Poisson’s Equation and Fast Method

Poisson's Euqation and its Fundamental Solution

$$\nabla^2 \phi = -\rho$$

$$\phi(x \to \infty) = 0$$

$$\phi(x) = \int_{\Omega} \frac{\rho(y)}{4\pi \|x-y\|_2} \, dy$$

$$\phi_j = \sum_{j=1}^{N} \frac{m_j}{4\pi R_{ij}}$$

For Example: the Gravitational Problem

$$f(x) = \nabla \phi$$

$$f(x) = -\sum_{j=1, j \neq i}^{N} \nabla_{x_i} \left(\frac{\rho_j v_j}{4\pi \|x_i - x_j\|_2} \right)$$

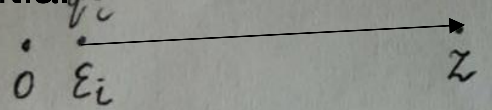
$$f_i = -\sum_{j=1, j \neq i}^{N} \frac{\rho_j v_j \, (x_i - x_j)}{4\pi \|x_i - x_j\|_2^3}$$

Given N particles and M evaluation position, direct computation requires O(NM) time!

Introduction to Fast Summation

2D. Complex Number. $\phi(z)=Re\left(\sum_j q_j \log(z-z_j)\right)$

Consider a source and its potential



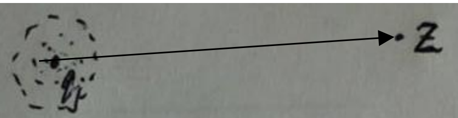
Apply Taylor expansion gives:

$$\phi(z)=q_i\log(z-\varepsilon_i)$$

$$=q_i\log z-\sum_{k=1}^{p}\frac{q_i\varepsilon_i^k}{kz^k}$$

$\Rightarrow$

Multipole Expansion:



$$\phi(z)=\sum_j q_j\log(z-z_j)$$

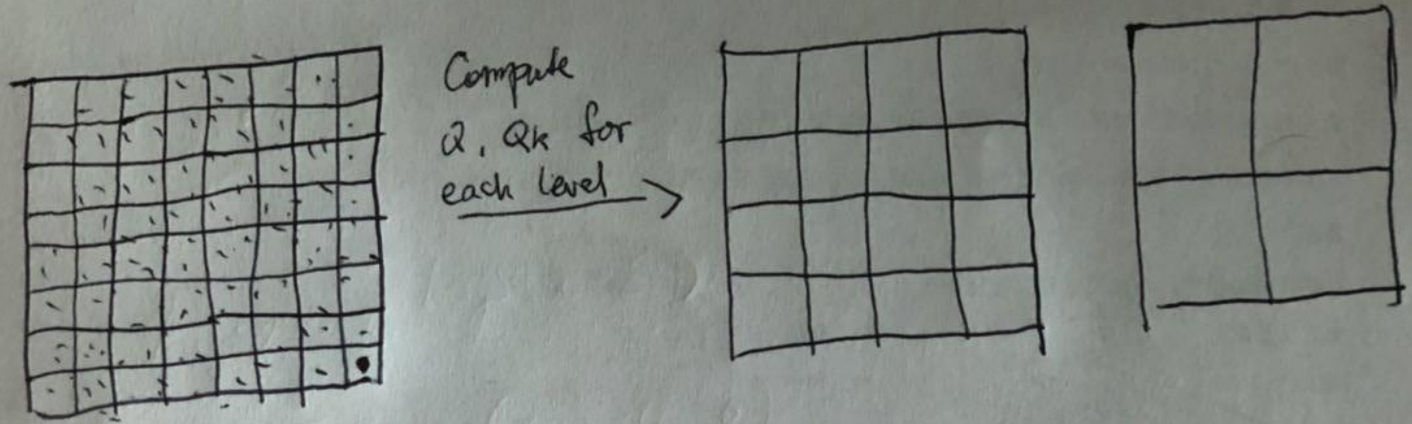
$$=\Big(\sum_j q_j\Big)\log(z)-\sum_{k=1}^{p}\frac{\sum_j \frac{q_j z_j^k}{k}}{z^k}$$

$$Q=\sum_j q_j.$$

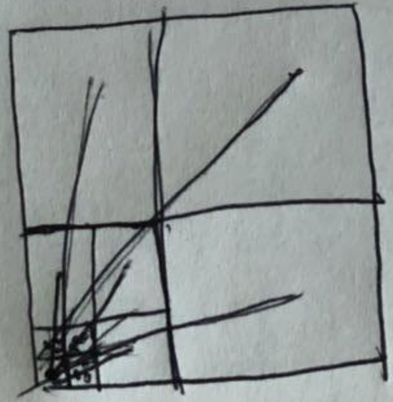
$$Q(k)=-\sum_j\frac{q_j z_j^k}{k}$$

$$\phi(z)=Q\log z+\sum_{k=1}^{p}\frac{Q_k}{z^k}$$

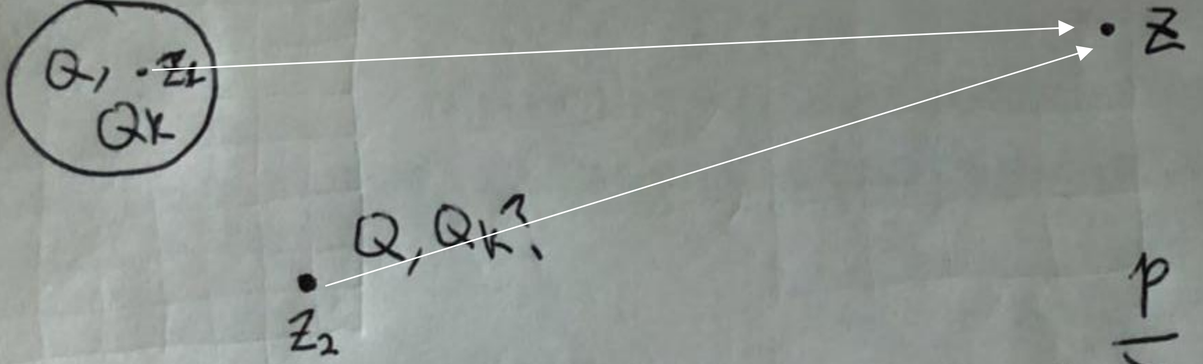
N logN Algorithm: Tree Code



Eval:



Take one step further:
If we know M-Expansion at z1 (M1={z1, Q, Q_k} ,
What is the M-Expansion at z2 (M2={z2, Q2, Q2_k}?
We want to obtain the coefficients from M1, not q_i's



$$\phi(z) = Q \log(z - z_2) + \sum_{k=1}^{p} \frac{b_k}{z^k}$$

Recall: $\phi(z) = Q \log z + \sum_{k=1}^{p} \frac{Q_k}{z^k}$.

b_k is a generalization of Q_k:

$$b_k = \underbrace{-\frac{Q(z_1-z_2)^k}{k}} + \underbrace{\sum_{i=1}^{k} Q_i (z_1-z_2)^{k-i}\binom{k-1}{i-1}}_{\text{Rest of terms}}$$

