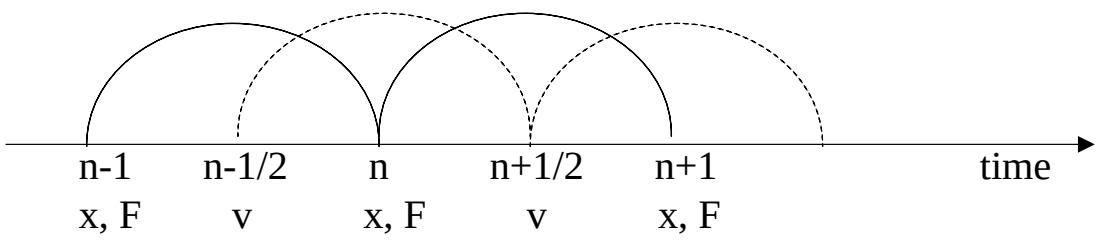
1. Setup distribution of N particles
2. Compute forces between particles
3. Evolve positions using ODE solver
4. Display/analyze results

Leap-frog scheme

Define positions (x) and forces (F) at time level n
velocities (v) at time level n+1/2
Then, for i^{th} particle

$$x_i^{n+1} = x_i^n + v_i^{n+1/2}\Delta t$$
$$v_i^{n+1/2} = v_i^{n-1/2} + F_i(x_i^n)\Delta t/m$$



To *start* integration, need initial x and V at two *separate* time levels.

Specify x_0 and v_0 and then integrate V to $\Delta t/2$ using high-order scheme

Accuracy of leap-frog scheme

Can show the truncation error in leap-frog is second order in Δt

Evolution eqns:
$$x_i^{n+1} = x_i^n + v_i^{n+1/2}\Delta t$$
$$v_i^{n+1/2} = v_i^{n-1/2} + F_i(x_i^n)\Delta t/m$$

Replace $V^{n-1/2}$ in second equation using first

$$v_i^{n+1/2} = (x_i^n - x_i^{n-1})/\Delta t + F_i(x_i^n)\Delta t/m$$

Substitute this back into first equation

$$x_i^{n+1} = x_i^n + (x_i^n - x_i^{n-1}) + F_i(x_i^n)\Delta t^2/m$$

Rearrange
$$\frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} = F_i(x_i^n)/m$$

This is central difference formula for F=ma.

Let X(t) be the "true" (analytic) solution.

Then
$$\frac{X_i^{n+1} - 2X_i^n + X_i^{n-1}}{\Delta t^2} = F_i(X_i^n)/m + \delta \qquad \text{EQ. 1}$$

Use a Taylor expansion to compute X^{n+1} and X^{n-1}

$$X_i^{n+1} = X_i^n + \Delta t\frac{\partial X}{\partial t} + \frac{\Delta t^2}{2}\frac{\partial^2 X}{\partial t^2} + \frac{\Delta t^3}{6}\frac{\partial^3 X}{\partial t^3} + \frac{\Delta t^4}{24}\frac{\partial^4 X}{\partial t^4} + \ldots$$
$$X_i^{n-1} = X_i^n - \Delta t\frac{\partial X}{\partial t} + \frac{\Delta t^2}{2}\frac{\partial^2 X}{\partial t^2} - \frac{\Delta t^3}{6}\frac{\partial^3 X}{\partial t^3} + \frac{\Delta t^4}{24}\frac{\partial^4 X}{\partial t^4} - \ldots$$

Thus $X_i^{n+1} - 2X_i^n + X_i^{n-1} = \Delta t^2\frac{\partial^2 X}{\partial t^2} + \frac{\Delta t^4}{12}\frac{\partial^4 X}{\partial t^4} + \ldots$

Substitute these back into eq. 1

$$\frac{\partial^2 X}{\partial t^2} + \frac{\Delta t^2}{12}\frac{\partial^4 X}{\partial t^4} = F_i(X_i^n)/m + \delta$$

Truncation error $O(\Delta t^2)$

Truncation versus round-off error

Note the error we have just derived is truncation error

Unavoidable result of approximating solution to some order in x or t

Completely unrelated to round-off error, which results from representing the continuous set of real numbers with a finite number of bits.

