

# From low to high level languages and back

Timo Betcke

[t.betcke@ucl.ac.uk](mailto:t.betcke@ucl.ac.uk)

University College London

Joint with S. Kailasa (UCL)



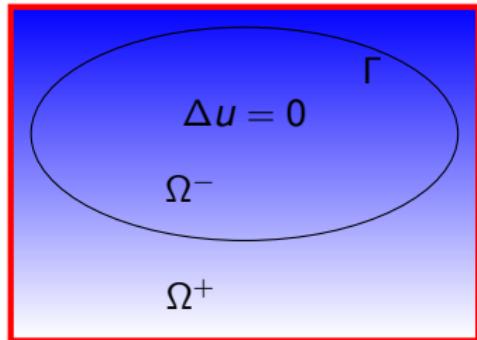
- Studied Computational engineering at Hamburg University of Technology (1998 - 2002)
- DPhil at Oxford Numerical Analysis Group, (2002 - 2005, graduated 2006)
- Postdocs in Braunschweig, Manchester, Reading (2002 - 2009)
- EPSRC Career Acceleration Fellow, Reading (2009 - 2011)
- Since 2011 at UCL Department of Mathematics
- Since 2018 Professor of Computational Mathematics
- Since 2022 ARC Associate Director

# What you don't find in my CV



- First PC: 25MHz 386DX with 2MB Ram, just enough to play Civilisation
- I owned the original Nintendo Entertainment System and played Super Mario on a black and white TV
- Favourite Youtube Channels: PBS Spacetime and Digital Foundry
- Long term Vim user, now overcome by the dark side and moved to VS Code
- I beat Elden Ring (my son still tells me I am bad at games since I never won Fall Guys)
- Best song of all times: Mad World by Tears for Fears (or as my son says, old and cringy)

# Some Maths to set the stage



$$\Delta u(\mathbf{x}) := \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$$

Application areas: Heat diffusion, electrostatics, etc.

As boundary condition on  $\Gamma$  we use  $u(\mathbf{x}) = f(\mathbf{x})$ .

Green's fct.: The function  $g(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi|\mathbf{x}-\mathbf{y}|}$  satisfies  $\Delta_{\mathbf{x}} g(\mathbf{x}, \mathbf{y}) = 0$  for every  $\mathbf{x} \neq \mathbf{y}$ . This function is special. It is called the Green's fct. (or fundamental solution) of the PDE.

# Representing the solution

We want to find a function  $\phi$  defined on  $\Gamma$  such that

$$u(\mathbf{x}) = \int_{\Gamma} g(\mathbf{x}, \mathbf{y})\phi(\mathbf{y}), \quad \mathbf{x} \in \Omega$$

We plugin the known boundary data and arrive at the **boundary integral equation**

$$f(\mathbf{x}) = \int_{\Gamma} g(\mathbf{x}, \mathbf{y})\phi(\mathbf{y}), \quad \mathbf{x} \in \Gamma$$

We call  $V$  the single-layer boundary integral operator defined by

$$[V\phi](\mathbf{x}) = \int_{\Gamma} g(\mathbf{x}, \mathbf{y})\phi(\mathbf{y}), \quad \mathbf{x} \in \Gamma$$

# How to discretise the integral equation?

We want to solve  $V\phi = f$ . Introduce a triangulation  $\mathcal{T}$  of the domain  $\Omega$  into triangles  $\tau_j$ . Define basis fct.

$$\phi_j(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \tau_j \\ 0, & \text{otherwise} \end{cases}$$

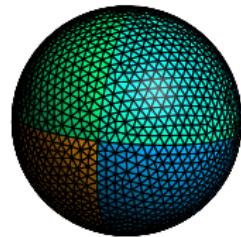
We approximate  $\phi = \sum_{j=1}^N c_j \phi_j$ . Multiplying with  $\phi_i$  and integrating gives

$$\int_{\Gamma} \phi_i(\mathbf{x}) [V\phi](\mathbf{x}) d\mathbf{s}(\mathbf{x}) = \int_{\Gamma} f(\mathbf{x}) \phi_i(\mathbf{x}) d\mathbf{s}(\mathbf{x}), i = 1, \dots, N$$