Recall: $\dfrac{q_j (z_j - c)^k}{k}$

$$\phi(z) = Q \log(z-c) + \sum_{k=1}^{p} \frac{b_k}{(z-c)^k}$$

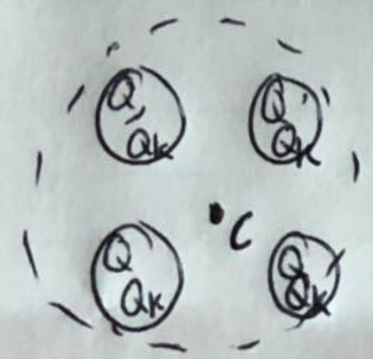
View Source as Multipole:

$$Q = q_i ,$$

Reveal "Multipole Expansion"

$$\phi(z) = Q\log(z-c) + \sum_{k=1}^{p} \frac{\sum_j \frac{-q_j (z_j - c)^k}{k}}{(z-c)^k}$$

From "Multipoles":



compute b_k with "Rest of terms"

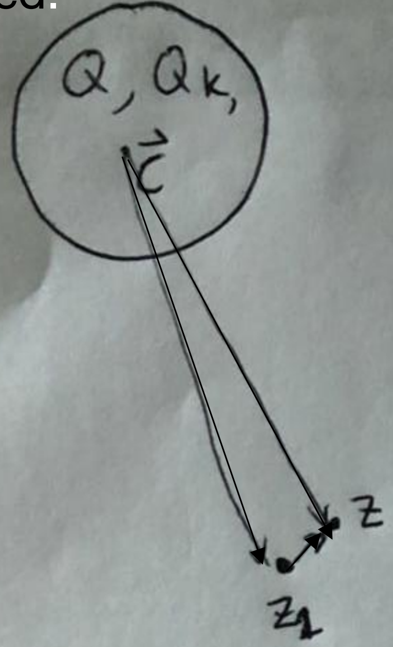
M2M Transform

```
struct Multipole{
        vec2 center;
        complex   q[p];
};
//source charge is a special Multipole,
//with center = charge pos, q0 = qi, q1…q4 =0

Multipole M2M(std::vector<Multipole> &qlist)
{
      Multipole res;
      res.center = weightedAverageof(qlist[i].center*qlist[i].q[0]);
      q[0] = sumof(qlist[i].q[0]);

      for(k=1:p){
              res.q[k]=0;
              for(j=0:qlist.size()) {
                    res.q[k] += computeBk(qlist[i]);
              }
      }
      return res;
}
```

Question:

If we know M-Expansion at c1 (M1={c1, Q, Q_k} ,

What is the polynomial at z1, so that potentials at neighbor z can be evaluated.



M2L Transform

$$\phi(z) = Q\log(z-c) + \sum_{k=1}^{p} \frac{b_k}{(z-c)^k}$$

$$= Q\log(z_1 - c + z - z_1) + \sum_{k=1}^{p} \frac{b_k}{(z_1 - c + z - z_1)^k}$$

$$= \underbrace{Q\log(z_1 - c) + \sum_{k=1}^{p} \frac{b_k}{(z_1 - c)^k}}_{\phi(z_1)}$$

$$+ \text{H.O.T}$$

$$\text{H.O.T.} = \sum_{l=1}^{p} b_l (z - z_1)^l$$

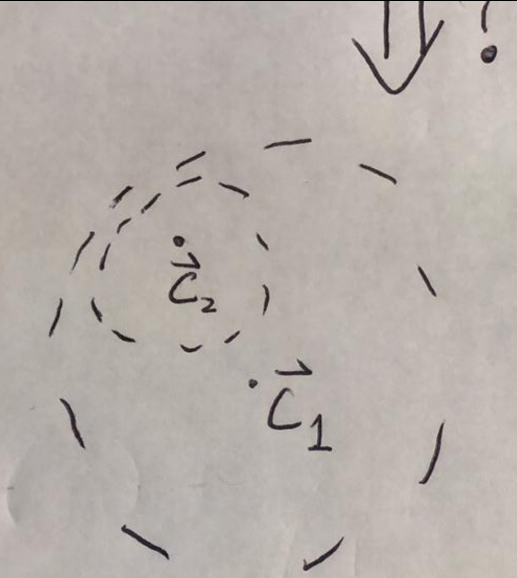
$$b_l = -\frac{Q}{l(c-z_1)^l} + \frac{1}{(c-z_1)^l} \sum_{k=1}^{p} \frac{b_k}{(z_1-c)^k} \binom{l+k-1}{k-1}$$

```
struct Localpole{
        vec2 center;
        complex  b[p];
};
```

Question:

If we know L-Expansion at c1 (L1={c1, B}),

What is the polynomial at c2, so that potentials at neighbor z around c2 can be evaluated.

