Writing a grid library for finite and boundary element methods

Matthew Scroggs
University College London

mscroggs.co.uk
matthew.scroggs.14@ucl.ac.uk
mscroggs
@mscroggs@mathstodon.xyz

Timo Betcke
University College London

Ignacia Fierro Piccardo
University College London

Srinath Kailasa
University College London

Bempp / BEM++

2012 BEM++ v1.0 (C++ library)

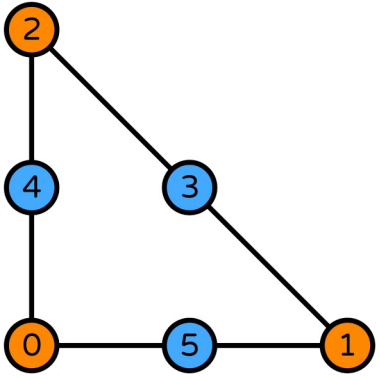
2016 BEM++ v3.0 (C++ core, Python interface)

2019 Bempp-cl (Python library, OpenCL kernels)

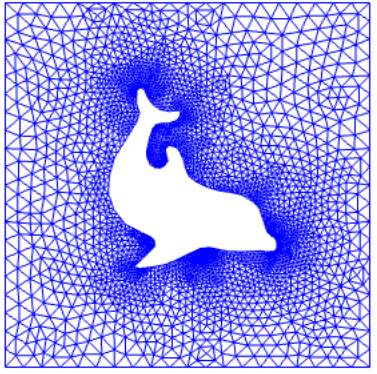
2020 Work started on Rust FMM code

2024 Bempp-rs v0.1 (Rust library)

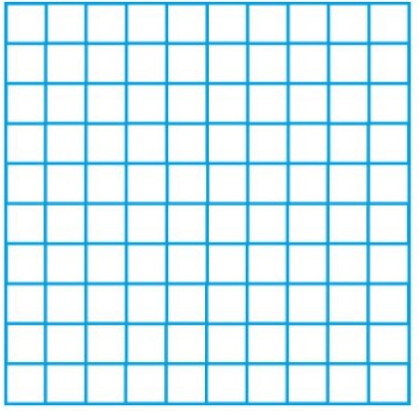
Bempp-rs to do list



Finite elements



Grids/meshes



Assembly

See *RLST: linear algebra frameworks in Rust for the solution of partial differential equations* (Timo Betcke)

Linear algebra

Element trait

```
pub trait FiniteElement {
    type T: RlstScalar;

    fn cell_type(&self) -> CellType;
    fn embedded_superdegree(&self) -> usize;
    fn dim(&self) -> usize;
    fn value_shape(&self) -> &[usize];
    fn value_size(&self) -> usize;
    fn tabulate(&self, points: &Array2<T>, nderivs: usize, data: &mut Array4<T>);
    fn entity_dofs(&self, entity_dim: usize, entity_number: usize) -> Option<&[usize]>;
    fn map_type(&self) -> MapType;
    fn tabulate_array_shape(&self, nderivs: usize, npoints: usize) -> [usize; 4];
}
```

Element implementation

```
pub struct CiarletElement<T: RlstScalar> {
    cell_type: ReferenceCellType,
    degree: usize,
    embedded_superdegree: usize,
    map_type: MapType,
    value_shape: Vec<usize>,
    value_size: usize,
    continuity: Continuity,
    dim: usize,
    coefficients: Array3<T>,
    entity_dofs: [Vec<Vec<usize>>; 4],
    interpolation_points: EntityPoints<T::Real>,
    interpolation_weights: EntityWeights<T>,
}
```

Grid traits

```
pub trait Grid {
    const GDIM: usize;
    const TDIM: usize;
    type T: RealScalar;
    type Point: ...;
    type Entity: ...;
    type Topology: ...;
    type Geometry: ...;
    type EntityDescriptor: ...;
    type EntityIter: Iterator<Item = Self::Entity>;
    type PointIter: Iterator<Item = Self::Point>;

    fn geometry_dim() -> usize {
        Self::GDIM
    }
    fn topology_dim() -> usize {
        Self::TDIM
    }
    fn entity(&self, dim: usize, local_index: usize) -> Self::Entity;
    fn entity_iter(&self, dim: usize) -> Self::EntityIter;
    fn entity_from_id(&self, dim: usize, id: usize) -> Option<Self::Entity>;
}
```

