BOUT++ performance

Ben Dudson, Joseph Parker, Peter Hill,
David Dickinson

BOUT++ Workshop, LLNL
15th August 2018

EPSRC
Engineering and Physical Sciences Research Council
EUROfusion

Outline

- **Overall scaling**
- **Low level optimisation**
  - Loop vectorisation, OpenMP parallelisation
  - Inner vs Outer loops
- **Time integration**
  - Physics based preconditioning
  - Implicit-Explicit methods (IMEX)
  - Jacobian Coloring + AMG
- **Discussion**

Profiling with Scalasca / Scorep

- Instrument functions by using SCORP0() macro

```
#include “bout/scorepwrapper.hxx”

int init(bool restart) {
  SCOREP0();
  ...
}
```

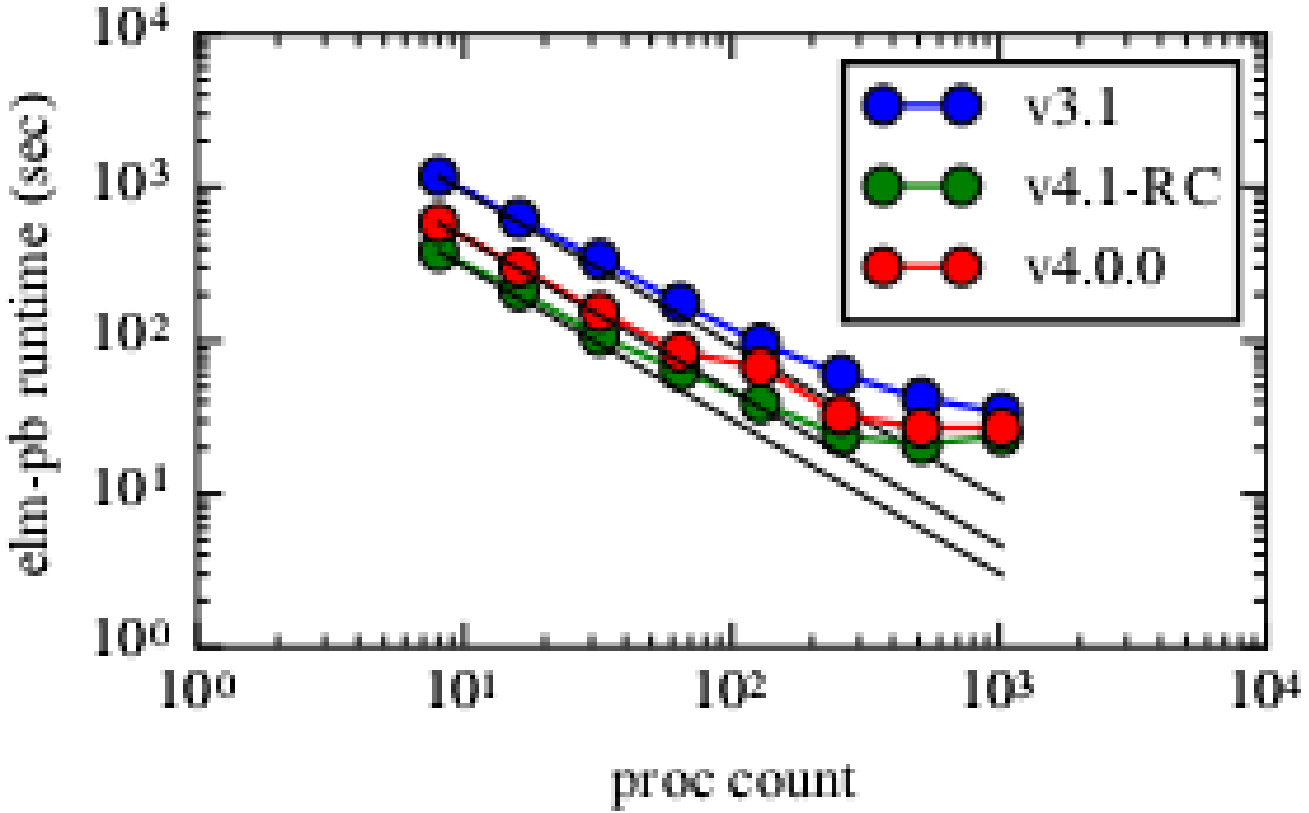
- Load module (if needed) and configure

```
module load scalasca
./configure –enable-scorep
make
scalasca -analyse mpirun -np 48 elm-pb
```

Scaling: elm-pb

J.Parker

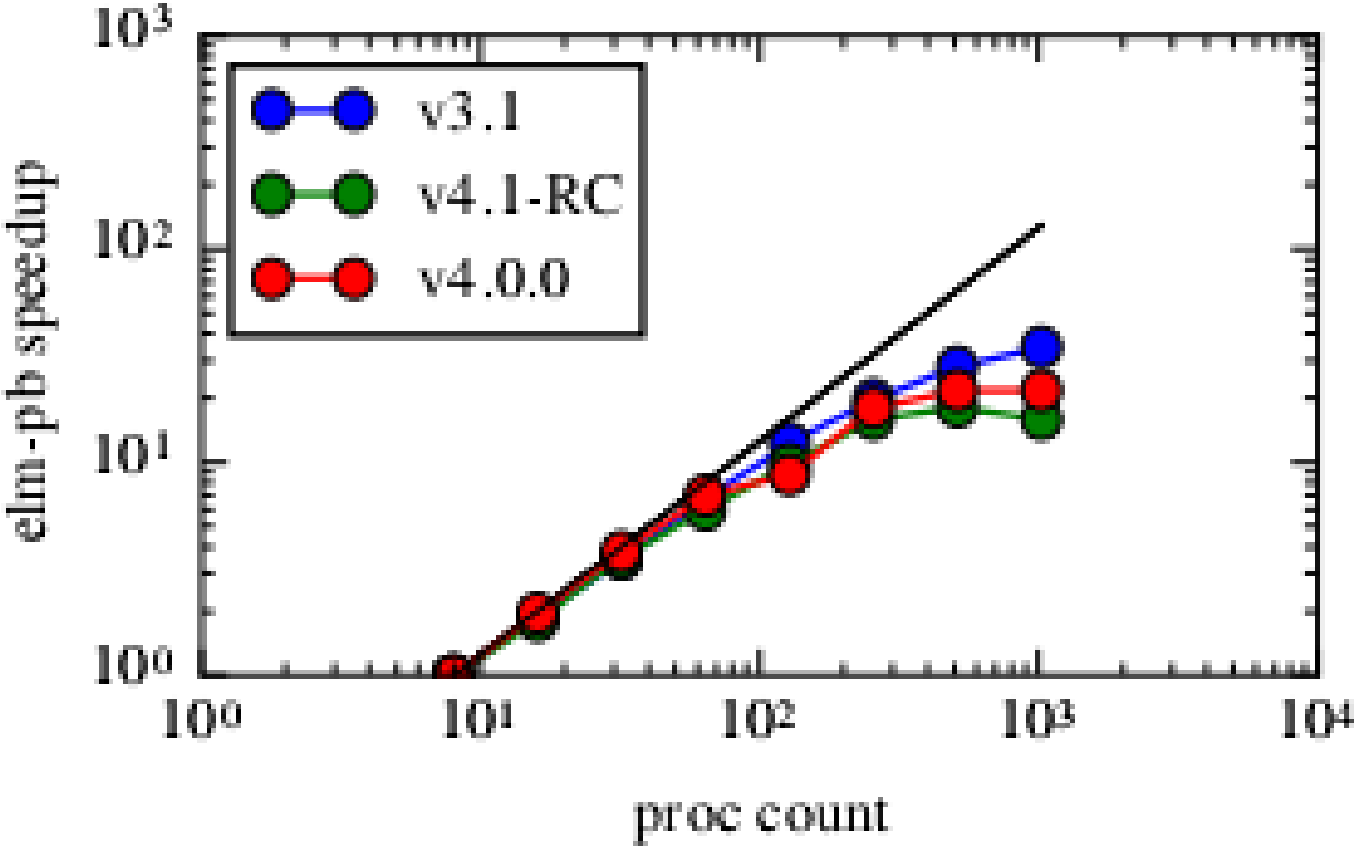
Over the last ~ 2 years the speed has improved significantly
→ Can do with 32 processors what would take 128 before



