Fast direct methods for molecular electrostatics

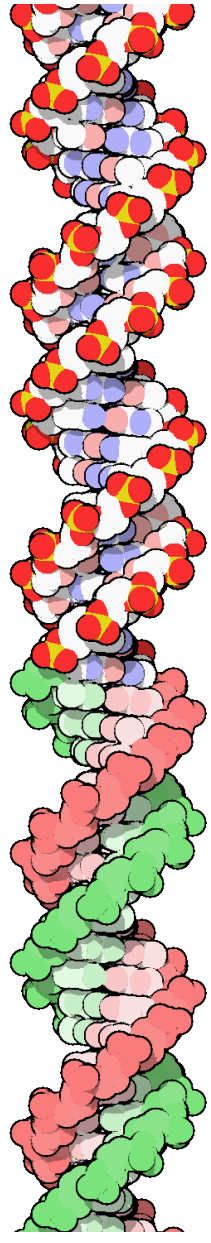
Kenneth L. Ho

Program in Computational Biology, NYU

COB Colloquium, May 2012

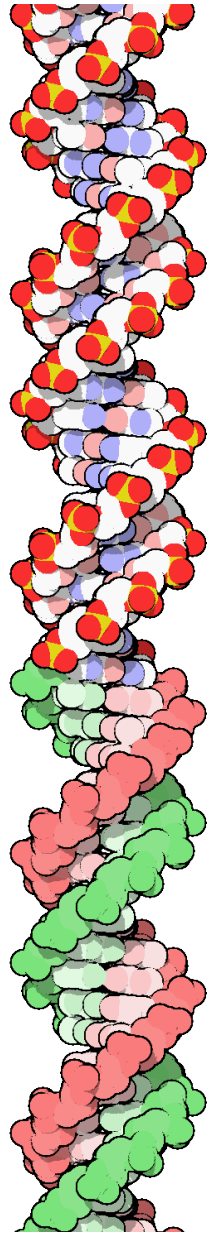
Suppose I give you a structure.

What can you tell me about its function?



Suppose I give you a structure.

What can you tell me about its function?
(What are the physics acting on it?)

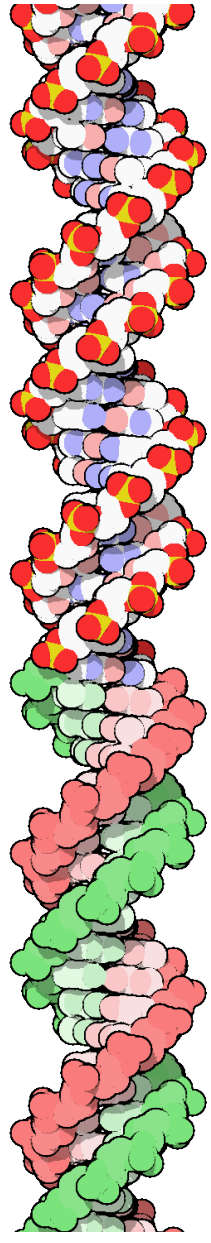


Suppose I give you a structure.

What can you tell me about its function?
(What are the physics acting on it?)

> *Electromagnetism is the force of chemistry.*
>
> Davis ME, McCammon JA (1990) *Chem Rev*

- Charge complementarity
- Conformation and dynamics
- Long-range steering
- Polarization and ionization



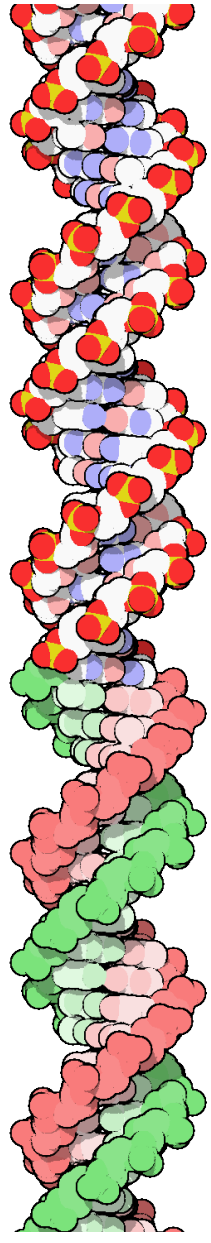
Suppose I give you a structure.

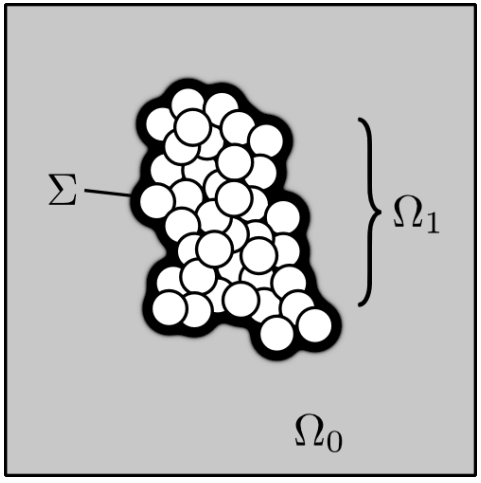
What can you tell me about its function?
(What are the physics acting on it?)

> *Electromagnetism is the force of chemistry.*
>
> Davis ME, McCammon JA (1990) *Chem Rev*

- Charge complementarity
- Conformation and dynamics
- Long-range steering
- Polarization and ionization

In this talk, we focus on electrostatics.





Molecule: discrete collection of charged atoms

Ω_0: solvent

Ω_1: (solvent-excluded) molecular volume

Σ: molecular surface

Explicit solvent:

- Discretize Ω_0
- Coulomb's law:

$$\varphi(\mathbf{r}) = k_e \sum_i \frac{q_i}{|\mathbf{r} - \mathbf{r}_i|}$$

- Can be expensive!

Implicit solvent:

- Continuum dielectric
- Poisson equation:

$$-\nabla \cdot (\varepsilon \nabla \varphi) = \rho$$

For many applications, implicit solvation provides a good balance of physical realism and computational efficiency.

Poisson equation: $-\nabla \cdot (\varepsilon \nabla \varphi) = \rho$

Poisson equation: $-\nabla \cdot (\varepsilon \nabla \varphi) = \rho$

In the molecule: $-\Delta\varphi = \frac{1}{\varepsilon_1} \sum_i q_i \delta(\mathbf{r} - \mathbf{r}_i)$

Poisson equation: $-\nabla \cdot (\varepsilon \nabla \varphi) = \rho$

In the molecule: $$-\Delta\varphi = \frac{1}{\varepsilon_1} \sum_i q_i \delta(\mathbf{r} - \mathbf{r}_i)$$

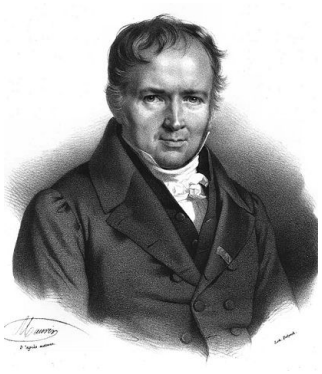
In the solvent:

$$-\Delta\varphi = \frac{1}{\varepsilon_0} \sum_i q_i c_i$$

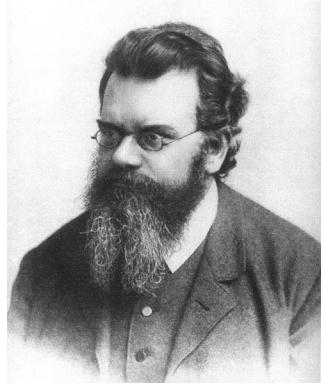
$$= \frac{1}{\varepsilon_0} \sum_i q_i c_i^\infty \exp\left(-\frac{q_i \varphi}{k_B T}\right)$$

$$\approx \frac{1}{\varepsilon_0} \left(\sum_i q_i c_i^\infty - \sum_i \frac{q_i^2 c_i^\infty}{k_B T} \varphi \right)$$

$$-\Delta\varphi \equiv -\kappa^2 \varphi$$



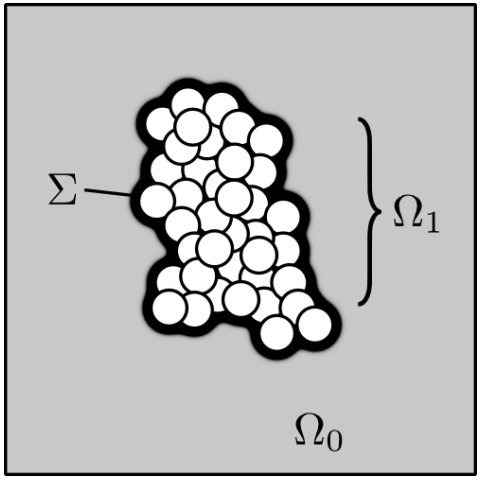
linearized Poisson-Boltzmann equation



$$-\left(\Delta - \kappa^2\right)\varphi = 0 \qquad \text{in } \Omega_0$$

$$-\Delta\varphi = \frac{1}{\varepsilon_1}\sum_i q_i \delta\left(\mathbf{r} - \mathbf{r}_i\right) \qquad \text{in } \Omega_1$$

$$[\varphi] = \left[\varepsilon \frac{\partial \varphi}{\partial \nu}\right] = 0 \qquad \text{on } \Sigma$$

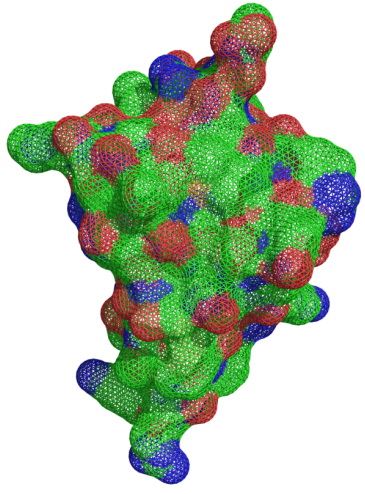


Many ways to solve: finite differences, finite elements

- Can be ill-conditioned
- Artificial domain truncation

We use instead boundary integral equation methods:

- Provably well-conditioned
- Exact boundary conditions
- Dimensional reduction



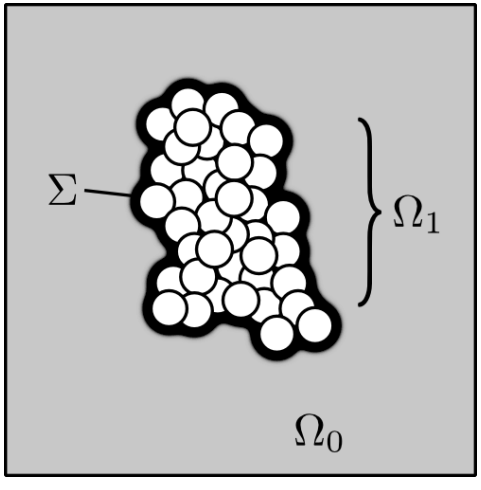
Integral equation basics

Green's function: $$G_k(\mathbf{r},\mathbf{s}) = \frac{e^{-k|\mathbf{r}-\mathbf{s}|}}{4\pi|\mathbf{r}-\mathbf{s}|}$$

Single-layer potential: $$S_k[\sigma](\mathbf{r}) = \int_\Sigma G_k(\mathbf{r},\mathbf{s})\,\sigma(\mathbf{s})\,dA_\mathbf{s} \qquad \text{in } \Omega_{0,1}$$

Double-layer potential: $$D_k[\mu](\mathbf{r}) = \int_\Sigma \frac{\partial G_k}{\partial \nu_\mathbf{s}}(\mathbf{r},\mathbf{s})\,\mu(\mathbf{s})\,dA_\mathbf{s} \qquad \text{in } \Omega_{0,1}$$

Jump relations as $\mathbf{r} \to \mathbf{s} \in \Sigma$:

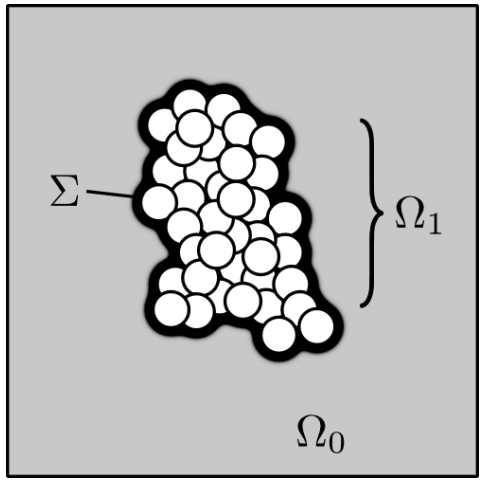


$$\left.\begin{aligned} S_k'[\sigma](\mathbf{r}) &\to \mp\frac{1}{2}\sigma(\mathbf{s}) + S_k'^{*}[\sigma](\mathbf{s}) \\ D_k[\mu](\mathbf{r}) &\to \pm\frac{1}{2}\mu(\mathbf{s}) + D_k^{*}[\mu](\mathbf{s}) \end{aligned}\right\} \quad \text{if } \mathbf{r} \in \Omega_{0,1}$$

$$-\left(\Delta-\kappa^2\right)\varphi=0 \qquad \text{in } \Omega_0$$

$$-\Delta\varphi=\frac{1}{\varepsilon_1}\sum_i q_i\delta\left(\mathbf{r}-\mathbf{r}_i\right) \qquad \text{in } \Omega_1$$

$$[\varphi]=\left[\varepsilon\frac{\partial\varphi}{\partial\nu}\right]=0 \qquad \text{on } \Sigma$$



Solution representation:

$$\varphi\equiv\begin{cases}S_\kappa\sigma+D_\kappa\mu & \text{in } \Omega_0,\\ S_0\sigma+\alpha D_0\mu+\varphi_s & \text{in } \Omega_1,\end{cases} \qquad \alpha\equiv\frac{\varepsilon_0}{\varepsilon_1}, \quad \varphi_s\left(\mathbf{r}\right)\equiv\frac{1}{\varepsilon_1}\sum_i q_i G_0\left(\mathbf{r},\mathbf{r}_i\right)$$

Boundary integral equation on Σ:

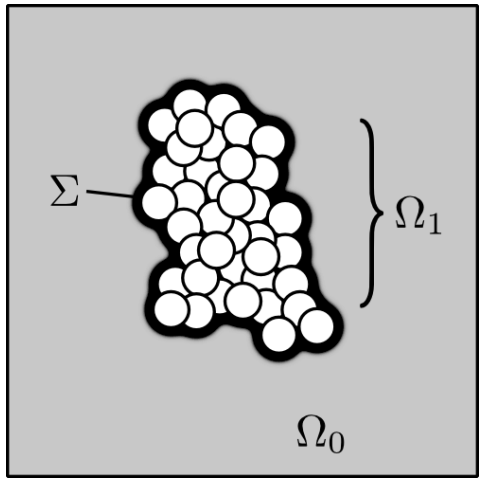
$$\frac{1}{2}\left(1+\alpha\right)\mu+\left(S_\kappa-S_0\right)\sigma+\left(D_\kappa-\alpha D_0\right)\mu=\varphi_s,$$

$$-\frac{1}{2}\left(1+\alpha\right)\sigma+\left(\alpha S'_\kappa-S'_0\right)\sigma+\alpha\left(D'_\kappa-D'_0\right)\mu=\frac{\partial\varphi_s}{\partial\nu}$$

$$-\left(\Delta-\kappa^2\right)\varphi=0 \qquad \text{in } \Omega_0$$

$$-\Delta\varphi=\frac{1}{\varepsilon_1}\sum_i q_i\delta\left(\mathbf{r}-\mathbf{r}_i\right) \qquad \text{in } \Omega_1$$

$$[\varphi]=\left[\varepsilon\frac{\partial\varphi}{\partial\nu}\right]=0 \qquad \text{on } \Sigma$$



Solution representation:

$$\varphi\equiv\begin{cases}S_\kappa\sigma+D_\kappa\mu & \text{in } \Omega_0,\\ S_0\sigma+\alpha D_0\mu+\varphi_s & \text{in } \Omega_1,\end{cases} \qquad \alpha\equiv\frac{\varepsilon_0}{\varepsilon_1}, \quad \varphi_s\left(\mathbf{r}\right)\equiv\frac{1}{\varepsilon_1}\sum_i q_i G_0\left(\mathbf{r},\mathbf{r}_i\right)$$

Boundary integral equation on Σ:

$$\frac{1}{2}\left(1+\alpha\right)\mu+\left(S_\kappa-S_0\right)\sigma+\left(D_\kappa-\alpha D_0\right)\mu=\varphi_s,$$

$$-\frac{1}{2}\left(1+\alpha\right)\sigma+\left(\alpha S'_\kappa-S'_0\right)\sigma+\alpha\left(D'_\kappa-D'_0\right)\mu=\frac{\partial\varphi_s}{\partial\nu}$$

$$\left(I+\lambda K\right)\begin{bmatrix}\mu\\ \sigma\end{bmatrix}=\lambda\begin{bmatrix}\varphi_s\\ -\partial\varphi_s/\partial\nu\end{bmatrix}$$

Let $A \in \mathbb{C}^{N \times N}$ be a matrix discretization of some non-oscillatory Green's function integral operator. Note that A is dense.

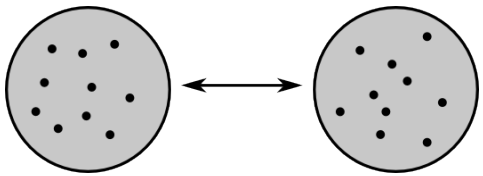
- Cost of applying A: $\mathcal{O}(N^2)$
- Cost of inverting A: $\mathcal{O}(N^3)$

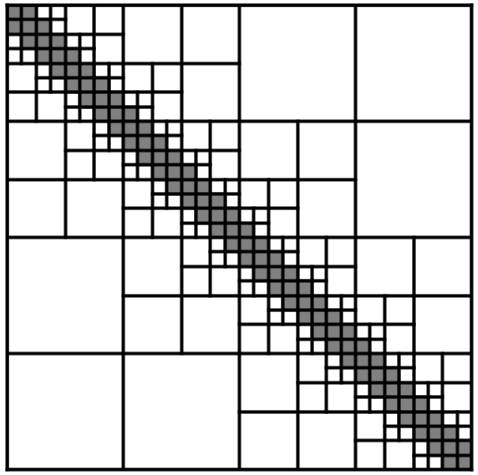
Fast iterative solvers:

- Krylov subspace methods (GMRES, BiCG, CGR)
- Fast matrix-vector product algorithms (treecode, FMM, panel clustering)
- Cost: $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$

Basic idea:

- Non-oscillatory Green's functions have smooth far fields
- Exploit smoothness with a hierarchical decomposition of space





Fast iterative solvers have been very successful, but they remain inefficient in certain important regimes:

- When A is ill-conditioned (multiphysics, singular geometries)
- When $Ax = b$ must be solved with many right-hand sides b or many perturbations of a base matrix A (optimization, design, time marching)
- Biological context: $\mathrm{p}K_\mathrm{a}$ calculation, structure prediction, docking

Fast iterative solvers have been very successful, but they remain inefficient in certain important regimes:

- When A is ill-conditioned (multiphysics, singular geometries)
- When $Ax = b$ must be solved with many right-hand sides b or many perturbations of a base matrix A (optimization, design, time marching)
- Biological context: pK_a calculation, structure prediction, docking

One solution: direct solvers (construct A^{-1}).

- Robust: insensitive to conditioning, always works
- Fast solves and inverse updates following initial factorization

Fast iterative solvers have been very successful, but they remain inefficient in certain important regimes:

- When A is ill-conditioned (multiphysics, singular geometries)
- When $Ax = b$ must be solved with many right-hand sides b or many perturbations of a base matrix A (optimization, design, time marching)
- Biological context: pK_a calculation, structure prediction, docking

One solution: direct solvers (construct A^{-1}).

- Robust: insensitive to conditioning, always works
- Fast solves and inverse updates following initial factorization

Can we accelerate direct solvers to the same extent?

Fast iterative solvers have been very successful, but they remain inefficient in certain important regimes:

- When A is ill-conditioned (multiphysics, singular geometries)
- When $Ax = b$ must be solved with many right-hand sides b or many perturbations of a base matrix A (optimization, design, time marching)
- Biological context: $\mathrm{p}K_\mathrm{a}$ calculation, structure prediction, docking

One solution: direct solvers (construct A^{-1}).

- Robust: insensitive to conditioning, always works
- Fast solves and inverse updates following initial factorization

Can we accelerate direct solvers to the same extent?

The answer is yes (more or less).

The same basic ideas apply, though some numerical machinery is required.

Related work:

- $\mathscr{H}$-matrices (Hackbusch et al.)
- HSS matrices (Chandrasekaran, Gu, et al.)
- Skeletonization (Martinsson, Rokhlin, Greengard et al.)
  - BIEs in 2D
  - One-level BIEs in 3D

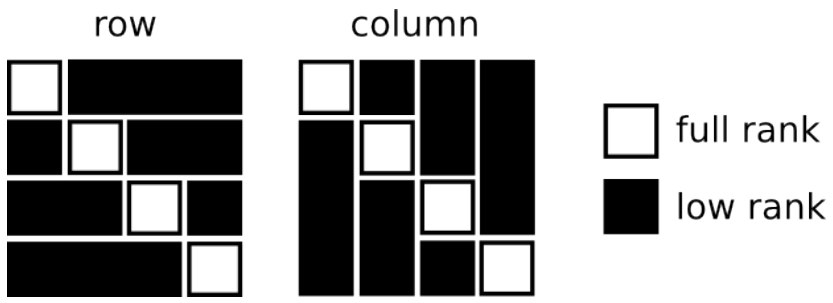
Here, we present a multilevel fast direct solver in general dimension. For BIEs:

	2D	3D
precomp	$\mathcal{O}(N)$	$\mathcal{O}(N^{3/2})$
solve	$\mathcal{O}(N)$	$\mathcal{O}(N \log N)$

Each solve is very fast, often beating the FMM by several orders of magnitude.

