Parallel gyrokinetic simulations with Python

Emily Bourne[1] Yaman Güçlü[2]

[1]Technische Universität München, Germany

[2]Max-Planck-Institut für Plasmaphysik, Garching, Germany

25th October 2018

Motivation

Scientific Computing requirements:

- Fast algorithm prototyping
- Flexible
- Interactive
- Single-core optimization
- Shared-memory and MPI parallelization
- Strict quality control

Strategy

- Code is written in Python
- Bottlenecks are translated automatically to Fortran using pyccel

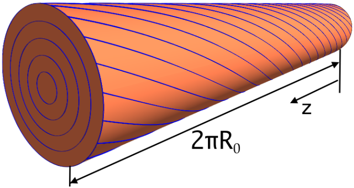
Thanks

Thanks to Ahmed Ratnani and Saïd Hadjout for their work on pyccel.

Outline

1. Screw-Pinch Simulation
2. Parallelisation Method
3. Acceleration with Pyccel
4. Results

Screw-Pinch Simulation[1]



$$\partial_t f + \{\phi, f\} + v_\parallel \vec{\nabla}_\parallel f - \vec{\nabla}_\parallel \phi \, \partial_{v_\parallel} f = 0$$

$$\{\phi, f\} = -\frac{\partial_\theta \phi}{r B_0} \partial_r f + \frac{\partial_r \phi}{r B_0} \partial_\theta f$$

$$-\left[\partial_r^2 \phi + \left(\frac{1}{r} + \frac{\partial_r n_0}{n_0}\right) \partial_r \phi + \frac{1}{r^2} \partial_\theta^2 \phi\right] + \frac{1}{T_e} \phi = \frac{1}{n_0} \int_{-\infty}^{+\infty} (f - f_{eq}) dv_\parallel$$

[1] G. Latu, M. Mehrenberger, Y. Güçlü, M. Ottaviani, E. Sonnendrücker, "Field-aligned interpolation for semi-lagrangian gyrokinetic simulations" Journal of Scientific Computing, vol. 74, pp. 1601–1650, March 2018

Advection Operators

$$\boxed{\partial_t f + \{\phi, f\} + v_\parallel \vec{\nabla}_\parallel f - \vec{\nabla}_\parallel \phi \, \partial_{v_\parallel} f = 0}$$

$$\{\phi, f\} = -\frac{\partial_\theta \phi}{r B_0} \partial_r f + \frac{\partial_r \phi}{r B_0} \partial_\theta f$$

- Lie and Strang splitting are used for the predictor and corrector steps

Advection on poloidal plane:	$\partial_t f + \{\phi, f\} = 0 \quad (1)$
Advection on flux surface:	$\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0 \quad (2)$
V-parallel advection:	$\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0 \quad (3)$

Advection Operators

Advection on poloidal plane: $\partial_t f + \{\phi, f\} = 0$

- Semi-lagrangian method
- Explicit second order Euler determines trajectory

Advection on flux surface: $\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0$

- Constant velocity semi-lagrangian method

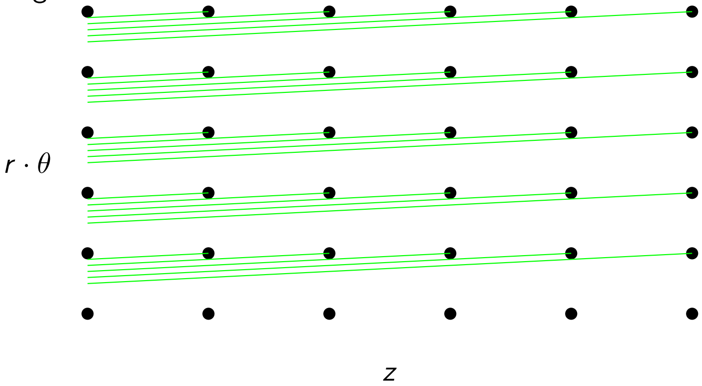
V-parallel advection: $\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0$

- Constant velocity semi-lagrangian method
- $\nabla_\parallel \phi$ determined using 6th order finite differences

