N-body Problem

- N bodies, with point mass m_1, m_2, … went into a bar ...

N-body Problem

- There are N bodies with point mass m_1, m_2, …
- "Point mass" means the bodies are approximated as single points with the given mass.
- Each *pair* of bodies are attracted by gravitational force of magnitude:

$$F = G\, m_i\, m_j / d^2$$

- where d is distance between the points.

Variations

- Charged particles, coulomb force
- Molecular dynamics, particles are atoms
- Fluid dynamics, particles are droplets

N-body

- Force is mass x acceleration.
- Acceleration is force/mass.
- Acceleration is also change in velocity.
- So the **net acceleration** of the points can be obtained by:
  - Computing for each point i, the vector *sum* over all *other* points j of

$$G\, m_j (x_j - x_i)/d^3$$

to account for x_j-x_i in numerator

N-body: Euler's Method

- Knowing the acceleration, we can compute the change in vector velocity for a small time-step.
- Knowing the vector velocity, we can compute the new position for a small time-step.

N-body

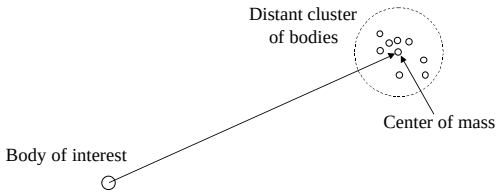
- The computation of the acceleration for a single point is O(N).
- The computation for N points is $O(N^2)$.
- This computation must be repeated each time step.
- For large N, this can be significant.

N-body

- Each point's accelerations can be computed independently of the others.
- Is this not a pleasantly-parallel problem?
- Actually is an embarassingly less-parallel problem, because a significant *algorithmic* speedup is possible.

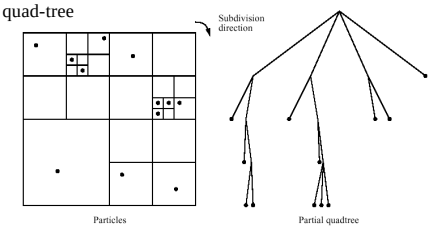
Barnes-Hut Algorithm for the N-body problem

If a cluster of bodies is a long distance away from a given point, the effect of the entire cluster on the point is well-approximated by a composite mass located at the center of mass of the cluser.



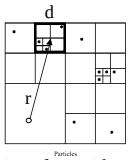
Barnes-Hut Algorithm (*Nature*, **324**, December 1986)

Divide space up into an oct-tree (3D) or quad-tree (2D).
Stop when a cell contains 0 or 1 point masses.



Barnes-Hut Algorithm

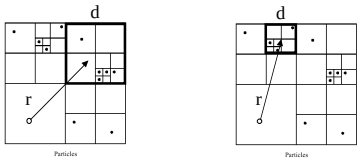
Each cell is approximated, for *distant* points, by the contained mass located at its center of gravity.
A criterion is needed for determining what is *distant.*



Let d be the dimension of one side of a box.
Let r be the distance to the centroid of the box.
The smaller d/r is, the less error in this approximation.

Barnes-Hut Algorithm

The smaller d/r is, the less error in this approximation.
For an arbitrary point and box, d/r may be **too large**.



We generally must sub-divide the boxes (walk down the tree) until d/r becomes acceptable, *then* use the approximation on that box.
Computing the net acceleration is thus recursive.

Barnes-Hut Algorithm

- The approximation is called the Multipole Approximation.
- The acceptability criterion is called the MAC:
  Multipole Acceptability Criterion.
- Recommended value is d/r < 1/sqrt(3) = 0.57735.
- Many other criteria exist.

Barnes-Hut Algorithm

- On the preceding slides we indicated how the accelerations can be computed for one state.
- The particles will be in *different positions* in the next time step.
- Therefore the tree will need to be reconstructed on ***every*** time step.

Complexity of One Time-Step of Barnes-Hut Algorithm

- If the particles are reasonably-uniformly distributed over the bounding region, then
  - The height of the tree will be O(log N).
  - The number of steps to compute the tree, including the centroids of each sub-box, will be O(N log N).
  - The cost of computing the acceleration on *one* particle is O(log N).
  - Therefore the cost of one time step is O(N log N).

