第20回NEXT研究会プログラム
2015年1月13,14日
京都テルサ　東館3階D会議室

Intel MIC とGPUでの粒子コードの高速化
Acceleration of PIC Code by Intel MIC and GPU

H. Naitou　Yamaguchi University

Acknowlegements:

V.K. Decyk	UCLA
M. Yagi, Y. Kagei	JAEA
Y. Tauchi	Yamaguchi University
S. Mitani, H. Morio	Yamaguchi University (Students)

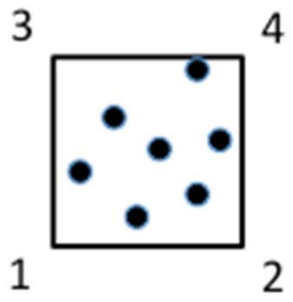
Outline

- Introduction
- Acceleration of PIC code by GPU
- Acceleration of PIC code by Intel MIC coprocessor
- Conclusions

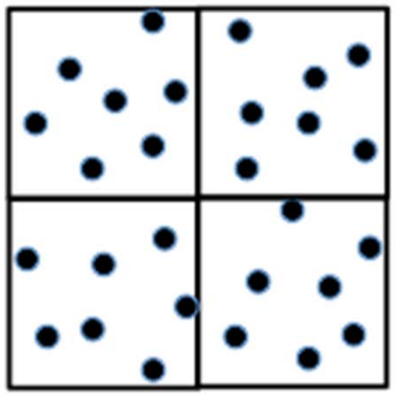
PICコードからメモリーへのランダムアクセスを消去
　　Eliminate random access to memory from PIC code.
極めて細かい粒度（セル or タイル）での独立性
　　fine grain parallelism (cell or tile)

= streaming algorithm

Cell or Tile



cell



tile

mx = 2
my = 2

Gyrokinetic PIC Code for Kinetic MHD Simulation

Parallelization on GpicMHD code on
Plasma Simulator and Helios

H. Naitou et al., J. Plasma and Fusion Res. SERIES 8, 1158 (2009).

H. Naitou et al., Plasma and Fusin Res. 6, 2401084 (2011).

H. Naitou et al, Progress in Nuclear Science and Technology 2, 657 (2011).

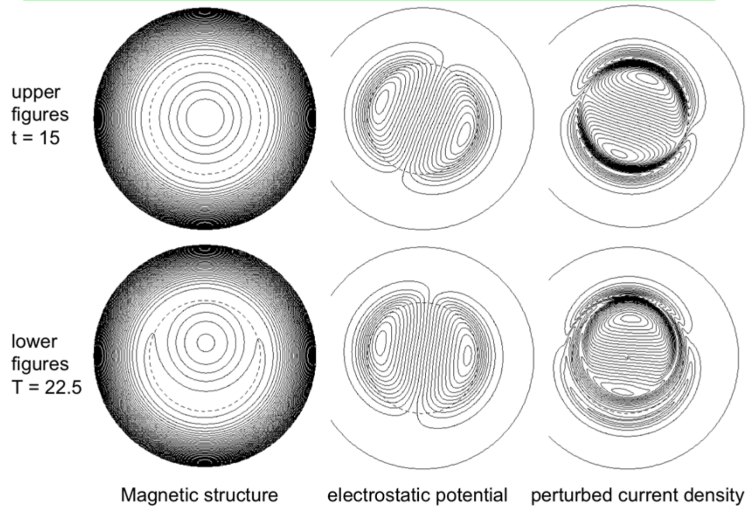
Lecture of parallelization of PIC code

"Methods of Fusion Plasma Simulation (in Japanese)
– Utilizing Massively-Parallel Computation –
5. Coding Techniques of Particle Simulations",
H. Naitou, S. Satake, J. Plasma Fusion Res. 89, 245 (2013).

Proposal of Advanced Algorithm

H. Naitou et al., Plasma Science and Technology 13, 528 (2011).

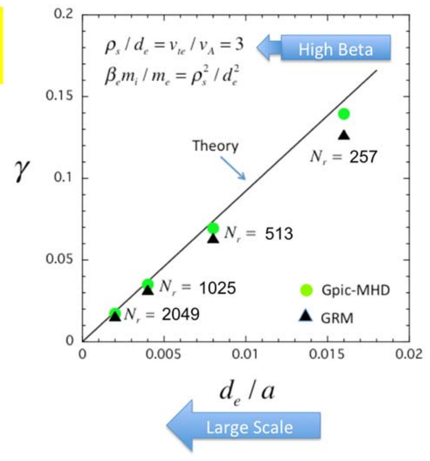
Mode profiles



Growthrate versus collisionless electron skin depth

SR16000: "plasma simulator" at NIFS
scalar SMP cluster system, 128 nodes × 64 logical cores

advanced version of Gpic-MHD



$A/a = 18$

$N_\theta = 64$

256 – 512 processes

16 threads: SMP=16

1000000 - 4000000 particles/process

High Performance Computing of PIC Code

- Massive-Parallel Computer

  Thread Parallel ------ OpenMP

  shared memory autoparallelization

  Process Parallel ------ MPI (domain/particle decomposition)

  didtributed memory

- Accelerators

  GPU (Graphics Processing Unit)

  ------ GPGPU (General-Purpose computing on GPUs)

  SIMD (Single Instruction Multiple Data)

  single-precision (32-bit) +

  double-precision (64-bit)(slow)

  intel MIC coprocessor

  double-precision

Plasma PIC (Particle-In-Cell) Simulation

- PIC code

  Particles move freely in the system.

  Fields are calculated only at grid points.

  Particles interact with nearest grids.

- Gyrokinetic-PIC code

  Based on gyrokinetic theory.

  Keep the basic algorithm of PIC code.

To use GPU as accelerator

- Conventional GPU was developed for computer graphics
- GPGPU (General-Purpose GPU)

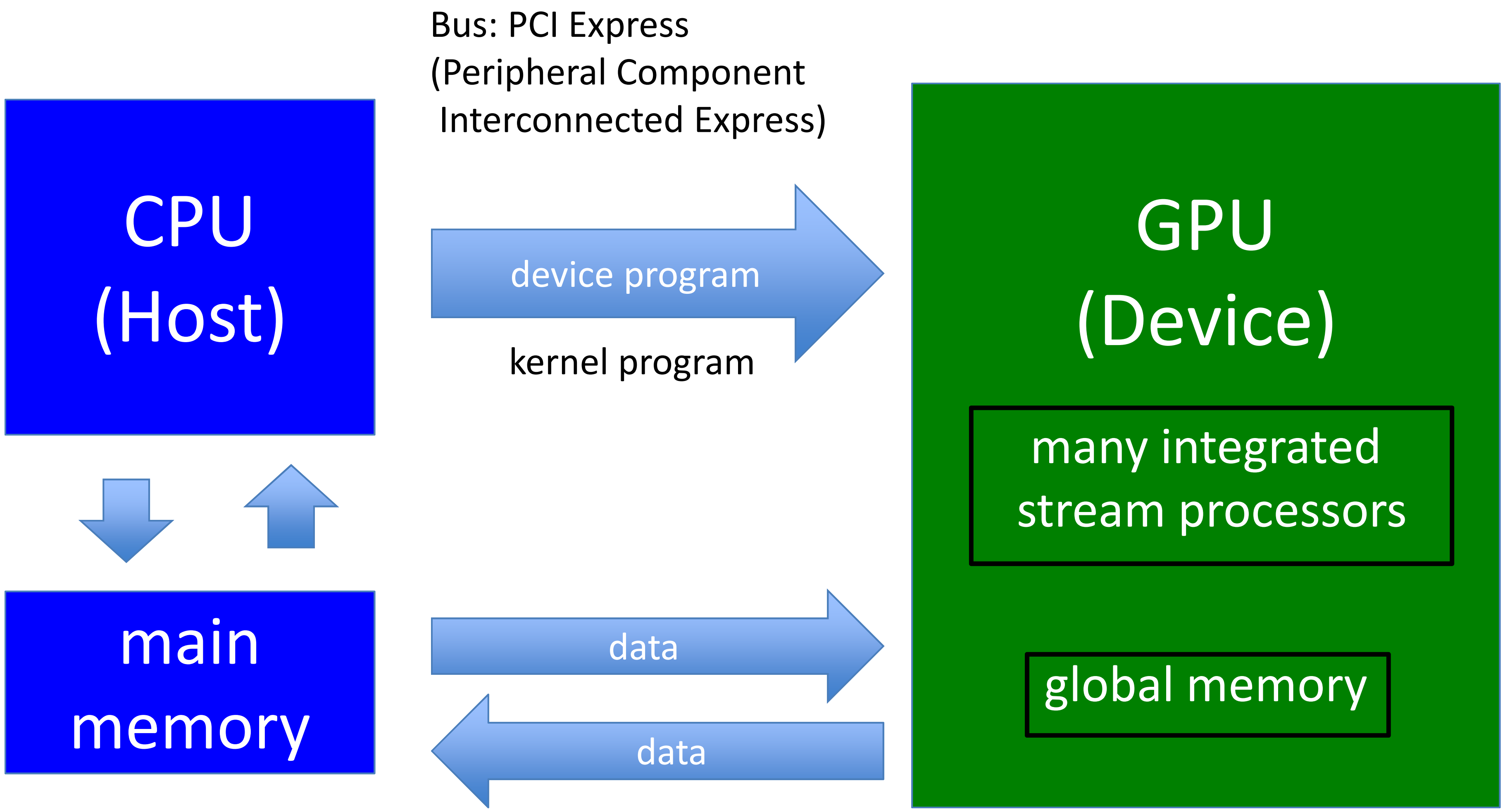
  specialized for high-performance computing

