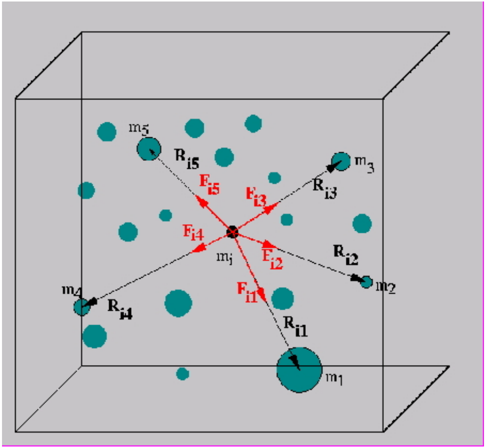
N-body Problems

Victor Eijkhout

Fall 2022

Summing forces



Particle interactions

for each particle i

  for each particle j

    let $\bar{r}_{ij}$ be the vector between i and j;

    then the force on i because of j is

      $f_{ij} = -\bar{r}_{ij}\frac{m_i m_j}{|r_{ij}|}$

    (where m_i, m_j are the masses or charges) and

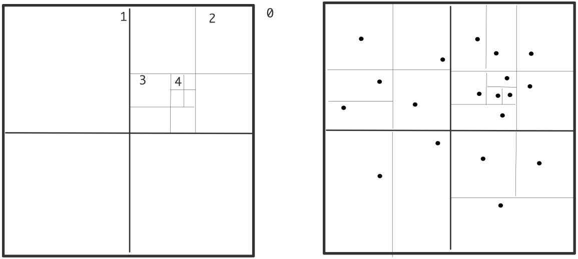
    $f_{ji} = -f_{ij}$.

  Sum forces and move particle over Δt

Complexity reduction

- Naive all-pairs algorithm: $O(N^2)$
- Clever algorithms: $O(N \log N)$, sometimes even $O(N)$
- Octtree algorithm: Barnes-Hut

Octtree



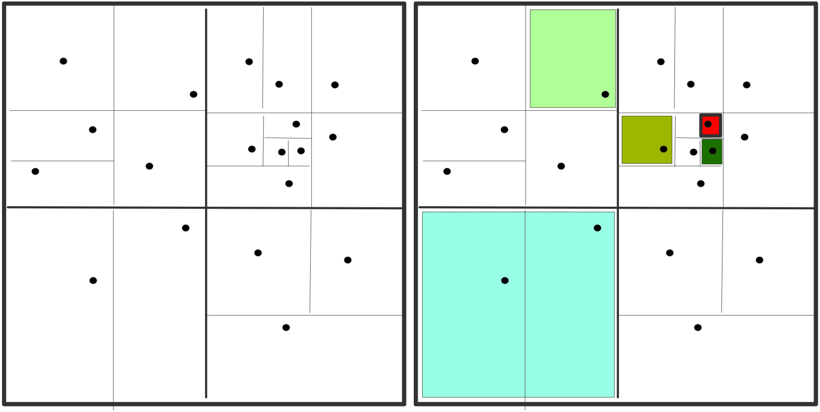
Dynamic octree creation

```
Procedure Quad_Tree_Build
    Quad_Tree = {empty}
    for j = 1 to N  // loop over all N particles
         Quad_Tree_Insert(j, root)  // insert particle j in QuadTree
    endfor
    Traverse the Quad_Tree eliminating empty leaves

Procedure Quad_Tree_Insert(j, n) // Try to insert particle j at node n in
    if n an internal node              // n has 4 children
        determine which child c of node n contains particle j
        Quad_Tree_Insert(j, c)
   else if n contains 1 particle   //  n is a leaf
        add n's 4 children to the Quad_Tree
        move the particle already in n into the child containing it
        let c be the child of n containing j
        Quad_Tree_Insert(j, c)
    else                                              //  n empty
        store particle j in node n
    end
```

Octree algorithm

- Consider cells on the top level
- if distance/diameter ratio small enough, take center of mass
- otherwise consider children cells