Parallel Computation of Barnes-Hut Algorithm

- Can Barnes-Hut Exploit Parallelism?
  - Steps are:
    - O(N) finding outer bounds of set of particles
    - O(N log N) tree-construction computation
    - O(N) center of mass computation (for tree)
    - O(N log N) point acceleration computation ⇦ dominant
  - Which steps can be parallelized?

Parallel Computation of Barnes-Hut Algorithm

- Conjectured parallel versions for large N, p processors, ignoring communication costs
  - Each step is:
    - O(N/p) finding outer bounds of set of particles
    - O(N log N/p) tree-construction computation
    - O(N/p) center of mass computation (for tree)
    - O((N log N)/p) point acceleration computation
  - O((N logN)/p) overall

Parallel Computation of Barnes-Hut Algorithm

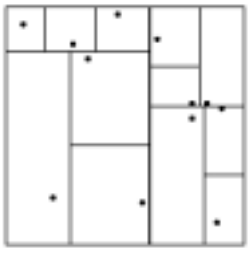
- What if we can't ignore communication costs (e.g. on distributed-memory system)?
  - One problem is **load-balancing**:
    - How do we distribute the computational work onto p processors so that each one is approximately equally busy?
      - finding outer bounds of set of particles
      - tree-construction computation
      - center of mass computation (for tree)
      - point acceleration computation ⇦ dominant

Orthogonal Recursive Bisection for load-balancing

Position vertical line so as to divide number of points in half.
Within each half, position horizontal line so as to divide those numbers of points in half, etc., until there are as many divisions overall as there are processors.

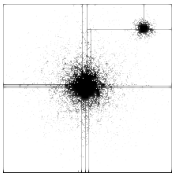
This is separate from the partitioning used in the quad-tree.

Load-balancing does not demand that spatially-close points be on the same processor.



Orthogonal Recursive Bisection

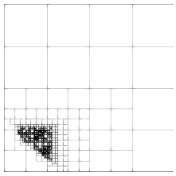
Example: Decomposition Resulting from Orthogonal Recursive Bisection of a System with Two Galaxies



From: http://www.npac.syr.edu/copywrite/pcw/node281.html

A Related Issue: Locality of Data can be Used to Advantage

Optimizing the acquisition of locally-essential data.



Each square represents a datum required to compute force on a point in the lower-left quadrant.

See: http://www.npac.syr.edu/copywrite/pcw/node282.html

Case Study

(from http://www.npac.syr.edu/copywrite/pcw/node283.html)

- 1992, 512-processor Intel Delta at Caltech
- 17.15 million bodies for approximately 600 time steps
- simulated regions of the universe 100 megaparsec
- ran at an aggregate speed exceeding 5000 MFLOPS/sec, generating 25 GB of data
- initialized with random-density fluctuations consistent with the "cold dark matter" hypothesis
- 1992 Gordon Bell Price for performance in practical parallel processing research

Fast Multipole Method

(L.. Greengard and V. Rokhlin, J. Comp. Phys. 73 (1987) 325.)

- This is an alternate method for the N-body problem.
- It is based on establishing a **vector field** in which the force on a particle can be evaluated.
- (A multiple expansion is akin to a Taylor's series.)
- Like Barnes-Hut, it uses a quad or oct-tree.
- It is O(N) per step, but the constant may be higher than in the O(N log N) Barnes-Hut algorithm.
- O(N/p) parallelization is possible.

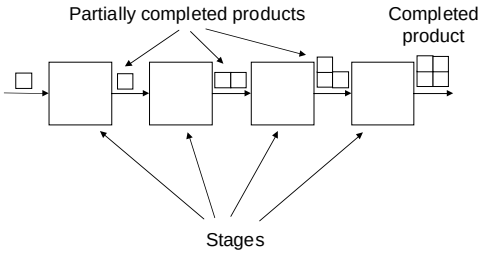
Pipelining

PP Ch. 5

Pipelining Defined

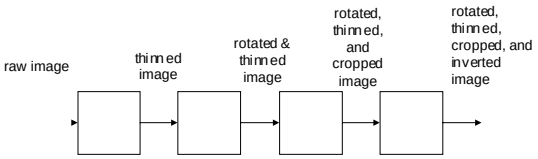
- A "product" in a stage of partial completion, is moved along through a series of "stations", becoming more complete at each station.
- The stations operate in parallel on multiple products, each working on a product in its respective stage of completion
- There may or may not be parallelism within a given station.