  several thousands of cores

  SIMD instructions

  NVIDIA, AMD

CPU-GPU System



GPU Programing

- CUDA (Compute Unified Device Architecture)
  - GPGPU language for NVIDIA GPUs
  - Tesla, Quadro, GeForce
  - C/C++ FORTRAN
- APP (AMD Accelerated Parallel Processing)
  - AMD (Advanced Micro Devices)
- OpenCL (Open Computing Language)
  - open framework for environments across heterogeneous platforms
    - CPU, GPU, DSP (Digital Signal Processors) etc.
  - Khronos Group (non-profit technology consortium)
    - C/C++
- OpenACC (will merge into OpenMP)
  - programing standard for CPU/GPU systems
    - C/C++ FORTRAN

An Example of GPU Acceleration of a PIC Code

- 2D electrostatic PIC code.
- Single floating-point precision.
- Follow only electron dynamics.
- Linear interpolations for charge assignment and particle acceleration.
- No external magnetic field.

Particle Parallel Method

One thread treats one particle.

PUSH: Particle pushing ○

SOURCE: Charge Assignment ×

FFT ○ ----- CUDAFFT

It is very easy to modify the PIC code to CPU-GPU system.

Cell Parallel Method

One thread treats one cell and particles inside the cell.

PUSH: Particle pushing ○

SOURCE: Charge Assignment ○

FFT ○ ------ CUDAFFT

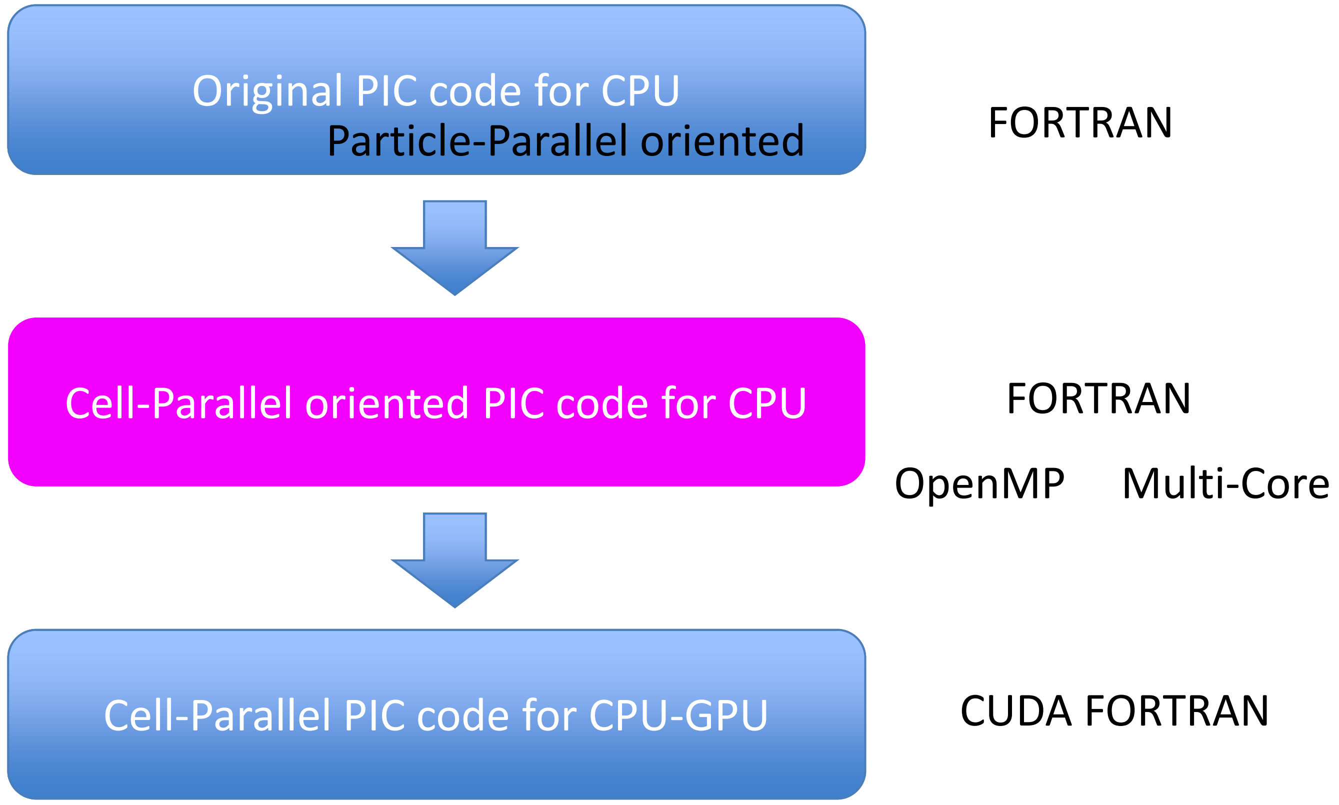
Additional computation is needed:

SORT: after every pushing, particles must move to proper cells.

Ref.1 V. K. Decyk, T. V. Singh,
"Adaptable Particle-in-Cell algorithms for graphical processing units",
Computer Physics Communications 182 (2011) 641-648.

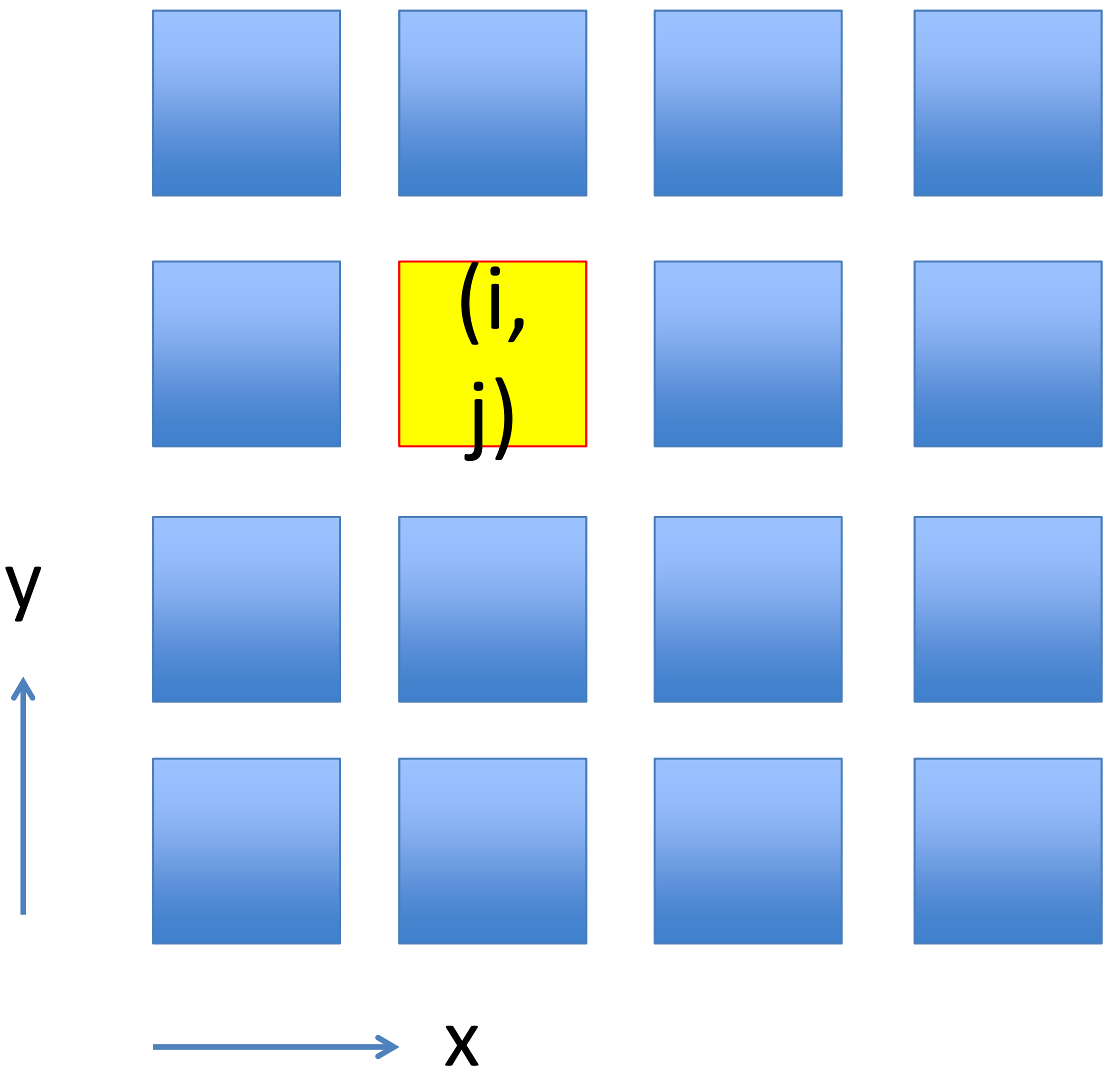
Ref.2 V. K. Decyk, T. V. Singh
"Particle-in-Cell algorithms for emerging computer architectures"
Computer Physics Communications 185 (2014) 708-719.

