BOUT++ overview

Ben Dudson
BOUT++ workshop 9th January 2023

Thanks to contributors including: Peter Hill, Mike Kryjak, Joseph Parker, John Omotani, David Dickinson, Yining Qin, Steven Glenn, Xueqiao Xu, and the BOUT++ team

This work was performed in part under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC
BOUT++ is an ecosystem of plasma simulation tools

**Pre-processing**
- Hypnotoad grid generator

**Models**
- elm-pb (3,4,5,6-field)
- Gyro-fluid models
- Trans-neut
- SD1D

**BOUT++**
- Meshing
- I/O
- FD, FV methods
- MPI
- Time integrators
- Testing

**Hermes**

**Post-processing**
- STORM
- SOLT3D

**Tools**
- PETSc
- hypre
- sundials
- SLEPc
- RAJA
- xarray
- MPI
- Testing
BOUT++ underpins many different models

- Solves nonlinearly coupled hyperbolic, parabolic and elliptic equations
- MPI-parallelised, scales to ~4,000 cores, depending on problem size
- Turbulence $\sim 10^6 - 10^8$ unknowns, $\sim 10^5$ core-hours

![Edge Localised Modes (LLNL)](image)

![Magnetic reconnection](image)

![Plasma turbulence](image)

![Transport (MAST-U)](image)
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 Schworer
- John Omotani
- Michael Loiten
- Joseph Parker
- Jens Madsen
- Jarrod Leddy
- George Breyiannis
- Brendan Shanahan
- Ilon Joseph
- Hong Zhang
- + ~35 others

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 Schworer
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/
Overview

- BOUT++ structure

- Major changes
  - GPUs: RAJA and Hypre
  - 3D geometries: The FCI method

- Hermes-3: Building on BOUT++
BOUT++ uses matrix-free Method of Lines

- Physical model (**rhs** function)
  - Elliptic solvers
  - Arithmetic operators
  - Spatial derivatives
  - Communication
- Evolving fields e.g. density
- ODE time integrator
- Time derivatives of each field
Close correspondence between model and code

\[
\frac{\partial n}{\partial t} = -\frac{1}{B} b \times \nabla \phi \cdot \nabla n + 2 \rho_s \frac{\partial n}{\partial z} + D_n \nabla^2 n
\]

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

\[
\omega = \nabla^2 \phi
\]

Domain-specific language in C++

\[
\text{ddt}(n) = -\text{bracket}(\phi, n)
+ 2 \times \text{DDZ}(n) \times \rho_s
+ D_n \times \text{Delp2}(n);
\]

\[
\text{ddt}(\omega) = -\text{bracket}(\phi, \omega)
+ 2 \times \text{DDZ}(n) \times \rho_s / n
+ D_{vort} \times \text{Delp2}(\omega) / n;
\]

\[
\phi = \text{laplacian}\rightarrow\text{solve}(\omega);
\]

Elliptic inversion

https://github.com/boutproject/BOUT-dev/tree/master/examples/blob2d
Guiding principle of BOUT++ is flexibility

- Choice of numerical method for each operator
- Can be specified at runtime or compile time
- A flexible input configuration format, with arbitrary expressions (Turing complete)

