# N-Body Methods

*Source:*

Load Balancing and Data Locality in Adaptive Hierarchical N-Body Methods: Barnes-Hut, Fast Multipole, and Radiosity

by Singh, Holt, Totsuka, Gupta, and Hennessy

*(except Sections 4.1.2., 4.2, 9, and 10)*

# N-body problems

- Given are $N$ bodies (molecules, stars, ...)
- The bodies exert *forces* on each other (Coulomb, gravity, ...)
- Problem: simulate behavior of the system over time
- Many applications:

  Astrophysics (stars in a galaxy)
  Plasma physics (ion/electrons)
  Molecular dynamics (atoms/molecules)
  Computer graphics (radiosity)

# Basic N-body algorithm

```
for each timestep do
   Compute forces between all bodies
   Compute new positions and velocities
od
```

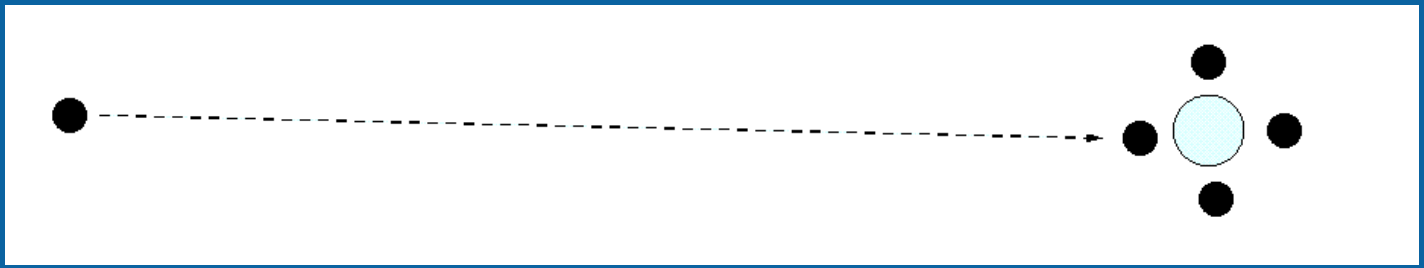
- O($N^2$) compute time per timestep
- Too expensive for realistics problems (e.g., galaxies)
- Barnes-Hut is O($N \log N$) algorithm for hierarchical N-body problems

# Hierarchical N-body problems

- Exploit physics of many applications:

  Forces fall very rapidly with distance between bodies
  Long-range interactions can be approximated

- Key idea: *group* of distant bodies is approximated by a *single* body with same mass and center-of-mass



# Data structure

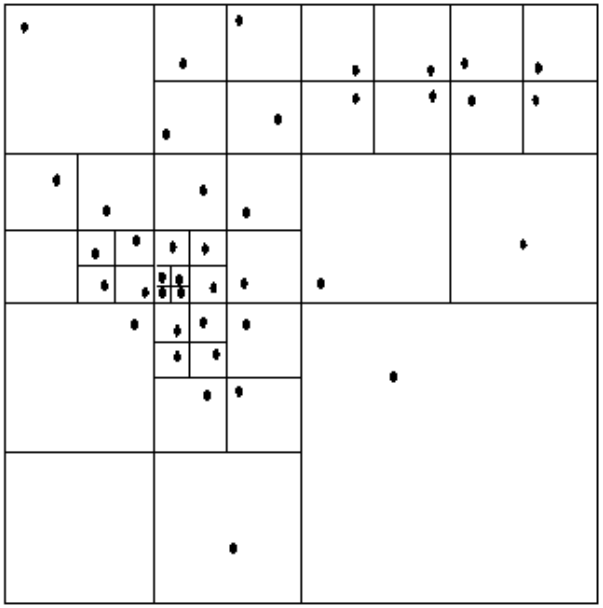
- Octree (3D) or quadtree (2D):