Solve  $\mathbf{V}\mathbf{c} = \mathbf{b}$

$$\mathbf{V}_{ij} = \int_{\tau_i} \int_{\tau_j} g(x, y) d\mathbf{s}(y) d\mathbf{s}(x)$$

$$\mathbf{b}_i = \int_{\Gamma} f(\mathbf{x}) \phi_i(\mathbf{x}) d\mathbf{s}(\mathbf{x})$$



## Two difficulties

- The Green's function  $g$  is singular for  $\mathbf{x} = \mathbf{y}$ . Each integral of a triangle with itself or and adjacent triangle requires special singularity treatment.
- The matrix  $\mathbf{V}$  is dense with  $N^2$  elements if there are  $N$  triangles. Huge assembly cost, and for large problems need linear scaling in complexity. (Can solve this by using Fast Multipole Methods for large problems, but complex and won't go into detail here)

# The Bempp boundary element software

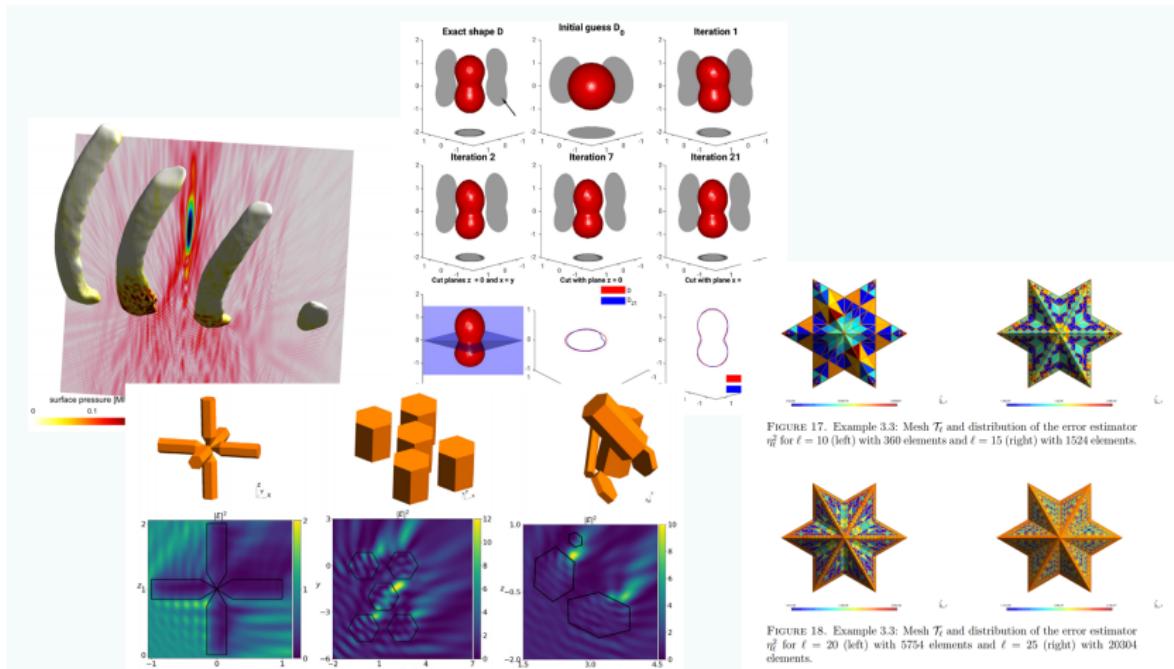


FIGURE 17. Example 3.3: Mesh  $T_\ell$  and distribution of the error estimator  $\eta_\ell^2$  for  $\ell = 10$  (left) with 360 elements and  $\ell = 15$  (right) with 1524 elements.



FIGURE 18. Example 3.3: Mesh  $T_\ell$  and distribution of the error estimator  $\eta_\ell^2$  for  $\ell = 20$  (left) with 5754 elements and  $\ell = 25$  (right) with 20304 elements.

- **Grid objects.** Managing grids and geometric information.
- **Grid functions.** Discrete functions defined on grids.
- **Boundary Operators.** Definition of the various boundary operators
- **Discrete boundary operators.** Boundary operators after discretisation

# The early C++ years of Bempp

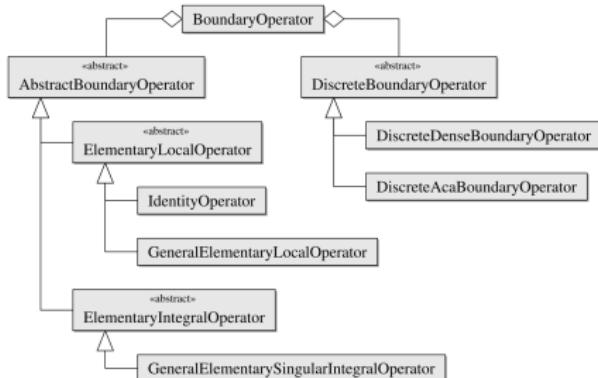


Fig. 3. Relationships between the main classes representing boundary operators.

