

Parallel Programming in a Nutshell

load balancing vs communication

This is the eternal problem in parallel computing. The basic approaches to this problem include:

- data partitioning - moving different parts of the data set across several nodes
- task partitioning - give separate tasks to different nodes

Problem Taxonomy

Problems in parallel computing can be described in terms of the geometry of the problem. Three main areas need to be considered to understand the complexity of the problem when it is to be implemented in parallel:

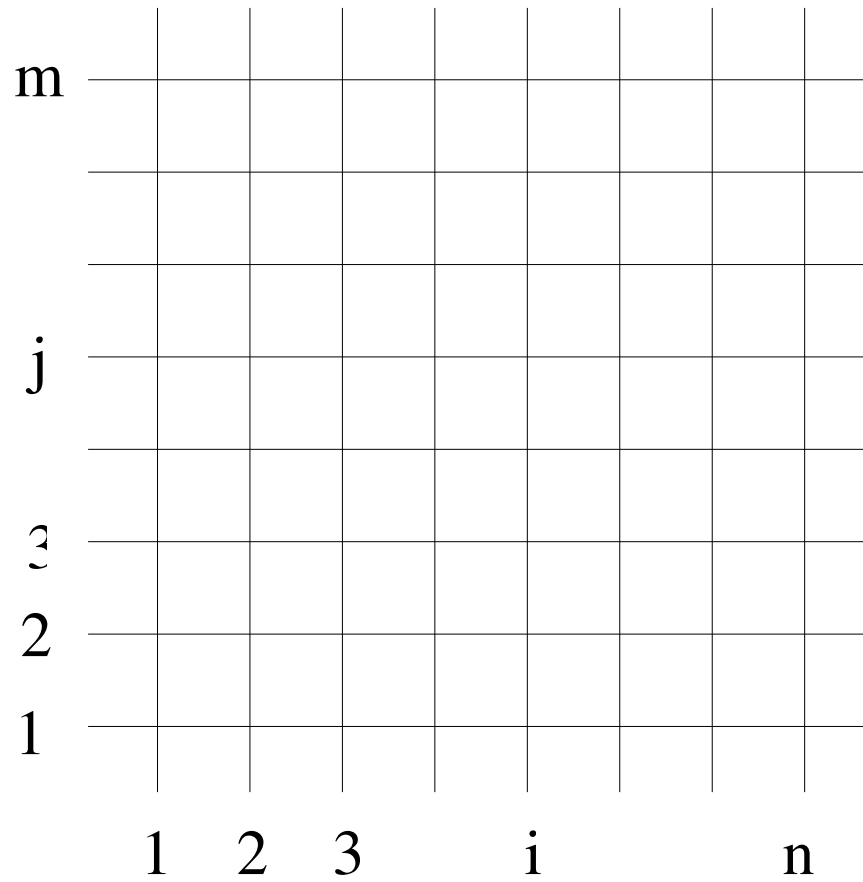
- **structured vs unstructured** - is the computational grid regular or irregular?
- **explicit vs implicit** - does the computation involve only nearest neighbors or elements across the grid?
- **static vs dynamic** - does the computational grid change between iterations?

Embarrassingly Parallel

Finally, some problems are considered to be “embarrassingly parallel”. This means parts of the problem can be run independently of each other.

Generally, the simpler problems parallelize more efficiently. However, it is possible to build high performance codes for complex problems.

