Introduction to BOUT++

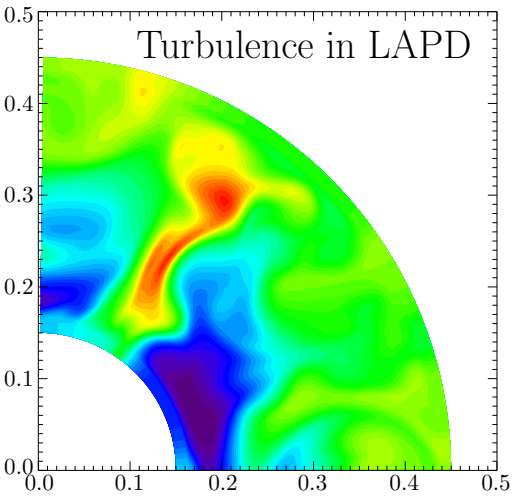
Ben Dudson
benjamin.dudson@york.ac.uk

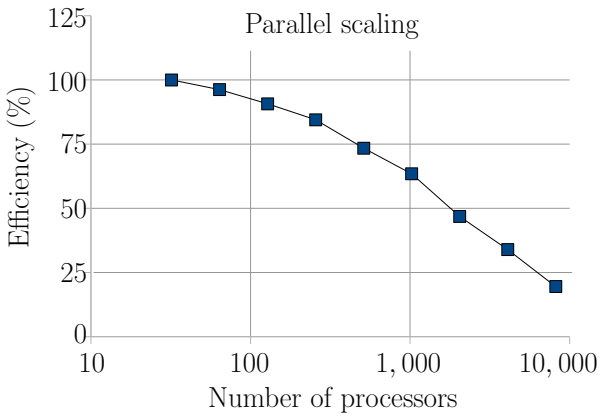
Department of Physics, University of York, Heslington, York YO10 5DD, UK

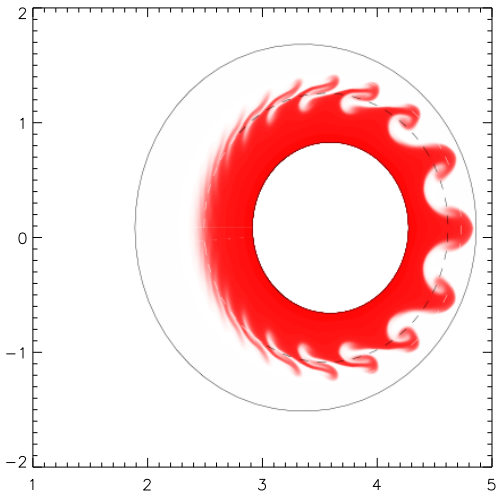
LLNL, 14th September 2011

The BOUT++ code

- Plasma fluid simulation framework[1]
- Solves an arbitrary number of fluid equations in curvilinear coordinates
- Finite difference with implicit or explicit timestepping. Methods can be changed at run-time, and include 4th-order Central differencing, Arakawa, and 3rd-order WENO.
- Written in C++, open source (LGPL)[2]







[1] B.D.Dudson et. al. Comp. Phys. Comm. 180 (2009), pp. 1467-1480

[2] Available at `http://github.com/bendudson/BOUT`

What is BOUT++

- Framework for writing fluid / plasma simulations in curvilinear geometry
- Finite-difference code, variety of numerical methods and time-integration solvers
- Written from scratch in C++, borrowing some ideas from the original BOUT code
- Intended to be quite modular, enabling fast testing of numerical methods
- Can evolve any number of equations, with equations appearing in a readable form
- Primarily designed and tested with reduced plasma fluid models in mind

What **isn't** BOUT++

- Not a general parallel simulation library. Better tools such as PETSc exist for that
- Not a magic bullet. It doesn't automate the process of choosing an appropriate numerical scheme, just makes it easier to implement and test different ones
- Not suitable for every problem. The numerical methods currently implemented are quite general, but cannot cover all problems

Overall aims

- Started with the aim of simulating ELMs. Appropriate physics model not known. Wanted to make the code easy to change
- Large codes often hard to understand, so wanted to isolate the model-specific code into a small number of lines
- Still hard to understand whole code, but clearer what problem is being solved

Overall aims

- Started with the aim of simulating ELMs. Appropriate physics model not known. Wanted to make the code easy to change
- Large codes often hard to understand, so wanted to isolate the model-specific code into a small number of lines
- Still hard to understand whole code, but clearer what problem is being solved

Now becoming more widely used, and aim is to build a community to use and develop the code further
Separated into model-specific and general code, so we can

- Work on multiple different physics problems separately
- Benefit from each other's improvements to the core code

Status and capabilities

- Equations appear in a form which is (reasonably) clear e.g.

```
ddt(Apar) = - Grad_par(phi);
```

- Can simulate a variety of fluid models. Mainly subsets of Braginskii, but also full MHD and compressible gas equations

Status and capabilities

- Equations appear in a form which is (reasonably) clear e.g.

```
ddt(Apar) = - Grad_par(phi);
```

- Can simulate a variety of fluid models. Mainly subsets of Braginskii, but also full MHD and compressible gas equations
- Usually uses a field-aligned Clebsch coordinate system, but most operators are general. Only needs the metric tensor components to be specified.