Scaling: elm-pb

J.Parker

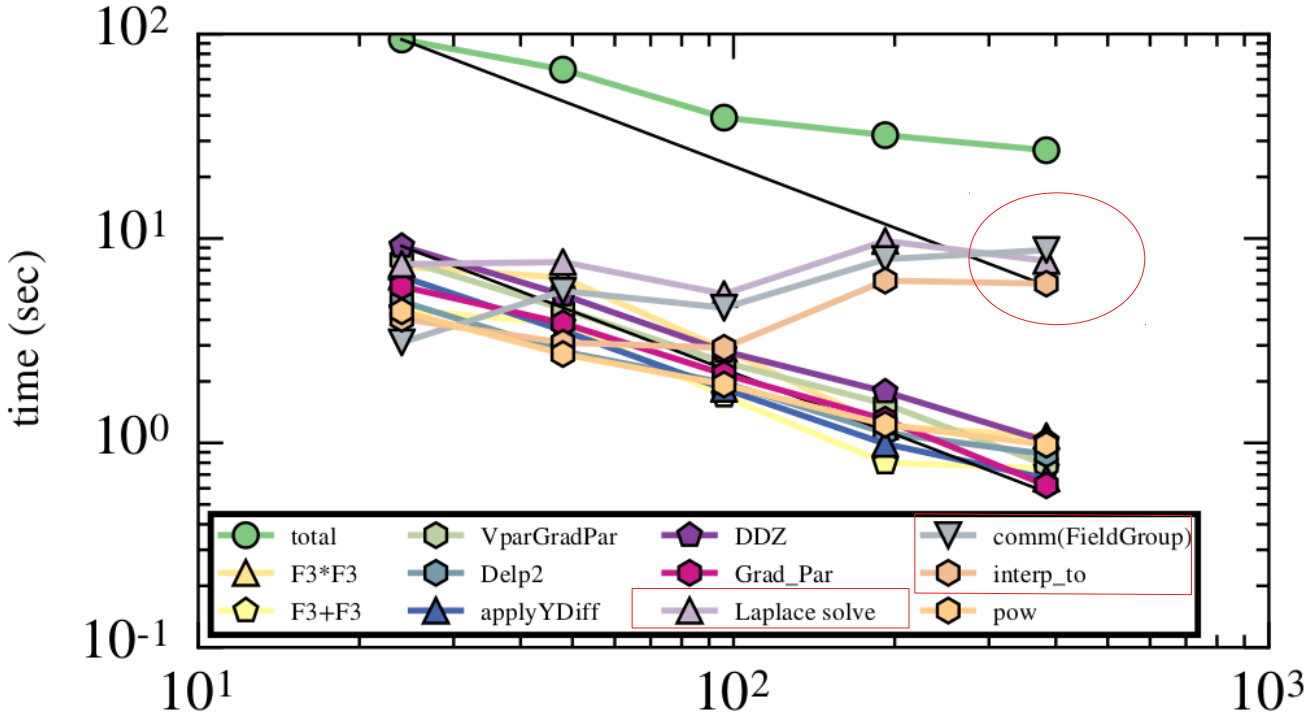
Unfortunately this improved efficiency means the scaling doesn't look quite as good...



Scaling: STORM

J.Parker

Scaling using Scalasca on Marconi, 196 x 16 x 128 grid
Staggered grids, RK4 timestepping, MPI only.



Bottlenecks: Laplacian inversion, communication, interpolation

Outline

- Overall scaling
- **Low level optimisation**
  - Loop vectorisation, OpenMP parallelisation
  - Inner vs Outer loops
- **Time integration**
  - Physics based preconditioning
  - Implicit-Explicit methods (IMEX)
  - Jacobian Coloring + AMG
- **Discussion**

C++11, clearer faster loops

P.Hill, D.Dickinson, J.Parker, B.Dudson

Previous BOUT++ versions used C-style pointers, nested loops

```
for(int i=0;i<mesh->ngx;i++)
  for(int j=0;j<mesh->ngy;j++)
    for(int k=0;k<mesh->ngz;k++)
      result[i][j][k] = a[i][j][k] + b[i][j][k];
```

Now use C++11 range-based loops:

```
for(auto i : result)
  result[i] = a[i] + b[i];
```

