# SUMMARY AND CONCLUSIONS

Two important considerations in considering different approaches:

- How easy is it to express the parallelism in an application?
  In Fortran M it is very easy to write parallel codes in which processes communicate via channels and mergers.

- How good is the performance?
  - ◇ So far we have only looked at the performance of the gather and scatter steps
  - ◇ Thread based implementation should improve performance
  - ◇ Important factor is how easy is it to control the granularity of communication. If processes are sufficiently light-weight and memory is cheap, we would like many small processes that buffer all their communication till the end of a processing step. Fortran M runtime system can manage overlap of communication and calculation.

# IMPROVEMENTS TO FORTRAN M

Current version of Fortran M uses Unix processes for which switching times are slow.

- Implementation based on light-weight threads is being developed.

This should significantly improve performance.

# OPTIMIZATION ISSUES

Currently communication is done at a fine granularity on a per grid point basis (inner loop). We expect better performance if the frequency of communication is reduced. Some possibilities:

- to communication on a per grid cell basis. Scatter all the data from a particle to to the root of its enclosing cell. Similarly with the gather. Reduces communication frequency by a factor of 8.
- buffer messages. Defer sending messages until a threshold is reached. Requires more memory.

# MULTIPROCESSOR RESULTS

We have held the number of particles and the grid points in each processor constant and varied the number of SP1 nodes.

**Run 1.** 8 processors, 32768 particles $16 \times 16 \times 16$ grid. $N = M = 8$, one subgroup and one subgrid per processor.
**Run 2.** 9 processors, 32768 particles $16 \times 16 \times 16$ grid. $N = 8, M = 1$, one subgroup on each of 8 processors, all of grid on the other.
**Run 3.** 8 processors, 32768 particles $16 \times 16 \times 16$ grid. $N = M = 4$, subgroups on 4 processors, and subgrids on other 4 processors.
**Run 4.** 8 processors, 4096 particles $8 \times 8 \times 8$ grid. $N = M = 8$, one subgroup and one subgrid per processor.

**Results:**

Run 1: 205 seconds

Run 2: 335 seconds

Run 3: 251 seconds

Run 4: 28 seconds

# PROCESS PLACEMENT

Fortran M allows control over the placement of processes on processors. Some possible schemes for process placement are:

1. Subgroup and subgrid processes share the $N_p$ processors.
   Subgroup $i$ on processor $\lfloor i/P \rfloor$, where $P = \lceil N/N_p \rceil$.
   Subgrid $i$ on processor $\lfloor i/Q \rfloor$, where $Q = \lceil M/N_p \rceil$.
2. Subgroup and subgrid processes are on different sets of processors.

Scheme 1 has the least load imbalance, but involves context switching between subgroup and subgrid processes.

Scheme 2 has load imbalance, but reduced switching.

# INITIAL RESULTS

We present results for runs on

- a single Sun 4 processor
- a single SP1 node
- multiple SP1 nodes

Two observations;

- using dynamic channels in the gather process slows down the code by a factor of about 20. In the current implementation of Fortran M it is expensive to create a channel and to migrate a port to another process.
- a simple optimization is to perform the scatter/gather routing within the particle subgroup and mesh subgrid processes. This reduces the overhead associated with creating and managing the router and forwarding processes.

Elapsed times for a small problem:

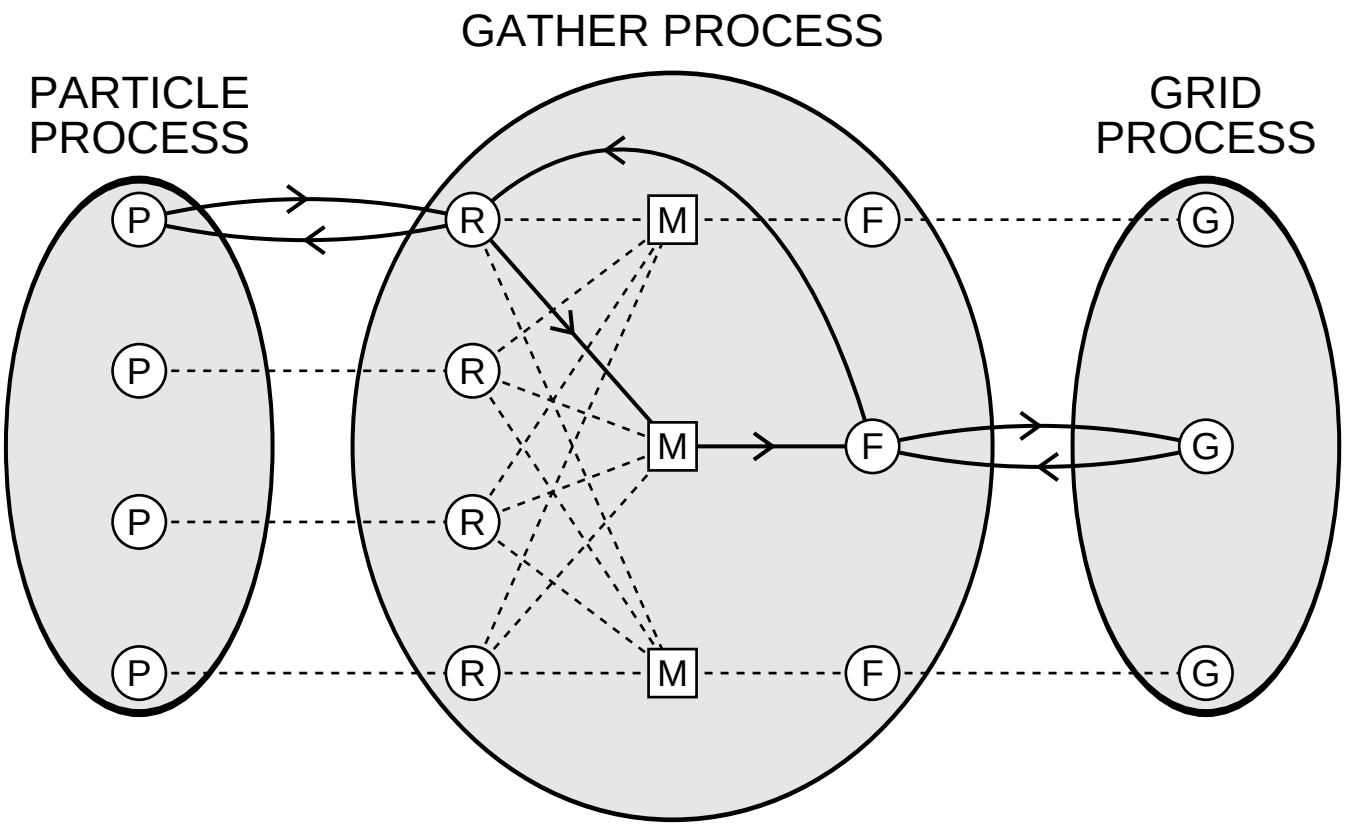
4096 particles, $8 \times 8 \times 8$ grid

