EUROfusion

Multigrid solver by using PETSc

Kab Seok Kang kskang@ipp.mpg.de

High Level Support Team (HLST)
Department of Computational Plasma Physics
Max-Planck-Institut für Plasmaphysik,
Boltzmannstraße 2, D-85748 Garching, Germany

December 7, 2018

HLST

IPP Max-Planck-Institut für Plasmaphysik

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

OUTLINE

1. BOUT++
2. Laplace Solvers
3. PETSc Library
4. Using PETSc libraries in BOUT++
5. Numerical results
6. Summary

Purpose of AMGBOUT

- Use or develope algebraic multigrid solver in BOUT++
- Use existence AMG in PETSc
- Find optimal options for 2D Laplace solver
- Prepare AMG solver for 3D problem in BOUT++

BOUT++ Overview

What BOUT++ is:

- A toolbox for solving PDEs on parallel computers. Aims to reduce duplication of effort, and allow quick development and testing of new models
- Focused on flute-reduced plasma models in field-aligned coordinate systems, though has more general capabilities

A toolbox for plasma simulations

- Collection of useful data types and associated routines.
  Occupies a middle ground between problem-specific codes and general libraries (PETSc, Trilinos, Overture, Chombo,...)
- Researchers can make use of a well tested library of simulation code and input/output tools
- Greatly reduces the time needed to develop a new simuation

Key features:

- Finite difference initial value code in 3D
- Implicit or explicit time integration methods
- Coordinate system set in metric tensor components
- Handles topology of multiple X-points
- Written in C++, quite modular design
- Open Source (LGPL), available on `github`

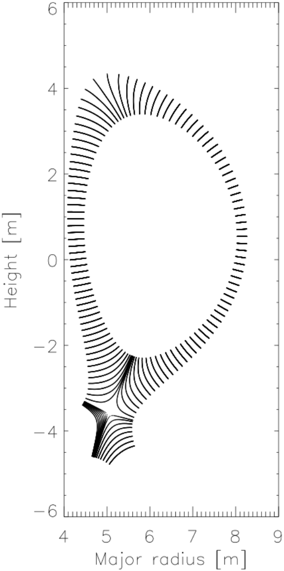
Coordinate system

- Logically rectangular. Defined by metric tensor
- Field aligned: x (radial), y (parallel), z (toroidal)

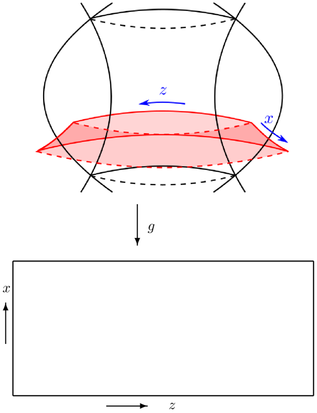
$$x = \psi - \psi_0; \quad y = \theta; \quad z = \phi - \int_{\theta_0}^{\theta} \frac{B_\phi}{RB_\theta} dl_\theta$$

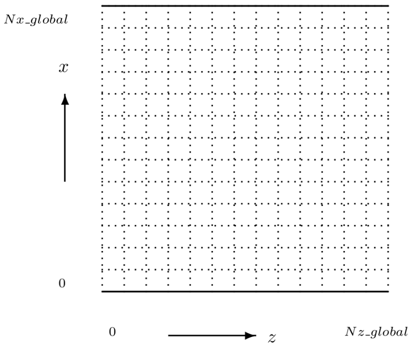
(The y unit vector $\hat{e}_y$ is along the magnetic field)
→ Has a singularity at the X-point

- A branch-cut, hole at the X-point itself
- Flux-Coordinate-Independent (FCI) scheme : Construct grid poloidal planes. Field-align parallel derivatives, but not grid. Under active development in BOI
- Varios time solvers: Choice of fully-implicit or fully-explicit methods available
  Sundials CVODE, PVODE, RK4, PETSc,
  RK3SSP, Sundials IDA Karniadakis, POWER, Euler
  $-$User-supplied preconditioner possible for implicit solvers: CVODE, PETSc,
  Physics-based preconditioner can improve performance



