Parallel Particle-in-Cell Codes in Plasma Physics

Viktor K. Decyk
UCLA
July, 2007

Abstract:
PIC codes, which integrate the trajectories of charged particles in the electromagnetic fields they generate, are widely used in plasma physics. The first parallel versions were developed in 1987 at JPL. Parallelization strategies and issues in achieving high performance will be discussed.

What are Particle-in-Cell Codes?

PIC codes integrate the trajectories of many particles interacting self-consistently via electromagnetic fields.
A grid is used as a scaffolding to calculate fields rather than direct binary interactions => reduces calculation to order N

Simplest code is electrostatic:

1. Calculate charge density on a mesh from particles

$$\rho(r) = \sum_i q_i S(r - r_i)$$

2. Solve Poisson's equation, Spectral or Finite-Difference

$$\nabla \cdot E = 4\pi\rho$$

3. Advance particle's coordinates using Newton's Law

$$m_i \frac{dv_i}{dt} = q_i E(r_i) \qquad \frac{dx_i}{dt} = v_i$$

Particle-in-Cell Codes

More complex is electromagnetic:

1. Calculate current density from particles

$$j(r) = \sum_i q_i v_i S(r - r_i)$$

2. Solve Maxwell's equation, Spectral or Finite-Difference

$$\nabla \times B = \frac{4\pi}{c} j + \frac{1}{c}\frac{\partial E}{\partial t} \qquad \nabla \times E = -\frac{1}{c}\frac{\partial B}{\partial t}$$

3. Advance particle coordinates using Lorentz force:

$$m_i \frac{dv_i}{dt} = q_i \left[E(r_i) + v_i \times B(r_i)/c \right] \qquad \frac{dx_i}{dt} = v_i$$

Major source of numerical error in PIC: aliasing from grid

PIC codes make the fewest physics approximations

- Used when continuum or approximate models fail
- Used to test realm of validity of new approximate models
- Used to debug other codes

BUT, most computationally demanding

⇒ 3D PIC generally requires parallel computers

Variety of PIC codes exist:

- Electrostatic, Darwin, Electromagnetic, Relativistic
- Other external forces can be added

PIC codes used in many areas of physics

- Fusion research, Tokamak modeling
- Space plasma physics, magnetosphere
- Electronic devices, gyrotrons
- Laser-plasma interactions, plasma accelerators
- Plasma processing
- Linear accelerators
- Astrophysics, cosmology
- Quantum plasmas

Distributed Memory Computing

Conventional wisdom: difficult to program

- Message-passing very low level

True, but:

- Message-passing isolated
- Communication patterns can be reused
- Excellent performance
- Shared Memory conflicts do not occur
- Scales well to thousands of processors

Figuring out how to partition the problem is the hard part, not the message-passing.

Parallelization of PIC Code

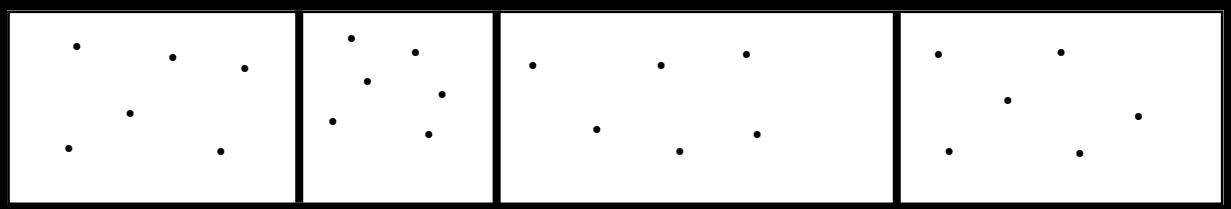
Domain decomposition for distributed memory computers

Primary Decomposition load balances particles

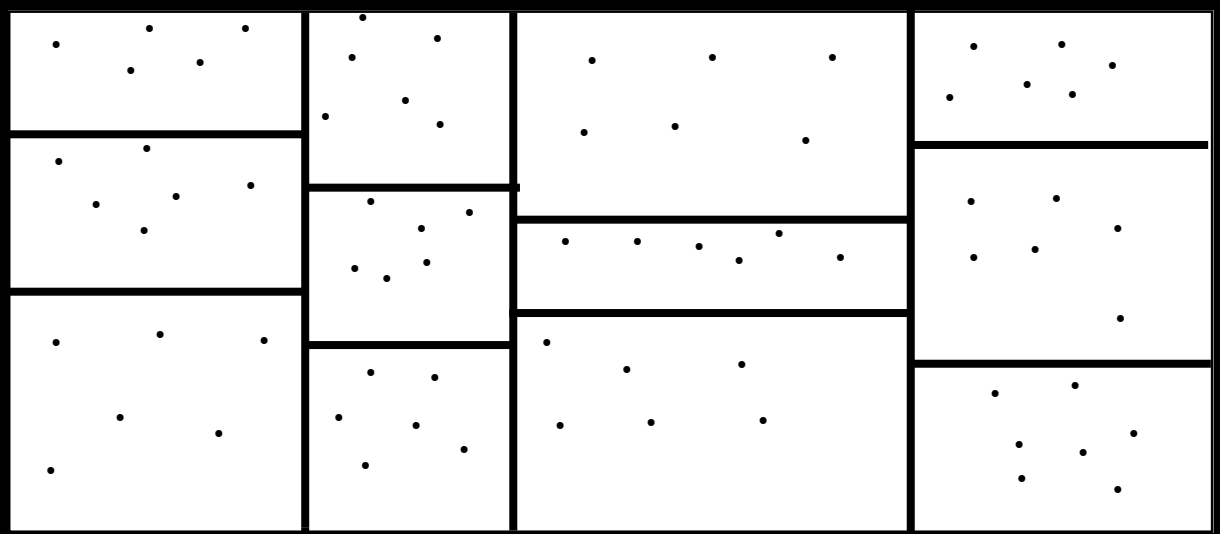
- Sort particles according to spatial location
- Same number of particles in each non-uniform domain
- Scales to thousands of processors