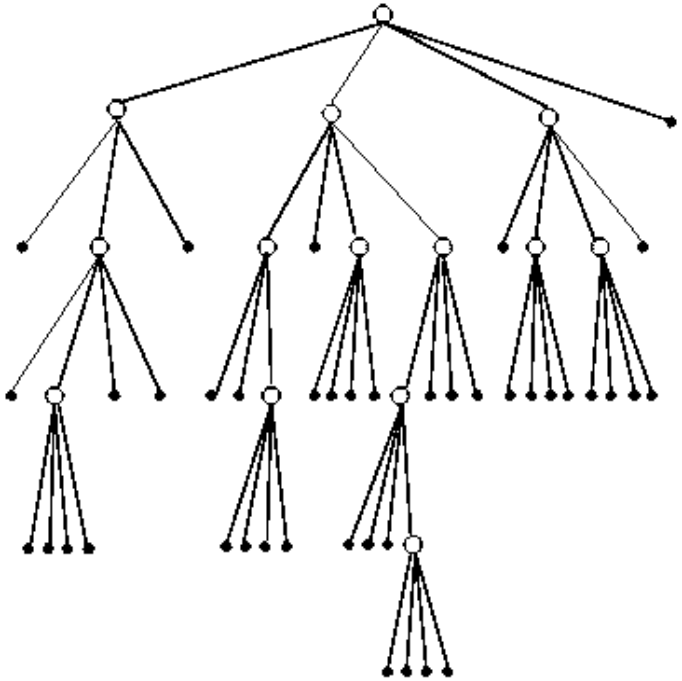
  Hierarchical representation of physical space

- Building the tree:

  Start with one cell with all bodies (bounding box)
  Recursively split cells with multiple bodies into subcells

# Example (Fig. 5 from paper)





# Barnes-Hut algorithm

```
for each timestep do
   Build tree
   Compute center-of-mass for each cell
   Compute forces between all bodies
   Compute new positions and velocities
od
```

- Building the tree: recursive algorithm (can be parallelized)
- Center-of-mass: upward pass through the tree
- Compute forces: 90% of the time
- Update positions and velocities: simple (given the forces)

# Force computation of Barnes-Hut

```
for each body B do
   B.force := ComputeForce(tree.root, B)
od

function ComputeForce(cell, B): float;
   if distance(B, cell.CenterOfMass) > threshold then
      return DirectForce(B.position, B.Mass,
                         cell.CenterOfMass, cell.Mass)
   else
      sum := 0.0
      for each subcell C in cell do
          sum +:= ComputeForce(C, B)
      return sum
```

# Parallelizing Barnes-Hut

- Distribute bodies over all processors

  In each timestep, processors work on different bodies
- Communication/synchronization needed during

  Tree building

  Center-of-mass computation

  Force computation
- Key problem is efficient parallelization of force-computation
- Issues:

  Load balancing

  Data locality

# Load balancing

- Goal:

  Each processor must get same amount of work

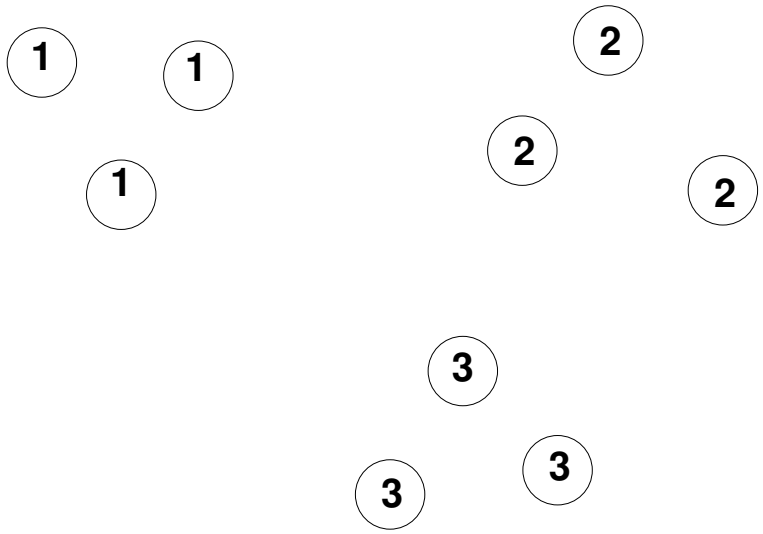
- Problem:

  Amount of work per body differs widely

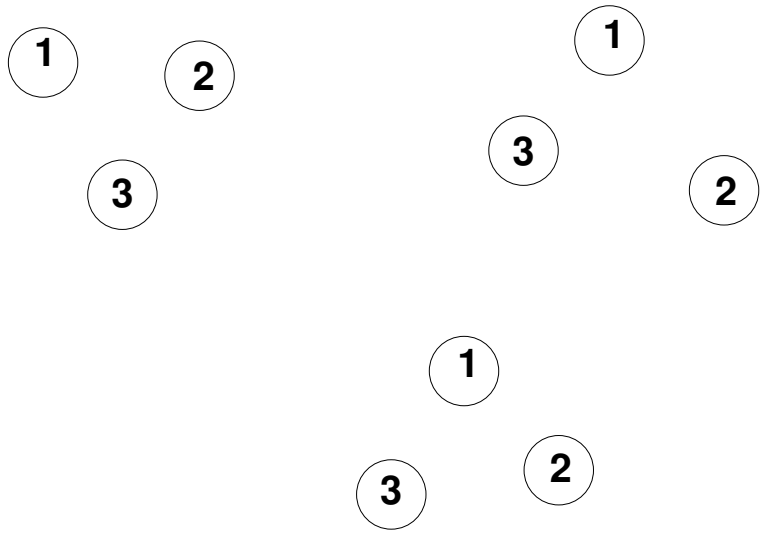
# Data locality

- Goal:
  - ⋆ Each CPU must access small number of bodies many times
  - ⋆ Reduces communication overhead
- Problems
  - ⋆ Access patterns to bodies not known in advance
  - ⋆ Distribution of bodies in space changes (slowly)

# Example Data locality




Good locality




Bad locality

# Simple distribution strategies

- Distribute iterations

  Iteration = computations on single body in 1 timestep

- Strategy-1: Static distribution

  Each processor gets equal number of iterations

- Strategy-2: Dynamic distribution

  Distribute iterations dynamically

- Problems

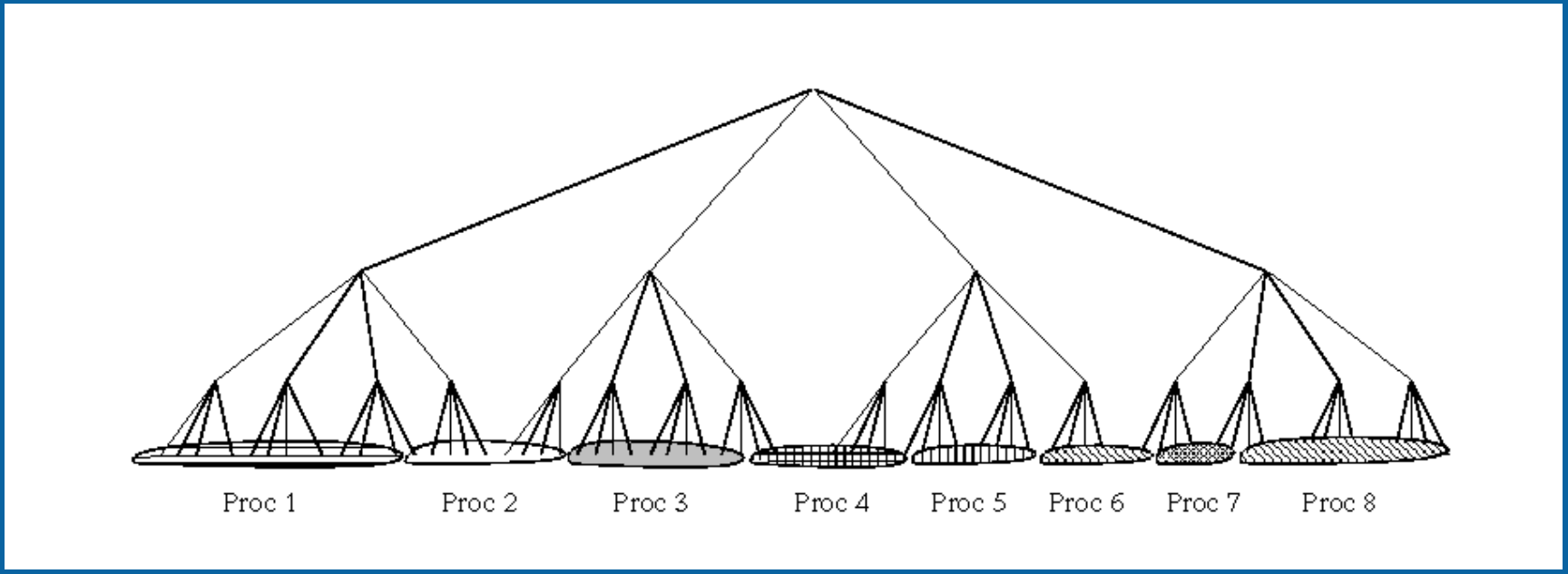
  Distributing iterations does not take locality into account