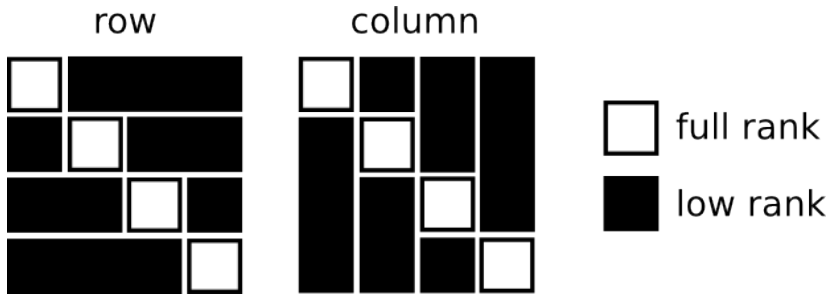
A block matrix A is block separable if

$$\underbrace{\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix}}_{A_{ij}} = \underbrace{\begin{bmatrix} \times \\ \times \end{bmatrix}}_{L_i} \underbrace{\begin{bmatrix} \times \end{bmatrix}}_{S_{ij}} \underbrace{\begin{bmatrix} \times & \times \end{bmatrix}}_{R_j} \quad , \quad i \neq j.$$

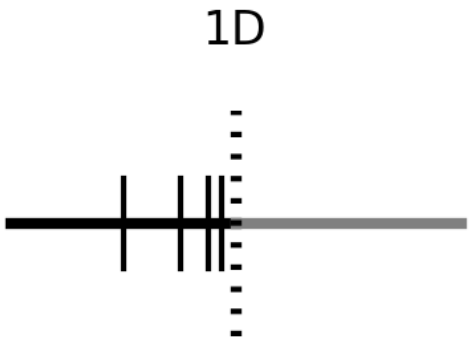


A block matrix A is block separable if

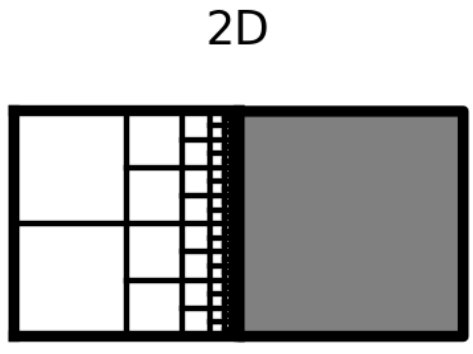
$$\underbrace{\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix}}_{A_{ij}} = \underbrace{\begin{bmatrix} \times \\ \times \end{bmatrix}}_{L_i} \underbrace{\begin{bmatrix} \times \end{bmatrix}}_{S_{ij}} \underbrace{\begin{bmatrix} \times & \times \end{bmatrix}}_{R_j}, \quad i \neq j.$$



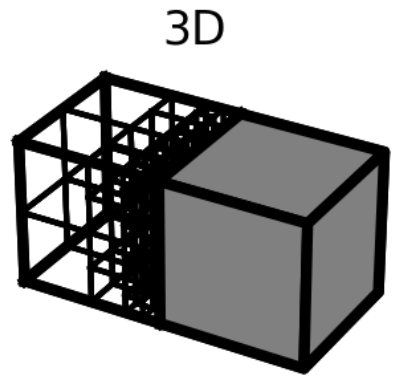
Integral equation matrices are block separable.



$\mathcal{O}(\log n)$

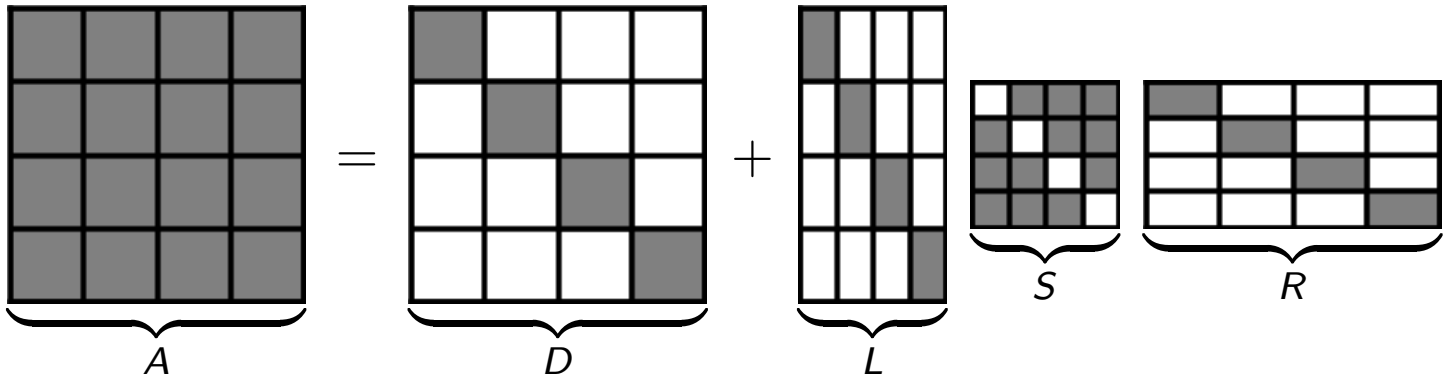


$\mathcal{O}(n^{1/2})$

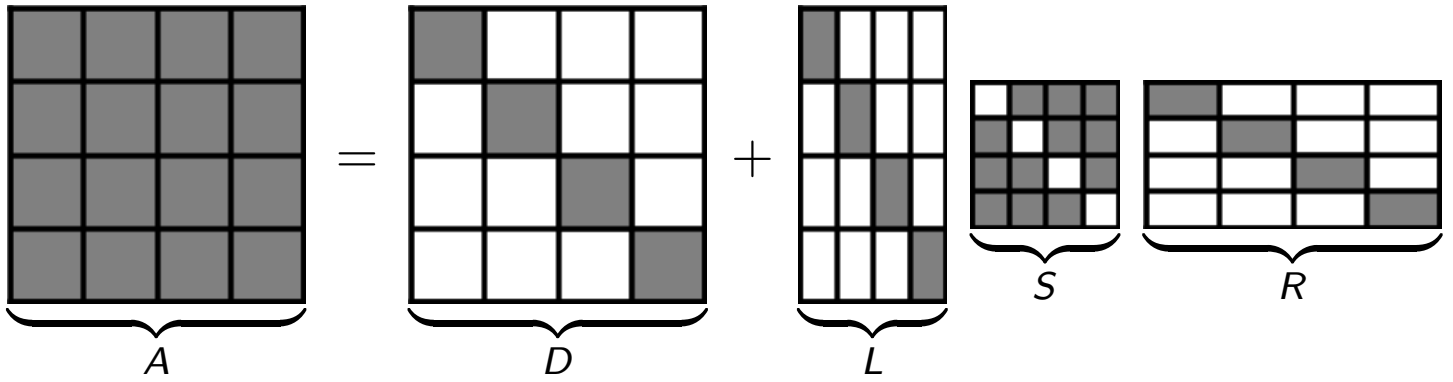


$\mathcal{O}(n^{2/3})$

If A is block separable, then



If A is block separable, then



The inverse can be written in essentially the same form:

$$A^{-1} = \mathcal{D} + \mathcal{L}\mathcal{S}^{-1}\mathcal{R},$$

where

$$\mathcal{D} = D^{-1} - D^{-1}L\Lambda RD^{-1}, \quad \mathcal{L} = D^{-1}L\Lambda, \quad \mathcal{R} = \Lambda RD^{-1}, \quad \mathcal{S} = \Lambda + S,$$

with $\Lambda = \left(RD^{-1}L\right)^{-1}$. If A has $p \times p$ blocks and each $S_{ij} \in \mathbb{C}^{k\times k}$, then A^{-1} can be computed in $\mathcal{O}(p(N/p)^3 + (pk)^3)$ operations.

We can also adopt a sparse matrix perspective. For

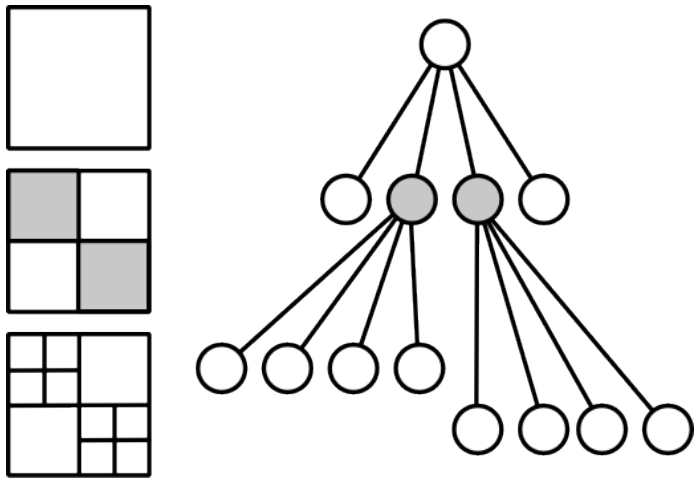
$$Ax = (D + LSR)\, x = b,$$

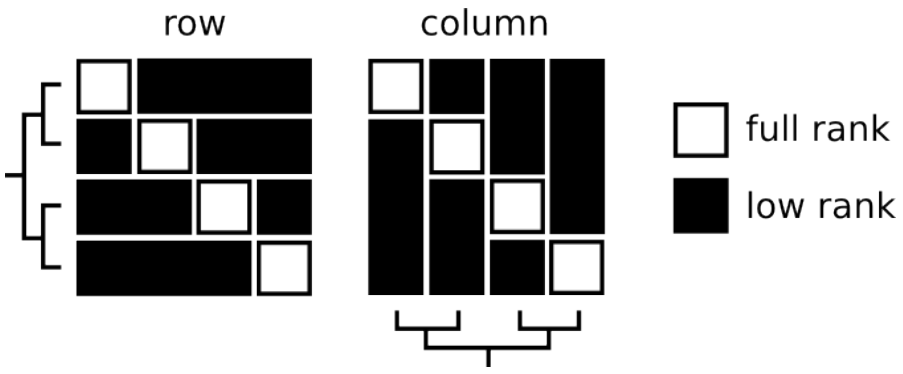
let $z \equiv Rx$ and $y \equiv Sz$. Then this is equivalent to the structured sparse system

$$\begin{bmatrix} D & L & \\ R & & -I \\ & -I & S \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}.$$

Factor using UMFPACK, SuperLU, MUMPS, Pardiso, etc.

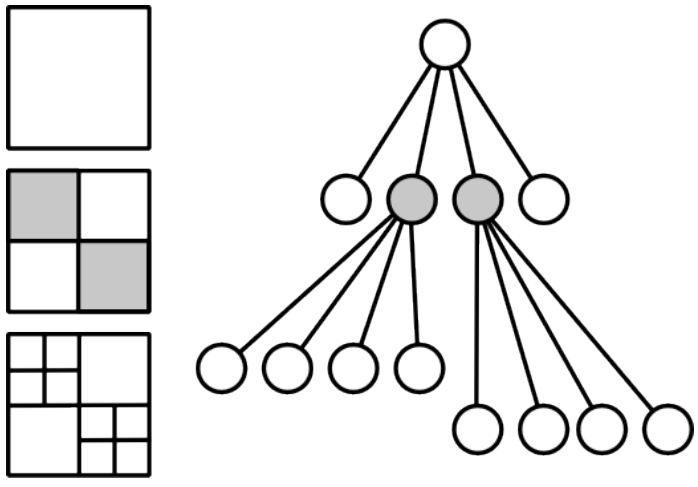
Integral equation matrices are, in fact, hierarchically block separable, i.e., they are block separable at every level of an octree-type ordering.

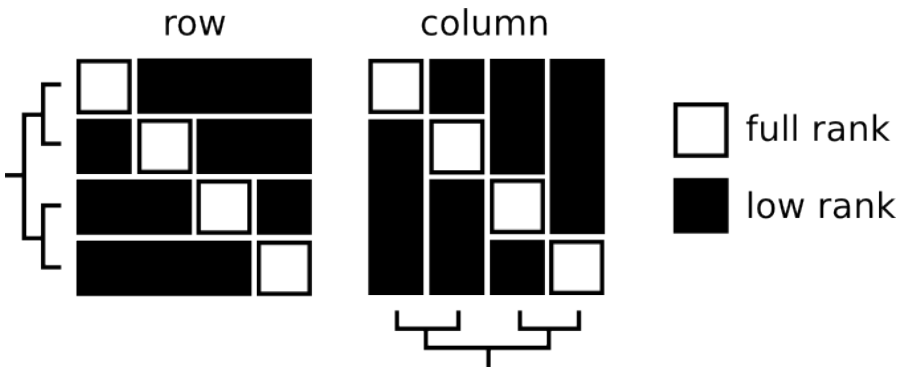




In this setting, much more powerful algorithms can be developed.

Integral equation matrices are, in fact, hierarchically block separable, i.e., they are block separable at every level of an octree-type ordering.





In this setting, much more powerful algorithms can be developed.

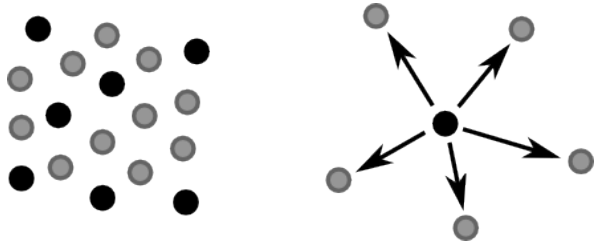
How to compress to block separable form?