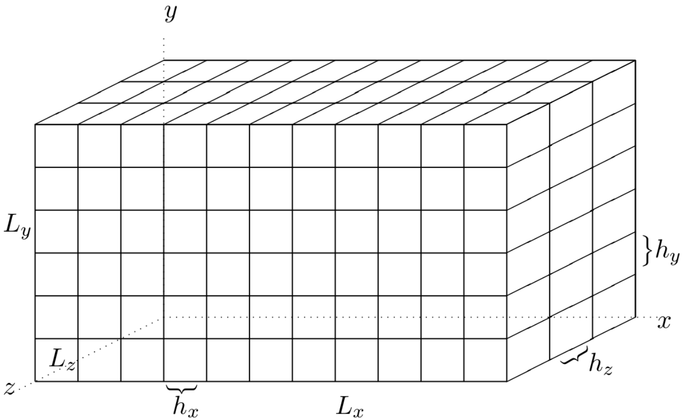
Computational domain





- 2D($x - z$)-1D (y) decoupling: Current implementation
- $nx \times nz$ on ny planes.
- Use *NPEX* MPI tasks through x-direction for $nx \times nz$
- Use *NPEY* MPI tasks through y-direction

Computational domain for 3D problem



– *z*-direction: Periodic boundary condition

– *x* and *y* direction: Periodic, Neumann, or Dirichlet BD condition

Laplace solvers

∗ Solve the equation

$$d\nabla_\perp^2 f + \frac{1}{c_1}\nabla c_2 \cdot \nabla_\perp f + af = b$$

− Inverted for the potential f where d, c_1, c_2 , a are constant.
∗ With conformal mapping g on the reference domain

$$\begin{aligned}\nabla_\perp^2 f = {} & G_1\frac{\partial f}{\partial x} + \left(G_2 - \frac{1}{J}\frac{\partial}{\partial y}\left(\frac{J}{g_{22}}\right)\right)\frac{\partial f}{\partial y} + G_3\frac{\partial f}{\partial z} + g^{11}\frac{\partial^2}{\partial x^2} \\ & + \left(g^{22} - \frac{1}{g^{22}}\right)\frac{\partial^2 f}{\partial y^2} + g^{33}\frac{\partial^2 f}{\partial z^2} + 2g^{12}\frac{\partial^2}{\partial x\partial y} + 2g^{13}\frac{\partial^2 f}{\partial x\partial z} + 2g^{23}\frac{\partial^2 f}{\partial y\partial z} \\ \nabla c \cdot \nabla_\perp f = {} & \left(g^{11}\frac{\partial c}{\partial x} + g^{12}\frac{\partial c}{\partial y} + g^{13}\frac{\partial c}{\partial z}\right)\frac{\partial f}{\partial x} + \left(g^{12}\frac{\partial c}{\partial x} + \left(g^{22} - \frac{1}{g^{22}}\right)\frac{\partial c}{\partial y} + g^{23}\frac{\partial c}{\partial z}\right)\frac{\partial f}{\partial y} \\ & + \left(g^{13}\frac{\partial c}{\partial x} + g^{23}\frac{\partial c}{\partial y} + g^{33}\frac{\partial c}{\partial z}\right)\frac{\partial f}{\partial z}.\end{aligned}$$

∗ poloidal fileds are perpendicular with toroidal field → In red can be neglected for 2D-1D formulation

2D-1D approach vs. 3D solver

- 2D-1D approach: Fit when g^{12}, g^{23} are small

- Limitations

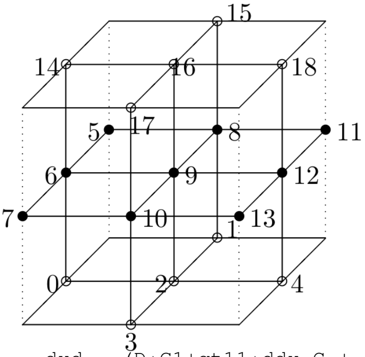
– Cannot implement parallel boundary conditions
→ The sheath boundary conditions on the potential (critical to the correct description of the SOL) cannot be applied

– The integrability conditions restrict the perpendicular boundary conditions that can be applied
→ Do not allow all-Neumann boundary conditions

– Impose spurious constraints on the parallel variation of the solution
→ Fail to be s good approximation to the solution of the 3D problem, introducing unphysical behaviour.