Truncation error can be reduced by using smaller step Δt, or higher-order algorithm

Round-off error can be reduced by using higher precision (64 bit rather than 32, etc.), and by ordering operations carefully.

In general, truncation error is much larger than round-off error

Stability of leap-frog scheme

Easiest to illustrate with an example. Suppose the force is given by a harmonic oscillator, that is:

$$F(x)/m = -\Omega^2 x$$

Then "true" (analytic) solution is

$$\boxed{X(t) = A\cos\Omega t + B\sin\Omega t}$$

Substitute force law into leap-frog FDE for F=ma

$$x^{n+1} - 2x^n + x^{n-1} = -\Delta t^2 \Omega^2 x^n$$

Look for oscillatory solutions of the form $x = x_0 e^{i\omega t}$

$$e^{i\omega\Delta t} - 2 + e^{-i\omega\Delta t} = -\Delta t^2\Omega^2$$

giving $\sin\frac{\omega\Delta t}{2} = \pm\frac{\Omega\Delta t}{2}$

This is good! Leap-frog (correctly) gets oscillatory solutions, but at a modified frequency

$$\omega' = \frac{2}{\Delta t} \arcsin \frac{\Omega \Delta t}{2}$$

Note this gives correct solution (Ω) as Δt --> 0

For ΩΔt > 2, frequency becomes complex.

Real part of ω' gives oscillatory solution, imaginary part gives exponentially growing (unstable) solution.

So stability limit is Δt < 2/Ω; or $\boxed{\Delta t < 2/[(dF/dx)/m]^{1/2}}$ in general

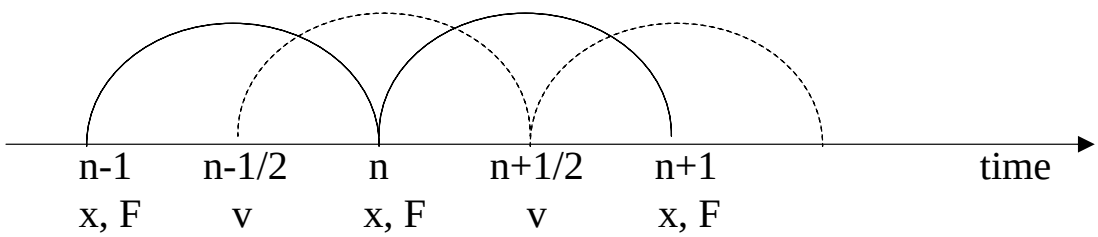
Above is a simple example of a von-Neumann stability analysis

Consistency of leap-frog scheme

Leap-frog is consistent in the sense that as Δt --> 0, the difference equations converge to the differential equations

$$\frac{\partial^2 X}{\partial t^2} + \frac{\Delta t^2}{12} \frac{\partial^4 X}{\partial t^4} = F_i(X_i^n)/m + \delta$$

Leap-frog is also a *symplectic method* (time symmetric). Scheme has same accuracy for Δt negative.



Efficiency of leap-frog scheme

Leap-frog is extremely efficient in terms of computational cost (only 12 flops per particle excluding force evaluation)

Also extremely efficient in terms of memory storage (does not require storing multiple time levels).

All the work (and memory) is in force evaluation: 10N flops per particle for direct summation

To update all particle positions in one second on a 1 Gflop processor requires $N < 10^4$

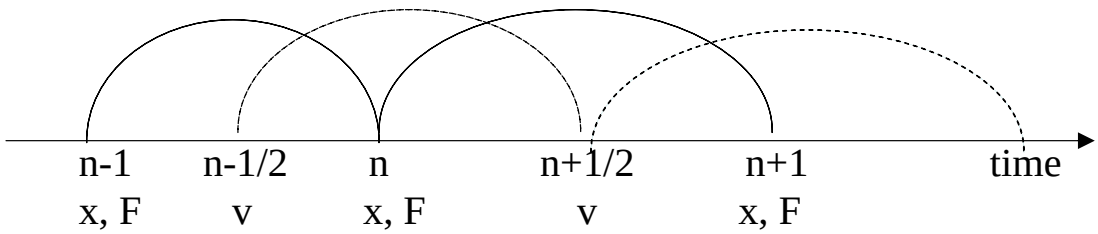
Extra efficiency can be gained by using different timesteps for each particle (more later).