- Complex object hierarchy
- Highly templated code
- Dependencies on complex packages (Boost, BLAS, Armadillo, Trilinos, DUNE, Intel TBB)

## Python Interface

- Auto-generated with Swig
- Thin layer on C++ code
- Almost no logic in the Python layer, all handled by C++

# The mixed C++/Python years



# Poisson-Boltzmann for virus simulations



Solve Poisson-Boltzmann equation

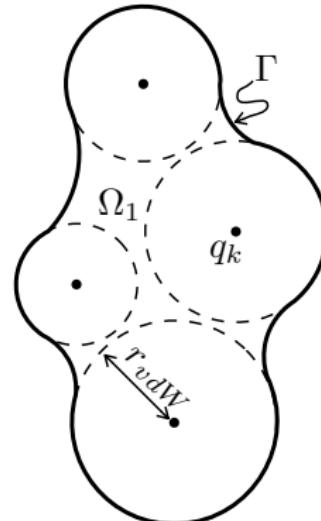
$$\Delta\phi_1 = \frac{1}{\epsilon_1} \sum_k q_k \delta(\mathbf{r}, \mathbf{r}_k) \text{ in } \Omega_1$$

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

Interface conditions on  $\Gamma$ :

$$\begin{aligned}\phi_1 &= \phi_2, \\ \epsilon_1 \frac{\partial \phi_1}{\partial \mathbf{n}} &= \epsilon_2 \frac{\partial \phi_2}{\partial \mathbf{n}}\end{aligned}$$

Let  $\phi_1 = \phi_{reac} + \phi_{coul}$  ( $\phi_{coul}$  is potential generated from the point charges only). Then want to compute the solvation energy.



$$\Delta G_{solv}^{polar} = \frac{1}{2} \sum_{k=1}^{N_q} q_k \phi_{reac}(\mathbf{r}_k)$$

Exterior field formulation<sup>1</sup>

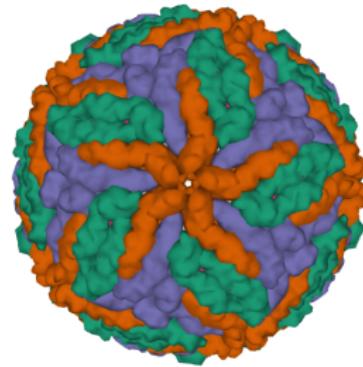
$$\begin{aligned}
 & \frac{\phi_{2,\Gamma}}{2} \left( \frac{\epsilon_1}{\epsilon_2} + 1 \right) - \left( K_Y^\Gamma - \frac{\epsilon_1}{\epsilon_2} K_L^\Gamma \right) (\phi_{2,\Gamma}) \\
 & + (V_Y^\Gamma - V_L^\Gamma) \left( \frac{\partial}{\partial \mathbf{n}} \phi_{2,\Gamma} \right) = \sum_{k=0}^{N_q} \frac{q_k}{4\pi\epsilon_2 |\mathbf{r}_\Gamma - \mathbf{r}_k|} \\
 & - \frac{\epsilon_1}{\epsilon_2} (W_Y^\Gamma - W_L^\Gamma) (\phi_{2,\Gamma}) + \frac{1}{2} \frac{\phi_{2,\Gamma}}{\partial \mathbf{n}} \left( 1 + \frac{\epsilon_1}{\epsilon_2} \right) \\
 & + \left( \frac{\epsilon_1}{\epsilon_2} K_Y'^\Gamma - K_L'^\Gamma \right) \left( \frac{\partial}{\partial \mathbf{n}} \phi_{2,\Gamma} \right) = \sum_{k=0}^{N_q} \frac{\partial}{\partial \mathbf{n}_k} \left( \frac{q_k}{4\pi\epsilon_2 |\mathbf{r}_\Gamma - \mathbf{r}_k|} \right)
 \end{aligned}$$

Also possible to derive interior field formulation (due to Juffur) or simple direct coupling from first line of Calderòn projector. Exterior formulation most favourable in terms of conditioning.

