THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

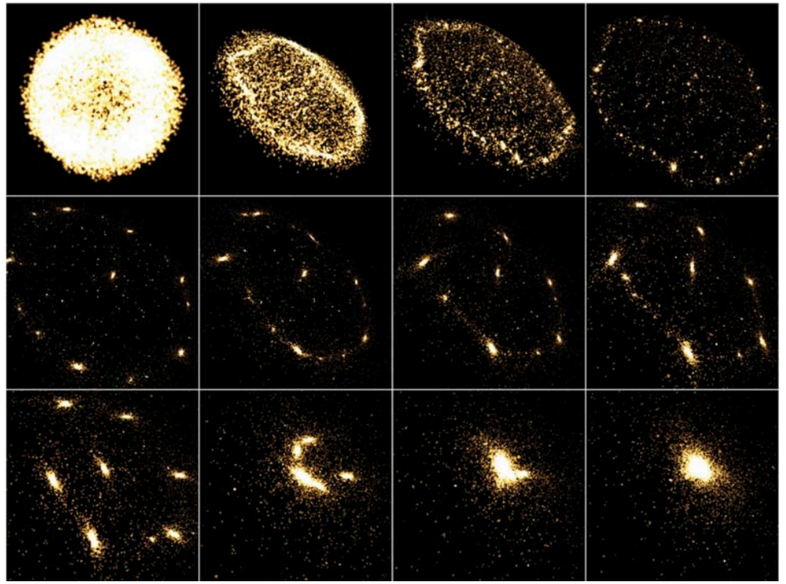
A high-performance and highly reusable fast multipole method library and its application to solvation energy calculations at virus-scale

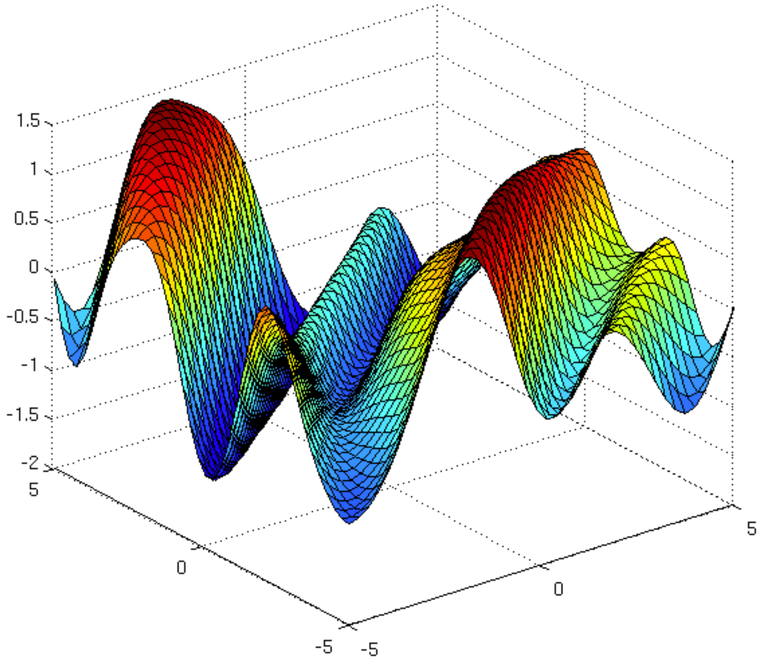
Tingyu Wang

Department of Mechanical and Aerospace Engineering
The George Washington University

N-body problems

- Particles interact with each other: stars, galaxies, atoms, data points
  - Astrophysics
  - Molecular dynamics
  - Machine learning





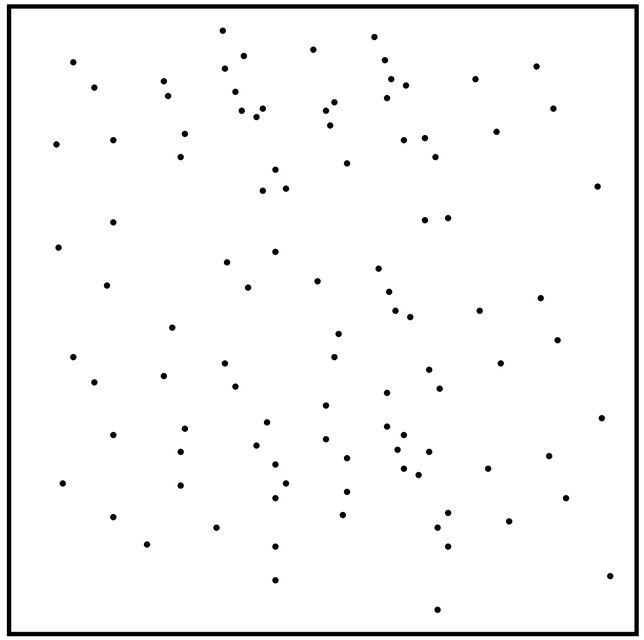
Figures from:
https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

N-body problems

- Compute potentials induced by all charges

$$\phi_i = \sum_{j=1}^{N} G\left(\mathbf{x}_i, \mathbf{x}_j\right) q_j \quad i = 1, \ldots, N$$

- Direct method: dense mat-vec, $\mathcal{O}(N^2)$
- 1986: Barnes-Hut algorithm (tree-code): $\mathcal{O}(N \log N)$
- 1987: Fast Multipole method (**FMM**): $\mathcal{O}(N)$
- Top 10 algorithms of 20th century (*Dongarra & Sullivan, 2000*)



position $\mathbf{x}_i$ charge q_j

kernel function, for example, $G\left(\mathbf{x}, \mathbf{y}\right) = \dfrac{1}{\|\mathbf{x} - \mathbf{y}\|}$

Advancement of FMM over the past 30 years

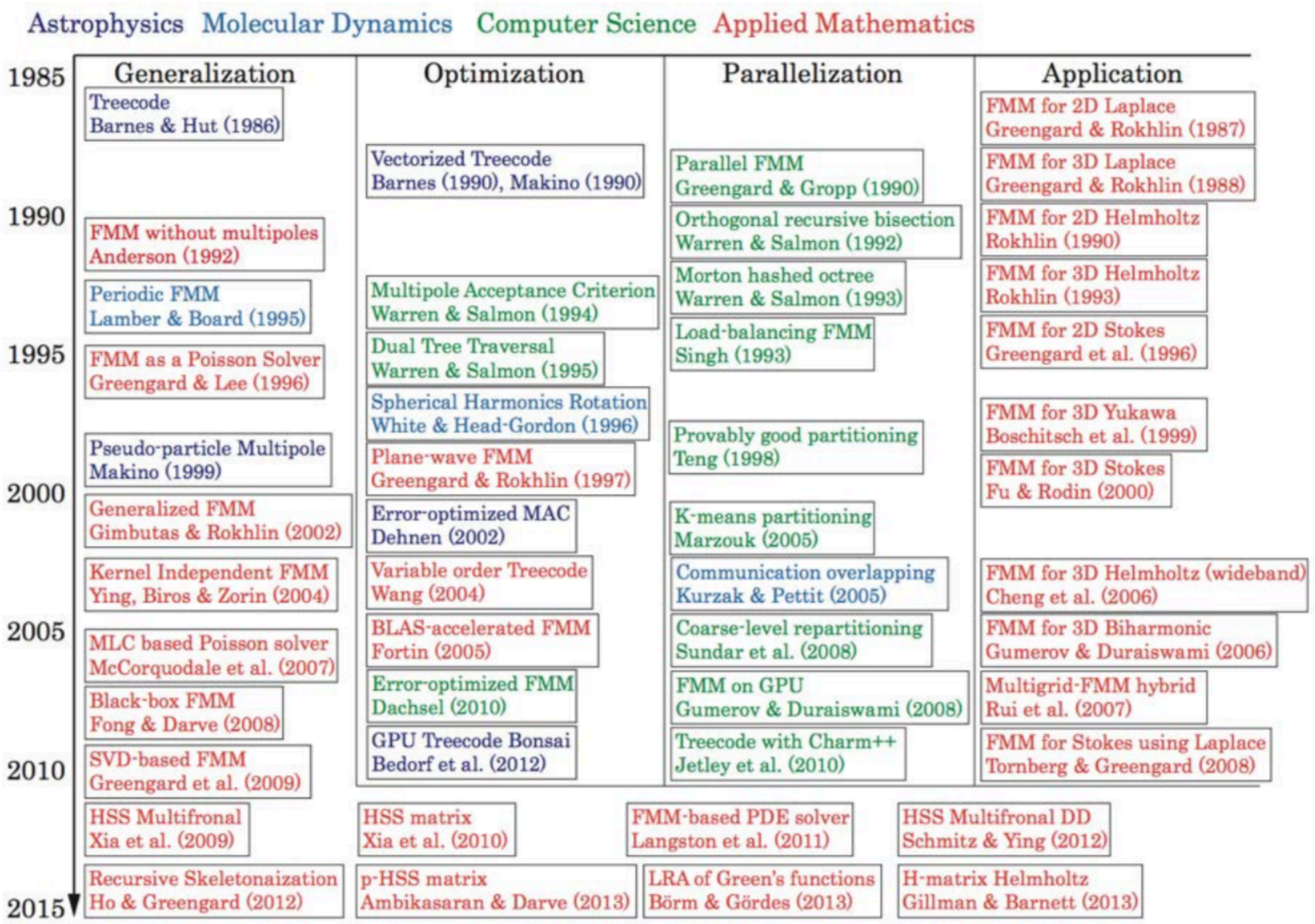


Figure from: Dr. Yokota's presentation

Challenges

- Many high-performance implementations are experimental and thus not easy to use
  - Users need to have some expertise in programming to compile them
  - Applications have to written in low-level languages (C, C++, Fortran)
  - Open-source but no-longer actively maintained

