Partitioning & Divide-and-Conquer Strategies

- PP, chapter 4

Divide-and-Conquer

- Divide problem into 2 parts
- Sub-divide the parts, etc.
- Continue until?
  - All processors occupied

Divide-and-Conquer

- Divide problem into 2 parts
- Sub-divide the parts, etc.
- Continue until?
  - All processors occupied
  - Remaining problem not large enough to sub-divide (to approach cost-optimality)

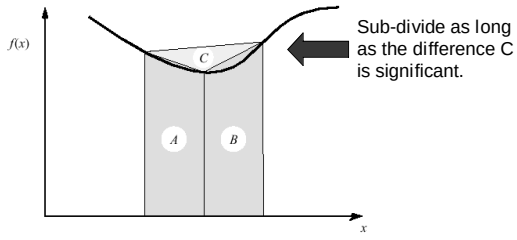
Divide-and-Conquer

- Divide problem into 2 parts
- Sub-divide the parts, etc.
- Continue until?
  - All processors occupied
  - Remaining problem not large enough to sub-divide (to approach cost-optimality)
  - Sufficient precision achieved

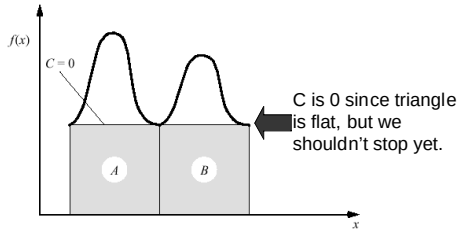
Sufficient-Precision Example

- Adaptive quadrature:
  - Like numeric integration problem discussed earlier
  - Instead of a fixed number of sub-divisions, divide where function is “less flat”.

Adaptive Quadrature



False Termination Possibility



C is 0 since triangle is flat, but we shouldn't stop yet.

Divide & Conquer in rex Primitive: sow

- Starts a parallel *thread* to evaluate an expression
- Returns a *seed* (aka *future*): an object that can be handled while its result is being computed

```
rex >   x = sow( expensiveFunction(args) );
```

Task Generation = sow + recursion

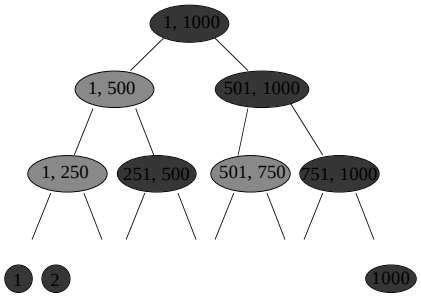
```
// parallel factorial, for illustration only

pfac(N) = product(1, N);

product(M, N) => M > N ? 1;

product(M, N) =>
      H = (M + N) / 2,              // midpoint
      K = SOW(product(M, H)),       // lower half
      L = product(H+1, N),          // upper half
      K * L;                        // result
```

Parallel factorial Execution



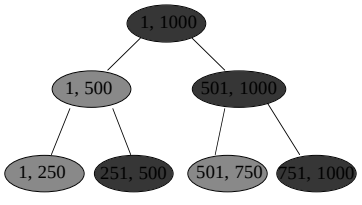
Grain-size control

```
pfac(N, pieces) =
     N < pieces ?
       fac(1, N) : product(1, N, pieces);

product(M, N, pieces) =>
     pieces <= 1 ? fac(M, N);          // use built-in

product(M, N, pieces) =>
      H = (M + N) / 2,                   // midpoint
      K = SOW(product(M, H, pieces/2)), // lower half
      L = product(H+1, N, (pieces+1)/2), // upper half
      K * L;                             // result
```

Execution with Grain Control



Larger grain leaves, computed sequentially

Parallel Mapping in rex

```
pmap( F, [x0, x1, x2, x3, ....] ) ==>
        [ F(x0), F(x1), F(x2), F(x3), .... ]
```

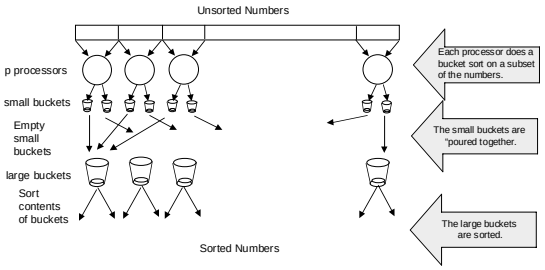
evaluated concurrently in different threads

There is also a set of primitives in rex that facilitate Linda implementation Linda construction in rex.

Natural Divide-and-Conquer Examples

- Tree search:
  - Looking for a node satisfying a certain property
  - Oct-trees: maybe represent 3-d objects or images
- Sorting:
  - Bucket sort
  - Quicksort
  - Merge sort

PP'S Parallel Bucket Sort



Quicksort

Merge Sort

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

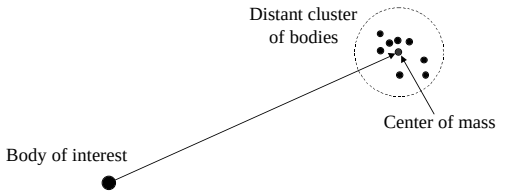
- The computation of the acceleration for a single point is O(N).
- The computation for N points is $O(N^2)$.
- This computation must be repeated each time step.
- For large N, this can be significant.