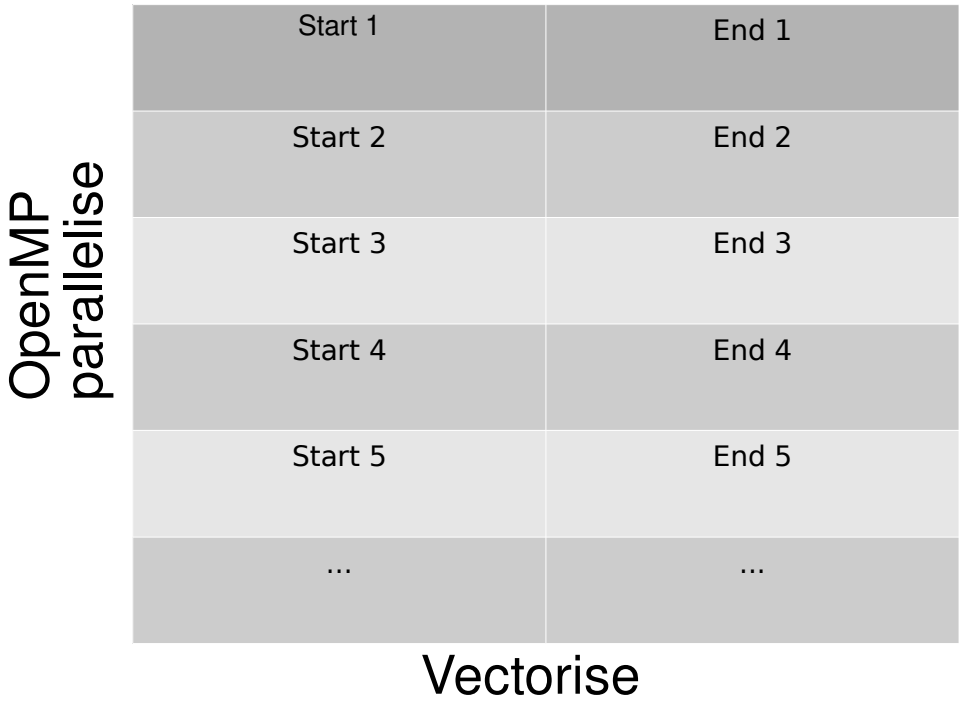
We would like this loop to be over arbitrary regions (e.g. for differential operators, boundary conditions)

How to get this to both OpenMP parallelise and vectorise?

C++11, clearer faster loops

P.Hill, D.Dickinson, J.Parker, B.Dudson

The solution used is to separate into an inner loop which can vectorise, and an outer loop which can be OpenMP parallelised



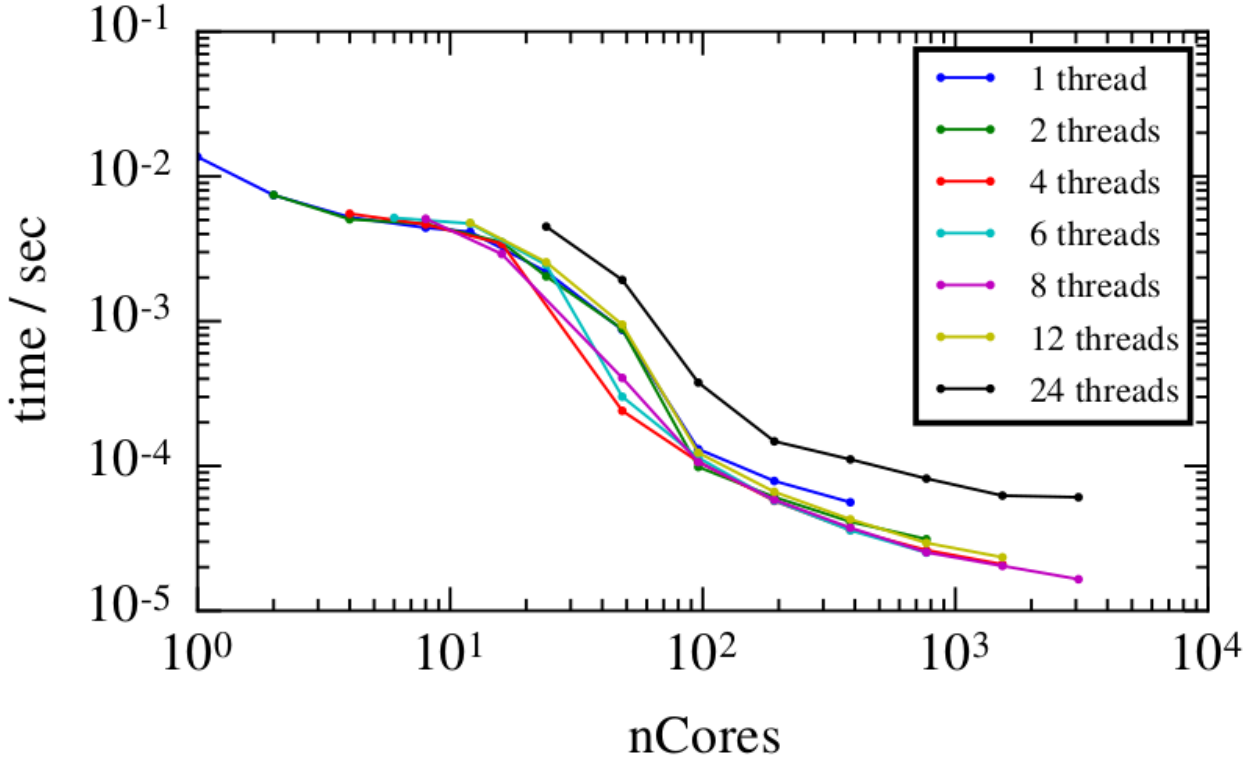
Need sufficient contiguous block size to fill a vector buffer (~128 usually enough)