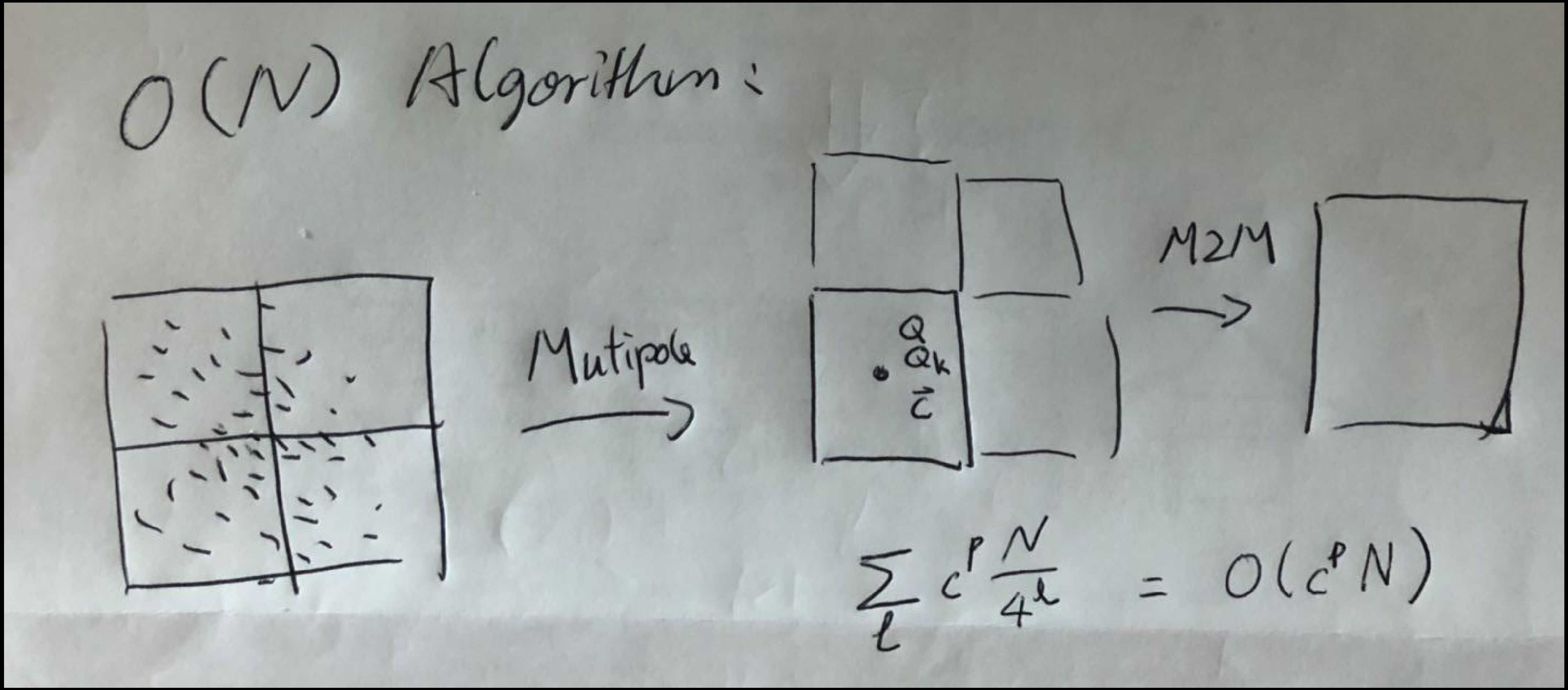


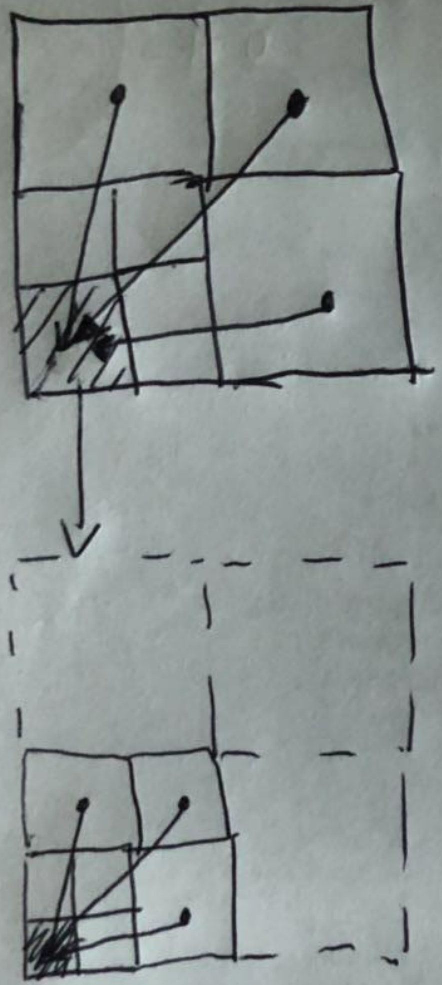
L2L
Transform

$$\sum_{k=0}^{p} a_k (z - z_0)^k \longrightarrow \sum_{k=0}^{p} b_k (z)^k$$

Honer scheme

Multipole Expansion : Coarsening
Localpole Expansion : Interpolation



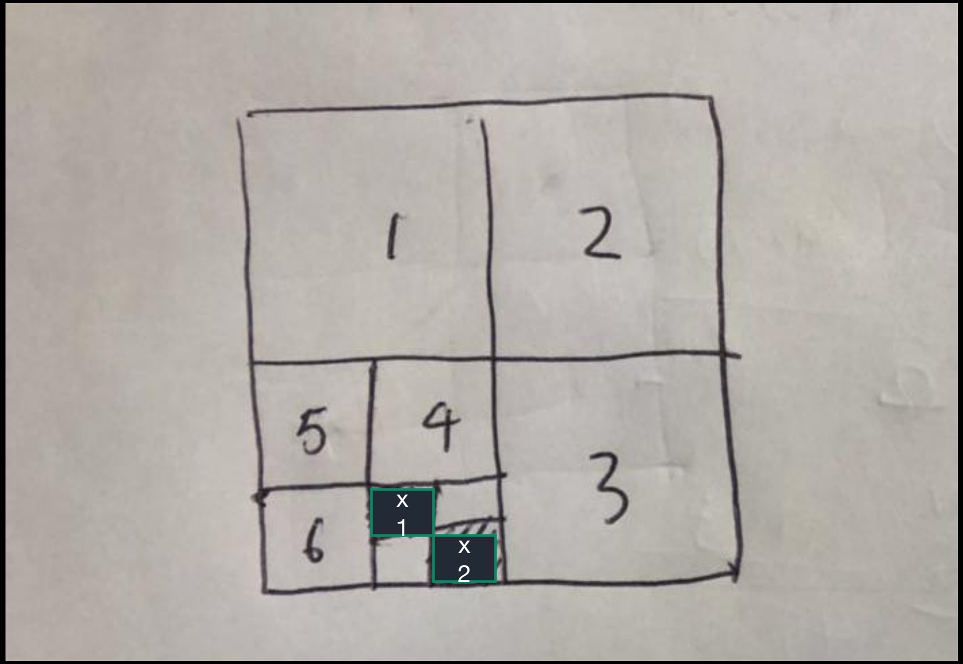


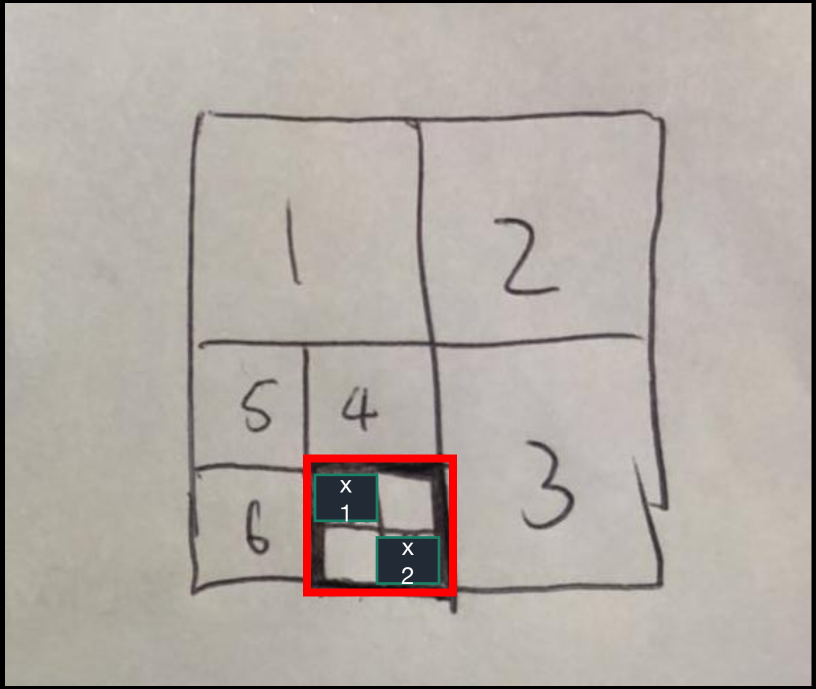
M2L

L2L + M2L

$$\sum_{\ell} c^p \frac{N}{4^\ell} = O(c^p N)$$

Tree code VS FMM





Phi(x1) = contribution from (node1, 2, 3, 4, 5, 6)
Phi(x2) = contribution from (node1, 2, 3, 4, 5, 6)