How to easily modify PIC code for GPU.

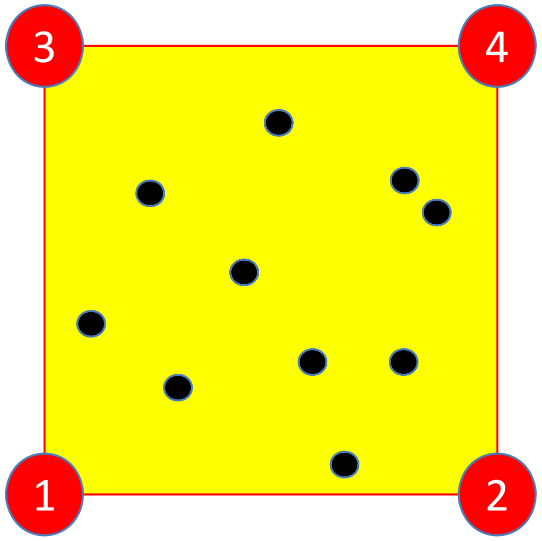

Original PIC code for CPU
Particle-Parallel oriented
FORTRAN
Cell-Parallel oriented PIC code for CPU
FORTRAN
OpenMP
Multi-Core
Cell-Parallel PIC code for CPU-GPU
CUDA FORTRAN

Modify each subroutine to DEVICE PROGRAM one by one.

Basic Idea of Cell-Parallel Method (1)



Basic Idea of Cell-Parallel Method (2)



parcount = 10

STRUCT
- parcount:
  - number of particles in a cell
- meshpxy(4,200):
  - keep particle data(x, y, vx, vy)
- efx(4):
  - electric field in x
- efy(4):
  - electric field in y
- rho(4):
  - keep assigned charge

PUSH: particle acceleration

Local Ex: efx(4), efy(4)

↑

Global Ex: efx_global(meshx, meshy)
efy_global(meshx, meshy)

←

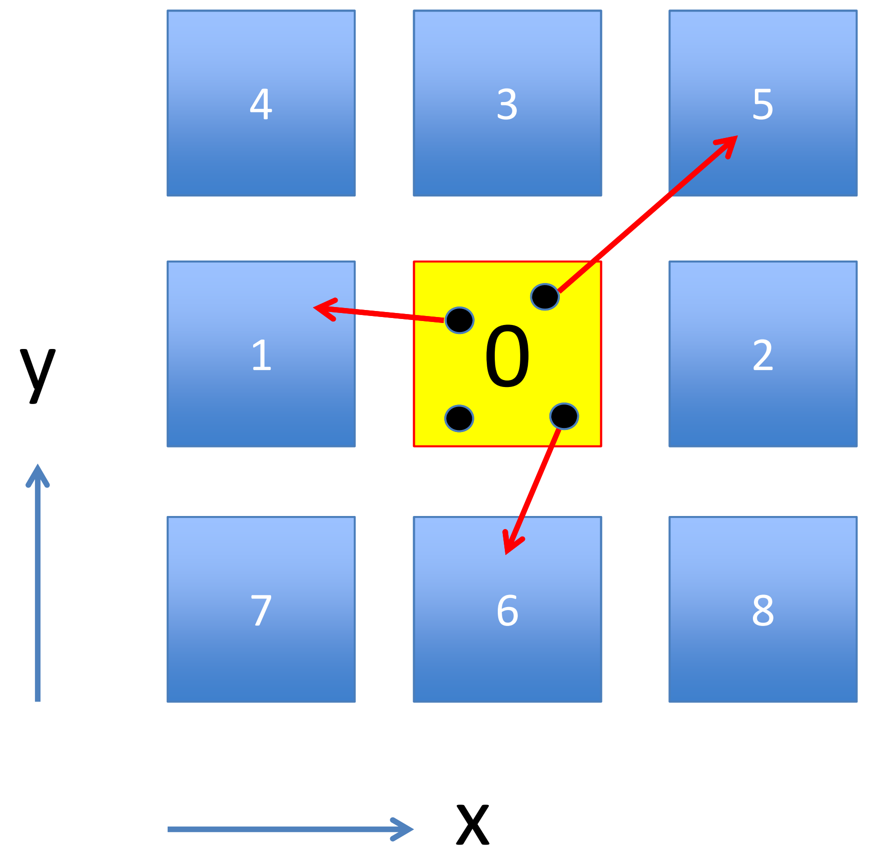
SOURCE: charge assignment

Local rho: rho(4)

↓

Global rho: rho_global(meshx,meshy)

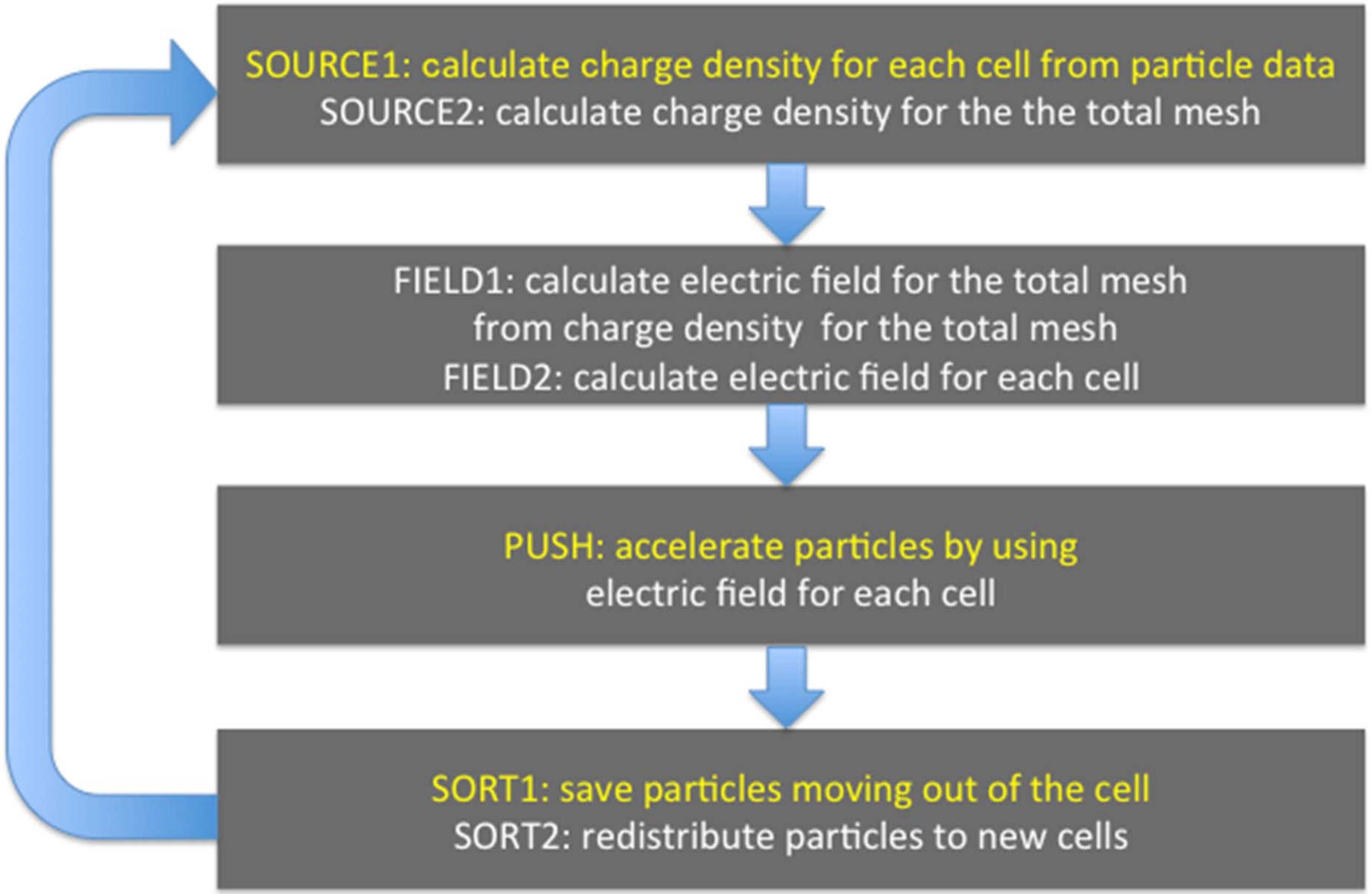
SORT ・・・・ reordering



After particle pushing, each particle will move to 8 adjacent cells or stay In the original cell.

