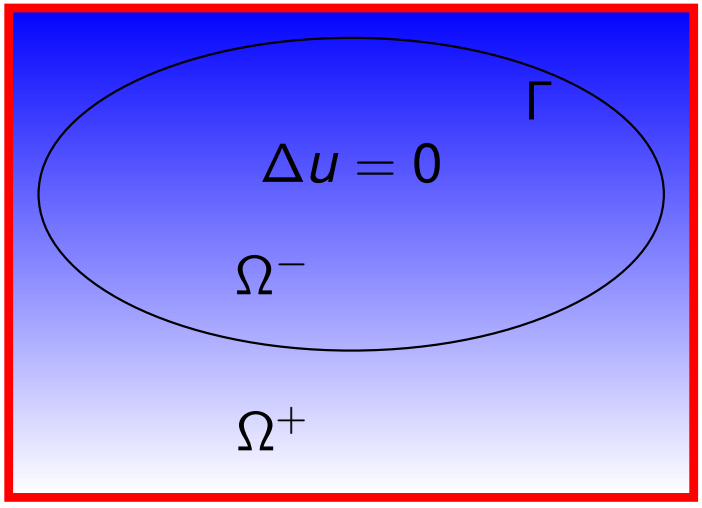
Towards integral equations at exascale in Rust

Timo Betcke
t.betcke@ucl.ac.uk
University College London

Joint with S. Kailasa, M. Scroggs and many other collaborators

Supported by EPSRC Excalibur grants EP/W026260/1 and EP/W007460/1

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.

Boundary condition:
$u(\mathbf{x}) = f(\mathbf{x})$.

Represent solution as

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

Need to solve $V\phi = f$ with

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

How to discretise the integral equation?

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

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

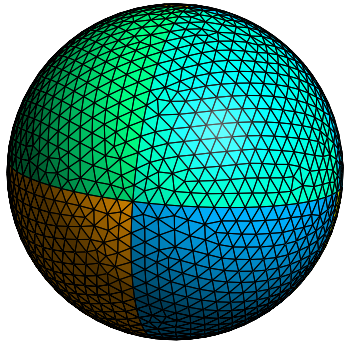
We approximate $\phi = \sum_{j=1}^{N} c_j \phi_j$. Multiplying with ϕ_i and integrating gives

$$\int_\Gamma \phi_i(\mathbf{x}) \left[V\phi\right](\mathbf{x}) ds(\mathbf{x}) = \int_\Gamma f(\mathbf{x})\phi_i(\mathbf{x}) ds(\mathbf{x}), i = 1, \ldots, N$$

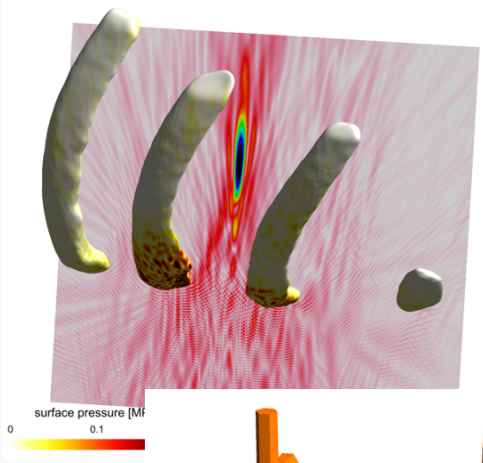
Solve $\mathbf{Vc} = \mathbf{b}$

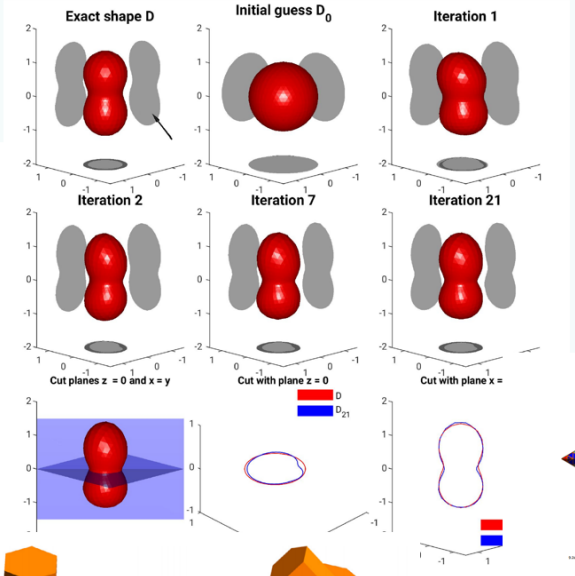
$$\mathbf{V}_{ij} = \int_{\tau_i} \int_{\tau_j} g(\mathbf{x}, \mathbf{y}) ds(\mathbf{y}) ds(\mathbf{x})$$