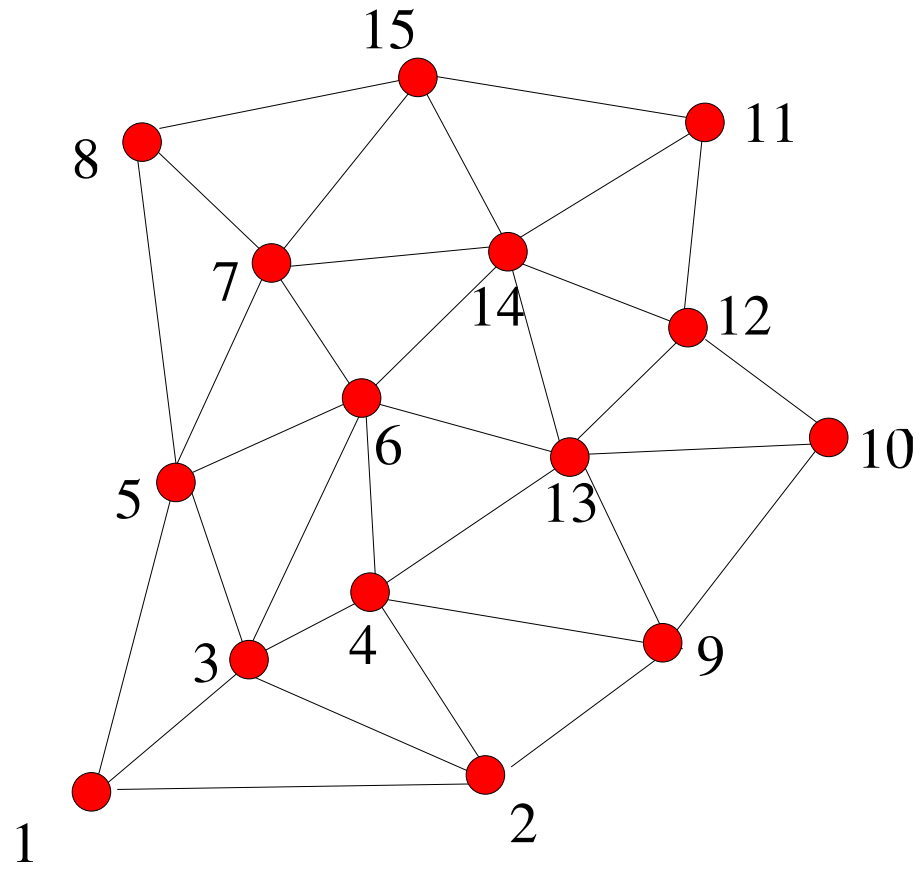
Structured Grids



Structured Grids

- All Neighbors can be identified by array location
- Simple geometric relationships define grid size
- Simple indexing scheme defines location
- Closeness of neighbors is uniform or easy to calculate

