Overview of the BOUT++ code structure

Ben Dudson

York Plasma Institute, University of York, UK

benjamin.dudson@york.ac.uk

3rd September 2013
BOUT++

- A toolbox for solving PDEs on parallel computers, together with pre- and post-processing codes. Aims to reduce duplication of effort, and allow quick development and testing of new physics models and simulations.
- A collection of examples and test cases.
- Focused on flute-reduced plasma models in field-aligned coordinate systems, but more general capabilities.

Is not:

- A single plasma model or simulation.
- A general library of numerical methods for parallel computing. Other tools like PETSc are available for that.
- Magic. Appropriate numerical schemes depend on the problem, and must be chosen intelligently by the user.
Key features

- Finite difference initial value code in 3D
- Implicit (e.g. BDF, C-N) or explicit (e.g. RK4, Karniadakis) time integration
- Coordinate system set in metric tensor components
- Handles complicated topology of X-point tokamak geometry
- Written in C++, quite modular design
- A growing community working to develop and exploit simulations using fluid and gyro-fluid models
Improvements since version 1.0

- Interfaces to PETSc (timestepping + linear solves) and
  MUMPS (linear solves).
  → Many sophisticated methods, more general problems

- Linear solvers for new classes of problems
  → Fast parabolic solves along (equilibrium) field lines

- Preconditioning schemes → faster simulations

- New differencing methods, flux conservative and limiter
  schemes, boundary conditions, ...

- Pre- and post-processing in more languages
  → IDL, Python, Matlab, Mathematica, Octave

- 3D visualisation using VisIt and Mayavi

- Many updates, fixes, restructuring
  configure scripts, manual, ...
Getting BOUT++

- Workshop release version 2.0
  
  [https://github.com/boutproject/BOUT-2.0](https://github.com/boutproject/BOUT-2.0)

- Version control using **git**, a distributed system designed for large collaborative projects (e.g. Linux kernel)

  → See [http://git-scm.com](http://git-scm.com)

- To download, run in terminal:

  ```
  $ git clone
  https://github.com/boutproject/BOUT-2.0.git
  ```

- To later update to latest version, change to BOUT-2.0 directory and run

  ```
  $ git pull
  ```
**BOUT++ structure**

- Separates generic methods from model-specific code
- Most of the code doesn't know or care about what a variable represents, its normalisation etc. Only needs to know the geometry and which operation to perform

---

**Physics model**

- Evolving variables
- Normalisation
- Model equations
  - ~100 - 1000 lines

---

**BOUT++ library**

- Time integration
- Differencing methods
- Boundary value solvers
- Mesh comms
- Input / Output
  - ~30,000 lines
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- manual
- src
- examples
- tools

- configure and make scripts
  (see user manual, and Maxim's talk)
- README and COPYING
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- manual
- src
- examples
- tools

User manual describes how to get started with BOUT++. 
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- manual
- src
- examples
- tools

### Data handling routines

- Memory allocation and handling
- Scalar and vector fields, and operations on them
- Initial and boundary conditions
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

**Data input and output**

- Reading and writing binary file formats (pdb, netcdf, ...)
- Interface for writing scalar and vector fields
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- manual
- src
- examples
- tools

**PDE solvers for boundary-value problems**

- Common routines for inverting Laplacian type equations
- Interfaces to PETSc, MUMPS, ... for some problems
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- **manual**
- **src**
  - **examples**
  - **tools**
- **field**
- **fileio**
- **invert**
- **mesh**
- **solver**
- **sys**

**Mesh**
- Distribution of grid points across processors
- Local ↔ Global
- Communications
- Metric tensor
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- manual
- src
- examples
- tools

Time integration

- Explicit and implicit methods
- Interface to external solvers in SUNDIALS and PETSc
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- **manual**
- **src**
- **examples**
- **tools**

### Miscellaneous utilities

- Options get/set to control behaviour
- Exception handling, timing and debugging aids
- MPI and PETSc start / stop
- Differencing methods
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- **manual**
- **src**
- **examples**
- **tools**

**Models and test cases**

- Test suite uses Python to run and check correctness
- Linear stability examples
- Slab (2D/3D) examples
- ELM simulations

- **test_suite**
  - **conduction**
  - **interchange-instability**
  - **blob2d**
  - **elm-pb**
  - **6field-simple**
Finding your way around

After downloading BOUT++ (or browsing online), you'll see

- manual
- src
- examples
- tools
- idlllib
- pylib
- matlablib
- Mathematica lib
- Tokamak grids

Pre- and post-processing

- “Collect” routines to read output data for various languages
- Processing and visualisation
- Mesh generation
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);

phiSolver = Laplacian::create();