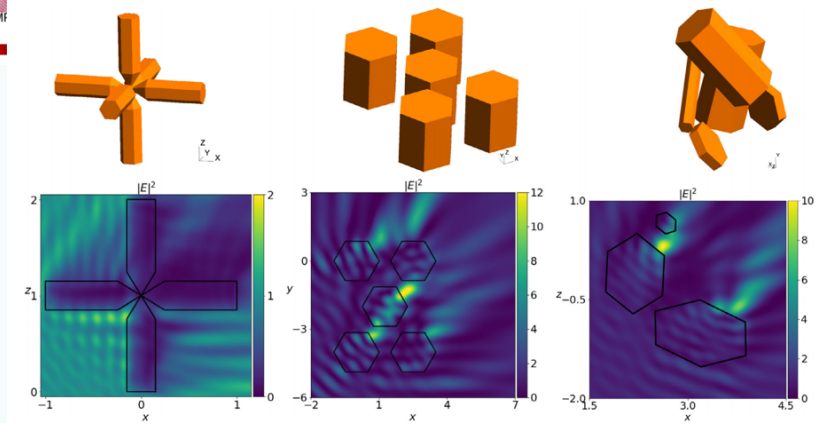
$$\mathbf{b}_i = \int_\Gamma f(\mathbf{x})\phi_i(\mathbf{x}) ds(\mathbf{x})$$



The Bempp boundary element software







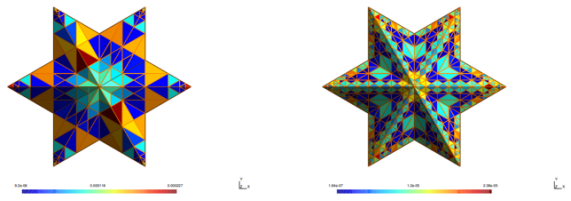


FIGURE 17. Example 3.3: Mesh $\mathcal{T}_\ell$ and distribution of the error estimator η_ℓ^2 for $\ell = 10$ (left) with 360 elements and $\ell = 15$ (right) with 1524 elements.

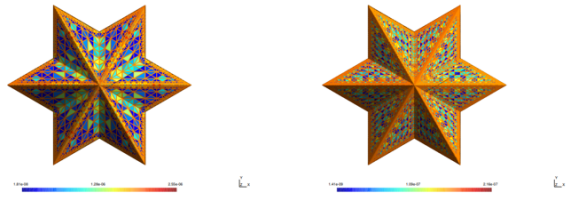


FIGURE 18. Example 3.3: Mesh $\mathcal{T}_\ell$ and distribution of the error estimator η_ℓ^2 for $\ell = 20$ (left) with 5754 elements and $\ell = 25$ (right) with 20304 elements.

The early C++ years of Bempp[1]

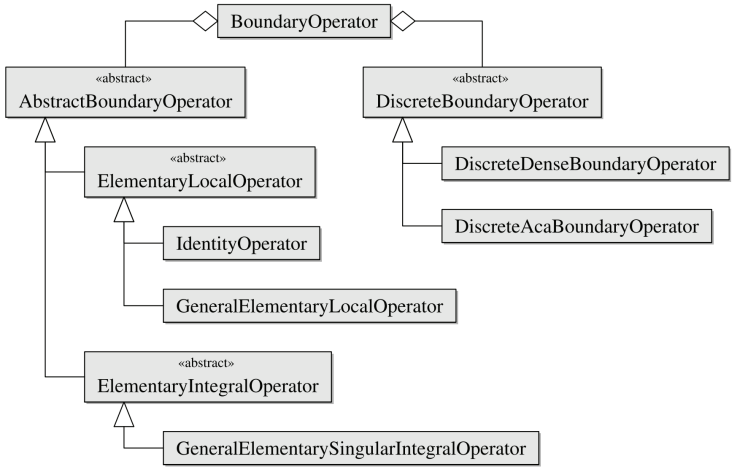


Fig. 3. Relationships between the main classes representing boundary operators.