Sun 4:193.6

SP1 node:111.2

# THE GATHER PHASE

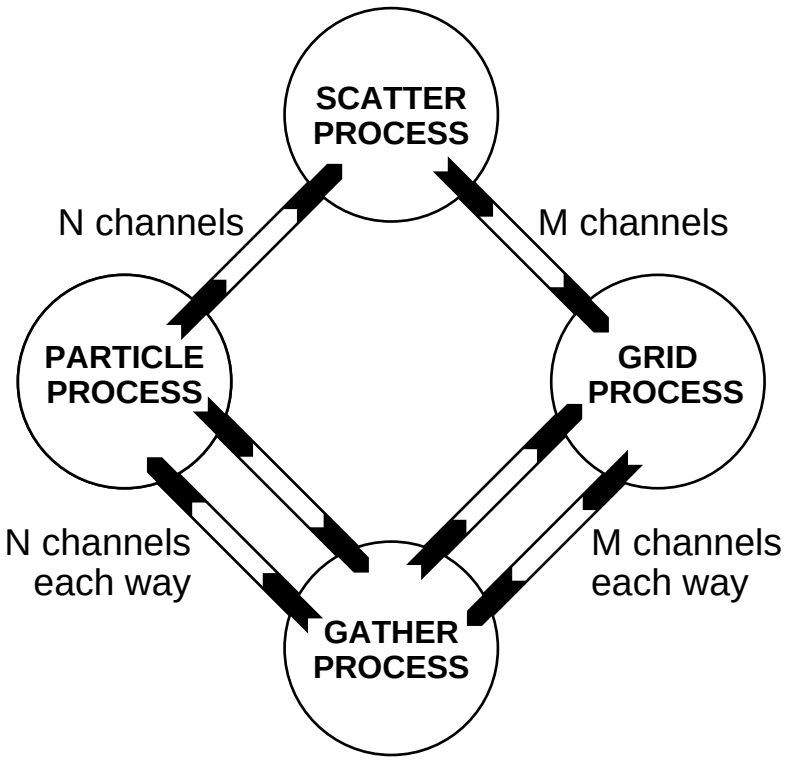
The gather phase makes use of Fortran M's dynamic channels capability to return EM field information to the particles.




1. Particle requests information from grid
2. Router process creates a channel to be used to return data
3. Router process routes the request and the channel outport to merger
4. Forward process passes on data request to subgrid, and receives back data
5. Forward process passes data back to router node via the dynamic channel
6. Router process passes data back to particle

# THE SECOND IMPLEMENTATION

- Scatter phase is the same as before (but we do not send or store particle global ID numbers).
- Gather phase uses a two-phase protocol.



# THE GATHER PROCESS

In our first implementation the gather phase is done by scattering from the subgrid processes to the particles. Thus the gather process is similar in structure to the scatter process.