Particle Manager responsible for moving particles

- Particles can move across multiple nodes



1D Domain decomposition



2D Domain decomposition

Each partition has equal number of particles

Particle Manager

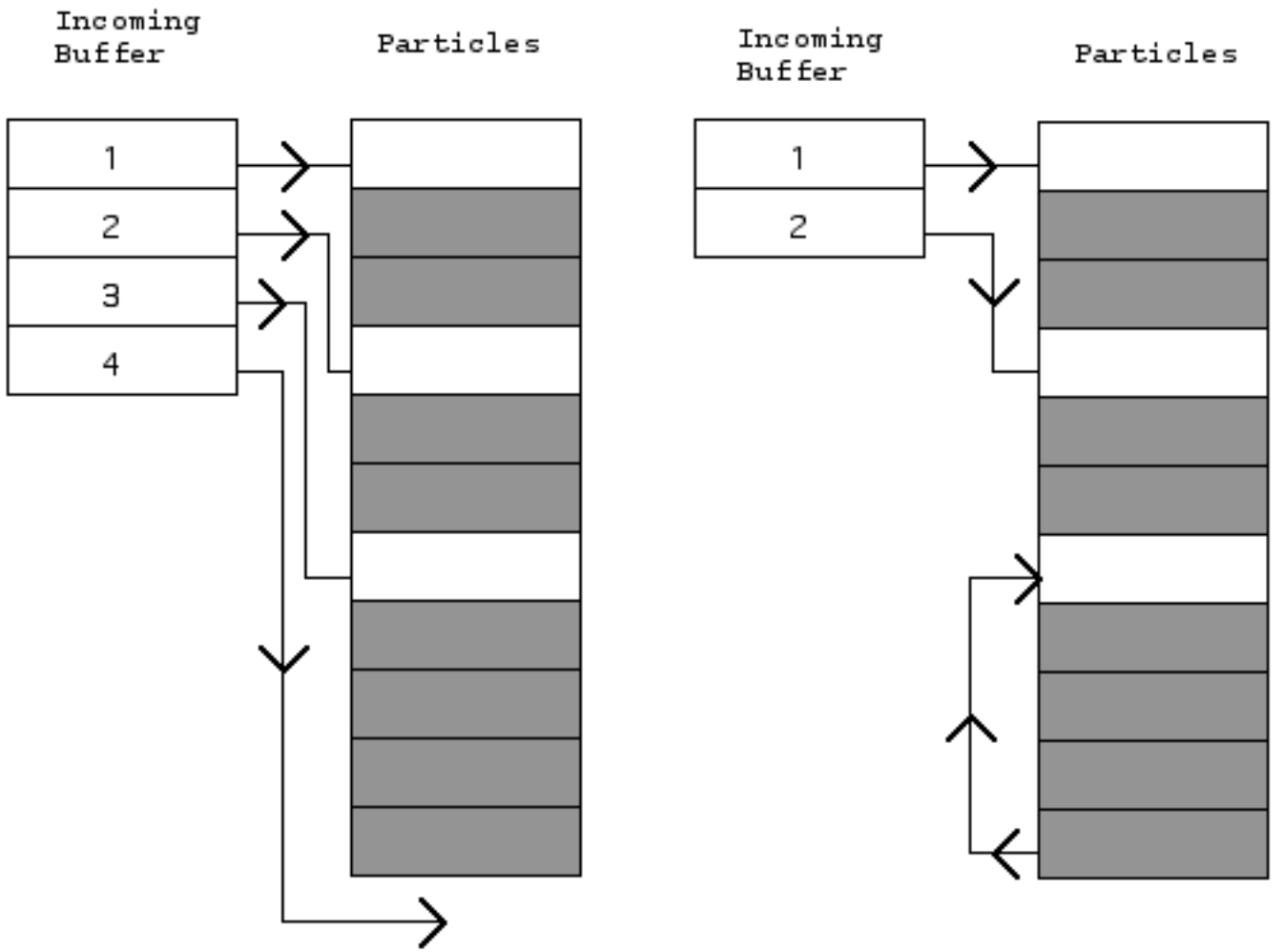
Nodes organized by physical location in space. After update, particles distributed to appropriate node

All holes in particle array are filled

- Particles can move across multiple nodes
- Particle data will be processed in pieces, if memory low
- Particle data will not be overwritten if array is too small

Particle boundaries can be changed drastically

If overflow is detected, partition can be changed without failure



PLIB: Particle Manager Message-Passing

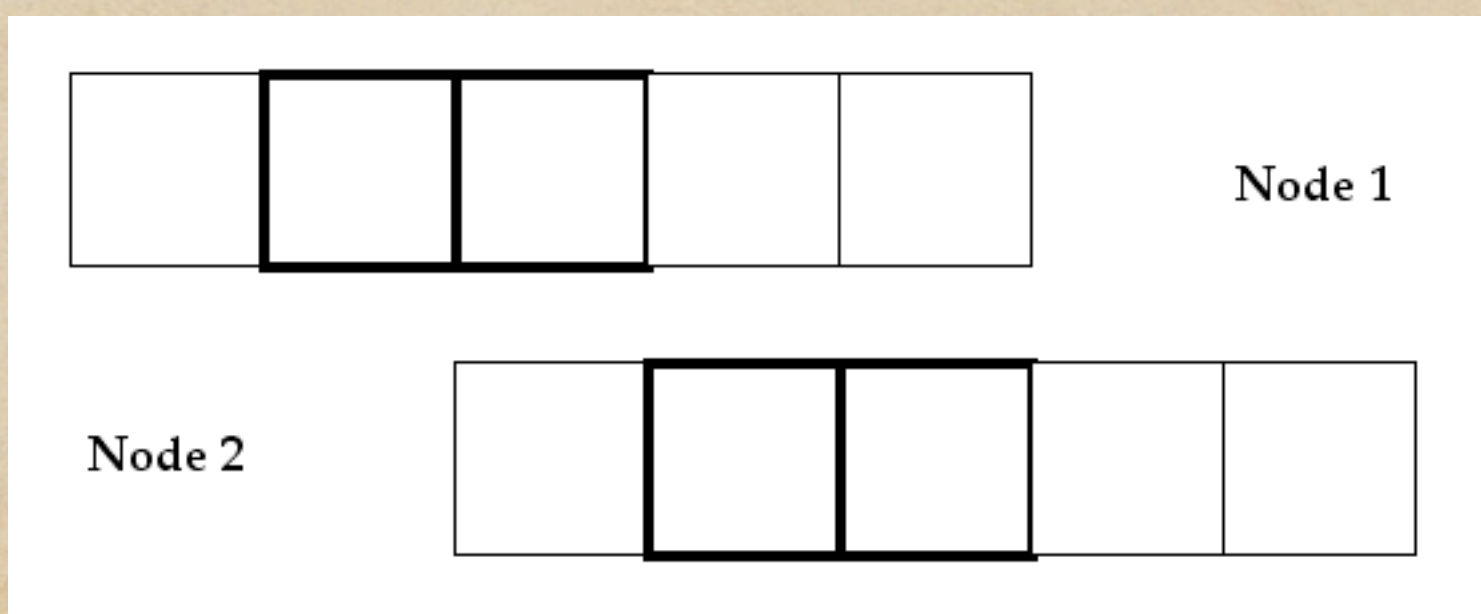
Communication pattern used by Field manager