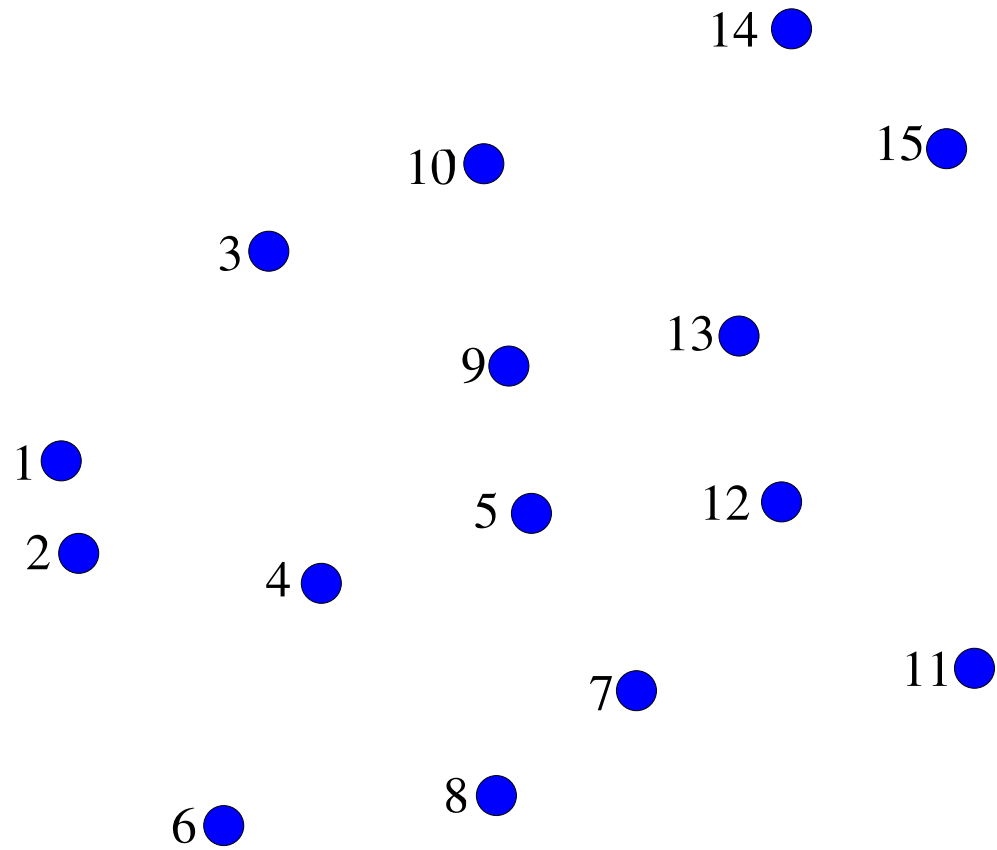
Unstructured Grids



Unstructured Grids

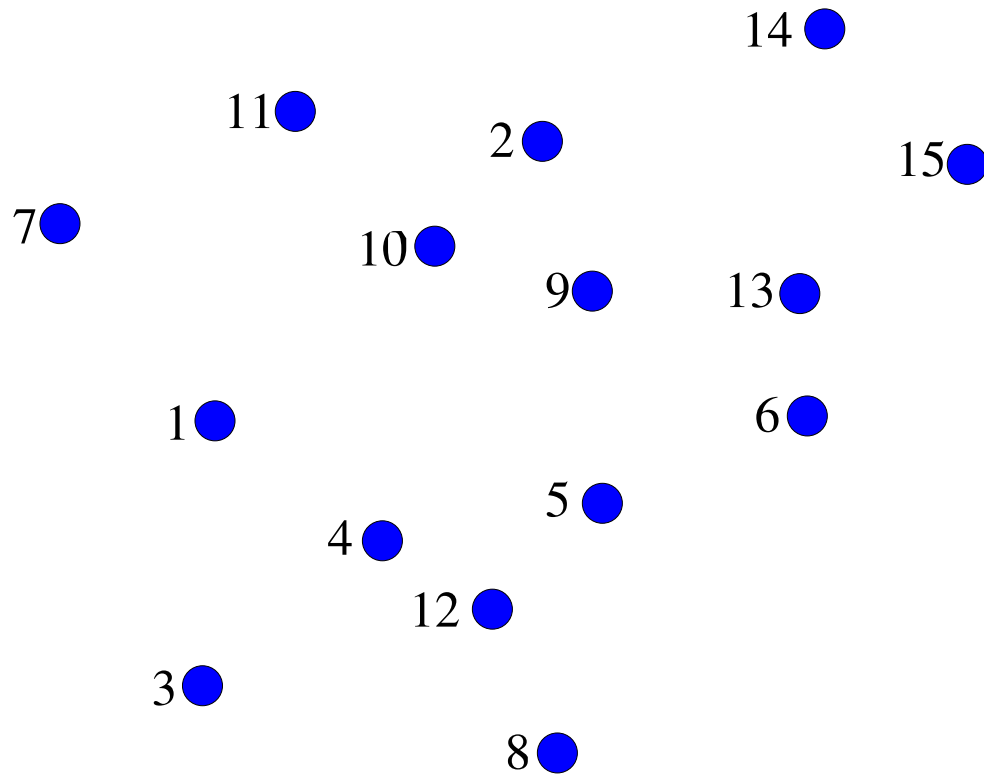
- Neighbors must be identified through a list
- Number of neighbors varies
- Node numbering does not represent physical location in a simple way
- Closeness of neighbors varies between pairs