∗ So we need the 3D solver
⇐ Need to handle boundary conditions

Discretization



```
V = dhx*dhy*dhz,
gt11 = g11, gt12 = g12,
gt13 = g13, gt22 = g22 - 1.0/g22
gt23 = g23, gt33 = g33
ddJ = (J/g22 - J/g22)/2./dhy/J
ddx_C = (C2(i2+1)-C2(i2-1))/2./dhx/C1
ddy_C = (C2(k2+1)-C2(k2-1))/2./dhy/C1
ddz_C = (C2(j2+1)-C2(j2-1))/2./dhz/C1
ddx = D*gt11/dhx/dhx,
ddy = D*gt22/dhy/dhy,
ddz = D*gt33/dhz/dhz
dxdy = 2.0*D*gt12/dhx/dhy
dxdz = 2.0*D*gt13/dhx/dhz
dydz = 2.0*D*gt23/dhy/dhz
dxd = (D*G1+gt11*ddx_C + gt12*ddy_C + gt13*ddz_C)/dhx
dyd = (D*(G2-ddJ)+gt12*ddx_C + gt22*ddy_C + gt23*ddz_C)/dhy
dzd = (D*G3+gt33*ddz_C +gt13*ddx_C + gt23*ddy_C)/dhz
```

M0 = dydz*V/4.0, M1 = dxdy*V/4.0, M2 = (ddy + dyd/2.0)*V,
M3 = -dxdy*V/4.0, M4 = -dydz*V/4.0
M5 = dxdz*V/4.0, M6 = (ddx - dxd/2.0)*V, M7 = -dxdz*V/4.0,
M8 = (ddz - dzd/2.0)*V, M9 = (A - 2.0*(ddx+ddy+ddz))*V, M10 = (ddz + dzd/2.0)*V
M11 = -dxdz*V/4.0, M12 = (ddx+dxd/2.0)*V, M13 = dxdz*V/4.0
M14 = -dydz*V/4.0, M15 = -dxdy*V/4.0, M16 = (ddy+dyd/2.0)*V,
M17 = dxdy*V/4.0, M18 = dydz*V/4.0

Parallelization and boundary conditions

– Divide Rectangular shape with Guarde cell
- *x* and *y* directional parallelization
$\rightarrow$ Easily extentable for *z* directional parallelization

– Use Index set: Standard in PETSc.
- Can define a differnt shape fo domain by imposing the connectivity
- Don't need guad cell for periodic boundary condition

Overview of the PETSc

∗ The Portable, Extensible Toolkit for Scientific Computation (PETSc)

- Easy the development of large-scale scientific application codes in Fortran, C, C++, and Python
- A powerful set of tools for the numerical solution of partial differential equations and related problems on high performance computers
- Provides clean and effective codes for the various phases of solving PDEs, with a uniform approach for each class of problem.
  → Enables easy comparison and use of different algorithms
- Provides a rich environment for modeling scientific applications as well as for rapid algorithm design and prototyping.
- Enable easy customization and extension of both algorithms and implementations.
  → Promotes code reuse and flexibility, and separates the issues of parallelism from the choice of algorithms

PETSc modules

- index sets (IS), including permutations, for indexing into vectors, renumbering, etc;
- vectors (Vec);
- matrices (Mat) (generally sparse);
- over thirty Krylov subspace methods (KSP);
- dozens of preconditioners, including multigrid, block solvers, and sparse direct solvers (PC);
- managing interactions between mesh data structures and vectors and matrices (DM);
- nonlinear solvers (SNES);
- time steppers for solving time-dependent (nonlinear) PDEs, including support for differential algebraic equations, and the computation of adjoints (sensitivities/gradients of the solutions) (TS)

Solvers for linear system

$\ast$ Krylov Subspace Methods (KSP)
- Need the operators (Mat), vectors (Vec), and preconditioners (PC)
- Support various iterative solvers including direct solver

$\ast$ Algebraic multigrid preconditioner for GMRES
- GAMG: PETSc's native AMG framework
→ GAMG: parallel sparce matrices with the AIJ format

- Two 3rd party solvers: hypre(BoomerAMG) and Trililos(ML)
← Need configure before compilation PETSc
• Add `-download-hyper` for hypre and `-download-ml` for ML

- Framework for multigrid method (MG) for AMG and GMG

Configures and selecting solver

- Configure option: `$ ./configure -with-petsc`
- Include header files: `<petscksp.h>`