= contribution from (node1, 2, 3)

Phi(x1) = L2L from + contribution from (4, 5, 6)

Phi(x2) = L2L from + contribution from (4, 5, 6)

In 3D, the algorithm can be obtained via:

- Taylor expansion of the Green's function in Cartesian system
- Taylor expansion of the Green's function in Spherical coordinates

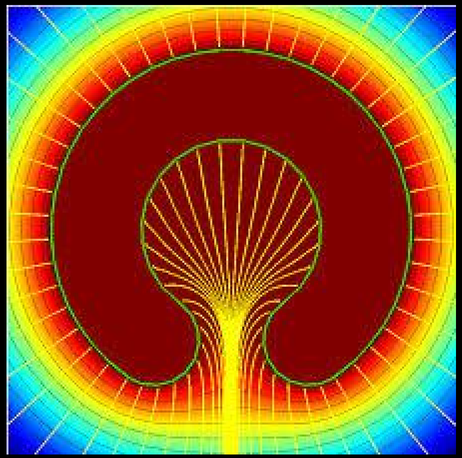
For more details, checkout:
https://math.nyu.edu/faculty/greengar/shortcourse_fmm.pdf

Many, Many important applications

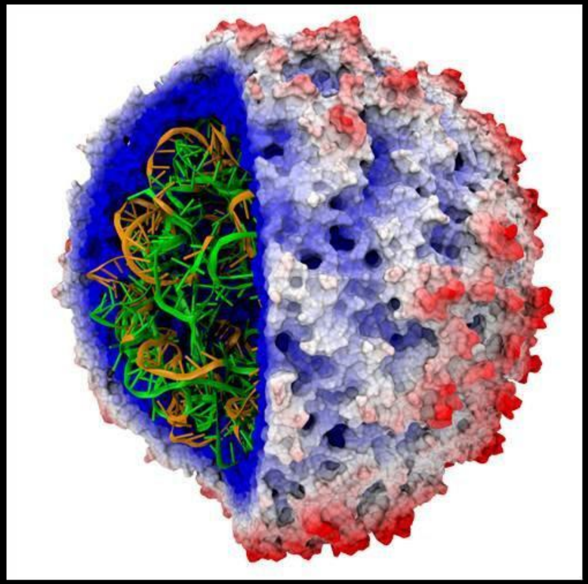
- Gravitational Force → Dark matter, cosmology…

Electrostatic

$$\nabla^2 \phi = \epsilon \rho$$
$$f = \nabla \phi$$



Major force between moleculars!!

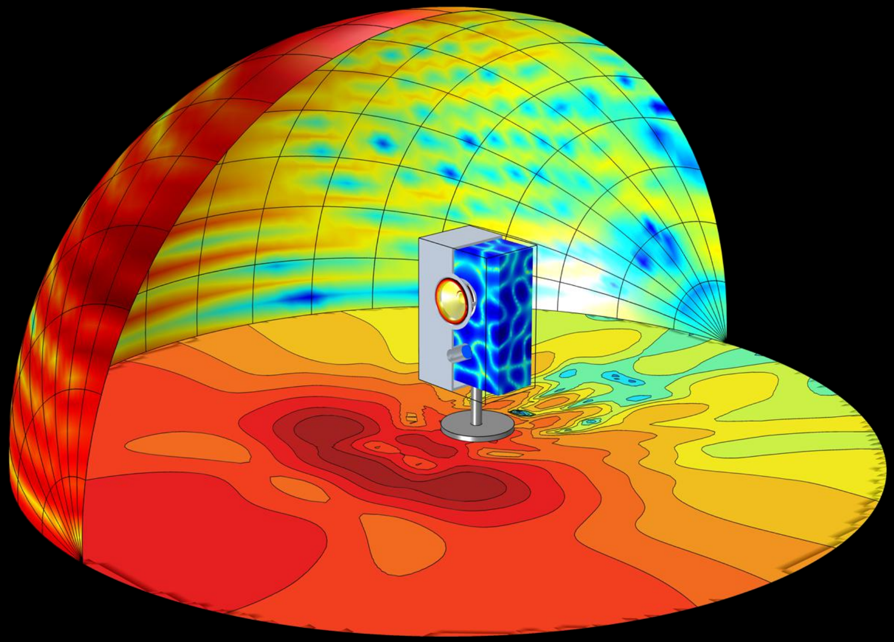


Cancer research
Drug Design
Virus Analysis

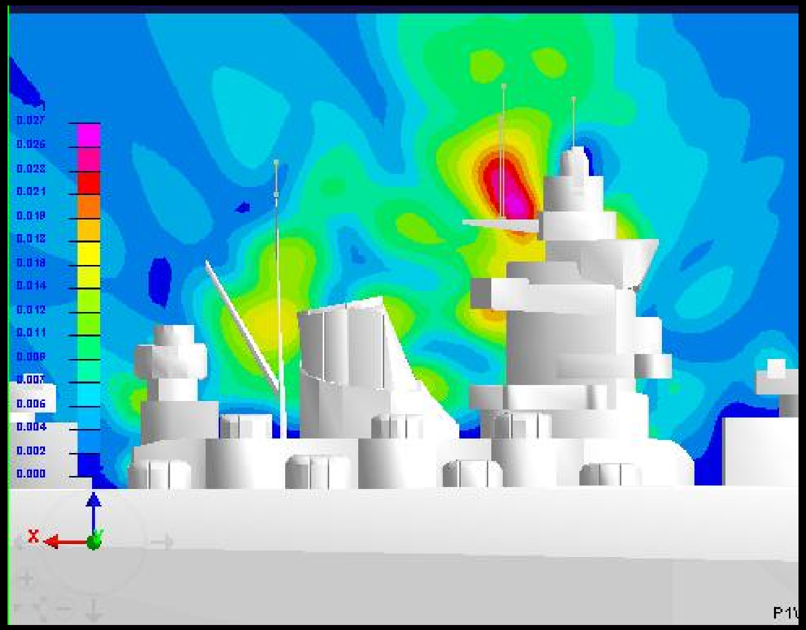
Helmholtz

$$\nabla^2\phi + k^2\phi = -f$$

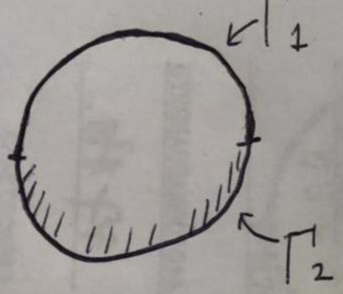
Acoustics



Electromagnetic



Boundary Element Method



$$\nabla^2 \phi = 0$$

$$q = \frac{\partial \phi}{\partial n} = f \text{ on } \Gamma_2$$

$$u = \phi = g \text{ on } \Gamma_1$$

Solve for u on Γ_2 and q on Γ_1, s.t.