**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
```
Improvement in performance over time

Joseph Parker, David Dickinson, Peter Hill and Ben Dudson

BOUT++ Workshop 2018

CPU only, pure MPI, 512 x 64 x 16 grid

<table>
<thead>
<tr>
<th>proc count</th>
<th>v3.1 Oct 16</th>
<th>v4.0 Feb 17</th>
<th>v4.1 Sep 17</th>
<th>v4.2 Oct 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>96</td>
<td>67</td>
<td>39</td>
<td>30</td>
</tr>
<tr>
<td>256</td>
<td>60</td>
<td>33</td>
<td>25</td>
<td>23</td>
</tr>
</tbody>
</table>

Elliptic inversion bottleneck
Overview

- BOUT++ structure

- Major changes
  - GPUs: RAJA and Hypre
  - 3D geometries: The FCI method

- Hermes-3: Building on BOUT++
Major changes [Oct 2018 – Jan 2022]

- Regions for iterating over arbitrary domains. Improved vectorization and OpenMP performance [v4.2, 4.3]
- Consistent support for staggered grids [v4.2 – v5]
- Improved support for 3D coordinates and complex boundaries [v4.3 – v5]
- Input file language extended, internationalized [v4.3]
- Adopted the CMake build system [v4.4 – v5]
- Input & output data provenance tracking [v4.4-v5]
- Replaced I/O system, using flexible dictionary structure to exchange data [v5]
- GPU and CPU improved performance with RAJA [v5]
- New steady-state solver for transport problems, borrowing from UEDGE [v5]
- Many more tests: Now 1853 unit, 61 integrated, and 22 MMS tests

Version 5 has 3,699 commits, 91k lines changed, compared to v4.4.2
https://github.com/boutproject/BOUT-dev/pull/2604
Currently in the “next” github branch
Balancing usability and performance

- In BOUT++ functions operate on whole fields (arrays of data)
  - Simple interface for non-C++ experts
  - Each operation (+, -, *, /, DDX) loops over the domain
  - These loops are too small to parallelise efficiently (esp on GPUs)

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

\[
\text{ddt}(n) = -\text{bracket}(\phi, n) + 2 \times \text{DDZ}(n) \times \rho_s + D_n \times \text{Delp2}(n);
\]

This has 9 separate kernels, each with a loop over the domain.
Balancing usability and performance

- In BOUT++ functions operate on whole fields (arrays of data)
  - Simple interface for non-C++ experts
  - Each operation (+, -, *, /, DDX) loops over the domain
  - These loops are too small to parallelise efficiently (esp on GPUs)

---

Steve Glenn, Bill Meyer (LLNL)
**Code transformation methods**

- **MACROS**
  - ✔ Conceptually simple
  - ✔ Familiar to most programmers

- **C++ templates**
  - ✗ Works on a text level, not semantic
  - ✗ Can lead to surprising bugs (text mangling)
  - ✗ No access to type information
  - ✗ Not suitable for complex transformations

- **Code generation**

  Used in BOUT++ to reduce “boilerplate”, and define loops that can be changed at compile time (e.g. OpenMP, RAJA).
Code transformation methods

MACROS

- General transformations (Turing complete)
- Widely used to merge loops
  e.g. Eigen, xtensor, Blitz++, Kokkos, ...

C++ templates

- Unhelpful error messages
- Compilation can be slow
- Complex, requires experienced developers to maintain and extend code
- Can run into compiler bugs

Used in BOUT++ to enable compile-time checks, use types to specialise code

Code generation
Code transformation methods

MACROS
- ✔ General transformations
- ✔ Many different tools available
- ✔ Abstracts over implementation and architecture details

C++ templates
- ✗ Debugging can be very difficult
- ✗ Link between user code and performance can be unclear
- ✗ May need to maintain code generation tool

Code generation

Jinja template engine used in BOUT++ to generate repetitious code (https://jinja.palletsprojects.com)
Using RAJA & Umpire to port to GPUs

- RAJA provides mechanisms to generate CUDA code from C++:
  
  
  ```cpp
  RAJA::forall<EXEC_POL>(RAJA::RangeSegment(0, indices.size()),
  [=] RAJA_DEVICE(int id) {
    /* ... your code here ... */
  })
  ```

  Execution policy e.g CUDA. Compile-time choice
  
  Iteration range
  
  C++ lambda function body

  ✓ Provides a path for incremental porting existing code to GPUs
  
  – Requires additional tools to manage memory. Umpire used here.
  
  X A “leaky abstraction”: Details of memory, CUDA limitations matter
  (especially with complex data structures)
Opt-in performance tuning

- Aim to maintain usability, readability for physicists
- Incrementally transition from original code to improve performance
- Ease debugging (c.f. templates, code generation)

\[\text{ddt}(n) = -\text{bracket}(\phi, n) + 2 \times \text{DDZ}(n) \times \rho_s + D_n \times Delp2(n);\]

Original

\[
\text{BOUT}\_\text{FOR}(i, \text{region}) \{ \\
\text{ddt}(n)[i] = -\text{bracket}(\phi, n, i) + 2 \times \text{DDZ}(n, i) \times \rho_s + D_n \times Delp2(n, i); \\
\}
\]

Merged loops

Macro: OpenMP, vectorise, RAJA
Opt-in performance tuning: checks outside loops

- Borrow an API idea from SYCL: Lightweight wrappers of raw buffers
- Run-time checking performed on construction (outside loop)
- Template arguments enable compile-time checks, optimisations

