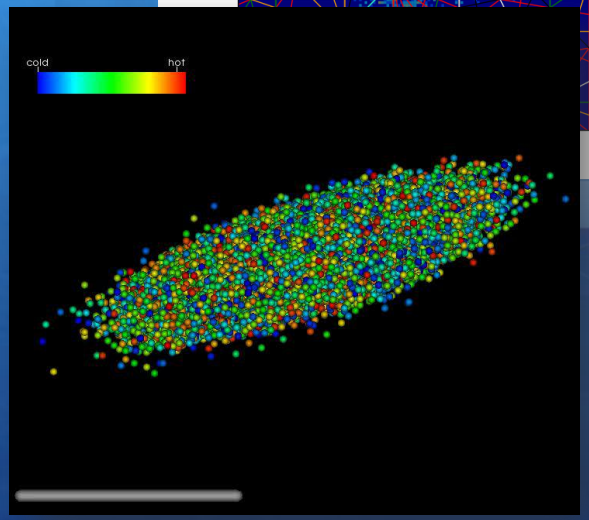
Fermilab

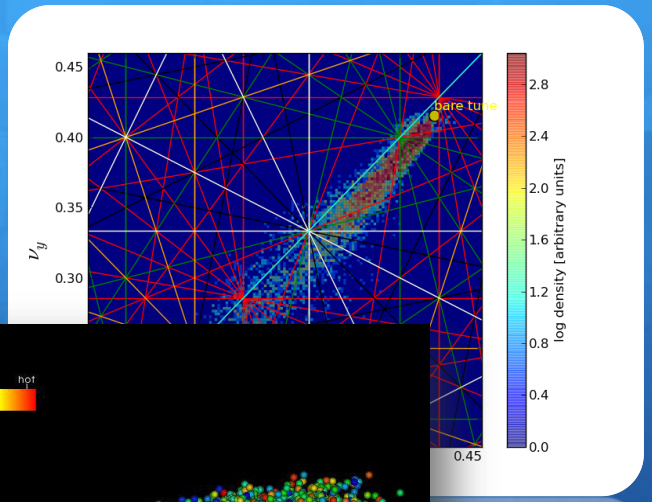
Fermi National Accelerator Laboratory
Office of Science / U.S. Department of Energy
Managed by Fermi Research Alliance, LLC

Multi-Core and GPU Optimizations for Synergia 2

Qiming Lu & Jim Amundson
Fermilab

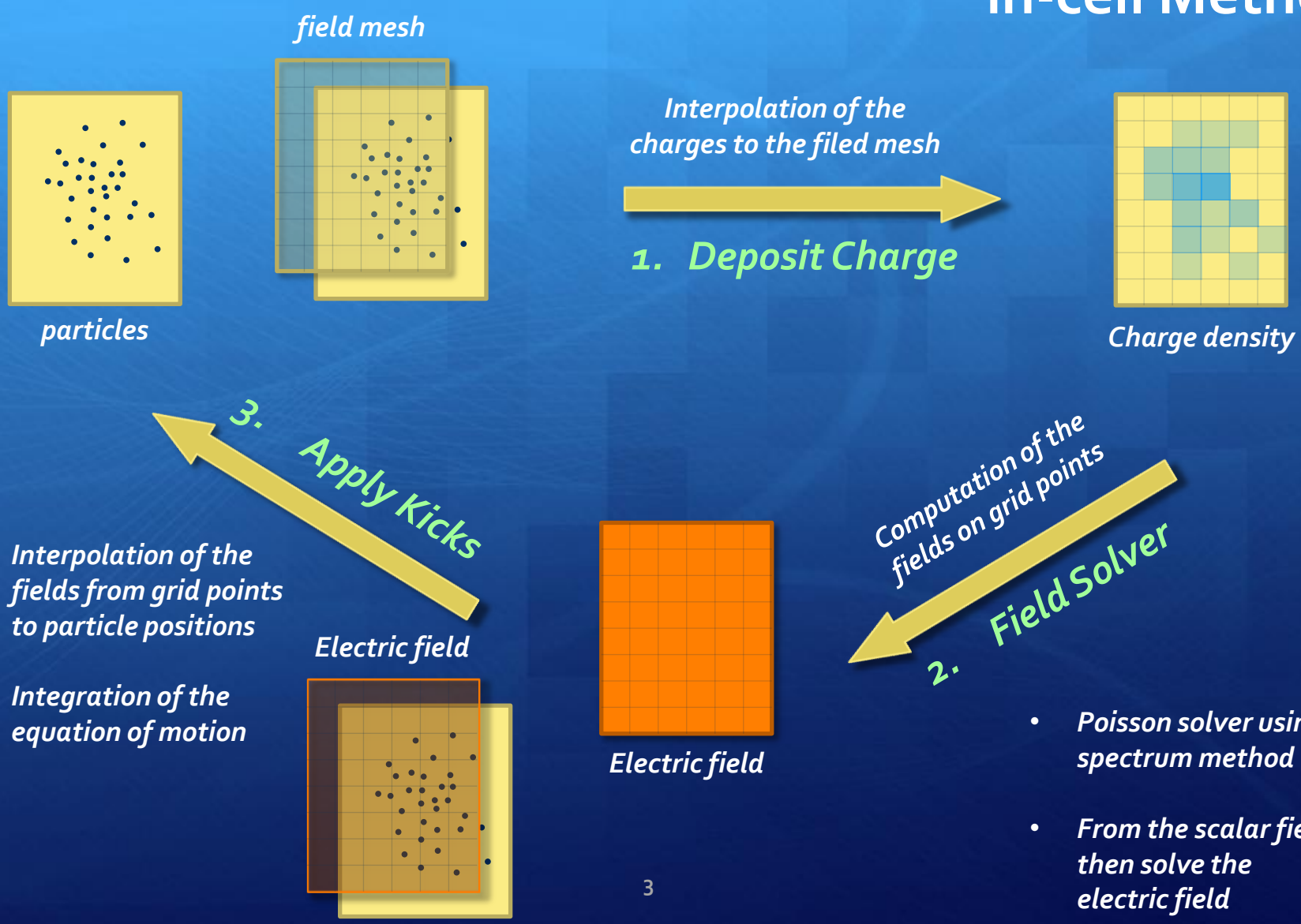
SYNERGIA





- Beam dynamics simulation
  - Single-particle dynamics
  - Multi-particle dynamics where the particle-particle interactions are important
    - Space charge / beam-beam interaction / electron cloud / wakefields
- Parallel, 3d space charge Particle-in-cell (PIC) code

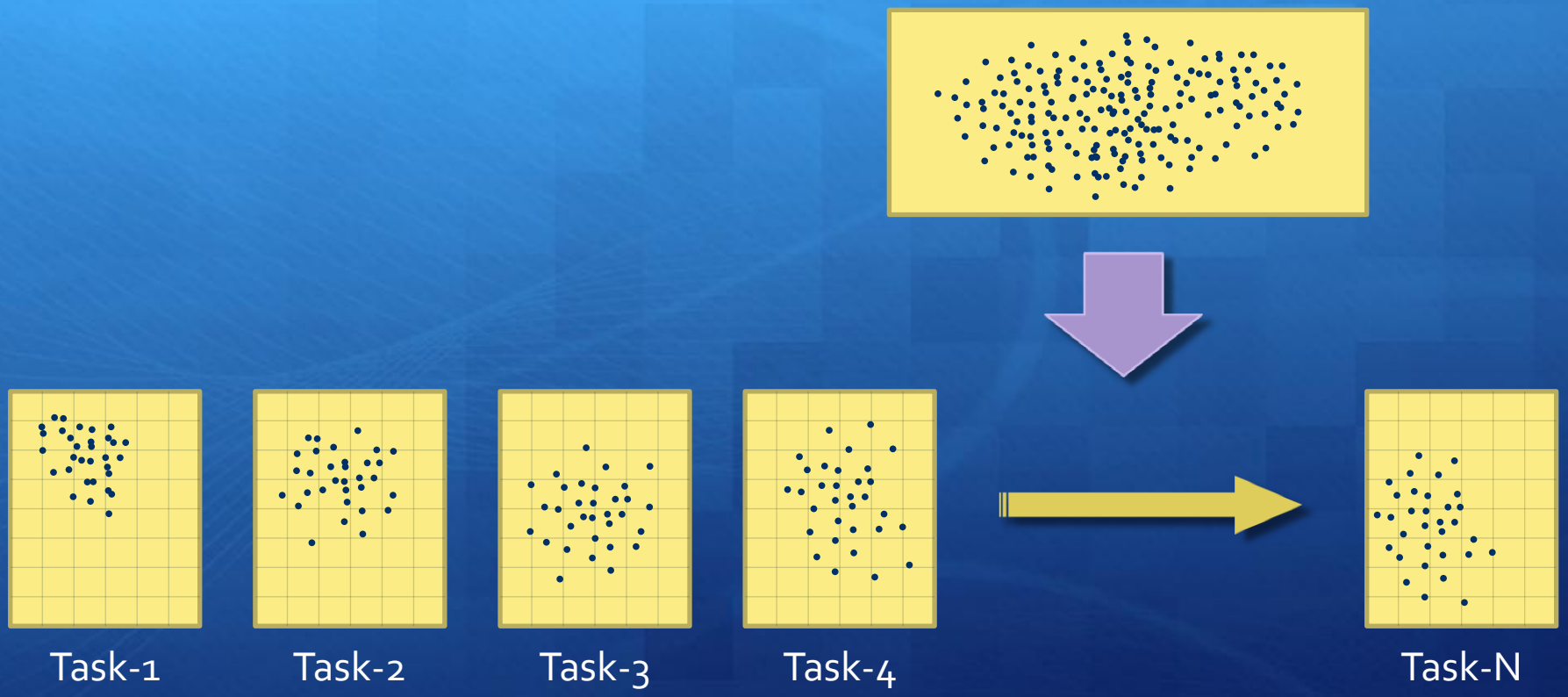
Procedures of Particle-in-cell Method



- Interpolation of the fields from grid points to particle positions
- Integration of the equation of motion

