BOUT++

PDE Solver Framework for Plasma Models

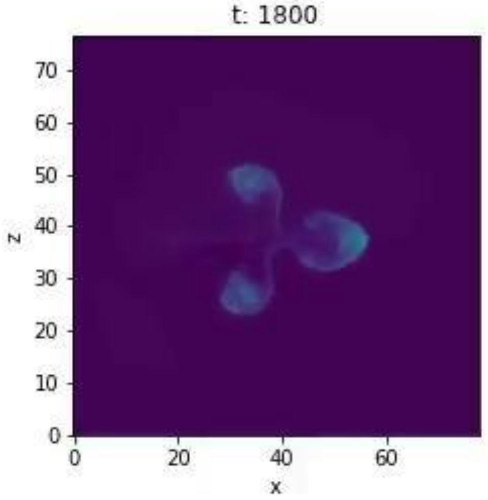
Ben Dudson[1], **Peter Hill**[1], Joseph Parker[2], John Omotani[2], David Dickinson[1], **David Schwörer**[3], the BOUT++ team

[1] Department of Physics, University of York, UK
[2] Culham Centre for Fusion Energy, Abingdon, UK
[3] Max-Planck-Institut für Plasmaphysik, Germany

BOUT++

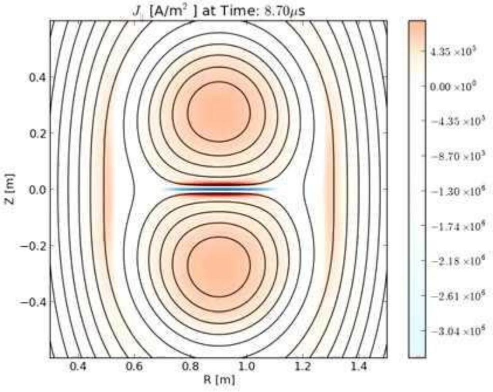
- Internationally used framework for solving PDEs in curvilinear geometry
- Specialised operators for plasma applications
- Domain-specific language in C++

```
phi = laplacian->solve(omega);
ddt(n) = -bracket(phi, n)
         + 2 * DDZ(n) * rho_s
         + D_n * Delp2(n);
ddt(omega) = -bracket(phi, omega)
              + 2 * DDZ(n) * rho_s / n
              + D_vort * Delp2(omega) / n;
```

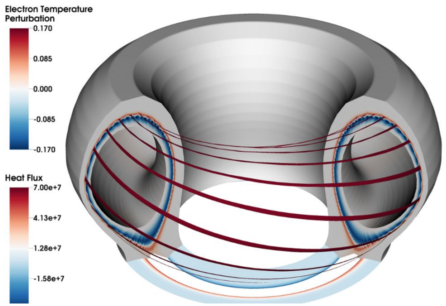


BOUT++ Capabilities

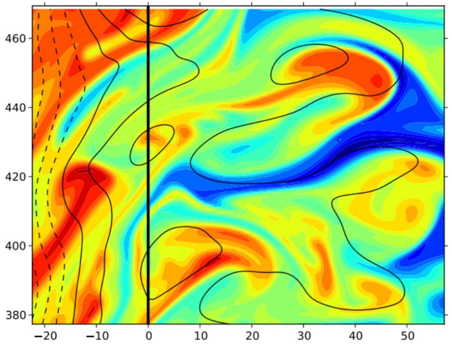
- Finite difference/volume in 3D mapped multi-block geometry
- Solves nonlinearly coupled hyperbolic, parabolic and elliptic equations
- MPI-parallelised, scales to ~4,000 cores, depending on problem
- Turbulence ~10^6—10^8 unknowns, ~10^5 core-hours