1st step: store particle data for 8 different orientations
2nd step: move particles to the new mesh

1 step for Cell-Parallel PIC Code



SOURCE1: calculate charge density for each cell from particle data
SOURCE2: calculate charge density for the the total mesh

FIELD1: calculate electric field for the total mesh from charge density for the total mesh
FIELD2: calculate electric field for each cell

PUSH: accelerate particles by using electric field for each cell

SORT1: save particles moving out of the cell
SORT2: redistribute particles to new cells

Caution： join SORT1 to PUSH for eliminating multiple data access

Computing Environment

Intel Core i7-4770 3.4-3.9 GHz

GeForce GTX TITAN

2688 streaming processors

Linux Ubuntu

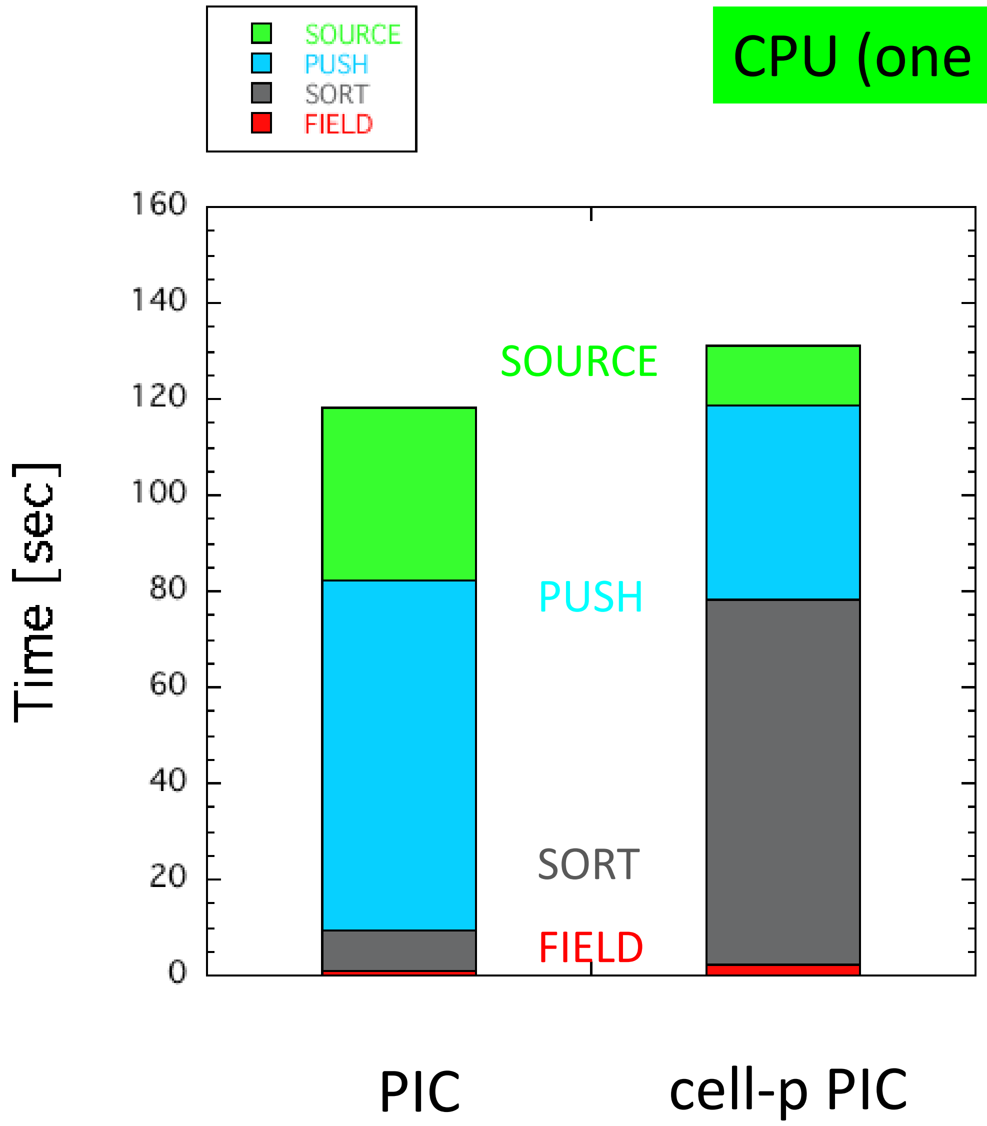
PGI CUDA Fortran

CUDA ver.   5.5

Kepler architecture

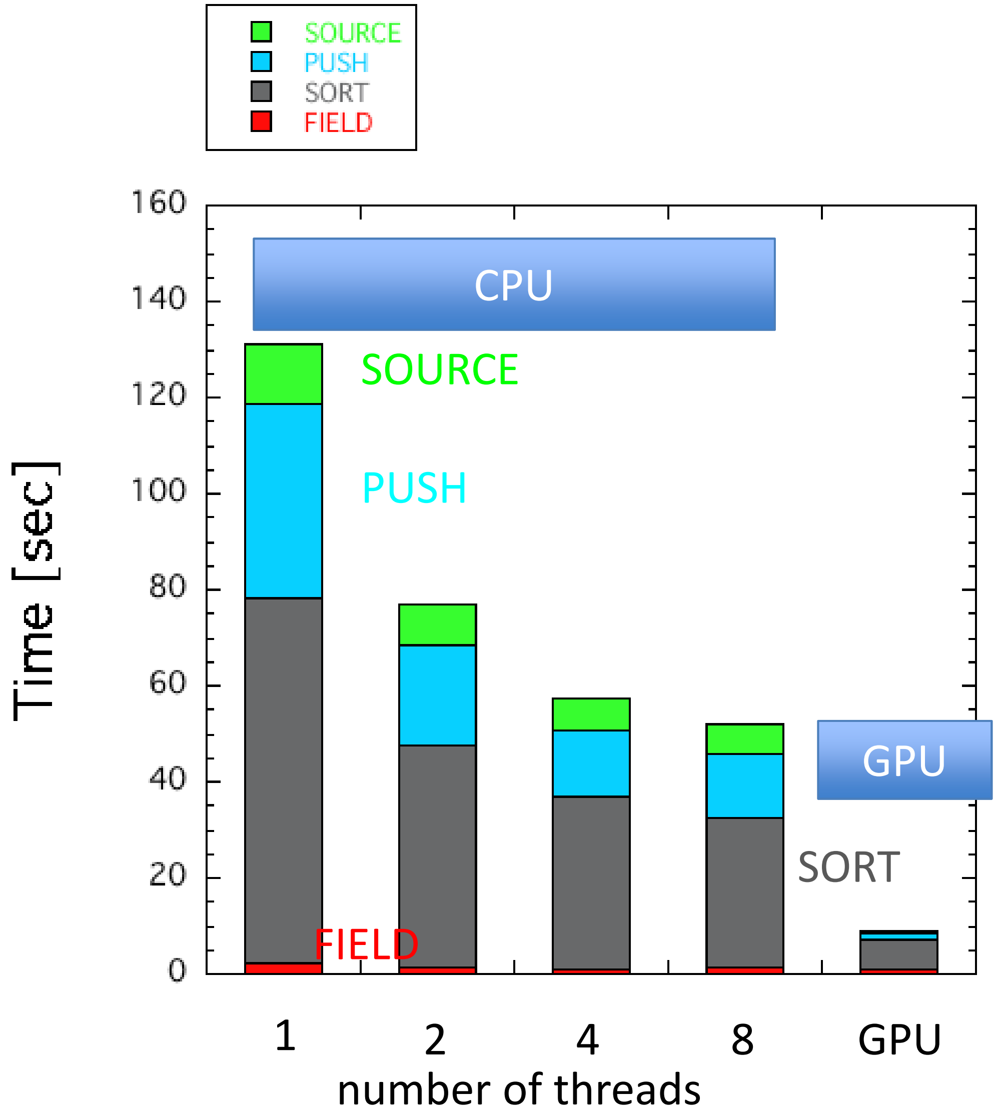
PIC vs. Cell-Parallel Oriented PIC

CPU (one core)