→ call `<petscpc.h>, <petscmat.h>, <petscvec.h>`

– For direct solver: ksptype = preonly   pctype = pclu

`KSPSetType(ksp,KSPREONLY)      PCSetType(pc,PCLU)`

– For GMRES with BJacobi: ksptype = kspgmres   pctype = bjacobi

`KSPSetType(ksp,KSPGMRES)     PCSetType(pc,PCBJACOBI)`

– For GAMG: `PCSetType(pc,PCGAMG)`

∗ Use Aggregation options with `PCGAMGSetType(pc,PCGAMGAGG)`

- Set the number of Smooth Aggregation: 1 (default)

  or `PCGAMGSetNSmooths(pc,2(or 4))`

– ML in Trillios: `PCSetType(pc,PCML)`

– BoomerAMG in HYPRE: `PCSetType(pc,PCHYPRE)`

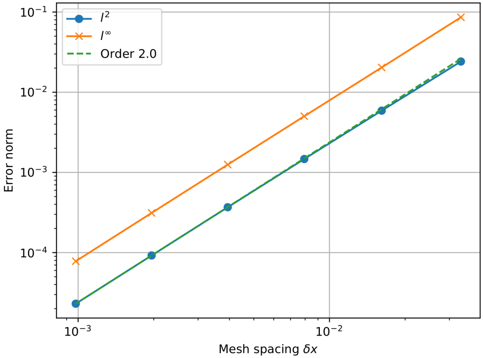
  and `PCHYPRESetType(pc,"boomeramg")`

Setting options in BOUT++ in BOUT.inp

```
myg = 1
mxg = 1
solution = f(x,y,z); input = f(x,y,z)
[mesh]
symmetricGlobalX = true
nx = 34 ; ny = 4; nz = nx-2*mxg
dx = 0.00363636363636364/(nx-2*mxg)
dy = 2.*pi/ny
dz = 2.*pi/nz/600
g11 = ; g22 = ; g33 = ; g12 = ; g13 = ; g23 = ; g_11 = ; g_22 = ;
g_33 = ; g_12 = ; g_13 = ; g_23 = ; J = ; Bxy = ; G1_11 = ; G1_22 = ;
G1_33 = ; G1_12 = ; G1_13 = ; G1_23 = ; G2_11 = ; G2_22 = ; G2_33 = ;
G2_12 = ; G2_13 = ; G2_23 = ; G3_11 = ; G3_22 = ; G3_33 = ; G3_12 = ;
G3_13 = ; G3_23 = ; G1 = ; G2 = ; G3 = ;
Lx = 0.00363636363636364
Lz = 0.0209439510239320
[laplace]
type = petscamg
flags = 0
[petscamg]
solvertype = hypre
multigridlevel = 6
```

Test problems and verification

- Using test cases: prepared by J. Omotani (CCFE)
- $nx = nz$, $ny = 16$ according to given $nx = 2^k$ for $k = 5, 6, \ldots, 15$
- Numerical error in l^2 and l^∞ for $k = 5, 6, \ldots, 10$



→ Error convergent factor is 2: typical for centered finite diff. method

→ Verified matrix generation and data conversion

Solution time on single core

– Tested several preconditioners:

- Some with default options fail: GAMG with Classical and Geometric
- Some options also fail. Might need more implementation

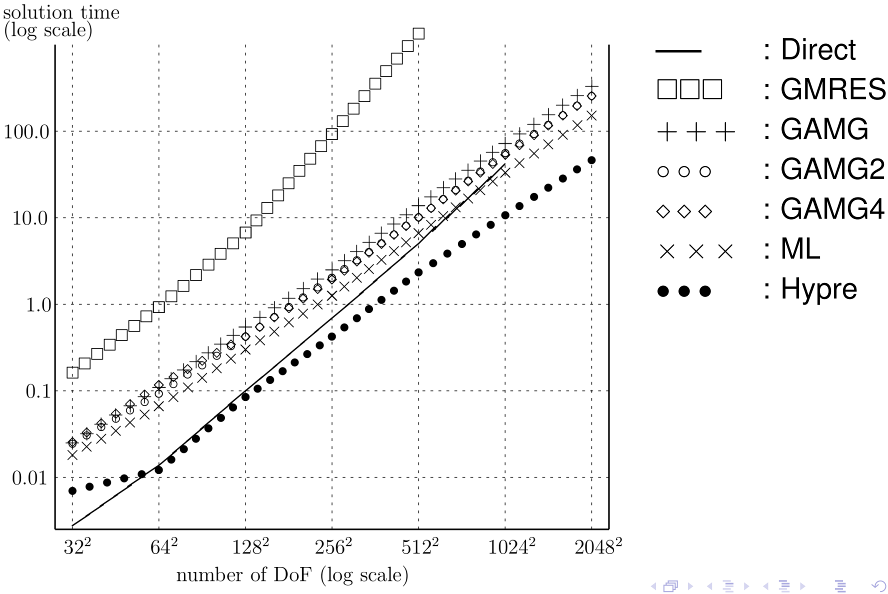
– GAMG : 1 aggregation smoothed GAMG (default options)