RHS function evaluation (called by solver)

phi = phiSolver->solve(vort, phi);

mesh->communicate(n, vort, phi);

ddt(n) = -bracket(phi, n, bm) + alpha*(phi - n) - kappa*DDZ(phi) - Dn*Delp4(n);
ddt(vort) = -bracket(phi, vort, bm) + alpha*(phi - n) - Dvort*Delp4(vort);
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);
phiSolver = Laplacian::create();

RHS function evaluation (called by solver)

phi = phiSolver->solve(vort, phi);
mesh->communicate(n, vort, phi);

ddt(n) = -bracket(phi, n, bm) + alpha*(phi - n) – kappa*DDZ(phi)
- Dn*Delp4(n);

ddt(vort) = -bracket(phi, vort, bm) + alpha*(phi – n)
- Dvort*Delp4(vort);

Tree of options controlling behaviour. Set in input file:

[hw]
alpha = 0.4
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);

phiSolver = Laplacian::create();

phi = phiSolver->solve(vort, phi);

mesh->communicate(n, vort, phi);

ddt(n) = -bracket(phi, n, bm) + alpha*(phi - n) - kappa*DDZ(phi)
    - Dn*Delp4(n);

ddt(vort) = -bracket(phi, vort, bm) + alpha*(phi - n)
    - Dvort*Delp4(vort);

Macros for common tasks:

options->get(alpha, "alpha", 1.0);
options->get(kappa, "kappa", 0.1);
solver->add(n, "n");
solver->add(vort, "vort")
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);

phiSolver = Laplacian::create();

RHS function evaluation (called by solver)

phi = phiSolver->solve(vort, phi);

mesh->communicate(n, vort, phi);

ddt(n) = -bracket(phi, n, bm) + alpha*(phi - n) - kappa*DDZ(phi) - Dn*Delp4(n);
ddt(vort) = -bracket(phi, vort, bm) + alpha*(phi - n) - Dvort*Delp4(vort);
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);
phiSolver = Laplacian::create();

RHS function evaluation (called by solver)

phi = phiSolver->solve(vort, phi);

mesh->communicate(n, vort, phi);

ddt(n) = -bracket(phi, n, bm) + alpha*(phi - n) - kappa*DDZ(phi)
   - Dn*Delp4(n);

ddt(vort) = -bracket(phi, vort, bm) + alpha*(phi - n)
   - Dvort*Delp4(vort);

Guard cell communication explicit to allow optimisation (send...calculate ...wait)

Equations appear in easily readable form
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);
phiSolver = Laplacian::create();

RHS function evaluation (called by solver)

phi = phiSolver->solve(vort, phi);
mesh->communicate(n, vort, phi);

\[
\begin{align*}
\text{ddt}(n) &= -\text{bracket}(\phi, n, bm) + alpha*(\phi - n) - kappa*DDZ(\phi) \\
&\quad - Dn*Delp4(n); \\
\text{ddt}(\text{vort}) &= -\text{bracket}(\phi, \text{vort}, bm) + alpha*(\phi - n) \\
&\quad - D\text{vort}*Delp4(\text{vort});
\end{align*}
\]

Equations appear in easily readable form

Overloaded operators, not template expressions currently
Example: Hasegawa-Wakatani

Field3D n, vort, phi;

Options *options = Options::getRoot()->getSection("hw");
OPTION(options, alpha, 1.0);
OPTION(options, kappa, 0.1);

SOLVE_FOR2(n, vort);

phiSolver = Laplacian::create();

RHS function evaluation (called by solver)

phi = phiSolver->solve(vort, phi);
mesh->communicate(n, vort, phi);

ddt(n) = -bracket(phi, n, bm) + alpha*(phi - n) – kappa*DDZ(phi)
- Dn*Delp4(n);

Boundary conditions in input file e.g.
[n] bndry_all = dirichlet

Boundary conditions in input file e.g.
[n] bndry_all = dirichlet

Derivative methods set in options e.g.
[ddz] first = C4  # 4th-order Central difference
BOUT++ component patterns

Most components now follow the same “factory” pattern

Options

Factory

Implementation A

Implementation B

Implementation B

A header file defines the interface
include/invert_laplace.hxx

Common routines and factory in src:
src/invert/laplace/laplacefactory.cxx

Individual implementations in subdirectory
src/invert/laplace/impls/...

The factory is the only place where individual headers are included, so forces rest of the code to be independent.

See developer manual for more details
Using BOUT++ (conclusions)

- BOUT++ is open source, under the LGPL license.
- Allows linking to proprietary code, but modifications to core of BOUT++ come under LGPL.

- You are free to take and modify BOUT++ for any purpose
- Please contribute improvements and fixes back to the community
- Use of BOUT++ and contributed components should be acknowledged through co-authorship and/or citations

- One aim of this workshop is to establish a solid community basis for collaboration