Masses calculation

```
// Compute the CM = Center of Mass and TM = Total Mass of all the particl
( TM, CM ) = Compute_Mass( root )

function ( TM, CM ) = Compute_Mass( n )
  if n contains 1 particle
     store (TM, CM) at n
     return (TM, CM)
  else       // post order traversal
             // process parent after all children
     for all children c(j) of n
           ( TM(j), CM(j) ) = Compute_Mass( c(j) )
     // total mass is the sum
     TM = sum over children j of n: TM(j)
     // center of mass is weighted sum
     CM = sum over children j of n: TM(j)*CM(j) / TM
     store ( TM, CM ) at n
     return ( TM, CM )
```

Force evaluation

```
// for each particle, compute the force on it by tree traversal
for k = 1 to N
    f(k) = TreeForce( k, root )
    // compute force on particle k due to all particles inside root

function f = TreeForce( k, n )
    // compute force on particle k due to all particles inside node n
    f = 0
    if n contains one particle  // evaluate directly
        f = force computed using formula on last slide
    else
        r = distance from particle k to CM of particles in n
        D = size of n
        if  D/r  < theta  // ok to approximate by CM and TM
             compute f
        else              // need to look inside node
             for all children c of n
                   f = f + TreeForce ( k, c )
```

Complexity

- Each cell considers ‘rings’ of equi-distant cells
- but at doubling distance
- $c \log N$ cells to consider for each particle
- $N \log N$ overall

Computational aspects

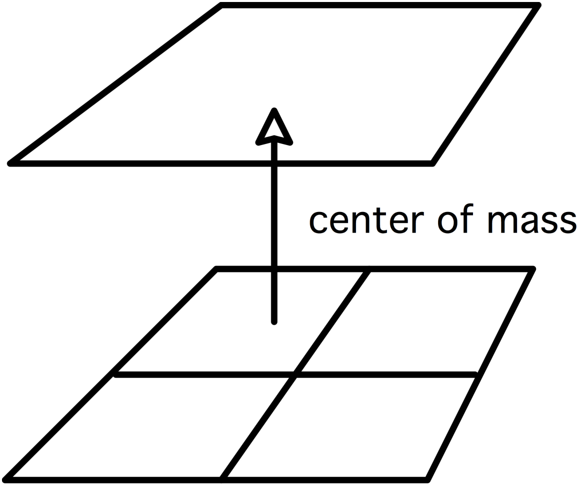
- After position update, particles can move to next box: load redistribution
- Naive octree algorithm is formulated for shared memory
- Distributed memory by using inspector-executor paradigm

Step 1: force by a particle

for level ℓ from one above the finest to the coarsest:

for each cell c on level ℓ

let $g_c^{(\ell)}$ be the combination of the $g_i^{(\ell+1)}$ for all children i of c


center of mass

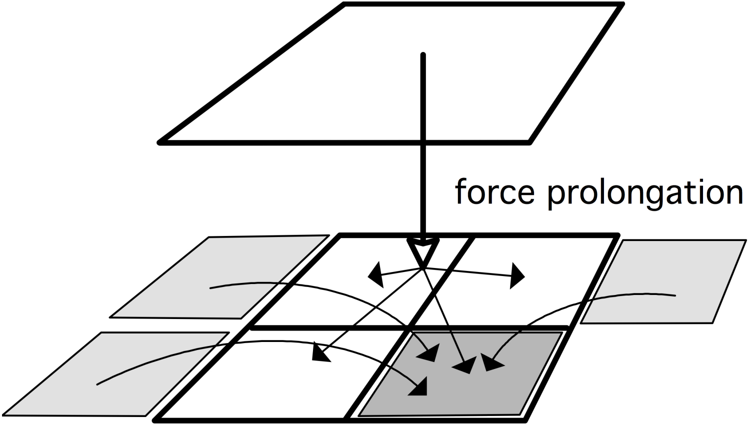
Step 2: force on a particle

for level ℓ from one below the coarses to the finest:

for each cell c on level ℓ:

let $f_c^{(\ell)}$ be the sum of

1. the force $f_p^{(\ell-1)}$ on the parent p of c, and
2. the sums $g_i^{(\ell)}$ for all i on level ℓ that satisfy the cell opening criterium