Field quantities (charge density, electric fields) have guard cells to avoid excessive communication

density deposited in guard cells passed to "owner"

electric field replicated to guard cells before advancing particles

Linear and quadratic interpolation supported



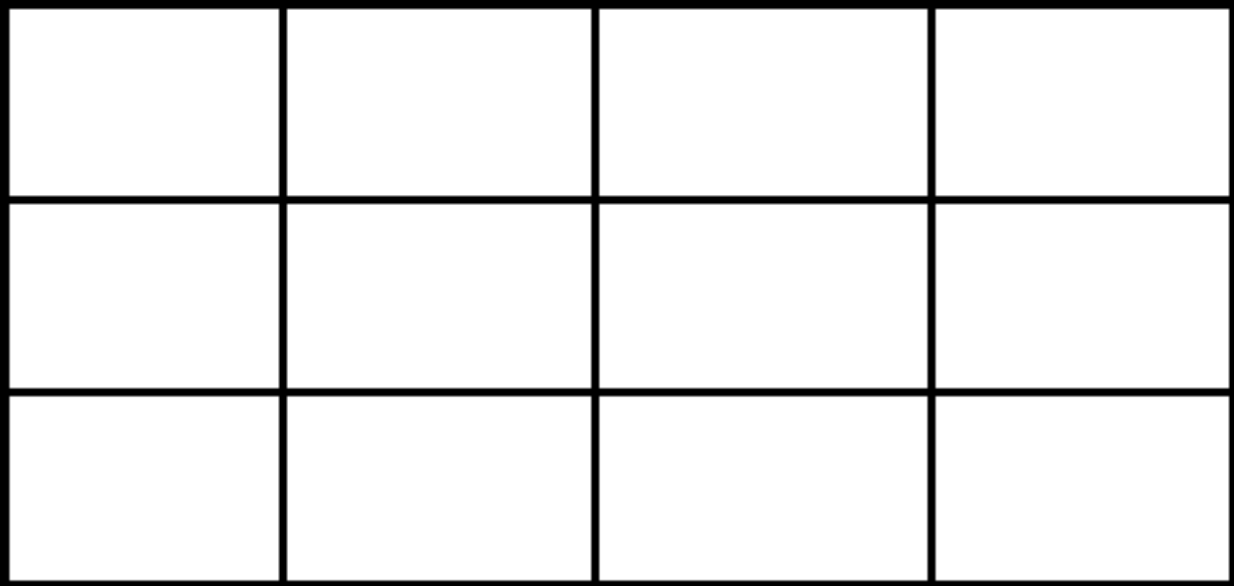
Secondary decomposition

To load balance particles, non-uniform partitions are used. This typically uses most CPU time.

However, to load balance field solver, uniform partitions are often best, for example for FFT.

Therefore, we use two different partitions.

1D Domain decomposition



2D Domain decomposition

Each partition has equal number of grids

Partition Manager

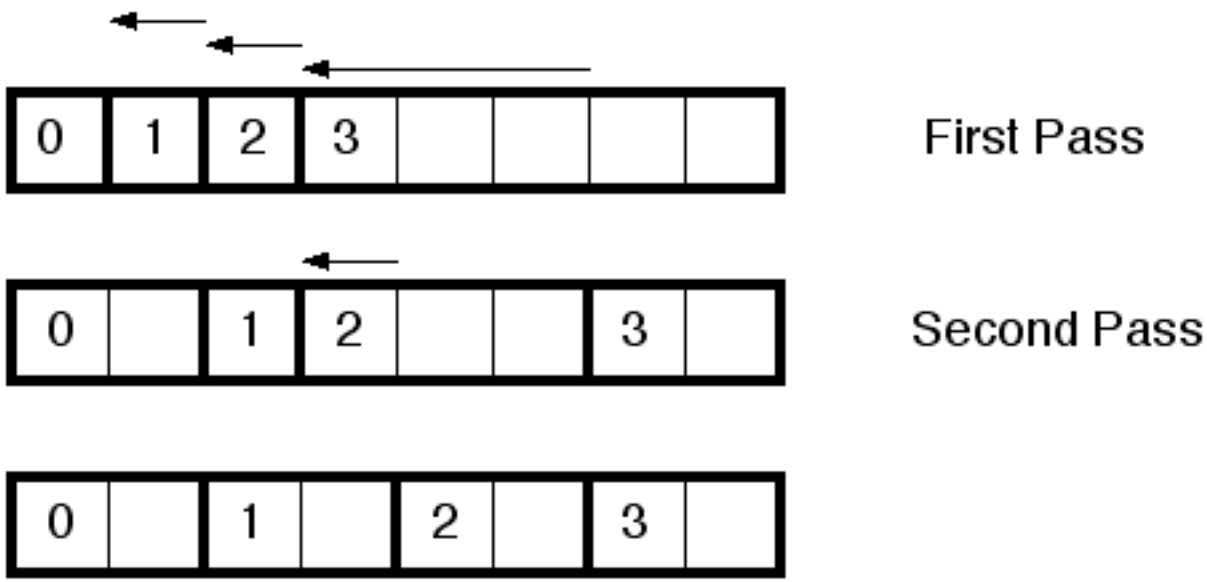
Moves data between non-uniform and uniform partitions

Fields which do not belong, passed to appropriate neighbor, then passed again if necessary. Remembers how many passes are required.

Finds new partitions based on number of particles per cell, accurate to nearest cell. Can also find new partitions from known distribution.