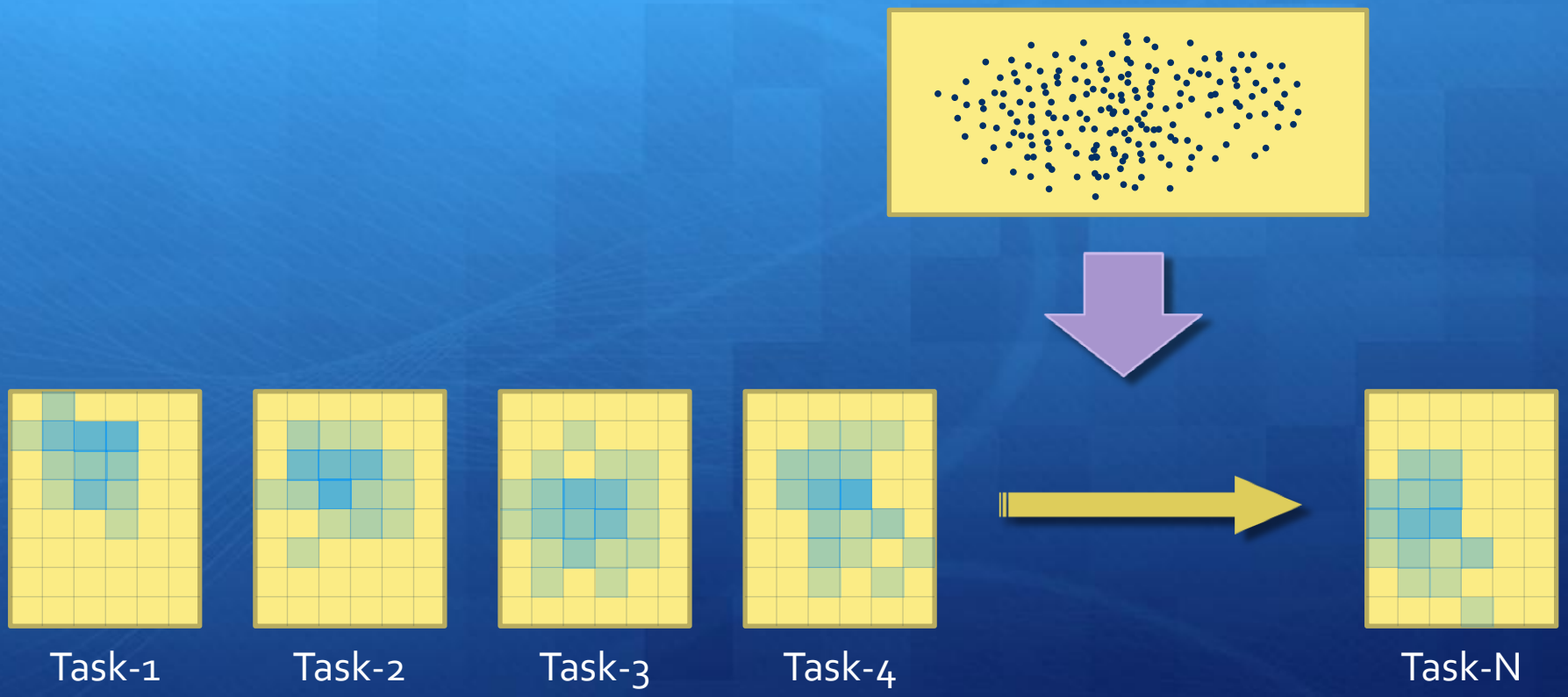
- Poisson solver using spectrum method
- From the scalar field then solve the electric field

Parallel PIC 1. charge deposition



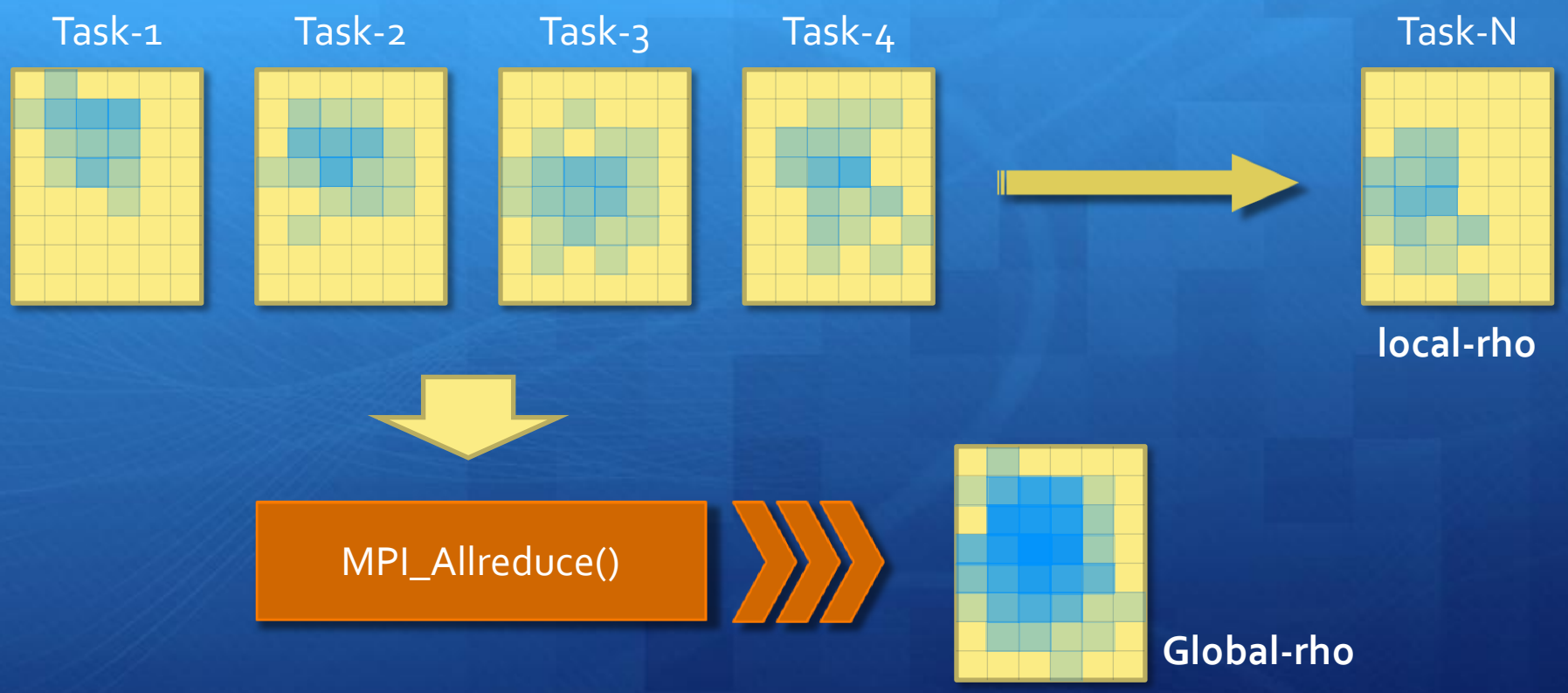
- *Randomly distribute particles into MPI tasks*
- *Each task has the entire spatial domain*