**Consequences:**

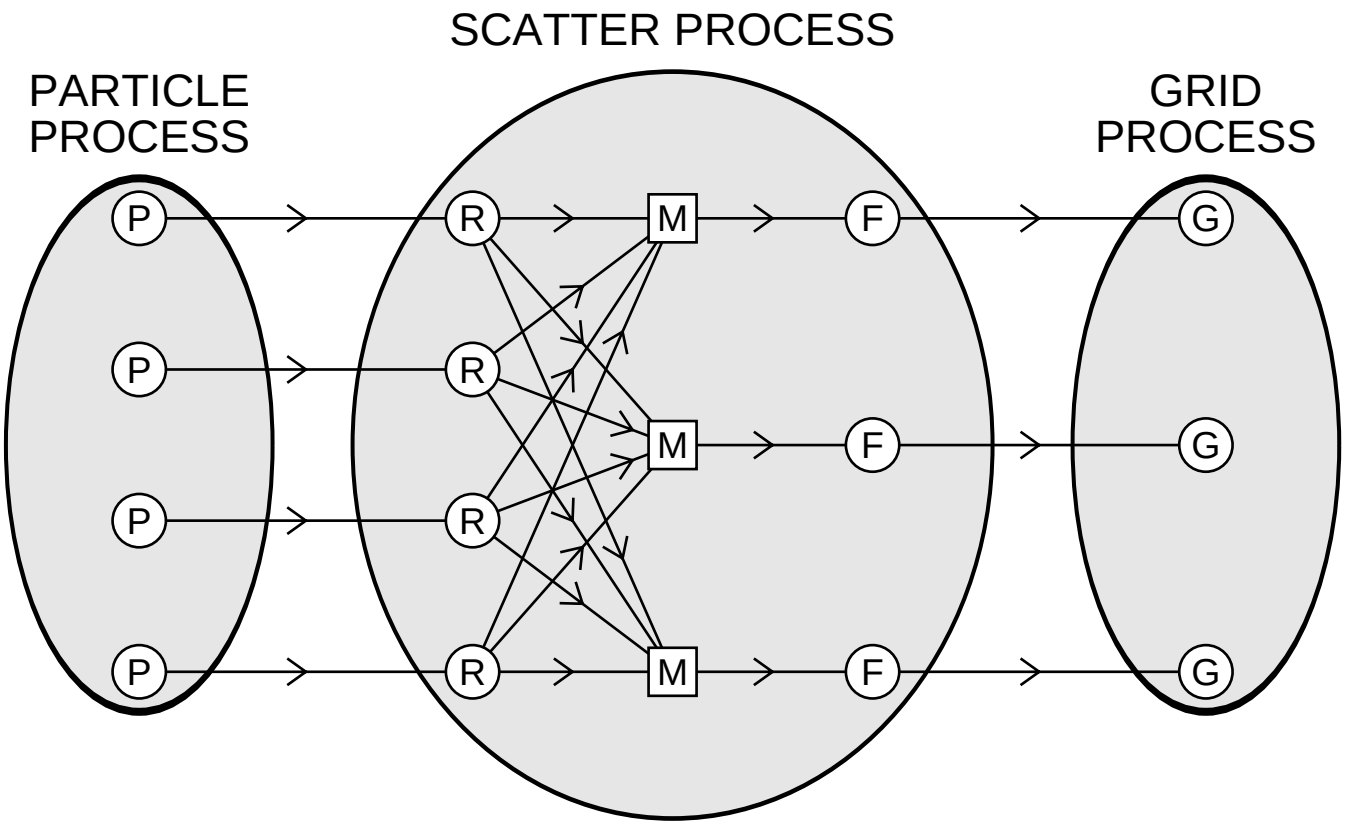
- Subgrid processes have to know which particles they must send data to.
- Particles send their global ID number along with the current density and grid point coordinates in the scatter phase.
- Subgrid processes use these ID numbers as "return addresses" in the gather phase.
- This leads to memory imbalance.

# THE SCATTER PROCESS

The scatter process has the following child processes

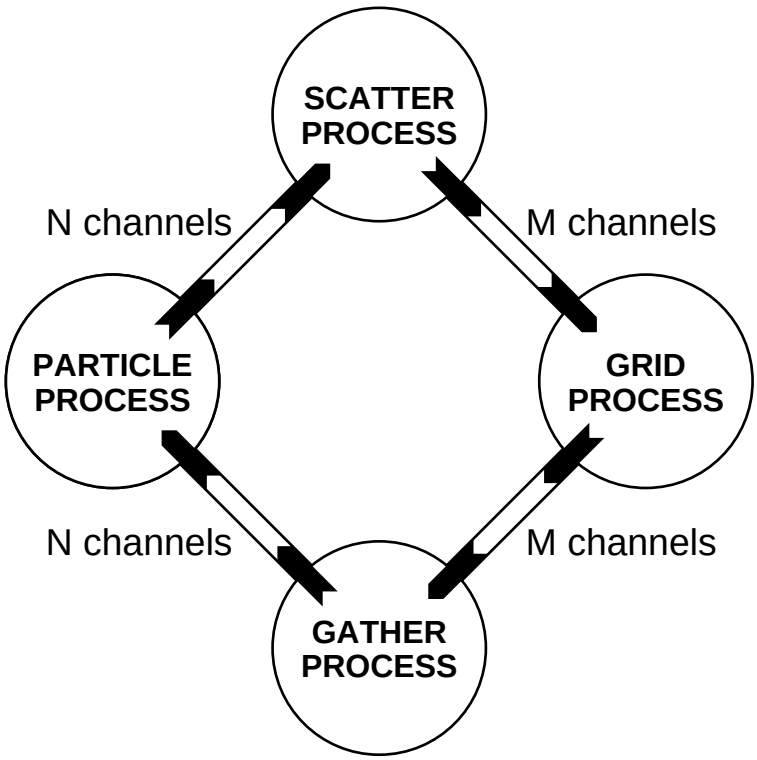
- $N$ router processes, and
- $M$ forwarding processes.

Routing is performed by a set of $M$ mergers. A merger is a Fortran M construct that connects multiple outports to a single inport.



# IMPLEMENTATION 1

At the highest level our first implemenation consists of 4 processes connected by channels.




- Particles are decomposed into $N$ subgroups
- Grid is decomposed into $M$ subgrids
- Scatter process routes current density from particles to grid
- Gather process routes EM field data from grid to particles

Each subgroup and subgrid is managed by a separate process.