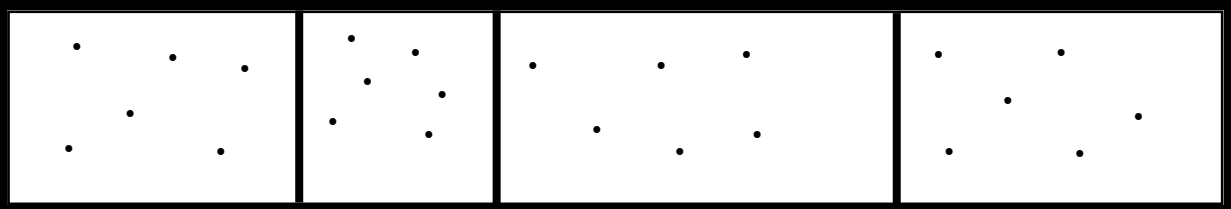
Partition manager is called every time step, since fields needed by particles are in non-uniform partitions, but FFT based solvers require uniform partitions.

Moving Data from Non-uniform to Uniform Partition

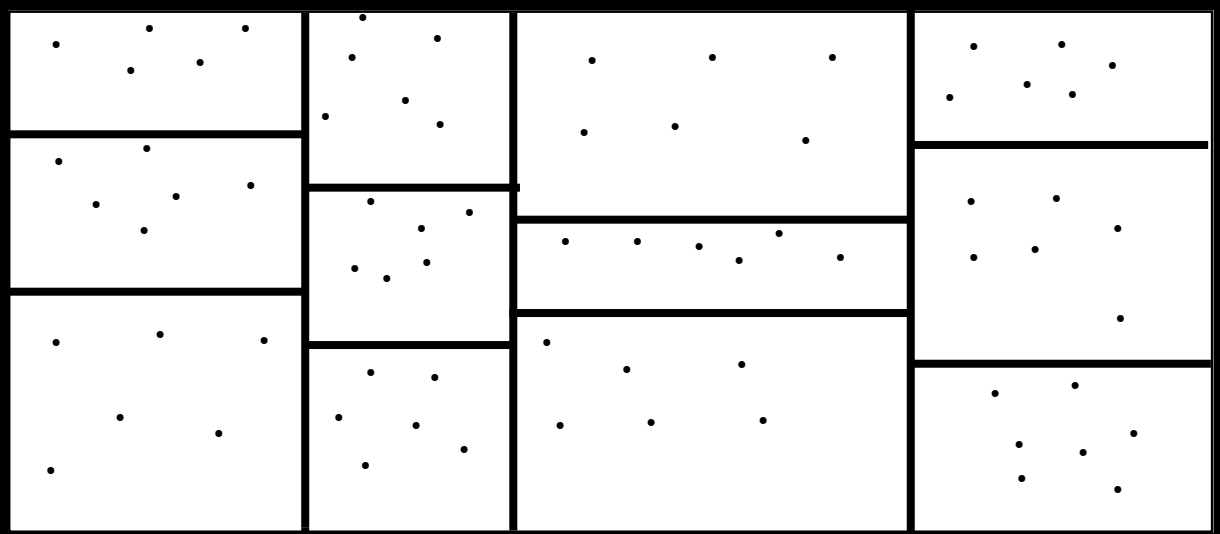


0,1,2,3 refer to processor number

Arrow indicates data movement



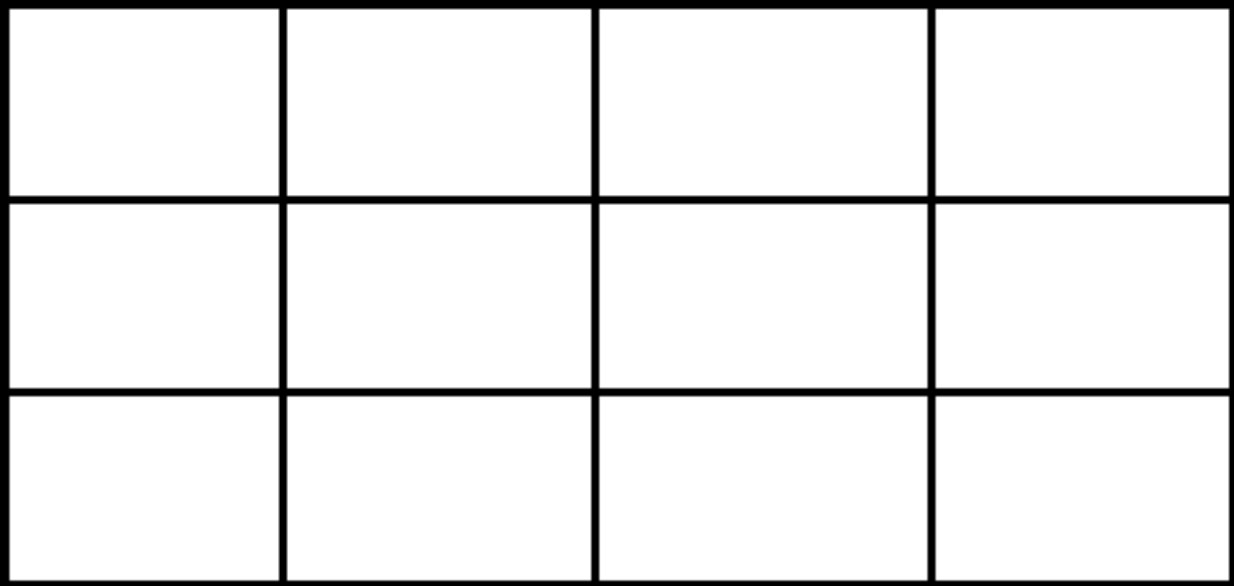
1D Domain decomposition



2D Domain decomposition

Each partition has equal number of particles

1D Domain decomposition



2D Domain decomposition

Each partition has equal number of grids

Transpose Routines

Many algorithms can be parallelized by operating on one co-ordinate which is local (not distributed), then transposing and operating on the other coordinate, which is now local