Parallel PIC 1. charge deposition



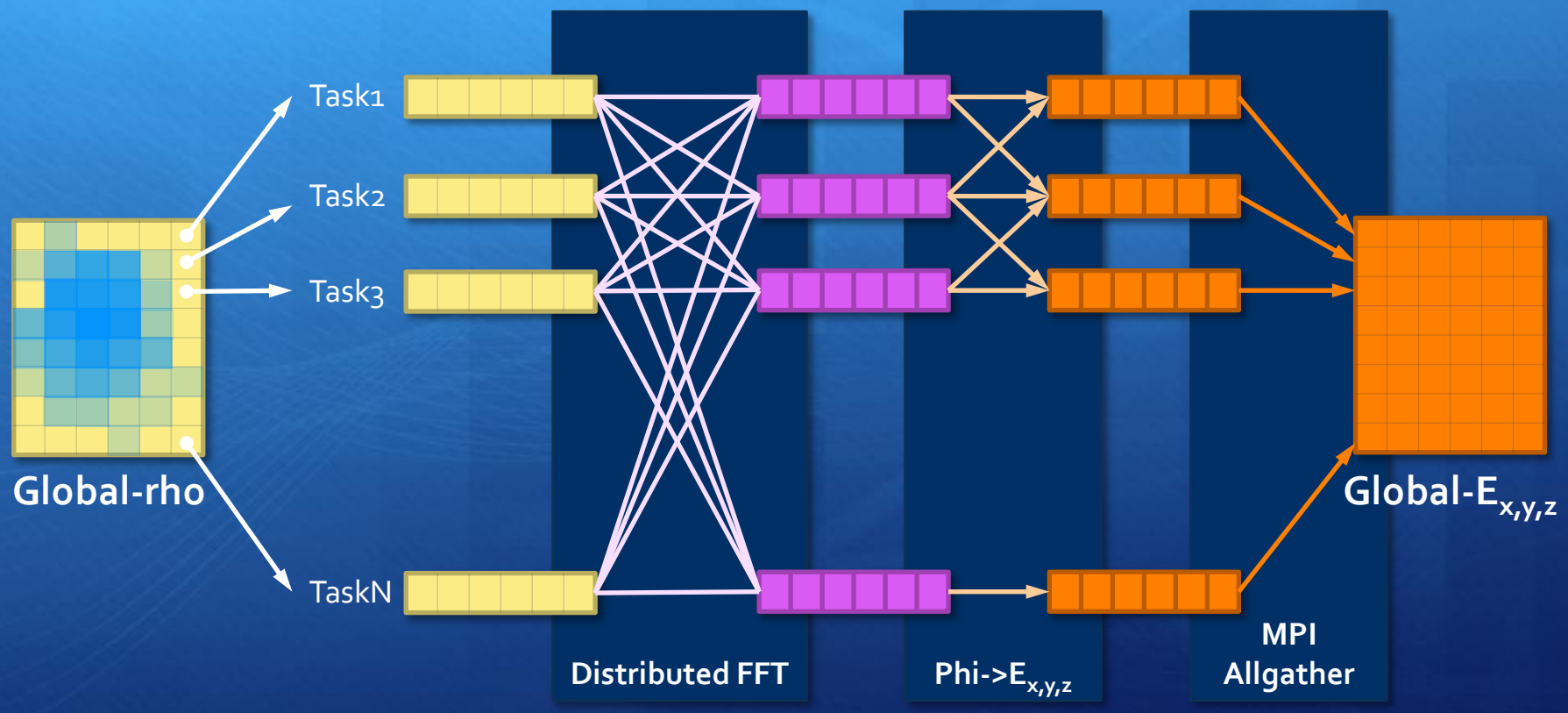
- ***Distributed charge deposition***

Parallel PIC 1. charge deposition



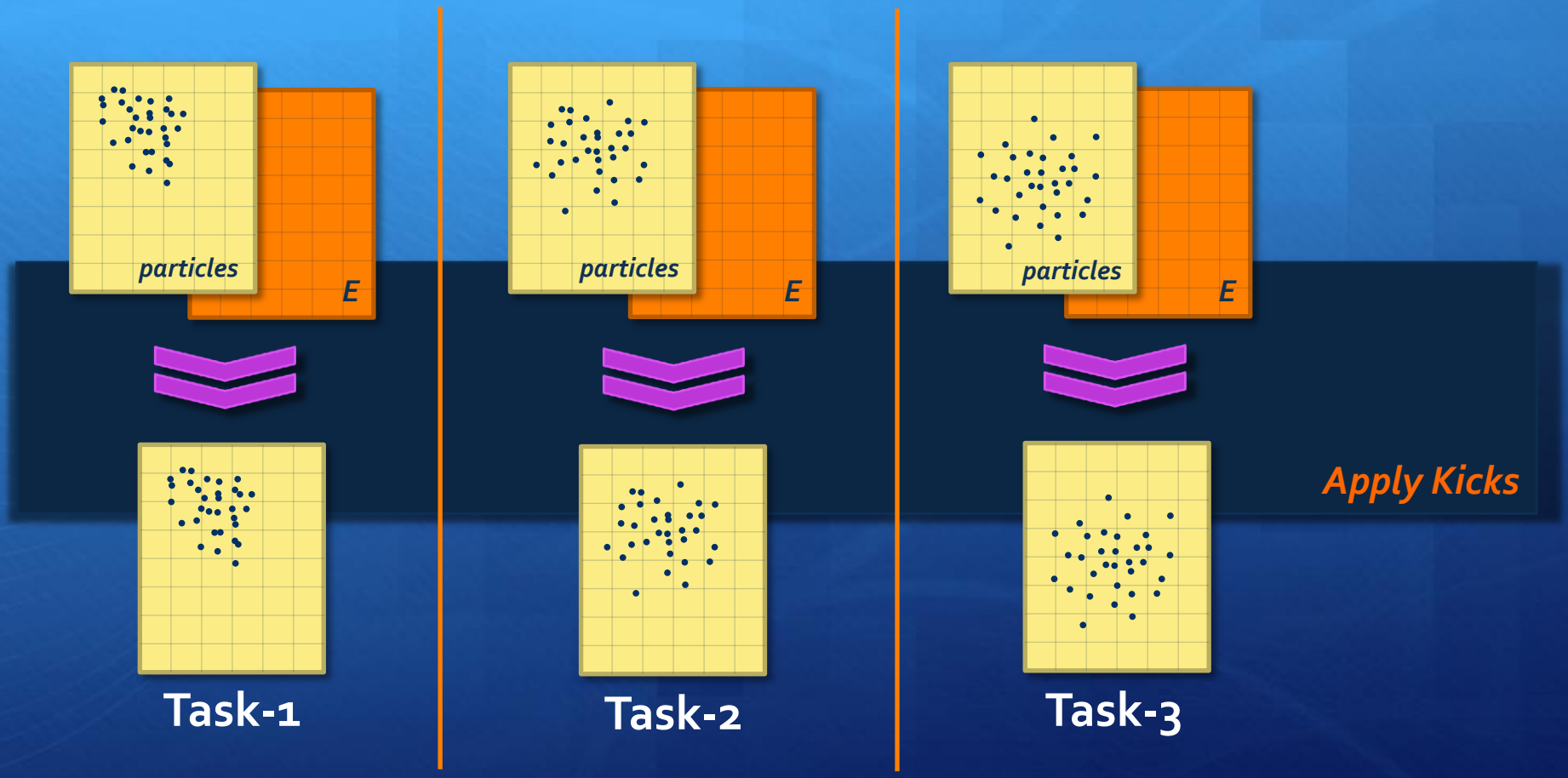
- *Collecting local charge densities via MPI_Allreduce()*

Parallel PIC 2. Field Solver (charge density -> fields)



- *Spatial decomposition in field solver*
- *Global-rho -> phi -> local-En -> global-En*

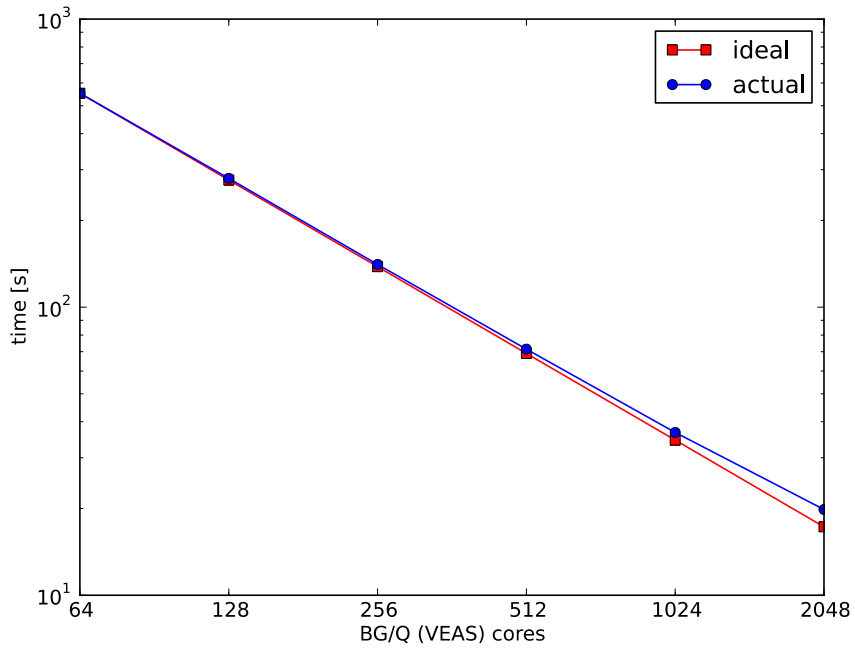
Parallel PIC 3. apply kicks



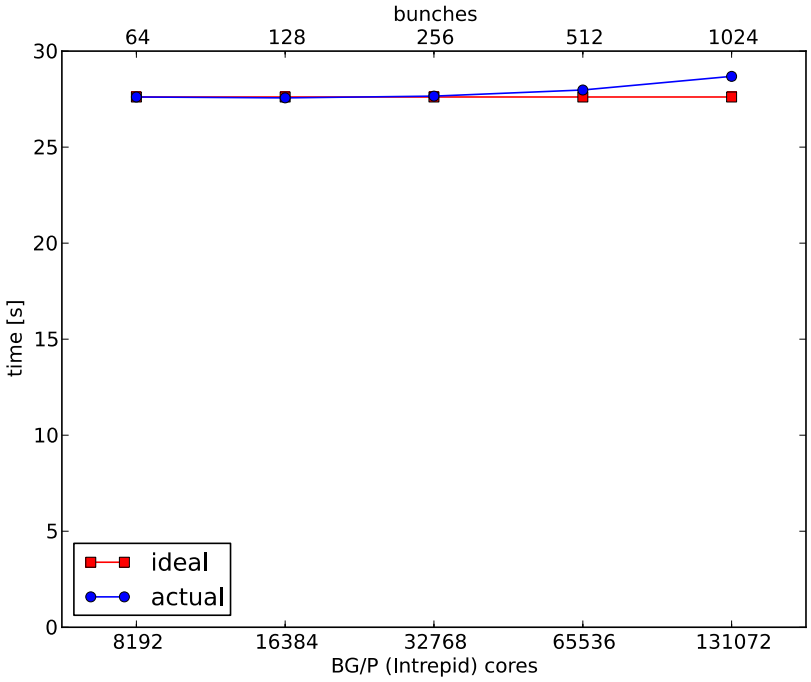
Scalability and Performance

Large problem sizes

Strong scaling



Weak scaling



Scalability and Performance

Medium problem sizes

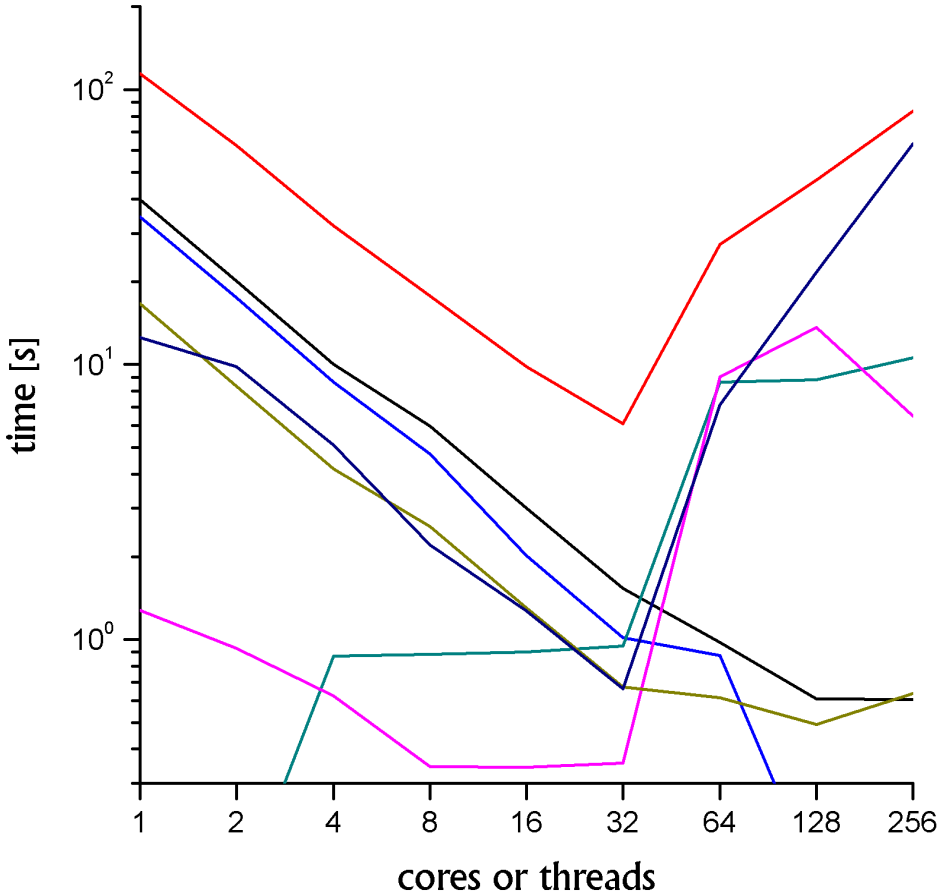
Benchmark Problem

- Four FODO cells
  - 8 steps/cell
  - 2nd order maps
- Space Charge
  - 32 x 32 x 256 grid
- 2,621,440 particles

Machine

- Fermilab's Wilson AMD Cluster
- AMD 6128 Opteron 2GHz
- Quad-socket eight-core (32 cores total)
- Infiniband inter-connection