Dynamic “Grid” Systems



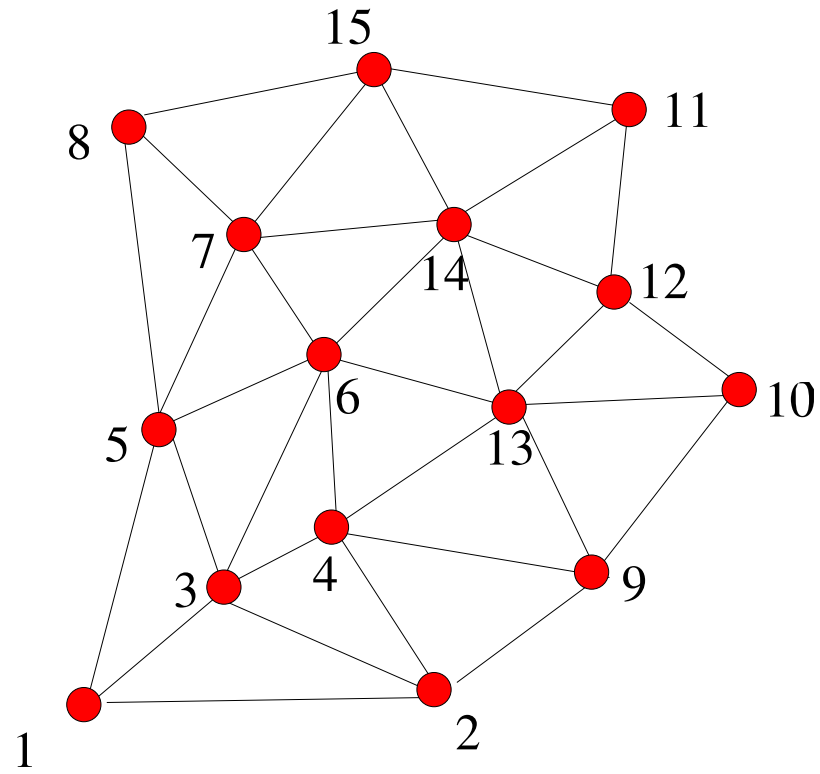
Initial Positions of Particles

Dynamic “Grid” Systems



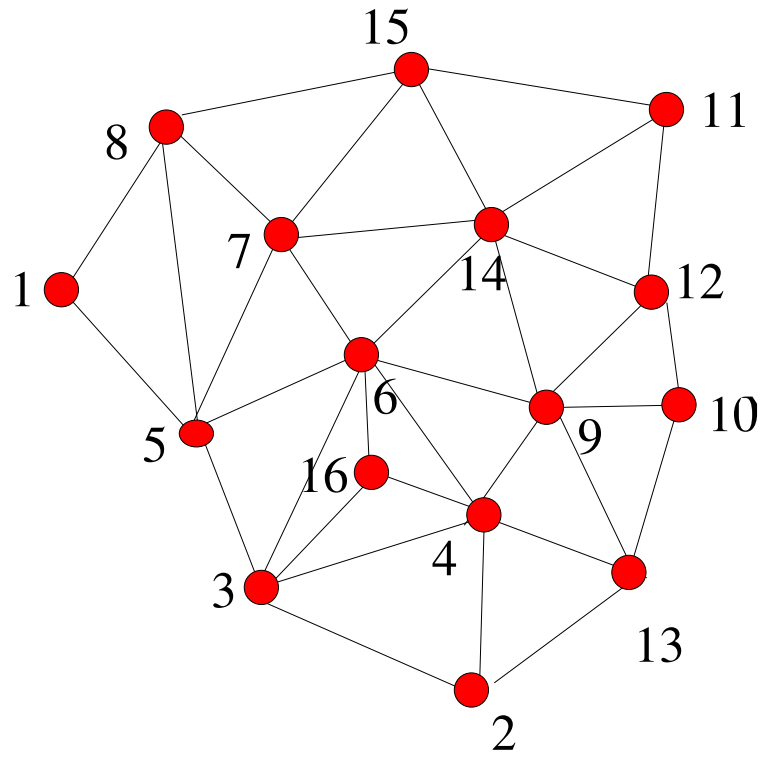
Later Positions of Particles - relative locations have greatly changed

Dynamic Grids



Timestep 1

Dynamic Grids



Timestep 2

Dynamic Grids

- New nodes are added
- Old nodes are deleted
- Connections between nodes change
- Connection strengths between nodes change

There is no a priori way to know the connections or how they will change as the system evolves.

The Heat Equation

- Analytic

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

- Explicit

$$\frac{1}{k}[u(x_i, t^{j+1}) - u(x_i, t^j)] =$$
$$\frac{1}{h^2}[u(x_{i+1}, t^j) - 2u(x_i, t^j) + u(x_{i-1}, t^j)]$$

- Implicit

$$\frac{1}{k}[u(x_i, t^{j+1}) - u(x_i, t^j)] =$$
$$\frac{1}{h^2}[u(x_{i+1}, t^{j+1}) - 2u(x_i, t^{j+1}) + u(x_{i-1}, t^{j+1})]$$

Matrix form of Explicit Equations

$$\begin{bmatrix} u_1^{j+1} \\ u_2^{j+1} \\ u_3^{j+1} \\ \vdots \\ \vdots \\ \vdots \\ u_n^{j+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ b_1 & b & b_1 & 0 & 0 & \cdots & 0 \\ 0 & b_1 & b & b_1 & 0 & \cdots & 0 \\ \vdots & 0 & \cdots & \cdots & \cdots & \vdots & 0 \\ \vdots & \vdots & 0 & \cdots & \cdots & \vdots & 0 \\ \vdots & \vdots & \vdots & 0 & b_1 & b & b_1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ u_3^j \\ \vdots \\ \vdots \\ \vdots \\ u_n^j \end{bmatrix}$$

where $b = (1 - 2\frac{k}{h^2})$ and $b_1 = \frac{k}{h^2}$

Solving Implicit Systems

$$\frac{1}{k}[u_i^{j+1} - u_i^j] = \frac{1}{h^2}[u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}]$$

How do you solve for u_i^{j+1} when it depends on u_{i-1}^{j+1} and u_{i+1}^{j+1} ?

The coefficients are easier to understand if we rewrite our equation as

$$u_i^{j+1} - \frac{k}{h^2}[u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}] = u_i^j$$

$$-\frac{k}{h^2}u_{i+1}^{j+1} + (1 + 2\frac{k}{h^2})u_i^{j+1} - \frac{k}{h^2}u_{i-1}^{j+1} = u_i^j$$