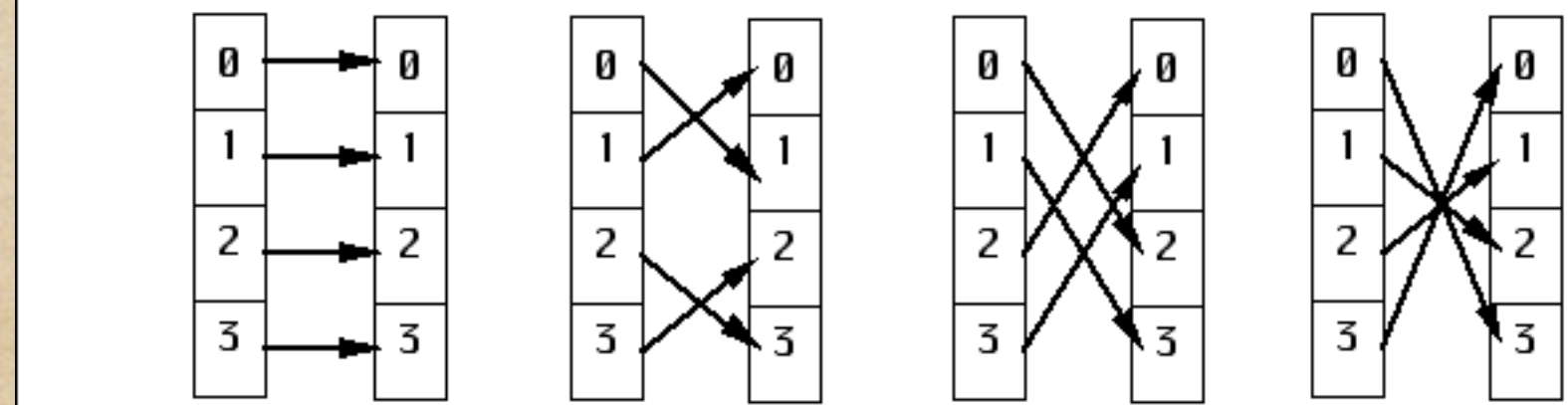
FFT can be parallelized this way
Also, mixed spectral and finite-difference methods

Limitations

- Cannot use more processors than grids
- Assumes domains are regular

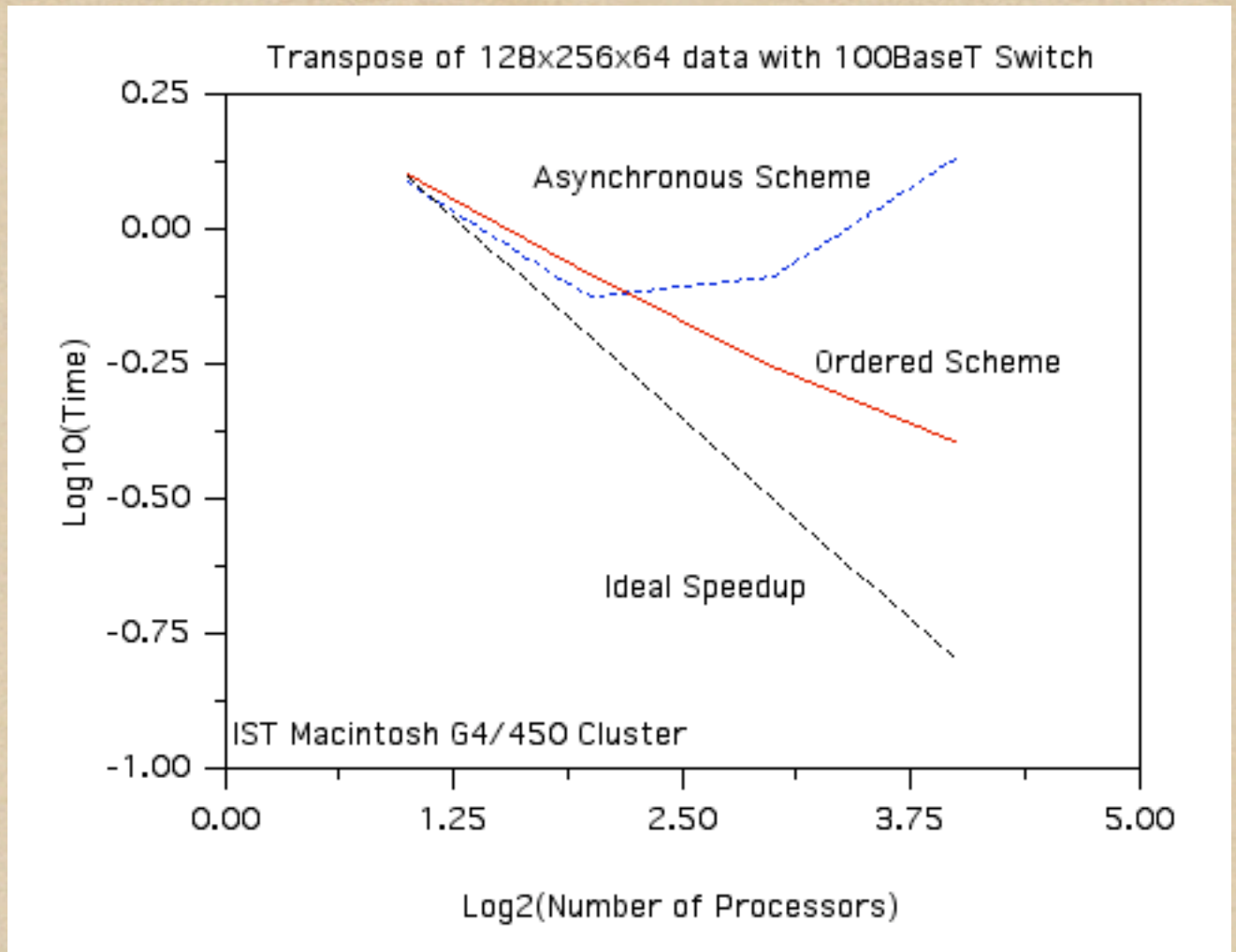
Transpose is very communication intensive, all-to-all