- 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++

Over time moved all high-level functions into Python, only kept low-level performance critical data structures in C++.

[1] W. Śmigaj et. al., *Solving boundary integral problems with BEM++*, ACM Transactions on Mathematical Software 41 (2015), pp. 1 - 40

Fast JIT compiled Galerkin BEM in Python

UCL

Numba for grid iterations

- Grid topology computations
- Python callables
- Routines on grid functions, (L^2-norm, etc.)
- Fallback for operators

OpenCL for matrix assembly/evaluation

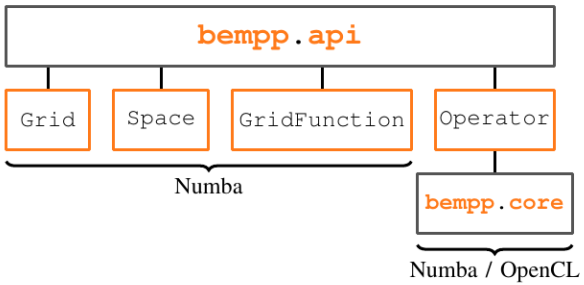
- All potential evaluations
- Mass matrices
- FMM Near-Field

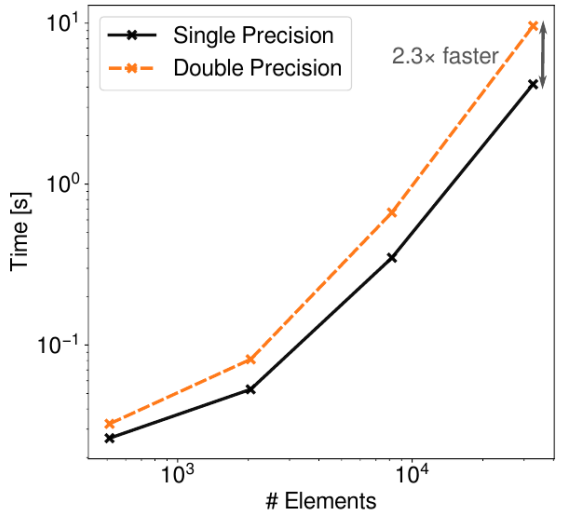
Other technologies

- Scipy for spare matrix operations and iterative solvers
- Numpy dense solvers

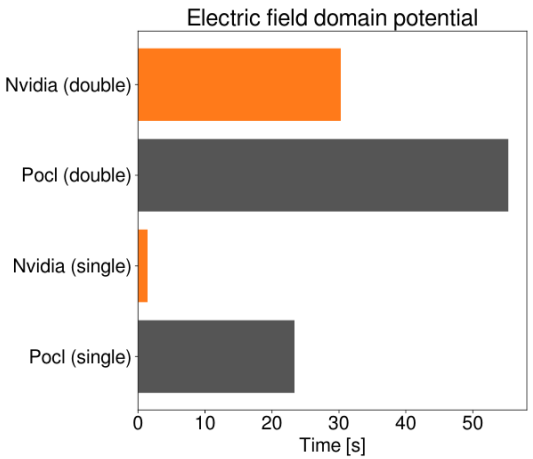
- All logic controlled from Python (easy to hack on).
- Very fast SIMD optimized dense matrix assembly.
- But need lots of OpenCL C99 kernel code.