Need sufficient number of blocks to spread work >= number of threads

C++11, clearer faster loops

P.Hill, D.Dickinson, J.Parker, B.Dudson

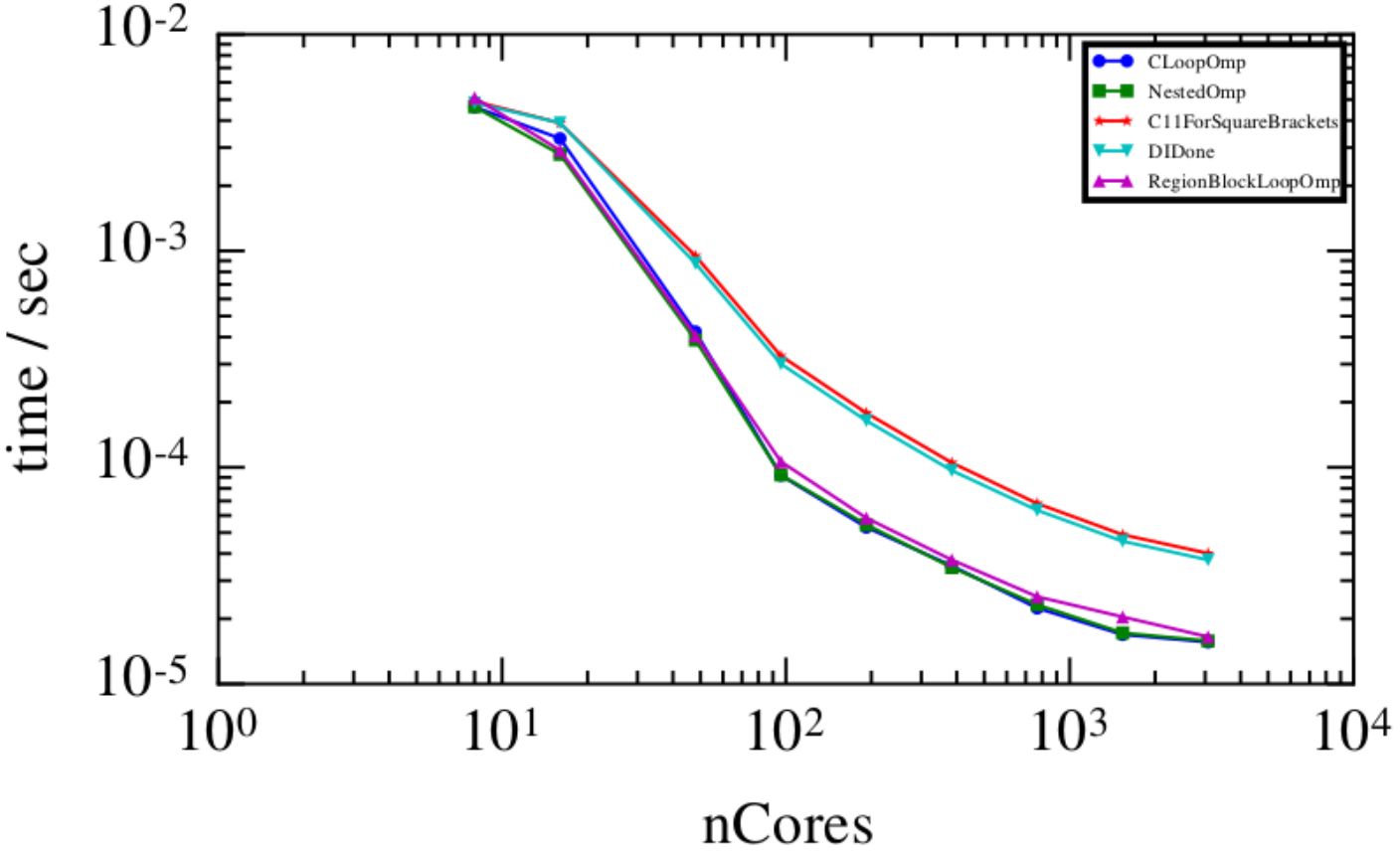
Testing a simple loop with algebraic operations on fields
New method shows good scaling, particularly with ~ 8 threads
(NUMA region has 12 cores)



C++11, clearer faster loops

P.Hill, D.Dickinson, J.Parker, B.Dudson

Comparing to other methods, ~same as C loop, faster than basic C++11 range-for loop.



Inner vs Outer loops

Operator overloading is used for arithmetic operators

```
r = a*b + c
```

Needs 2 loops over the domain. Can be inefficient for small grids or many cores

```
r.allocate();
BLOCK_REGION_LOOP(mesh->getRegion3D(“RGN_ALL”), i) {
  r[i] = a[i]*b[i] + c[i];
}
```

Needs only one loop. More work for each thread, and should have fewer cache misses.

Inner vs Outer loops

The **blob2d** example is a simplified 2D model of a plasma blob

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

4.27 x 10^{-2} sec. per iteration

```
5.000e+01        62        2.65e+00    71.7   17.2    0.0    0.3    10.7
1.000e+02        28        1.21e+00    71.5   17.2    0.0    0.8    10.6
```

The **blob2d-outerloop** example uses operators at given index

```
ddt(n)[i] = -bracket_arakawa(phi, n, i)
          + 2 * DDZ_C2(n, i) * (rho_s / R_c)
```

3.26 x 10^{-2} sec. per iteration

```
5.000e+01        54        1.76e+00    62.0   22.4    0.0    0.5    15.0
1.000e+02        19        6.35e-01    60.8   21.8    0.0    1.7    15.6
```

Summary of low-level optimisation

- Loops are now about as efficient as they can be
- Many parts of BOUT++ now OpenMP parallel and use vectorisation
- Further improvements may come from reducing the number of loops, rearranging operations, and higher level changes
- Laplacian inversion, communications and interpolation the main scaling bottlenecks (for STORM model)

Outline

- **Overall scaling**
- **Low level optimisation**
  - Loop vectorisation, OpenMP parallelisation
  - Inner vs Outer loops
- **Time integration**
  - Physics based preconditioning
  - Implicit-Explicit methods (IMEX)
  - Jacobian Coloring + AMG
- **Discussion**

Plasma equations

- Coupled set of nonlinear partial differential equations. Typically < 10 scalar quantities evolved per grid point, with ~10^6 - 10^8 grid points.
- Approximately incompressible fluid in 2D plane perpendicular to the magnetic field. Coupled hyperbolic and elliptic problem (some parabolic terms: cross-field diffusion, viscosity).
- Compressible fluid along magnetic field. Mixed hyperbolic and parabolic: fast heat conduction with strongly nonlinear conduction coefficient ~ T^(5/2)
- Light wave removed from the system analytically, but retains e.g.
  - shear Alfven wave, ~ 10^7m/s,
  - Sound speed ~ 10^5 m/s,
  - electron thermal speed ~10^6m/s.
- Ion cyclotron timescale removed analytically (~ 0.1 microseconds), but typical timesteps much smaller, typically limited by 3D (parallel) dynamics

Large range of timescales

- Initial Alfvenic oscillations f~500 kHz damp on ~20 μs timescale
- Followed by slower oscillation with f ~ 6.7 kHz
- Profiles evolving on ~ 10ms timescale (0.1 kHz)

Shear Alfven wave (global)