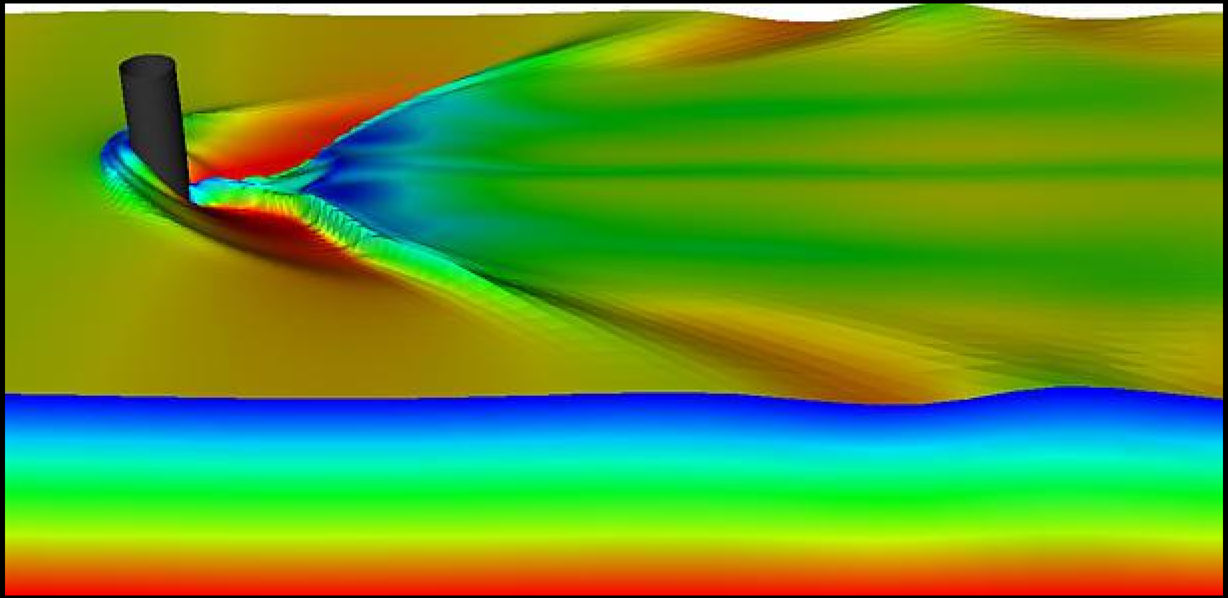
$$\frac{1}{2} u_i + \sum_{j=1}^{N} u_j \vec{n}_j \cdot \nabla \frac{1}{R_{ij}} dA_j = \sum_{j=1}^{N} q_j \frac{1}{R_{ij}} dA_j$$

PDE → Integral Equations

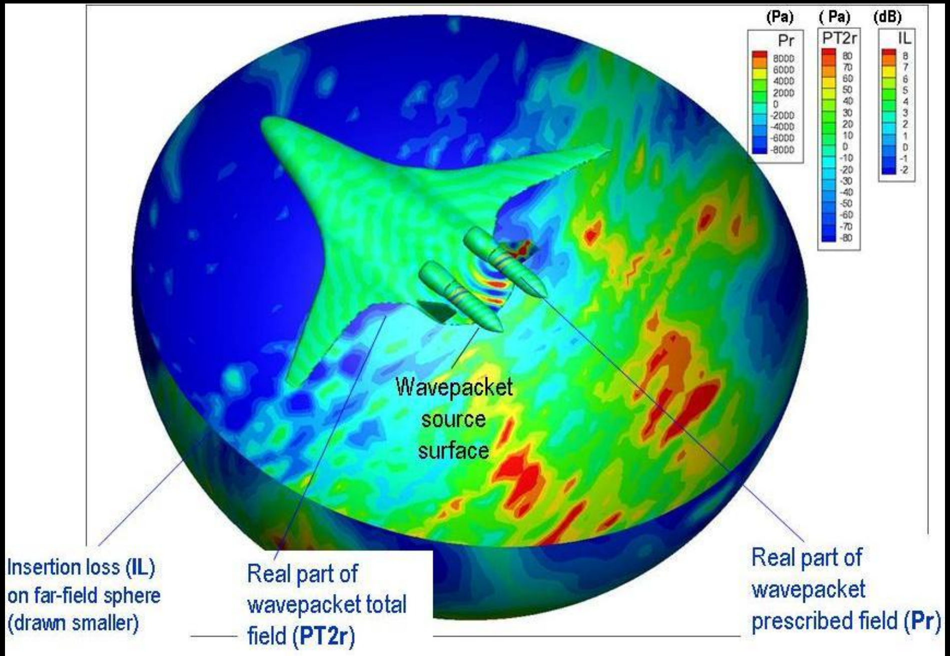
- Matrix is Dense!
- Condition number is Good!(usually converge in O(1) iterations)

Boundary Element Method

- In 2D
  - Full Domain is N^2, with Multigrid Methods, O(N^2) computation.
  - Boundary element has N elements, BiCGSTAB method converges in constant iteration, and each iteration took O(N^2) for Dense Matrix-vector!
- In 3D
  - Full Domain is N^3, with Multigrids, O(N^3)
  - Boundary element has N^2 elements, in total N^4 computation!
- With FMM replacing the matrix vector multiplication operation
  - O(N) in 2D
  - O(N^2) in 3D.
  - Semi-Analytic!



Large scale, deep wave



Sound Pressure

Vortex flow

$$u_i = \sum_{j=1, j\neq i}^{N} \frac{v_j \omega_j \times (x_i - x_j)}{4\pi \left\| x_i - x_j \right\|_2^3} \quad \Longleftrightarrow \quad \begin{aligned} \nabla^2 \psi &= -\omega \\ u &= \nabla \times \psi \end{aligned}$$

Type of summations

$$\nabla_x \phi(x) = \sum_{j=1}^{N} \frac{m_j \vec{d}_{ij}}{R_{ij}^3}$$

B.I.E: $\sum_{j=1}^{N} \vec{n}_j \cdot \nabla \left(\frac{1}{R_{ij}} \right) \sigma_j A_j$

Biot-savart $\sum_{j=1}^{N} \frac{\omega_j \times \vec{d}_{ij}}{R_{ij}^3} dv_j.$

Compute them with same routine

- Given Routine:
  - F = computeGradPhi(s); //returns gravity force using FMM from many source s

- B.I.E:
  - Let s1[i] = n[i].x*s[i], s2[i] = n[i].y*s[i], s3[i] = n[i].z*s[i];
  - F1 = computeGradPhi(s1), F2 = computeGradPhi(s2), F3 = computeGradPhi(s3)
  - Res = F1.x + F2.y + F3.z;
- Biot-Savart:
  - Your homework

Other fast summation methods:

- PPPM

PPPM: Combining PDE form and summation forms

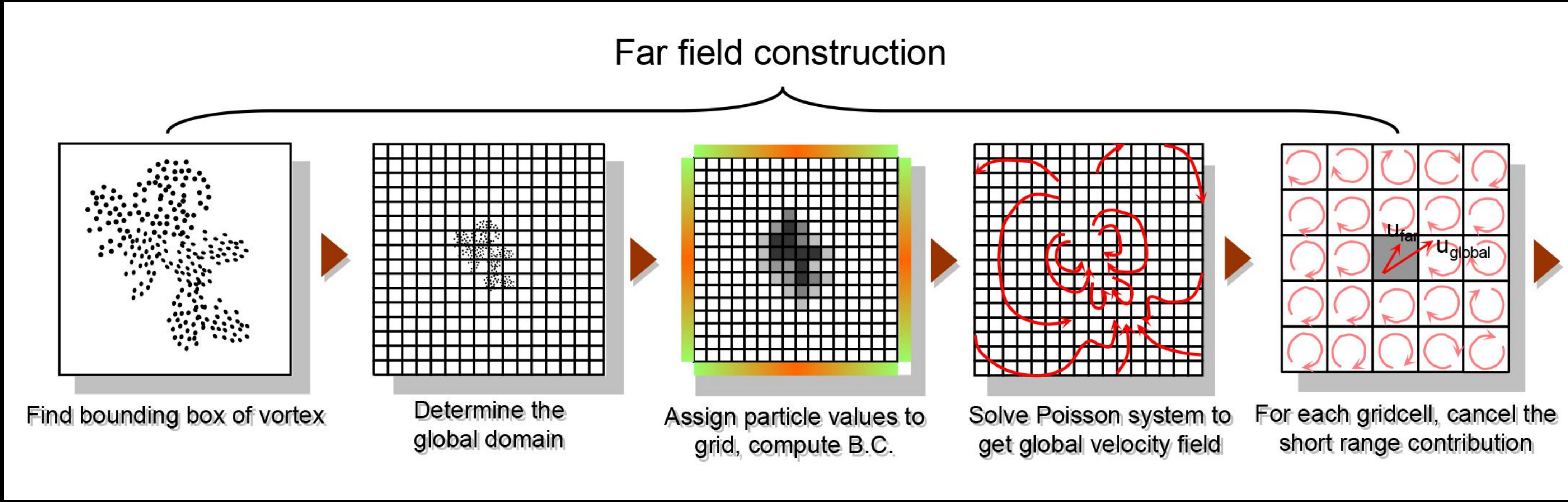
$$u_i = \sum_{j=1, j\neq i}^{N} \frac{v_j \omega_j \times (x_i - x_j)}{4\pi \|x_i - x_j\|_2^3} \quad \Longleftrightarrow \quad \begin{gathered} \nabla^2 \psi = -\omega \\ u = \nabla \times \psi \end{gathered}$$

$$f_i = -\epsilon \sum_{j=1, j\neq i}^{N} \frac{\rho_j v_j (x_i - x_j)}{4\pi \|x_i - x_j\|_2^3} \quad \Longleftrightarrow \quad \begin{gathered} \nabla^2 \phi = \epsilon \rho \\ f = \nabla \phi \end{gathered}$$

Direct summation for the turbulent part ⟺ Poisson's Equation for the smooth part

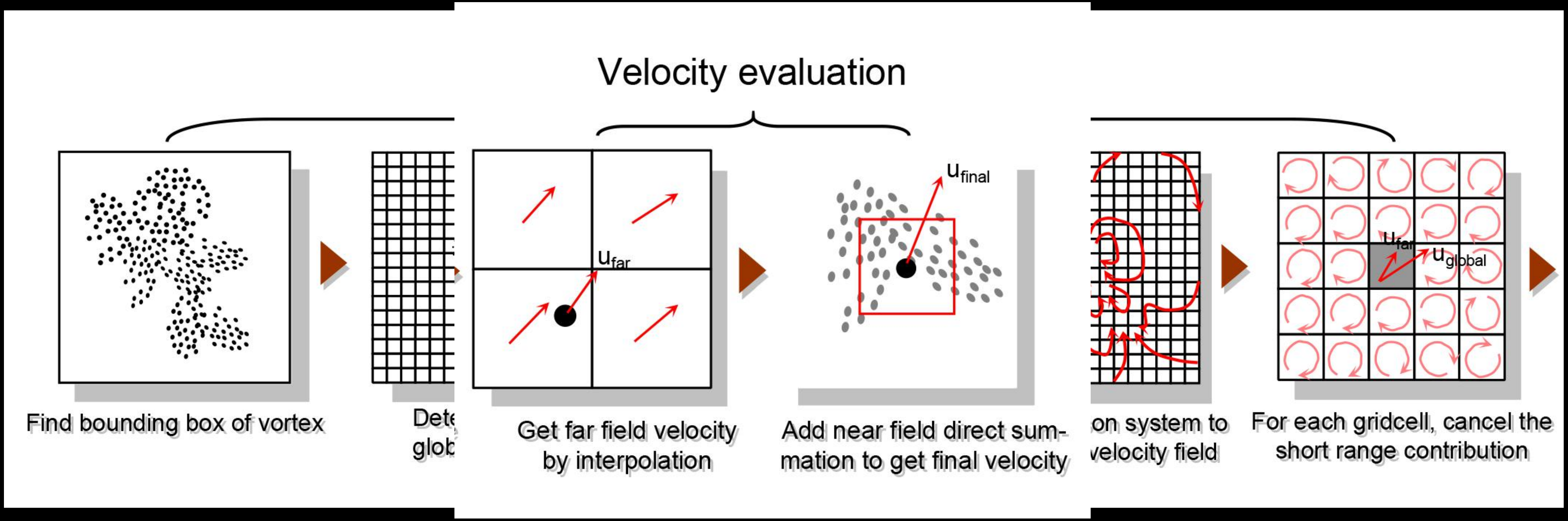
PPPM

- Fast solution uses near-far decomposition to get acceleration. Can we do similar thing on a particle-mesh setup?



PPPM

- Fast solution uses near-far decomposition to get acceleration. Can we do similar thing on a particle-mesh setup?