Advection on flux surface

$$\partial_t f + v_{\|} \cdot \nabla_{\|} f = 0 \tag{4}$$

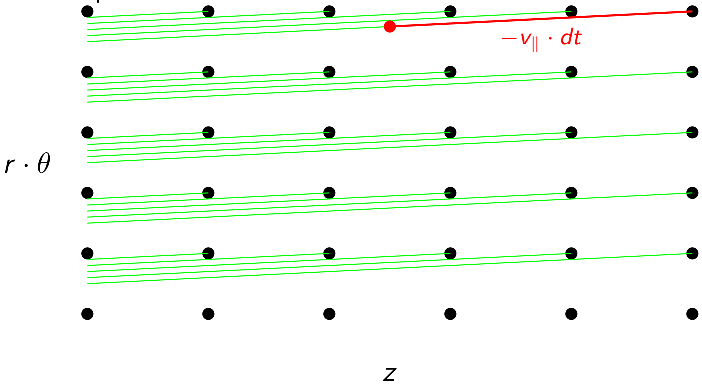
Magnetic Field lines:



Advection on flux surface

$$\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0 \tag{4}$$

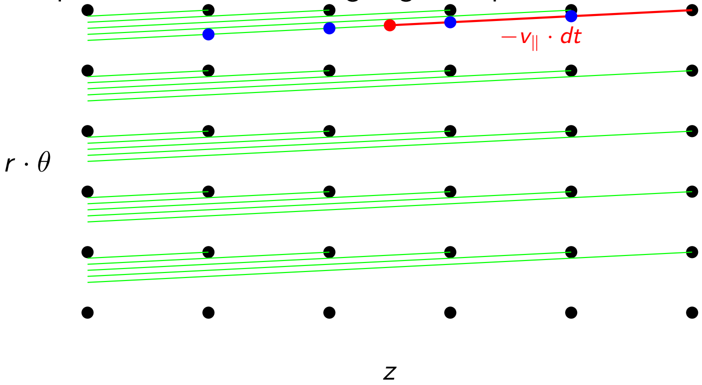
Retrace path:



Advection on flux surface

$$\partial_t f + v_{\parallel} \cdot \nabla_{\parallel} f = 0 \tag{4}$$

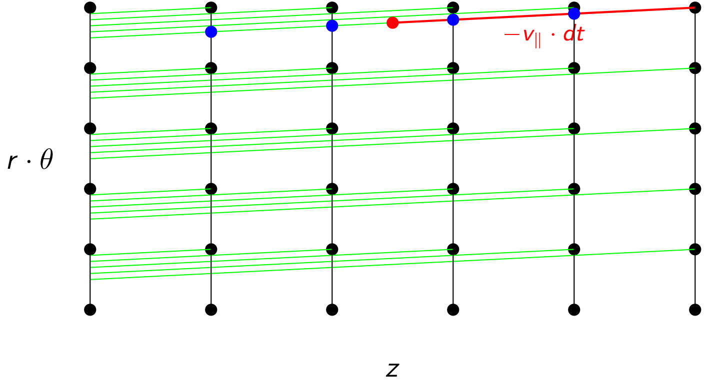
Find points either side for Lagrange interpolation:



Advection on flux surface

$$\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0 \tag{4}$$

Use θ-spline to find values:



Quasi-Neutrality Equation

$$-\left[\partial_r^2\phi+\left(\frac{1}{r}+\frac{\partial_r n_0}{n_0}\right)\partial_r\phi+\frac{1}{r^2}\partial_\theta^2\phi\right]+\frac{1}{T_e}\phi=\frac{1}{n_0}\int_{-\infty}^{+\infty}(f-f_{eq})dv_{\|}$$

- Theta component is handled using Fourier transforms
- Poisson equation solved using Finite Elements on a b-spline basis

Screw-Pinch Simulation - Method summary

1. Compute ϕ from f^n by solving the quasi-neutrality equation
2. Compute $f^{n+\frac{1}{2}}$ from f^n using Lie splitting
3. Compute ϕ from $f^{n+\frac{1}{2}}$ by solving the quasi-neutrality equation again
4. Compute f^{n+1} from f^n using Strang splitting

Code building blocks

- 2D Quasi-Neutrality solver
- 1D/2D Spline interpolators
- Advection operator on poloidal plane
- Advection operator on flux surface
- V parallel advection operator
- Parallel management

Will be translated to fortran using pyccel

Parallelisation Strategy - $(r, \theta, z, v_\parallel)$

Advection on flux surface: $\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0$

$(r, \theta, z, v_\parallel)$

V-parallel Advection: $\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0$

$(r, \theta, z, v_\parallel)$

Advection on poloidal plane: $\partial_t f + \{\phi, f\} = 0$

$(r, \theta, z, v_\parallel)$

Parallelisation Strategy - $(r, \theta, z, v_\parallel)$

Advection on flux surface: $\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0$

$(r , \boxed{\theta} , \boxed{z} , v_\parallel) \longrightarrow (r , v_\parallel , \boxed{\theta} , \boxed{z})$