Variable time steps with leap-frog

For efficiency, need to take variable time steps (evolve particles at center of cluster on smaller timestep than particles at edge).

Variable time steps with leap-frog

However, this destroys symmetry of leap-frog; greatly increases truncation error.



But, variable timestep leap-frog can be symmetrized

Hut, Makino, & McMillan 1995

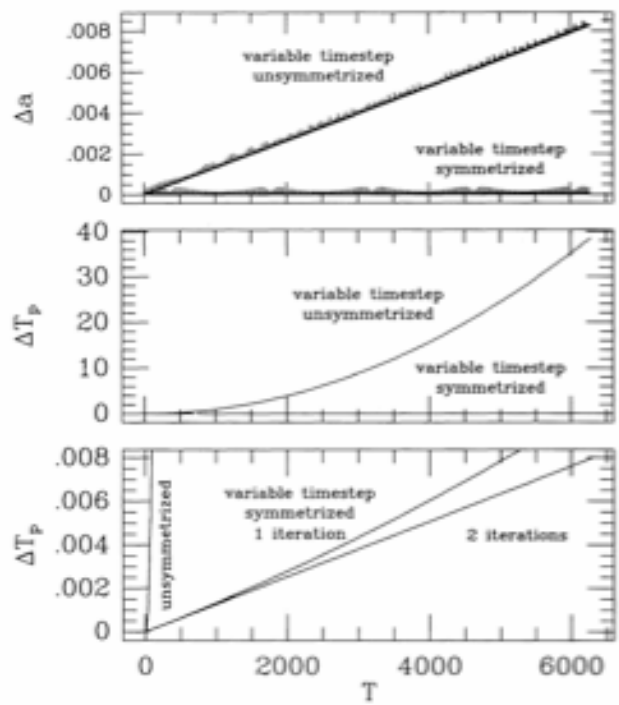


FIG. 2.—The same Kepler ellipse integration as in the previous figure,

Force evaluation with variable time steps.

Now particle positions are known at different time levels.

Greatly complicates force calculation. Must compute derivatives of force wrt time, and use Taylor expansion to compute total force on particle at current position.

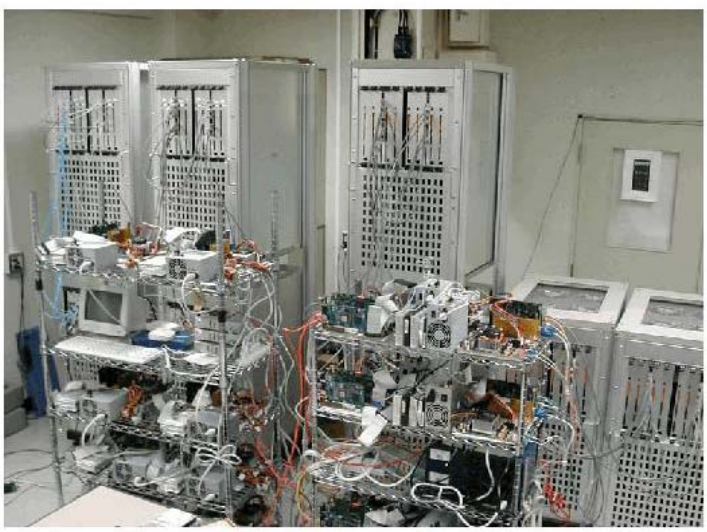
The Good: Allows higher-order (Hermite) integration methods.

The Bad: This just makes force evaluation even more expensive!