256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

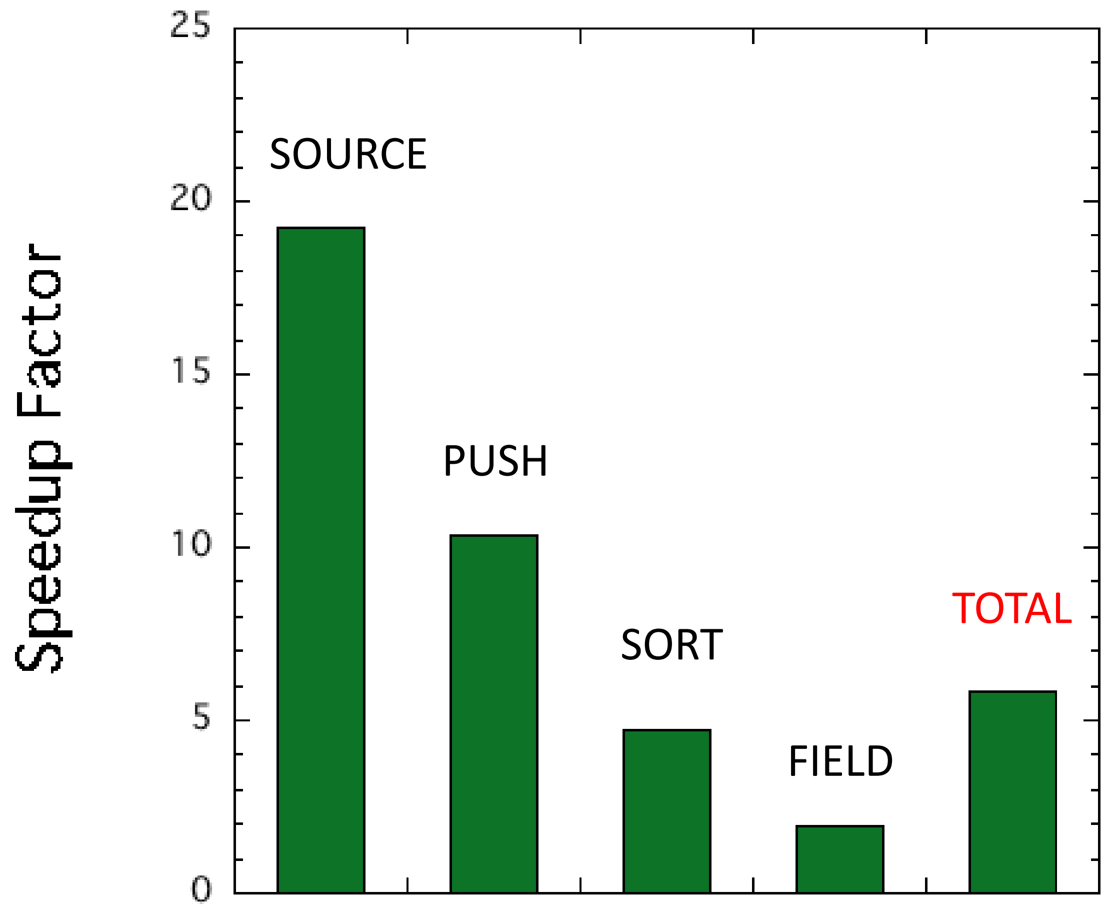
Multi-core vs. GPU



256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Speedup factor

CPU (8 threads) time / GPU time



256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Another Accelerator: Xeon Phi

Ref:Wikipedia

- Intel
- MIC (Many Integrated Core) architecture
- coprocessor

Xeon Phi 5110P

November 12, 2012

22nm 60 cores 1.053GHz

double precision 1.011 TFLOPS

TOP500 November 2013 world fastest supercomputer

Tianhe-2 Intel Ivy Bridge Xeon + Xeon Phi 33.86 PetaFLOPS

180 MIC nodes was added to Helios

One node consists of:

- Host CPU

  Xeon processor E5 2450 x 2

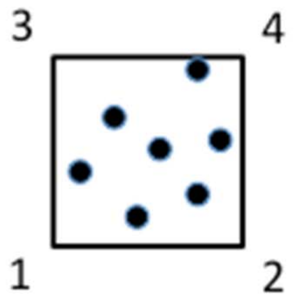
  8 cores, 24 GB
- Coprocessor

  Xeon Phi 5110P x 2

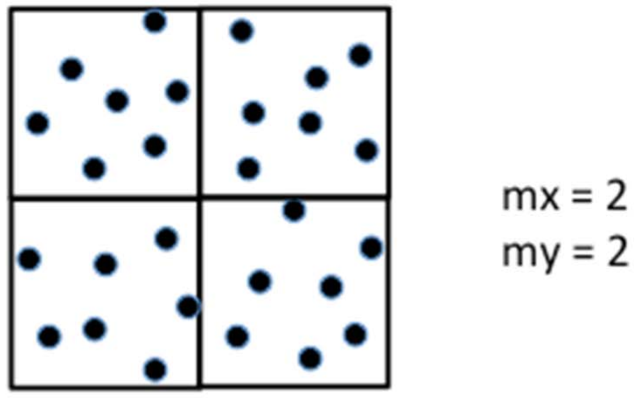
  60 cores, 8 GB

Offload execution mode	CPU -> MIC
Coprocessor native execution mode	MIC (ssh)
Symmetric execution mode	CPU+MIC (mpi)