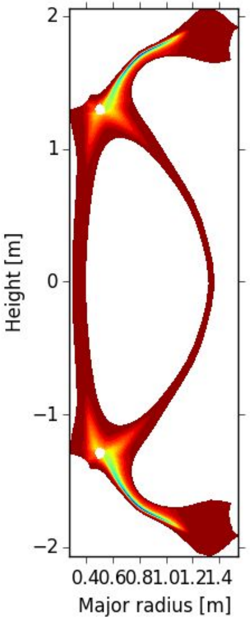
Magnetic reconnection



Edge Localised Modes



Turbulence



Transport

BOUT++ Applications

Many different models have been built on BOUT++

Different sets of equations, different geometries

<u>Topics</u>

- Edge Localised Modes
- Pedestal transport
- Filaments / blobs
- SOL turbulence
- Pellet injection
- RF interactions
- Magnetic Perturbations

<u>Formulation</u>

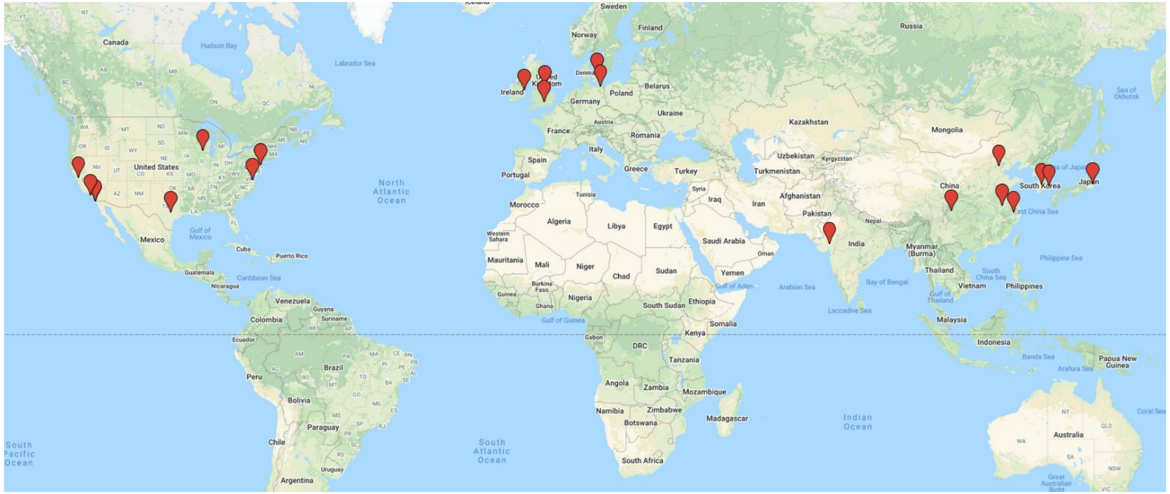
- Drift-reduced fluid
- Gyro-fluid
- Landau fluid closures
- Zonal flows
- Neutral interactions
- Impurity species

<u>Models</u>

- elm-pb/3f/5f/6f
- STORM
- Hermes
- SD1D
- CYTO
- Trans-neut
- BOUT++ GEM

BOUT++ Is Open Source

- Open source, users/developers worldwide
- Strong community, and investment in building capabilities to underpin research



<u>Top contributors</u>

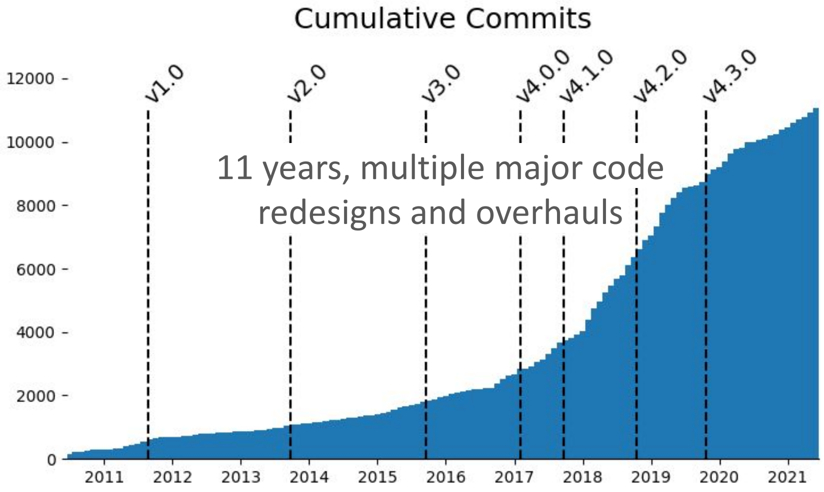
Peter Hill
Ben Dudson
David Dickinson
David Schwörer
John Omotani
Michael Loiten
Joseph Parker
Jens Madsen
Jarrod Leddy
George Breyiannis
Brendan Shanahan
Ilon Joseph
Hong Zhang
+ ~35 others