---

<sup>1</sup>Lu et. al., Proc. Natl. Acad. Sci. USA 103 (51) (2006) 19314-19319

- Around 1.6m atoms
- Generated mesh has around 10m surface elements



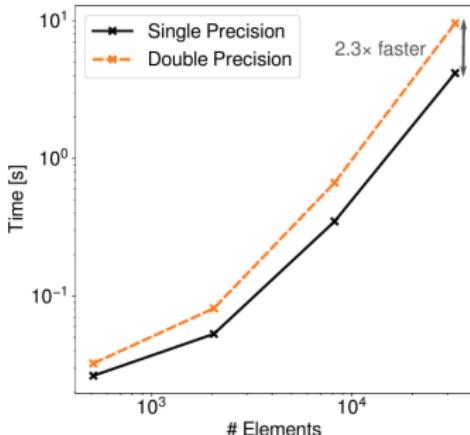
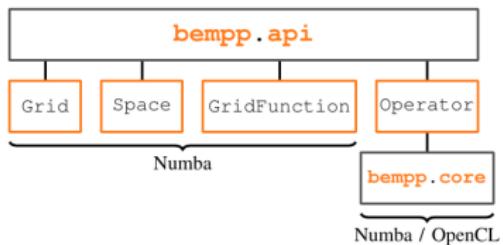
Use Galerkin BEM code Bempp-cl accelerated with fast particle interactions through Exafmm

Outline of rest of talk:

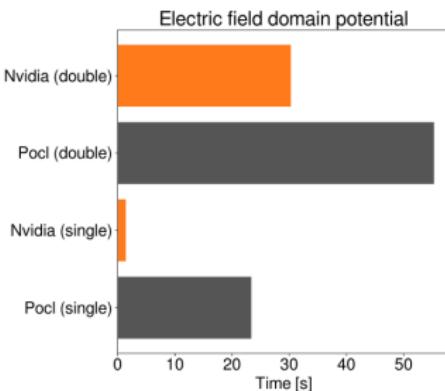
- A short introduction to Bempp-cl and Exafmm
- Black-box coupling strategies for Galerkin BEM
- Practical issues and ongoing work

- Successor of the Bempp boundary element library.
- Fully developed using Python with fast OpenCL kernels for computational routines.
- Galerkin discretisation of operators for Laplace, Helmholtz, and Maxwell problems.
- Compute kernels can be run on CPU or offloaded to GPU.
- Core routines optimised for fast dense assembly of operators.
- Library provides coupling interfaces to Exafmm, can be easily adapted to other fast particle summation libraries.

# Characteristics of Bempp-cl



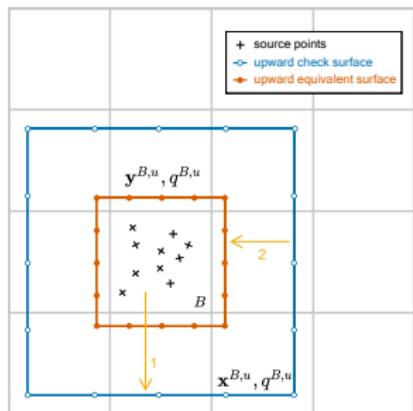
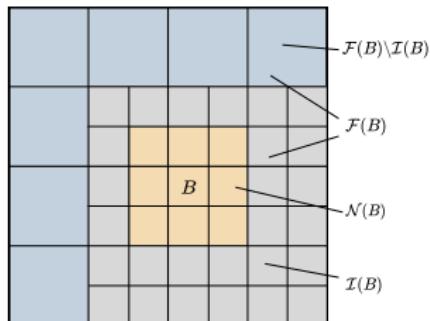
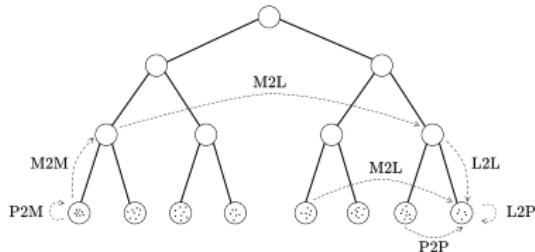
- Top: Structure of Bempp-cl
- Top-Right: AVX acceleration in Bempp-cl
- Bottom-Right: Offloading of a domain potential operator