V-parallel Advection: $\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0$

$(r , \theta , z , \boxed{v_\parallel}) \longrightarrow (r , z , \theta , \boxed{v_\parallel})$

Advection on poloidal plane: $\partial_t f + \{\phi, f\} = 0$

$(\boxed{r} , \boxed{\theta} , z , v_\parallel) \longrightarrow (v_\parallel , z , \boxed{\theta} , \boxed{r})$

Parallelisation Strategy - $(r, \theta, z, v_\parallel)$

Advection on flux surface: $\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0$

$(r , \theta , z , v_\parallel) \longrightarrow (r , v_\parallel , \theta , z)$

$n_r \cdot n_{v_\parallel}$

V-parallel Advection: $\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0$

$(r , \theta , z , v_\parallel) \longrightarrow (r , z , \theta , v_\parallel)$

$n_r \cdot n_\theta \cdot n_z$

Advection on poloidal plane: $\partial_t f + \{\phi, f\} = 0$

$(r , \theta , z , v_\parallel) \longrightarrow (v_\parallel , z , \theta , r)$

$n_z \cdot n_{v_\parallel}$

Parallelisation Strategy - $(r, \theta, z, v_\parallel)$

Advection on flux surface: $\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0$

$$(r , \theta , z , v_\parallel) \longrightarrow (r , v_\parallel , \theta , z)$$

$n_r \cdot n_{v_\parallel}$

V-parallel Advection: $\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0$

$$(r , \theta , z , v_\parallel) \longrightarrow (r , z , \theta , v_\parallel)$$

$n_r \cdot n_\theta \cdot n_z$

Advection on poloidal plane: $\partial_t f + \{\phi, f\} = 0$

$$(r , \theta , z , v_\parallel) \longrightarrow (v_\parallel , z , \theta , r)$$

$n_z \cdot n_{v_\parallel}$

Parallelisation Strategy - $(r, \theta, z, v_\parallel)$

Advection on flux surface: $\partial_t f + v_\parallel \cdot \nabla_\parallel f = 0$

$(r , \theta , z , v_\parallel) \longrightarrow (r , v_\parallel , \theta , z)$

$n_r \cdot n_{v_\parallel}$

V-parallel Advection: $\partial_t f + \nabla_\parallel \phi \cdot \partial_{v_\parallel} f = 0$

$(r , \theta , z , v_\parallel) \longrightarrow (r , z , \theta , v_\parallel)$

$n_r \cdot n_\theta \cdot n_z$

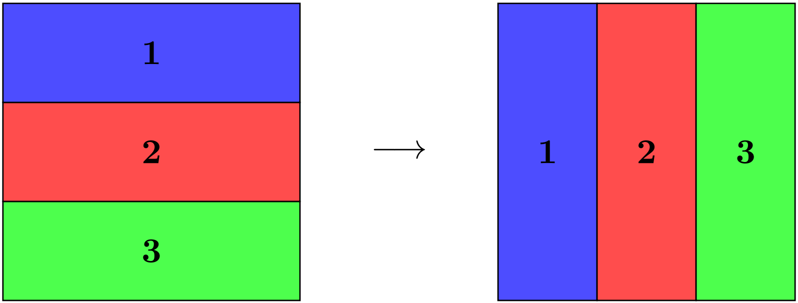
Advection on poloidal plane: $\partial_t f + \{\phi, f\} = 0$

$(r , \theta , z , v_\parallel) \longrightarrow (v_\parallel , z , \theta , r)$

$n_z \cdot n_{v_\parallel}$

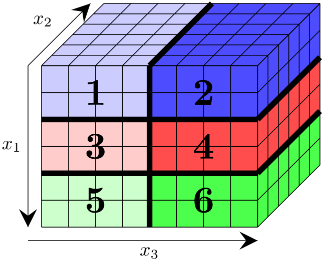
Parallelisation Strategy - Basic idea

All MPI commands in layout changes can be represented as a 2D transpose operation



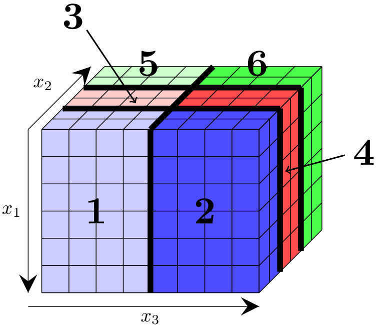
Parallelisation Strategy - 3D example

Decomposition in Layout A:



(1,3,5) independent of (2,4,6)