https://github.com/boutproject/

http://boutproject.github.io/

BOUT++ Is Open Source

- Open source, users/developers worldwide
- Strong community, and investment in building capabilities to underpin research



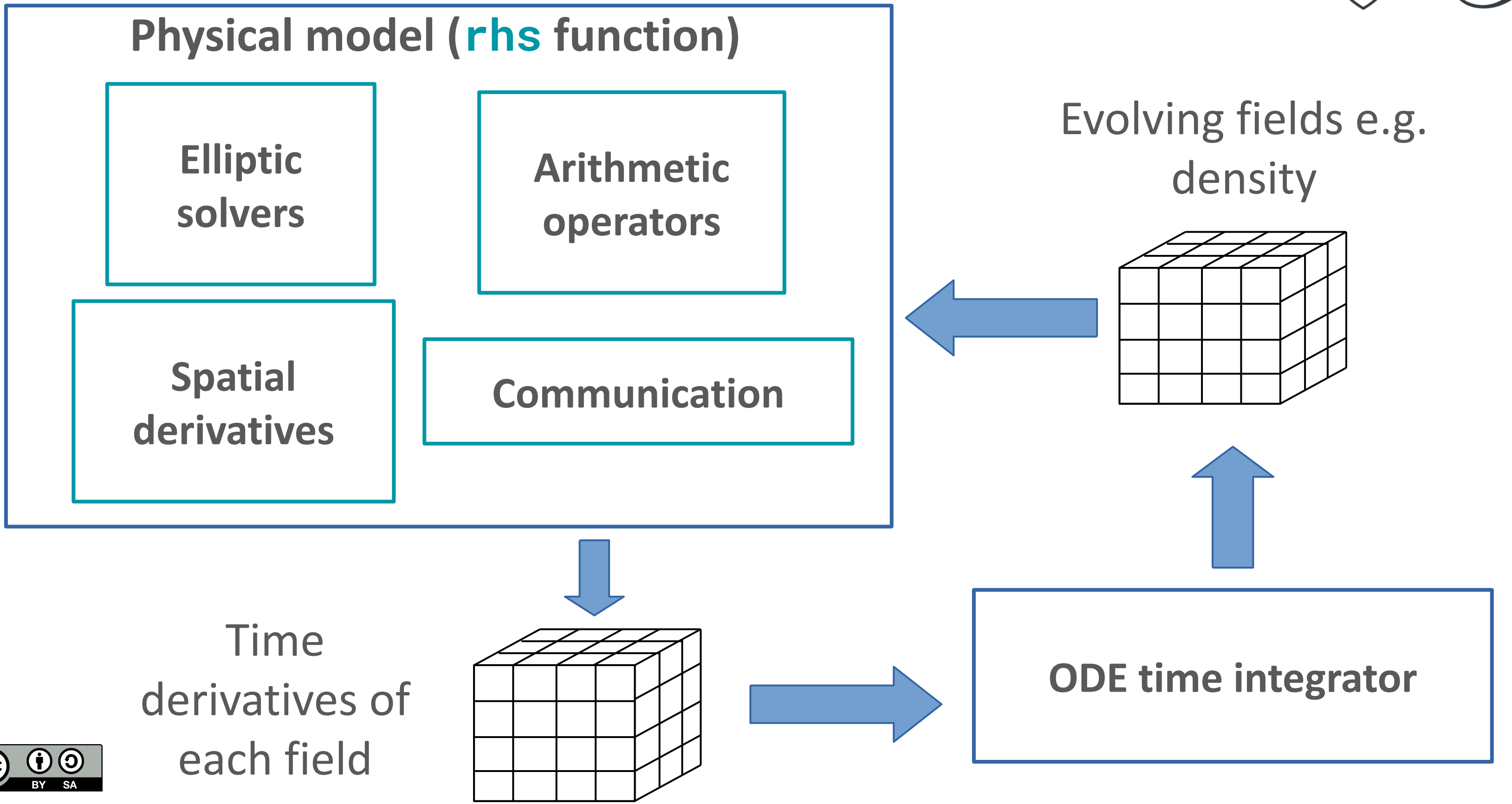
Top contributors

Peter Hill
Ben Dudson
David Dickinson
David Schwörer
John Omotani
Michael Loiten
Joseph Parker
Jens Madsen
Jarrod Leddy
George Breyiannis
Brendan Shanahan
Ilon Joseph
Hong Zhang
+ ~35 others

https://github.com/boutproject/

http://boutproject.github.io/

Matrix-free Method of Lines



BOUT++ Framework

UNIVERSITY of York

```
class Blob2D : public PhysicsModel {
private:
  // Evolving variables
  Field3D n, omega;
  …
public:
  int init(bool restarting);
  int rhs(BoutReal time);
};

BOUTMAIN(Blob2D);
```

Class inheriting from `PhysicsModel`

Model variables, input parameters

Model setup and initialisation

Calculation of time derivatives

Macro creating `main` entry point

BOUT++ Example Model

https://github.com/boutproject/BOUT-dev/tree/master/examples/blob2d

$$\frac{\partial n}{\partial t} = -\frac{1}{B}\underline{b} \times \nabla\phi \cdot \nabla n + 2\rho_s \frac{\partial n}{\partial z} + D_n \nabla_\perp^2 n$$

```
ddt(n) = -bracket(phi, n)
         + 2 * rho_s * DDZ(n)
         + D_n * Delp2(n);
```

$$\frac{\partial \omega}{\partial t} = -\frac{1}{B}\underline{b} \times \nabla\phi \cdot \nabla \omega + \frac{2\rho_s}{n} \frac{\partial n}{\partial z} + \frac{D_{vort}}{n} \nabla_\perp^2 \omega$$

