

Electrostatic simulations at Exascale – The beginning of a rusty journey

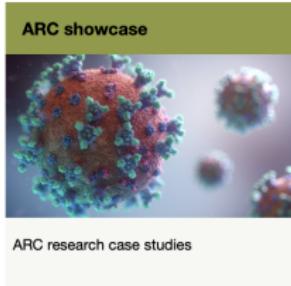
Timo Betcke

t.betcke@ucl.ac.uk

University College London

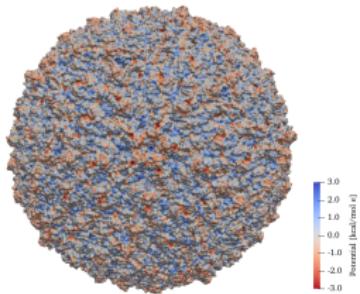


About me



- Professor of Computational Mathematics @UCL Maths
- Associate Director and Director of Research at UCL ARC
- UCL Lead for Excalibur projects on coupling at exascale, and software generation for novel architectures
- Big fan of anything Scientific Computing, Rust, and Python
- Not coming from an AI background but big interest in building modern integrated computational ecosystems

Exascale FEM/BEM modeling



- Electrostatic simulation of Zika virus (10m elements) on single compute node¹
- Mass-matrix preconditioned single trace formulation.
- FMM acceleration via ExaFMM.

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

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

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

Want to scale up to billions surface dofs.

¹T. Wang, C. D. Cooper, T. Betcke, L. A. Barba, *High-productivity, high-performance workflow for virus-scale electrostatic simulations with Bempp-Exafmm*, submitted (2021)

Python

- Good for higher level logic.
- Efficient libs for array processing.
- Performance for complex algorithms in Python (e.g. FMM) difficult².

Python + Code Gen

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

C++

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

²S. Kailas et. al., *PyExaFMM: an exercise in designing high-performance software with Python and Numba*, to be submitted

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



A screenshot of a code editor showing a simple Rust program. The code consists of four lines: a comment 'Run | Debug', a function definition 'fn main() {', a call to 'println!("Hello, world!");', and a closing brace '}'. The code is highlighted in yellow and blue, and the background of the editor is dark.

```
▶ Run | Debug
1 fn main() {
2     println!("Hello, world!");
3 }
4
```

Design a set of libraries in Rust that can be easily composed into large parallel FEM/BEM applications.

The Rust Scientific Computing ecosystem



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.
- householder - Early stages linear algebra library.
- 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 asynchronous execution.

Language bindings

- PyO3 - Binding complex Rust types to Python.
- maturin - Simple tools to wrap Rust C API bindings with Python ffi.
- 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.

What makes Rust great for HPC?



- Fully built on top of LLVM.
- Strong and rich type system with traits based generics resolved at compile time.
- All type conversions must be explicit (no accidental assignment of a 64 bit integer to a 32 bit variable).
- Very simple foreign function interface to integrate existing C code without overhead.
- Simple but powerful error handling.

...However, CUDA support still very early stages (but OpenCL works well)

Example: A new approach to large scale linear algebra



- Joint work with Jed Brown (Colorado Boulder)
- Design traits (and prototype implementations) that provide a unified approach to linear algebra in finite and infinite dimensional spaces
- Currently implementing minimum viable product
- Want to have a modern Rust native sparse linear algebra kernel

Novel unified linear algebra traits



```
// A base operator trait.  
impl Trait  
pub trait OperatorBase: Debug {  
    type Domain: LinearSpace;  
    type Range: LinearSpace;  
  
    /// Returns a reference to trait object that supports application of the operator.  
    /// By default it returns an 'Err'. But for concrete types  
    /// that support matvecs it is specialised to return  
    /// a dynamic reference.  
    fn as_apply(&self) -> Option<dyn AsApply<Domain = Self::Domain, Range = Self::Range>>;  
    None  
}  
  
fn has_apply(&self) -> bool {  
    self.as_apply().is_some()  
}
```

```
/// Definition of a linear space  
///  
/// Linear spaces are basic objects that can create  
/// elements of the space.  
6 implementations  
pub trait LinearSpace {  
    /// Field Type.  
    type F: Scalar;  
  
    /// Type associated with elements of the space.  
    type E<'b>: Element<Space = Self>  
    where  
        Self: 'b;  
  
    /// Create a new vector from the space.  
    fn create_element<'b>(&'b self) -> Self::E<'b> {  
        std::unimplemented!();  
    }  
}
```

- Use subtraits to model normed spaces, inner product spaces, etc.
- Can build iterative solvers on top of traits without knowledge of underlying implementation

tree	Fast distributed octree management
grid	Distributed parallel grids
element	Tabulation and evaluation of finite elements
green-kernel	Direct evaluation of Green's functions
quadrature	Collection of quadrature rules
fmm	Distributed fast multipole methods

- 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 for scientific computing and allows us to rethink how to design modern computational libraries.

Also talk to me about...

- Planned Centre for Doctoral Training (joint UCL/Imperial) on the interface of research software engineering, computational sciences, and data sciences.
- Planned UKRI Hub for Numerical Analysis Foundations of Artificial Intelligence.