An interpolative decomposition of a rank-k matrix is a representation

$$\underbrace{A}_{m\times n} = \underbrace{B}_{m\times k}\underbrace{P}_{k\times n},$$

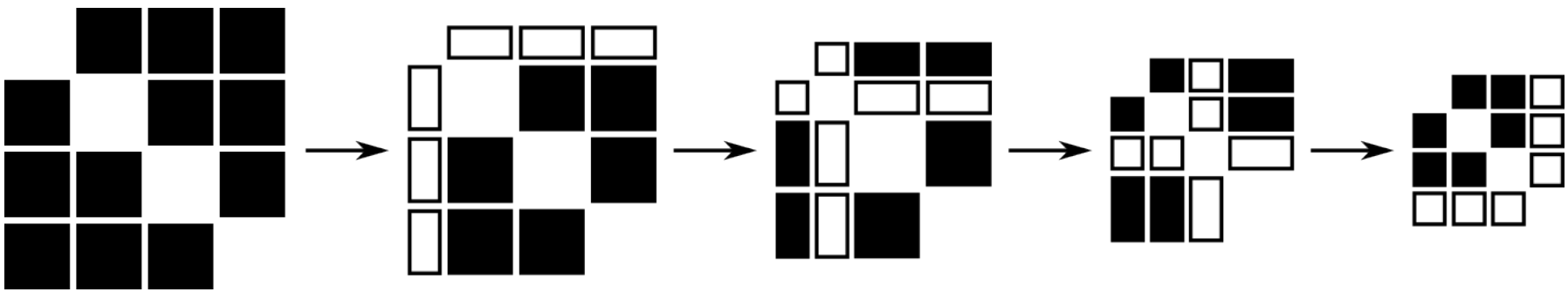
where B is a column-submatrix of A (with $\|P\|$ small).

- The ID compresses the column space; to compress the row space, apply the ID to A^{T}. We call the retained rows and columns skeletons.
- Adaptive algorithms can compute the ID to any specified precision $\epsilon > 0$.



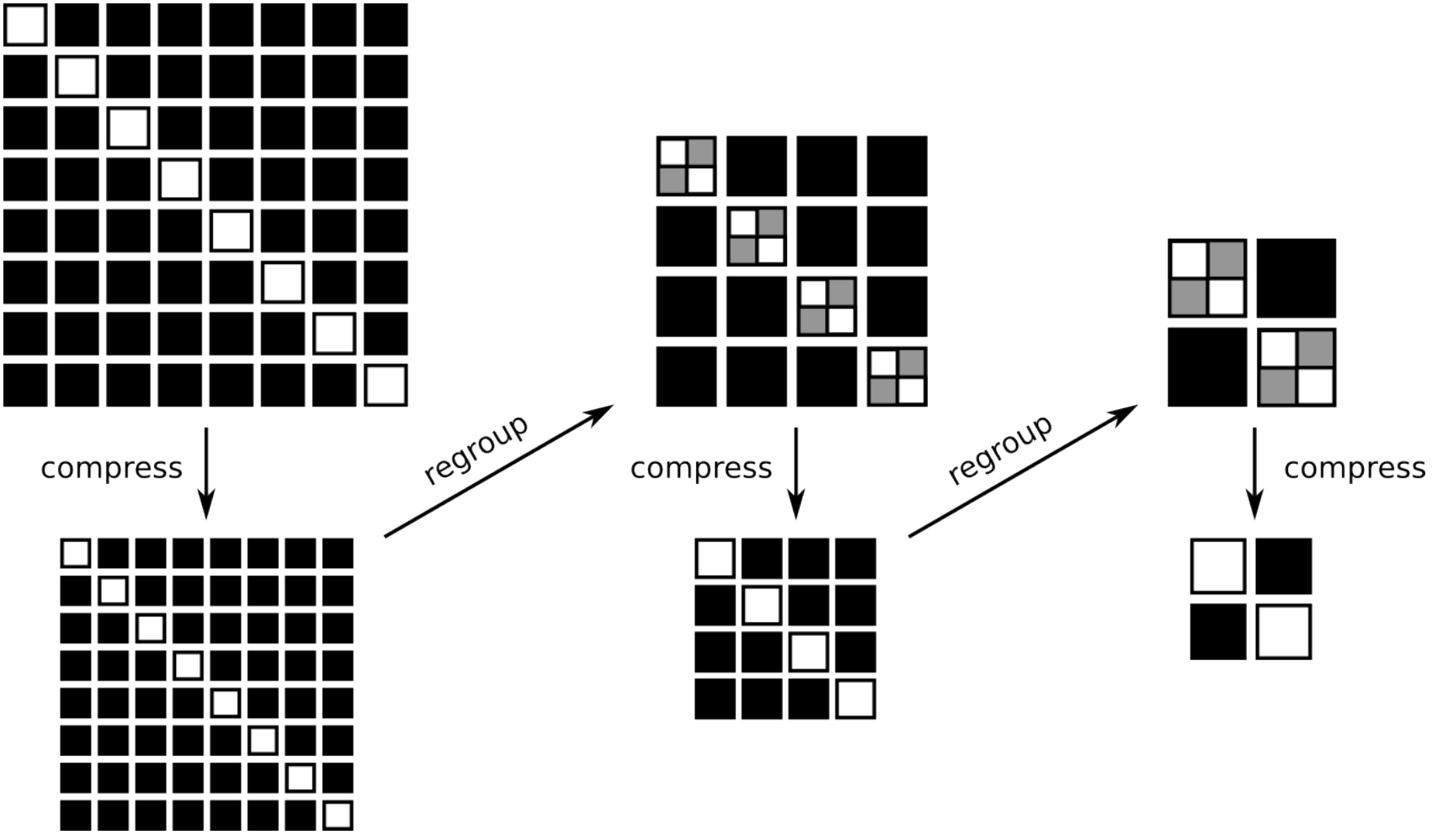
One-level matrix compression

- Compress the row space of each off-diagonal block row.
  Let the L_i be the corresponding row projection matrices.
- Compress the column space of each off-diagonal block column.
  Let the R_j be the corresponding column projection matrices.
- Approximate the off-diagonal blocks by $A_{ij} \approx L_i S_{ij} R_j$ for $i \neq j$.



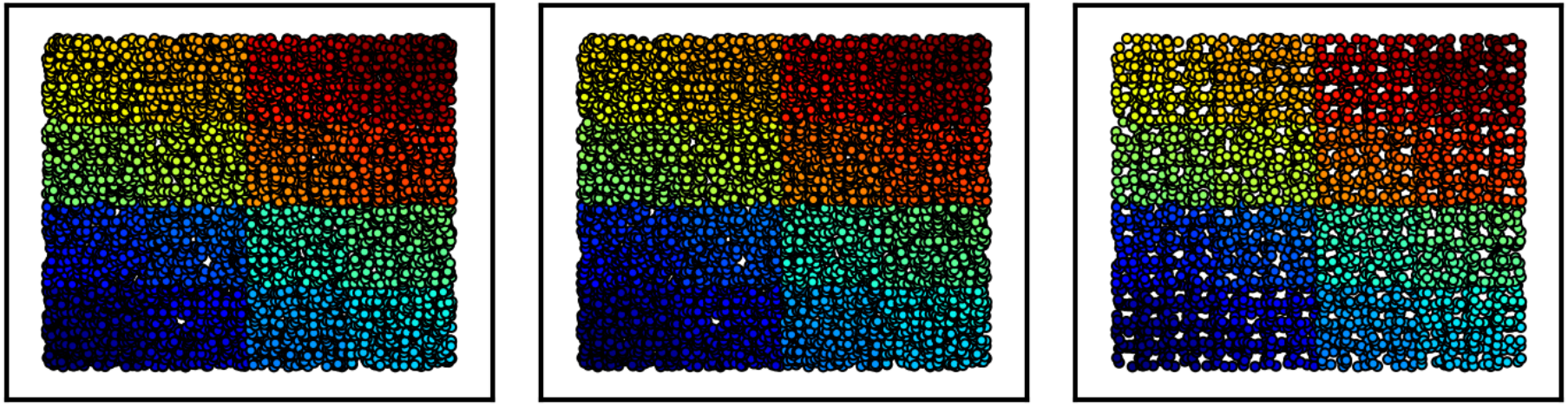
Skeletonization

Multilevel matrix compression

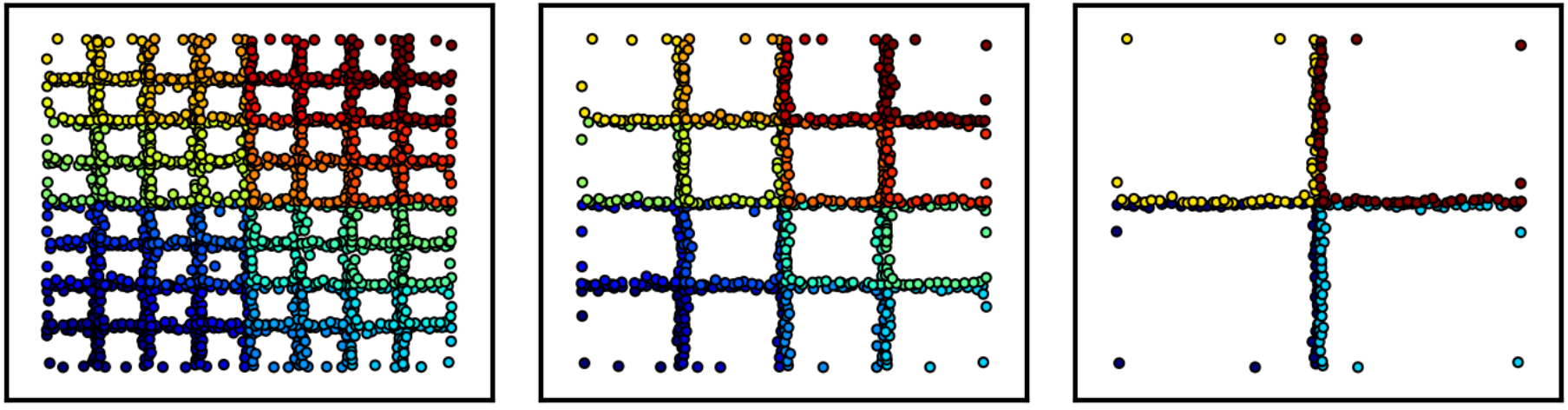


Recursive skeletonization

$N_0 = 8192$ $N_1 = 7134$ $N_2 = 4138$



$N_3 = 1849$ $N_4 = 776$ $N_5 = 265$

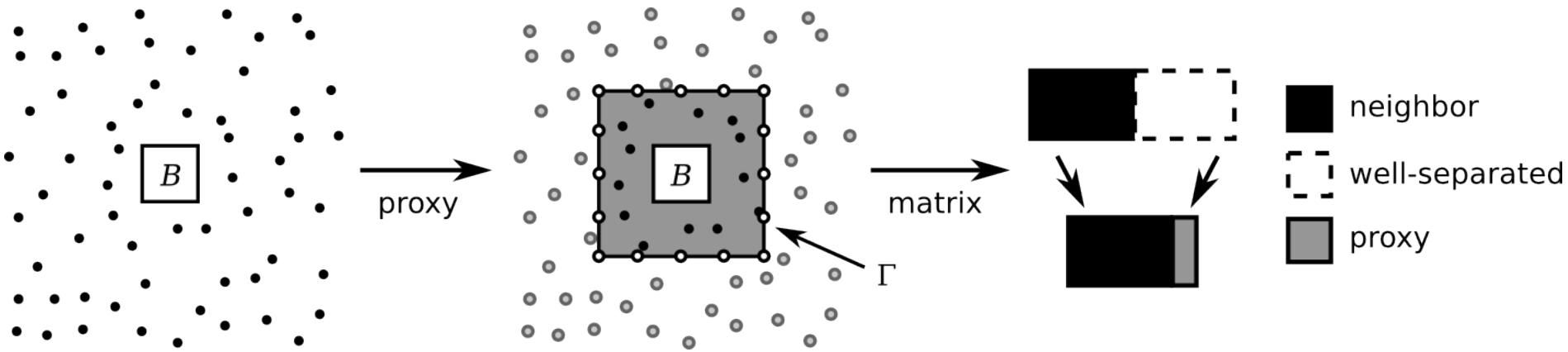


$$G(x,y) = -\frac{1}{2\pi}\log|x-y| \ , \quad \epsilon = 10^{-3}$$

- General compression algorithm is global and so at least $\mathcal{O}(N^2)$
- For potential fields, use Green's theorem to accelerate:

$$u(\mathbf{r}) = \int_\Gamma \left[u(\mathbf{s}) \frac{\partial G}{\partial \nu_\mathbf{s}}(\mathbf{r}, \mathbf{s}) - G(\mathbf{r}, \mathbf{s}) \frac{\partial u}{\partial \nu}(\mathbf{s}) \right] dA_\mathbf{s}$$