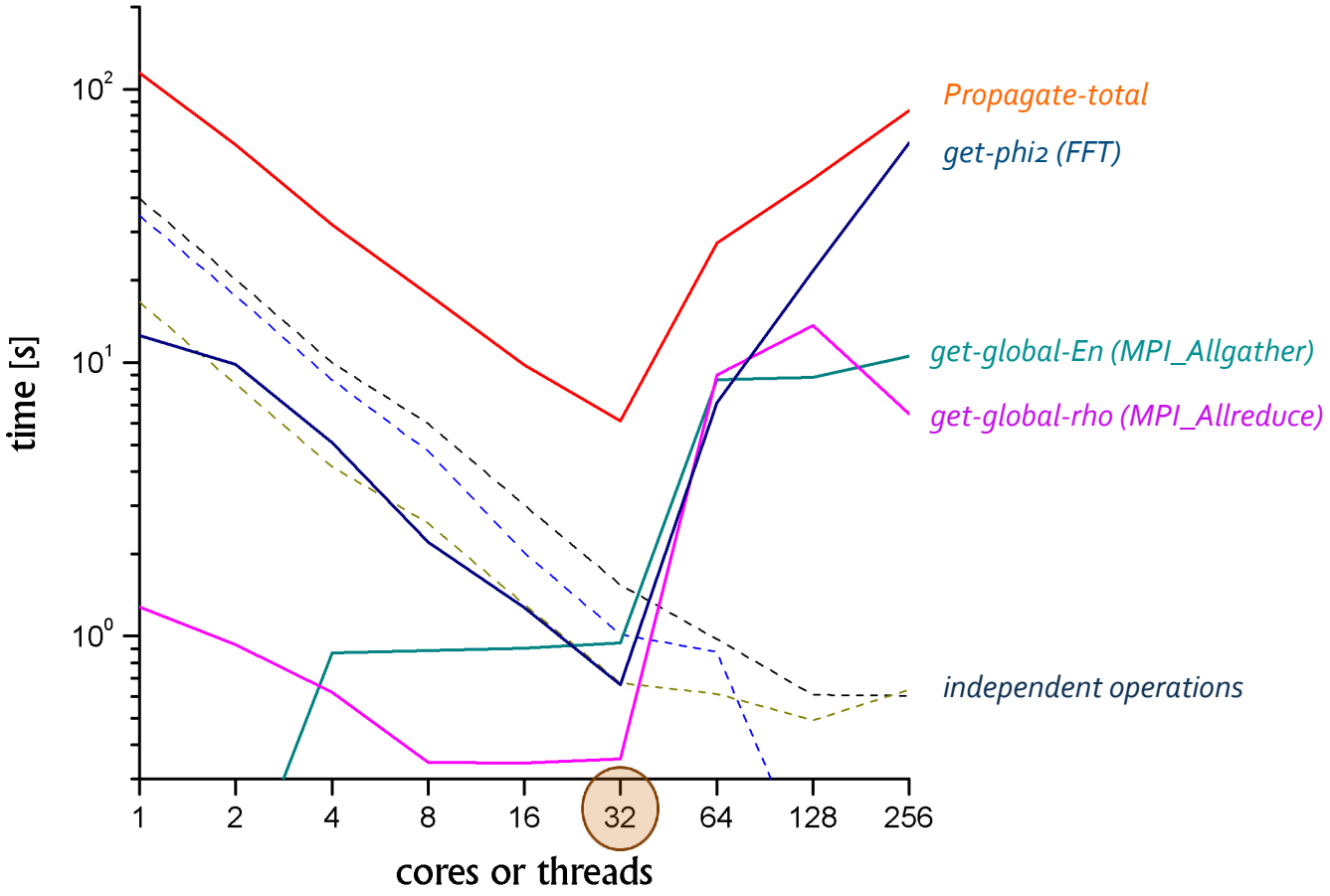
Base strong scaling / performances



- **Benchmark**
  - Four FODO cells
    - 8 steps/cell
    - 2nd order maps
  - Space Charge
    - 32 x 32 x 256 grid
  - 2,621,440 particles
- **Machine**
  - Fermilab's Wilson Cluster
  - 32-core AMD nodes with Infiniband interconnection

- independent-operator-apply
- propagate-total
- sc-apply-kick
- sc-get-global-en
- sc-get-global-rho
- sc-get-local-rho
- sc-get-phi2

Base strong scaling / performances



Optimization Approaches

Medium problem sizes

Multi-node FFT scales poorly using FFTW

- Extra FFT computation on each node
- Alternative FFT implementations

Global communication dominates

- Extra FFT computation on each node (comm. avoidance)
- Reduce MPI tasks (simplifies traffic routes)
- Use multi-threading to offload intra-node communications
- Alternative MPI implementations / optimizations options

Optimization Approaches

Medium problem sizes

Multi-node FFT scales poorly using FFTW

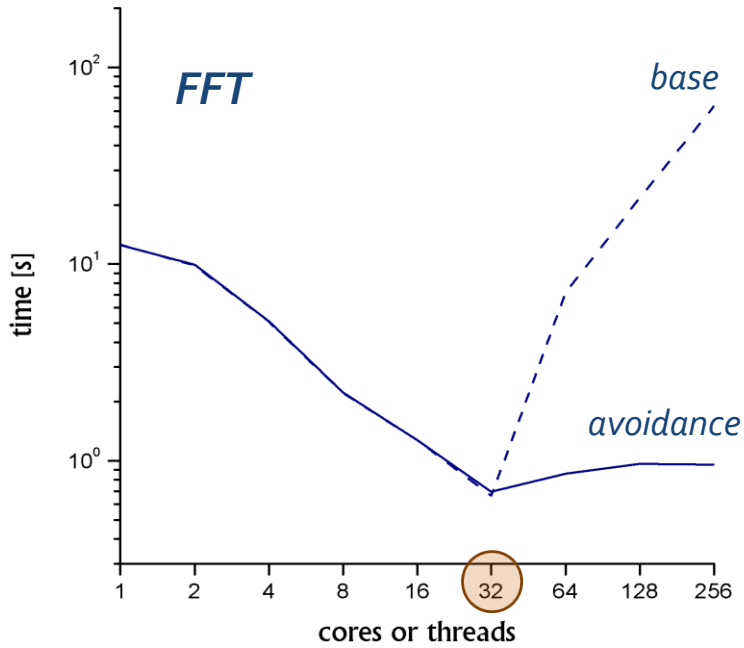
- ✓ **Extra FFT computation on each node**
- Alternative FFT implementations

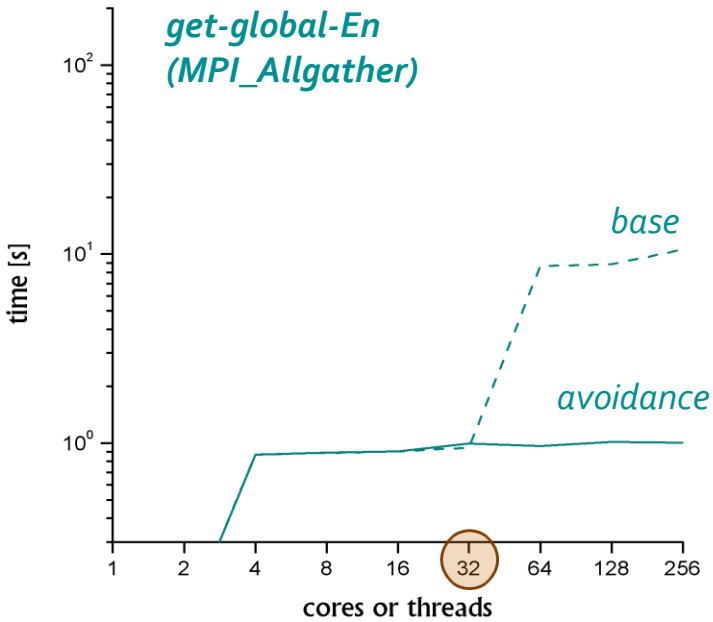
Global communication dominates

- ✓ **Extra FFT computation on each node (comm. avoidance)**
- Reduce MPI tasks (simplifies traffic routes)
- Use multi-threading to offload intra-node communications
- Alternative MPI implementations / optimizations options

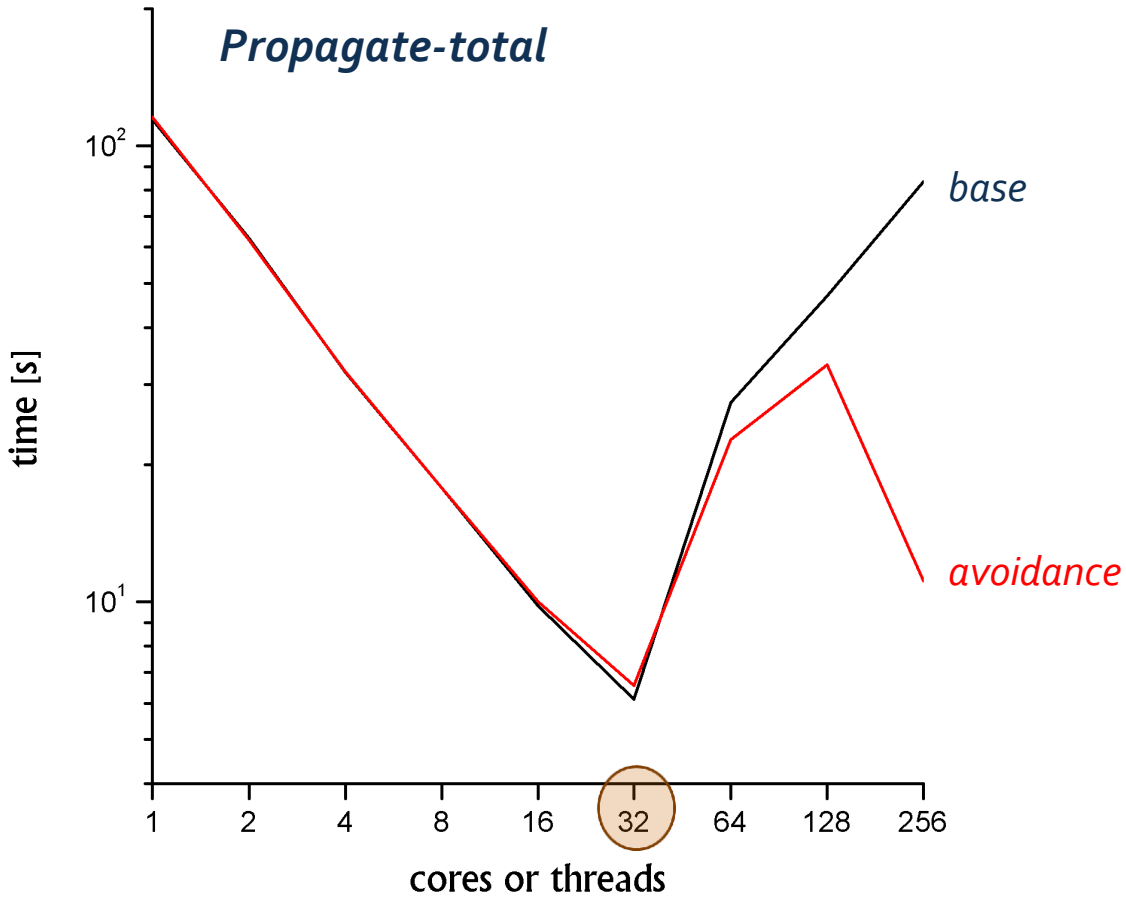
Optimization 1: Communication avoidance

- *Perform redundant FFT computation on each node (32 cores)*
  - Avoid broadcasting fields across nodes
  - Avoid poor scaling of multi-node FFT





Optimization 1: Communication avoidance



Scaling better

- *At 256 cores/8 nodes: 90 sec vs. 11 sec*

But still terrible

- *Best performance at 32 cores/ 1 node*

Optimization Approaches

Medium problem sizes

Multi-node FFT scales poorly using FFTW

- Extra FFT computation on each node
- Alternative FFT implementations

Global communication dominates

- Extra FFT computation on each node (comm. avoidance)
- ✓ **Reduce MPI tasks (simplifies traffic routes)**
- ✓ **Use multi-threading to offload intra-node communications**
- Alternative MPI implementations / optimizations options

