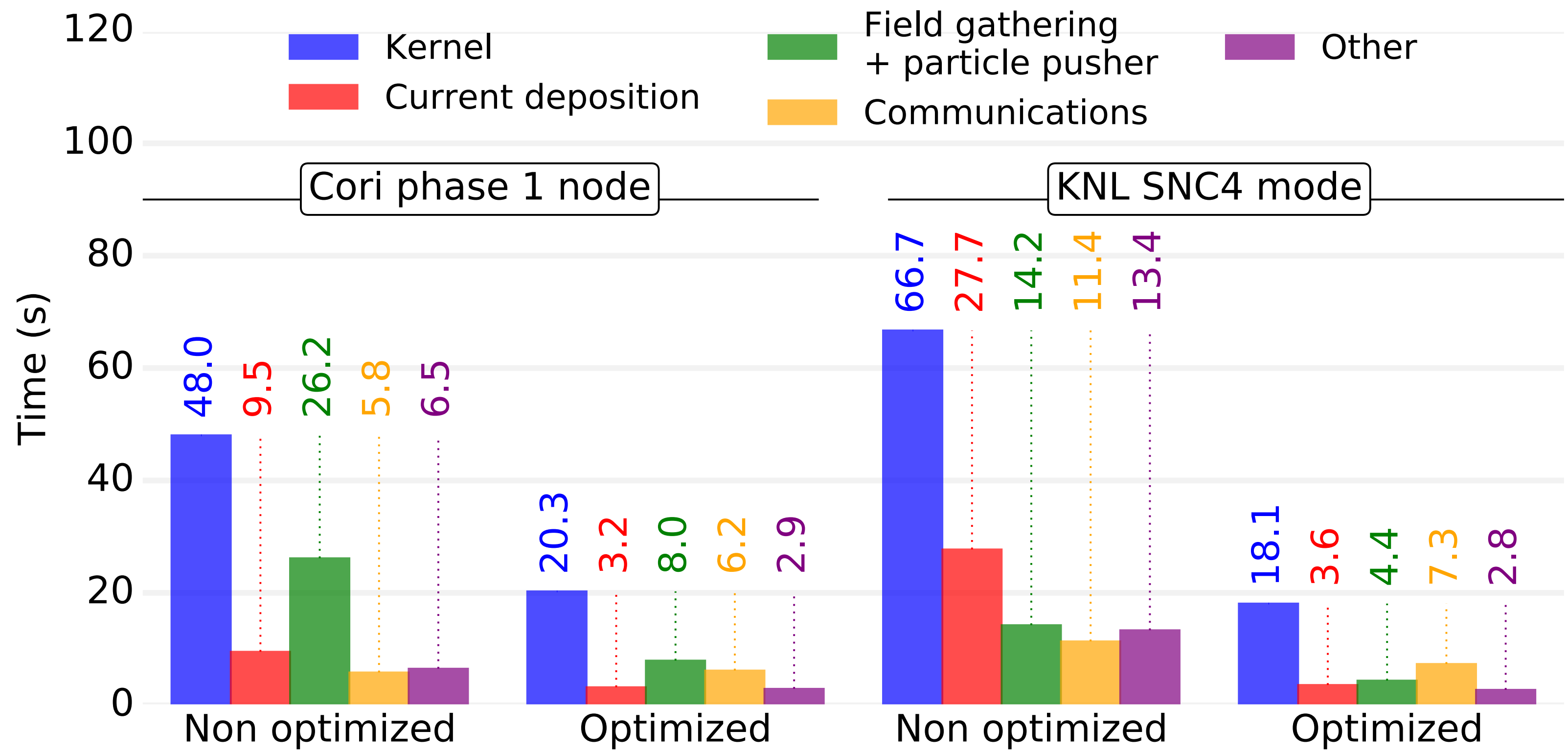
- Balanced load: same amount of particles between MPI domains and tiles

On KNL, the entire problem fits in MCDRAM

Systems	NERSC Cori phase 1 node 2 Intel Haswell processors	KNL NERSC white boxes 64 core KNL SNC4 flat mode*
Configuration 1 (Non optimized)	32 MPI processes	64 MPI processes
Configuration 2 (Optimized)	2 MPI processes, 16 OpenMP threads per processor	4 MPI processes, 32 OpenMP threads per task (hyperthreading)