- Represent well-separated points with a local proxy surface



Compressed telescoping matrix representation:

$$A \approx D^{(1)} + L^{(1)} \left[D^{(2)} + L^{(2)} \left(\cdots D^{(\lambda)} + L^{(\lambda)} S R^{(\lambda)} \cdots \right) R^{(2)} \right] R^{(1)}$$

- Efficient storage (data-sparse)

N	uncomp	comp
8192	537 MB	9.7 MB
131072	137 GB	184 MB

- Fast matrix-vector multiplication (generalized FMM)
- Fast matrix factorization and inverse application

Recursively expand in sparse form:

$$\begin{bmatrix} D^{(1)} & L^{(1)} & & & & & \\ R^{(1)} & & -I & & & & \\ & -I & D^{(2)} & L^{(2)} & & & \\ & & R^{(2)} & \ddots & \ddots & & \\ & & & \ddots & D^{(\lambda)} & L^{(\lambda)} & \\ & & & & R^{(\lambda)} & & -I \\ & & & & & -I & S \end{bmatrix} \begin{bmatrix} x \\ y^{(1)} \\ z^{(1)} \\ \vdots \\ \vdots \\ y^{(\lambda)} \\ z^{(\lambda)} \end{bmatrix} = \begin{bmatrix} b \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

This can be treated efficiently using any standard sparse direct solver.

Multilevel inversion formula (for analysis):

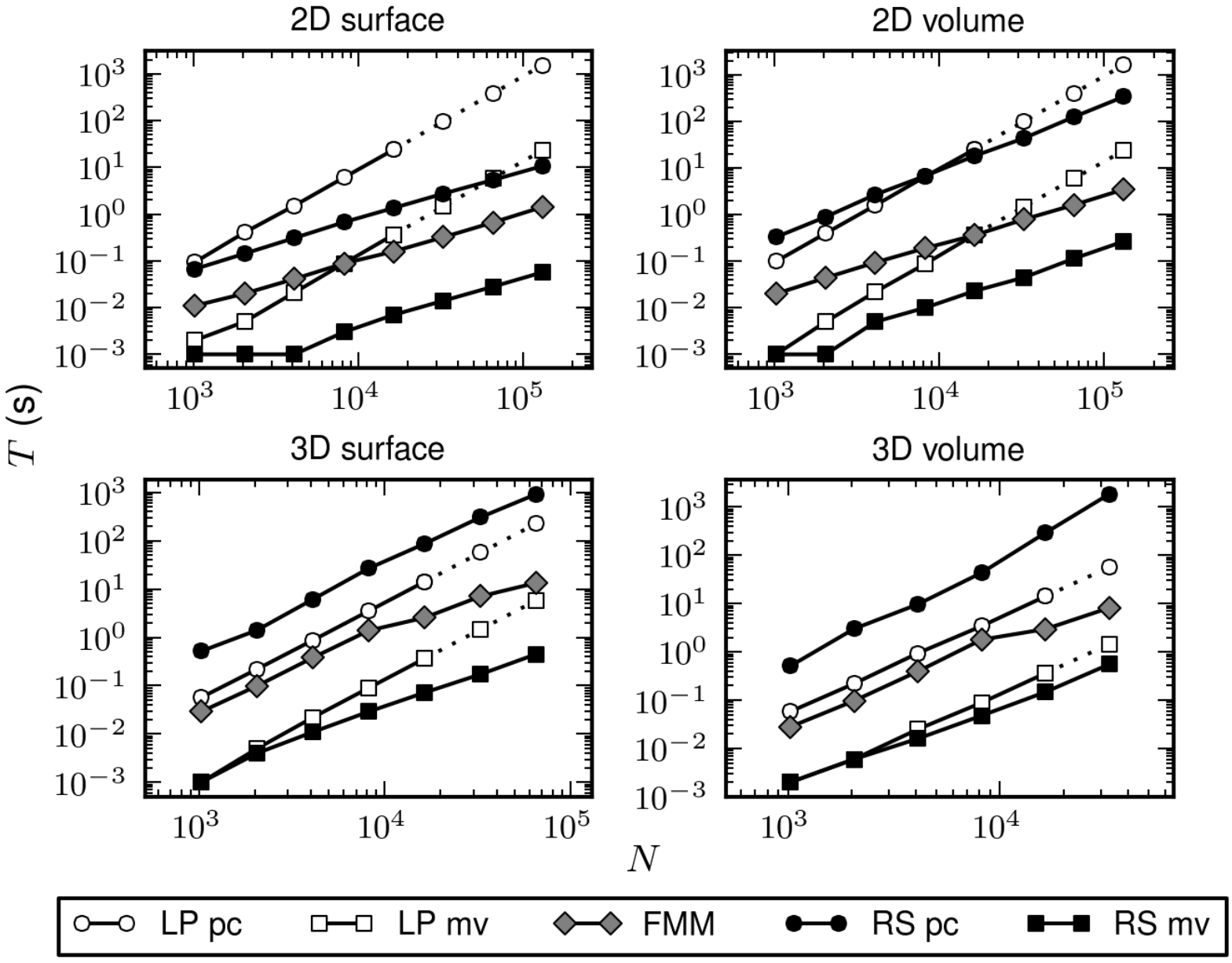
$$A^{-1} \approx \mathcal{D}^{(1)} + \mathcal{L}^{(1)} \left[\mathcal{D}^{(2)} + \mathcal{L}^{(2)} \left(\cdots D^{(\lambda)} + \mathcal{L}^{(\lambda)} \mathcal{S}^{-1} \mathcal{R}^{(\lambda)} \cdots \right) \mathcal{R}^{(2)} \right] \mathcal{R}^{(1)}$$

Complexities in d dimensions (BIEs in $d+1$ dimensions):

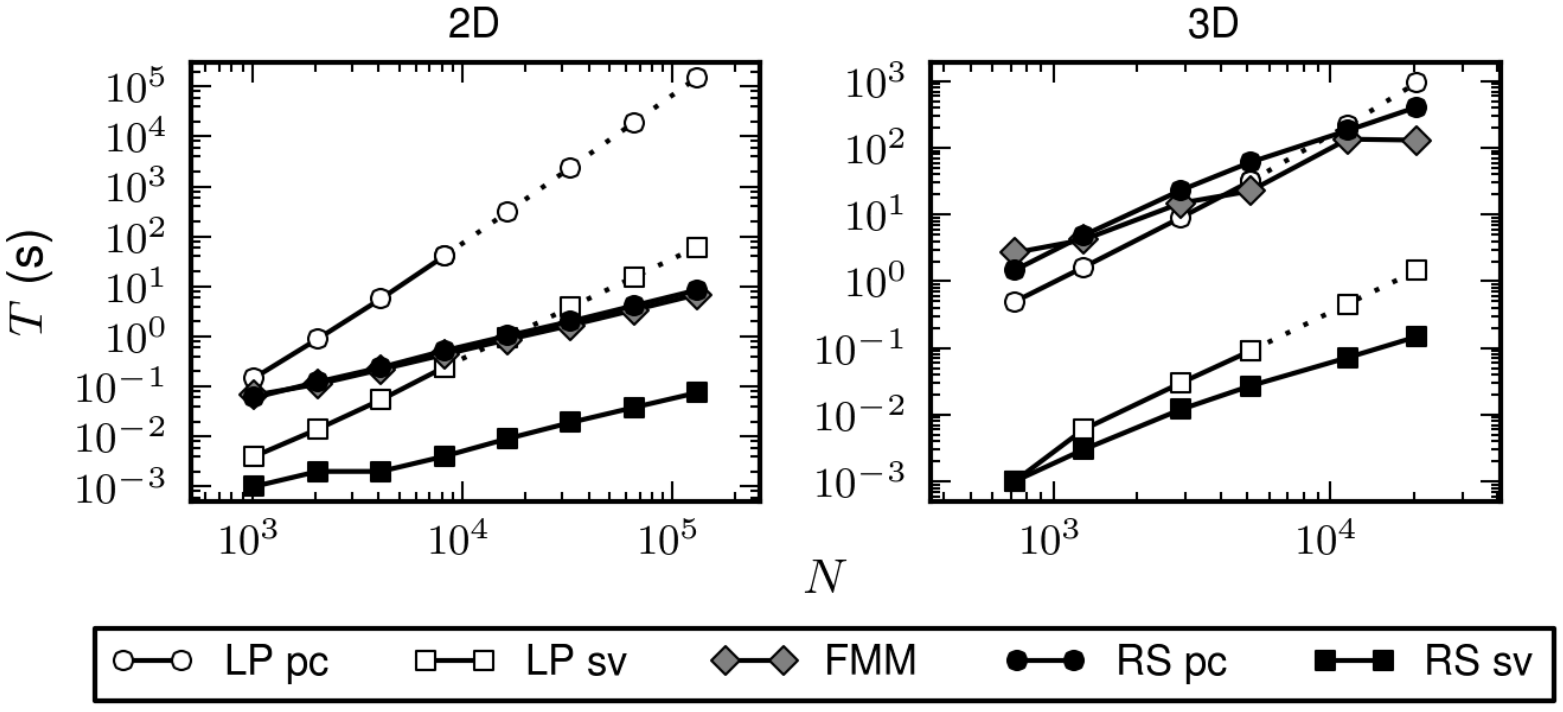
$$\text{precomp} \sim \begin{cases} N & \text{if } d = 1, \\ N^{3(1-1/d)} & \text{if } d > 1, \end{cases} \qquad \text{solve} \sim \begin{cases} N & \text{if } d = 1, \\ N \log N & \text{if } d = 2, \\ N^{2(1-1/d)} & \text{if } d > 2 \end{cases}$$

- Mild assumptions: low-rank off-diagonal blocks, Green's theorem
- Based on numerical linear algebra rather than analytic expansions
- Kernel-independent: Laplace, Stokes, Yukawa, low-frequency Helmholtz, etc.
- Compressed ranks are optimal for the problem at hand
- Like the FMM but with some near-field compression
- Trade accuracy for speed: user-specified precision
- Naturally parallelizable via block-sweep structure

Laplace FMM


2D surface
2D volume
3D surface
3D volume
T (s)
N
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^4
10^5
LP pc
LP mv
FMM
RS pc
RS mv