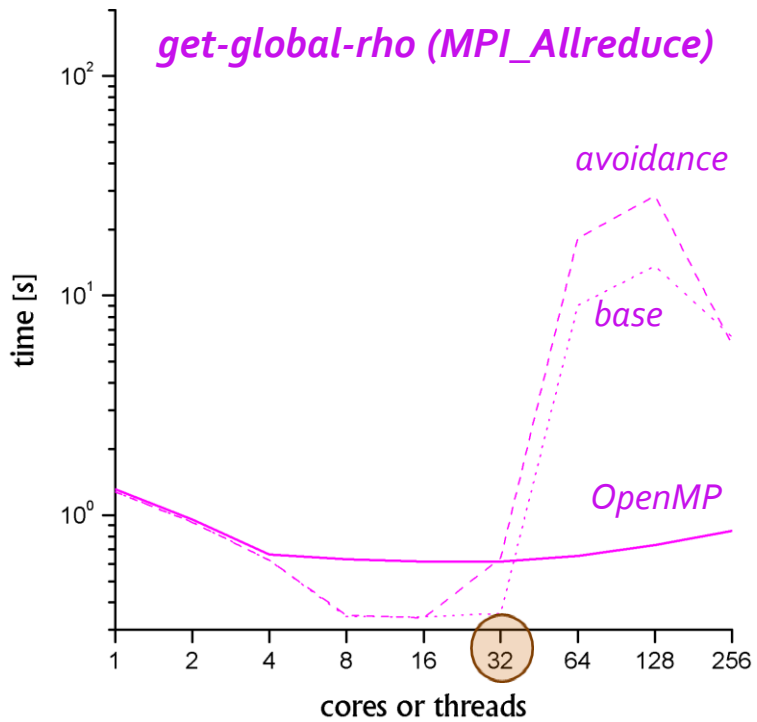
Optimization 2: MPI + OpenMP Hybrid

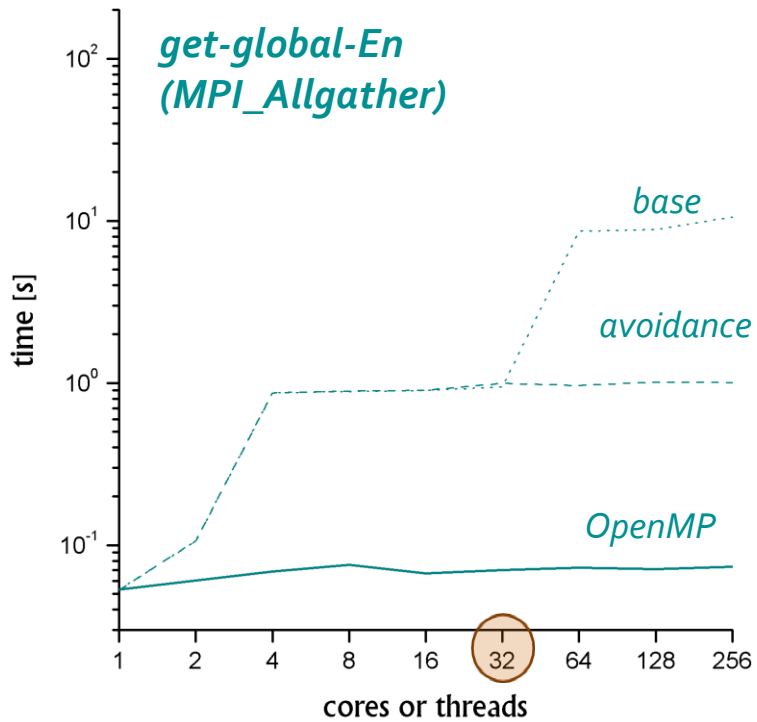
- MPI + OpenMP
  - ***OpenMP*** inside of a node, and ***MPI*** between nodes

- **Shared memory** (multi-threading)
  vs. **Distributed memory** (multi-tasking)
  - Direct access of other threads' memory without the need of communication overhead
  - **Need to deal with race condition to avoid data pollution**
  - Possible memory contention
  - NUMA effects

Optimization 2: MPI + OpenMP Hybrid

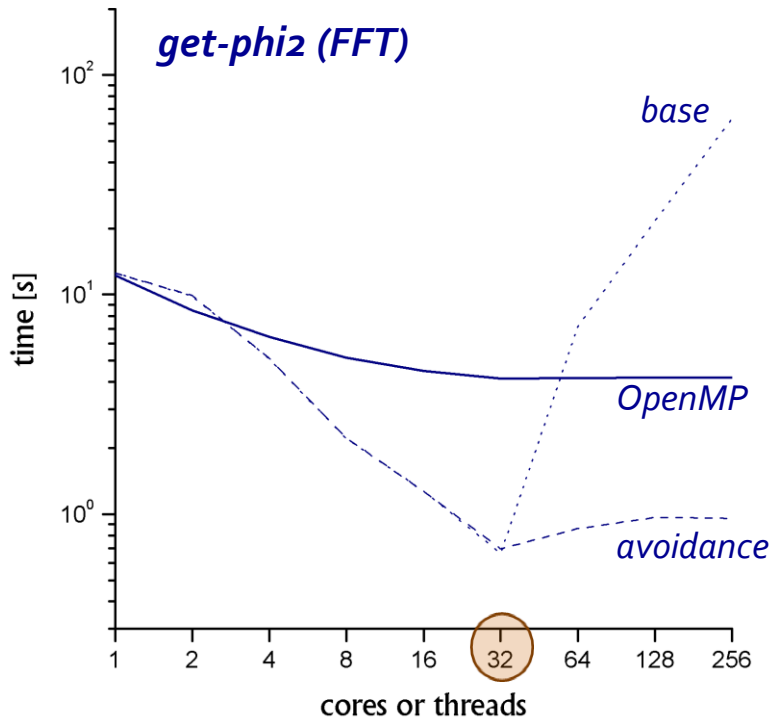
+ Communication improves significantly

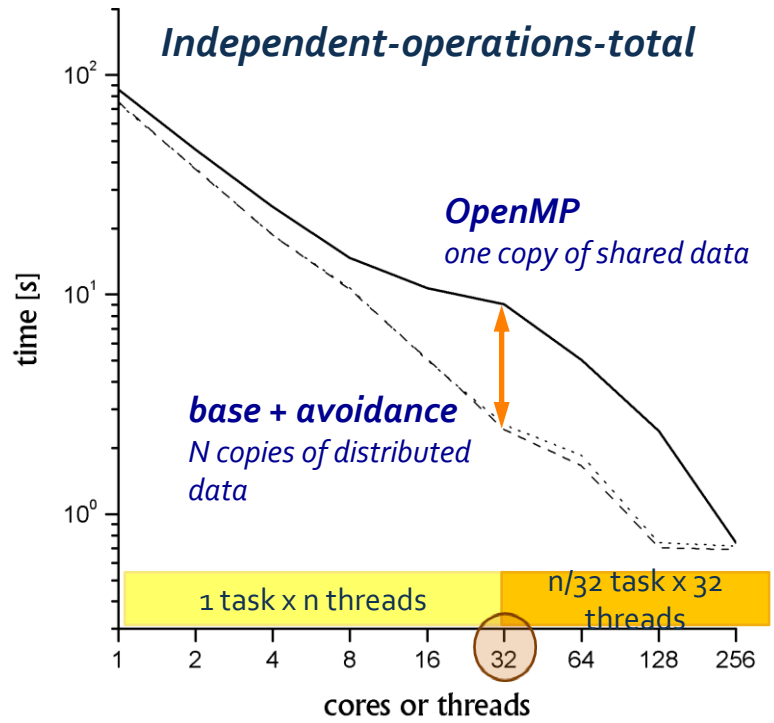




Optimization 2: MPI + OpenMP Hybrid

- FFT (FFTW) deteriorated; independent ops affect by NUMA





Optimization Approaches

Medium problem sizes

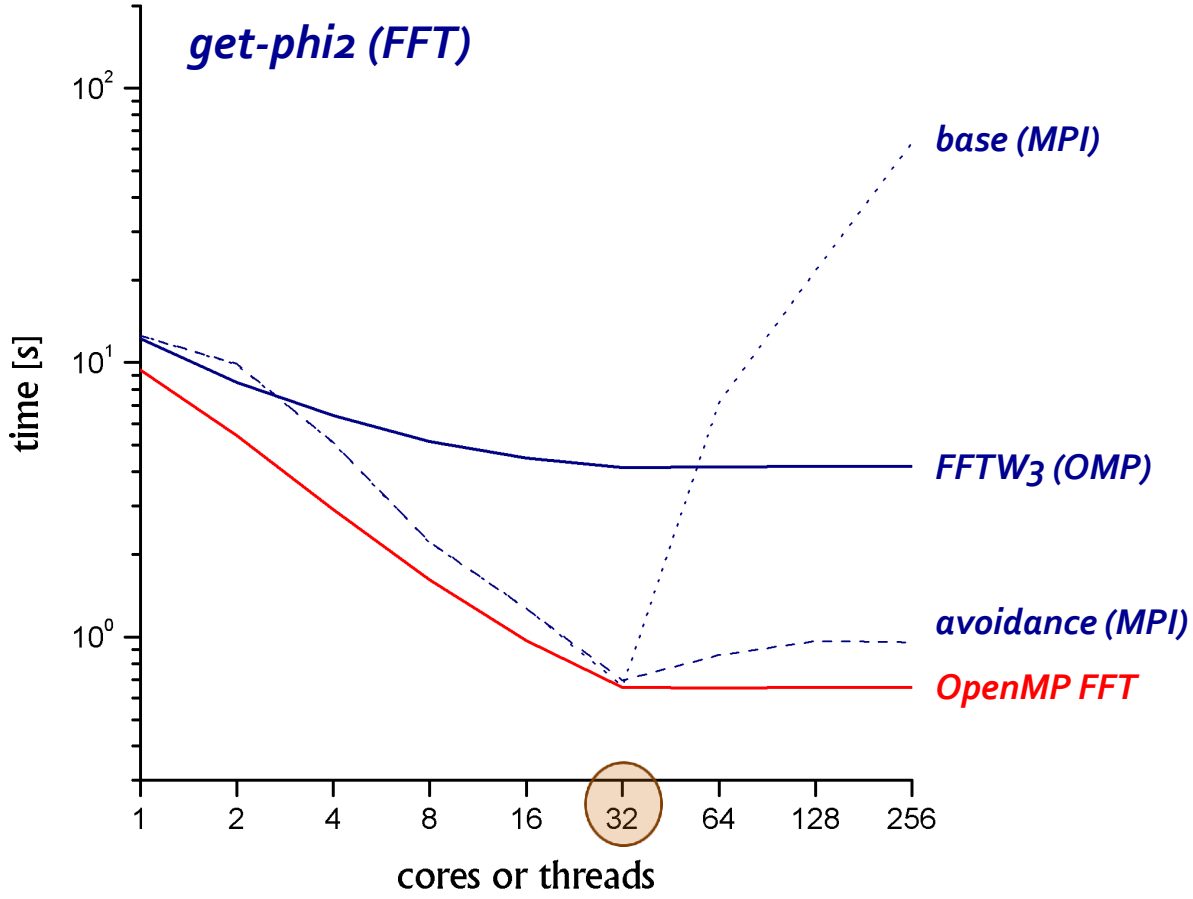
Multi-node FFT scales poorly using FFTW

- Extra FFT computation on each node
- ✓ Alternative FFT implementations

Global communication dominates

- Extra FFT computation on each node (comm. avoidance)
- Reduce MPI tasks (simplifies traffic routes)
- Use multi-threading to offload intra-node communications
- Alternative MPI implementations / optimizations options