- Lack a standard open-source FMM package that people outside of the FMM community can easily use

Motivation

- Develop ExaFMM that maximizes both performance and usability
  - Understand and combine many individual contributions in this field
  - Carefully design the software to make it highly reusable
    - Portable
    - Extensible
    - Provide interfaces in high-productivity language

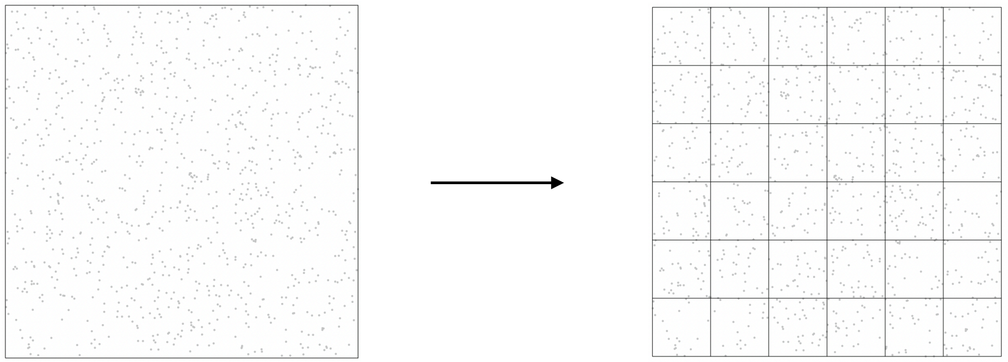
- Offer users an easy entry to this intricate algorithm

Introduction to kernel-independent FMM

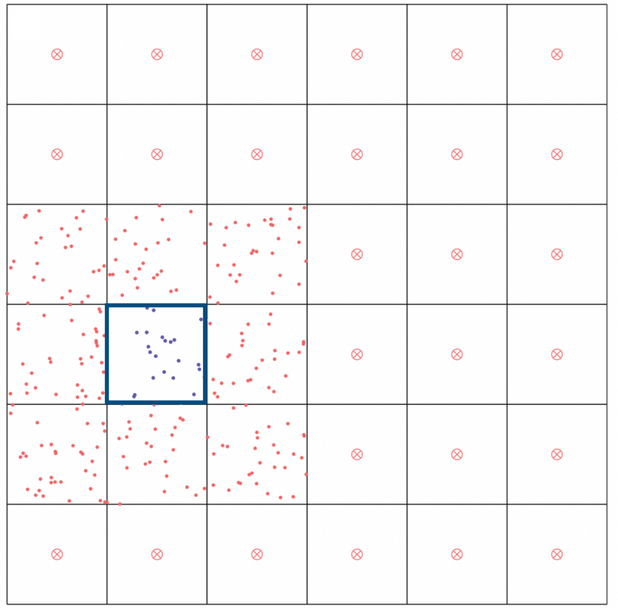
FMM Algorithm

- Idea 1: a tree structure that partitions the domain adaptively

N particles



Each leaf box has O(1) particles

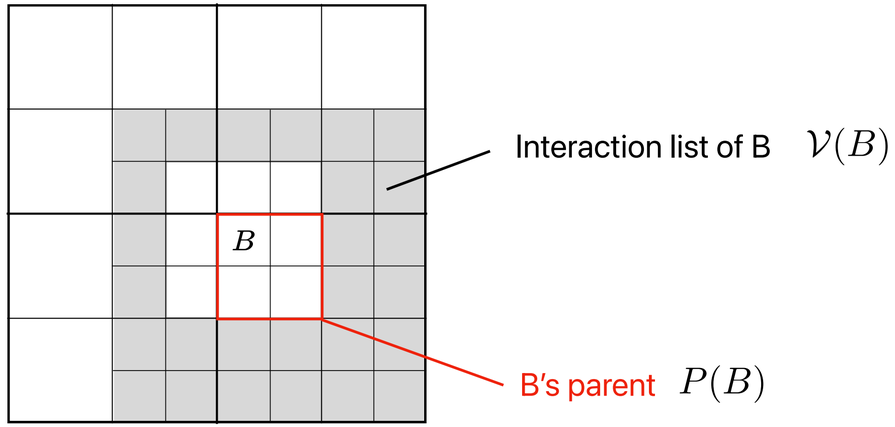


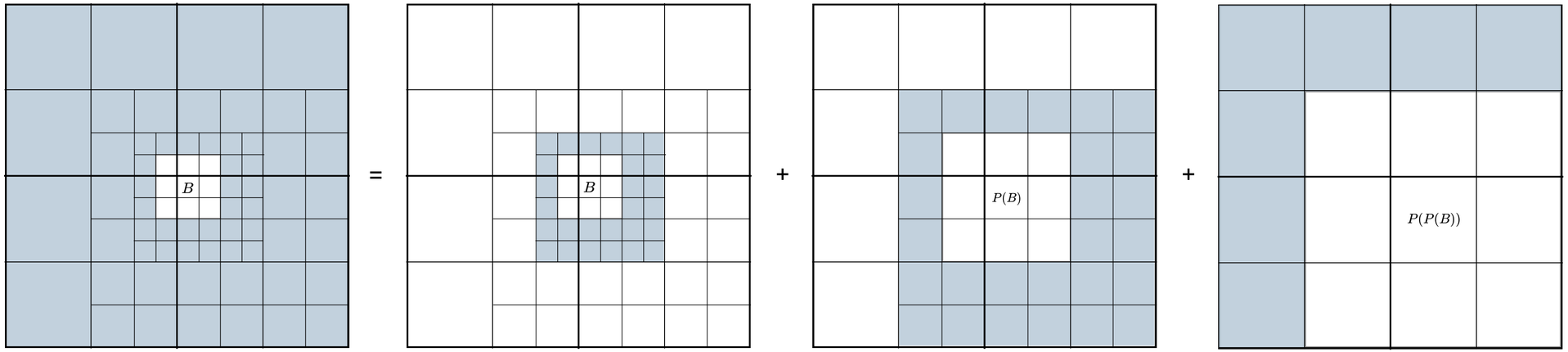
- Idea 2: split all interactions into near- and far-field contributions
  - Evaluate the near-field component directly (particle-to-particle)
  - Evaluate the far-field contribution using low-rank methods (box-to-box)

Figure from: Martinsson (2015), doi.org/10.1007/978-3-540-70529-1_448

FMM Algorithm

- Idea 3: interaction list





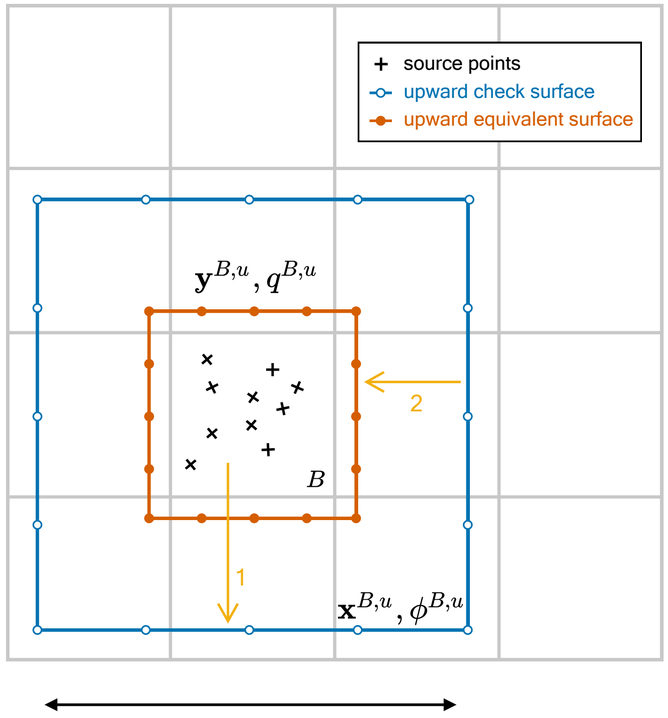
$\mathcal{F}(B)$ $\mathcal{V}(B)$ $\mathcal{V}(P(B))$ $\mathcal{V}(P(P(B)))$