Collision avoidance important for some hardware

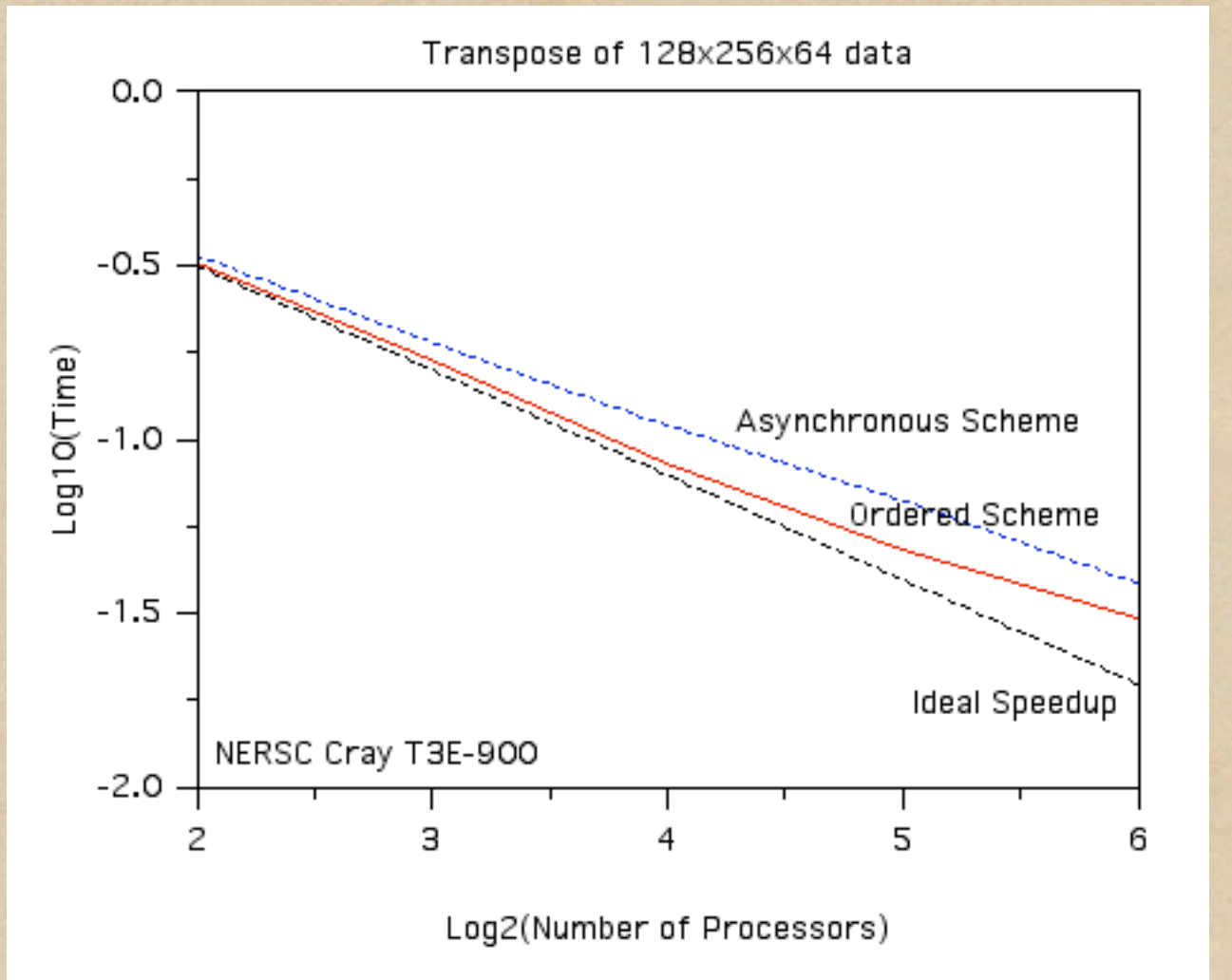


Controlled Data Transpose
Full Duplex, No collisions

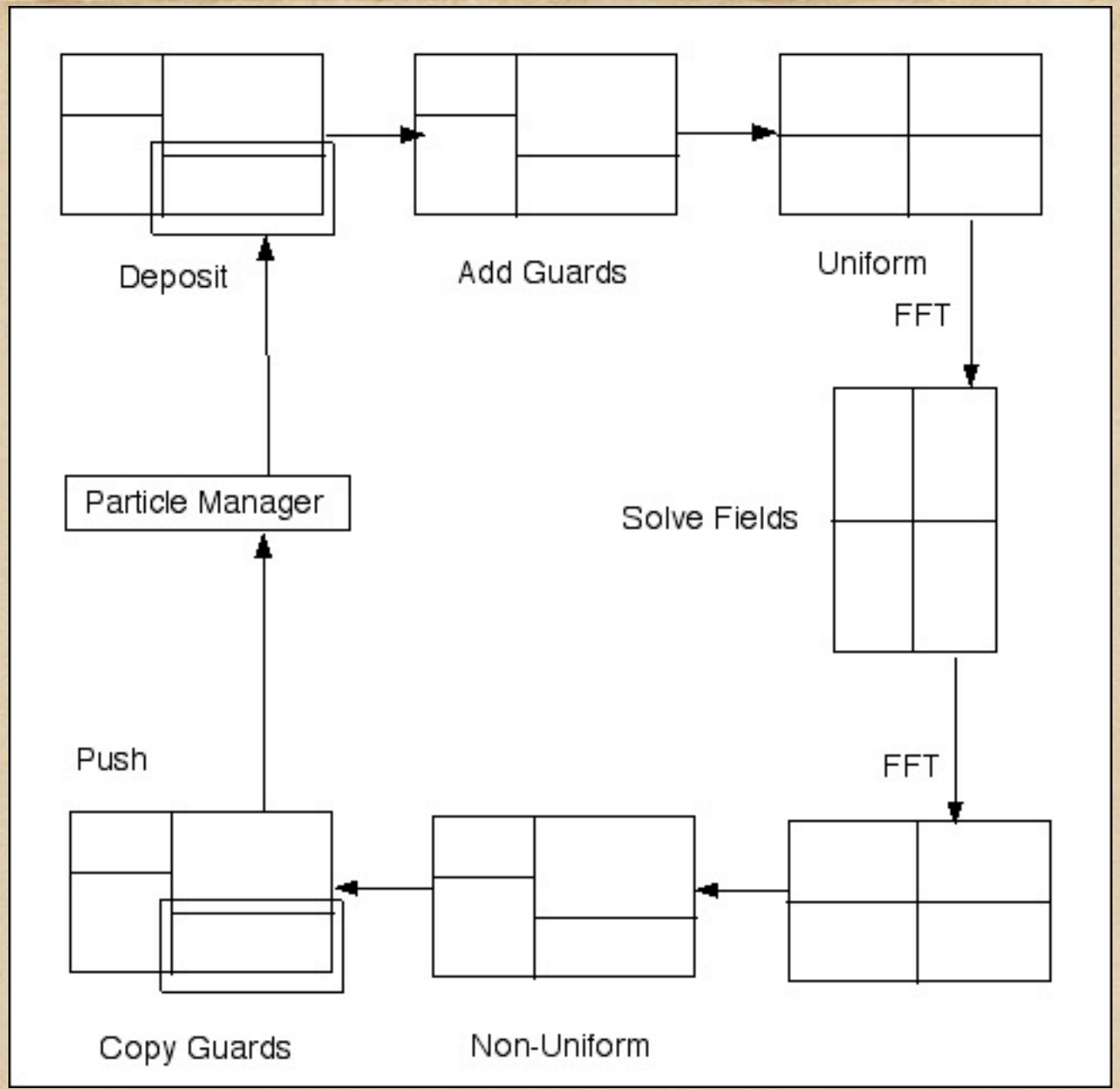
At any given pass, each process sends and receives to unique processor. Pause at each pass before continuing.