Optimization 3: Customized parallel 3D-FFT



Best 3D-FFT performance we have so far

Optimization Approaches

Medium problem sizes

Multi-node FFT scales poorly using FFTW

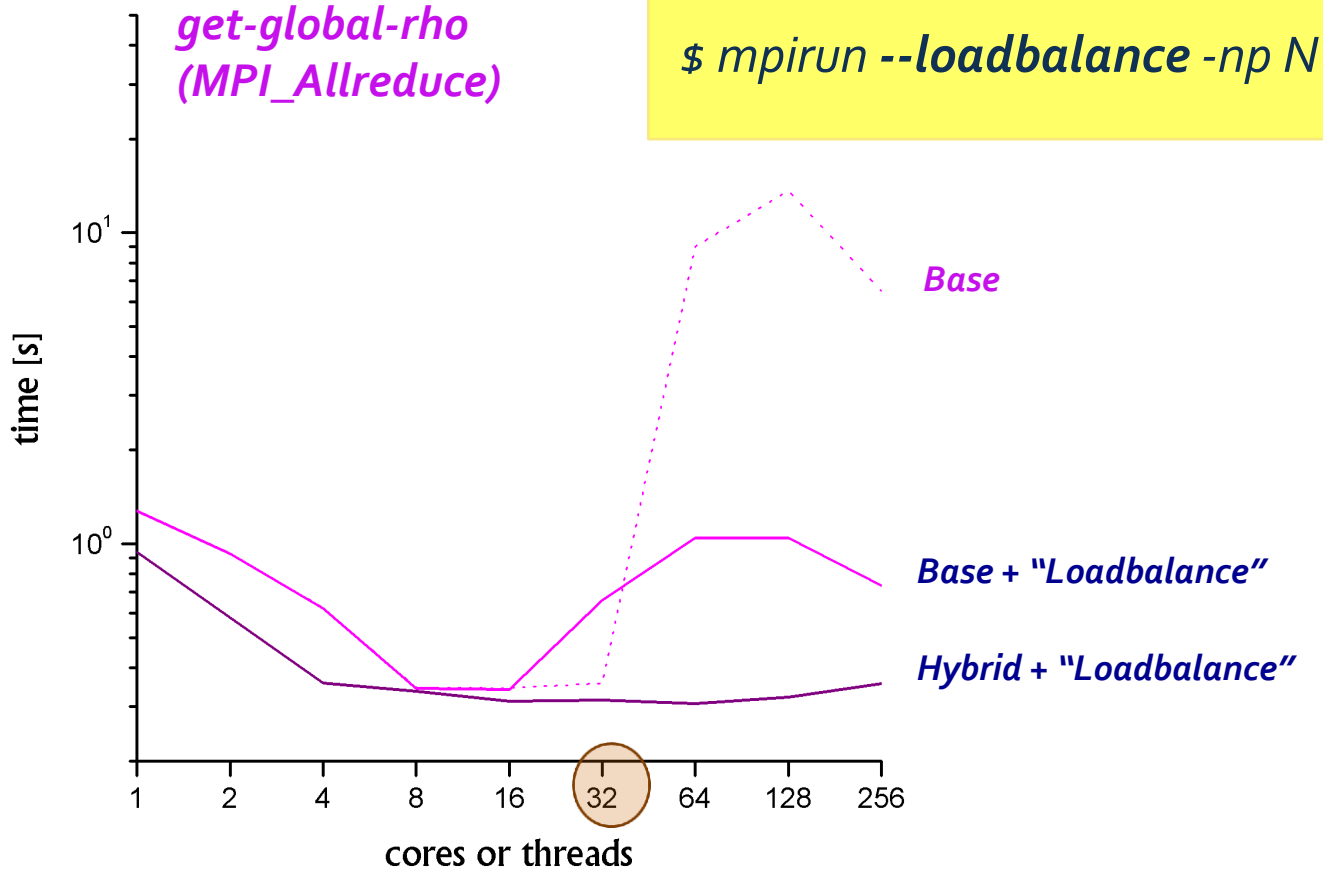
- Extra FFT computation on each node
- Alternative FFT implementations

Global communication dominates

- Extra FFT computation on each node (comm. avoidance)
- Reduce MPI tasks (simplifies traffic routes)
- Use multi-threading to offload intra-node communications
- ✓ Alternative MPI implementations / optimizations options

Optimization 4: MPI optimizations

*$ mpirun **--loadbalance** -np N application*



Optimization Approaches

Medium problem sizes

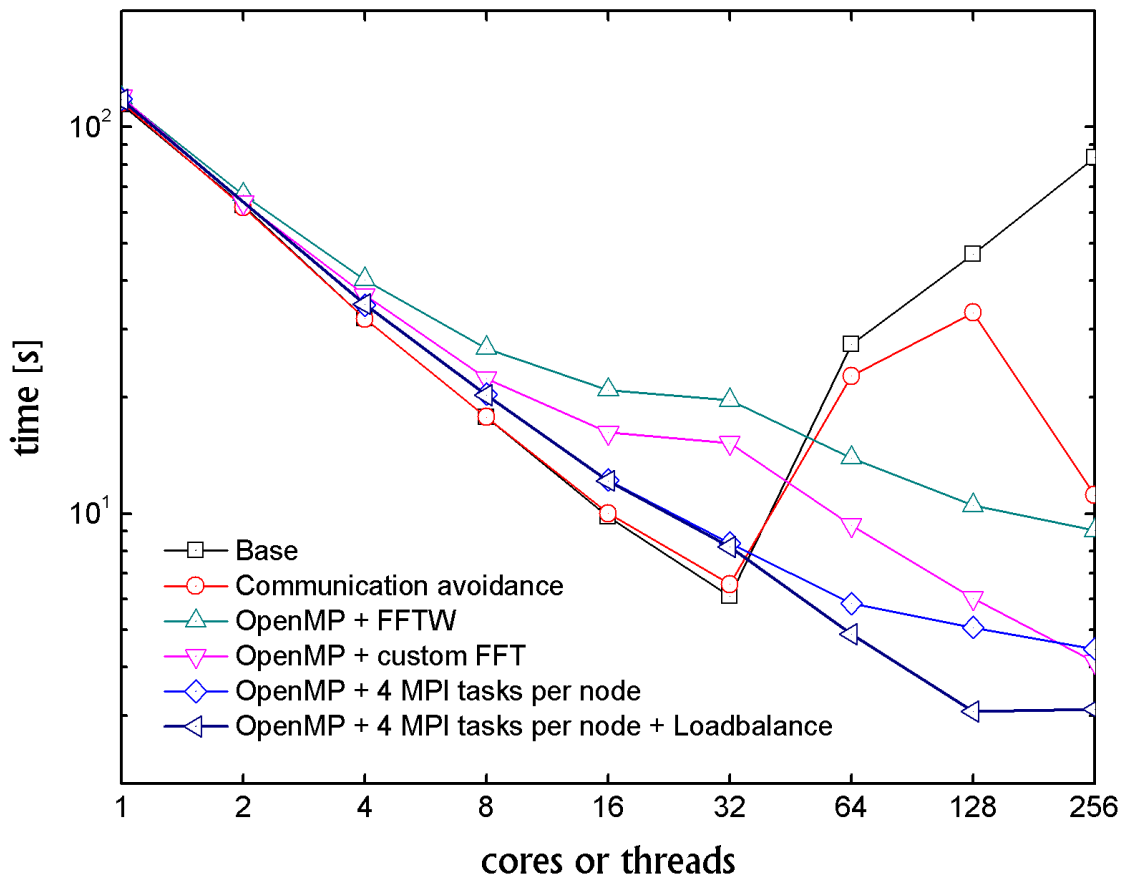
Multi-node FFT scales poorly using FFTW

- ✓ Extra FFT computation on each node
- ✓ Alternative FFT implementations

Global communication dominates

- ✓ Extra FFT computation on each node (comm. avoidance)
- ✓ Reduce MPI tasks (simplifies traffic routes)
- ✓ Use multi-threading to offload intra-node communications
- ✓ Alternative MPI implementations / optimizations options