The Ugly: Direct N-body must be optimized if we are to go beyond 10^4 particles.

Solving the force problem with hardware.

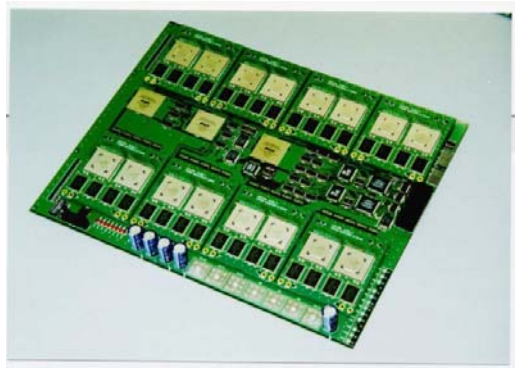
The 64-Tflops GRAPE-6 system



Special purpose hardware to compute force:

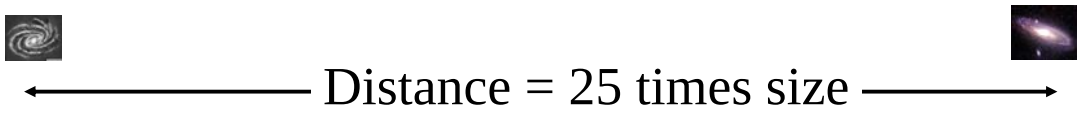
$$F_i = \Sigma_j \frac{GM_iM_j}{R_{ij}^2}$$

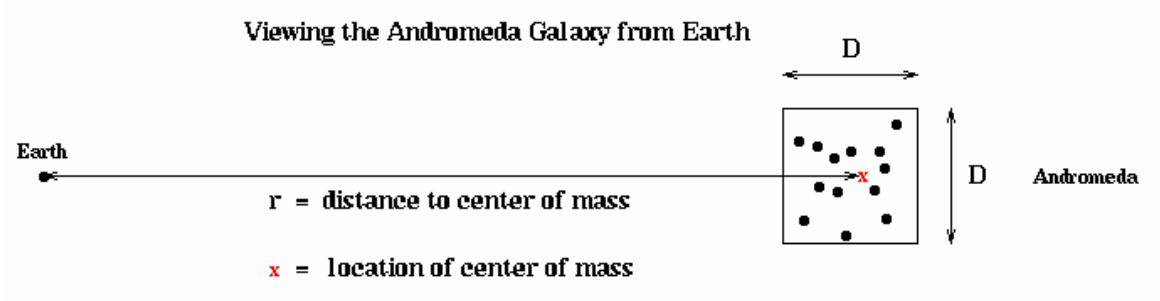
GRAPE-6



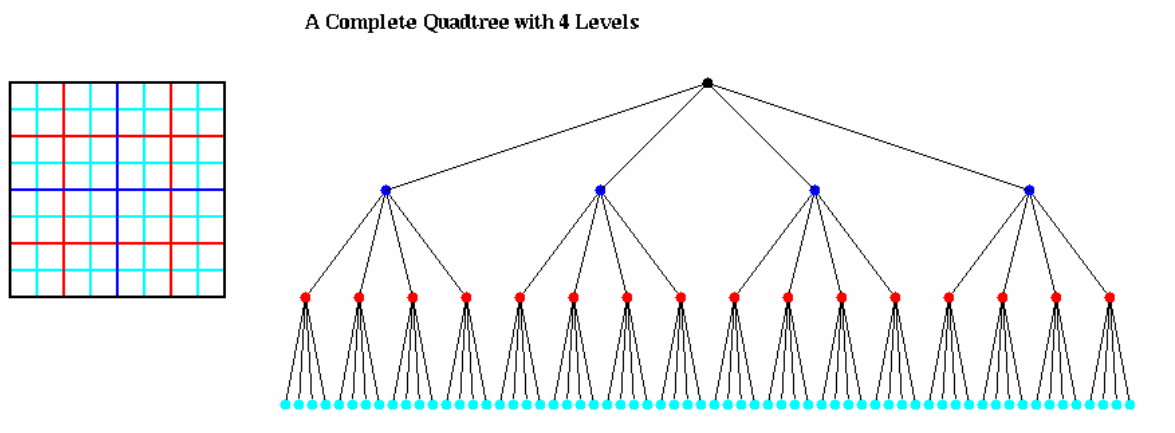
- The 6th generation of GRAPE (Gravity Pipe) Project
- Gravity calculation with 31 Gflops/chip
- 32 chips / board ⇒ 0.99 Tflops/board
- 64 boards of full system is installed in University of Tokyo ⇒ 63 Tflops
- On each board, all particle data are set onto SRAM memory, and each target particle data is injected into the pipeline, then acceleration data is calculated
- Gordon Bell Prize at SC2000, SC2001 (Prof. Makino, U. Tokyo) also nominated at SC2002