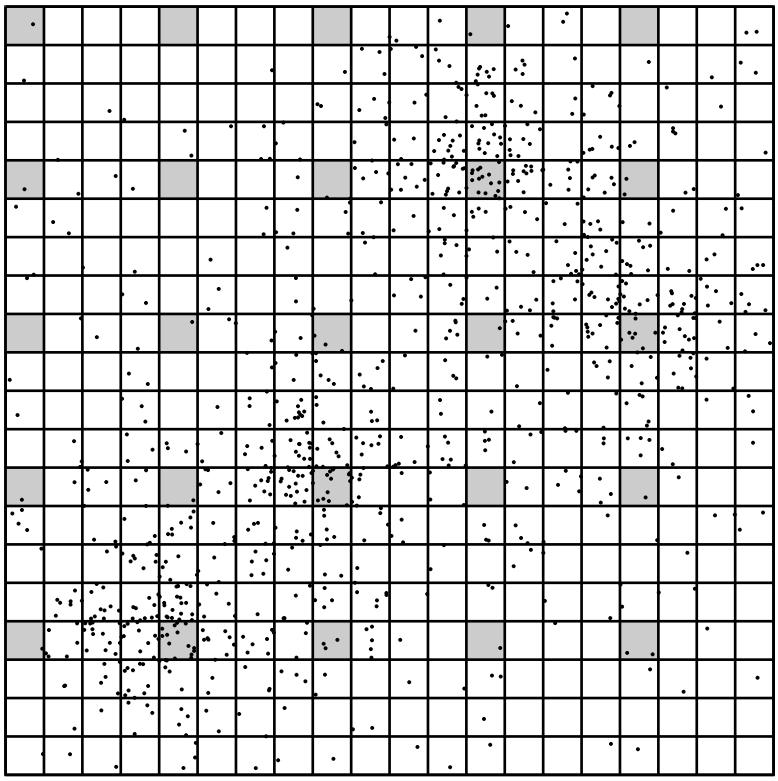
# FORTRAN M

Work of Foster (Argonne) and Chandy (Caltech).

- Extensions to Fortran 77 to support modular message passing programs
- Channels are used to plug together modules called processes
- A process encapsulates common data, subprocesses, and internal communication channels.
- Channels are typed.
- Message passing over channels is by means of nonblocking SEND and blocking RECEIVE routines.

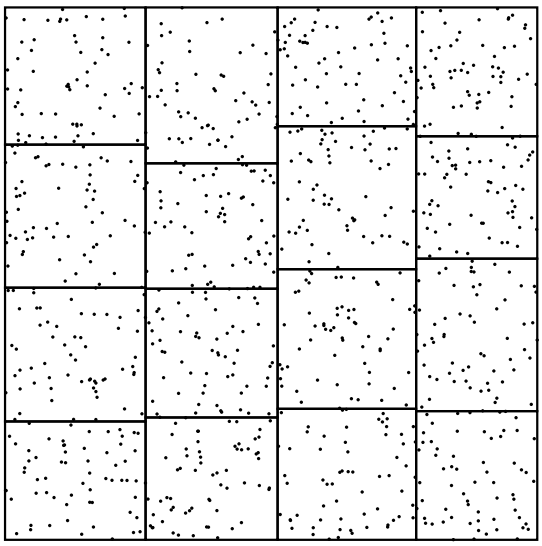
# SCATTERED DECOMPOSITION

Used in tightly coupled case.

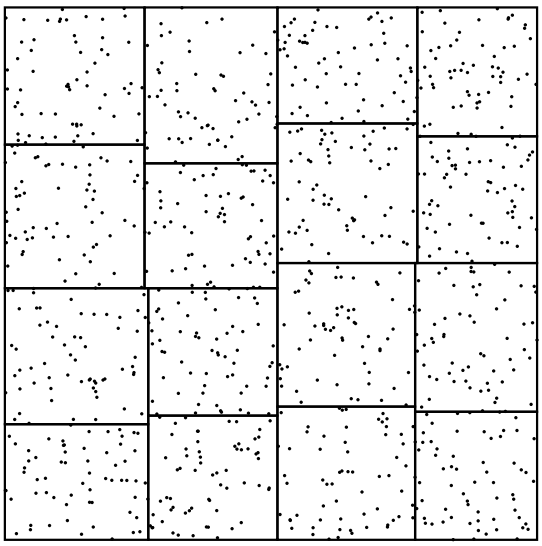


- gives statistical load balance
- increases surface area to volume ratio, so more communication is needed in gather/scatter steps than in block distributed case.
- also increases amount of particle migration.