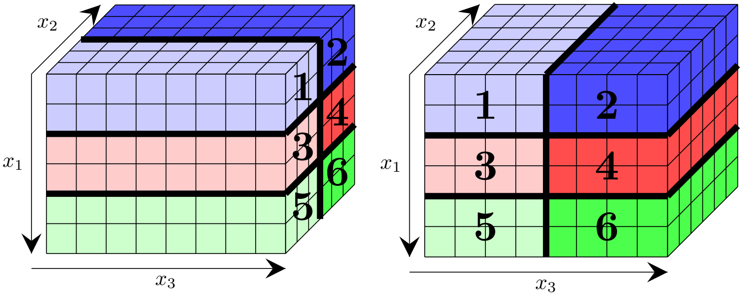
Decomposition in Layout B:



Parallelisation Strategy - 3D example

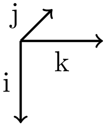
Decomposition in Layout C:

Decomposition in Layout A:

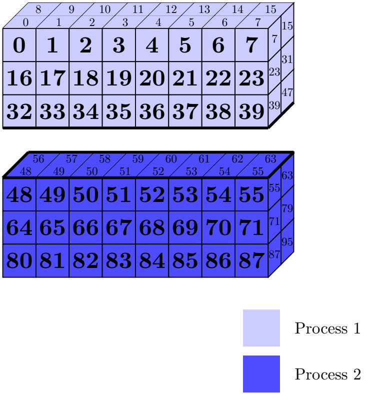


(1,2) independent of (3,4) and (5,6)

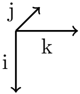
Parallelisation Strategy



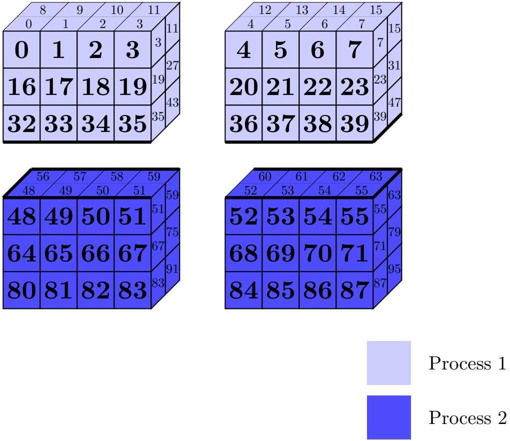
- Layout in memory
- Numbers indicate contiguous locations in memory
- A correct layout change preserves ordering
- C-ordering $A[i,j,k]$