# Exafmm-t: High-Performance KIFMM



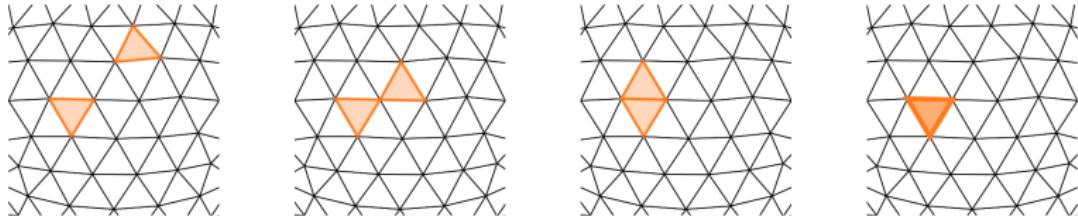
- Kernel-Independent FMM library.
- Written in C++, Provides Python Interface.
- Highly efficient at higher accuracies.



Let  $A$  be the matrix representation of a Galerkin discretised boundary integral operator. Let  $x$  be any vector. Reformulate  $Ax$  as

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

- $P_{test}$  and  $P_{dom}$  highly sparse matrices that map function space coefficients to particle weights at quadrature points.
- $G$  is black-box operator evaluating particle sums over weights across all quadrature points across all elements.
- $C$  is sparse matrix that contains the particle interactions between neighboring triangles.
- $S$  is sparse matrix containing all singular Galerkin integrals in adjacent elements.



Problem: Any FMM code also evaluates the near-field. Near-field can contain adjacent triangles (singular quadrature rules), and non-adjacent triangles (standard triangle Gauss rule).

The sources and target points in the FMM are arising from the non-singular quadrature rule.

Solution: After FMM evaluation subtract out the regular quadrature rule contribution from adjacent triangles and add in the correct singular quadrature rule contribution.

# Number of FMM passes per Operator



Operator	$V$	$K$	$K'$	$W_Y, W_L$
FMM Passes	1	3	1	6, 3

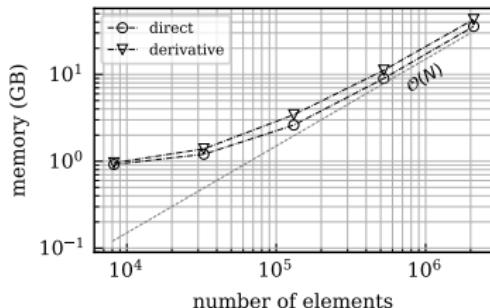
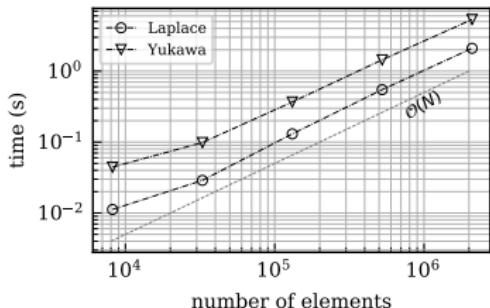
Example: Double-Layer Potential Operator

$$\begin{aligned}[K\phi](\mathbf{x}) &= \int_{\Gamma} \partial_{\mathbf{n}(\mathbf{y})} g(\mathbf{x}, \mathbf{y}) \phi(\mathbf{y}) ds(\mathbf{y}) \\ &= - \sum_{j=1}^3 [\nabla_{\mathbf{x}} g(\mathbf{x}, \mathbf{y})]_j \mathbf{n}_j(\mathbf{y}) \phi(\mathbf{y}) ds(\mathbf{y})\end{aligned}$$

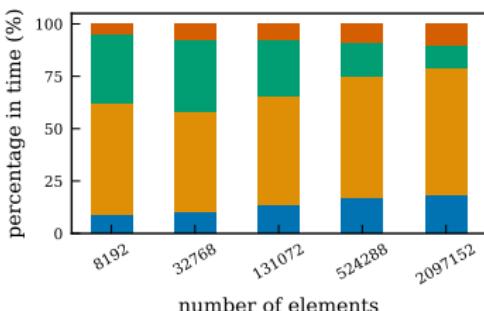
Require three FMM passes for the three different densities  
 $\tilde{\phi}_j(\mathbf{y}) := \mathbf{n}_j(\mathbf{y}) \phi(\mathbf{y}), j = 1, \dots, 3.$

The transmission operator for the exterior Poisson-Boltzmann formulation requires **19 FMM passes**.