– GAMG2(4): 2 (4) aggregation smoother GAMG

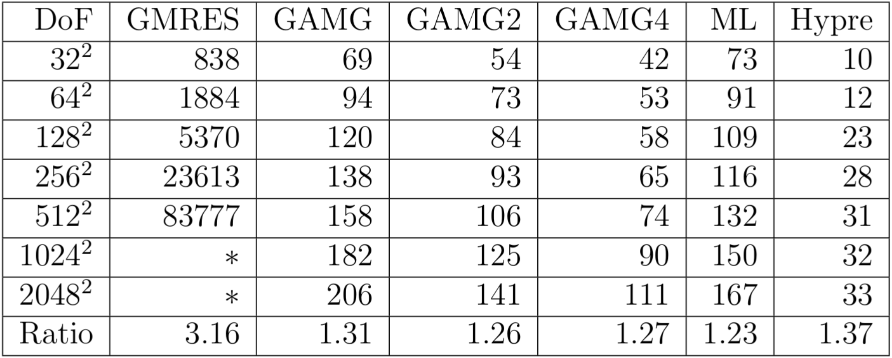
DoF	Direct	GMRES	GAMG	GAMG2	GAMG4	ML	Hypre
32^2	0.003	0.15	0.03	0.02	0.03	0.02	0.007
64^2	0.014	0.85	0.11	0.09	0.09	0.07	0.012
128^2	0.098	6.43	0.54	0.43	0.41	0.30	0.081
256^2	0.682	87.68	2.50	1.95	1.90	1.27	0.410
512^2	5.045	1275.77	14.01	10.20	10.16	6.69	2.337
1024^2	42.523	$*$	73.11	54.71	51.99	33.11	10.656
2048^2	$*$	$*$	329.75	251.34	251.49	147.66	45.155
Ratio	6.88	9.58	4.85	4.69	4.64	4.47	4.33

– BoomerAMG (Hypre) has the best performance on 1 core from 64^2 DoF and the ratio

Figure of solution time on single core



Required number of iteration on single core

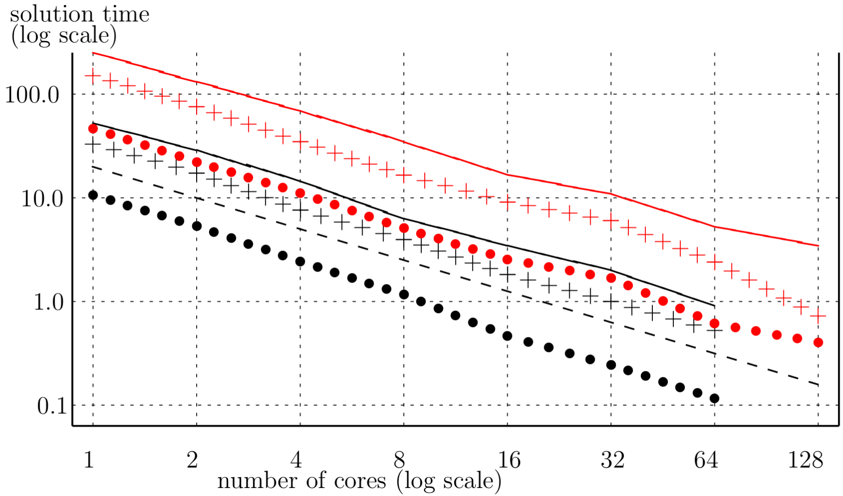


DoF	GMRES	GAMG	GAMG2	GAMG4	ML	Hypre
32^2	838	69	54	42	73	10
64^2	1884	94	73	53	91	12
128^2	5370	120	84	58	109	23
256^2	23613	138	93	65	116	28
512^2	83777	158	106	74	132	31
1024^2	$*$	182	125	90	150	32
2048^2	$*$	206	141	111	167	33
Ratio	3.16	1.31	1.26	1.27	1.23	1.37