Kernel-independent FMM (KIFMM)

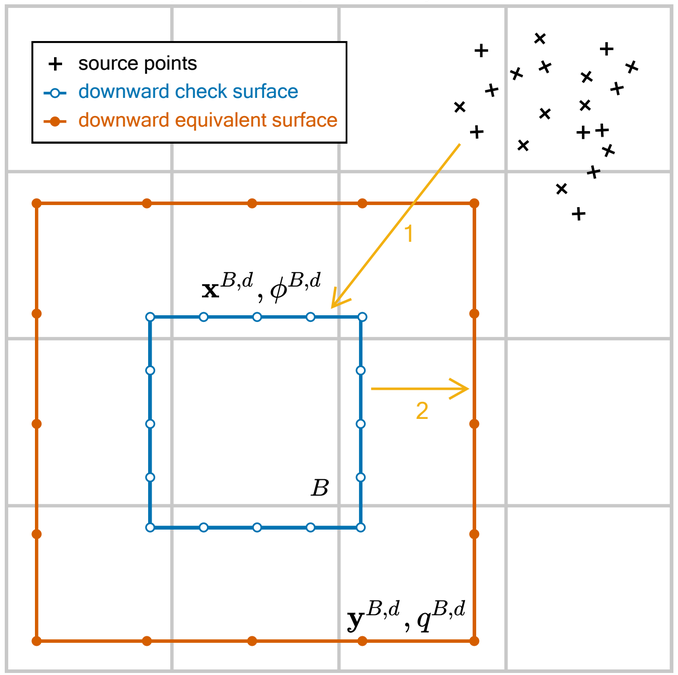
- Low rank representation in KIFMM: equivalent charges (on the red boxes below)

represent the influence from the charges in B to the potential in B's far-field.

upward equivalent density
(multipole expansion)



downward equivalent density
(local expansion)



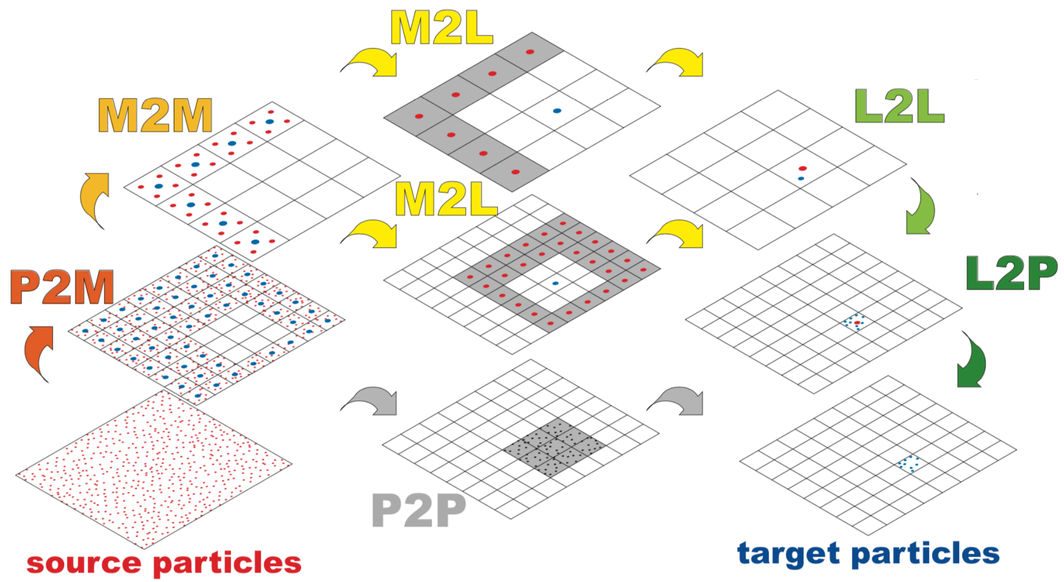
represent the influence from the charges in B's far-field to the potential in B

P

number of discretization points on each side
or (FMM expansion order)

The complete algorithm

- Construct the tree
- P2M (particle-to-multipole) for all leaf boxes
- M2M in post-order tree traversal
- M2L for all boxes
- L2L in pre-order tree traversal
- L2P (local-to-particle) for all leaf boxes
- P2P (particle-to-particle) for all leaf boxes (near-field interactions)

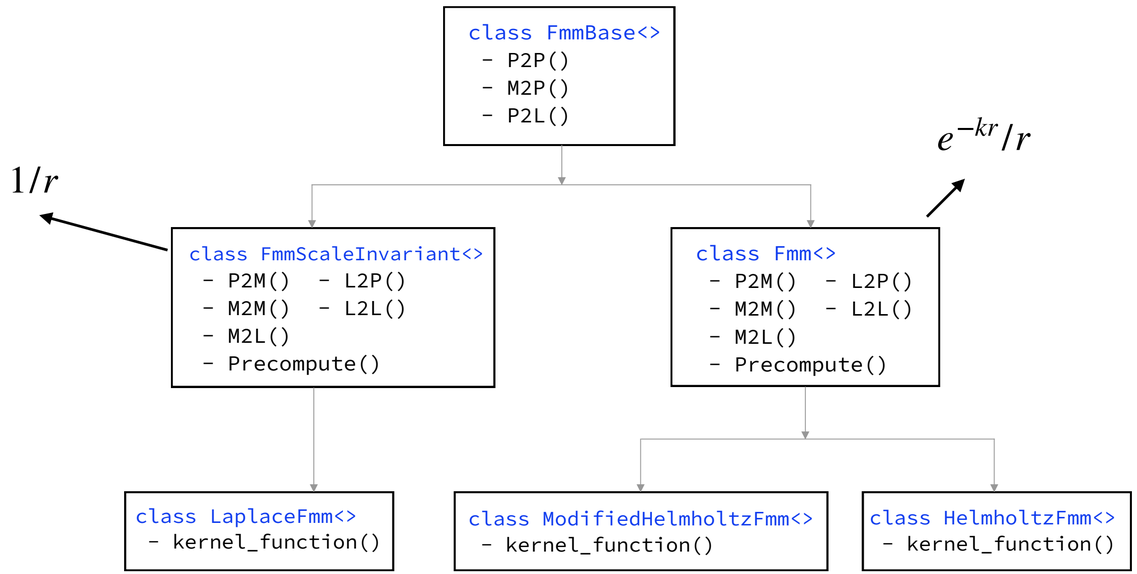


ExaFMM’s features and performance

ExaFMM software features

- ExaFMM-t: https://github.com/exafmm/exafmm-t
  - Laplace, Yukawa, Helmholtz kernels
  - Compute both potential and gradient

- Easy to extend to new kernels



- Moderately object-oriented, easy to understand
  - Few user-defined types (`class Node`, `Body` and `Fmm`)
  - Simple data structures
- Portable
  - Only use C++ STL containers
  - Standard dependencies:
    - BLAS, LAPACK
    - FFTW3
    - OpenMP
- Straightforward parallelism
  - M2M, L2L: OpenMP task
  - Other operators: OpenMP loop

ExaFMM software features

- Concise in terms of lines of code

code	# lines
exafmm-t	6k
TBFMM	16k
ScalFMM	70k
PVFMM	20k

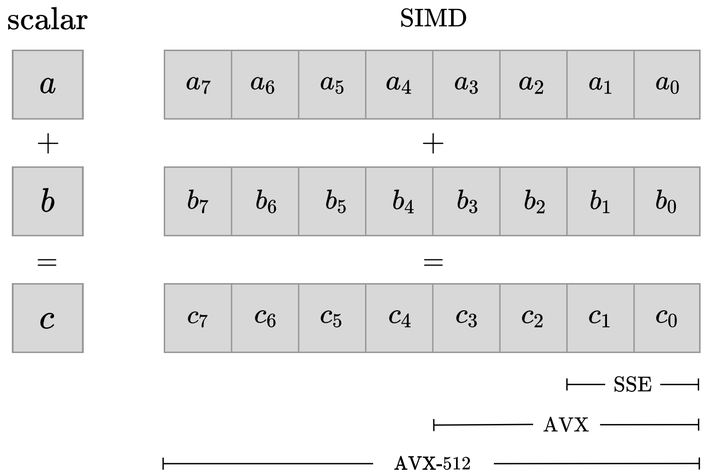
- High-level interface in Python

```python
import exafmm.laplace as laplace

# create sources and targets
# src_coords, src_charges are NumPy arrays
sources = laplace.init_sources(src_coords, src_charges)
targets = laplace.init_targets(trg_coords)

fmm = laplace.LaplaceFmm(p=10, ncrit=200, filename="laplace.dat")
tree = laplace.setup(sources, targets, fmm)

# trg_values is a NumPy array with potentials and gradients
trg_values = laplace.evaluate(tree, fmm)
```