  Static distribution leads to load imbalances

# More advanced distribution strategies

- Load balancing: cost model
  - Associate a computational *cost* with each body
  - Cost = amount of work (number of interactions) during previous timestep
  - Each processor gets same total cost
  - Works well, because system changes slowly
- Data locality: costzones
  - Observation: octree more or less represents spatial (physical) distribution of bodies
  - Thus: partition the tree, not the iterations
  - Costzone: contiguous zone of costs

# Example costzones




Optimization: improve locality using clever child numbering scheme

# Experimental system–DASH

- DASH multiprocessor
  - Designed at Stanford university
  - One of the first NUMAs (Non-Uniform Memory Access)
- DASH architecture
  - Memory is physically distributed
  - Programmer sees shared address space
  - Hardware moves data between processors and caches it
  - Implemented using directory-based cache coherence protocol

# DASH prototype

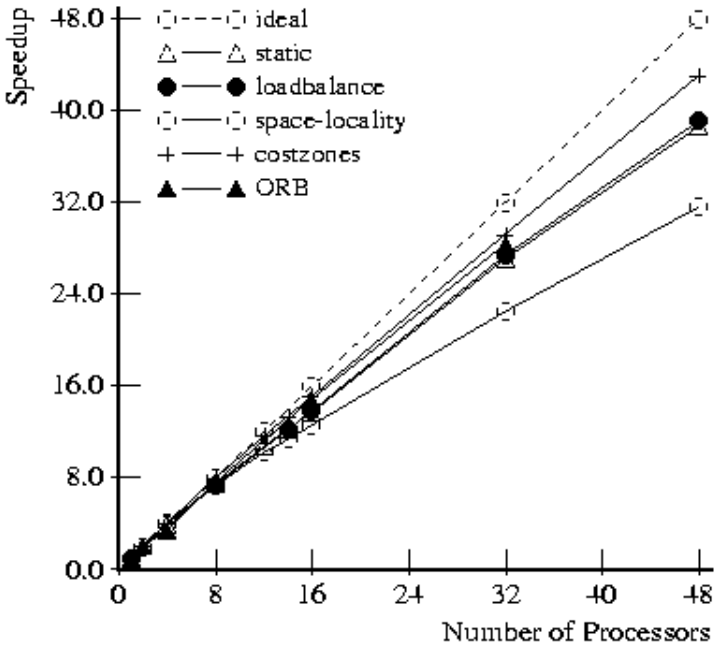
- 48-node DASH system
  - 12 clusters of 4 processors (MIPS R3000) each
  - Shared bus within each cluster
  - Mesh network between clusters
  - Remote reads 4x more expensive than local reads
- Also built a simulator
  - More flexible than real hardware
  - Much slower

# Performance results on DASH

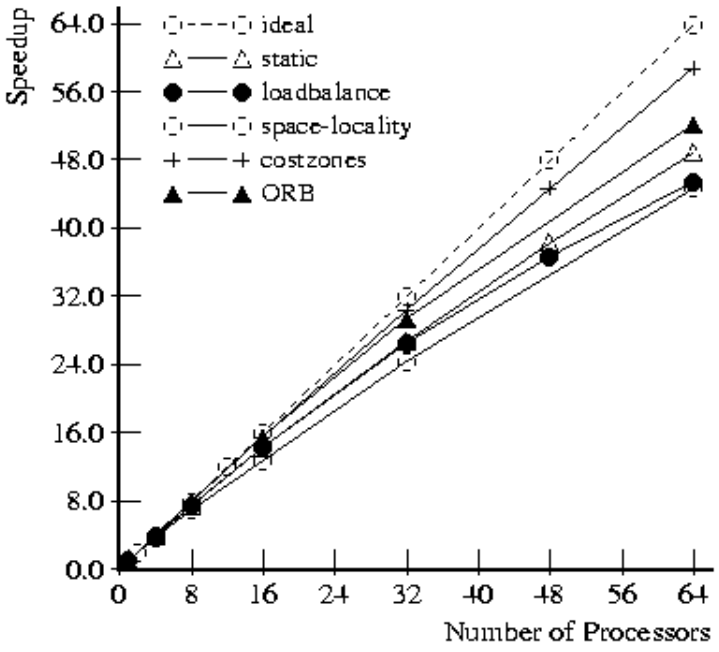
- Costzones reduce load imbalance and communication overhead
- Moderate improvement in speedups on DASH