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

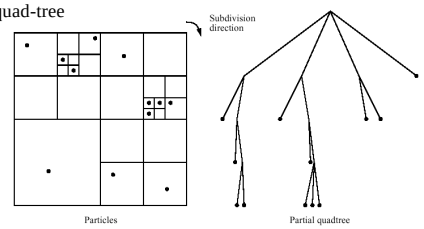
If a cluster of bodies is a long distance away from a given point, the effect of the entire cluster on the point is well-approximated by a composite mass located at the center of mass of the cluser.



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

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

quad-tree

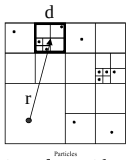


Assumption

- In the following, we make the assumption that the points are distributed evenly enough that the height of the quad- or oct-tree is bounded by log N.
- There might be some fixed precision, b, of bits that are used to index the subtrees, in which case log N could be replaced with b.

Barnes-Hut Algorithm

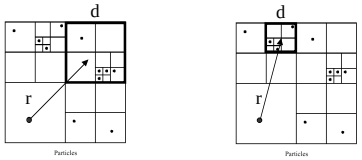
Each cell is approximated, for *distant* points, by the contained mass located at its center of gravity.
A criterion is needed for determining what is *distant.*



Let **d** be the dimension of one side of a box.
Let **r** be the distance of this point 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

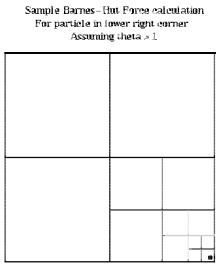
- On the preceding slides we indicated how the accelerations can be computed for one state.
- The particles will be in *different positions* in the next time step.
- Therefore the tree will need to be reconstructed on ***every*** time step.

Complexity of One Time-Step of Barnes-Hut Algorithm

- The height of the tree was assumed to 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).

Justified next page

Justification for Force-Computation Bound



Number of cells examined = 3*depth of point ∈ O(log N)

Pseudo-code for tree construction

(source: http://www.cs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html)

```
procedure QuadtreeBuild
  Quadtree = {empty}
  For i = 1 to n                ... loop over all particles
    QuadInsert(i, root)   ... insert particle i in quadtree
  end for
  ... at this point, the quadtree may have some empty
  ... leaves, whose siblings are not empty
  Traverse the tree (via, say, breadth first search),
    eliminating empty leaves
```

```
procedure QuadInsert(i,n)
  ... Try to insert particle i at node n in quadtree
  ... By construction, each leaf will contain either
  ... 1 or 0 particles
  if the subtree rooted at n contains more than 1 particle
    determine which child c of node n particle i lies in
      QuadInsert(i,c)
  else if the subtree rooted at n contains one particle
    ... n is a leaf
    add n's four children to the Quadtree
    move the particle already in n into the child
      in which it lies
    let c be child in which particle i lies
    QuadInsert(i,c)
  else if the subtree rooted at n is empty
    ... n is a leaf
    store particle i in node n
  endif
```

Pseudo-code for force computation

(source: http://www.cs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html)

```
... For each particle, traverse the tree
    ... to compute the force on it.
    For i = 1 to n
        f(i) = TreeForce(i,root)
    end for

    function f = TreeForce(i,n)
        ... Compute gravitational force on particle i
        ... due to all particles in the box at n
        f = 0
        if n contains one particle
            f = force computed using formula (*) above
        else
            r = distance from particle i to
                center of mass of particles in n
            D = size of box n
            if D/r < theta
                compute f using formula (*) above
            else
                for all children c of n
                    f = f + TreeForce(i,c)
                end for
            end if
        end if
```

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 log N) point acceleration computation
  - 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

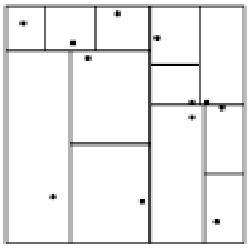
- What if we can't ignore communication costs (e.g. on distributed-memory system)?
  - One problem is **load-balancing**:
    - How do we distribute the computational work onto p processors so that each one is approximately equally busy?
      - finding outer bounds of set of particles
      - tree-construction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

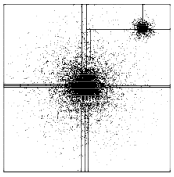
This is *separate* from the partitioning used in the quad-tree.

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



Orthogonal Recursive Bisection

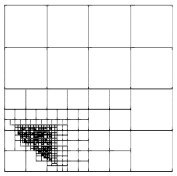
Example: Decomposition Resulting from Orthogonal Recursive Bisection of a System with Two Galaxies



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.