Characteristics of Bempp-cl[2]





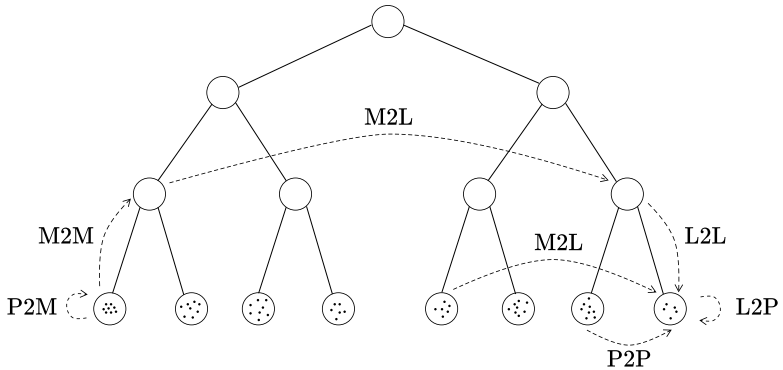
- Top: Structure of Bempp-cl
- Top-Right: AVX acceleration in Bempp-cl
- Bottom-Right: Offloading of a domain potential operator

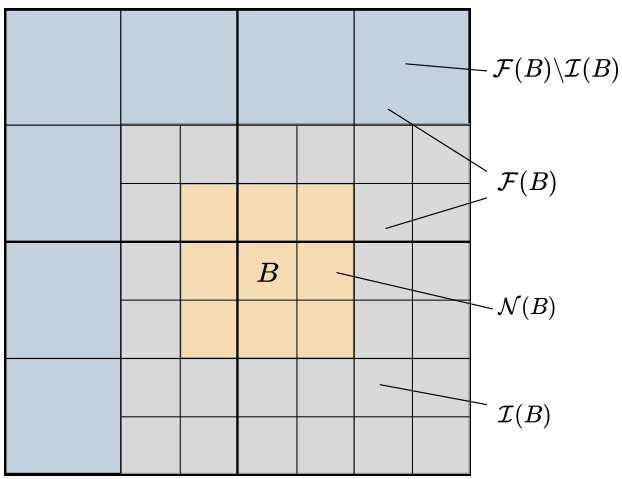


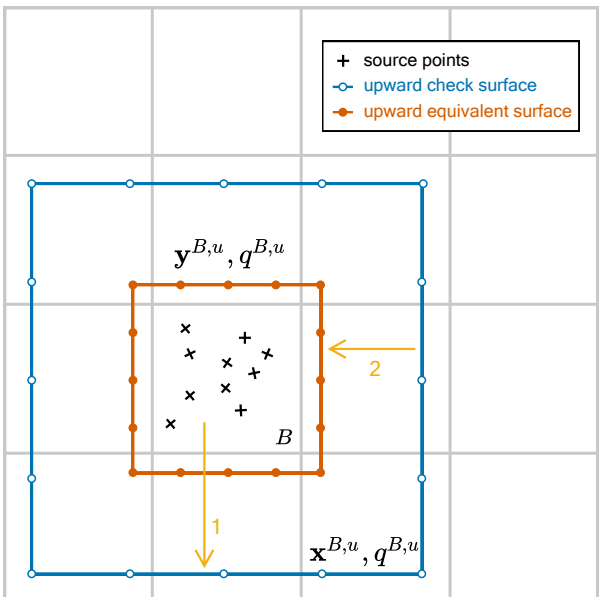
[2] T. Betcke, M. Scroggs, *Designing a high-performance boundary element library with OpenCL and Numba*, Computing in Science and Engineering, 23 (2021), pp. 18 - 28

Exafmm-t: High-Performance KIFMM[3]