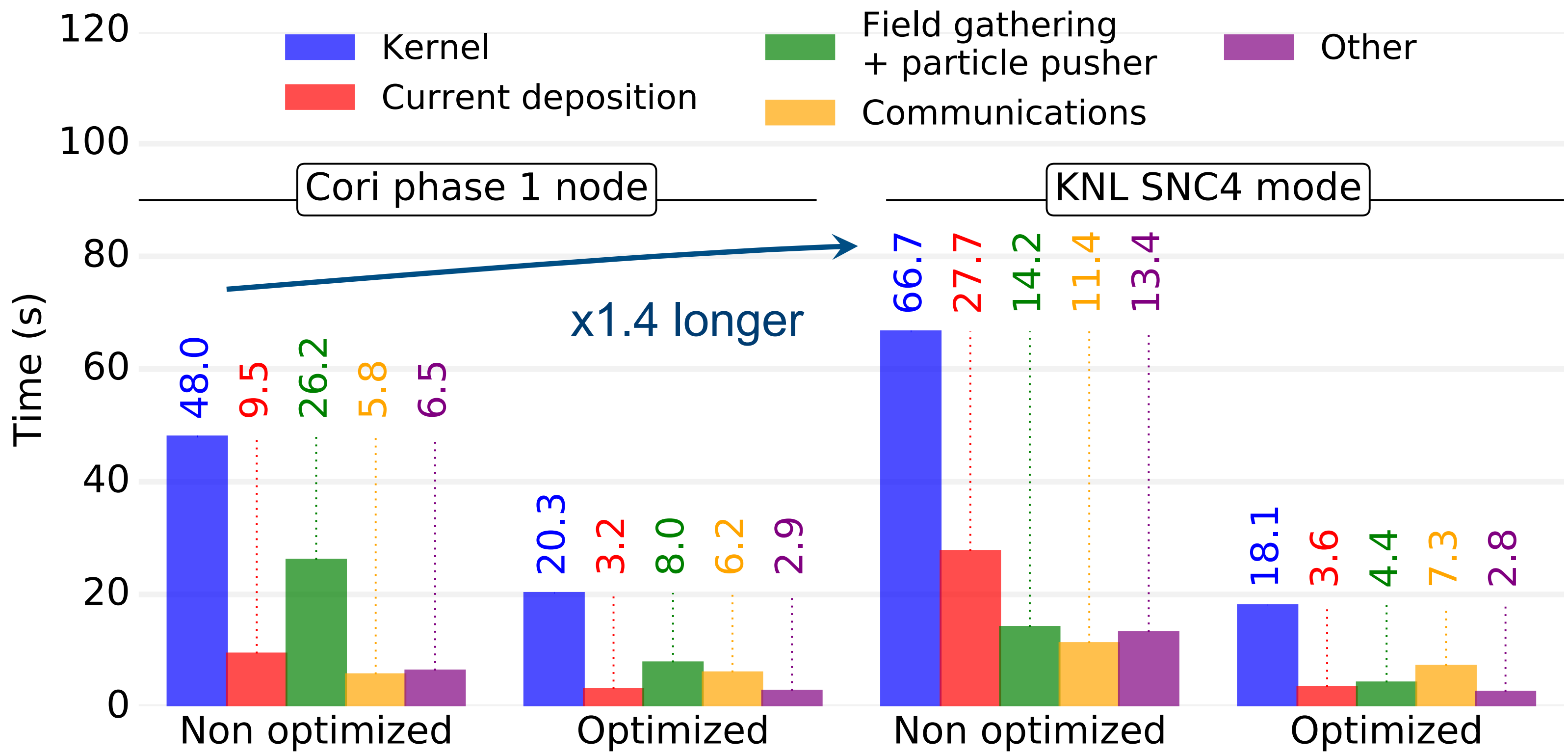
*Similar performance results with Quadrant flat and SNC2 modes