```cpp
auto n_acc = FieldAccessor<>(n);
auto phi_acc = FieldAccessor<>(phi);
auto jpar_acc = FieldAccessor<CELL_YLOW>(Jpar);

BOUT_FOR(i, region) {
    ddt(n)[i] = -bracket(phi_acc, n_acc, i)
               + 2 * DDZ(n_acc, i) * rho_s
               + D_n * Delp2(n_acc, i);
}
```

Run-time checks

Compile-time type checking
Performance improvements: RAJA

Running on Lassen, LLNL

- Only one kernel launch per iteration, due to loop merging
- 1260 x 1256 grid for benchmarking

- GPU loop speedup = $\frac{77}{17} \approx 4.5X$
- Overall speedup = $\frac{423}{376} \approx 1.13X$

**Single-Thread, CPU-only**

- Loop: 239 ms
- Overall time: 423 ms

**GPU-enabled Loop**

- Loop: 248 ms
- Overall time: 376 ms

![Diagram showing performance improvements](image)
Ongoing work to port to GPUs

- Merging kernels and RAJA works well
- Significant time can be spent in inversion of elliptic operators
- Keeping GPUs busy can be hard
- Setup costs are significant
  (note: matrix is time-dependent!)

Test on Lassen

16M points, IBM Power 9, NVIDIA V100, x1.59 speedup

GPU-enabled Loop

- Laplacian Inversion: 376 ms
- Solve: 248 ms
- Loop: 17 ms
GPU functionality is available in v5

- Check out the “next” (development) branch of BOUT++:

  ```
  $ git clone -b next https://github.com/boutproject/BOUT-dev.git
  ```


- Examples
  - blob2d-outerloop
  - hasegawa-wakatani-3d
    [https://github.com/boutproject/BOUT-dev/tree/next/examples/hasegawa-wakatani-3d](https://github.com/boutproject/BOUT-dev/tree/next/examples/hasegawa-wakatani-3d)
  - elm-pb-outerloop
    [https://github.com/boutproject/BOUT-dev/tree/next/examples/elm-pb-outerloop](https://github.com/boutproject/BOUT-dev/tree/next/examples/elm-pb-outerloop)
Grid generation using Hypnotoad

- Python grid generator, mainly written by J.Omotani
- Can generate non-orthogonal grids, here using orthogonal grids
- Interactive GUI or automated script
- Can adjust packing of cells around separatrix and/or close to targets as needed
- Sequence of grids created for convergence and performance testing
- Python tools for interpolating between grids
Complex meshing problems (2D & 3D)

X-point

Snowflake

Stellarator

W.A.J. Vijvers et al 2014 Nucl. Fusion 54 023009

FCI: Field-line following + interpolation

Shifted metric (Dimits / Scott) : 1D interpolation


An illustration of the Flux Coordinate Independent method for parallel derivatives [9].
Gridding of poloidal plane independent of magnetic field structure

Zoidberg

http://bout-dev.readthedocs.io/en/latest/user_docs/zoidberg.html

BSTING project
B.Shanahan, D.Bold
IPP Greifswald
Overview

- BOUT++ structure

- Major changes
  - GPUs: RAJA and Hypre
  - 3D geometries: The FCI method

- Hermes-3: Building on BOUT++
Model development: SOL & Divertor turbulence

Hermes (2D)

Hermes (3D)

Key features:
‘full-f’ : Evolve profiles + fluctuations
Includes transport physics e.g. neutrals

Hermes-3: Multi-species transport and turbulence models

- Arbitrary number of species and ionisation states: D, T, He, Ne, …
- Full-f, flux-driven transport & turbulence
Recent simulations in 1-, 2- and 3D

- 1D transport of neon in deuterium SOL plasma
  - Evolving each charge state and atomic species as a separate fluid

- 2D transport in DIII-D geometry
  - Solve for deuterium, tritium and helium ions and atoms

- 3D turbulence and transport in LAPD
  - Both isothermal, and hot electron (ionisation source). Single ion species

- 3D turbulence and transport in DIII-D & limiter plasmas
  - Isothermal, single ion species to start with
System of equations is specified in the input file

- Top-level components specify the species, collective effects and modifiers