- Access almost all C++ data structure via Python interface
  - Multipole expansion coefficients as NumPy arrays
  - Interaction list as a Python list
- Kernel, tree, list and full FMM tests

Optimizations

- Vectorization on P2P (near-field interactions)



- Performance improvement due to manual vectorization
  - Test case: P2P with 2×10^4 particles
  - 1 core of Intel Xeon 6148 CPU, support AVX-512

Single-precision

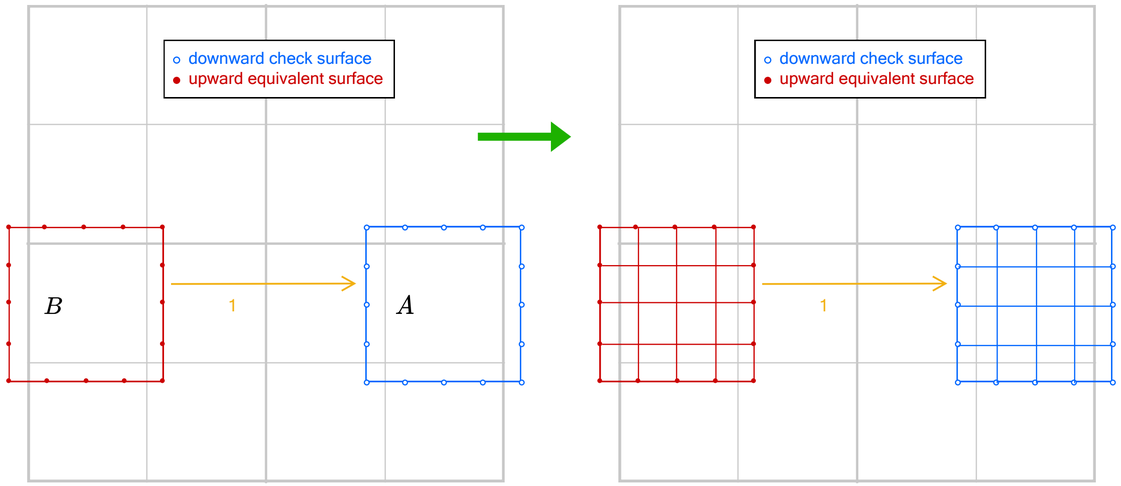
	Time with auto-vectorization (s)	Time with manual-vec (s)	Speedup
Laplace	1.19	0.124	9.6
Yukawa	4.64	0.308	15.1
Helmholtz	11.9	0.745	16.0

Double-precision

	Time with auto-vectorization (s)	Time with manual-vec (s)	Speedup
Laplace	1.37	0.271	5.1
Yukawa	5.58	0.729	7.7
Helmholtz	13.8	1.77	7.8

Optimizations

- Using FFT to accelerate M2L, M2L is memory-bound



- Performance improvement due to cache optimization on Hadamard (element-wise) product in M2L (from PVFMM)
  - Perform M2L interaction in sibling groups
  - Test case: 10^6 particles randomly distributed in unit cube, 5-level tree, on Intel i9-7940X CPU

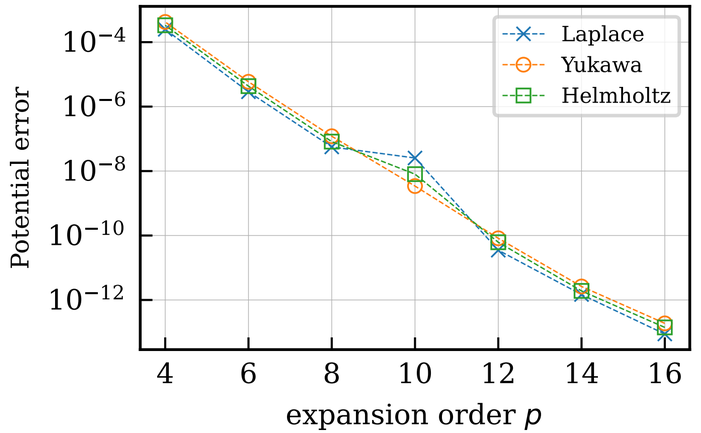
Hadamard product time

P	time w/o optimization (s)	time with optimization (s)	Speedup
4	0.16	0.064	2.5
7	1.99	0.52	3.8
10	6.44	1.04	6.2

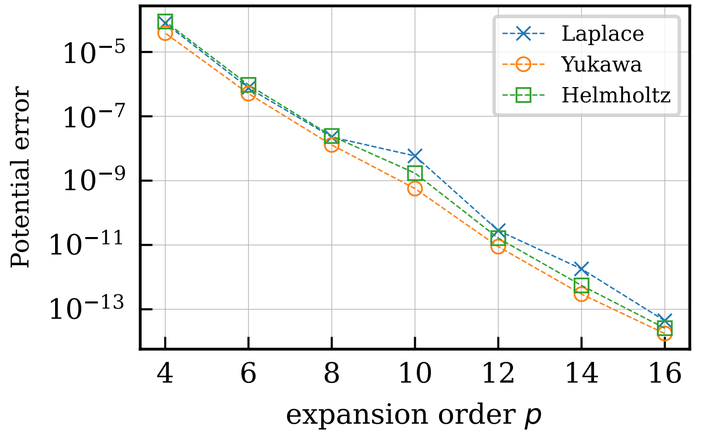
Convergence

- Using all three kernels available in ExaFMM
- 1 million particles:
  - Uniform distribution
  - Non-uniform distribution
- FMM order p from 4 to 16
- Observed exponential convergence

Uniform distribution



Non-uniform distribution

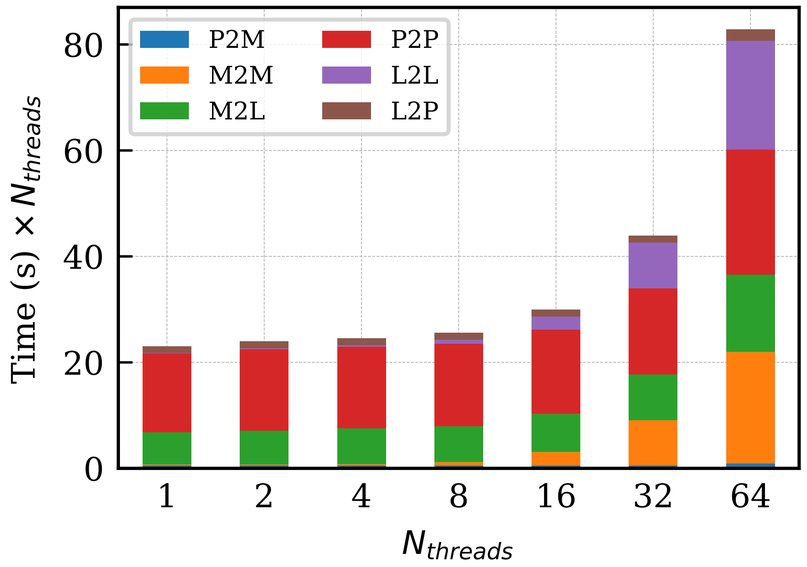


Scalability

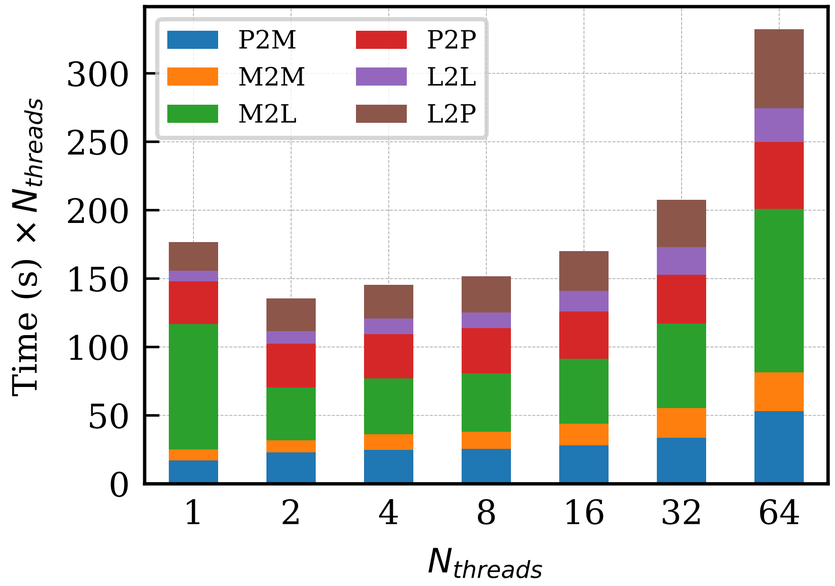
- 64-core Intel Xeon Phi 7210 CPU, support AVX-512
- Laplace problem with 10^6 particles
- tree construction time not included
- Two accuracy settings:
  - P = 4
  - P = 16