Pipelining



Examples of Computational Products

Product	Partially Complete Product
Number	Approximation to a number
Sorted array	Partially sorted array
Image	Transformed image

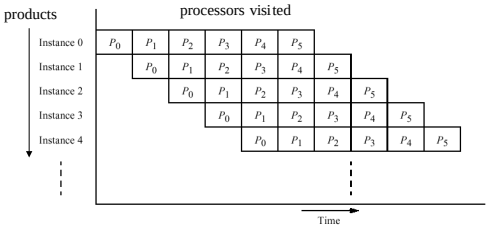


Utility of Pipelines

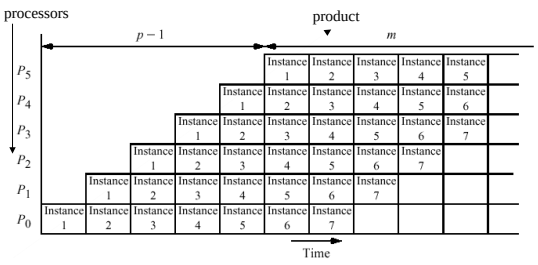
- Pipelines can provide a modular approach to constructing functions, rather than trying to invent or manage one single comprehensive function.
- Unix users are used to this kind of thing:

  (thin < image) | rotate | crop | invert

Pipeline Timing from the products' view



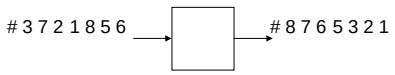
Pipeline Timing from the processors' view



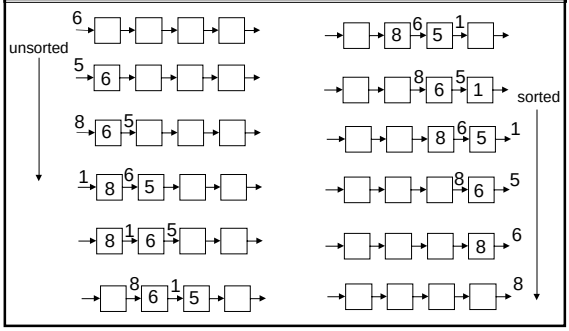
Distributed Pipelining

(my definition)

- In *distributed* pipelining, the result is not what comes out the last stage in one step, but rather the **collective equence** of things coming out.



Example of Distributed Pipeline (adapted from PP): Sorting



Pipeline Sorting

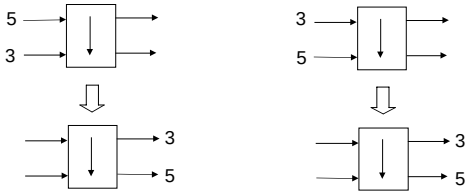
- The method shown is O(N) and requires N processors.
- Also, the number of items to be sorted is limited by the number of processors.
- As soon as a sequence clears the first processor, the next sequence can be started, so N-1 processors can be kept busy.

Pipelinable, Parallel, Sorting Methods

- Sorting networks were first proposed by Kenneth Batcher in the 1960's.
- They are networks constructed from a number of simple devices, called **comparators**.
- They are discussed in 9.2.7 and 9.2.8. We discuss them here with pipelining.

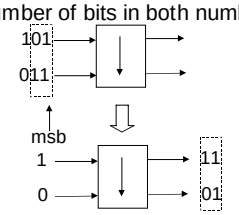
Sorting Network Comparator

- A single comparator inputs two numbers and outputs them in sorted order.