```
[hermes]
components = (e, d+, sound_speed, vorticity, sheath_boundary, collisions, polarisation_drift)
```

- Each species’ equations are:

```
[e]
type = evolve_density, evolve_momentum, isothermal
```

```
[d+]
type = quasineutral, evolve_momentum, isothermal
```
Solving drift-reduced fluid equations

**Electrons**

\[
\frac{\partial n_e}{\partial t} = - \nabla \cdot \left[ n_e (v_{E \times B} + bv_{\| e}) \right]
\]

\[
\frac{\partial}{\partial t} (m_e n_e v_{\| e}) = - \nabla \cdot \left[ m_e n_e v_{\| e} (v_{E \times B} + bv_{\| e}) \right] - \partial \| p_e - e n_e E\| + m_e n_e \nu_{ei} (v_{\| d} - v_{\| e})
\]

\[
p_e = n_e T_e
\]

**Ions**

\[n_{d+} = n_e\]

\[
\frac{\partial}{\partial t} (m_{d+} n_{d+} v_{\| d+}) = - \nabla \cdot \left[ m_{d+} n_{d+} v_{\| d+} (v_{E \times B} + bv_{\| d+}) \right] - \partial \| p_{d+} + e n_{d+} E\| + m_e n_e \nu_{ei} (v_{\| e} - v_{\| d})
\]

\[
p_{d+} = n_{d+} T_{d+}
\]

**Drifts**

\[
v_{E \times B} = \frac{b \times \nabla \phi}{B}
\]

**Vorticity**

\[
\nabla \cdot \left( \frac{m_i n}{B^2} \nabla_{\perp} \phi \right) = \omega
\]

\[
\frac{\partial \omega}{\partial t} = - \nabla \cdot (\omega v_{E \times B}) + \nabla \cdot (n_{d+} v_{\| d} - n_e v_{\| e})
\]
3D turbulence in LAPD geometry

- Isothermal, single ion species, no neutrals
- Uniform source of particles in domain; sheath boundary at both ends
- Resolution: 64 x 16 x 64 (radial x parallel x azimuthal)

Density near axis, middle of domain

Electron density [m$^{-3}$]
We can run turbulence simulations with an arbitrary number of ion species (e.g. D + He)

- No code changes needed. Input file specifies species and equations
- Here showing deuterium and helium (1+) ions
- Fuelling at 50/50 ratio, enhanced helium fraction near sheaths
1D multi-fluid transport

- Model a 1D domain, from “upstream” (no-flow) to “target” (sheath)
- D+, Ne+ … Ne+10 ions, D & Ne atoms. Only plotting highest density species.

![Graph showing temperature and density distribution](image-url)

- Power into D+ ions and electrons
- Log density scale

Upstream (midplane)  Target
2D transport in DIII-D geometry

- Resolution: 64 x 128 (radial x poloidal, excluding boundary)
- D+, T+ and He+ ions; D, T and He atoms (fluids)
2D transport in DIII-D geometry

- Resolution: 64 x 128 (radial x poloidal, excluding boundary)
- D+, T+ and He+ ions; D, T and He atoms (fluids)
Flexible tool for edge simulations
Note: under development

- All of these simulations run the same executable
  - Species, equations & reactions are configured in input file
  - Geometry (1,2,3D; linear, tokamak) in input or separate mesh file

- Atomic reactions and multi-ion support:
  - Hydrogen and helium atomic reactions from Amjuel
  - Neon reactions from ADAS
  - Tskhakaya & Kuhn multi-ion sheath boundary conditions

- Many solver / time integration options, making use of PETSc, Hypre & SUNDIALS

Some applications
Note: under development

- Tokamak edge and divertor transport & turbulence modelling
  - Including neutrals, impurities and drifts
  - Comparison to DIII-D data, particularly impurity injection effects on turbulence

- Multi-ion plasma turbulence
  - Validation on LAPD (experiments proposed)
  - More work on multi-ion closures probably needed

- Interested?
  - Github repository: https://github.com/bendudson/hermes-3
Conclusions

- BOUT++ underpins a wide range of research
- Continues to develop to meet research needs, with contributions from a global community

Thank you to all contributors!

Welcome to the 2023 BOUT++ workshop!