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







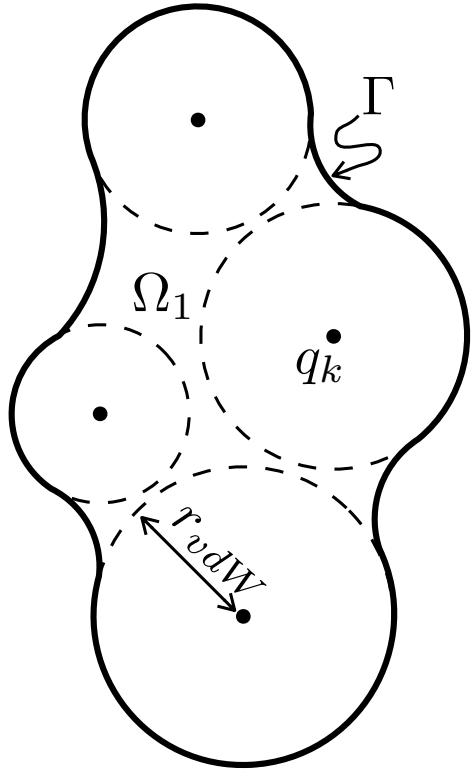
[3] T. Wang, R. Yokota, L. Barba, *ExaFMM: a high-performance fast multipole method library with C++ and Python interfaces*, Journal of Open Source Software 6(61), 3145

Why large-scale BEM?

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 Γ:

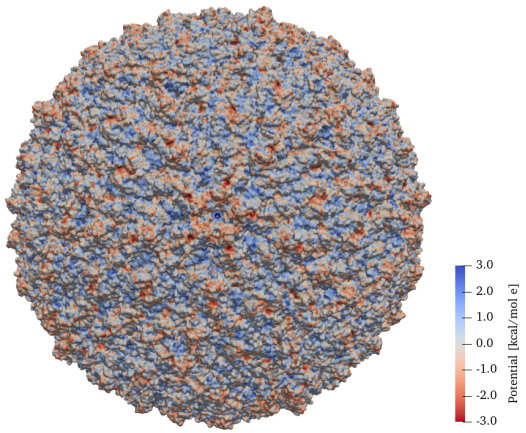
$$\phi_1 = \phi_2,$$

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

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

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

Exascale FEM/BEM modeling



- Electrostatic simulation of Zika virus (10m elements) on single compute node[4]
- Mass-matrix preconditioned single trace formulation.
- FMM acceleration via ExaFMM.

Goal: Full FEM/BEM modelling of complex protein surfaces.

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

A: FEM matrix, K: Double-Layer operator, V: Single-Layer operator, M: Mass matrix

Want to scale up to billions surface dofs.

[4] T. Wang, C. D. Cooper, T. Betcke, L. A. Barba, *High-productivity, high-performance workflow for virus-scale electrostatic simulations with Bempp-Exafmm*, submitted (2021)

Deconstructing a BEM operator call

Discrete integral operator evaluation takes the form

$$Vx = P_{dual}^T (G - C) P_{dom} \overline{x} + S\overline{x}$$

- G dense.
- All other matrices highly sparse.

- $P_{dom/dual}$: Map from domain/dual space cefficients to values at quadrature points
- G matrix of Green's fct. evaluations at regular Gauss points
- C: Correction matrix containing contribution of G associated with adjacent triangles
- S: Contributions of singular quadrature rule on adjacent triangles

- Parallel fast methods (H-matrices/FMM) for G.
- Distributed sparse linear algebra for everything else.

Software Technologies

Python

- Good for higher level logic.
- Efficient libs for array processing.
- Performance for complex algorithms in Python (e.g. FMM) difficult[5].

Python + Code Gen

- Most flexible solution.
- Complex framework.
- What low-level language to use?

C++

- Complex Makefiles.
- Deployment difficult.
- C++20 support still patchy.

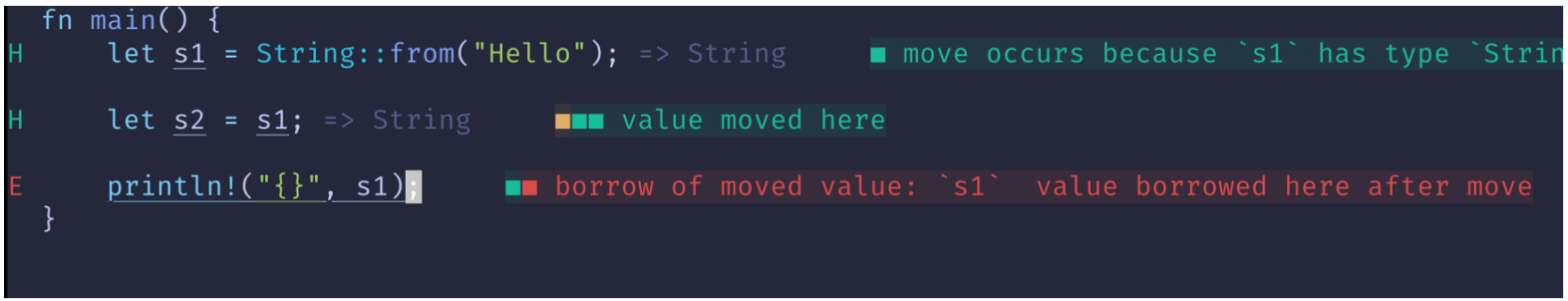
[5] S. Kailas et. al., *PyExaFMM: an exercise in designing high-performance software with Python and Numba*, to be submitted