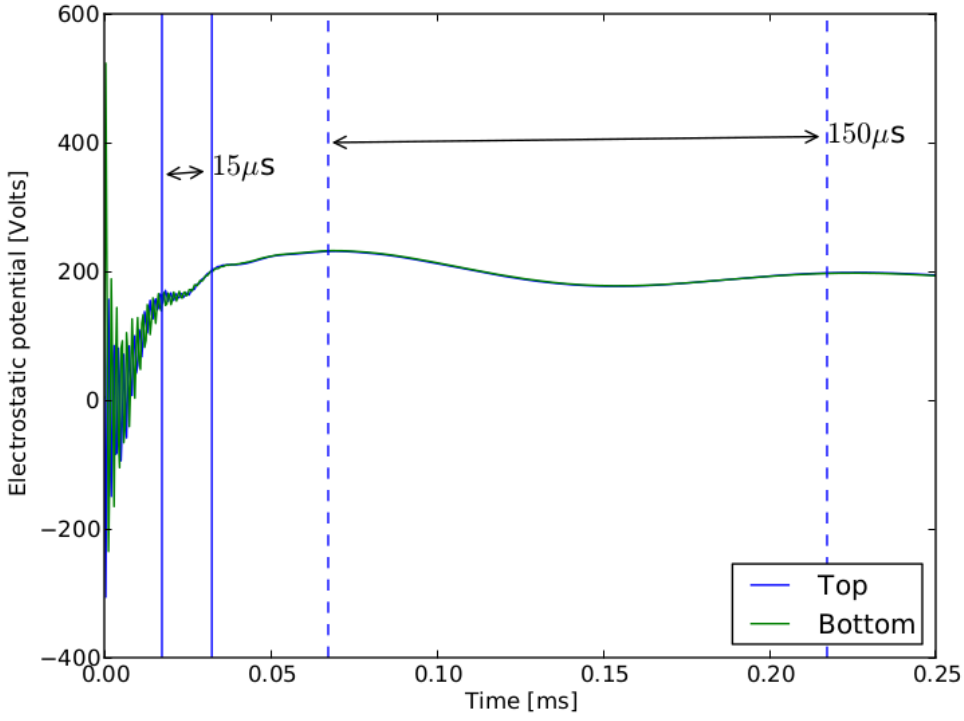
$$f_A = v_A/(2\pi Rq)$$
$$\simeq 550 - 1100\text{kHz}$$

Geodesic Acoustic Mode

$$f_{GAM} = \frac{c_s}{2\pi R}\sqrt{2 + 1/q^2}$$
$$\simeq 3 - 11\text{kHz}$$

Parallel sound wave

$$f_s = c_s/(2\pi Rq)$$
$$\simeq 0.5 - 2.3\text{kHz}$$



Challenges

- Relatively long simulation times are needed for profile evolution
  ~ 10s – 100s of milliseconds
- Turbulence is on short timescales ~ microseconds. Simulation timestep can't be bigger than this.
  → Each timestep must be fast

- Typically timesteps are limited to a small fraction of the ion cyclotron time (~ 1e-8 seconds)
- Usually limited by electron parallel dynamics: 2D (drift-plane) simulations have much larger timesteps than 3D
- Fully implicit time integration (JFNK) hard to precondition for these problems: complicated operator with many dispersive waves, strong 3D coupling of electrostatic potential through vorticity and electron dynamics
  → Better preconditioning strategies needed

Implicit time integration

See BOUT++ workshop 2011 (L.Chacon), 2013 (B.Dudson)

- Example: a first-order scheme (Backwards Euler):

$$\frac{\partial \mathbf{f}}{\partial t} = \mathbf{G}\left(\mathbf{f}\right) \qquad \mathbf{f}^{n+1} \simeq \mathbf{f}^{n} + \Delta t \mathbf{G}\left(\mathbf{f}^{n+1}\right)$$

- Newton-Krylov solvers used to solve this nonlinear system of equations

$$\mathbf{G}\left(\mathbf{f}^{n+1}\right) \simeq \underbrace{\left.\frac{\partial \mathbf{G}}{\partial \mathbf{f}}\right|_{n}}_{\mathbb{J}} \mathbf{f}^{n+1} \qquad \boxed{\left(\mathbb{I} - \Delta t \mathbb{J}\right)\mathbf{f}^{n+1} \simeq \mathbf{f}^{n}}$$

- Typically **f** is ~10 – 100 million variables, so J is a large matrix
- Never need to calculate or store J. Use Jacobian Free method:

$$\mathbb{J}\mathbf{v} \simeq \left[\mathbf{G}\left(\mathbf{f}^{n} + \epsilon \mathbf{v}\right) - \mathbf{G}\left(\mathbf{f}^{n}\right)\right]/\epsilon$$

- Fast time scales make this equation more singular and harder to solve
  → We need a **preconditioner**

Preconditioning

- Preconditioning improves the condition number of this matrix inversion, enabling convergence in fewer iterations, or larger time step.

$$(\mathbb{I} - \Delta t \mathbb{J})\, \mathbf{f}^{n+1} \simeq \mathbf{f}^{n}$$

- Usually requires an approximate Jacobian, containing the fastest timescales (largest eigenvalues)

- Two main approaches tried so far:
  - Physics-based preconditioning
  - Jacobian coloring

Example: MHD model

2-fluid collisional plasma model, with density, Te and Ti

$$\frac{\partial \rho}{\partial t} = -\mathbf{v}_{E\times B}\cdot\nabla\left(\rho+\rho_0\right)+\left(\nabla\cdot\mathbf{v}_{E\times B}\right)\left(\rho+\rho_0\right)+D_\perp\nabla_\perp^2\rho$$

$$\frac{\partial T_s}{\partial t} = -\mathbf{v}_{E\times B}\cdot\nabla\left(T_s+T_{s0}\right)-\frac{2}{3}\left(\nabla\cdot\mathbf{v}_{E\times B}\right)\left(T_s+T_{s0}\right)$$
$$+\frac{1}{\rho+\rho_0}\left[\nabla_{||}\cdot\left(\chi_{s||}\partial_{||}T_s\right)+\chi_\perp\nabla_\perp^2 T_s\right]+\frac{2}{3\left(\rho+\rho_0\right)}W_s$$

$$\frac{\partial U}{\partial t} = -\mathbf{v}_{E\times B}\cdot\nabla U+\frac{1}{\rho+\rho_0}\left[B_0^2\nabla_{||}\left(\frac{J_{||}+J_{||0}}{B_0}\right)+2\underline{b}_0\times\underline{\kappa}_0\cdot\nabla P+\nu_{||}\partial_{||}^2 u+\nu_\perp\nabla_\perp^2 u\right]$$

$$\frac{\partial v_{||}}{\partial t} = -\mathbf{v}_{E\times B}\cdot\nabla v_{||}-\nabla_{||}P$$

$$\frac{\partial A_{||}}{\partial t} = -\nabla_{||}\phi-\eta J_{||}$$

$$P = \rho\left(T_e+T_i\right)$$

$$U = \frac{1}{B_0}\nabla_\perp^2\phi+\frac{1}{B_0}\nabla_\perp^2\frac{P}{en}$$

Chosen to be similar to JOREK:
G.Huysmans PPCF **51** (2009) 124012
but with addition of ion diamagnetic term

B Dudson, S Farley, L Curfmann-McInnes ArXiv: plasm-phys/1209.2054
B Dudson et al JPP 2015

Simplify the model

- Need an efficient way to find an approximate solution to

$$(\mathbb{I} - \Delta t \mathbb{J})\,\mathbf{f}^{n+1} \simeq \mathbf{f}^{n} \qquad\qquad \mathbb{M}^{-1}\,(\mathbb{I} - \Delta t \mathbb{J}) \simeq \mathbb{I}$$

  where J is the Jacobian of the system, evolving variables **f**
- Use what we know about the physics to design a preconditioner