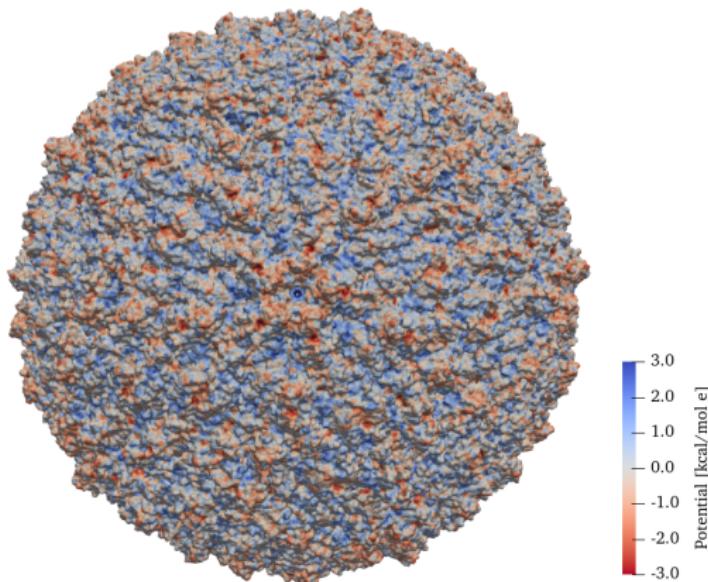
# Poisson-Boltzmann for a sphere



- FMM Expansion Order 5
- 6 regular quadrature points per element
- **Right Blue:** Laplace, **Yellow:** Yukawa, **Green:** Singular Correction, **Red:** Other



# Results for Zika Virus



40 Core Compute node. Total time: 139.5 minutes, GMRES time: 80 minutes, 18 GMRES iterations, Max RAM use: 43GB,  
 $\Delta G_{solv} = -116254.9 \text{ kcal/mol}$ . Diagonal preconditioner through inverse of mass matrix with mass lumping.

# A FEM/BEM formulation for inhomogeneities



Want to model the interior permittivity as variable. → Use FEM for interior/BEM for exterior domain.

(joint work with E. Burman, M. Bosy, C. Cooper)

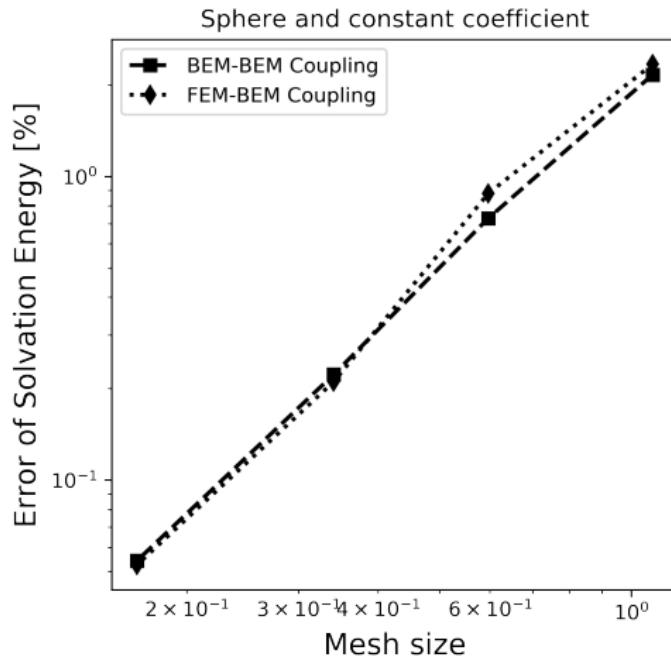
Find  $\phi_1 \in H^1(\Omega_1)$  and  $\lambda_2 \in H^{-\frac{1}{2}}(\Gamma)$  such that for all  $v \in H^1(\Omega_1)$  and  $\zeta \in H^{-\frac{1}{2}}(\Gamma)$

$$\begin{cases} (\epsilon_1 \nabla \phi_1, \nabla v)_{\Omega_1} - \langle \epsilon_1 \lambda_2, v \rangle_{\Gamma} &= \left( \sum_{k=1}^{N_q} q_k \delta(\mathbf{x}, \mathbf{x}_k), v \right)_{\Omega_1} \\ \langle \left( \frac{1}{2} I - K_Y^\Gamma \right) \phi_1, \zeta \rangle_{\Gamma} + \frac{\epsilon_1}{\epsilon_2} \langle V_Y^\Gamma \lambda_2, \zeta \rangle_{\Gamma} &= 0. \end{cases}$$