PPPM

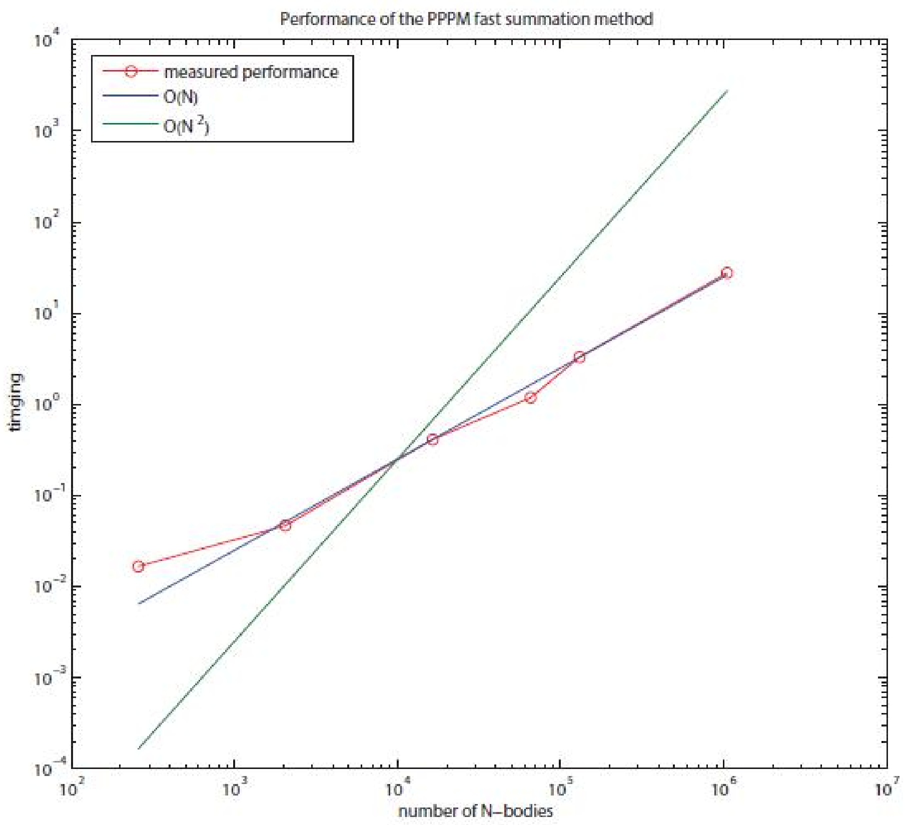


Figure 4: *Performance of the PPPM fast summation. Computation time grows linearly with the number of computational elements.*

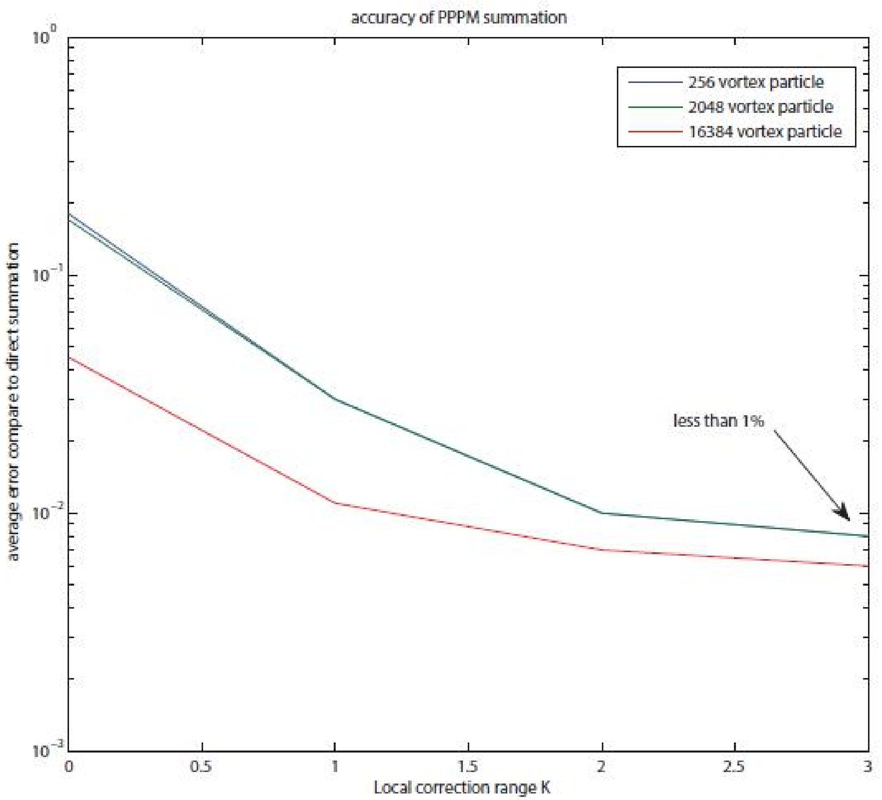


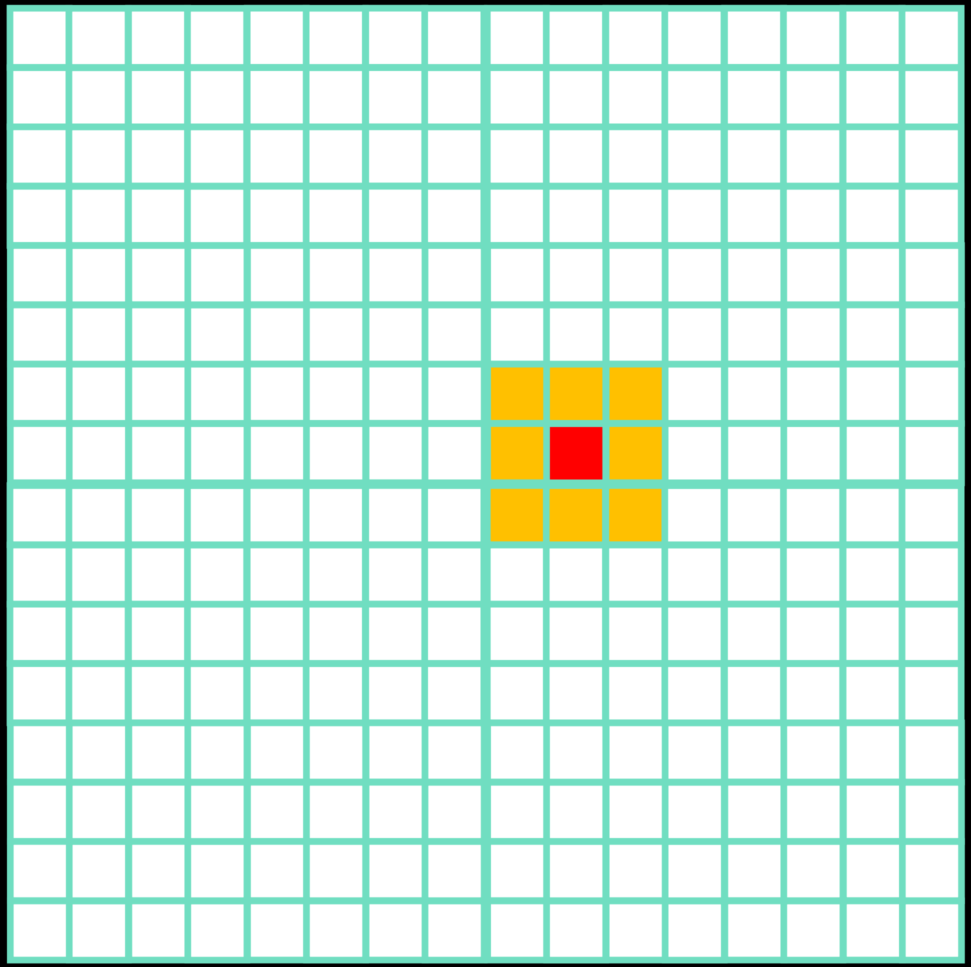
Figure 5: *Accuracy statistics of the PPPM fast summation.*

Local Correction

- In 3D, for a correction window of size K in each dimension, a local matrix of size $K^3 \times K^3$ can be precomputed to cancel the local influence from grid.