Status and capabilities

- Equations appear in a form which is (reasonably) clear e.g.

```
ddt(Apar) = - Grad_par(phi);
```

- Can simulate a variety of fluid models. Mainly subsets of Braginskii, but also full MHD and compressible gas equations
- Usually uses a field-aligned Clebsch coordinate system, but most operators are general. Only needs the metric tensor components to be specified.
- Contains a library of different numerical differencing methods, and time-integration schemes. Simple schemes built-in, but uses external libraries such as SUNDIALS and PETSc for advanced methods

Status and capabilities

- Equations appear in a form which is (reasonably) clear e.g.

```
ddt(Apar) = - Grad_par(phi);
```

- Can simulate a variety of fluid models. Mainly subsets of Braginskii, but also full MHD and compressible gas equations
- Usually uses a field-aligned Clebsch coordinate system, but most operators are general. Only needs the metric tensor components to be specified.
- Contains a library of different numerical differencing methods, and time-integration schemes. Simple schemes built-in, but uses external libraries such as SUNDIALS and PETSc for advanced methods
- Promising results for turbulence and ELM simulations. Xu will talk more about this next...
- For typical ELM simulations the code scales well to a few thousand cores

Work in progress

- Coupling to the PETSc library for time-stepping working and under development
- Gyro-fluid extensions (gyro-averaging operators) working and being tested
- Pre-processing routines to prepare equilibria functional, but needs improvement
- Test cases: many example problems, and some unit tests. More needed to allow regular regression testing
- Documentation. Quite extensive manuals, but lags behind code

Future developments

- Preconditioning methods, including physics-based
- More advanced numerical methods, both differencing and time-integration
- Improved handling of highly non-uniform meshes
- Additional differential operators to model effects like Landau damping
- Coupling to external databases or codes to model things like atomic physics, fuelling and interactions with core and walls
- Better visualisation tools, in languages other than IDL
- Use of external libraries (e.g. PETSc, hypre) for linear and nonlinear solvers
- Scalability beyond 10,000 cores

Access to BOUT++

The BOUT++ code is open source, and publically available at github.com

http://github.com/bendudson/BOUT

For this workshop, we have created a "stable" version 1.0, which may be updated with bugfixes, but no new features

http://github.com/bendudson/BOUT-1.0

Anyone can download a copy, but to make changes you will need to set up an account and SSH keys on github. Sean Farley will cover this after coffee...

Why use Git?

- Git was written by Linus Torvalds with Linux development in mind, so can easily handle very large collaborations and complicated merging
- Doesn't enforce any particular way of working, and doesn't have the concept of a "central" server - all copies of the code are equivalent
- A particular copy of BOUT++ is only "the" version by consent (or diktat)

This can seem strange coming from SVN, but makes it easier to work independently on features, then merge changes together afterwards.
$\Rightarrow$ Hopefully a help, rather than a hinderance to collaboration

Use of BOUT++

Contributing:

- BOUT++ is under the LGPL license, so code which uses it can be proprietry. Modifications to the BOUT++ library do come under the LGPL
- You're free to take and modify BOUT++ for any purpose
- We would appreciate it if you contributed back improvements you make to the code

Use of BOUT++

Contributing:

- BOUT++ is under the LGPL license, so code which uses it can be proprietry. Modifications to the BOUT++ library do come under the LGPL
- You're free to take and modify BOUT++ for any purpose
- We would appreciate it if you contributed back improvements you make to the code

Support:

- We're happy to help, but our time is limited
- One aim of this workshop is to get a group of people comfortable with using BOUT++ and (eventually) help support each other
- There is a BOUT++ development mailing list. Please let me know if you'd like to join it

Summary

- BOUT++ is a fluid simulation framework designed with plasma edge simulations in mind
- Less general than libraries like PETSc, still very flexible for plasma applications
- A tool to speed up development of new plasma models and numerical methods

Summary

- BOUT++ is a fluid simulation framework designed with plasma edge simulations in mind
- Less general than libraries like PETSc, still very flexible for plasma applications
- A tool to speed up development of new plasma models and numerical methods

- Not perfect...
  I look forward to working with you to improve it