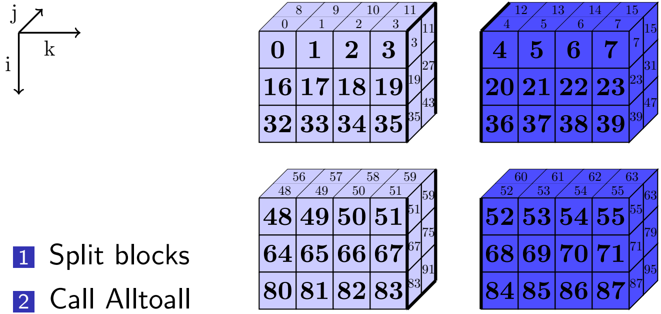
Parallelisation Strategy



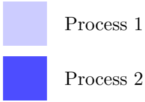
1. Split blocks



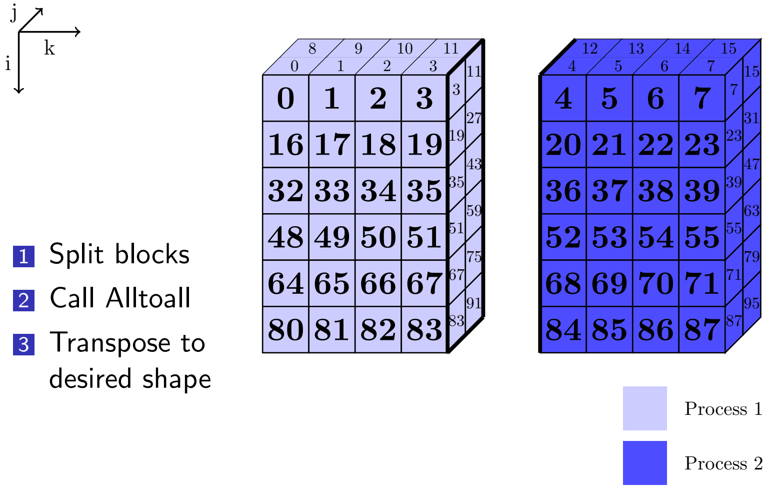
Parallelisation Strategy



1. Split blocks
2. Call Alltoall

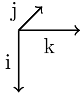


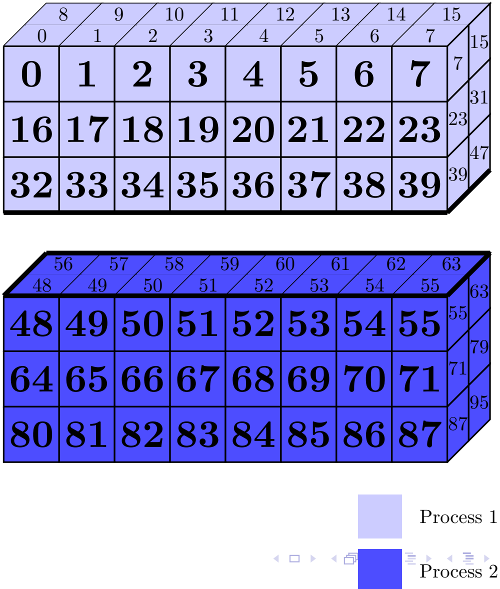
Parallelisation Strategy



1. Split blocks
2. Call Alltoall
3. Transpose to desired shape

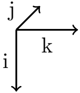
Parallelisation Strategy



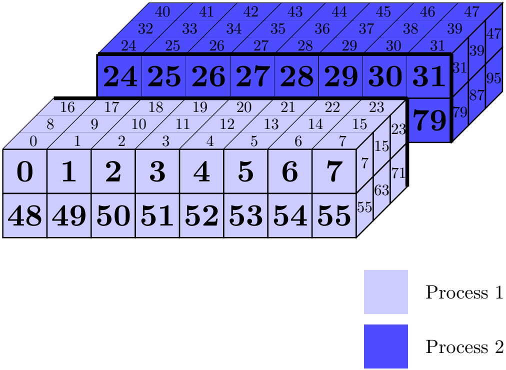


1. Split blocks
2. Call Alltoall
3. Transpose to desired shape

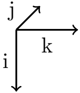
Parallelisation Strategy



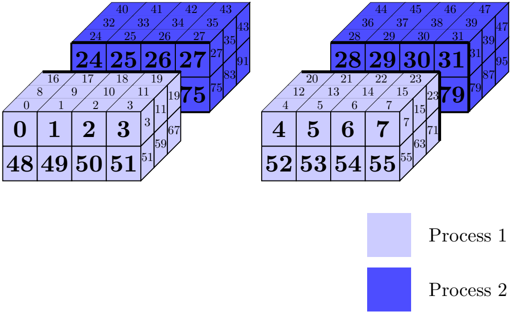
1. Split blocks
2. Call Alltoall
3. Transpose to desired shape



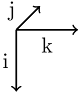
Parallelisation Strategy



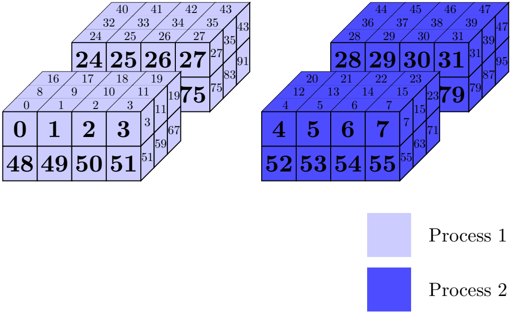
1. Split blocks
2. Call Alltoall
3. Transpose to desired shape



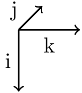
Parallelisation Strategy



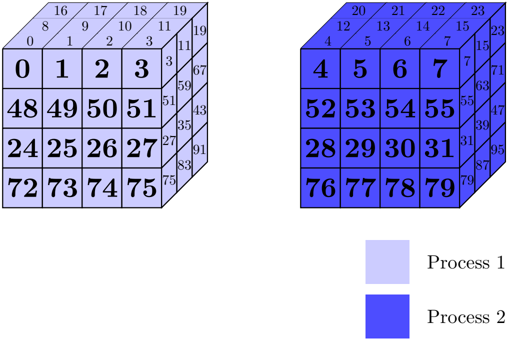
1. Split blocks
2. Call Alltoall
3. Transpose to desired shape



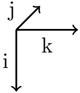
Parallelisation Strategy



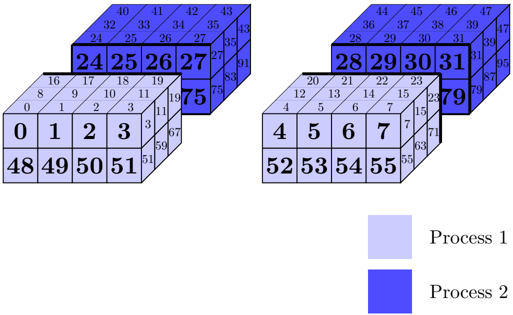
1. Split blocks
2. Call Alltoall
3. Transpose to desired shape