Cell or Tile



cell



tile

How to make MIC code

OpenMP version of
Tile-Parallel oriented PIC code for CPU

FORTRAN

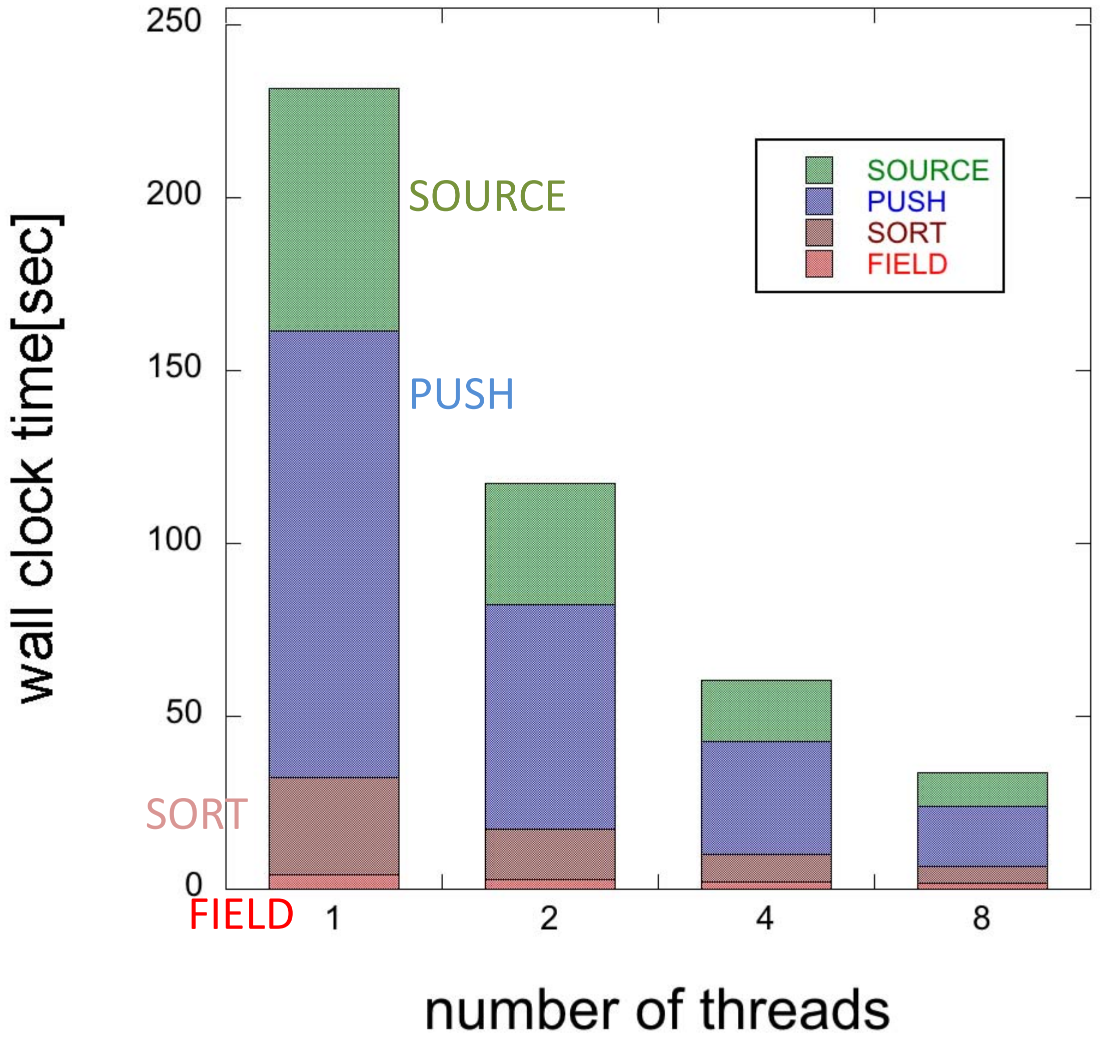
・thread parallel for multi-core

MIC version of
Tile-Parallel oriented PIC code for CPU
(native mode)

・same as OpenMP version
・No change is needed!

Scaling for Host CPU (1)

Tile-Parallel



mx = 4

my = 4

256 × 256 mesh

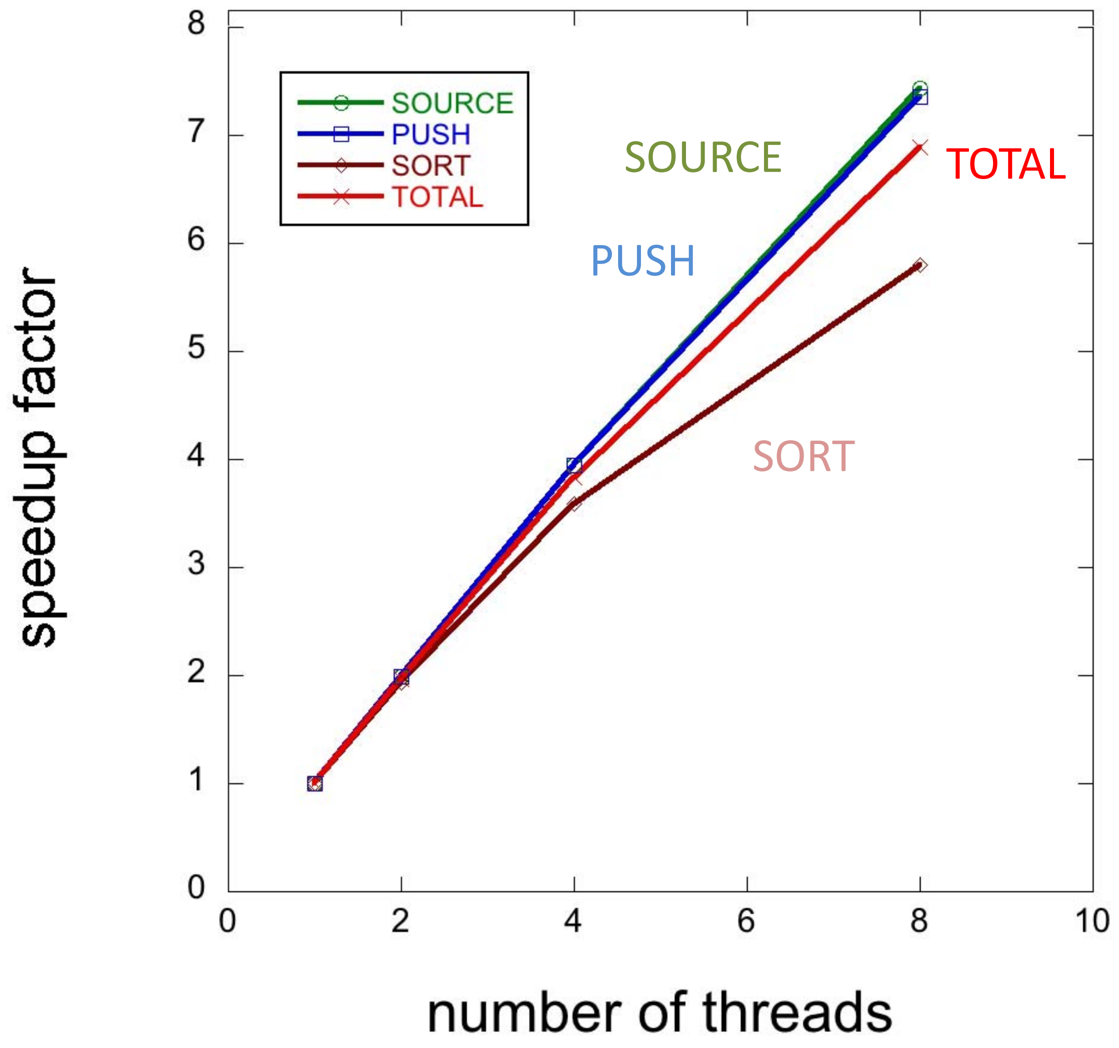
100 particles / mesh

1000 steps

Δt = 0.1

Scaling for Host CPU (2)

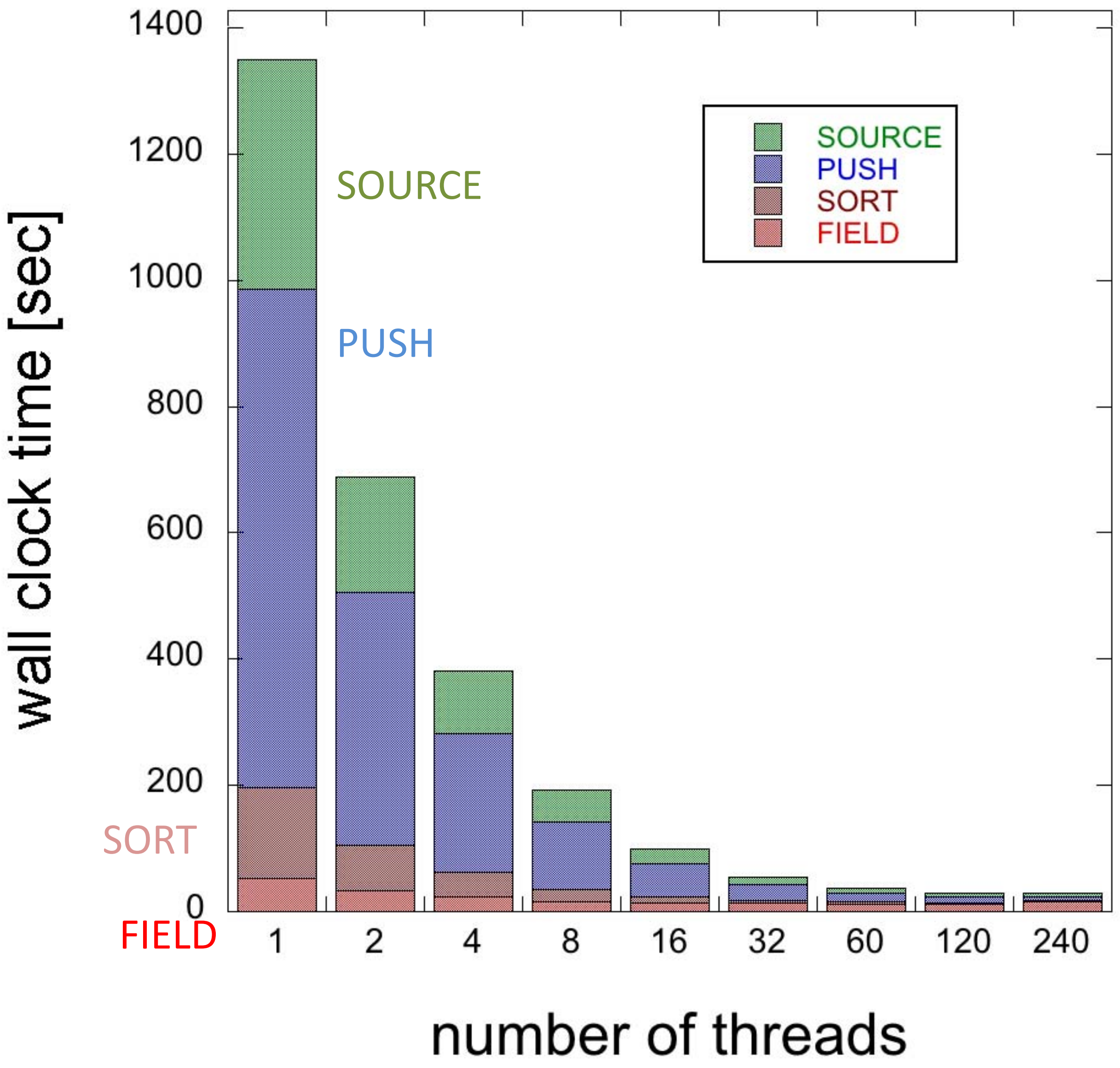
Tile-Parallel



mx = 4
my = 4

256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Scaling for Intel MIC (1)