Matrix form of Implicit Equations

$$\begin{bmatrix}
 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\
 a_1 & a & a_1 & 0 & 0 & \cdots & 0 \\
 0 & a_1 & a & a_1 & 0 & \cdots & 0 \\
 \vdots & 0 & \cdots & \cdots & \cdots & \vdots & 0 \\
 \vdots & \vdots & 0 & \cdots & \cdots & \vdots & 0 \\
 \vdots & \vdots & \vdots & 0 & a_1 & a & a_1 \\
 0 & 0 & 0 & \cdots & 0 & 0 & 1
 \end{bmatrix}
 \begin{bmatrix}
 u_1^{j+1} \\
 u_2^{j+1} \\
 u_3^{j+1} \\
 \vdots \\
 \vdots \\
 \vdots \\
 u_n^{j+1}
 \end{bmatrix}
 =
 \begin{bmatrix}
 u_1^j \\
 u_2^j \\
 u_3^j \\
 \vdots \\
 \vdots \\
 \vdots \\
 u_n^j
 \end{bmatrix}$$

where $a = (1 + 2\frac{k}{h^2})$ and $a_1 = -\frac{k}{h^2}$

Structured - Explicit - Static

A structured, regular grid that doesn't change between time-steps with simple neighbor to neighbor communications.

Examples-

- A diffusion problem solved on a regular rectangular grid with static connections
- Euler equation solution to fluids on regular grid

Unstructured - Explicit - Static

A unstructured grid that doesn't change between time-steps with simple neighbor to neighbor communications.

Examples-

- A diffusion problem solved using finite elements
- Euler equation solution to fluids solved with finite elements

Structured - Implicit - Static

A structured grid that does not change between time-steps with all-to-all communications needed.

Examples-

- Radiative transfer problems with photon scattering
- Gravity calculations on a uniform grid
- Elliptical PDE's and Poisson problems

Unstructured - Implicit - Dynamic

A unstructured grid that changes between time-steps with all-to-all communications needed

Examples-

- Gravitational particle codes
- Adaptive Mesh Refinement using Poisson Equation

Taxonomy of Parallel Applications

- EP - many codes
- SES - many finite difference codes, including point Jacobi method
- SED - finite difference Lagrangian codes
- SIS - radiative transfer codes on fixed, structured grids
- SID - radiative transfer on moving grids
- UES - finite element hydro
- UED - SPH codes
- UIS - finite element radiation and elliptical solvers
- UID - n-body codes with gravity

Parallel Programming Patterns

There is an excellent review of *Patterns in Parallel Programming* in book by Mattson, Sanders, and Massingill.

They divide parallel design into three parts-

- Decomposition
- Dependency Analysis
- Design Evaluation

Finding Concurrency in Algorithms

- Organize by Task
- Organize by Data Decomposition
- Organize by Flow of Data

You can't completely break the need for communication. However, some patterns work well to minimize it.

Organize by Task

- Task Parallelism
 - multiple physical effects, ray tracing
 - each ray or event is processed separately
- Divide and Conquer
 - sorts, trees, FFTs
 - create recursive algorithms to reduce workload

Organize by Data Decomposition

- Geometric Decomposition
 - Break up grids into spatially equal sets
 - Scatter decomposition
 - Each geometric region is processed separately
- Recursive Data
 - Understanding and mapping relationships within complex data structures
 - Performing parallel operations on recursive data structures

Organize by Flow of Data

- Pipelining
 - Signal processing
 - Break processing in to different assembly line stages
- Event-based coordination
 - tasks that are mostly independent of each other, working in a coordinated fashion
 - managing tasks because critical

Finding Concurrency in Algorithms

- Organize by Task
 - Task Parallelism
 - Divide and Conquer
- Organize by Data Decomposition
 - Geometric decomposition
 - Recursive data
- Organize by Flow of Data
 - Pipeline
 - Event-based coordination

Program Supporting Structures

Different programming paradigms have different types of supporting structures. The book by Mattson *et al.* break these into four categories.

- SPMD - single program multiple data
- Loop Parallelism
- Master/Worker
- Fork/Join

Putting Languages to Supporting Structures

Table 5-2 of Mattson *et al.*

	OpenMP	MPI	Java
SPMD	***	***	**
Loop Parallelism	***	*	***
Master/Worker	**	***	***
Fork/Join	***		***

Supporting Structures vs Algorithmic Structures

Table 5-1 of Mattson *et al.*

	SPMD	Loop Parallelism	Master/ Worker	Fork/ Join
Task Parallelism	★ ★ ★★	★ ★ ★★	★ ★ ★★	★★
Divide and Conquer	★ ★ ★	★★	★★	★ ★ ★★
Geometric Decomposition	★ ★ ★★	★ ★ ★	★	★★
Recursive Data	★★		★	
Pipeline	★ ★ ★		★	★ ★ ★★
Event-Based Coordination	★★		★	★ ★ ★★

Basic Steps in N-body Codes

All N-body codes have the same set of basic steps

- read in initial data
- initialize calculation
- calculate the forces on each particle
- update the particle positions based on the new forces
- output data as needed
- repeat the last three steps until done