Ordered vs. Asynchronous Transpose on Commodity Switches



Ordered vs. Asynchronous Transpose on High Performance Computers



Field Data Distributions in Main iteration loop

PLIB: Parallel Particle Simulation Library

A small low-level library (30 subroutines) which encapsulate all the communication patterns needed by parallel PIC codes

- designed for high-performance
- hides low-level communication details
- supports both message-passing and shared memory
- supports RISC and vector architecture
- supports linear and quadratic interpolation

Used in several “Grand-Challenge” Calculations

- Numerical Tokamak Turbulence Project
- Space Simulation
- Accelerator design

Written in Fortran77: fast, can be called from many languages

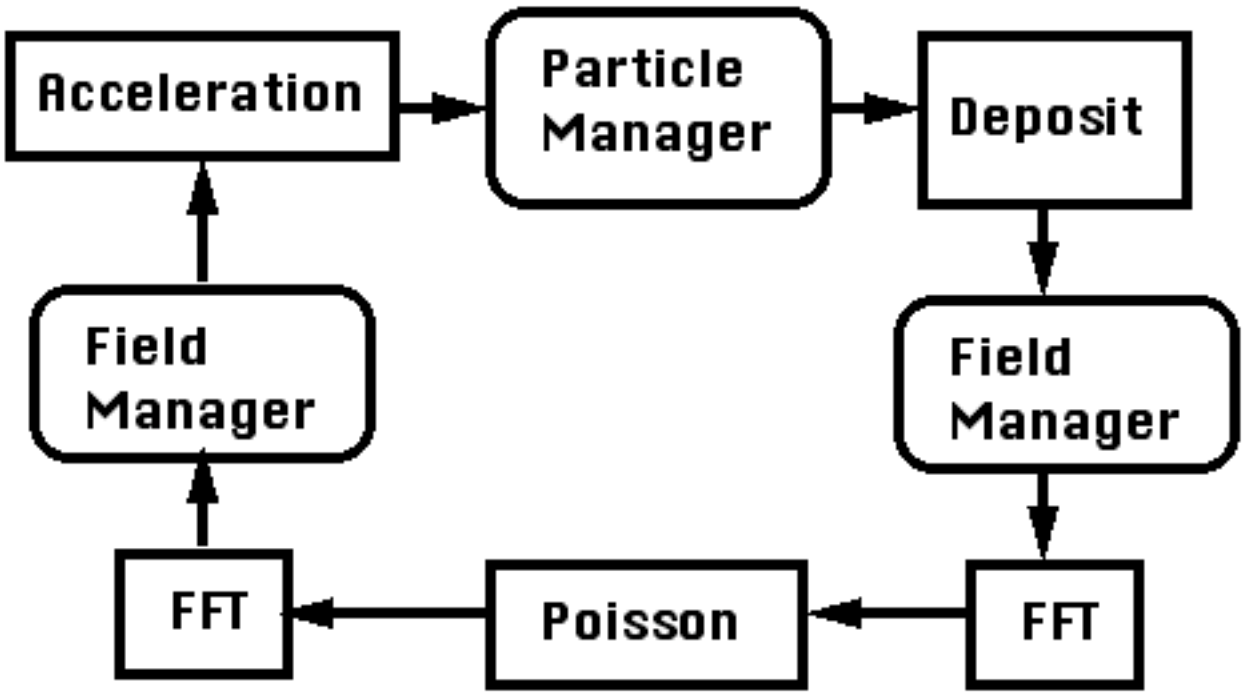
PLIB Philosophy

2D code supports 1D domain decomposition
3D code supports 1D and 2D decomposition

PLIB generally does only data movement, no physics
- Communications patterns can be reused

Physics modules only need to know current data layout
- Physicists can use PLIB without knowing about MPI