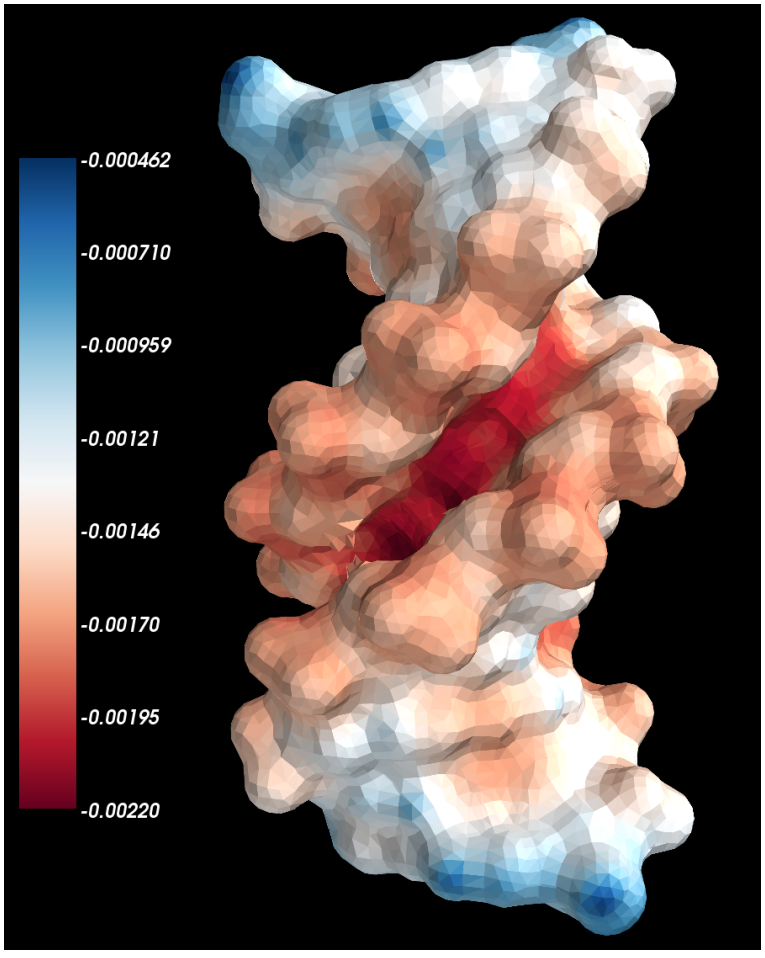
Laplace BIE solver



- Less memory-efficient than FMM/GMRES
- Each solve is extremely fast (in elements/sec)

ϵ	10^{-3}	10^{-6}	10^{-9}
2D	3.3×10^6	2.0×10^6	1.7×10^6
3D	6.0×10^5	1.4×10^5	6.2×10^4

Poisson electrostatics



$$-\Delta\varphi = 0 \qquad \text{in } \Omega_0$$

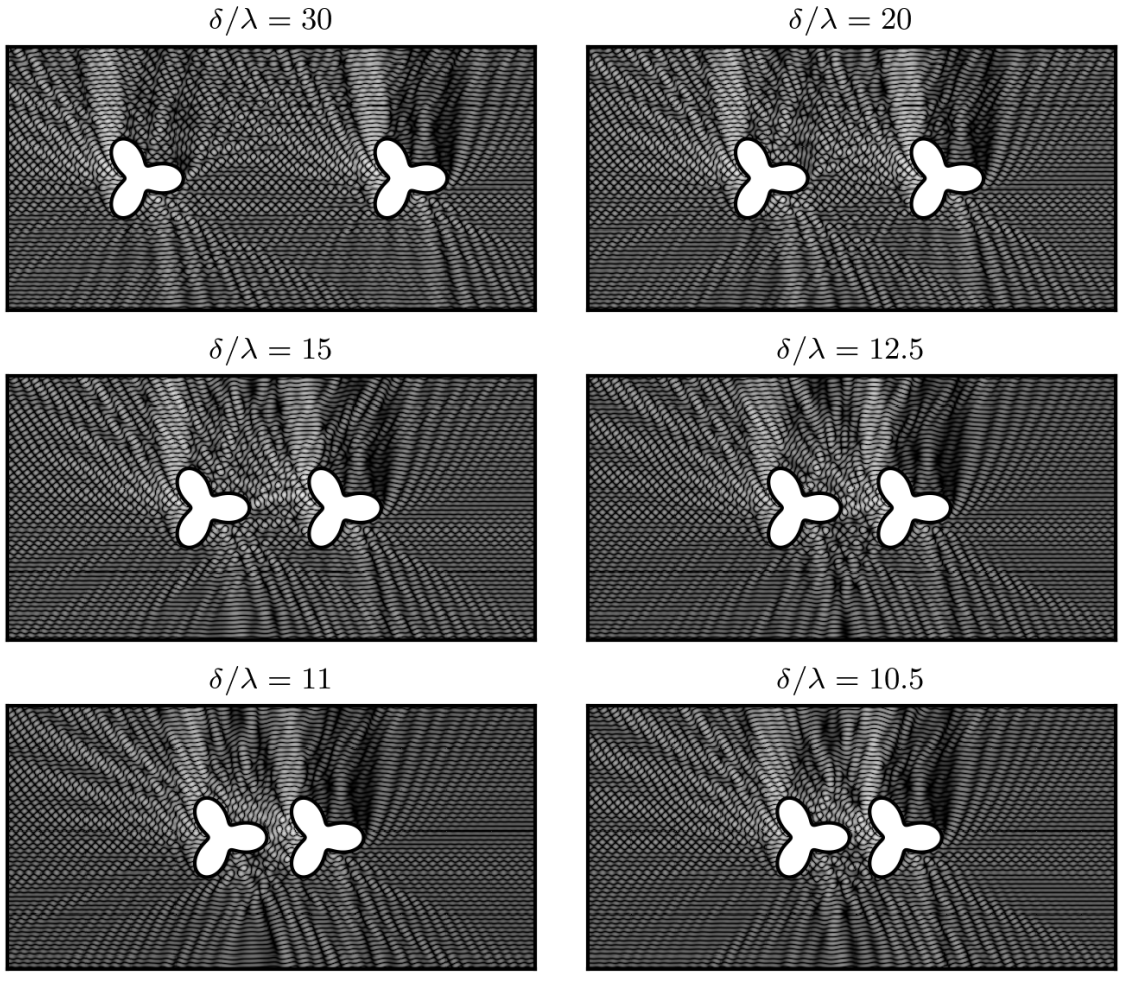
$$-\Delta\varphi = \frac{1}{\varepsilon_1}\sum_i q_i \delta\left(\mathbf{r} - \mathbf{r}_i\right) \qquad \text{in } \Omega_1$$

$$[\varphi] = \left[\varepsilon \frac{\partial\varphi}{\partial\nu}\right] = 0 \qquad \text{on } \Sigma$$

N	7612	19752
FMM/GMRES	12.6 s	26.9 s
RS precomp	151 s	592 s
RS solve	0.03 s	0.08 s

Break-even point: 10–25 solves

Multiple scattering



- Each object: 10λ

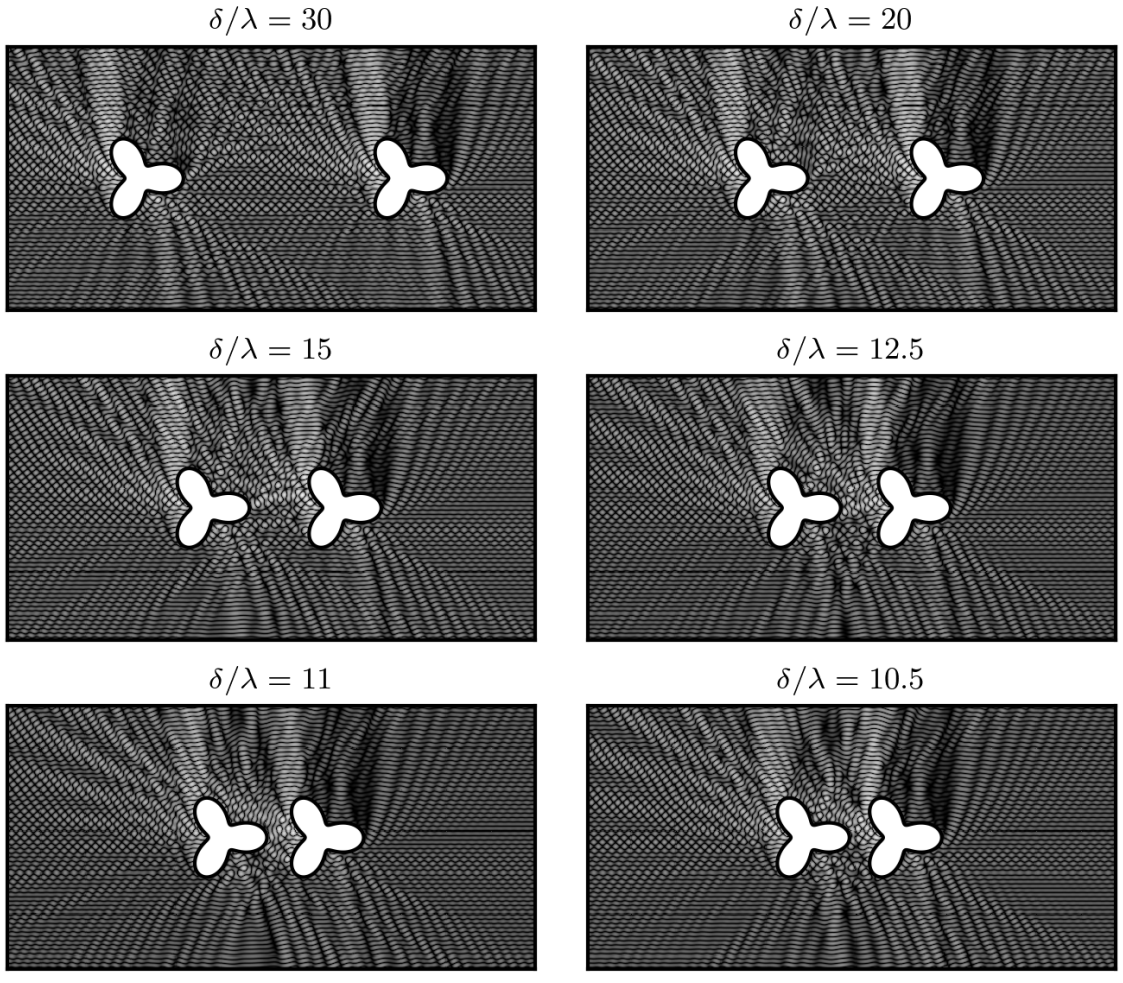
$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

- FMM/GMRES with block preconditioner via RS

$$\begin{bmatrix} A_{11}^{-1} & \\ & A_{22}^{-1} \end{bmatrix}$$

- Unprecon: 700 iterations
- Precon: 10 iterations
- $50\times$ speedup

Multiple scattering



- Each object: 10λ

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

- FMM/GMRES with block preconditioner via RS

$$\begin{bmatrix} A_{11}^{-1} & \\ & A_{22}^{-1} \end{bmatrix}$$

- Unprecon: 700 iterations
- Precon: 10 iterations
- $50\times$ speedup

Rigid-body "docking"

Main result:

- After precomputation, very fast solves (sub-second)
- Useful for systems involving many right-hand sides

Main result:

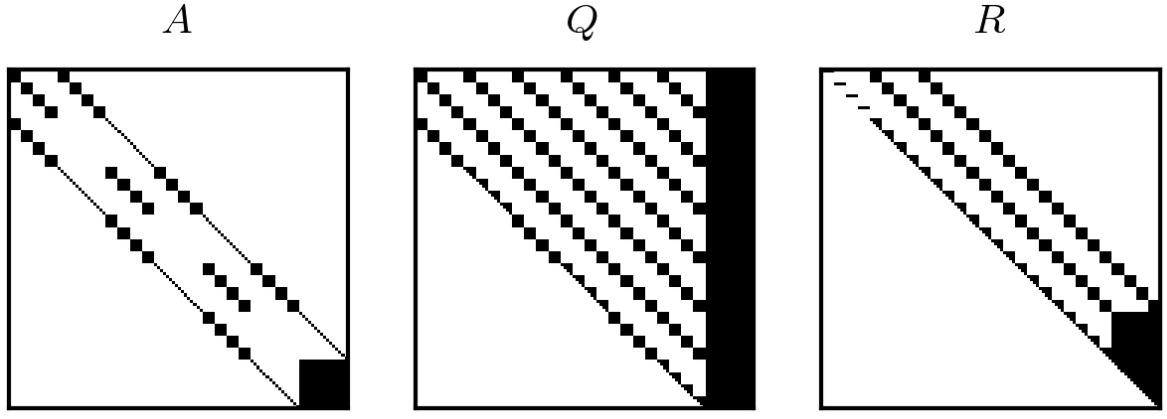
- After precomputation, very fast solves (sub-second)
- Useful for systems involving many right-hand sides

Extensions:

- Approximate inverse preconditioning
- Local geometric perturbations:

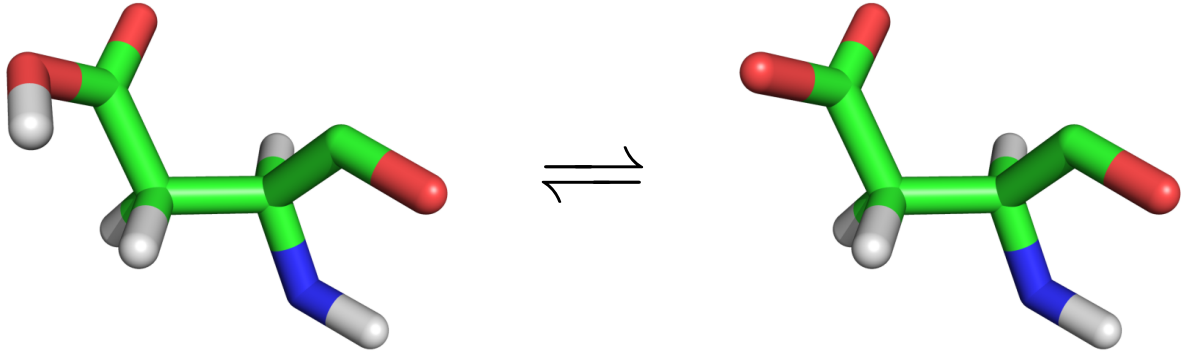
$$\begin{bmatrix} A & B_+ & B_- \\ C_+ & D_+ & D_* \\ C_- & & I \end{bmatrix} \begin{bmatrix} x \\ x_+ \\ x_- \end{bmatrix} = \begin{bmatrix} b \\ b_+ \\ 0 \end{bmatrix}$$

- Least squares (semi-direct QR)



- Compression-based FMM

Back to biophysics: protein pK_a calculations



$$\mathrm{p}K_a \equiv -\log_{10}\frac{[\mathrm{A}]\,[\mathrm{H}]}{[\mathrm{AH}]} = \log_{10}\frac{[\mathrm{AH}]}{[\mathrm{A}]} + \mathrm{pH}$$

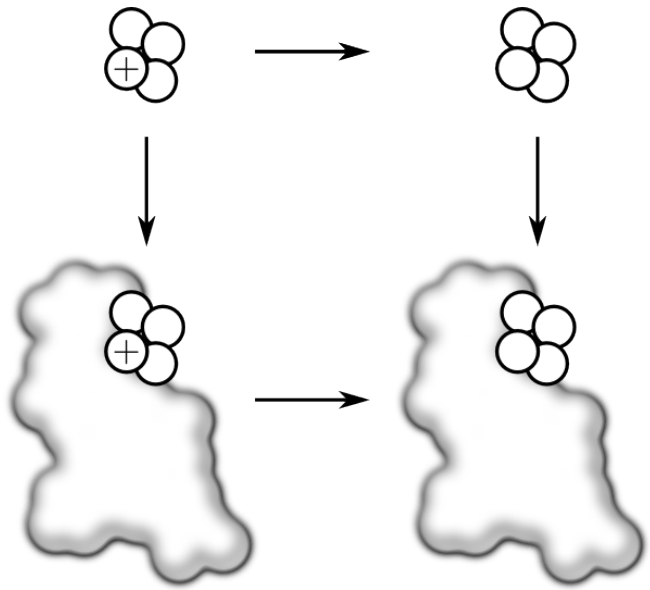
Ionization behavior is important for many biomolecular phenomena:

- Binding affinities
- Enzymatic activities
- Structural properties

$$\mathrm{p}K_\mathrm{a} = \frac{\beta}{\ln 10}\Delta G^\mathrm{p}_{\mathrm{AH}\to\mathrm{A}+\mathrm{H}}$$

$$\begin{aligned}\Delta G^\mathrm{p}_{\mathrm{AH}\to\mathrm{A}+\mathrm{H}} &= \Delta G^\mathrm{s}_{\mathrm{AH}\to\mathrm{A}+\mathrm{H}} + \Delta G^{\mathrm{s}\to\mathrm{p}}_\mathrm{A} - \Delta G^{\mathrm{s}\to\mathrm{p}}_\mathrm{AH} \\ &= \underbrace{\Delta G^\mathrm{s}_{\mathrm{AH}\to\mathrm{A}+\mathrm{H}}}_{\text{experiment}} + \underbrace{\Delta G^\mathrm{s}_{\mathrm{A}\to\mathrm{AH}} - \Delta G^\mathrm{p}_{\mathrm{A}\to\mathrm{AH}}}_{\text{electrostatic only}}\end{aligned}$$

$$\begin{array}{ccc} \mathrm{A_sH} & \xrightarrow{\Delta G^\mathrm{s}_{\mathrm{AH}\to\mathrm{A}+\mathrm{H}}} & \mathrm{A_s} + \mathrm{H} \\ \Big\downarrow{\scriptstyle \Delta G^{\mathrm{s}\to\mathrm{p}}_\mathrm{AH}} & & \Big\downarrow{\scriptstyle \Delta G^{\mathrm{s}\to\mathrm{p}}_\mathrm{A}} \\ \mathrm{A_pH} & \xrightarrow{\Delta G^\mathrm{p}_{\mathrm{AH}\to\mathrm{A}+\mathrm{H}}} & \mathrm{A_p} + \mathrm{H} \end{array}$$



$$\mathrm{p}K_\mathrm{a} = \underbrace{\mathrm{p}K^\mathrm{model}_\mathrm{a}}_{\text{experiment}} - \frac{\beta}{\ln 10}\underbrace{\Delta\Delta G^{\mathrm{s}\to\mathrm{p}}_{\mathrm{A}\to\mathrm{AH}}}_{\text{electrostatic}}$$

For M titrating sites, let $\theta \in \{0,1\}^M$ denote the protonation state of each site.

$$\mathrm{p}K_i^{\mathrm{intr}} \equiv \mathrm{p}K_i^{\mathrm{model}} - \frac{\beta}{\ln 10}\Delta\Delta G^{\mathrm{s}\to\mathrm{p}}_{\mathrm{A}\to\mathrm{A}(e_i)}$$

$$\Delta G_{\mathrm{A}\to\mathrm{A}(e_i)}(\mathrm{pH}) = -RT \ln 10 \left(\mathrm{p}K_i^{\mathrm{intr}} - \mathrm{pH}\right)$$

$$\Delta G_{\mathrm{A}\to\mathrm{A}(\theta)}(\mathrm{pH}) = -RT \ln 10 \sum_i \theta_i \left(\mathrm{p}K_i^{\mathrm{intr}} - \mathrm{pH}\right) + \frac{1}{2}\sum_i \theta_i \sum_{j\neq i} \theta_j \Delta G_{ij}$$

Mean site protonation:

$$\langle \theta_i \rangle (\mathrm{pH}) = \frac{1}{Z}\sum_\theta \theta_i e^{-\beta \Delta G_{\mathrm{A}\to\mathrm{A}(\theta)}(\mathrm{pH})}$$

Sample using Markov chain Monte Carlo.

Find $\mathrm{p}K_i$ such that $\langle \theta_i \rangle (\mathrm{p}K_i) = 1/2$.

How to calculate the titrating site interaction energies $\Delta G_{ij} = q_i^{\mathsf{T}} \varphi_j$? Recall:

$$\varphi = \begin{cases} S_\kappa \sigma + D_\kappa \mu & \text{in } \Omega_0, \\ S_0 \sigma + \alpha D_0 \mu + \varphi_s & \text{in } \Omega_1, \end{cases} \qquad (I + \lambda K) \begin{bmatrix} \mu \\ \sigma \end{bmatrix} = \lambda \begin{bmatrix} \varphi_s \\ -\partial \varphi_s / \partial \nu \end{bmatrix}.$$

Therefore, $\varphi_j = (CA^{-1}B + D) q_j$, where

$$A = I + \lambda K, \quad B = \lambda \begin{bmatrix} \varphi_s \\ -\partial \varphi_s / \partial \nu \end{bmatrix},$$

$$C = \begin{bmatrix} D_0 & \alpha S_0 \end{bmatrix}, \quad D_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \dfrac{1}{\varepsilon_1} G_0 (\mathbf{r}_i, \mathbf{r}_j) & \text{if } i \neq j. \end{cases}$$

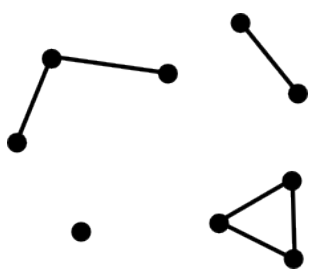
- Calculate φ_j for each site j.
- Compute $\Delta G_{ij} = q_i^{\mathsf{T}} \varphi_j$ for each site i.
- Requires M solves in total.
- Compress matrices as direct solver or generalized FMM.

pK_a algorithm

- Protein preparation
- Matrix precomputation
- Energy calculation
- Monte Carlo sampling
  - Reduced site approximation
  - Multi-site cluster moves
- Estimate pK_i
  - Error bars

pK_a algorithm

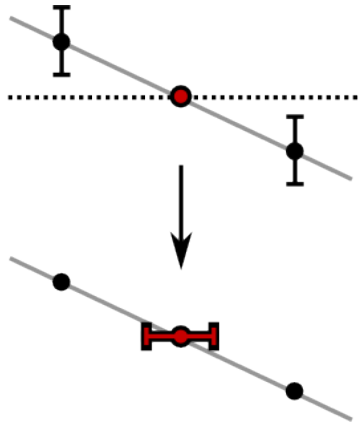
- Protein preparation
- Matrix precomputation
- Energy calculation
- Monte Carlo sampling
  - Reduced site approximation
  - Multi-site cluster moves
- Estimate pK_i
  - Error bars



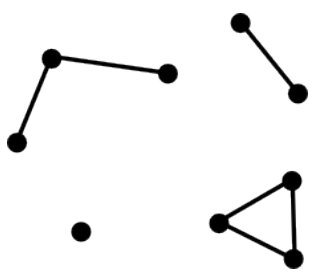
- Link sites by interaction energy
- Clusters: connected components
- Modify one cluster at random
- Pick move distance from geometric distribution

pK_a algorithm

- Protein preparation
- Matrix precomputation
- Energy calculation
- Monte Carlo sampling
  - Reduced site approximation
  - Multi-site cluster moves
- Estimate pK_i
  - Error bars