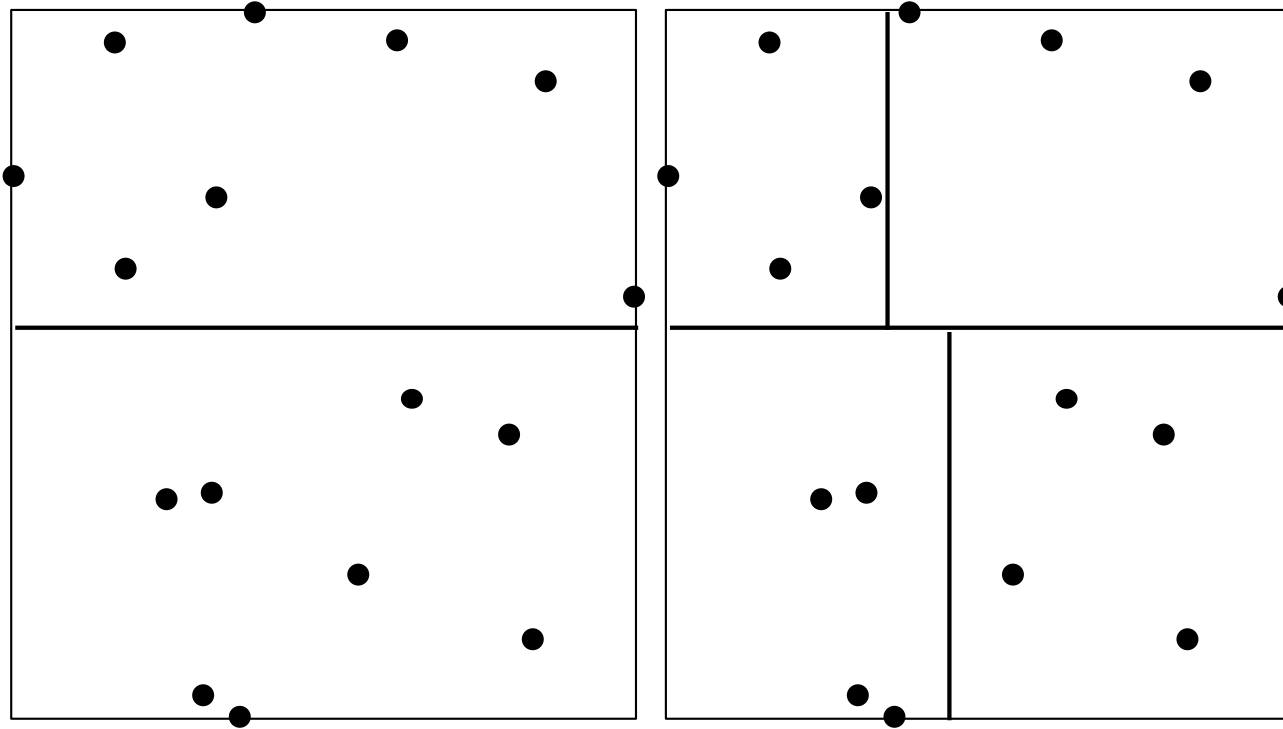
We can use tree data structures to approximate gravitational forces.