force prolongation

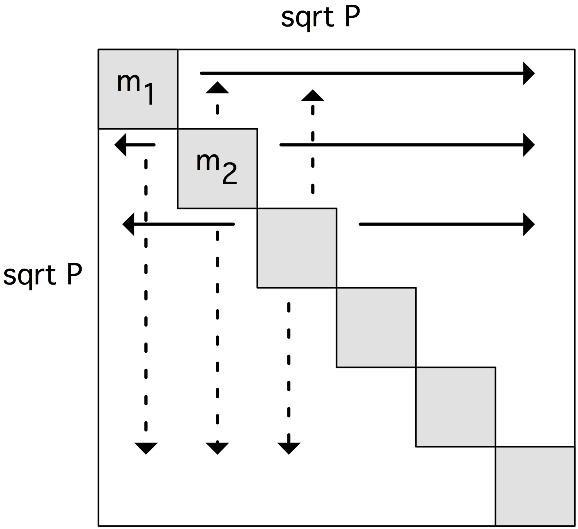
- Center of mass calculation and force prolongation are local
- Force from neighbouring cells is a neighbour communication
- Neighbour communication can start before up/down tree calculation is finished: latency hiding

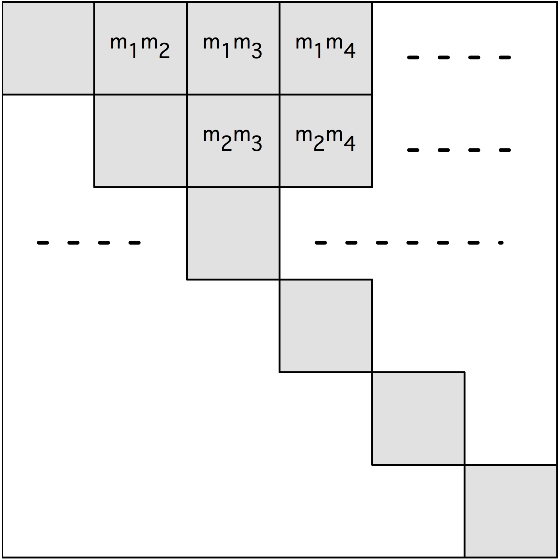
All-pairs methods

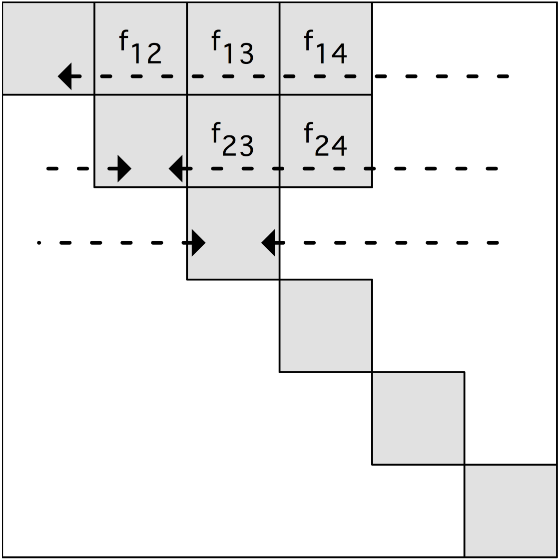
- Traditional algorithm: distribute particles, for each particle gather and update compute
- Problem: allgather has $O(N)\beta$ cost
- does not go down with P, so does not scale weakly

1.5D calculation

- Better algorithm: use $\sqrt{P} \times \sqrt{P}$ processor grid,
- Divide particles in bins of $N/\sqrt{P}$
- Processor (i, j) computes interaction of boxes i and j:


sqrt P
sqrt P
m_1
m_2


m1m2
m1m3
m1m4
m2m3
m2m4


f_{12}
f_{13}
f_{14}
f_{23}
f_{24}

- Better algorithm: use $\sqrt{P} \times \sqrt{P}$ processor grid,
- Divide particles in boxes of $M = N/\sqrt{P}$
- Processor (i, j) computes interaction of boxes i and j:
- this requires broadcast (for duplication) and reduction (for summing) in processor rows and columns
- Bandwidth cost $\beta N/\sqrt{P}$ which is M: scalable.