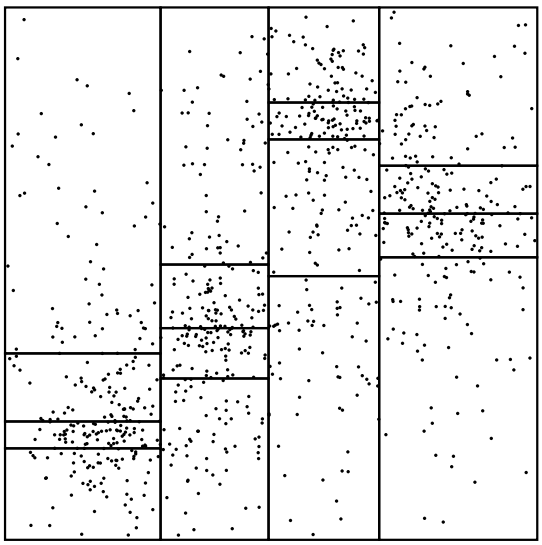
# DYNAMIC LOAD BALANCE



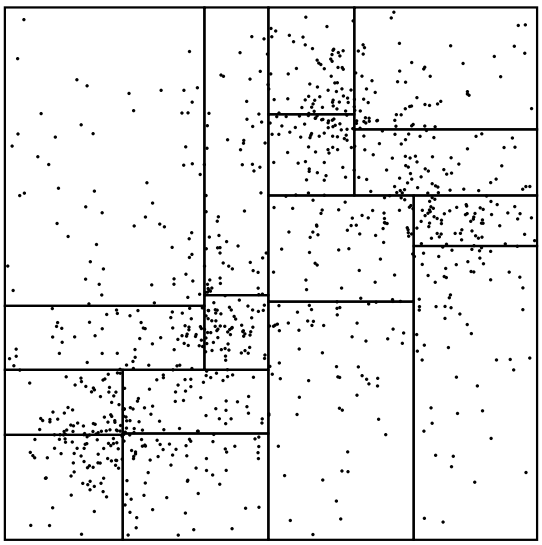
(a)



(b)

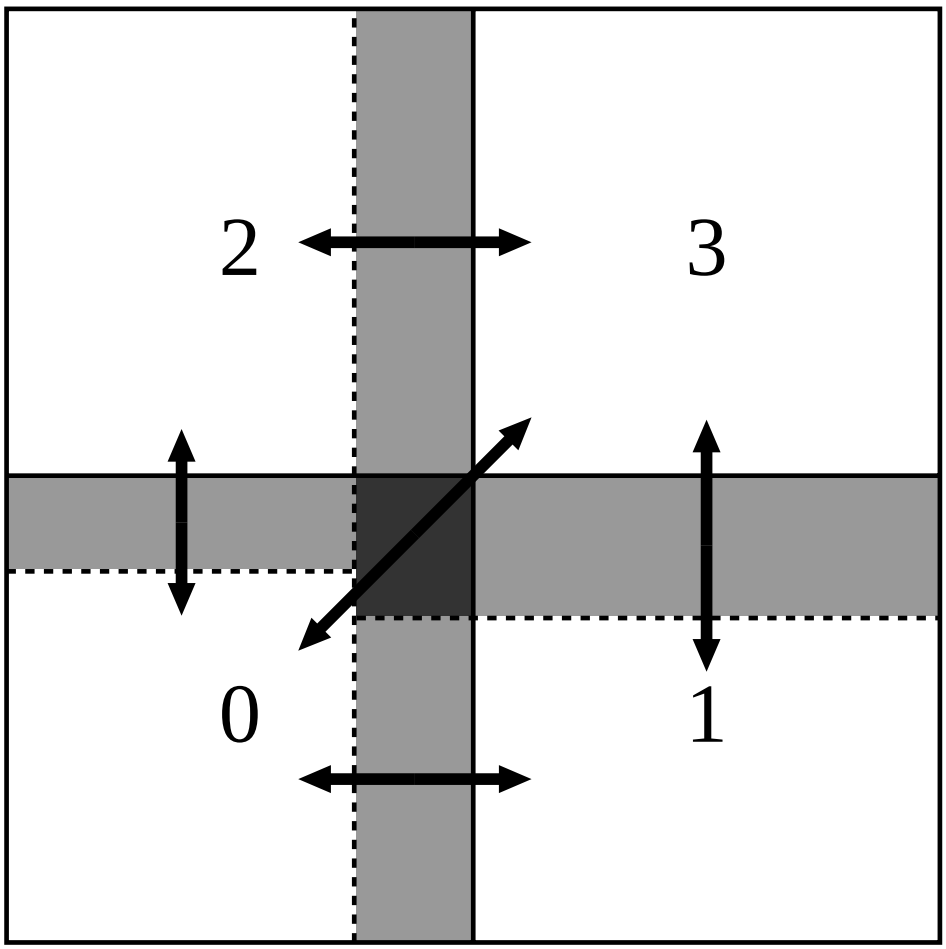


(c)



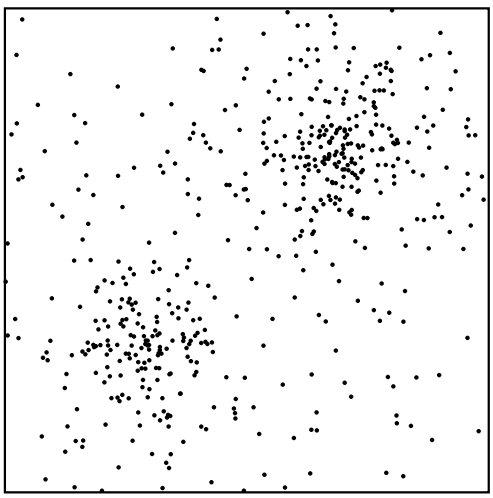
(d)