Optimization 1+2+3+4: Final scaling



Much improved scaling

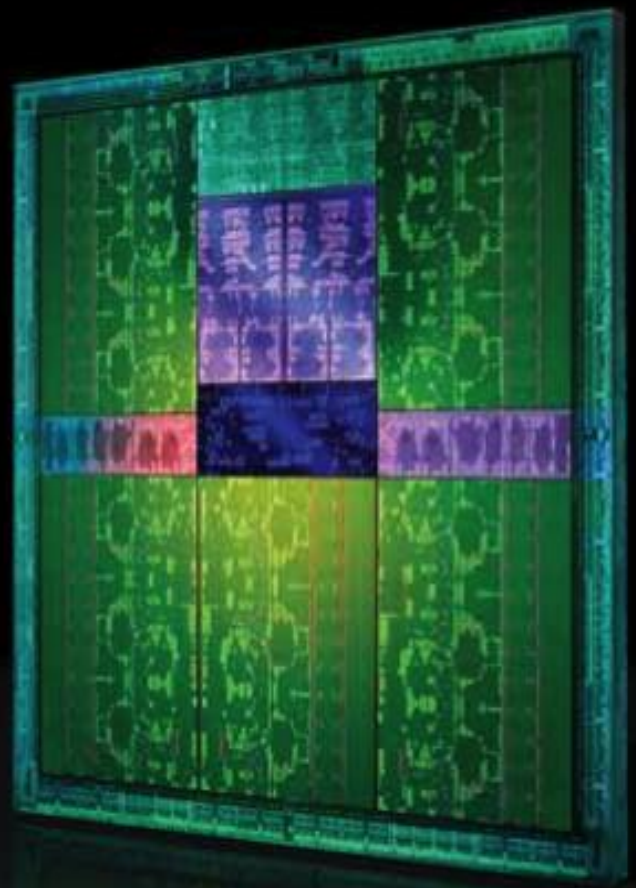
- *Scales up to 128 cores*

Better peak performance

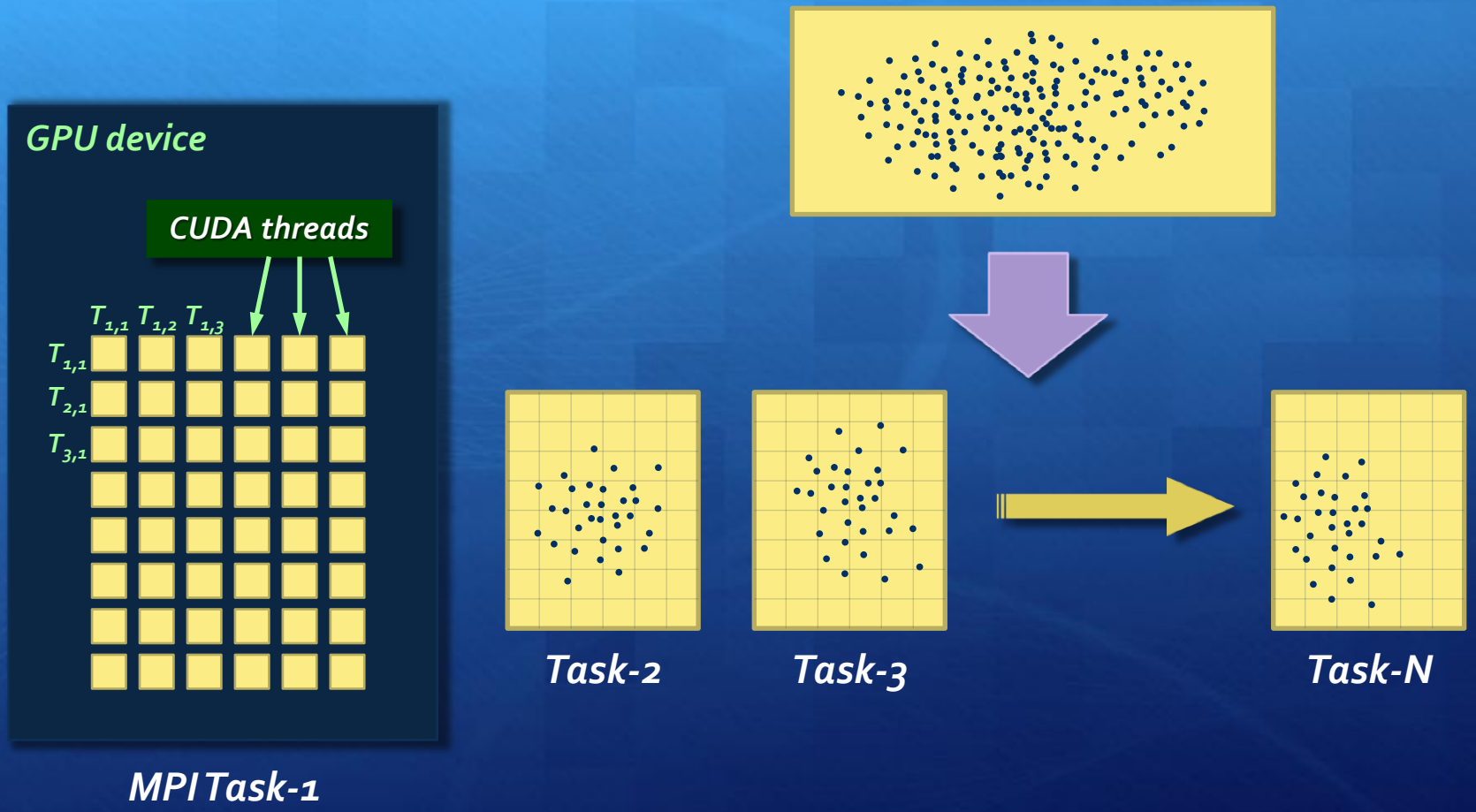
- *2.93 sec vs. 6.22 sec*
- ***A factor of 2 improvement***

GPU accelerated Synergia

NVIDIA® KEPLER GK110
NEXT-GENERATION CUDA®
COMPUTE ARCHITECTURE

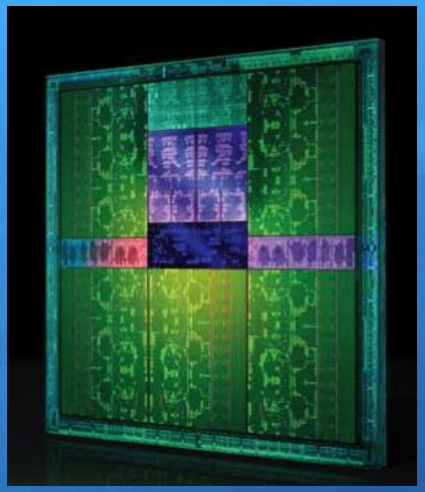


Parallel PIC + CUDA



- *Excessive number of threads at finer granularity*

CUDA-fication of the PIC Code



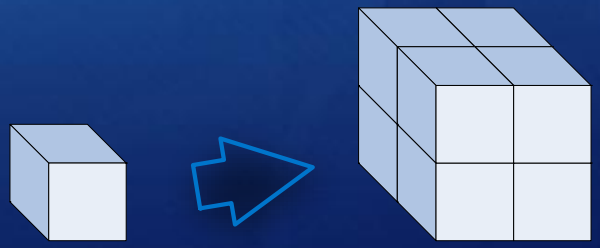
1. ***Bunch, statistics and diagnostics:***

   - Calculate the *mean* and *standard deviation* of coordinates and momentums for particles in the bunch

CUDA: *Parallel reduction*

1. ***Charge deposition:***

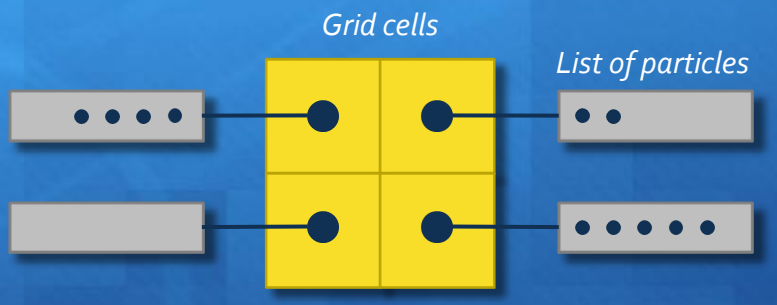
   - One macro particle can contribute up to 8 grid cells

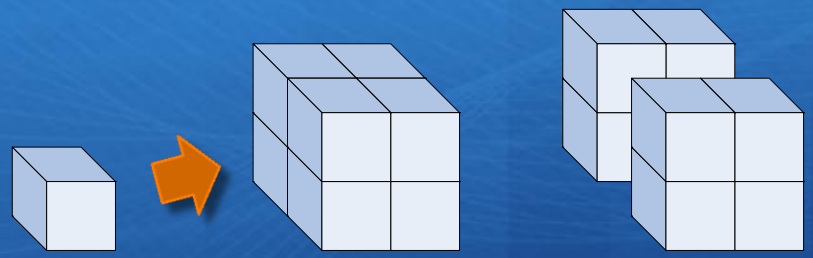


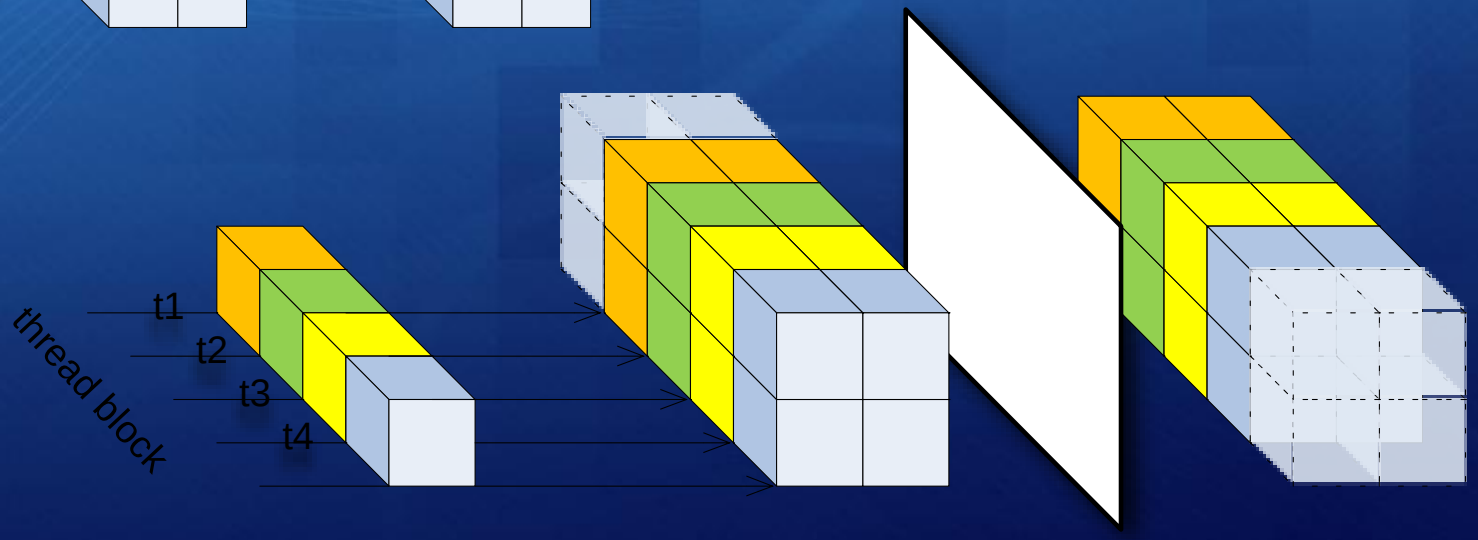
CUDA: *Sort particle to cell and use interleaved updates to avoid race condition*

CUDA-fication of the PIC Code

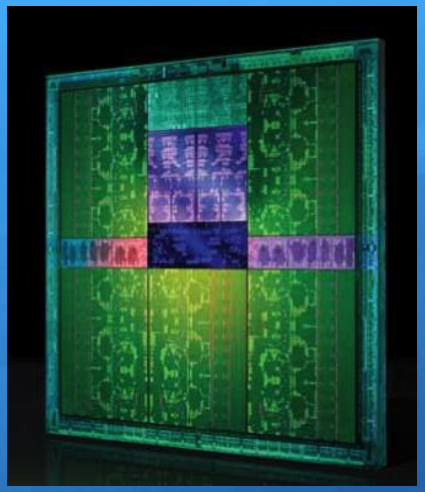
Sort particles to cells
& interleaved deposition