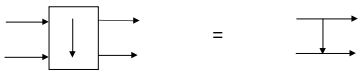
Sorting Network Comparator

- If desired, a comparator *can* be pipelined at the *bit-level*, as a finite-state machine accepting inputs MSB first (assuming the same number of bits in both numbers:

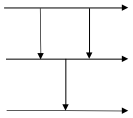


Sorting Networks

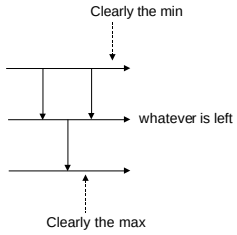
- Comparator abstraction (for drawing networks)



- A network that sorts 3 numbers



Analysis



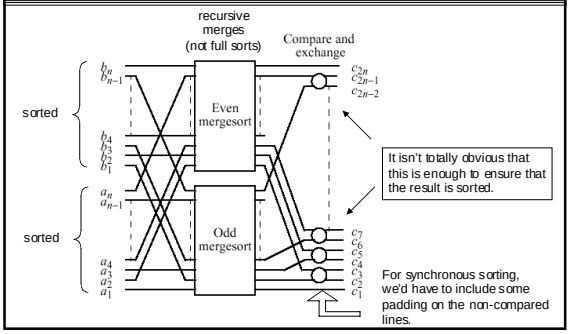
Exercises

- Construct a sorting network for 4 numbers
- Construct a sorting network for 8 numbers

General Constructions

- Odd-Even Merging
  - Assume 2n elements to be sorted.
  - Split the elements into two groups.
  - Sort each half recursively.
  - Merge the odd-indexed components of the result. (parallel)
  - Merge the even-indexed components of the result. (parallel)
  - Merge the output of the merges. (can be done in 1 stage)

Merging



Timing Analysis of Sorting by Odd-Even Merging

- Let S(n) = time to sort n elements
- Let M(n) = time to merge n/2 with n/2 elements
- Recurrences:
  - S(1) = 0
  - S(2n) = S(n) + M(2n)
  - M(2) = 1
  - M(2n) = M(n) + 1
- assuming we don't charge for splitting into 2 groups

Timing Analysis of Sorting by Odd-Even Merging

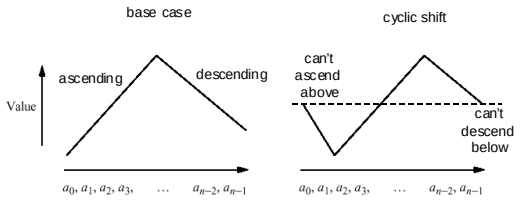
  - M(2) = 1
  - M(2n) = M(n) + 1
- Therefore
  - $M(2^n) = n-1$
  - $S(2^n) = S(2^{n-1}) + n-1$
- Giving $S(2^n) = (n-1) + (n-2) + \ldots + 1$

or $S(N) = O(\log^2 N)$

A Different Construction

- Bitonic sorting
- Define a bitonic sequence to be one in which either:
  - There is a strictly ascending portion, followed by a strictly descending portion, OR
  - Any cyclic shift of a bitonic sequence.

Bitonicity

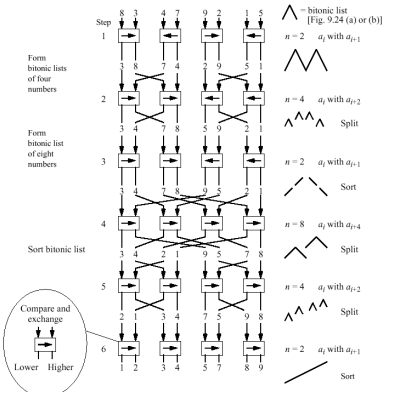


Bitonic Sorting

- Two *sorted* sequences can be made into a bitonic sequence by reversing the second one and abutting them.

Bitonic Sorting

- A bitonic sequence can be sorted by the following scheme:
  - Do compare-exchanges between elements j and j+n/2 for j = 1, 2, …, n/2
  - This gives two sequences that:
    - are bitonic
    - all elements of one are <= all elements of the other
  - Repeating this scheme recursively with the resulting two bitonic sequences gives a sorted sequence



Exercise

- Comment on the pipelinability of bitonic sorting
- Analyze the time taken for bitonic sorting