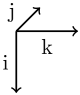
Parallelisation Strategy



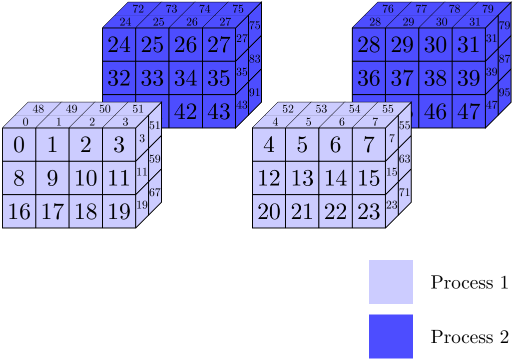
1. Split blocks
2. Call Alltoall
3. Transpose to desired shape



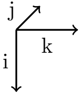
Parallelisation Strategy



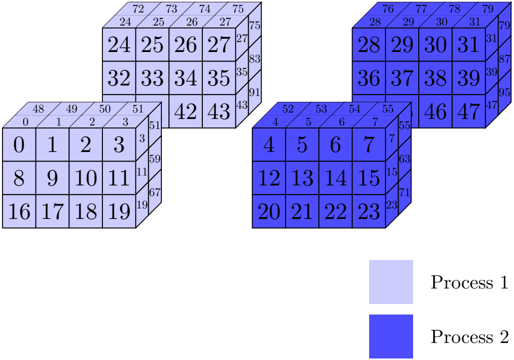
1. Split blocks
2. Transpose blocks so concatenate direction is first axis
3. Call Alltoall
4. Transpose to desired shape



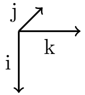
Parallelisation Strategy



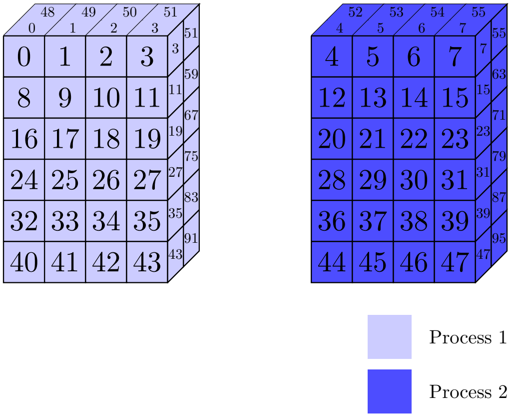
1. Split blocks
2. Transpose blocks so concatenate direction is first axis
3. Call Alltoall
4. Transpose to desired shape



Parallelisation Strategy



1. Split blocks
2. Transpose blocks so concatenate direction is first axis
3. Call Alltoall
4. Transpose to desired shape



Profiling before Pyccel

Function	Total time excluding sub functions [s]	Number of calls	Time per call [s]	Total time [s]
method 'Alltoall' from mpi4py	28.193	250	0.113	28.193
numpy.core.multiarray.array	16.070	5689347	0.000	16.070
bisplev from scipy	10.009	1006666	0.000	37.819
method scipy.interpolate._fitpack._bispev	8.503	1006666	0.000	8.503
atleast_1d from numpy	7.990	2915166	0.000	23.820
splev	7.334	901833	0.000	18.883
reshape from numpy	6.590	3397927	0.000	6.590

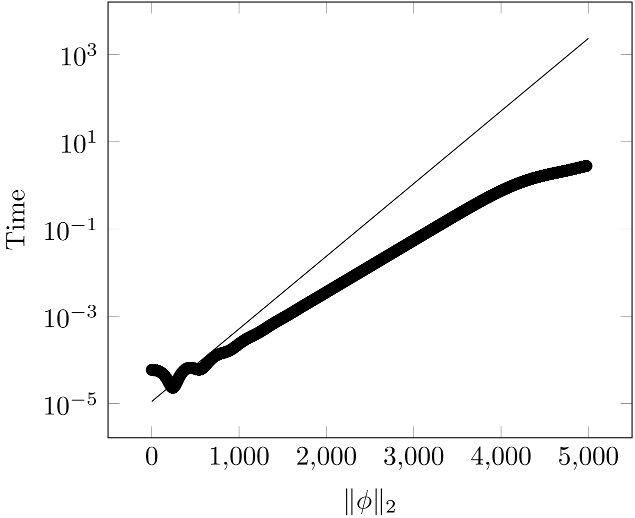
Table: The results of profiling the pure python implementation