Two options

`-> Box<dyn Entity>`

- Types not known at compile time
- Vertices and edges (eg) have different types

`-> Self::Entity`

- Types known at compile time
- All entities must be same type

Grid traits

```
pub trait Entity {
    type Topology: ...;
    type Geometry: ...;

    fn entity_type(&self) -> ReferenceCellType;
    fn local_index(&self) -> usize;
    fn global_index(&self) -> usize;
    fn geometry(&self) -> Self::Geometry;
    fn topology(&self) -> Self::Topology;
    fn ownership(&self) -> Ownership;
}
```

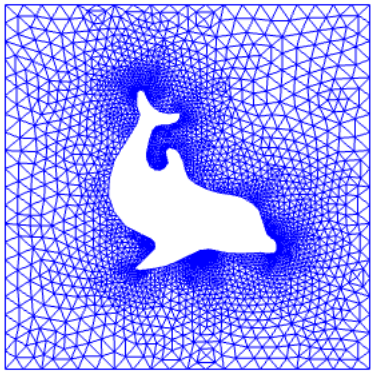
Grid traits

```
pub trait Topology {
    type EntityIndexIter: ...;
    type ConnectedEntityIndexIter: ...;

    fn connected_entity_iter(&self, dim: usize) -> Self::ConnectedEntityIndexIter;
    fn sub_entity_iter(&self, dim: usize) -> Self::EntityIndexIter;
    fn sub_entity(&self, dim: usize, index: usize) -> usize;
}

pub trait Geometry {
    type Point: ...;
    type PointIter: ...;

    fn points(&self) -> Self::PointIter;
    fn point_count(&self) -> usize;
    fn volume(&self) -> usize;
}
```

Grid traits

```
pub trait GmshIO {
    ...
}
```

Grid implementation

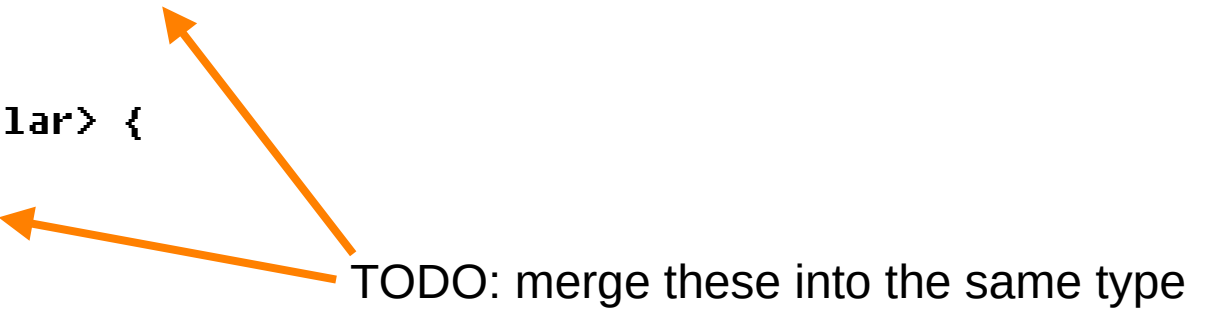


Grid topology
- Connectivity between grid entities
- Which vertices does each cell have?
- Which cells are neighbours?