- parallel efficiency:
  - Single-precision: 52% with 32 threads
  - Double-precision: 85% with 32 threads

p = 4, single-precision



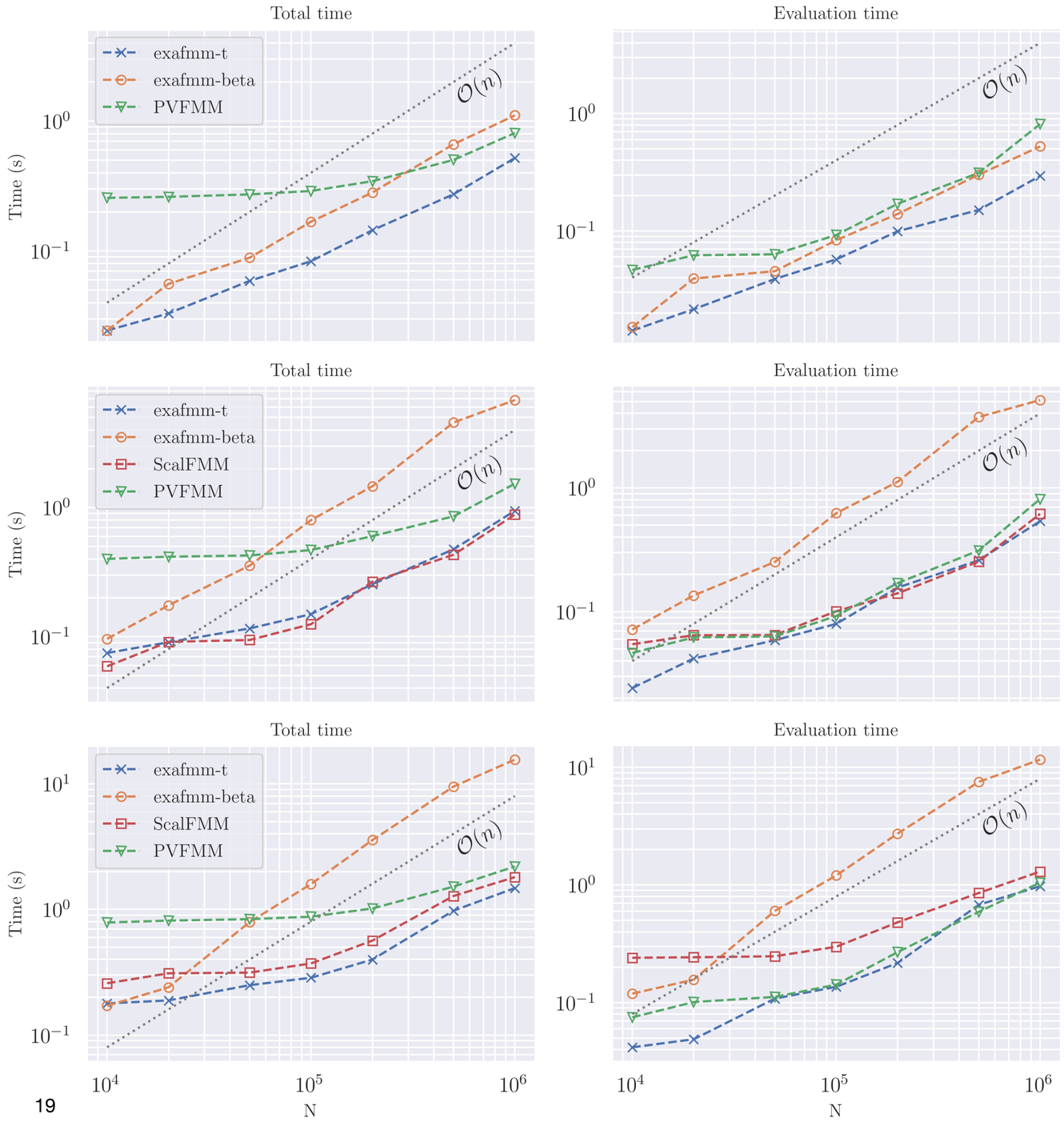
p = 16, double-precision



Compare with other codes

- Problem sizes: 10^4 to 10^6 particles
- 14-core Intel i9-7940X CPU
- three levels of accuracy
  - achieve 4, 7, 10 digits of accuracy in potential
- Optimal N_{crit} for each case
- Total time: include tree construction
- Evaluation time: only include FMM operators

	exafmm-t	exafmm-beta	PVFMM	ScalFMM
Expansion Type	equivalent charges	spherical harmonics	equivalent charges	interpolation
Threading Model	OpenMP	Intel TBB	OpenMP	OpenMP
SIMD	AVX-512	AVX-512	AVX	AVX-512
Language	C++	C++	C++	C++



FMM-accelerated Boundary Element method in biomolecular electrostatic applications

Joint work with:

Dr. Christopher Cooper (USM)

Dr. Timo Betcke (UK)

Boundary Element Method (BEM)

- Partial differential equation in domain Ω with Dirichlet boundary conditions on Γ

$$\Delta u(\mathbf{x}) = 0 \quad \text{in } \Omega$$
$$u(\mathbf{x}) = f \quad \text{on } \Gamma$$

- Boundary integral formulation

Green's function

$$u(\mathbf{x}) = \int_\Gamma G(\mathbf{x},\mathbf{y}) \frac{\partial}{\partial \mathbf{n}} u(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y}) - \int_\Gamma \frac{\partial}{\partial \mathbf{n}} G(\mathbf{x},\mathbf{y}) u(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y}) \quad \text{in } \Omega$$

- Limit $\mathbf{x}$ to the boundary:

$$\frac{1}{2} u(\mathbf{x}) = \int_\Gamma G(\mathbf{x},\mathbf{y}) \frac{\partial}{\partial \mathbf{n}} u(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y}) - \int_\Gamma \frac{\partial}{\partial \mathbf{n}} G(\mathbf{x},\mathbf{y}) u(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y}) \quad \text{on } \Gamma$$

- Rewrite using boundary operators notations:

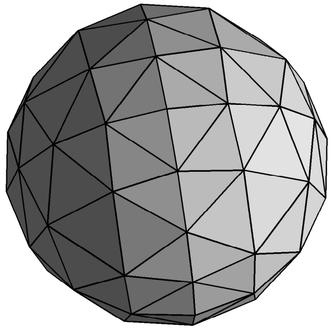
$$[V] \frac{\partial}{\partial \mathbf{n}} u = \left[\frac{I}{2} + K \right] u$$

$$[Vu](\mathbf{x}) := \int_\Gamma G(\mathbf{x},\mathbf{y}) u(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y})$$

Single-layer potential boundary operator

$$[Ku](\mathbf{x}) := \int_\Gamma \frac{\partial}{\partial \mathbf{n}} G(\mathbf{x},\mathbf{y}) u(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y})$$

Double-layer



Boundary Element Method (BEM)

- With Galerkin discretization

$$[\mathrm{V}]\left[\frac{\partial u}{\partial \mathbf{n}}\right] = \left[\frac{1}{2}\mathrm{M} + \mathrm{K}\right][u]$$

where

$$\mathrm{V}_{ij} = \int_\Gamma \psi_i(\mathbf{x}) \int_\Gamma G(\mathbf{x}, \mathbf{y}) \phi_j(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{y}) \mathrm{d}\Gamma(\mathbf{x})$$

$$\approx \sum_t \psi_i\left(\mathbf{x}_t\right) \left[\sum_s G\left(\mathbf{x}_t, \mathbf{y}_s\right) \phi_j\left(\mathbf{y}_s\right) w_s \right] w_t$$

regular Gauss quadrature points

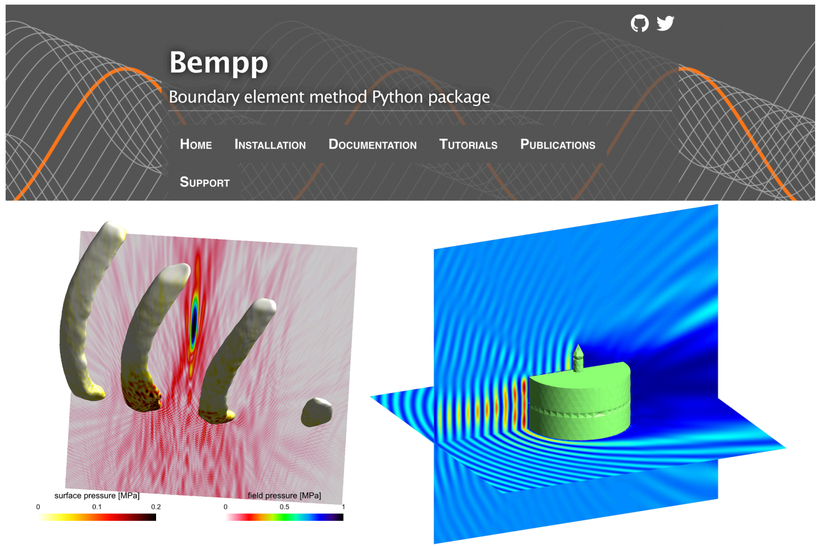
Bottleneck: V is dense, many mat-vec in the iterative solver