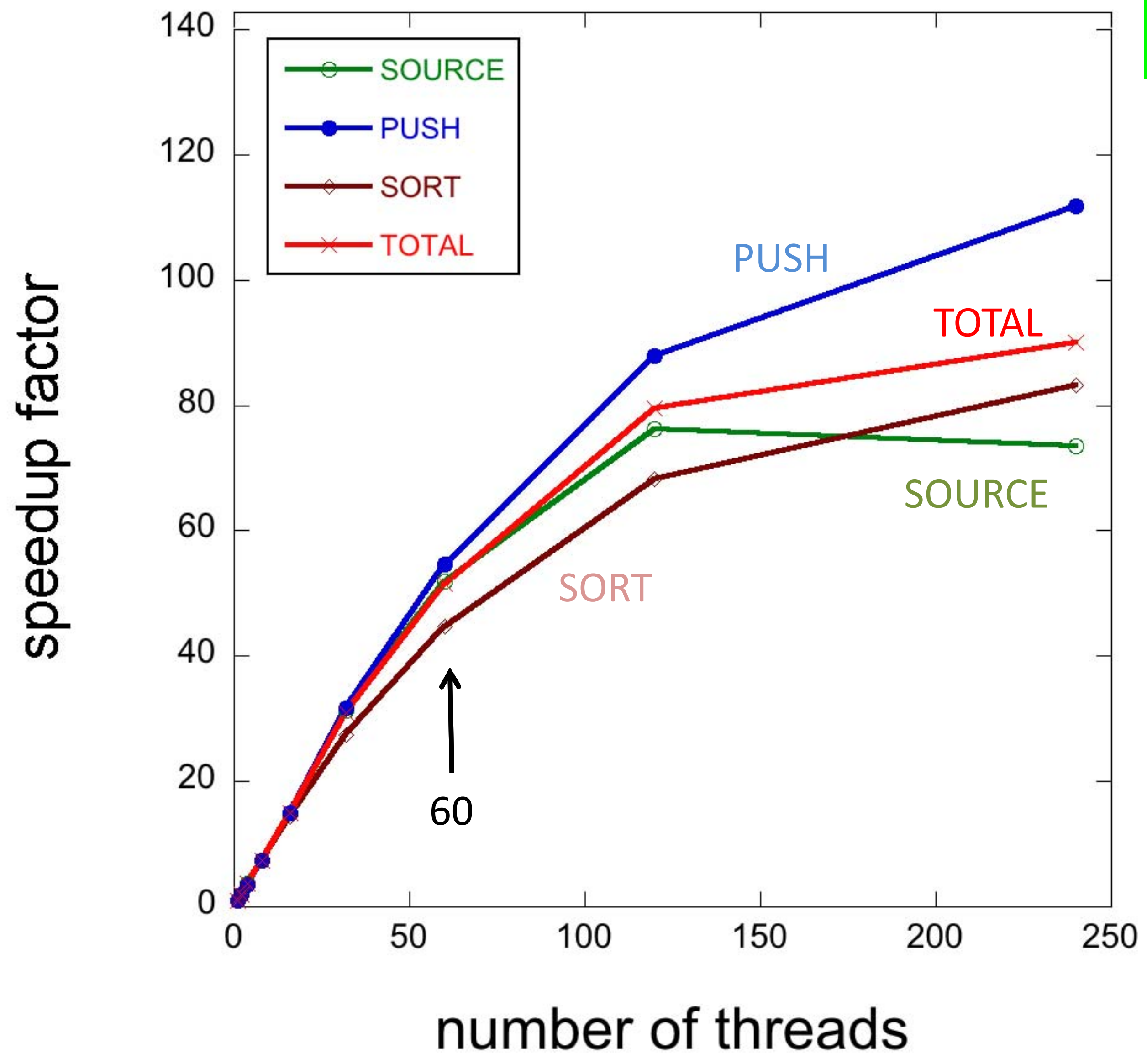


Tile-Parallel

mx = 4
my = 4

256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Scaling for Intel MIC (2)