Multiple FFTs can be done with single transpose



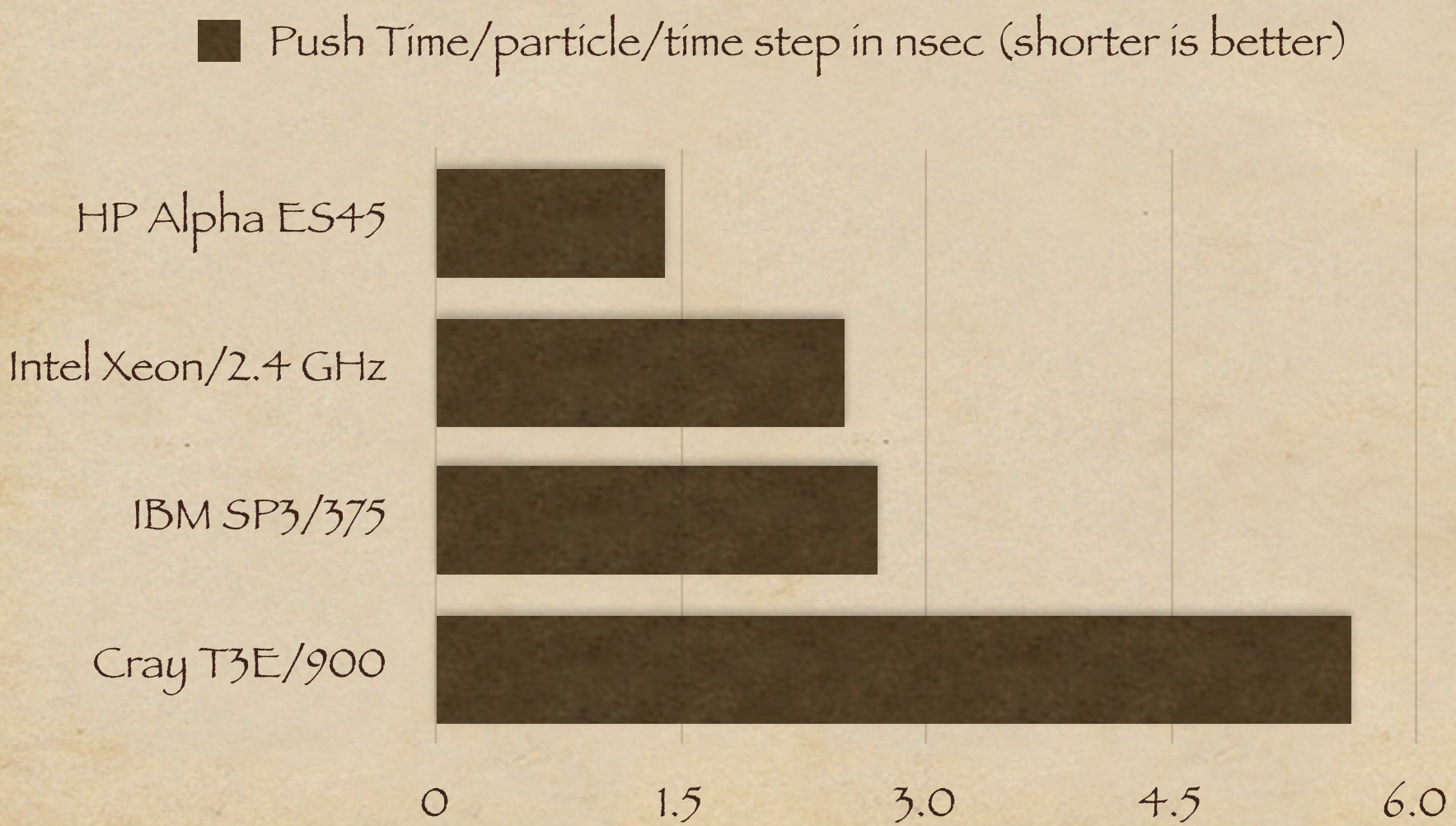
Main program loop

Rounded boxes: data layout routines contain MPI calls

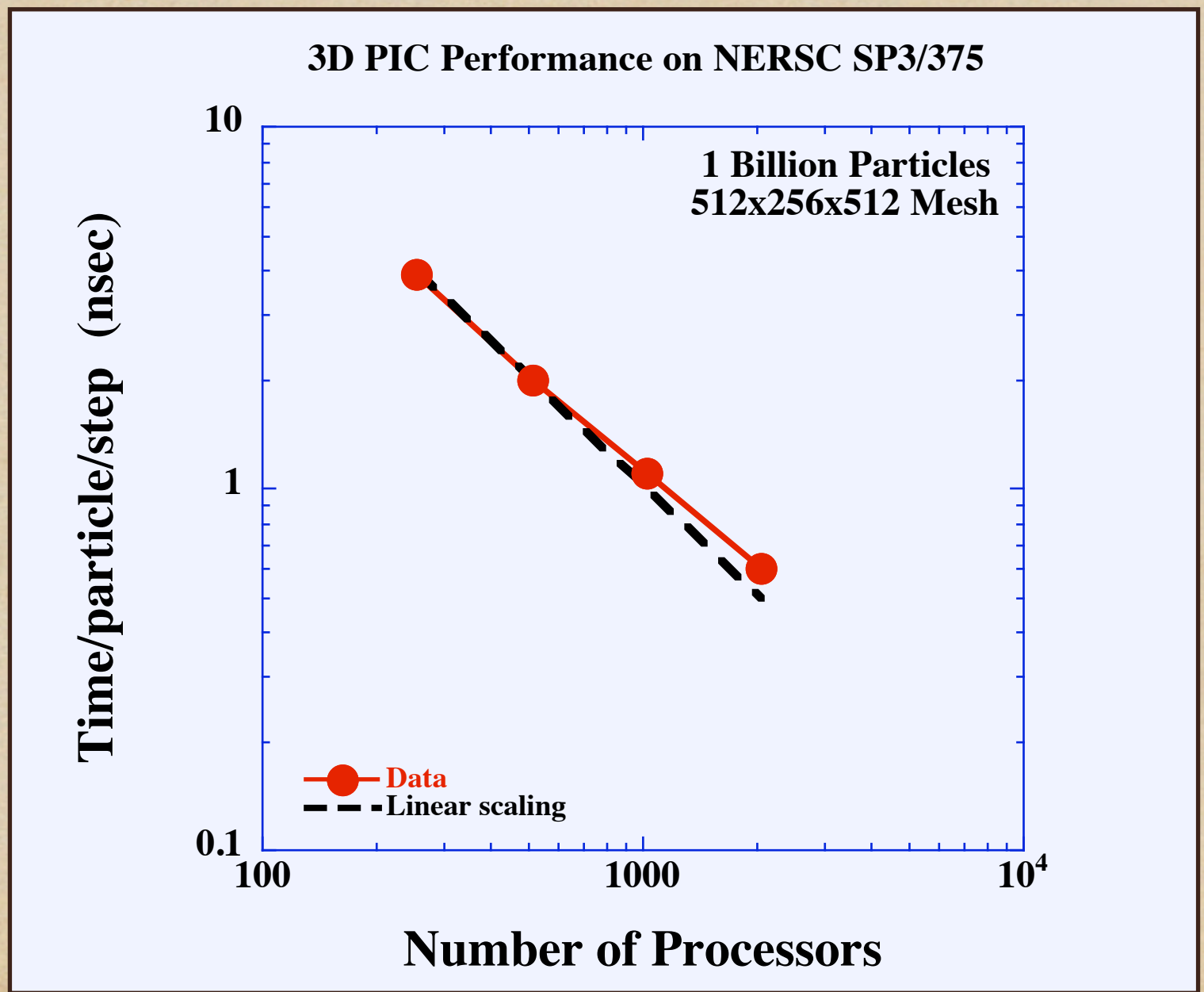
Square boxes: physics routines do not contain MPI

Communications are separated from physics

3D Electrostatic particle benchmark with 256 processors



Scaling of Large PIC Electrostatic Benchmark



AppleSeed Macintosh G5 Cluster, 8 cpus, 2004

UCLA Dawson cluster, Apple XServe G5, 512 cpus, 2004