CUDA-fication of the PIC Code



3. ***Green function:***

$$G(x,y,z) = (x^2 + y^2 + z^2)^{-1/2}$$

CUDA: *One thread per cell easy parallelizable*

4. ***Poisson solver:***

$$\hat{\rho}(u,v,w) = \iiint_{\infty} \rho(x,y,z) e^{-2\pi i(ux+vy+wz)} dxdydz$$

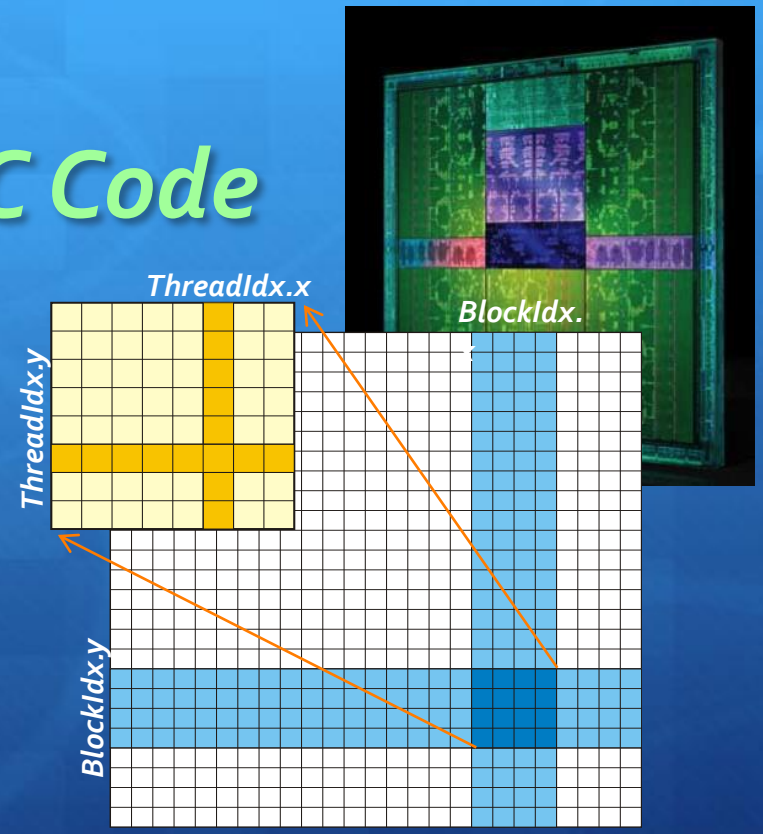
$$\hat{G}(u,v,w) = \iiint_{\infty} G(x,y,z) e^{-2\pi i(ux+vy+wz)} dxdydz$$

$$\hat{\varphi}(u,v,w) = \hat{\rho}(u,v,w) \times \hat{G}(u,v,w)$$

$$\varphi(x,y,z) = \iiint_{\infty} \hat{\varphi}(u,v,w) e^{2\pi i(ux+vy+wz)} dudvdw$$

CUDA: *CUFFT library for Fourier transforms*

CUDA-fication of the PIC Code



5. ***Electric field:***

$$E_x(x,y,z) = -\partial\varphi(x,y,z)/\partial x$$
$$E_y(x,y,z) = -\partial\varphi(x,y,z)/\partial y$$
$$E_z(x,y,z) = -\partial\varphi(x,y,z)/\partial z$$

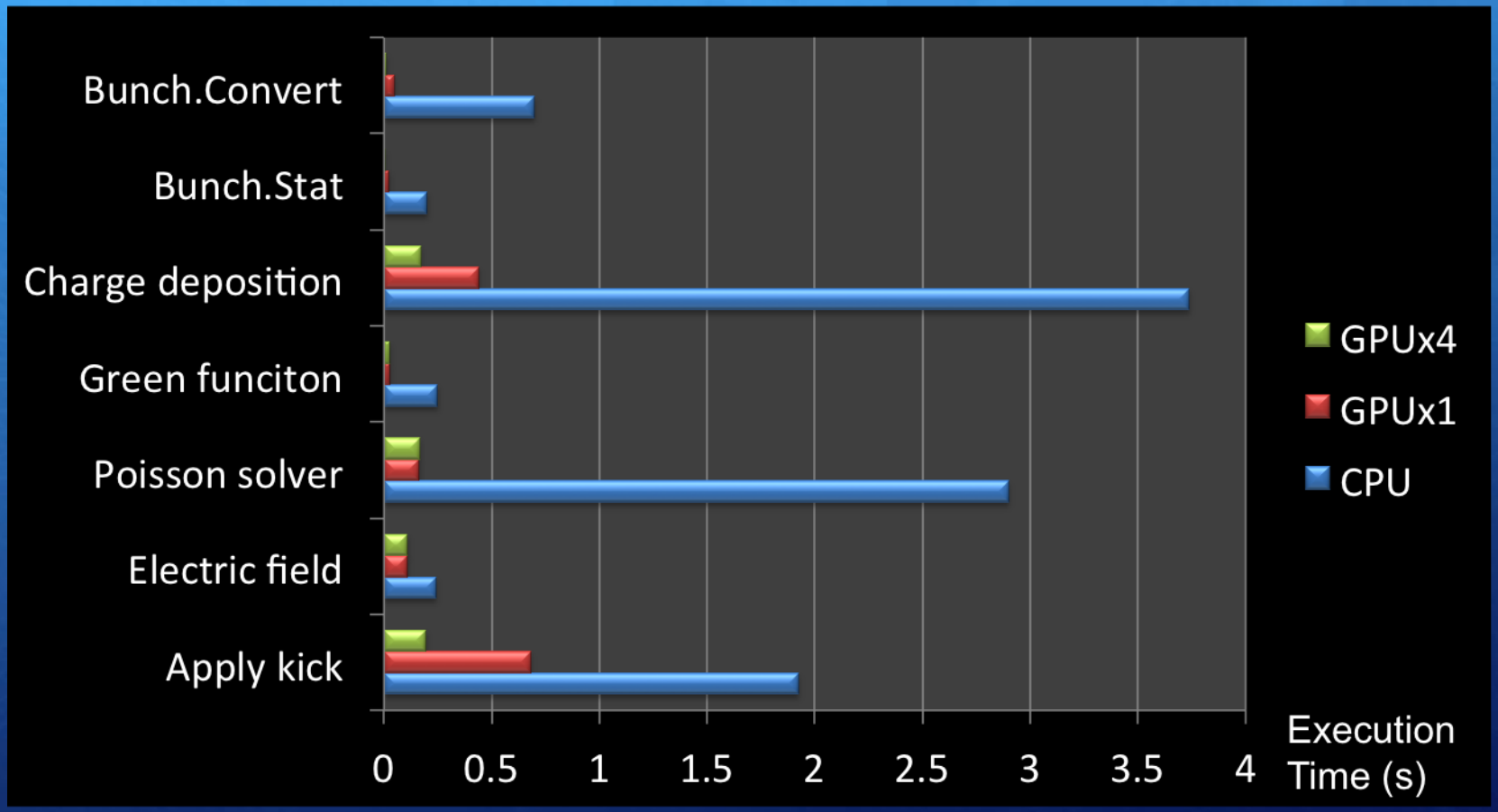
CUDA: *use shared memory to reduce the global memory access*

6. ***Apply kick:***

- Advance the *position* and *momentum* for each particle in the bunch

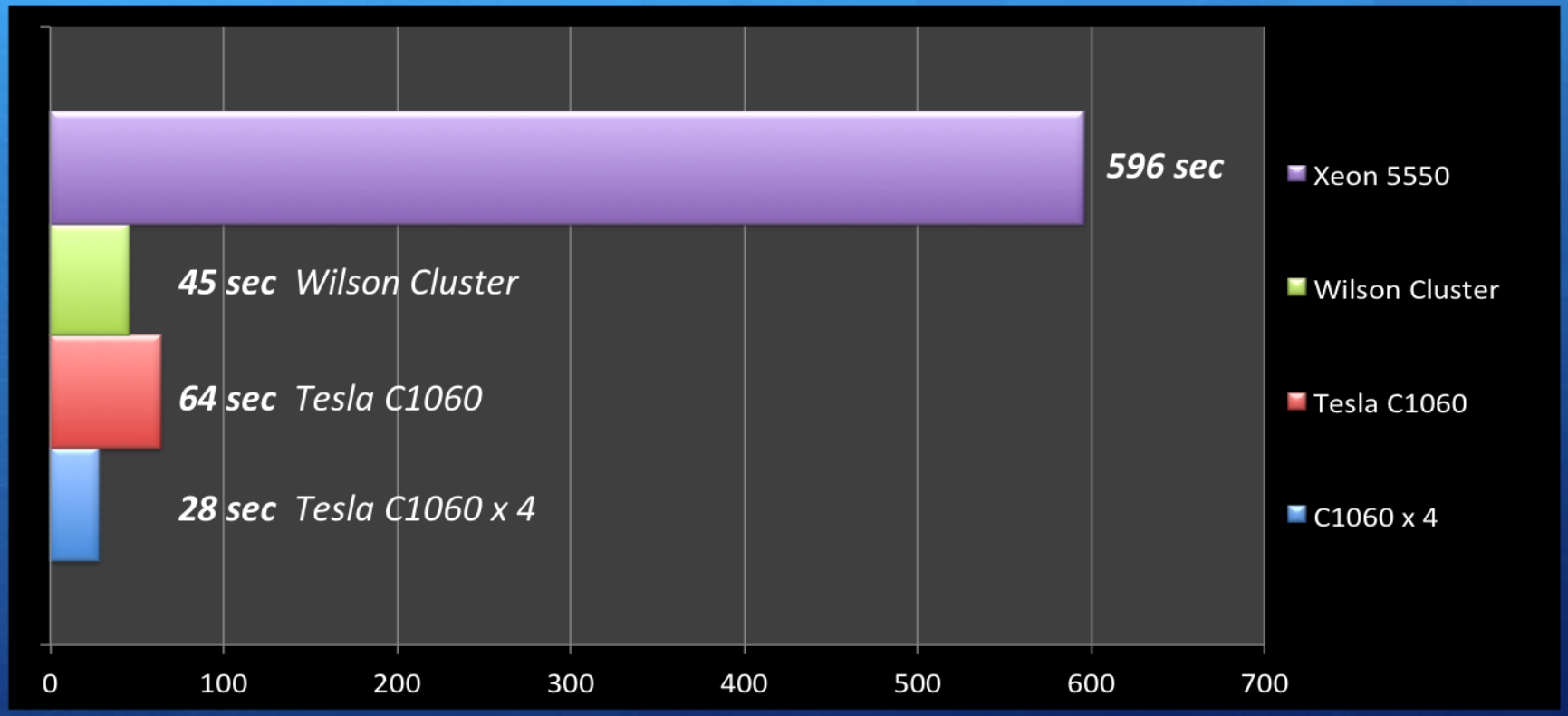
CUDA: *One thread per cell. Keep particles sorted for grouped access of En*

Performance Comparison for Operations



1. Intel Xeon X5550, single process @ 2.67GHz;
2. NVidia Tesla C1060, 30 streaming multi-processors @ 1.30GHz in a single GPU
3. Nvidia Tesla C1060 x 4

Overall Performance Comparison



1. Intel Xeon X5550, single process @ 2.67GHz;
2. Fermilab Wilson Xeon Cluster, dual Xeon X5650 2.67GHz nodes. 16 nodes / 128 cores used
3. NVidia Tesla C1060, 30 streaming multi-processors @ 1.30GHz in a single GPU
4. Nvidia Tesla C1060 x 4