Conventional BEM: $\mathcal{O}(N^2)$ in time and space

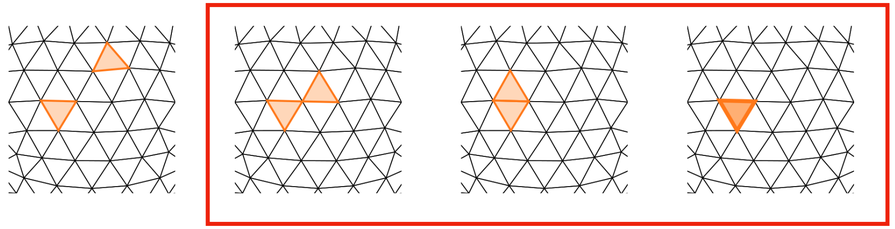
FMM-BEM: $\mathcal{O}(N)$

Bempp

- Open-source BEM platform
  - Python interface
  - OpenCL computational backends
  - Allows just-in-time compilation



- Interface with Bempp



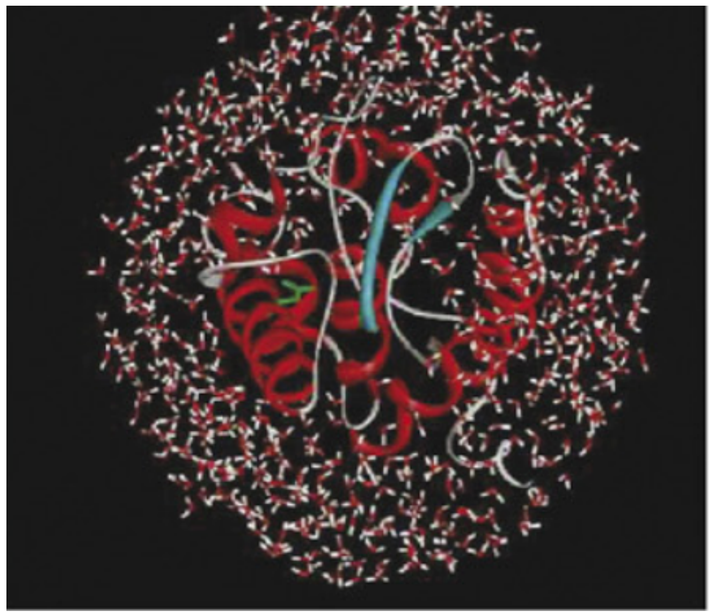
Singular integrals

$$Ax = P_{\text{test}}^T (G - C) P_{\text{trial}} x + Sx$$

- A: Matrix representation of a discretized boundary operator
- G: all interactions between all regular quadrature points (FMM)
- C: remove interactions between adjacent panels
- S: correct singular integral contributions

Bempp-ExaFMM in Biomolecular Electrostatic Applications

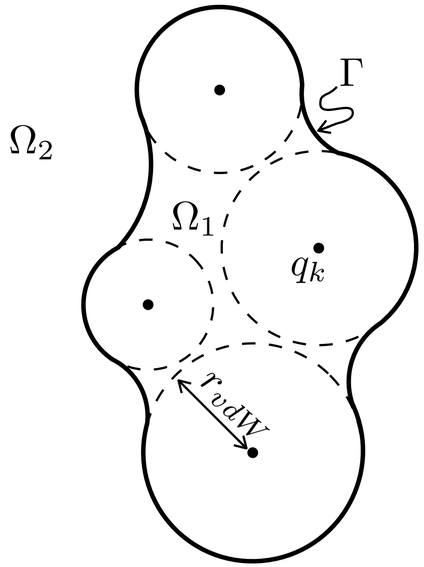
- What is solvation effect?
  - Solvation is the interaction between a solvent (water) and molecules or ions of a solute (protein).
  - Among various interactions, electrostatic interaction is one of the most important component.
- Why it is import?
  - Understand problems such as protein binding and folding, drug delivery, ...
- How to model the electrostatic interactions:
  - Treat water molecules explicitly: ex. Molecular dynamics
    - 4k compute nodes to simulate a H1N1 virus (*Durrant, 2020*)
  - Represent water as continuum medium: implicit solvent model
    - Classical theory of continuum electrostatics applies, we can use BEM



A. Warshel et al. (2006)

Bempp-ExaFMM to make virus-scale simulations accessible to every researcher

Implicit solvent model



Solute Region Ω_1: proteins

Solvent Region Ω_2 : water with ions

- Governing equations:

$$\Delta\phi_1 = \frac{1}{\epsilon_1}\sum_k q_k\delta\left(\mathbf{r},\mathbf{r}_k\right) \qquad \text{in } \Omega_1$$

$$\left(\Delta - \kappa^2\right)\phi_2 = 0 \qquad \text{in } \Omega_2$$

with interface conditions on Γ:

$$\phi_1 = \phi_2$$

$$\epsilon_1\frac{\partial\phi_1}{\partial\mathbf{n}} = \epsilon_2\frac{\partial\phi_2}{\partial\mathbf{n}}$$

- Compute solvation energy from the reaction potential

$$\Delta G^{\text{polar}}_{\text{solv}} = \frac{1}{2}\sum_{k=1}^{N_q} q_k\phi_{\text{reac}}\left(\mathbf{r}_k\right)$$

with $\phi_{\text{reac}} = \phi_1 - \phi_{\text{Coulomb}}$.

Boundary integral formulations

- Direct formulation (*Yoon and Lenhoff, 1990*), poorly conditioned but cheaper per iteration

$$\begin{bmatrix} \frac{1}{2}I + K_L & -V_L \\ \frac{1}{2}I - K_Y & \frac{\epsilon_1}{\epsilon_2}V_Y \end{bmatrix} \begin{bmatrix} \phi_{1,\Gamma} \\ \frac{\partial}{\partial \mathbf{n}}\phi_{1,\Gamma} \end{bmatrix} = \begin{bmatrix} \frac{1}{\epsilon_1}\sum_{k=1}^{N_q} \frac{q_k}{4\pi|\mathbf{r}_\Gamma - \mathbf{r}_k|} \\ 0 \end{bmatrix}$$

- Derivative formulation with exterior field (*Lu et al., 2006*), well-conditioned but operators are more involved

$$\begin{bmatrix} \frac{1}{2}\left(1+\frac{\epsilon_1}{\epsilon_2}\right)I - K_Y + \frac{\epsilon_1}{\epsilon_2}K_L & V_Y - V_L \\ \frac{\epsilon_1}{\epsilon_2}D_Y - \frac{\epsilon_1}{\epsilon_2}D_L & \frac{1}{2}\left(1+\frac{\epsilon_1}{\epsilon_2}\right)I + \frac{\epsilon_1}{\epsilon_2}T_Y - T_L \end{bmatrix} \begin{bmatrix} \phi_{2,\Gamma} \\ \frac{\partial}{\partial \mathbf{n}}\phi_{2,\Gamma} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{N_q} \frac{q_k}{4\pi\epsilon_2|\mathbf{r}_\Gamma - \mathbf{r}_k|} \\ \sum_{k=1}^{N_q} \frac{\partial}{\partial \mathbf{n}_\mathbf{r}}\left(\frac{q_k}{4\pi\epsilon_2|\mathbf{r}_\Gamma - \mathbf{r}_k|}\right) \end{bmatrix}$$

- Thanks to Bempp user-friendly interface, we can try multiple formulations in our study with modest effort.

```
ep = ep_ex/ep_in
A = bempp.api.BlockedOperator(2,2)
A[0,0] = 0.5*(1+1/ep)*id - (dlp_y - 1/ep*dlp_l)
A[0,1] = slp_y - slp_l
A[1,0] = 1/ep*(hyp_y - hyp_l)
A[1,1] = 0.5*(1+1/ep)*id + (1/ep)*adj_y - adj_l
```