# LOOSELY COUPLED COMMUNICATION

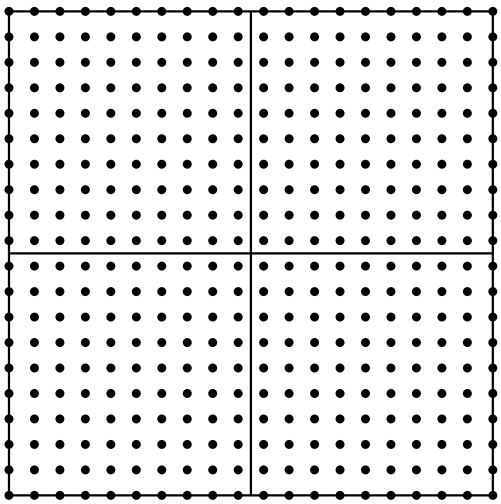


# LOOSELY COUPLED DECOMPOSITION

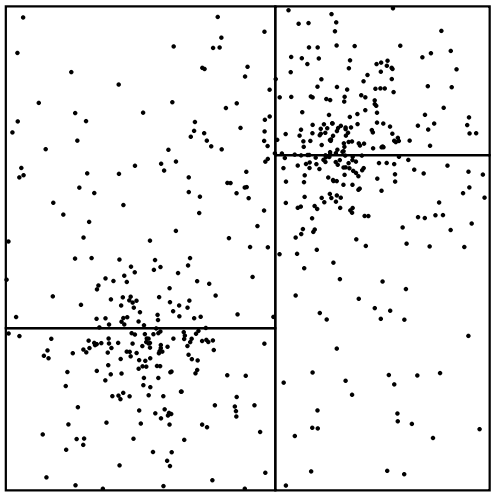
Particles and grid have different spatial decompositions.



Problem Domain



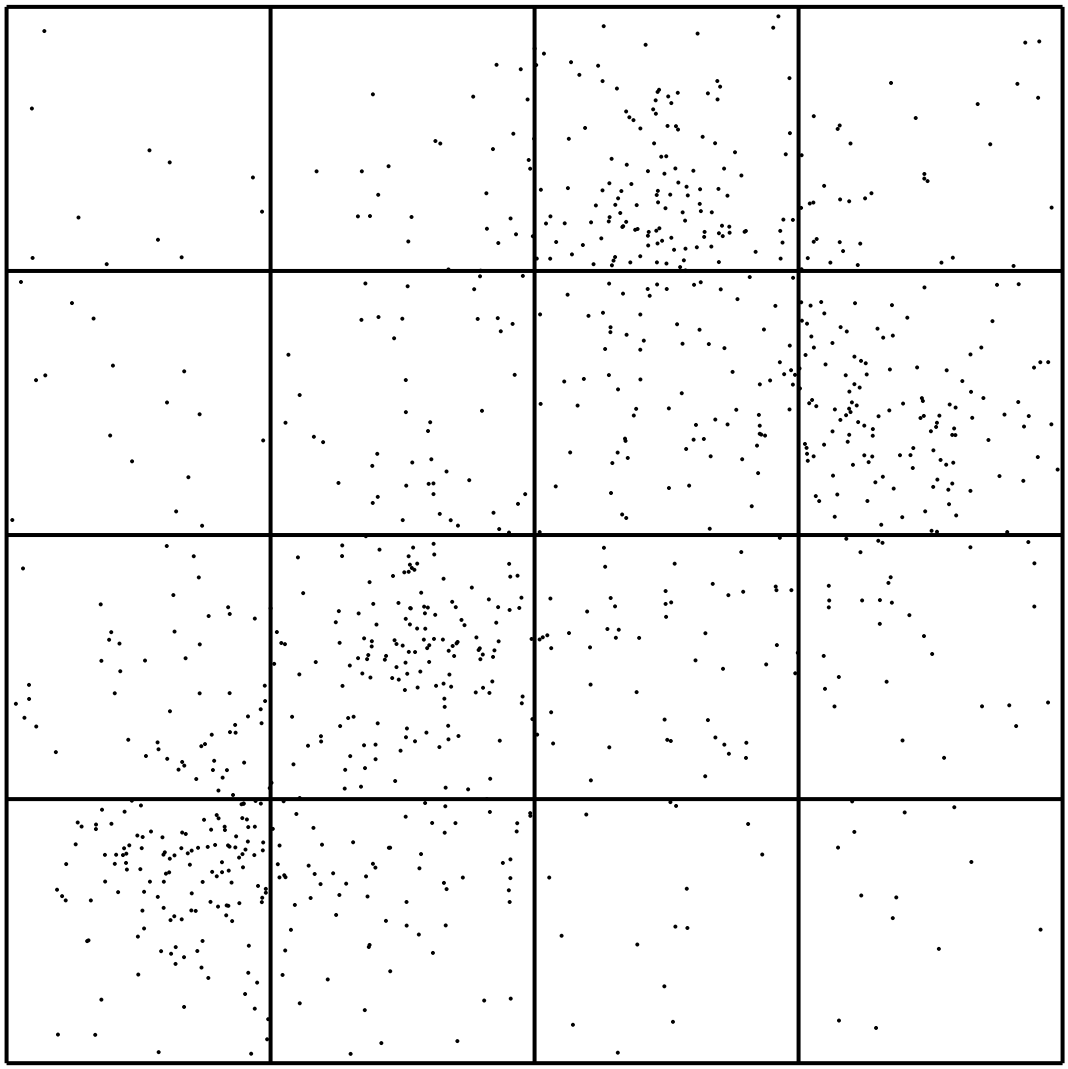
Mesh Decomposition



Particle Decomposition

# TIGHTLY COUPLED DECOMPOSITION

Regular spatial decomposition for grid, and particles have same decompostion.