– Required number of iterations of GMRES from 1024^2 are over the limit

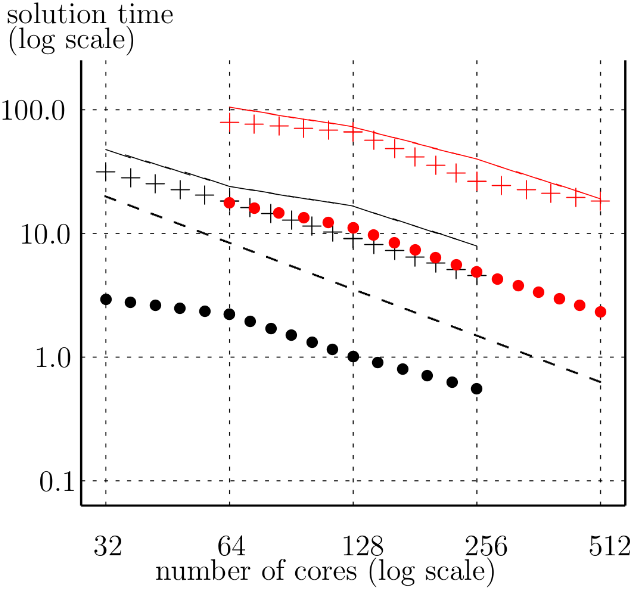
– Other preconditioners have a typical required number of iterations

Comparison preconditioners (Strong scaling 1)



- GAMG4(solid line), ML $(+++)$, BoomerAMG in Hypre $(\bullet\bullet\bullet)$
- 1024^2 DoF in black and 2048^2 DoF in red. - - - for Ideal

Comparison preconditioners (Strong scaling 2)



—— : GAMG4

+ + + : ML

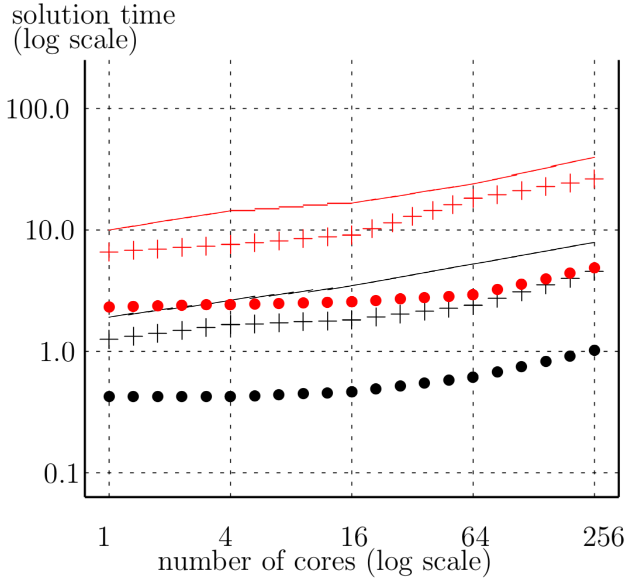
● ● ● : Hypre

- - - : Ideal

4096^2 DoF in black

8192^2 DoF in red

Comparison preconditioners (Weak scaling 1)



— : GAMG4

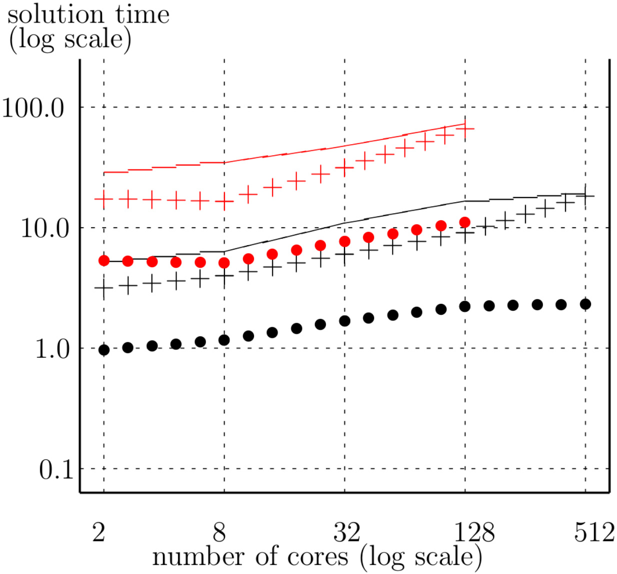
+ + + : ML

● ● ● : Hypre

256^2 DoF (in black) per core

512^2 DoF (in red) per core

Comparison preconditioners (Weak scaling 2)