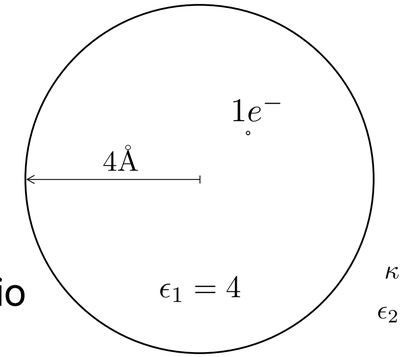
Subscript L, Y denote Laplace and Yukawa kernels

$$G_L(\mathbf{x},\mathbf{y}) = \frac{1}{4\pi|\mathbf{x}-\mathbf{y}|},$$

$$G_Y(\mathbf{x},\mathbf{y}) = \frac{e^{-\kappa|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|}$$

Code verification via mesh refinement studies

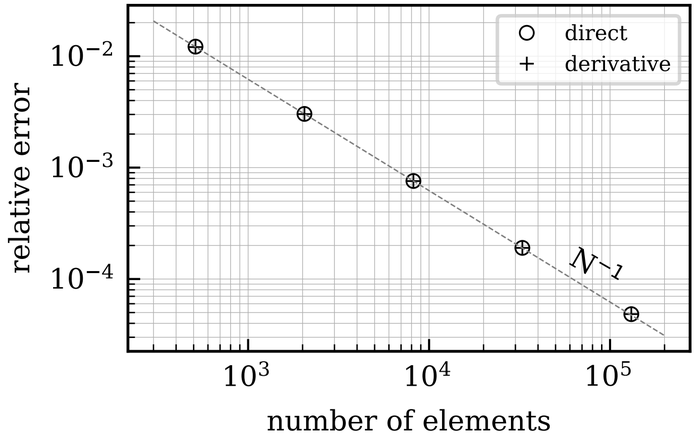
- Problem setup:
  - Sphere with an off-center charge
  - Real biomolecule: 5PTI
- 5 meshes with a constant refinement ratio
- Parameters
  - FMM order set to 10
  - 6 regular quadrature points
- Observed **linear convergence** with respect to number of elements in both formulations



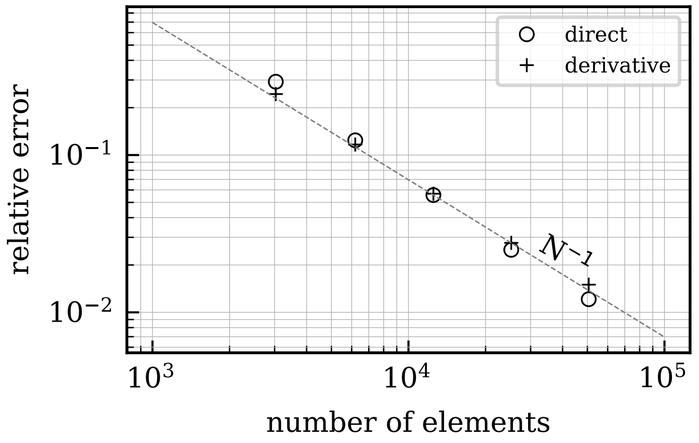
Sphere

5PTI

Error w.r.t *analytical* solution (Kirkwood, 1934)



Error w.r.t *extrapolated* solution



Result comparison with trusted software

- 9 proteins and compare the computed solvation energy with APBS.
- APBS: finite-difference PB solver, the “standard” software in biophysics community
- Use 3 meshes to obtain an extrapolated solvation energy for each software

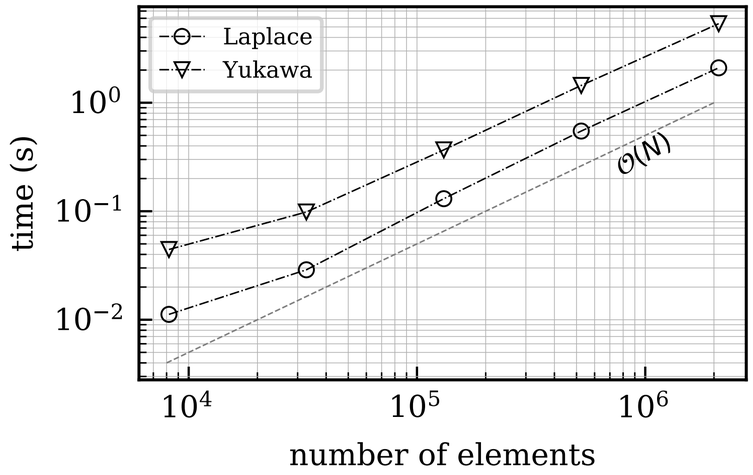
		APBS					Bempp				
ID	N_{atoms}	coarse	Error medium	fine	ΔG_{solv}	Order$(1/h)$	coarse	Error medium	fine	ΔG_{solv}	Order$(1/N)$
1AJJ	513	4.40e-02	1.33e-02	4.00e-03	-266.15	1.73	2.75e-02	1.12e-02	4.59e-03	-268.67	1.29
1VJW	826	3.09e-02	1.45e-02	6.77e-03	-297.06	1.09	4.92e-02	2.23e-02	1.01e-02	-302.50	1.14
5PTI	892	3.68e-02	1.38e-02	5.14e-03	-311.69	1.42	5.12e-02	2.23e-02	9.67e-03	-314.34	1.20
1R69	997	4.31e-02	2.10e-02	1.02e-02	-261.02	1.04	5.09e-02	2.34e-02	1.08e-02	-265.02	1.12
1A2S	1272	5.25e-02	2.40e-02	1.10e-02	-456.56	1.13	4.21e-02	1.93e-02	8.86e-03	-461.25	1.12
1SVR	1433	5.28e-02	2.21e-02	9.28e-03	-393.45	1.25	6.48e-02	2.89e-02	1.29e-02	-398.83	1.16
1A63	2065	4.51e-02	1.88e-02	7.82e-03	-559.39	1.26	5.82e-02	2.60e-02	1.16e-02	-567.24	1.16
1A7M	2804	4.13e-02	1.88e-02	8.53e-03	-524.29	1.14	4.93e-02	2.24e-02	1.02e-02	-531.48	1.14
1F6W	8247	6.45e-02	2.78e-02	1.20e-02	-1277.51	1.21	4.18e-02	1.80e-02	7.76e-03	-1301.08	1.22

The difference in the solvation energy is between 0.8% and 1.8% across 9 proteins.

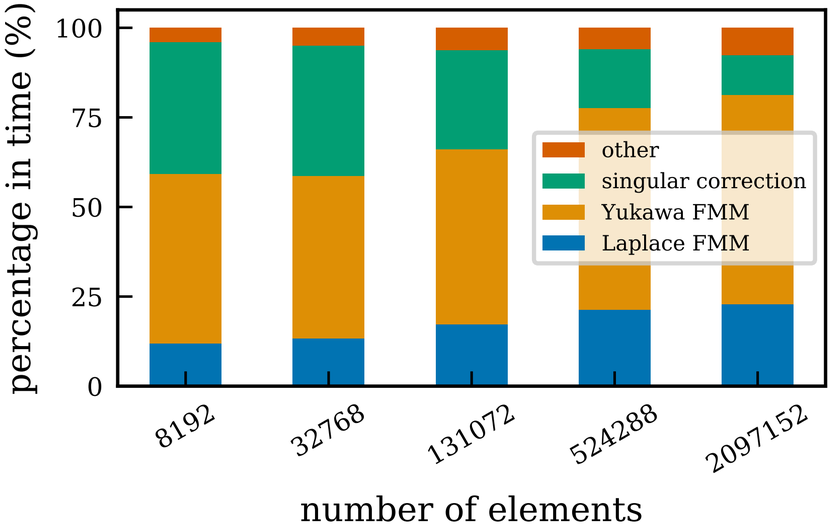
Performance Analysis using a sphere

- Problem setup:
  - Spherical molecule with 100 charges inside
  - Five discretizations: with number of elements from 8k to 2m
  - 6 quadrature points
  - FMM order set to 5