Performance overview on Haswells and KNL for order 1 interpolation method



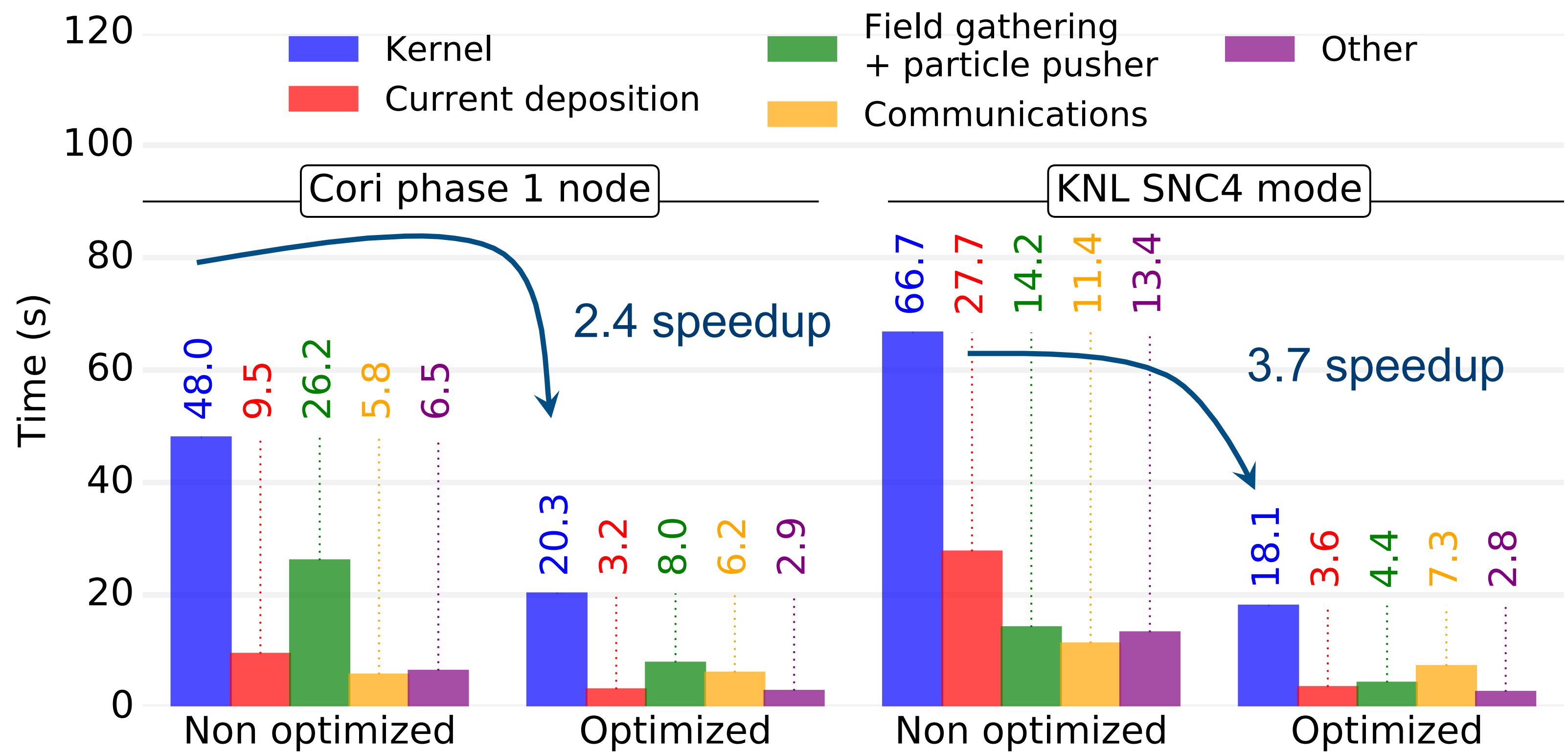
- Kernel: the main PIC loop (code without the initialization and diagnostics)
- Order 1: order 1 interpolation for the current/charge deposition and the field gathering
- Order 3: order 3 interpolation for the current/charge deposition and the field gathering
- Other: particle sorting, Maxwell solver, charge deposition