# DATA DISTRIBUTION IN PIC

Load balance and communication characteristics depend on degree of coupling between data distributions of particles and grid

1. **Uncoupled**.
   - Spatial decomposition for grid
   - Non-spatial decompositions for particles

   Good load balance, non-local scatter/gather.
2. **Tightly Coupled**.
   - Grid and particles have same spatial decomposition

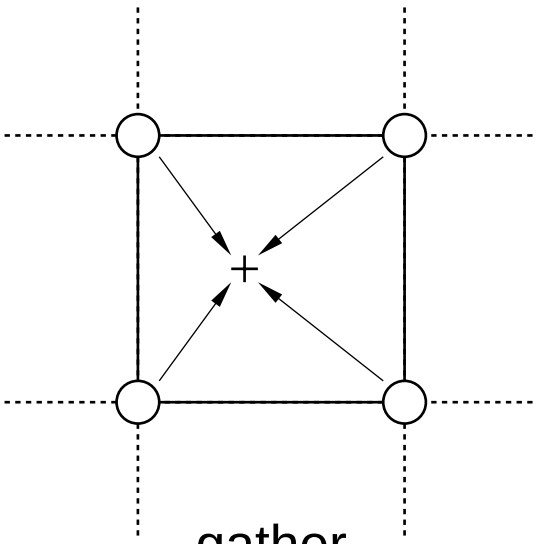
   Poor load balance for inhomogeneous particle distributions, local scatter/gather, requires particle migration.
3. **Loosely Coupled**.
   - Spatial decomposition for grid
   - Different spatial decomposition for particles.

   Scatter/gather is "mostly local," poor load balance, requires particle migration

The uncoupled case is the simplest to code as it avoids dynamic load balancing and particle migration.