L.Chacon, Phys. Plasmas 15 (2008) 056013

1) Simplify the equations

$$\frac{\partial \rho}{\partial t} \simeq -\mathbf{v}_{E\times B}\cdot\nabla_{\perp}\rho_0$$

$$\frac{\partial T_s}{\partial t} \simeq -\mathbf{v}_{E\times B}\cdot\nabla_{\perp}T_{s0} + \frac{1}{\rho_0}\nabla_{||}\cdot\left(\chi_{s||}\partial_{||}T_s\right)$$

$$\frac{\partial U}{\partial t} \simeq \frac{1}{\rho_0}\left[B_0^2\nabla_{||}\left(\frac{J_{||}}{B_0}\right) + 2\mathbf{b}_0\times\mathbf{k}_0\cdot\nabla P\right]$$

$$\frac{\partial v_{||}}{\partial t} \simeq 0$$

$$\frac{\partial A_{||}}{\partial t} \simeq -\nabla_{||}\phi$$

$$f = \begin{pmatrix} \rho \\ T_i \\ T_e \\ v_{||} \\ A_{||} \\ U \end{pmatrix}$$

B Dudson, S Farley, L Curfmann-McInnes ArXiv: plasm-phys/1209.2054
B Dudson et al JPP 2015

Simplify the model

- Need an efficient way to find an approximate solution to

$$(\mathbb{I} - \Delta t \mathbb{J})\,\mathbf{f}^{n+1} \simeq \mathbf{f}^{n} \qquad \mathbb{M}^{-1}(\mathbb{I} - \Delta t \mathbb{J}) \simeq \mathbb{I}$$

  where J is the Jacobian of the system, evolving variables **f**
- Use what we know about the physics to design a preconditioner

L.Chacon, Phys. Plasmas 15 (2008) 056013

1) Simplify the equations

Parallel heat conduction

$$\frac{\partial \rho}{\partial t} \simeq -\mathbf{v}_{E\times B}\cdot\nabla_{\perp}\rho_0$$

$$\frac{\partial T_s}{\partial t} \simeq -\mathbf{v}_{E\times B}\cdot\nabla_{\perp}T_{s0} + \frac{1}{\rho_0}\nabla_{||}\cdot(\chi_{s||}\partial_{||}T_s)$$

$$\frac{\partial U}{\partial t} \simeq \frac{1}{\rho_0}\left[B_0^2\nabla_{||}\left(\frac{J_{||}}{B_0}\right) + 2\mathbf{b}_0\times\mathbf{k}_0\cdot\nabla P\right]$$

$$\frac{\partial v_{||}}{\partial t} \simeq 0$$

$$\frac{\partial A_{||}}{\partial t} \simeq -\nabla_{||}\phi$$

Shear Alfven wave

$$f = \begin{pmatrix} \rho \\ T_i \\ T_e \\ v_{||} \\ A_{||} \\ U \end{pmatrix}$$

B Dudson, S Farley, L Curfmann-McInnes ArXiv: plasm-phys/1209.2054
B Dudson et al JPP 2015

Calculate Jacobian and factorise

1) Simplify the equations

2) Calculate Jacobian. Partial derivatives of RHS w.r.t variables

$$\mathbb{J} = \left[\begin{array}{ccccc|c} 0 & 0 & 0 & 0 & 0 & \frac{1}{B}\mathbf{b}\times\nabla\rho_0\cdot\nabla\nabla_\perp^{-2}[B_0 \\ 0 & \frac{\chi_{i,||}}{\rho_0}\nabla_{||}^2 & 0 & 0 & 0 & \frac{1}{B}\mathbf{b}\times\nabla T_{i0}\cdot\nabla\nabla_\perp^{-2}[B_0 \\ 0 & 0 & \frac{\chi_{e,||}}{\rho_0}\nabla_{||}^2 & 0 & 0 & \frac{1}{B}\mathbf{b}\times\nabla T_{e0}\cdot\nabla\nabla_\perp^{-2}[B_0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\nabla_{||}\nabla_\perp^{-2}[B_0 \\ \hline \frac{2}{\rho_0}\mathbf{b}\times\kappa\cdot\nabla\left[(T_{e0}+T_{i0}\right) & \frac{2}{\rho_0}\mathbf{b}\times\kappa\cdot\nabla\left[\rho_0\right. & \frac{2}{\rho_0}\mathbf{b}\times\kappa\cdot\nabla\left[\rho_0\right. & 0 & -\frac{1}{\rho_0}B_0\nabla_{||}\nabla_\perp^2 & 0 \end{array}\right] \begin{array}{c} \rho \\ T_i \\ T_e \\ v_{||} \\ A_{||} \\ U \end{array}$$

3) Shur factorise the matrix to be solved

$$(\mathbb{I}-\gamma\mathbb{J})^{-1} = \left[\begin{array}{cc} \mathbb{E} & \mathbb{U} \\ \mathbb{L} & \mathbb{D} \end{array}\right]^{-1} = \left[\begin{array}{cc} \mathbb{I} & -\mathbb{E}^{-1}\mathbb{U} \\ 0 & \mathbb{I} \end{array}\right]\left[\begin{array}{cc} \mathbb{E}^{-1} & 0 \\ 0 & \mathbb{P}_{Schur}^{-1} \end{array}\right]\left[\begin{array}{cc} \mathbb{I} & 0 \\ -\mathbb{L}\mathbb{E}^{-1} & \mathbb{I} \end{array}\right]$$

$$\mathbb{P}_{Schur} \simeq \mathbb{I} - \gamma^2\frac{B_0^2}{\rho_0}\nabla_{||}\nabla_\perp^2\nabla_{||}\nabla_\perp^{-2}\cdot$$ ← Shear Alfven wave

Decouple parallel and perpendicular

1) Simplify the equations
2) Calculate Jacobian. Partial derivatives of RHS w.r.t variables
3) Shur factorise the matrix to be solved
4) Use an approximation to decouple parallel and perpendicular derivatives

$$\nabla_{||}\nabla_{\perp}^{2}\nabla_{||}\nabla_{\perp}^{-2} \simeq \nabla_{||}^{2} - \frac{2\nabla_{||}^{2}(RB_{\theta})}{RB_{\theta}}$$

$$\mathbb{P}_{Schur}^{-1} \simeq \left(\mathbb{I} + \frac{2\gamma^{2}\nabla_{||}(RB_{\theta})}{RB_{\theta}B_{0}^{2}} - \gamma^{2}\frac{B_{0}^{2}}{\rho_{0}}\nabla_{||}^{2}\right)^{-1}$$