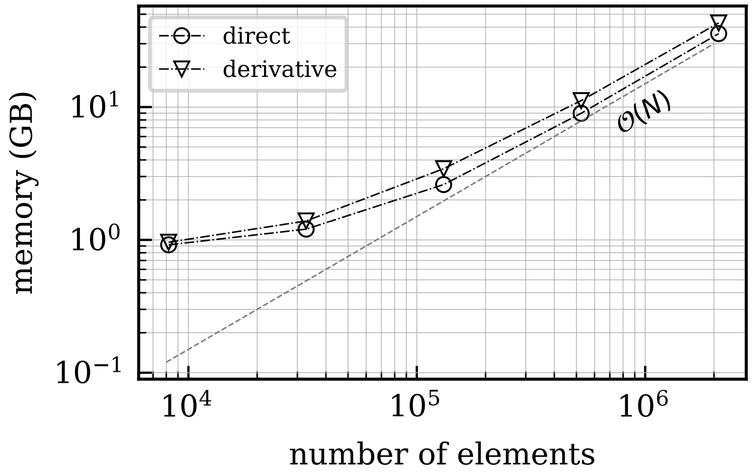
Time spent on 1 Laplace and Yukawa FMM call



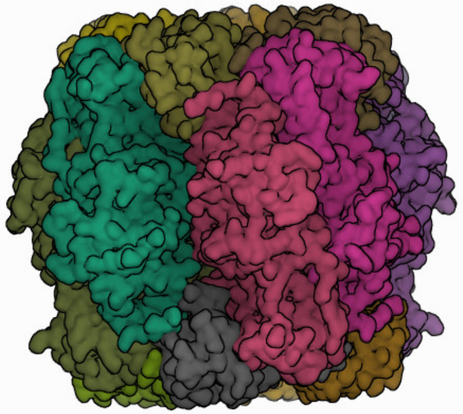
Time breakdown in GMRES



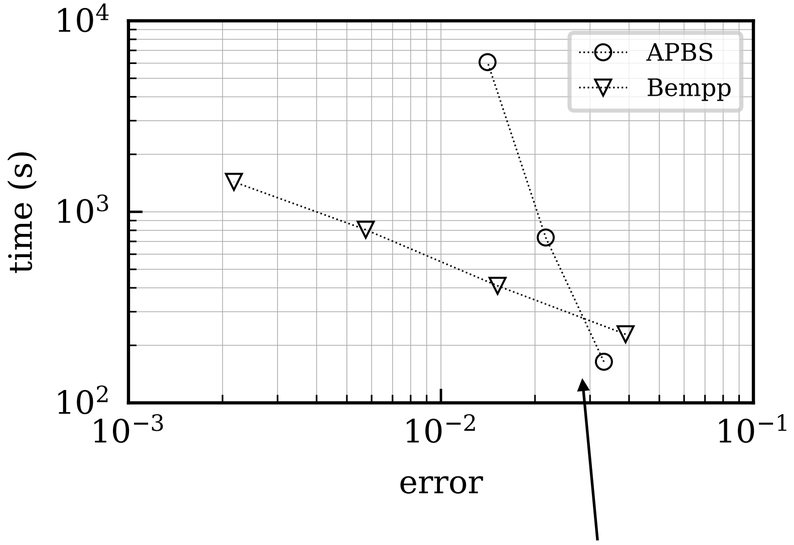
Space complexity



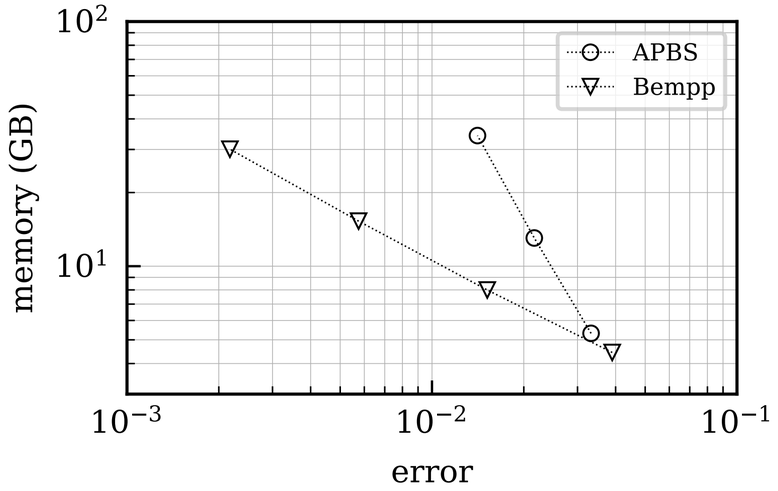
Performance comparison with APBS



- 1RCX: 40k atoms, 130Å in diameter
- A fair comparison: reaching the same accuracy
- Mesh
  - Bempp: 4 meshes, 237k to 2m boundary elements
  - APBS: 3 meshes: Δx set to 0.78, 0.58, 0.44 A
- Error measured against the *extrapolated solution*
- Bempp-ExaFMM has a *better scaling* in both time and space



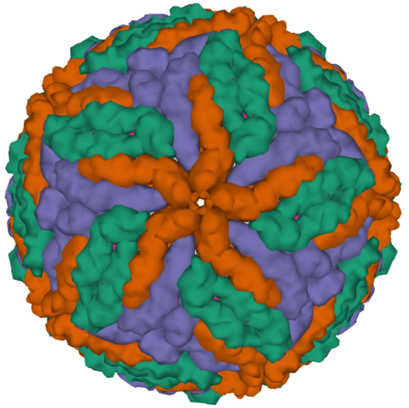
Crossover point: 3% error mark



Boundary element approach displays an advantage in

- solving bigger problems
- applications that require higher accuracy

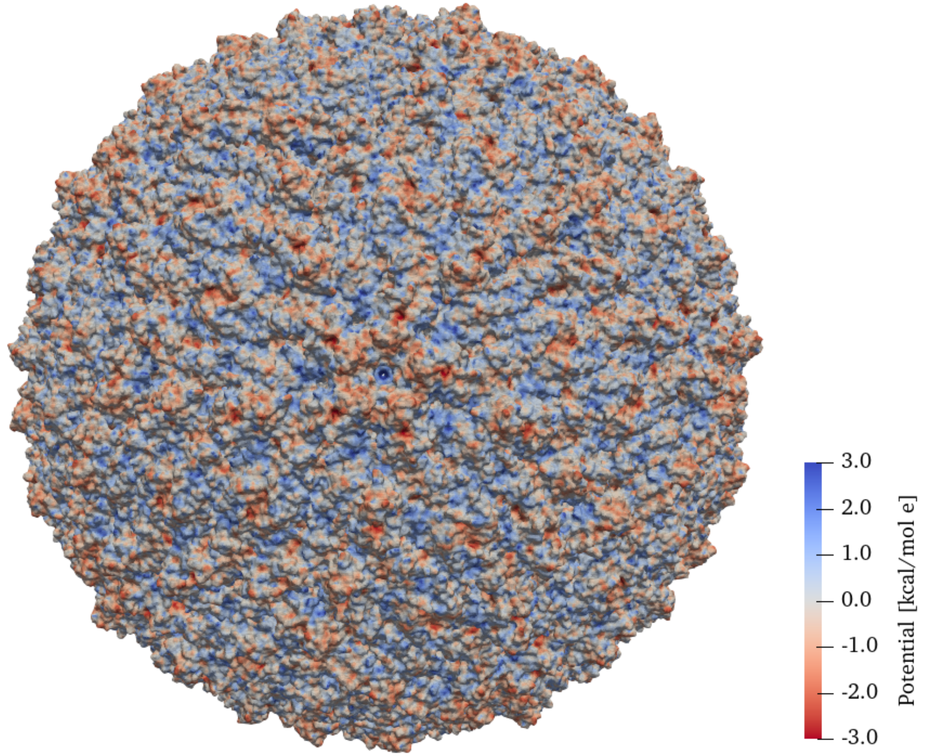
Results for Zika virus



- Structure: 1.6m atoms, Diameter: ~ 470Å
- Mesh: 10m boundary elements
- Parameters:
  - 3 quadrature points per element
  - FMM order of 4 (achieving 4 digits of accuracy)
- Machine: 1 compute node with 40 cores (*GWU pegasus*)
- Derivative formulation: 2hr 20min total time, 80 min in GMRES
- < 1% difference compared to results from *PyGBe* (-117261.1 kcal/mol)

	ΔG_{solv} (kcal/mol)	total time (s)	assembly time (s)	GMRES time (s)	memory (GB)	# iterations
direct	-116587.5	11005.4	1534.5	9470.9	109.7	105
derivative	-116254.9	8370.3	3553.9	4816.4	152.0	18

- N-body problem with 30 million particles:
  - 5s per Laplace FMM call
  - 13s per Yukawa FMM call



Conclusion

- ExaFMM
  - High-order accurate, able to reach 12 digits of accuracy
  - Competitive single-node performance
  - reusable: portability, extensibility, simple dependencies, interfaces in high-productively languages
- Bempp-ExaFMM in biomolecular electrostatic applications:
  - Code Verification
  - Performance comparison with APBS
  - Perform virus-scale simulations on a Jupyter Notebook
  - Interactive computing: Testbed for studying different formulations, preconditioning