  Low communication/computation ratio

# Speedups measured on DASH (figure 17)

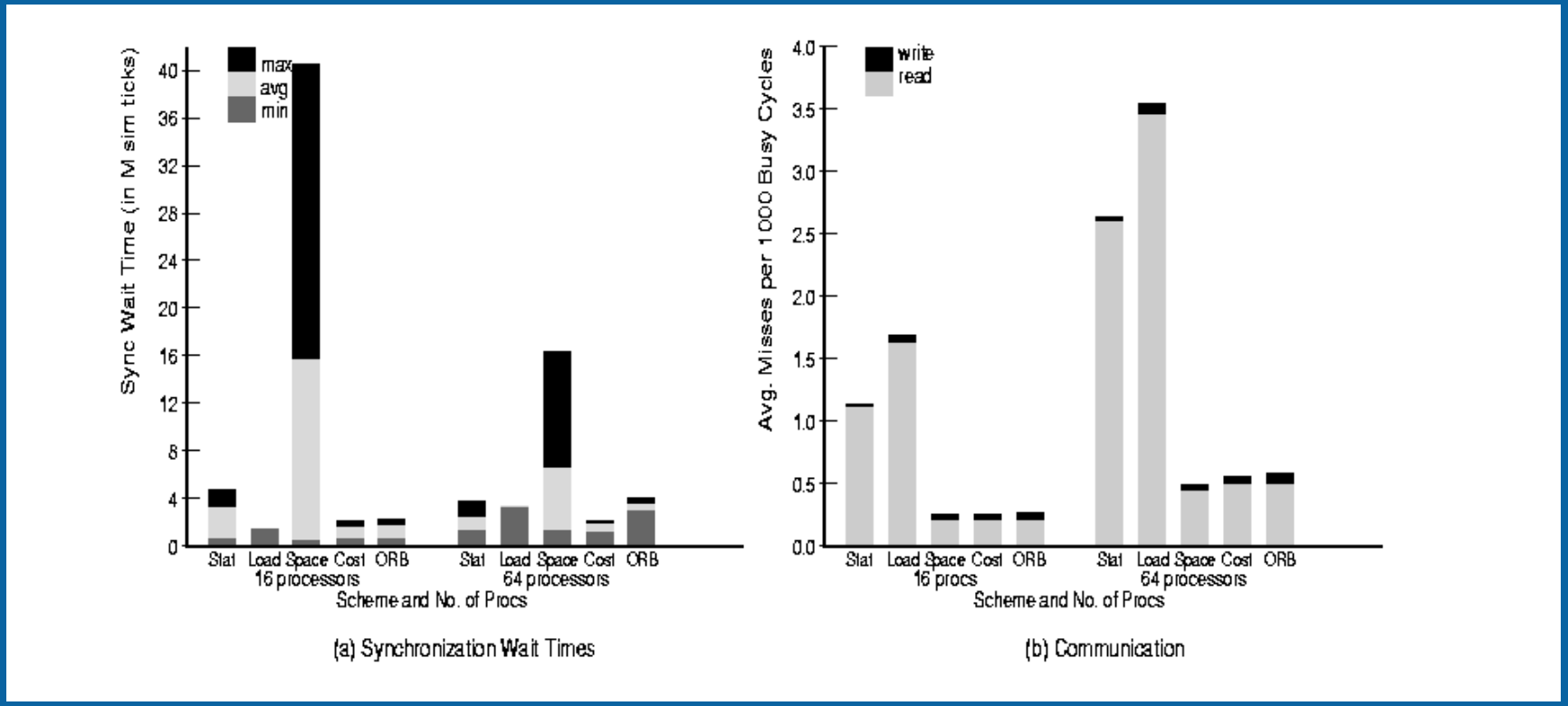



(a) 32K Particles on DASH




(b) 8K Particles on Simulator

# Simulator statistics (figure 18)



# Conclusions

- Parallelizing efficient O($N \log N$) algorithm is much harder that parallelizing O($N^2$) algorithm
- Barnes-Hut has nonuniform, dynamically changing behavior
- Key issues to obtain good speedups for Barnes-Hut

  Load balancing $\longrightarrow$ cost model

  Data locality $\longrightarrow$ costzones
- Optimizations exploit physical properties of the application