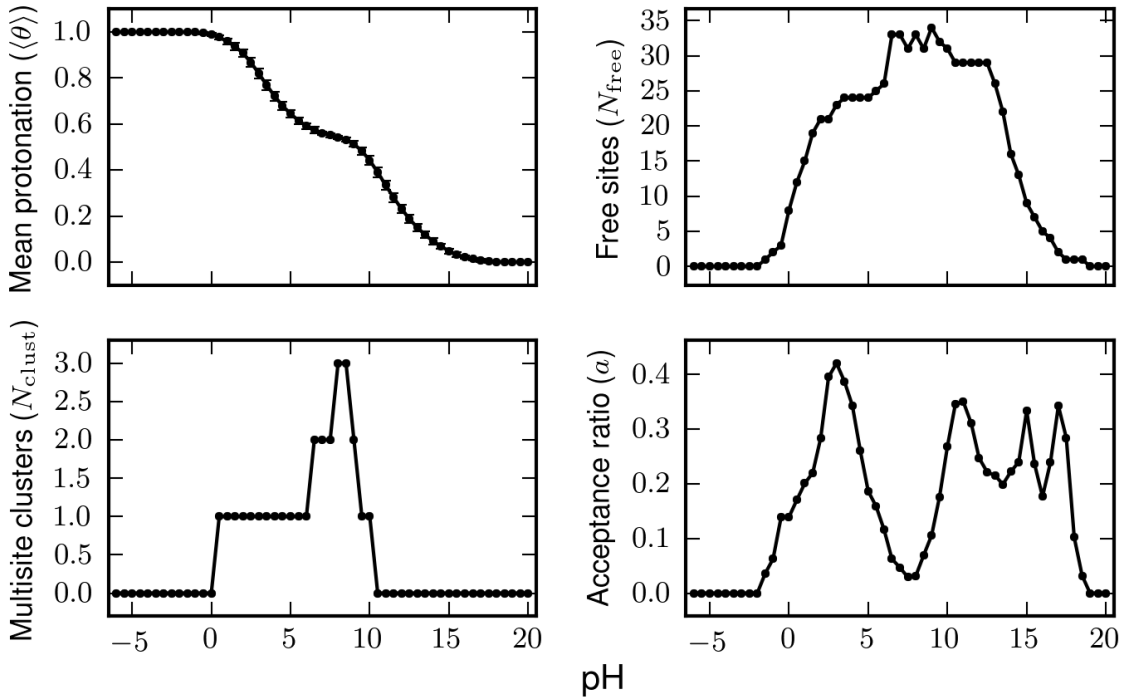
Apply delta method.

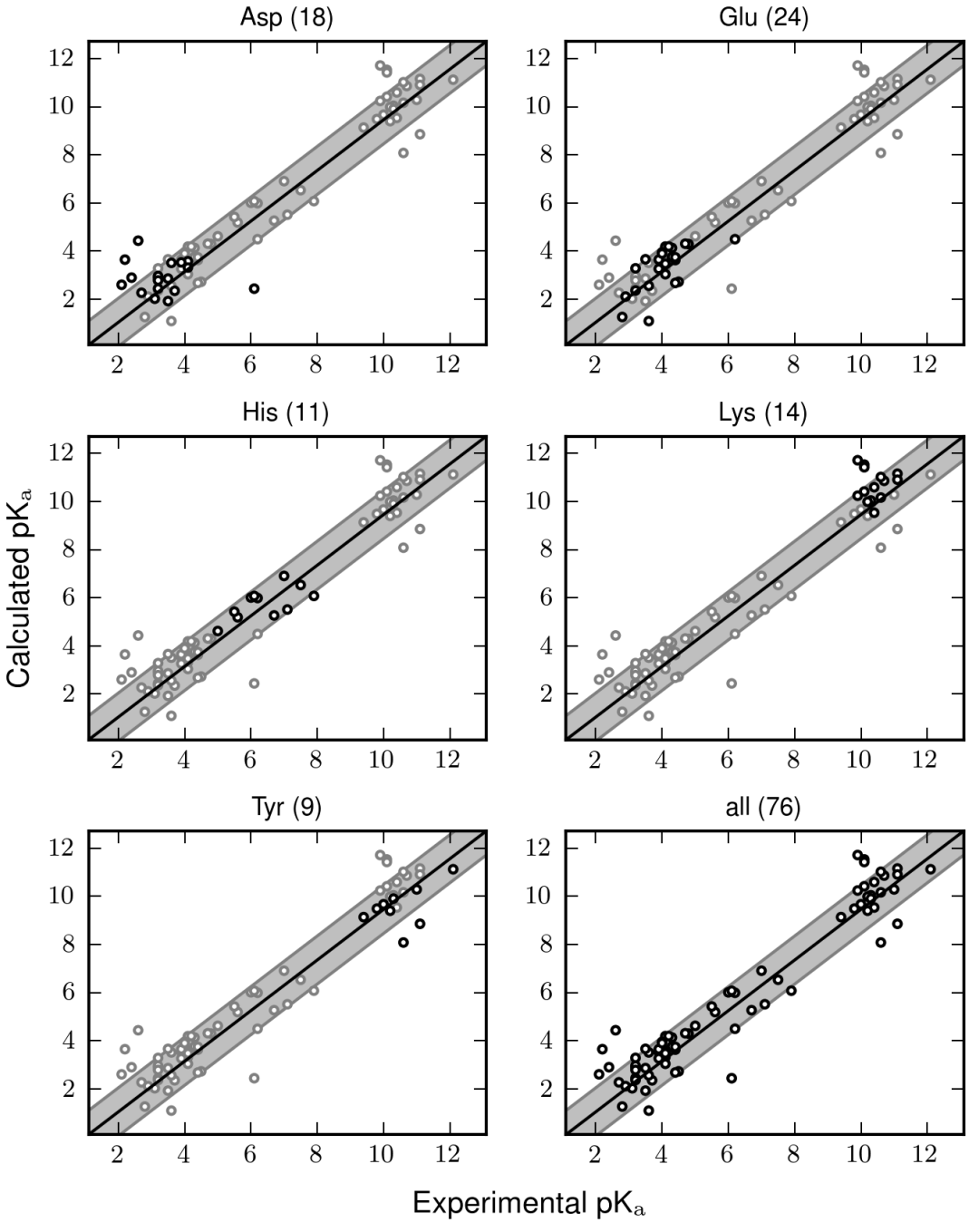


- Link sites by interaction energy
- Clusters: connected components
- Modify one cluster at random
- Pick move distance from geometric distribution

name	PDB ID	residues	atoms	sites
BPTI	4PTI	58	891	18
OMTKY3	2OVO	56	813	15
HEWL	2LZT	129	1965	30
RNase A	3RN3	124	1865	34
RNase H	2RN2	155	2474	53

- DoFs: 10,000–30,000
- Energy calc time: 10 s
- Much less memory than classical direct methods
- Much faster solves than iterative methods
- Precomputation is still somewhat expensive





RMSD	ε_1 4	8	20
BPTI	1.47	0.96	0.82
OMTKY3	1.77	1.07	1.09
HEWL	2.52	1.49	0.79
RNase A	3.22	2.25	0.85
RNase H	4.53	2.53	1.36

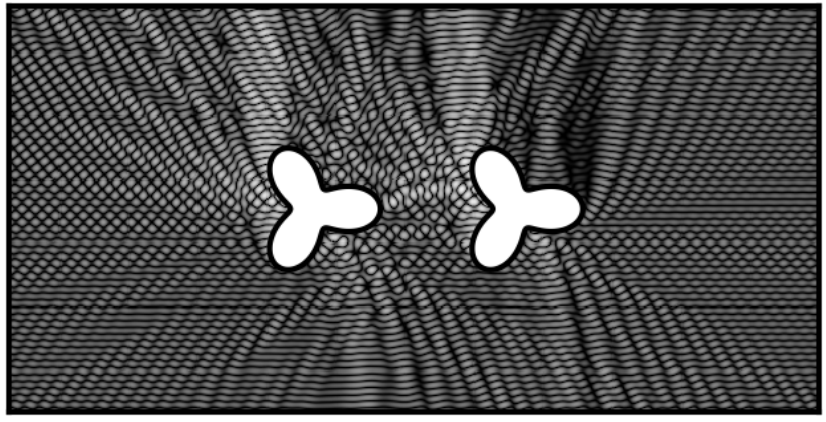
type	number	RMSD
Arg	12/18	1.23
Glu	17/24	1.00
His	8/11	0.92
Lys	11/14	0.79
Tyr	7/ 9	1.24
all	55/76	1.05

How to improve our pK_a predictions?

- One possibility is to include conformational flexibility (Gunner et al.)
- Treat with perturbative techniques

How to improve our pK_a predictions?

- One possibility is to include conformational flexibility (Gunner et al.)
- Treat with perturbative techniques



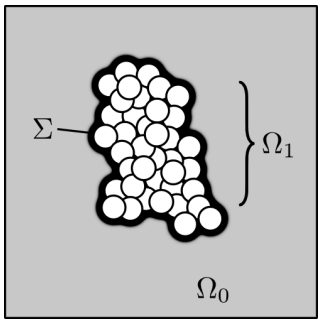
Similar ideas are also relevant for other biological problems.

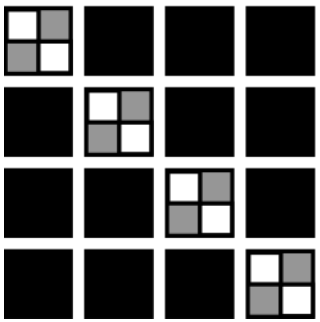
- Structure prediction: fixed backbone, rotamer optimization
- Rigid-body docking: like multiple scattering
- Flexible docking: combination of the above

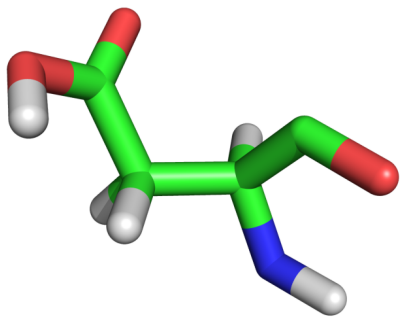
These are all characterized by much larger search spaces and hence enable more efficient amortization of the matrix precomputation costs.

Summary

- Molecular electrostatics: second-kind boundary integral equation
- Fast direct solver for non-oscillatory integral equations
  - Kernel-independent: Laplace, Stokes, Yukawa, low-frequency Helmholtz
  - Very fast solves following precomputation (~ 0.1 s)
- Application to protein pK_a calculations

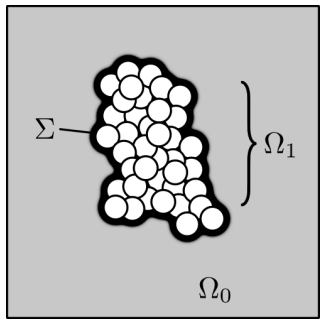


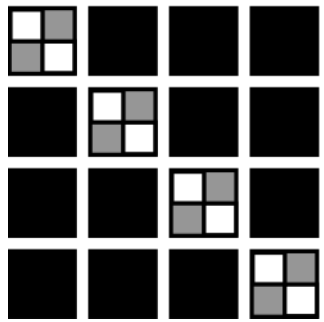


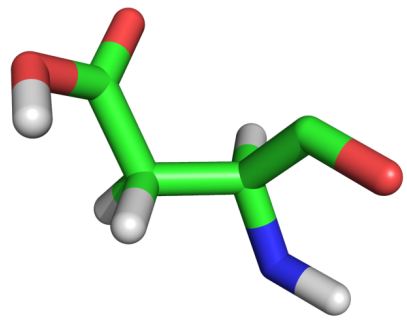


Summary

- Molecular electrostatics: second-kind boundary integral equation
- Fast direct solver for non-oscillatory integral equations
  - Kernel-independent: Laplace, Stokes, Yukawa, low-frequency Helmholtz
  - Very fast solves following precomputation (~ 0.1 s)
- Application to protein pK_a calculations







Next steps:

- Faster direct solvers: aim for $\mathcal{O}(N \log N)$
- Other compression-based numerical algorithms
- More realistic electrostatics: inhomogeneous dielectrics, solvent correlations
- Further biological applications: structure prediction, docking