Matrix form:

$$\begin{bmatrix} \epsilon_1 A & -\epsilon_1 M^T \\ \left( \frac{1}{2} I + K \right) & \frac{\epsilon_1}{\epsilon_2} V \end{bmatrix} \begin{bmatrix} \vec{\phi}_1 \\ \vec{\lambda}_2 \end{bmatrix} = \begin{bmatrix} \vec{f} \\ 0 \end{bmatrix}.$$

Use FEniCS for interior problem, Bempp-CL for exterior problem.



Error for the Kirkwood sphere.

# Comparison with APBS - varying permittivity



	Mesh size Å	$\Delta G_{solv}$ kcal/mol
APBS	$0.52 \times 0.52 \times 0.52$	-32.4652
	$0.39 \times 0.39 \times 0.39$	-32.4042
	$0.26 \times 0.26 \times 0.26$	-32.3375
	$0.17 \times 0.17 \times 0.17$	-32.3413
	Mesh dens. vert/Å <sup>2</sup>	
FEM-BEM coupling	2	-35.688
	4	-33.304
	8	-32.634
	16	-31.868

Solvation energy of arginine with a Gaussian-like permittivity, computed using the FEM-BEM approach and APBS. The mesh density for FEM-BEM corresponds to the vertex density of the surface mesh used to generate the volumetric mesh

# A Rust based scalable fast solver framework



Goal: Create a fast solver framework for fast parallel forward evaluation and approximate inversion of integral operators  
(<https://github.com/rusty-fast-solvers/roadmap>)

All component libraries developed in Rust with Rayon (multithreading) and MPI (across nodes) for parallelization.

Development Status:

- **rusty-green-kernel** AVX accelerated direct evaluation of Laplace, Helmholtz Green's functions. **released**.
- **rusty-tree** Serial and parallel Octree implementations. Serial tree implemented. Parallel tree early stages. **close to release**.
- **rusty-compression** Fast randomized compression routines for approximate low-rank matrices. **released**.
- **rusty-translation** Implementation of M2M, M2L, L2L, P2M, L2P operators based on numerical field compressions (KIFMM, etc.) **early development**
- **rusty-fmm** Serial and parallel FMM loops **In planning**
- **rusty-inverse** Evaluation of approximate inverses **Not yet started**

- Generic, practically usable and efficient black-box coupling of Galerkin BEM and Exafmm.
- All computational results shown today steered through interactive Jupyter notebooks.
- Python glue language between the codes. High productivity through Jupyter interface.
- For links to paper, reproducible data and codes see  
[https://github.com/barbagroup/bempp\\_exafmm\\_paper](https://github.com/barbagroup/bempp_exafmm_paper)
- Ongoing development of exascale ready software for EPSRC's new ARCHER2 system (AMD EPYC CPU Cluster)
- Goal: Large-scale coupled Finite element/Boundary element models

- T. Wang, C. Cooper, T. Betcke, L. Barba, *High-productivity, high-performance workflow for virus-scale electrostatic simulations with Bempp-Exafmm*, arXiv preprint arXiv:2103.01048 (2021),  
[https://github.com/barbagroup/bempp\\_exafmm\\_paper](https://github.com/barbagroup/bempp_exafmm_paper)
- T. Betcke, M. W. Scroggs, *Bempp-cl: A fast Python based just-in-time compiling boundary element library*, Journal of Open Source Software, 6(59), 2879 (2021), <https://doi.org/10.21105/joss.02879>
- T. Betcke, M. W. Scroggs, *Designing a high-performance boundary element library with OpenCL and Numba*, Computing in Science & Engineering 23, pp. 18–28 (2021)  
<https://doi.org/10.1109/MCSE.2021.3085420>
- T. Wang, R. Yokota, L. Barba, *ExaFMM: a high-performance fast multipole method library with C++ and Python interfaces*. Journal of Open Source Software, 6(61), 3145 (2021),  
<https://doi.org/10.21105/joss.03145>