Orthogonal Recursive Bisection

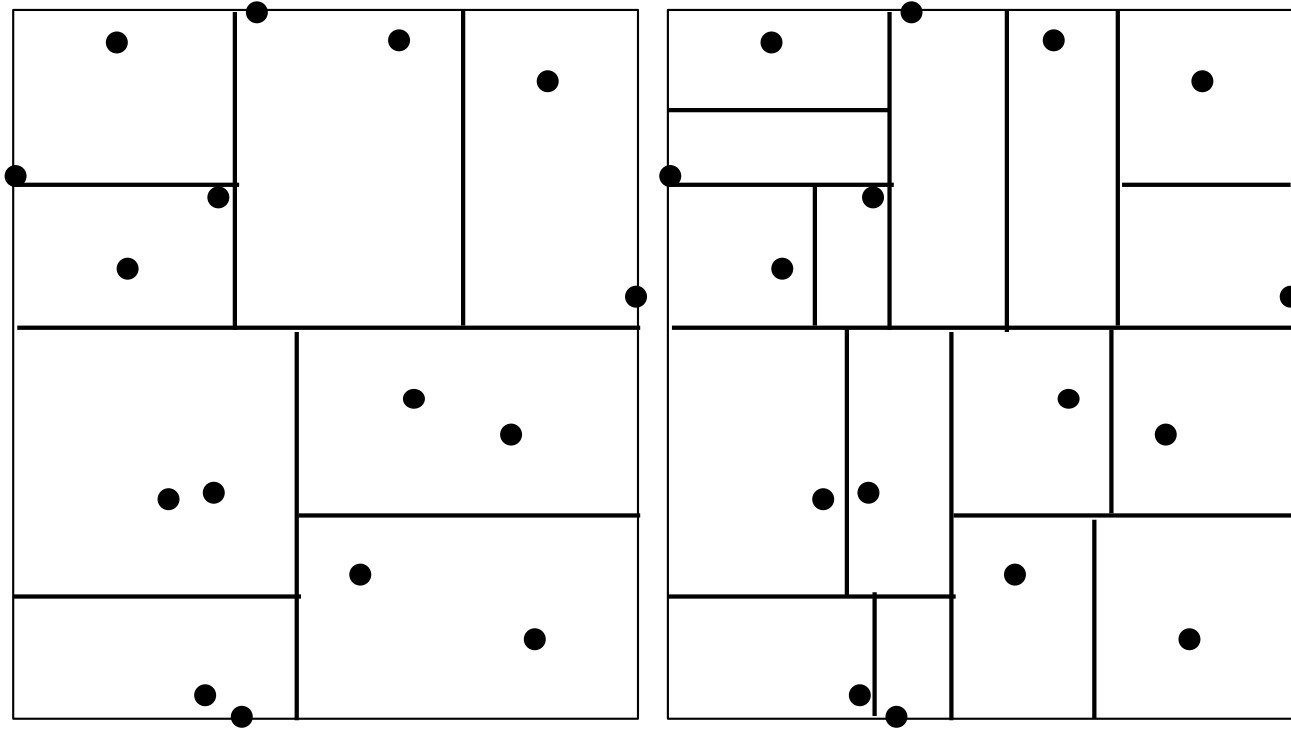
1. Define the group as the whole data set
2. Find the direction (x or y) that has the largest difference between max and min points
3. Sort along this direction*
4. Divide the group into two subgroups at the median
5. Repeat steps 2-5 with each subgroup if the subgroup size has more than one element

*Note: You only need to sort so the group is divided at the median. The actual order in each group is not important, so a fast median finder would work as well. However, sorting routines are usually fast and easy to find.