Why Rust?

- Modern language design.
- Great tooling.
- Compile time safety guarantees.
- Simple deployment.
- Python integration trivial.

Design a set of component libraries that can be easily composed into large-scale FEM/BEM applications.

Rust ownership



- Every variable in Rust has a unique owner
- Assigning s1 to s2 transfers ownership of the string.

The Borrow Checker

```
fn main() {
    let x = 5.0; => f64

    let y = &x; => &f64
    let z = &x; => &f64

    println!("{} {}", y, z);
}
```

- Creating references is called borrowing.
- We are not allowed to mutate a variable after it was borrowed.
- We can have either one mutable borrow or many non-mutable borrows, not both at the same time.

```
  fn main() {
      let mut x = 5.0; => f64

H     let y = &x; => &f64      ■ borrow of `x` occurs here

E     x = 2.0;      ■■ cannot assign to `x` because it is borrowed  assignment to borrowed `x` occurs here

H     println!("This won't work {}", &y);     ■ borrow later used here
  }
```

What is already done?

Grids and finite elements

- Various order Lagrange elements on triangles
- Order 1 and order 2 basis function on quadrilaterals
- Degree 1 Raviart-Thomas functions on triangles
- Grids on single nodes and prototype multi-node grid management

Integration

- Standard Gauss type rules for regular integrals over triangles and quadrilaterals
- Have fully numerical singular integrators on triangles (based on Erichsen/Sauter)
- Singular integrators on quadrilaterals almost done

FMM

- Distributed balanced octrees are implemented
- Single note experimental kernel independent FMM is working
- Currently working on distributed FMM loop

Other work

- Developing core sparse linear algebra data structures
- Rust native dense matrix decompositions for randomized linear algebra
- SIMD Optimized kernel evaluation
- C API and Python interfaces

The Rust Scientific Computing ecosystem

UCL

Core utilities

- Num - Traits and utility routines for floating point types.
- Cauchy - A complex scalar type library (uses Num).
- approx - Unit test routines for floating point types.
- Serde - Serialization library.

Linear Algebra/Machine Learning

- ndarray - multidimensional array type in Rust.
- nalgebra - Rust linear algebra library with Lapack bindings.
- Linfa - Native machine learning in Rust.
- tensorflow - Rust bindings to tensorflow.
- petsc-rs - Rust bindings for Petsc.

Parallelization

- rsmpi - MPI Interface for Rust.
- Rayon - Fork/Join threading, parallel loops and iterators.
- async/await (Tokio, etc.) - frameworks for asynchroneous execution.

Language bindings

- PyO3 - Binding complex Rust types to Python.
- maturin - Simple tools to wrap Rust C API bindings with Python cffi.
- Any language that supports C bindings can talk to Rust easily.

Still lots of moving parts in the ecosystem but an enormous amount of development happening.

Some final messages

- Using Python + JIT (e.g. Numba) works well for simple array oriented algorithms. More problematic for complex algorithms.
- We are at the start of a multi-year journey to develop very large-scale integral equation solvers in Rust.
- Rust is ready to be looked at by Scientific Computing Community (though far away from the level of maturity of the C++ ecosystem)
- Rust makes little sense when legacy C++ code exists or there is need for heterogeneous compute with SyCL or Cuda.
- But consider and evaluate Rust if you start a completely new project.