Profiling after Pyccel

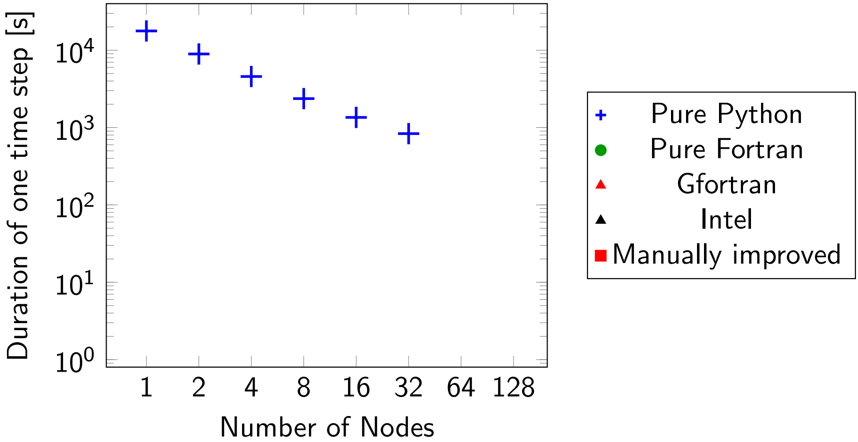
Function	Total time excluding sub functions [s]	Number of calls	Time per call [s]	Total time [s]
method 'Alltoall' from mpi4py	2.735	250	0.011	2.735
step in FluxSurfaceAdvection	1.826	5000	0.000	3.355
parallel_gradient in ParallelGradient	1.492	1000	0.001	1.903
step in PoloidalAdvection object	1.420	3333	0.000	3.004
method 'solve' from scipy 'SuperLU'	1.294	96666	0.000	1.294
getPerturbedRho in ParallelGradient	1.022	101	0.010	2.307
_solve_system_nonperiodic in SplineInterpolator1D	0.977	123700	0.000	0.977