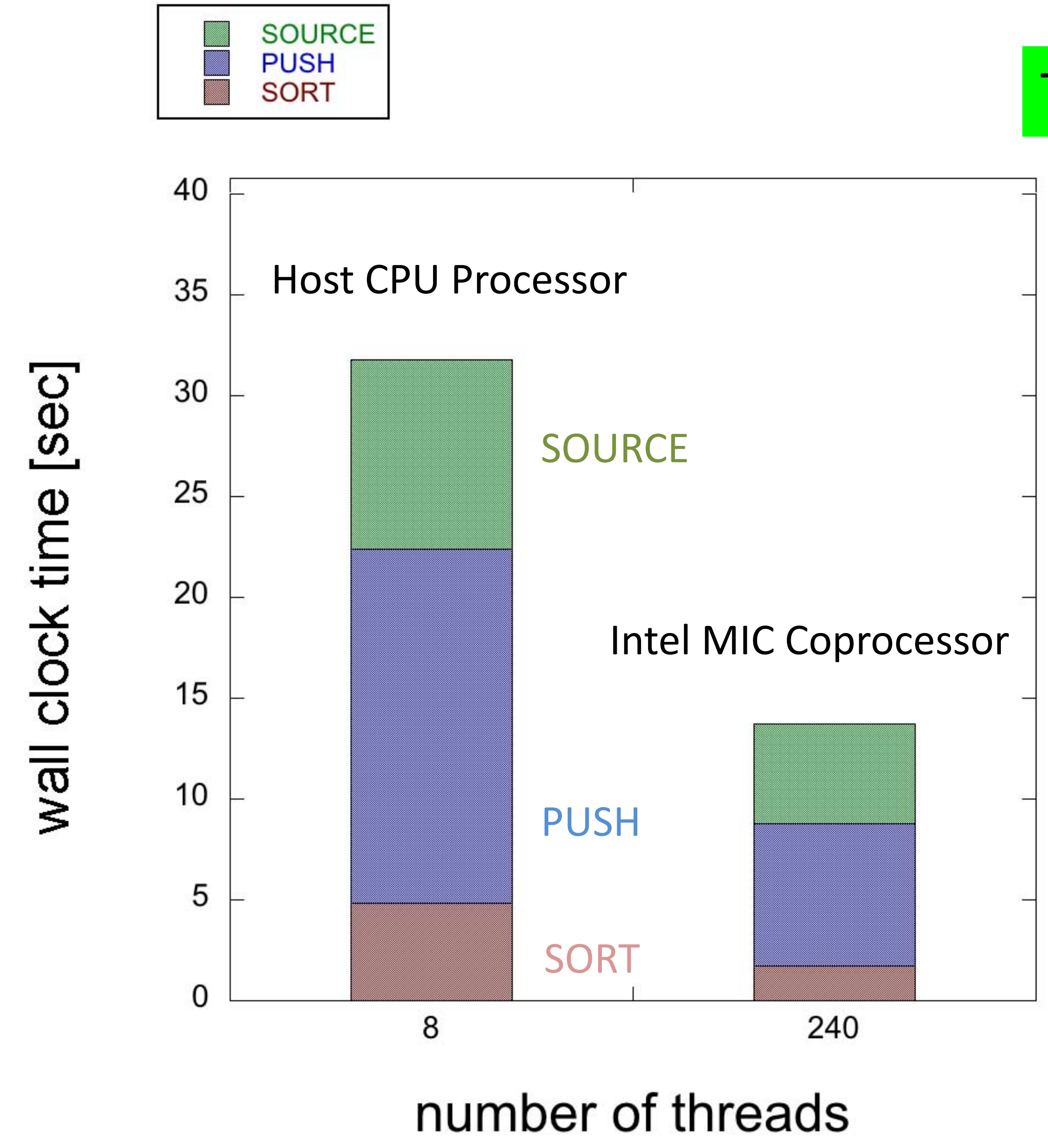
Tile-Parallel

mx = 4
my = 4

256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Host CPU vs. Intel MIC

Tile-Parallel

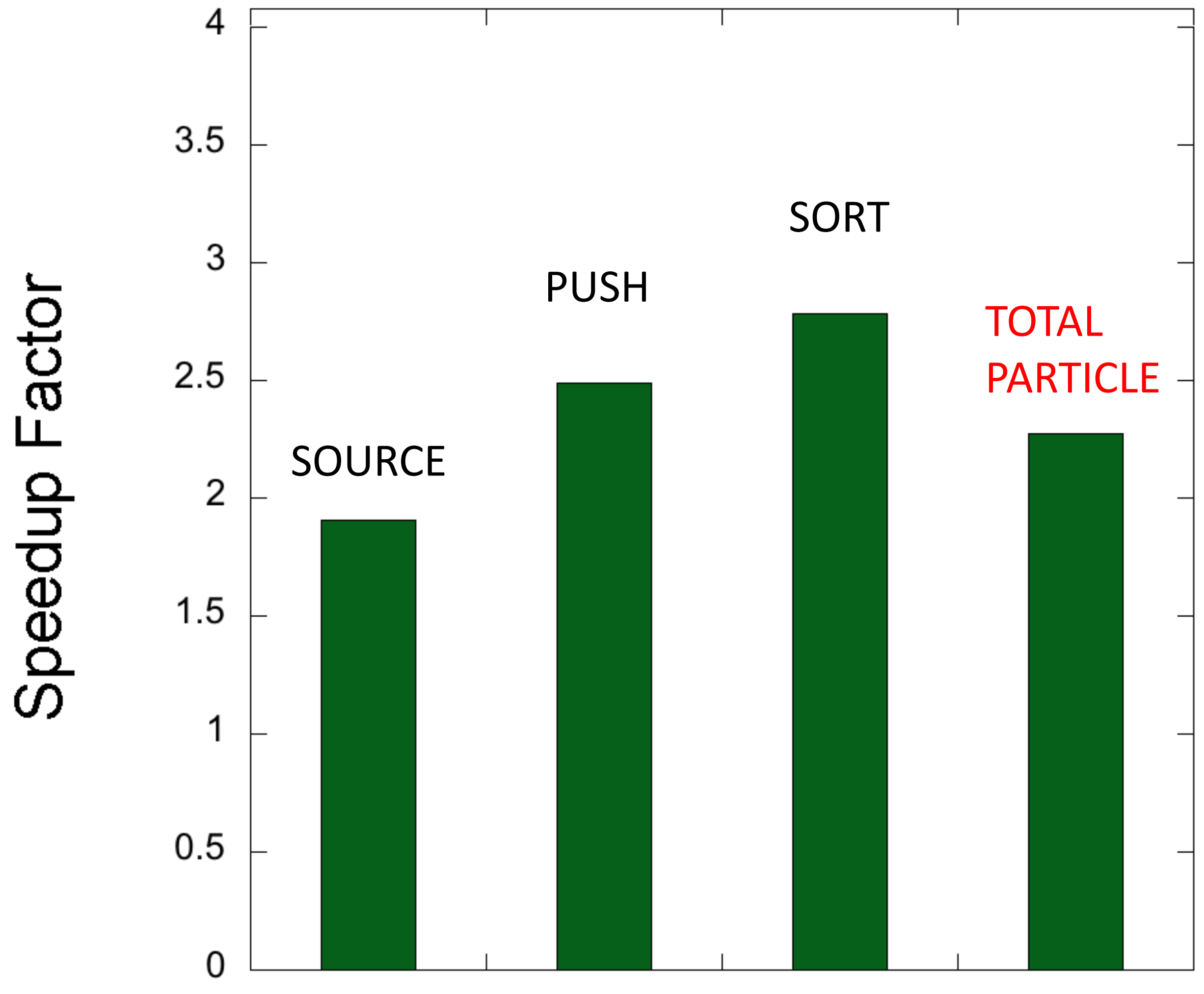


mx = 4
my = 4

256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Speedup Factor

CPU time (8 threads) / Intel MIC time (240 threads)

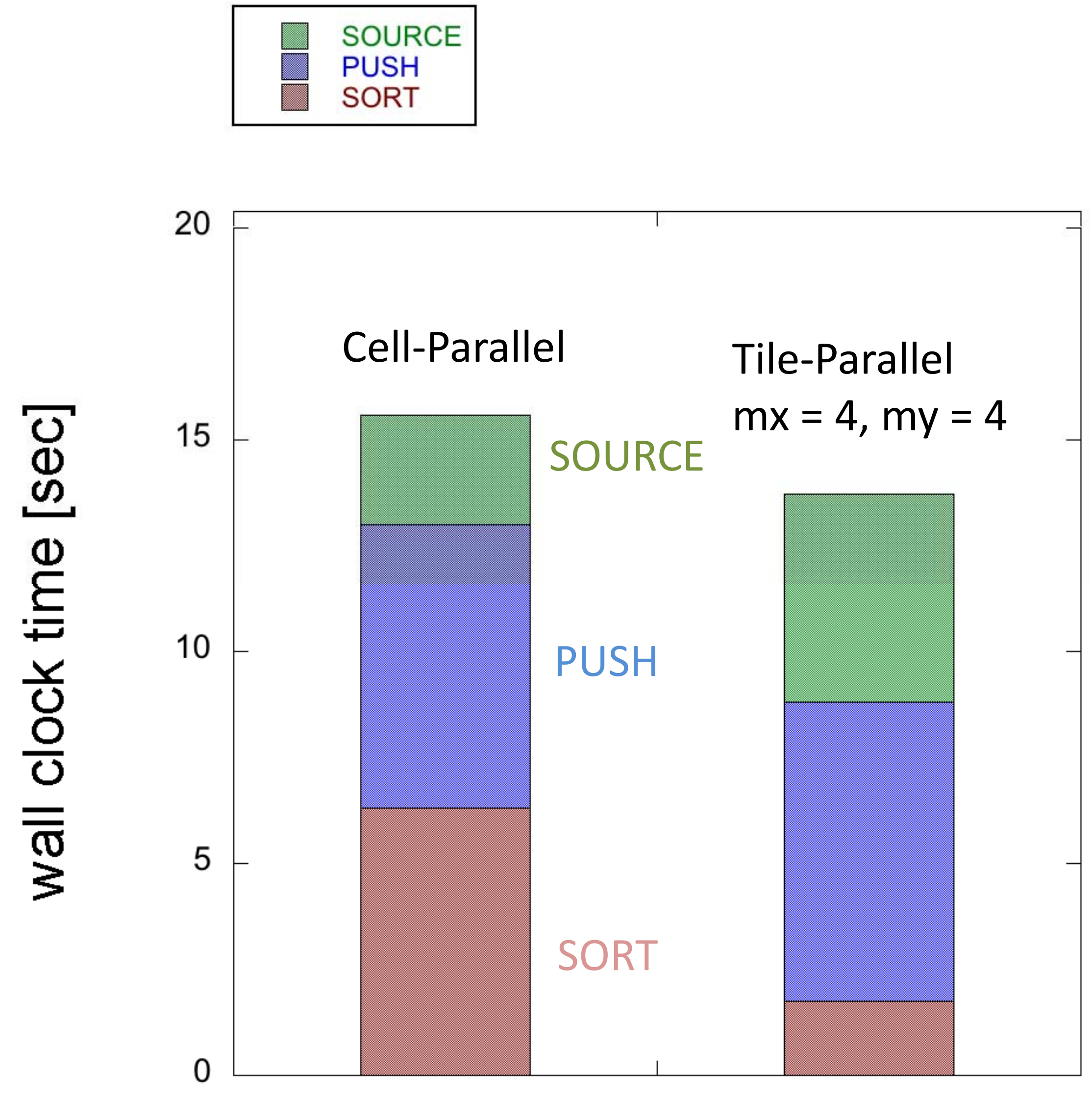


Tile-Parallel

mx = 4
my = 4

256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Cell vs. Tile



Intel MIC
240 threads

256 × 256 mesh
100 particles / mesh
1000 steps
Δt = 0.1

Conclusions

Performance of GPU

- Cell-parallel PIC code for CPU-GPU system is tested for GTX TITAN.
- Speed-up factor for SOURCE and PUSH is excellent.

  GPU is powerful for these type of algorithms.
- The total speedup factor obtained is 5.8.
- SORT is dominant for cell-parallel algorithm.
- Cell-Parallel vs. Tile-Parallel ?

Performance of Intel MIC Coprocessor

- Cell and tile-parlallel PIC code is parallelized for multi-cores by OpenMP.
- Above code is tested for intel MIC coprocessor Xeon Phi 5110P without any modification.
- Native mode (stand alone mode) is used.
- As the number of threads increases, excellent scaling is obtained.
- Speedup factor for total particle time is 2.3.
- Cell-parallel code is comparable to tile-parallel code.