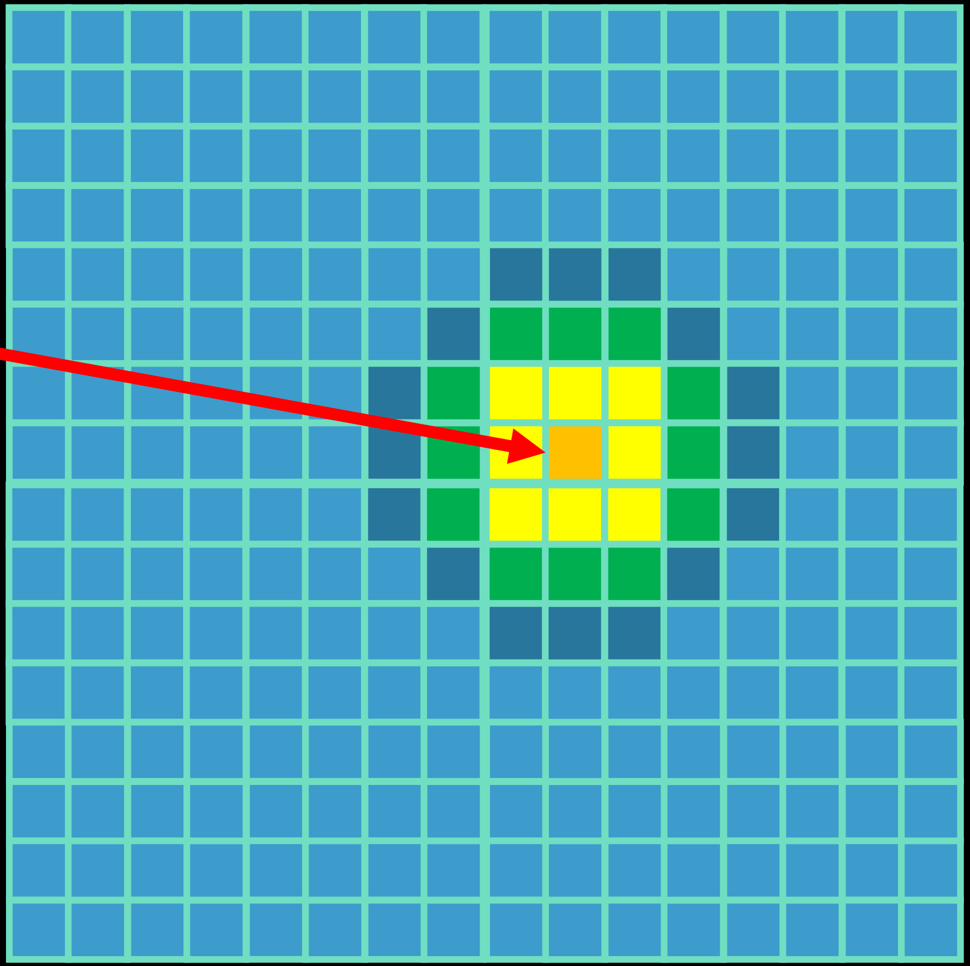
- T(N) = O(c K^6 N)

Local Correction



Local Correction

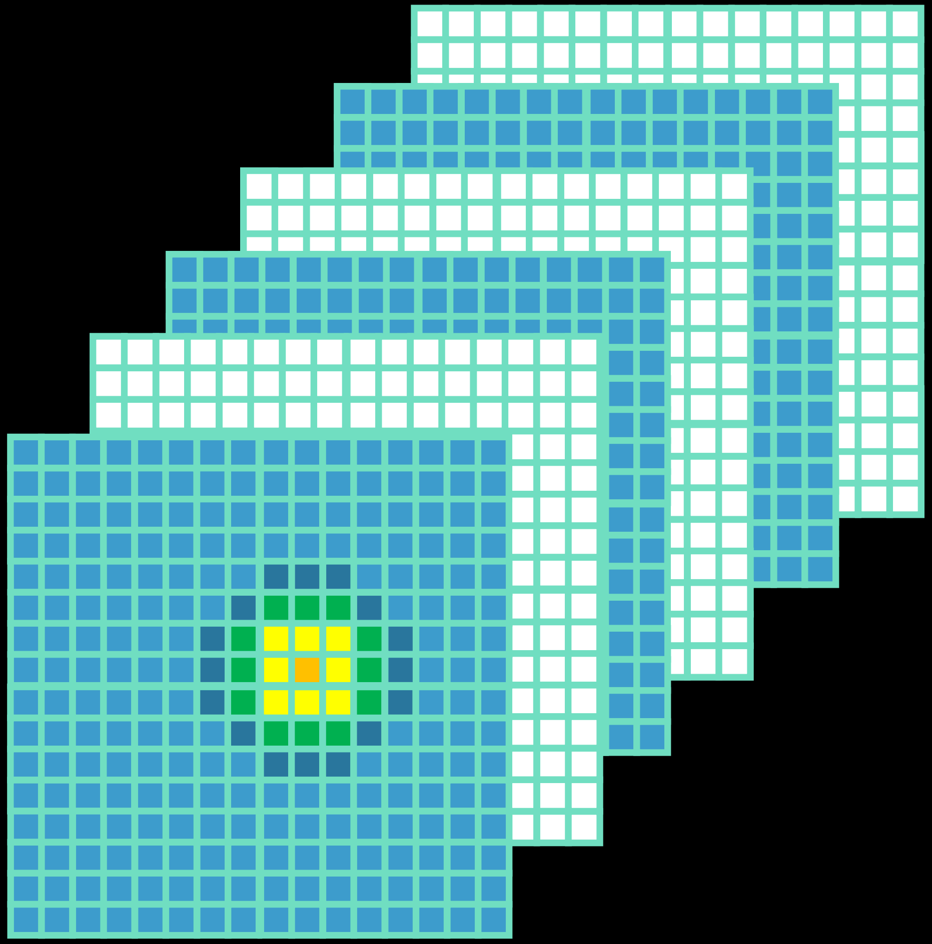
The influence made by neighbor cells.



Local Correction

- *The matrix inverse reveals how the center cell's value depends linearly on its neighbors(including itself).*

$$s_c = \sum_{j \in \eta} a_j r_j$$



PPPM in few lines

- w_bar = particle_to_grid(w_p);
- dw = w_p – interpolate(w_bar);
- Psi = Poisson.Solve(w_bar);
- v_smooth = curl(Psi);
- v_p = interpolate(v_smooth) + nearSum(dw);

Summarize

- Fast Summation Methods
  - FMM
  - PPPM
- Equations solved by Fast Summation Methods:
  - Poisson’s Equation
  - Laplace Equation
  - Helmholtz Equation
  - BEM
- Applications of Fast Summation Methods
  - Electrostatic::Molecular Dynamics::Cancer, drug design research
    Magnetics::Ship design
  - Acoustics::Urban planning, vehicle shape design, theatre design
  - Potential flow::aircraft, wave
  - Vortex method::turbulent flow