Performance overview on Haswells and KNL for order 1 interpolation method



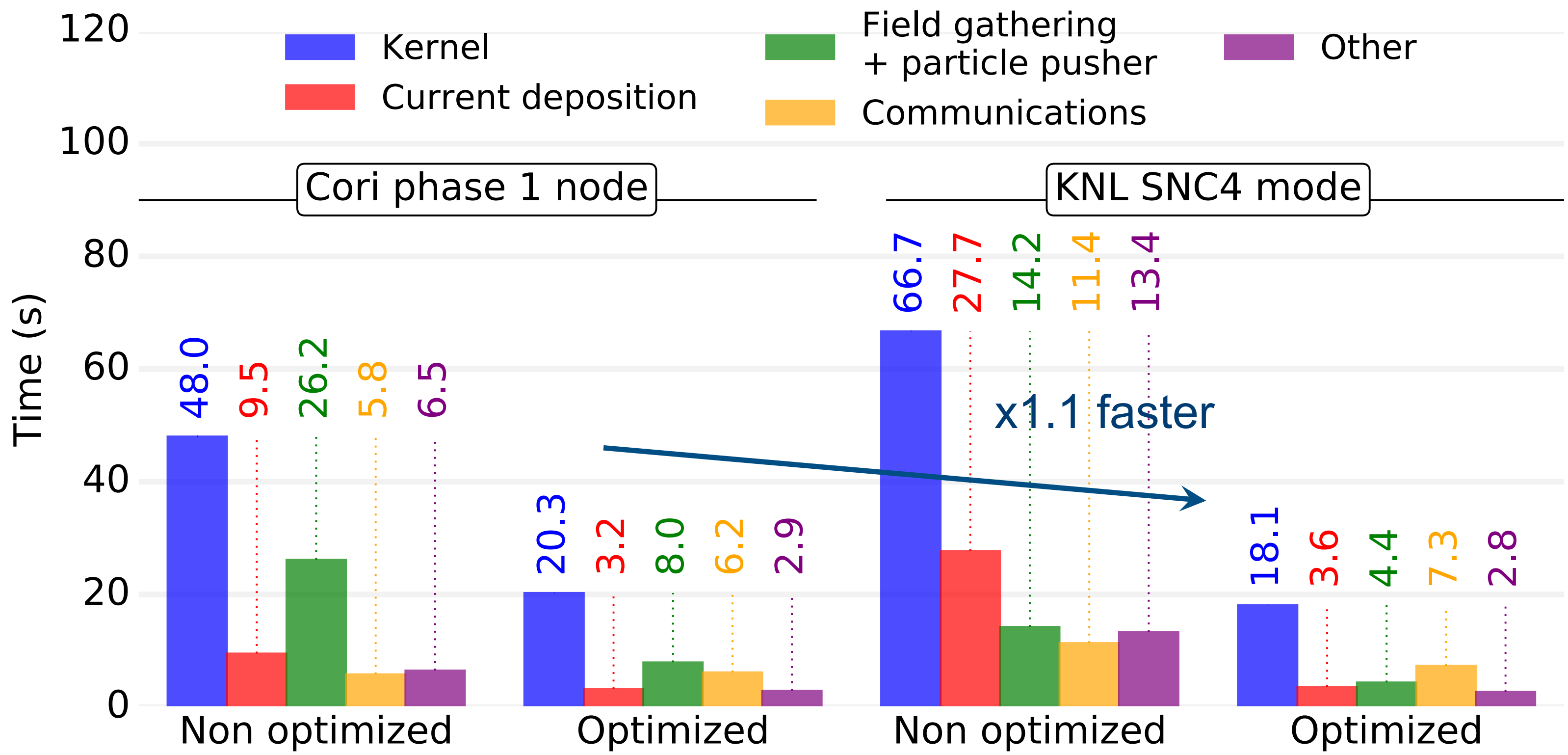
- Without optimization: simulation time 1.4 longer on KNL