Table: The results of profiling the implementation after pyccelisation

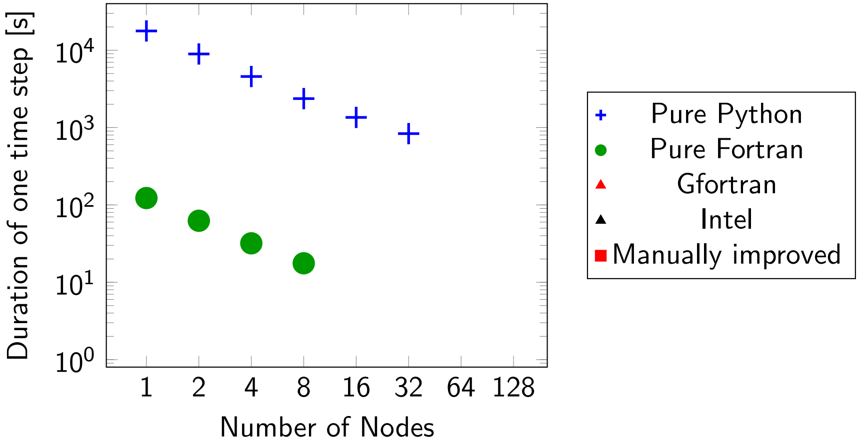
Ion Temperature Gradient Test



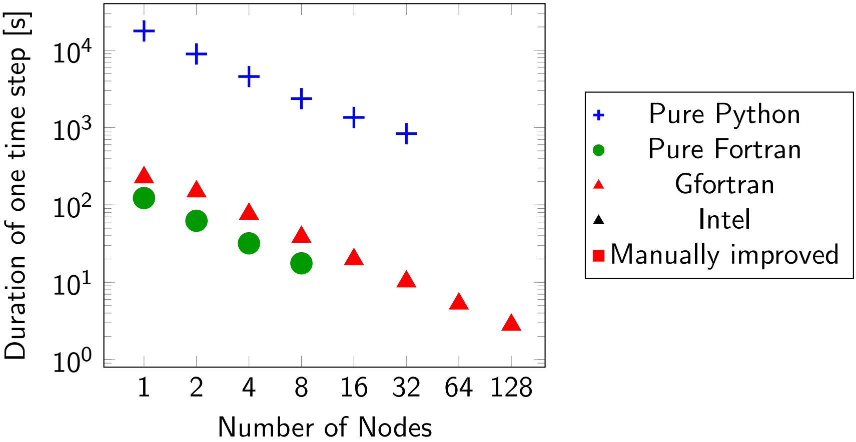
Performance Results



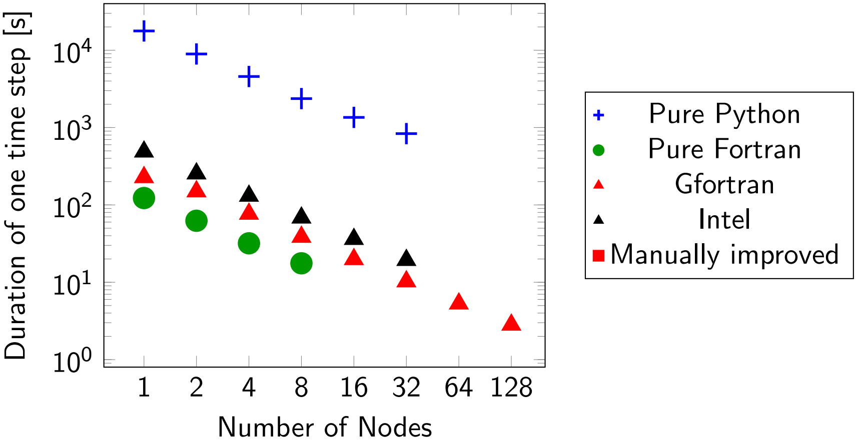
Performance Results



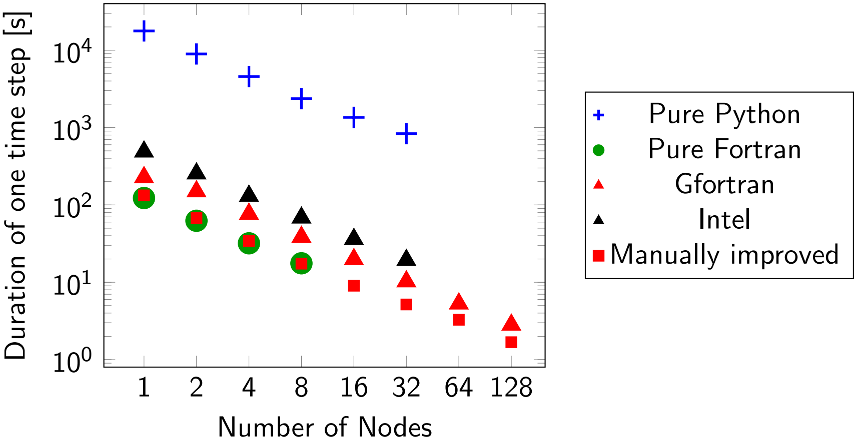
Performance Results



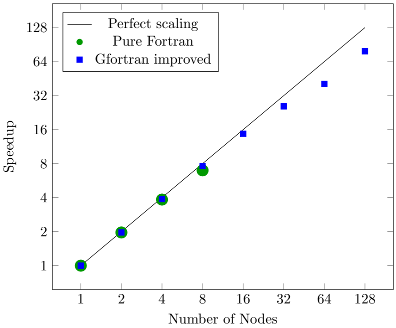
Performance Results

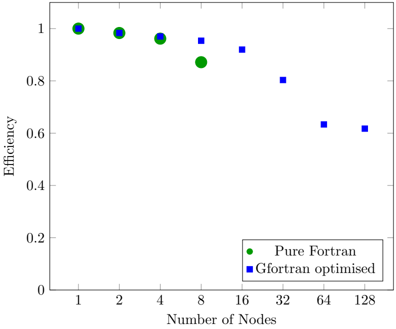


Performance Results



Scaling


Perfect scaling
Pure Fortran
Gfortran improved
Speedup
128
64
32
16
8
4
2
1
1
2
4
8
16
32
64
128
Number of Nodes


Efficiency
1
0.8
0.6
0.4
0.2
0
1
2
4
8
16
32
64
128
Number of Nodes
Pure Fortran
Gfortran optimised

Conclusion

Summary:

- Screw-Pinch Simulation can be effectively written with python
- Parallel method has very good scalability
- Code translated with pyccel has shorter development time and comparable run-time compared to fortran code

Conclusion

Summary:

- Screw-Pinch Simulation can be effectively written with python
- Parallel method has very good scalability
- Code translated with pyccel has shorter development time and comparable run-time compared to fortran code

Future Work:

- Extend model to include centre
- Tokamak geometry

Conclusion

Summary:

- Screw-Pinch Simulation can be effectively written with python
- Parallel method has very good scalability
- Code translated with pyccel has shorter development time and comparable run-time compared to fortran code

Future Work:

- Extend model to include centre
- Tokamak geometry

Thank you for listening. Any questions?