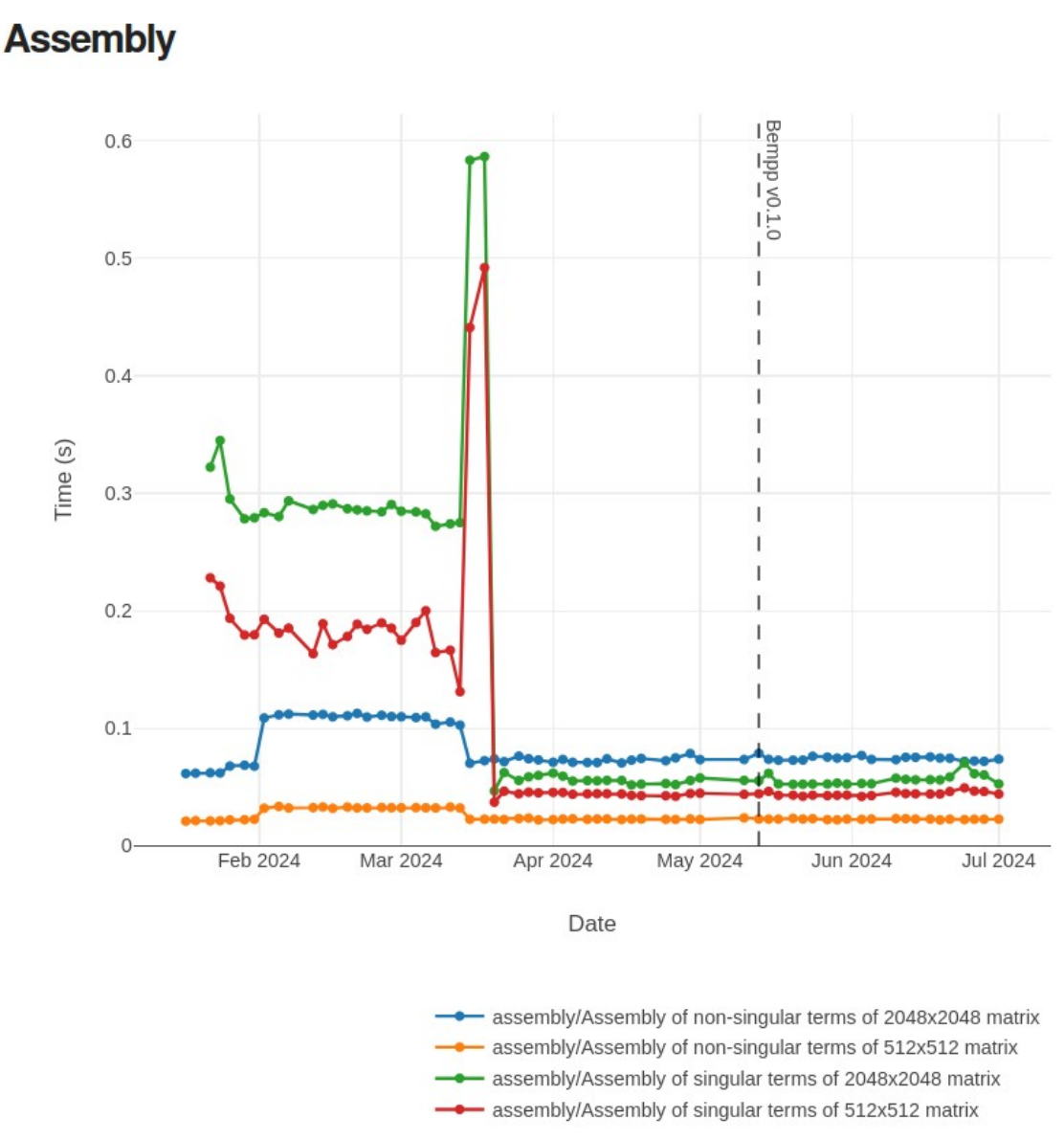
Grid geometry
- Positions of points in 3D space.
- Which points are associated with each cell?

Grid implementation

```
pub struct SingleElementGrid<T: RlstScalar> {
    topology: SingleElementTopology,
    geometry: SingleElementGeometry<T>,
}

pub struct MixedGrid<T: RlstScalar> {
    topology: MixedTopology,
    geometry: MixedGeometry<T>,
}
```



bempp.com/benchmarks



Conclusions

- Implemented single element and mixed grids in Rust
- Grid traits that *should* be general enough to be implemented on top of any other grid implementation

Future work

- Make assembly work in parallel with FMM

  see *Towards a distributed FMM in Rust* (Srinath Kailasa, Scientific Computing in Rust 2023)

Thanks for listening!

Matthew Scroggs
University College London

mscroggs.co.uk
matthew.scroggs.14@ucl.ac.uk
mscroggs
@mscroggs@mathstodon.xyz

Timo Betcke
University College London

Ignacia Fierro Piccardo
University College London

Srinath Kailasa
University College London