```
ddt(omega) = -bracket(phi, omega)
             + (2 * rho_s / n) * DDZ(n)
             + D_vort * Delp2(omega) / n;
```

$$\omega = \nabla_\perp^2 \phi$$

```
phi = laplacian->solve(omega);
```

BOUT++ Flexibility

- Guiding principle of BOUT++ is flexibility
- Choice of numerical method for each operator
- Can be specified at runtime or compile time

BOUT.inp input file:

```
[mesh]
nx = 64
ny = 1
nz = 64

[mesh:ddx]
first = C4
second = C2
[mesh:ddz]
first = U2
```

blob2d.cxx source code:

```
ddt(n) = -bracket(phi, n, BRACKET_ARAKAWA)
         + 2 * DDZ(n, CELL_CENTRE, "FFT") * rho_s
         + D_n * Delp2(n);
```

Command line:

```
./blob2d solver:type=rk4 laplace:type=petsc
    mesh:nx=128 mesh:ddx:first=C2
```

BOUT++ Python Interface

- Cython wrapper around C++ interface
- Useful for post-processing and term decomposition

https://github.com/boutproject/boutcore-examples/blob/main/blob2d.py

```
class Blob2D(PhysicsModel):
  def init(self, restart):
    …
  def rhs(self, time):
    …
    ddt_n = self.n.ddt() # Limitation of Python
    ddt_n.set(0)         # interface
    ddt_n += -bracket(self.phi, self.n)
             + 2 * DDZ(self.n) * self.rho_s
             + self.D_n * Delp2(self.n)
    self.phi = self.laplacian.solve(self.omega,
                                    self.phi)
```

```
# Create an instance
blob2d = Blob2D()
# Start the simulation
blob2d.solve()
```