Shear Alfven wave

$$\mathbb{E}^{-1} \rightarrow \left(\mathbb{I} - \gamma\frac{\chi_{||s}}{\rho_{0}}\nabla_{||}^{2}\right)^{-1}$$

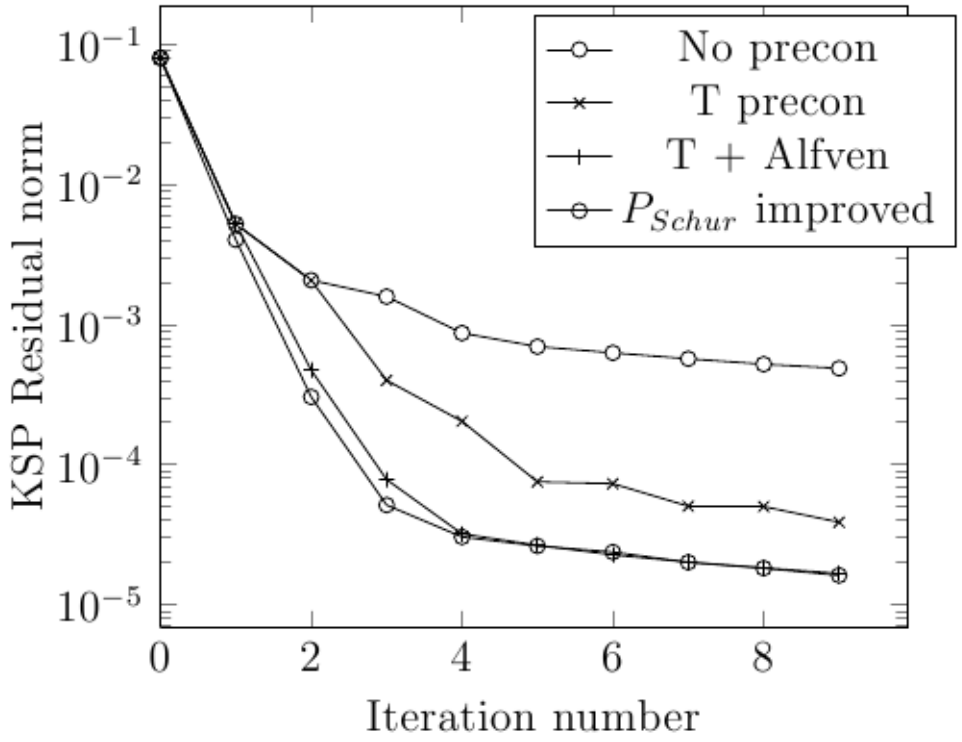
Parallel heat conduction

$$\left(a + b\nabla_{||}^{2}\right)^{-1}$$

Generic solver class, reduces this to many 1D systems, solved simultaneously

Preconditioning can be highly effective

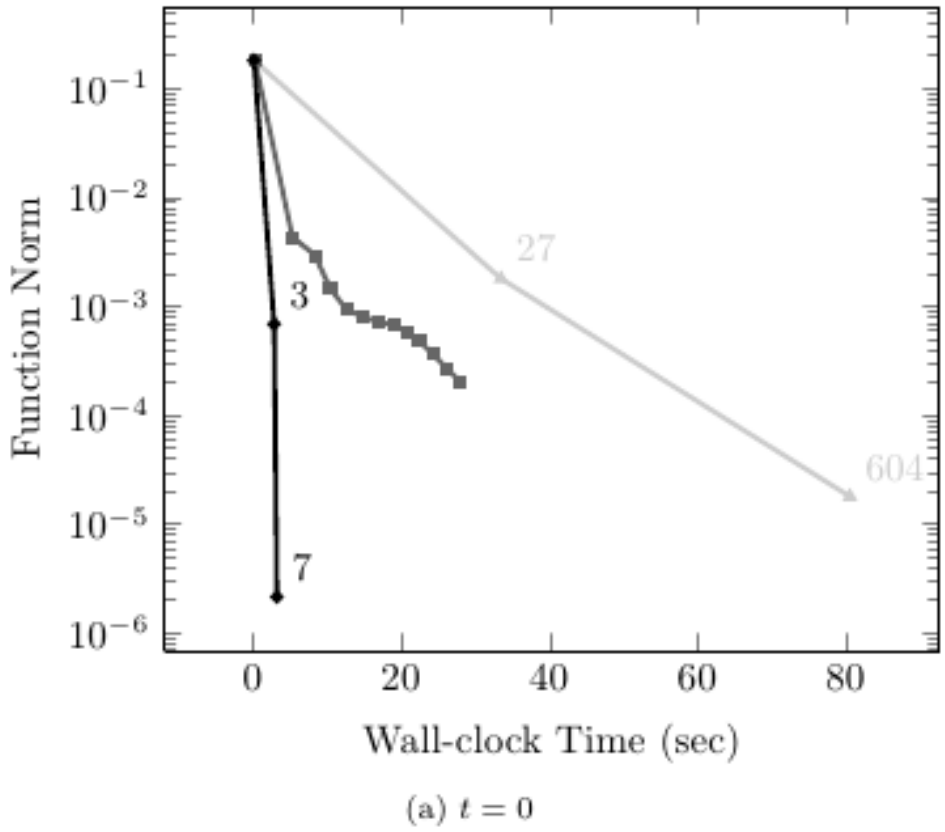
- Preconditioner uses the same operators as the time-derivative evaluation.
- Easily implemented as another call-back function
- Rapidly converges to solution, in cases which previously stalled



B Dudson, S Farley, L.C. McInnes
ArXiv: plasm-phys/1209.2054

Preconditioning can be highly effective

- Preconditioner uses the same operators as the time-derivative evaluation.
- Easily implemented as another call-back function
- Rapidly converges to solution, in cases which previously stalled
- Led to significant speedups (x10 – 100 in some cases)



(a) $t = 0$

B Dudson, S Farley, L.C. McInnes
ArXiv: plasm-phys/1209.2054

Preconditioning examples

Several examples with documentation:

- **`examples/preconditioning/wave`** is a simpler wave equation. Described in the online manual.
- **`examples/reconnect-2field`** evolves just vorticity and A||, a simplified version of the MHD problem shown earlier.
- **`examples/preconditioning/diffusion-nl`** solves a nonlinear diffusion equation using preconditioning and/or IMEX methods

Dispersive waves

- The preconditioner described previously works for MHD-type problems, but doesn't work with slightly more complicated models.