—— : GAMG4

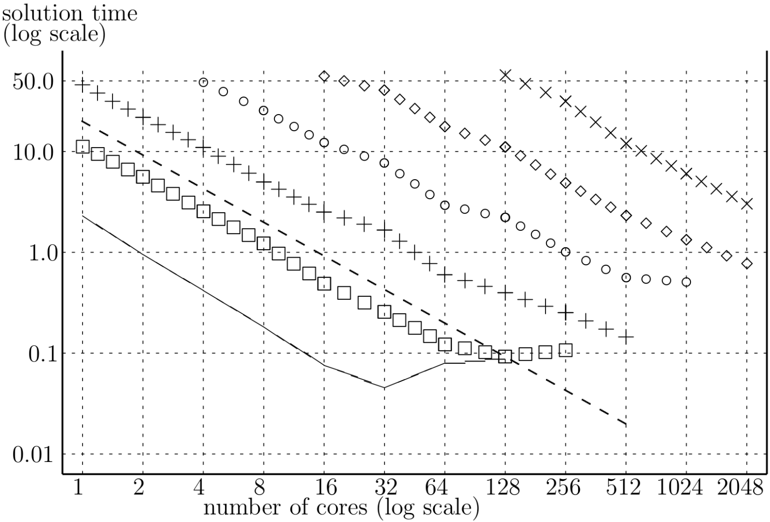
+ + + : ML

● ● ● : Hypre

$512^2/2$ DoF (in black) per core

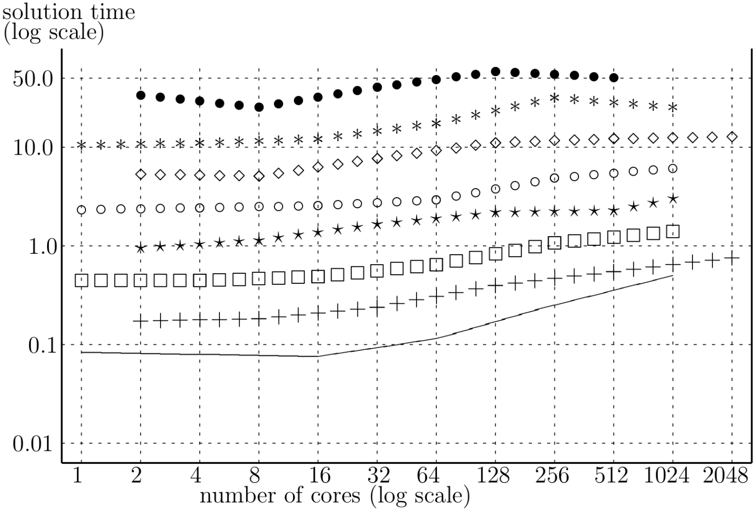
$1024^2/2$ DoF (in red) per core

Strong scaling of BoomerAMG



512^2(–), 1024^2(□), 2048^2($+$), 4096^2($\circ$), 8192^2($\diamond$), 8192^2($\times$), ideal (- -)

Weak scaling of BoomerAMG



2^{14}(–), 2^{15}(+), 2^{16}(□), 2^{17}(⋆), 2^{18}(○), 2^{19}(◇), 2^{20}(∗), 2^{21}(●) per MPI task

Current status and future work

- Set an algebraic multigrid branch in github of BOUT++
- Installed on the Marconi machine with PETSc with Hypre and ML
- Made verification of data conversion with discretization error for the test problems
- Investigate convergence behavior of several solvers:
  - Direct solver on single core, PGMRES with Block Jacobi, GAMG, ML, and BoomerAMG precoditioners
- BoomerAMG in Hypre is the best solvers and acceptable

- Developing 3D solvers including generating matrix