Solving the force problem with software -- tree codes

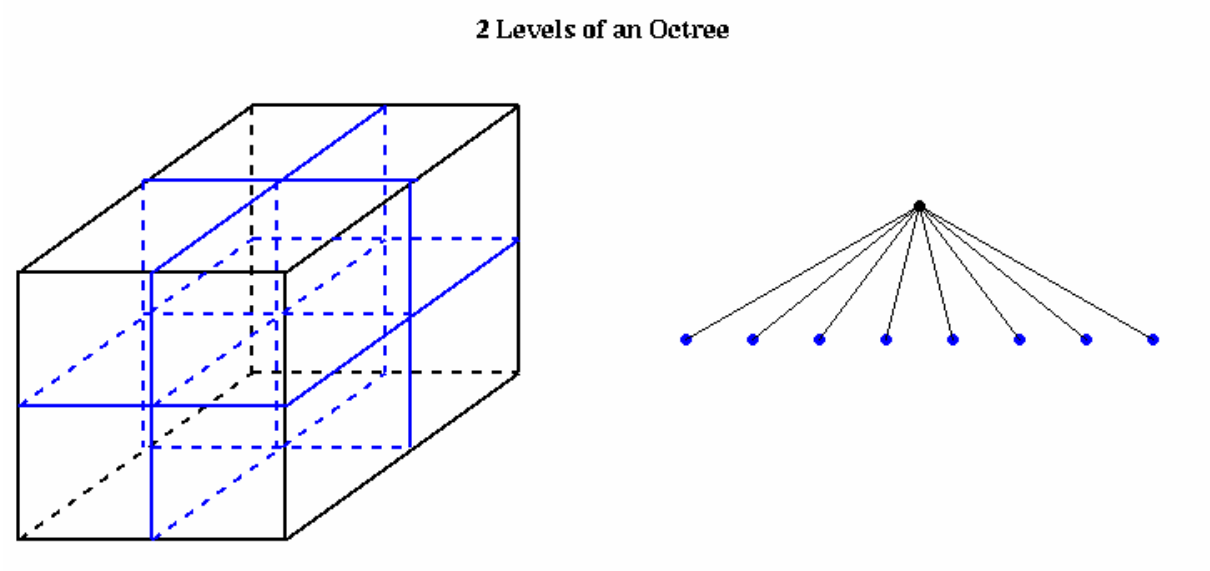


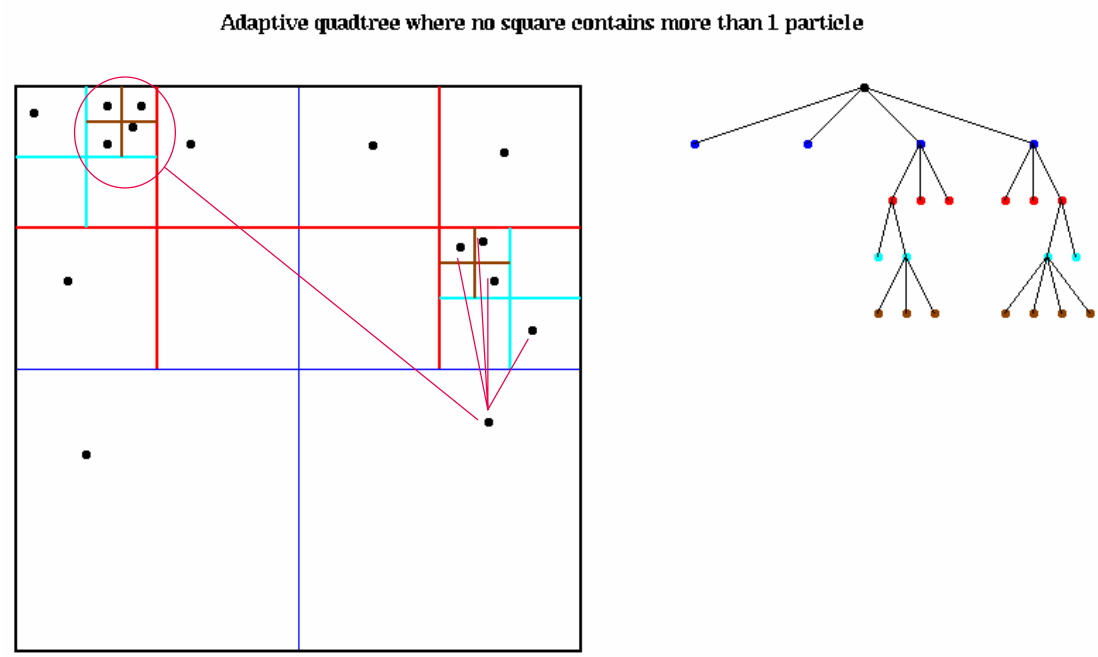


Organize particles into a tree. In Barnes-Hut algorithm, use a quadtree in 2D



In 3D, Barnes-Hut uses an octree

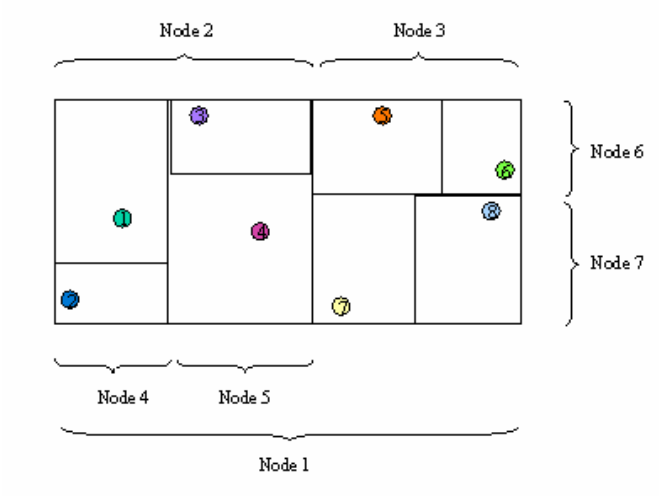




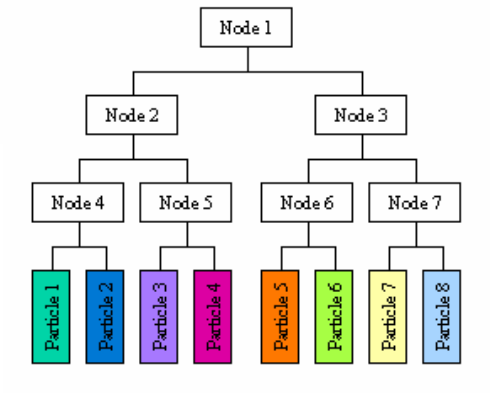
If angle subtended by the particles contained in any node of tree is smaller than some criterion, then treat all particles as one.

Results in an Nlog(N) algorithm.

Alternative to Barnes-Hut is KD tree.



- KD tree is binary - extremely efficient
- Requires N to be power of 2
- $N_{nodes} = 2N-1$



Parallelizing tree code.