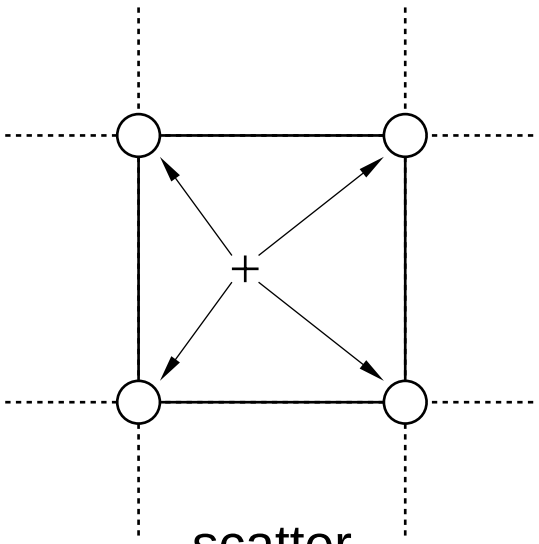
# SCATTER AND GATHER OPERATIONS

Find $\mathbf{E}(\mathbf{x}_i)$ and $\mathbf{B}(\mathbf{x}_i)$



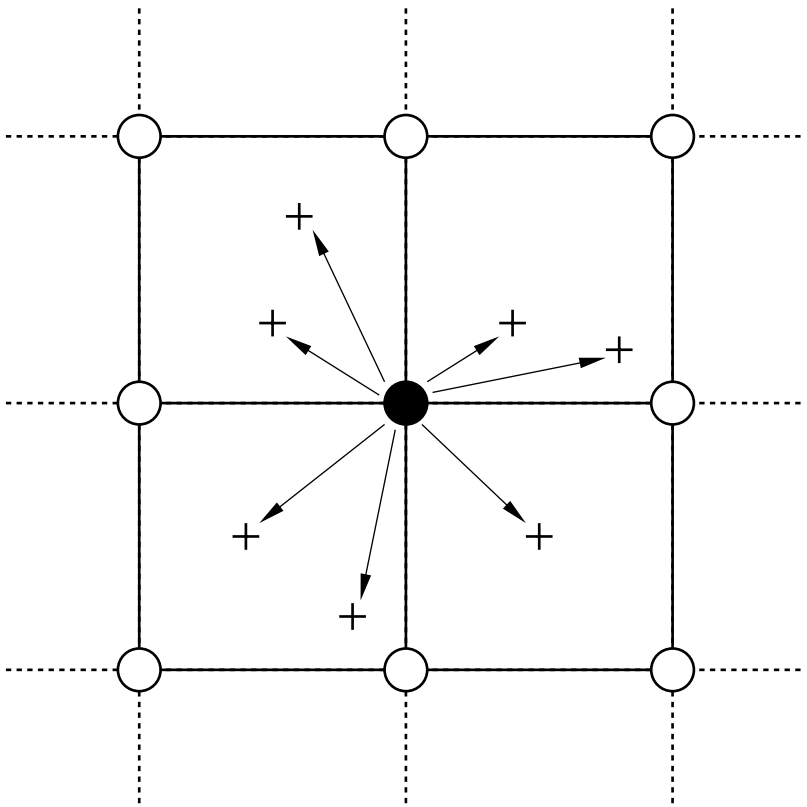
gather

Find $\mathbf{j}_k$

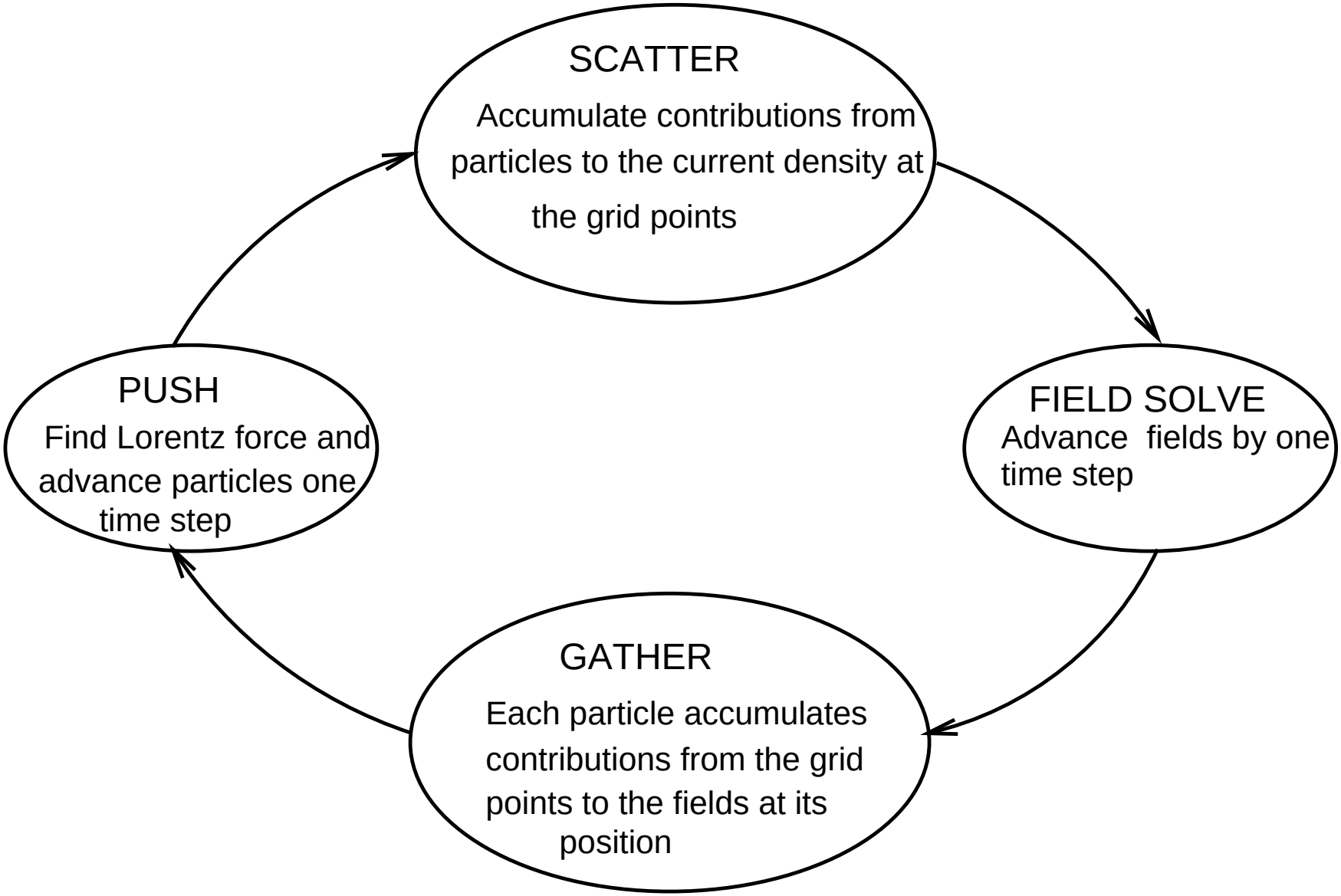


scatter

Find $\mathbf{E}(\mathbf{x}_i)$ and $\mathbf{B}(\mathbf{x}_i)$



scatter


SCATTER
Accumulate contributions from particles to the current density at the grid points
FIELD SOLVE
Advance fields by one time step
GATHER
Each particle accumulates contributions from the grid points to the fields at its position
PUSH
Find Lorentz force and advance particles one time step

# THE PARTICLE-IN-CELL ALGORITHM

- The PIC method self-consistently simulates the evolution of set of particles interacting via a long-range potential. External fields may also be present.
- The PIC method uses two computational objects:

(1) A set of particles, with each of which is associated a mass and other quantities such as charge.

(2) A numerical grid on which field variables such as mass density (for gravitational problems), or electric and magnetic fields (for plasma simulations) are defined.

# RESEARCH PLAN

Evaluate different approaches to exploiting parallelism in computational science using,

- a few example applications, and
- a few interesting programming methodologies.

Important factors,

- ease of programming
- performance
- portability
- generality
- modularity
- maintainability

Usually we evaluate applications against machines. We want to add a third axis of "methodologies".

# RESEARCH OBJECTIVES

**Primary objective:**

To investigate programmability and performance trade-offs of different approaches to exploiting parallelism.

**Rationale:**

- researchers would like to use distributed memory computers, but are put off by perception that they are "hard to program"
- they would like
  - languages, abstractions and mechanisms that hide details of parallelism
  - reusable software
  - tools to aid in parallelization of code
- they do not want to sacrifice (too much) performance
- many useful languages, abstractions, and mechanisms exist, but they are often ignored by mainstream computational science

# OVERVIEW

- Objectives of research
- Research plan
- Particle-In-Cell Application
- Fundamentals of Fortran M
- Implementation of FM/PIC codes
- Results
- Conclusions

**WWW pointer:**

http://www.mcs.anl.gov/fortran-m/FM.html

# Paradigms and Strategies for Scientific Computing on Distributed Memory Concurrent Computers

*David W. Walker*
*Oak Ridge National Laboratory*

*Ian T. Foster*
*Argonne National Laboratory*

Presented at:
High Performance Computing '94
La Jolla, California

April 11, 1994