ORB Steps 1 and 2



ORB Steps 3 and 4



ORB in an Array

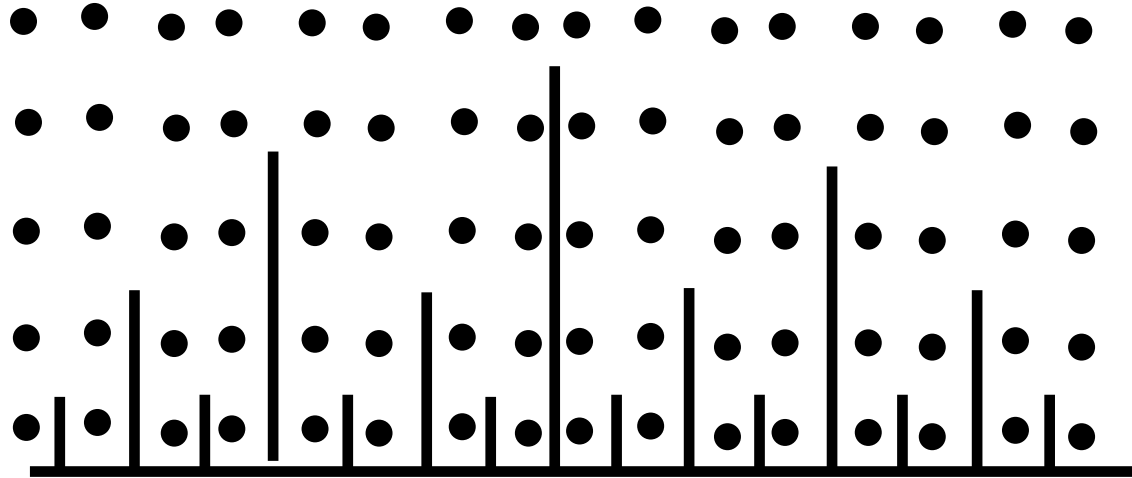


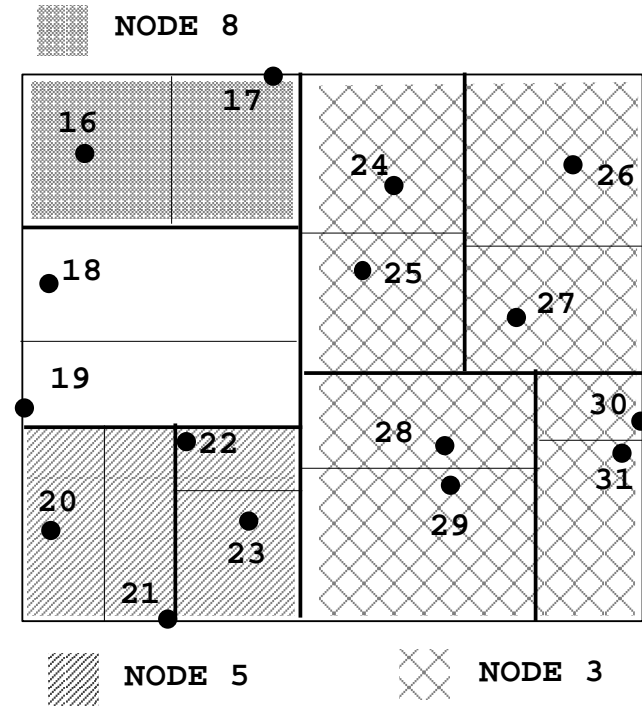
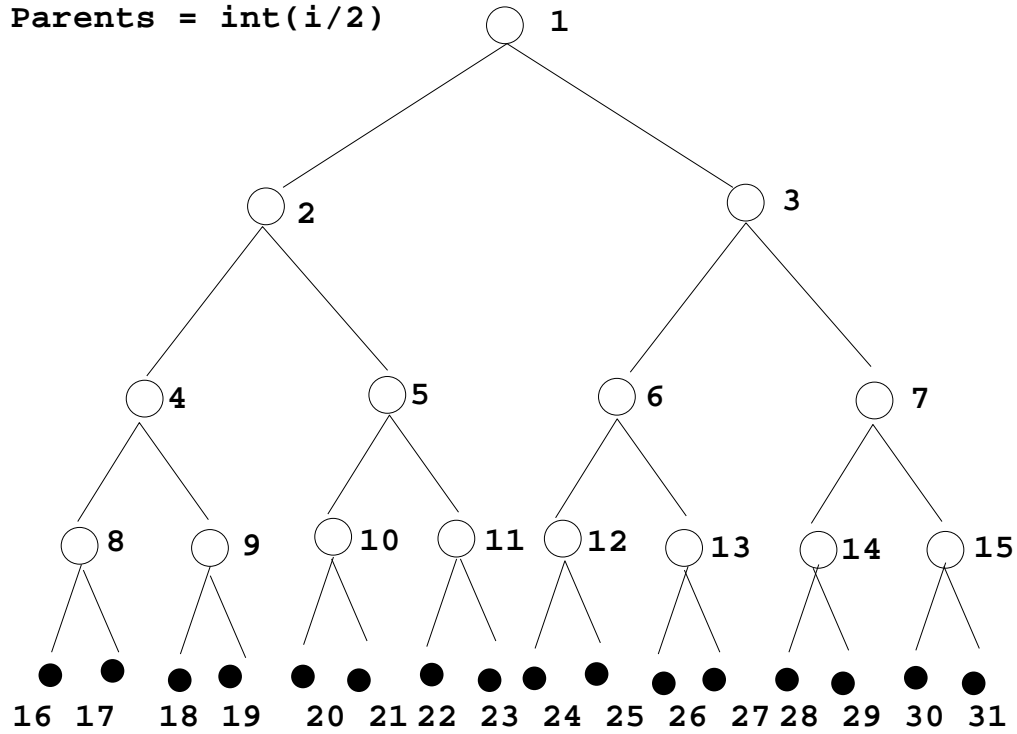
Figure 1: Data Set

Data is repeated rearranged and divided.

The Final Tree Structure

Daughters = $2i, 2i + 1$

Parents = $\text{int}(i/2)$



ORB in Parallel

- Distribute data across all nodes
- Find dimension with largest difference between the max and min in global data array
- Sort data along this dimension
- If there is more than one CPU in the group, divide PROCESSORS in this group into two groups, and repeat the last three steps
- Repeat on the particles internal to the node

This is a divide and conquer task model used to create geometric data decomposition.

Walking the Tree

o calculate gravitational forces, we need to walk each particle on the tree.

- each particle walks on the portion of the tree local to its node and accumulates the local part of the total gravitational force
- the necessary tree nodes from the neighboring trees are transferred to the local processor from the neighboring processors while this walk is going on
- the particles are walked on the non-local trees
- as space is freed up, additional data is transferred to the node

Each particle uses **task parallelism**. The transfer of data is based on **events-based coordination**.

Small Particle Case

If only a few particles are walked during a given time-step, the particles are distributed to all the nodes, and the forces are then summed together at the local nodes.

- identify particle that need to be walked
- if there is a small number, distribute particle positions to all nodes
- on every processor, walk the particle positions locally
- gather the results together

This is really a **task parallelism** implementation, that really could be done with loop unrolling.

Summary of Parallel N-body Gravity

- the code uses MPI so it runs well on clusters
- the main programming patterns used are:
 - geometric decomposition
 - divide and conquer
 - task parallelism
 - events based coordination