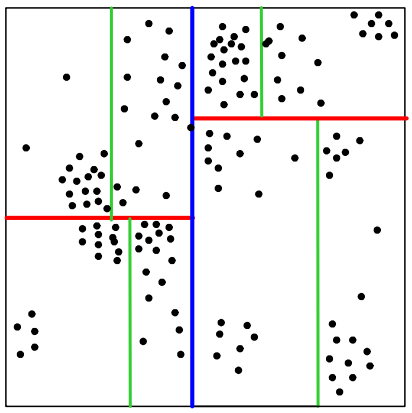
Best strategy is to distribute particles across processors. That way, work of computing forces and integration is distributed across procs.

Challenge is *load balancing*

- Equal particles ≠ equal work.
  - Solution: Assign costs to particles based on the work they do
- Work unknown and changes with time-steps
  - Insight : System evolves slowly
  - Solution: *Count* work per particle, and use as cost for next time-step.

A Partitioning Approach: ORB

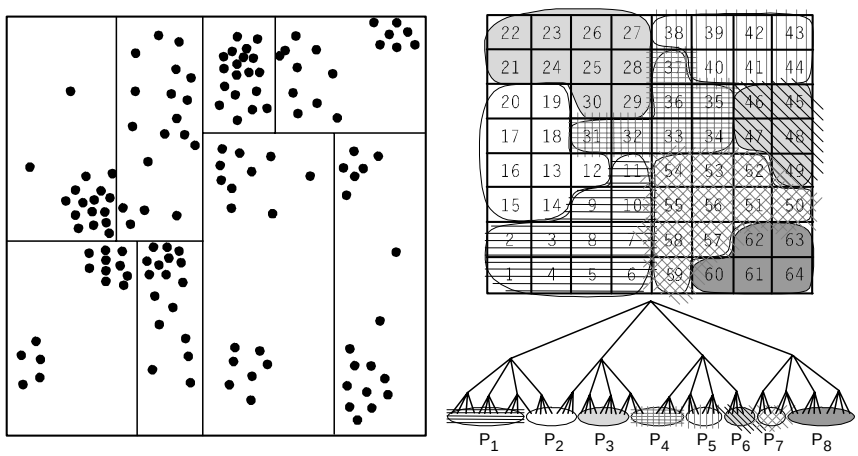
- Orthogonal Recursive Bisection:
  - Recursively bisect space into subspaces with equal work
    - Work is associated with bodies, as before
  - Continue until one partition per processor



- High overhead for large no. of processors

Another Approach: Costzones

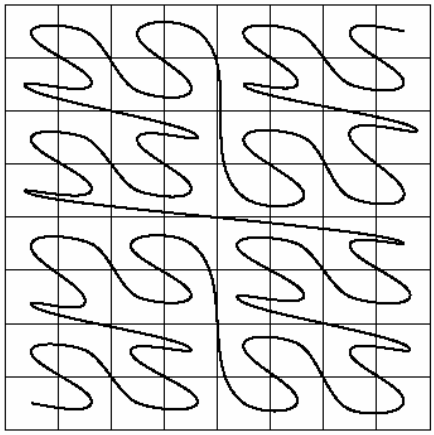
- Insight: Tree already contains an encoding of spatial locality.



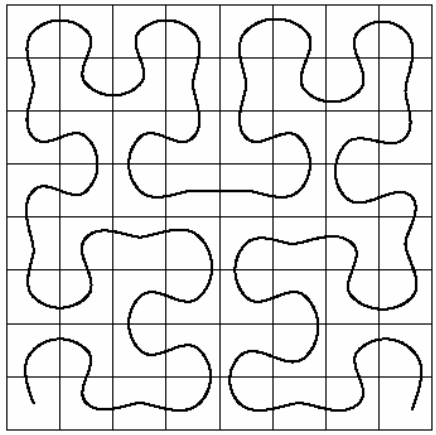
(a) ORB (b) Costzones

- Costzones is low-overhead and very easy to program

Space Filling Curves



Morton Order



Peano-Hilbert Order