Performance overview on Haswells and KNL for order 1 interpolation method



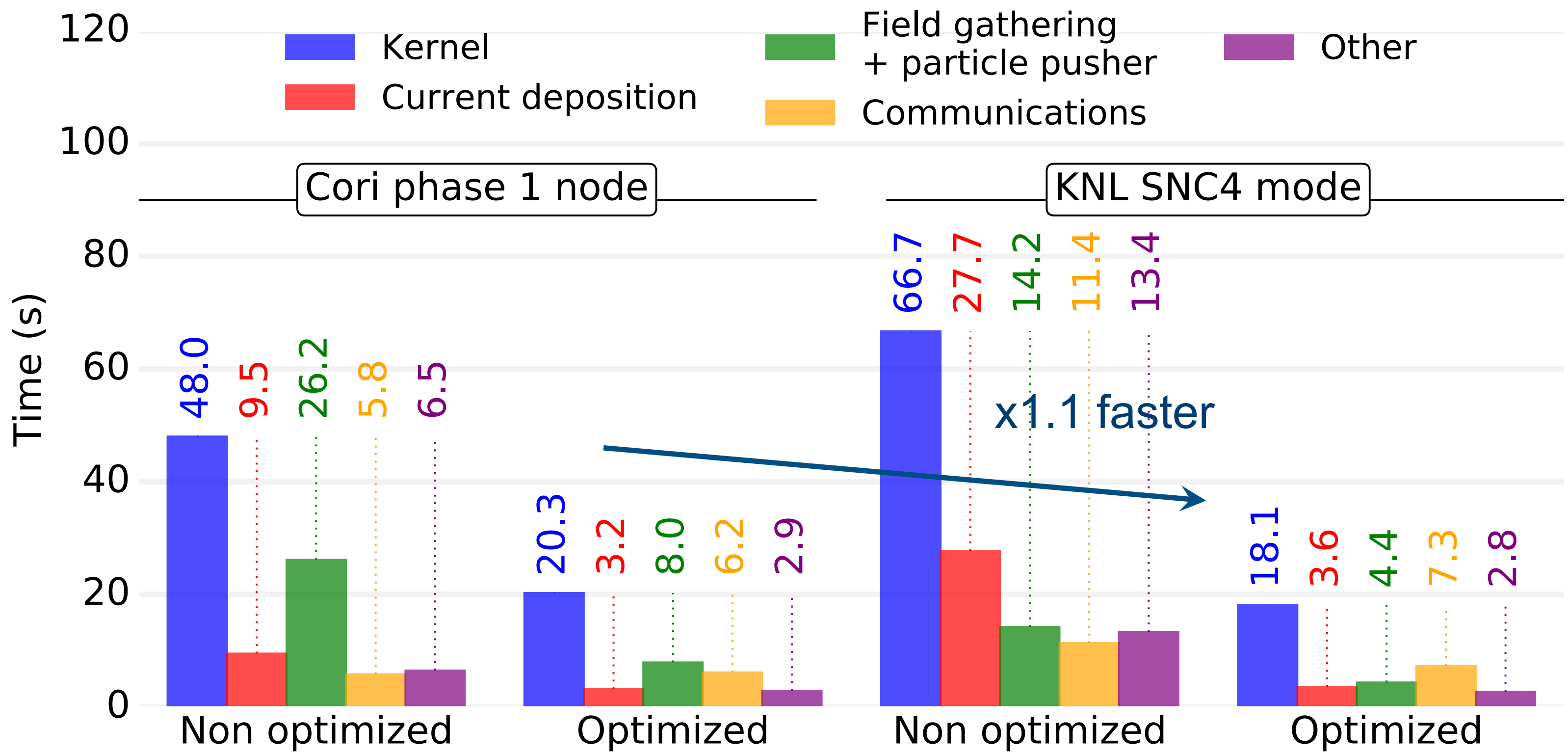
- Without optimization: simulation time 1.4 longer on KNL
- With optimizations: x2.4 speedup on Haswell and x3.7 on KNL

Performance overview on Haswells and KNL for order 1 interpolation method



- Without optimization: simulation time 1.4 longer on KNL
- With optimizations: x2.4 speedup on Haswell and x3.7 on KNL
- With optimizations: simulation time x1.1 faster on KNL at order 1 versus Haswell

Performance overview on Haswells and KNL for order 1 interpolation method



- Without optimization: simulation time 1.4 longer on KNL
- With optimizations: x2.4 speedup on Haswell and x3.7 on KNL
- With optimizations: simulation time x1.1 faster on KNL at order 1 versus Haswell

➢ Implemented optimizations essential on KNL to reach Haswell performance

➢ Implemented optimizations also speedup previous architectures (Haswell and Ivybridge)

Thank You