Isothermal electrons

$$\frac{1}{en_0}\partial_{||}P_e = \frac{T_e}{n_0}\partial_{||}n$$

$$\begin{aligned}
\frac{\partial n}{\partial t} &= \nabla\cdot\left(\mathbf{b}\frac{J_{||}}{e}\right) \\
\frac{\partial U}{\partial t} &= \nabla\cdot\left(\mathbf{b}J_{||}\right) \qquad U \simeq \frac{m_i n_0}{B^2}\nabla_\perp^2\phi \\
\frac{\partial A_{||}}{\partial t} &= -\partial_{||}\phi + \frac{1}{en_0}\partial_{||}P_e - \eta J_{||} \qquad J_{||} = -\frac{1}{\mu_0}\nabla_\perp^2 A_{||}
\end{aligned}$$

Dispersive waves

- The preconditioner described previously works for MHD-type problems, but doesn't work with slightly more complicated models.

$$\frac{\partial n}{\partial t} = \nabla \cdot \left(\mathbf{b} \frac{J_{||}}{e} \right)$$

$$\frac{\partial U}{\partial t} = \nabla \cdot \left(\mathbf{b} J_{||} \right) \qquad U \simeq \frac{m_i n_0}{B^2} \nabla_\perp^2 \phi$$

$$\frac{\partial A_{||}}{\partial t} = -\partial_{||} \phi + \frac{1}{e n_0} \partial_{||} P_e - \eta J_{||} \qquad J_{||} = -\frac{1}{\mu_0} \nabla_\perp^2 A_{||}$$

Isothermal electrons

$$\frac{1}{e n_0} \partial_{||} P_e = \frac{T_e}{n_0} \partial_{||} n$$

The dispersion relation becomes:

$$\omega^2 + \frac{i\eta}{\mu_0} k_\perp^2 \omega = k_{||}^2 V_A^2 \left(1 + \rho_s^2 k_\perp^2 \right)$$

$$\rho_s = c_s / \omega_{ci} = \sqrt{\frac{T_e m_i}{e B^2}}$$

$$V_A = B / \sqrt{\mu_0 m_i n_0}$$

→ Parallel wave speed depends on perpendicular wavenumber

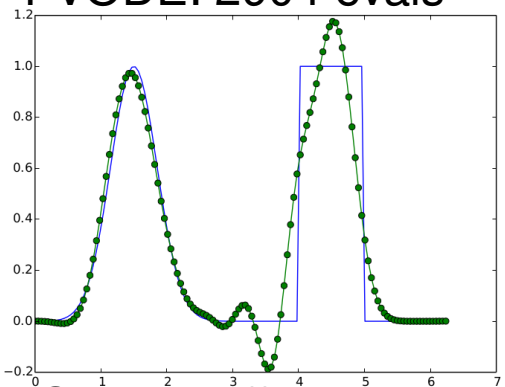
Implicit – Explicit (IMEX) schemes

- For many problems the implicit PVODE/CVODE solvers work well
- In some cases however they can fail

Example: Advection of a pulse in 1D

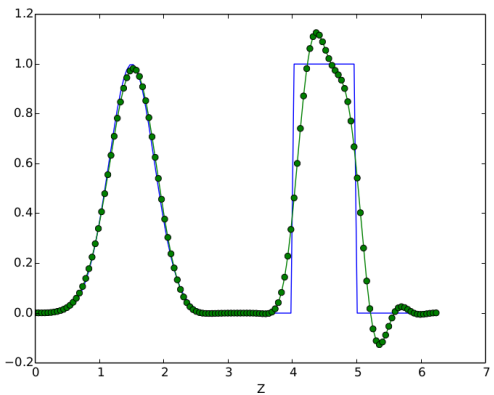
PVODE integrator, absolute tolerance 1e-12, relative tolerance 1e-5
CFL condition limits explicit methods to 128 steps, or **512 evaluations for RK4**
The RK3-SSP method requires dt < 0.2 dt(CFL), or 1920 evaluations

PVODE: 2904 evals



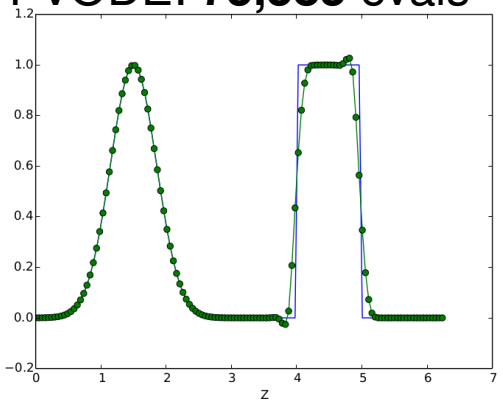
Central differencing with dissipation

PVODE: 3073 evals



Fromm (2nd order)

PVODE: **75,585** evals



XPPM (nonlinear limiter)

Implicit – Explicit (IMEX) schemes

- Allow separation of problem into stiff (timestep limiting) and non-stiff parts

- Treat parts of the problem implicitly, parts explicitly
- Simplifies implicit part, perhaps making preconditioning easier
- Explicit part no longer in GMRES loop
- Can use limiter schemes, strong nonlinearities which stall implicit solvers

- Some work on this already in BOUT++ using **IMEX-BDF2** scheme and coupling to **ARKODE** (Sundials). Matrix coloring (using PETSc) works very well in some cases

→ Works well for “toy” problems, too fragile for “real” problems

See **examples/IMEX/**

Jacobian coloring

- Usually matrix-free methods are used, to avoid having to calculate or store $\mathbb{J}$
- Knowing $\mathbb{J}$ enables many off-the-shelf preconditioners to be used
- $\mathbb{J}$ can be calculated using finite differences, but brute-force is very slow
- $\mathbb{J}$ must be sparse → potential must be solved as a constraint

Jacobian coloring (PETSc)

- Specify where the non-zero elements of $\mathbb{J}$ are
- If in doubt, include more non-zero elements
- Current system assumes 5-point stencil coupling all evolving fields

See **examples/IMEX/diffusion-nl** and **drift-wave-constraint**

Summary of time stepping

- For models where the shear Alfven wave is the limiting timescale, simple preconditioners can be very effective.
- Most models of interest have fast dispersive waves, requiring 3D solves.
- Several schemes have been tried, but beating Sundials' JFNK (without preconditioning) has proved hard for real problems.

Outline

- **Overall scaling**
- **Low level optimisation**
  - Loop vectorisation, OpenMP parallelisation
  - Inner vs Outer loops
- **Time integration**
  - Physics based preconditioning
  - Implicit-Explicit methods (IMEX)
  - Jacobian Coloring + AMG
- **Discussion**

Discussion issues

- **Avoid or improve 3D elliptic solves**
  - A bottleneck to scaling is global ϕ solves
  - Preconditioning of dispersive high frequency waves
- **Jacobian coloring strategies**
  - Handling interpolations, FFTs?
- **Improve cache, threading efficiency**
  - Better domain / threading decomposition
  - OpenMP tips/tricks