Changes in UTPROJ3D: A Three-Dimension Multiblock Mixed-Hybrid Finite Element Code

Gergina Pencheva and Ivan Yotov

1 Introduction

This report describes the changes made to UTPROJ3D in implementing the balancing domain decomposition preconditioner. The Balancing Domain Decomposition algorithm is described in [3], [1]. It involves solving local Neumann problems on each iteration. The numerical tests (see Section 5) indicate a substantial speedup in the convergence of the interface solves. In addition, an improved choice of initial guess for the local CG solvers reduces substantially the number of local iterations (see Section 4). Additional upgrades involve the use of variable Dirichlet BCs and permeability tensor K (see Section 4).

2 Changes related to balancing preconditioning

- Duplicate all structures and routines from Dirichlet problem for Neumann problem in files subdomain.C, subdomain.h, sdface.C, and sdface.h in directory utproj/proj. By convention, these have suffix N, e.g. MinvN, assembleOnElementN etc.

- In file utproj/proj/balPrecond.C (which contains most of the added routines) in InitBalance we construct coarse (balancing) matrix, symmetrize it, and do its Cholesky factorization (l_WPOTRF). If the factorization is successful and none of the pivot elements is too close to zero (10^{-12}), we set the flag balSlvType (globally defined in utproj/proj/masterdefs.h) equal to CholeskyType. Otherwise, we continue with eigenvalue/eigenvector decomposition (rs in file utproj/proj/rs.f) and set balSlvType equal to EigenType. Regardless of what is user's tolerance for local solvers, just for this routine use epsSDsolve = 10^{-16} when calculating the subdomain solutions. InitBalance is called just once at the beginning of interface-cg algorithm in cg.C.

- In routine pseudoInv (utproj/proj/balPrecond.C) we solve a coarse system. If balSlvType is CholeskyType we use the computed factors (l_WPOTRS). Otherwise, we find a solution in the form

$$x = \sum_i \frac{v_i^T b}{\lambda_i} v_i$$

  where the summation is only over eigenvalues λ_i that are large enough (cutoff = 10^{-10}), v_i are the corresponding eigenvectors and b is the RHS.

- The routine balance, depending on the flag izero, solves the coarse problem given by either equations (12)-(13) or (15)-(16) in [2].

- BalPC is the routine that gets r (unpreconditioned residual) and returns z (preconditioned residual) following the balancing preconditioning algorithm given in [1] (see also [3]).

- The routine swapbdry in file utproj/proj/sdface.C was fixed to handle the case avgtype = WEIGHT using a new routine divideface.
- The two conjugate gradient algorithms, conjGrad (inearOperatorConjGrad.C) for local problems and cg (cg.C) for the interface problem, require checking for convergence all residuals including the initial one, while the one in smoothSDF (cg.C) needs to take at least one step. Since all of them used the same member function "check" of class iterationController which didn't check the initial residual, we renamed it as "check_smooth" for use in smoothSDF and created another copy (named "check") which checks all residuals for use in the other two CG algorithms.
- In interface-cg algorithm change the stopping criteria by checking also the reduction of residual norm. We use epsSDFrel = $\max$(100*epsSDF, 10^{-7}). In addition, in conjGrad we check for convergence the absolute, not the relative tolerance.
- **NOTE:** The tolerance for the local solves need to be small enough, or otherwise the orthogonality between the RHS and the nullspace in local Neumann solves may be lost. This tolerance (epsSD) is set to be 10^{-12} in the input file.
- In utproj/proj/GNUmakefile add balPrecond.o to OBJS and rs.o to fOBJS.

3 Changes related to compiling the code

- In utproj/work/cmplrflags.mk change in section "Intel-Linux computers using Portland group compilers" the names of Fortran and C compilers:

  FC := ifc (FC := ifort) and CC := cc
- Since we don't have the library pb,
  - comment out the corresponding line in the same file:
    # LDFLAGS := -L$(HOME)/bin -lpb
  - surround each pbUser.h with #ifdef USE_PB and #endif:
    #ifdef USE_PB
    #include "pbUser.h"
    #endif
    in files linearOperatorConjGrad.C, driver.C, cg.C, sdface.C, mesh.C, solnMgr.C, subdomain.C, and mgrid.C in directory utproj/proj;
  - surround STARTPB(*routine name*), ENDPB(*routine name*) with #ifdef USE_PB:
    #ifdef USE_PB
    STARTPB(cg)
    #endif
    in files proj/cg.C, sdface.C, subdomain.C, mgrid.C in directory utproj/proj.
- Set KELP_HOME in .bashrc:

  export KELP_HOME="/afs/.comp.math.pitt.edu/usr/gepst12/utproj/kelp"
- Specify the directories for mpich in kelp/config/x86-g++-linux:

  MPI_INCLUDE = /usr/include
  MPI_LIB = /usr/lib

- Link to the right libraries in the same file:

  ARCHLIBS = -L$(MPI_LIB) -lmpich -lpmpich -lm -llapack -lblas -lg2c

- Suppress some output during compilation (optional) in the same file:

  WARNINGS = -Wstrict-prototypes -Wmissing-prototypes -Wno-deprecated

- In file utproj/proj/lowestlevel.h add the header file for math library:

  #include <math.h>

  and comment out the functions fabs, sqrt, exp, log and round.

- In utproj/proj/parseLine.h add (twice) class to friend Parseline:

  friend class ParseLine;

- Same for SDFaceArray in utproj/proj/sdface.h:

  friend class SDFaceArray;

4 Miscellaneous changes

- The routine checkNeumann in file utproj/proj/balPrecond.C checks if the global problem and the local Neumann problems are pure Neumann and sets flags GlobalNeumann (defined globally in masterdefs.h) and LocalNeumann (defined locally in file utproj/proj/subdomain.h). In the case of global full Neumann problem, we call the routine rmBdryFluxesG which calculates and removes the mean value of the flux so that the RHS for the interface-cg is perpendicular to the nullspace (constants) and the system is consistent.

- Introduce a global flag guessType and use it to specify initial guess for Dirichlet local solves. For computing $(\bar{\mathbf{u}}, \bar{p})$, in InitBalance, at the beginning of interface-cg set, and on the first call to balance set it to ZERO to use zero initial guess. When advancing in interface-cg, set it to IntFaceCG to use the stored solution (la_intfaceCG in subdomain.C) from the previous iteration as a initial guess. Likewise, for the next calls to balance set guessType equal to BALANCE to use the saved solution (la_balance) form previous call as a initial guess. The numerical tests that we run showed reduction of the number of iterations (in conjGrad) for Dirichlet local solves and increased robustness of the interface solver when using this special initial guess.

- In the case of local pure Neumann problem, the chosen tolerance for the local solves (epsSD) may lead to loss of orthogonality between the RHS and the nullspace and the linear system may become inconsistent. To resolve this issue we make use of the orthogonal projector onto Null(A). More precisely, instead of solving the system $Ax = b$ when $Ac = 0$ ($c = (1, 1, \ldots, 1)^T$) we solve

  $$P^T A P y = P^T b$$

  where

  $$P = P^T = I - \frac{c\, c^T}{c^T\, c} \qquad \text{i.e.} \qquad Pu = u - \frac{(c,u)}{(c,c)} c$$

- Allow variable Dirichlet BCs obtained from an analytical function (in routine applyPresBC in file utproj/proj/sdface.C) and variable permeability tensor K (givenTensor in utproj/proj/subdomain.C) using a global variables Test (int) and Ptype (PERMTYPE) defined in utproj/proj/masterdefs.h.

- Some auxiliary routines were implemented: rmBdryFluxes, dsumBdry, dsumBdryG, dsumBdryArea, dsumBdryGArea, ddotBdry, ddotBdryArea in file utproj/proj/balPrecond.C and fillOuterBdry in file utproj/proj/sdface.C.
- In utproj/proj/ftnParcel.h add the following blas and lapack routines: l_WPOTRF, l_WPOTRS, b_dasum used in balPrecond.C.
- The Fortran subroutine rs (utproj/proj/rs.f) calls the recommended sequence of subroutines from the eigensystem subroutine package (eispack) to find the eigenvalues and eigenvectors (if desired) of a real symmetric matrix.
- New debug routines in balPrecond.C: show and showA for outputting structures of type SDFaceArray, doubleVector2r, or doubleArray; sumA for finding the sum of doubleArray.
- In file utproj/proj/subdomain.C rename the old routine solveSubdomains as compflux and break it into two parts: solveSubdomains which sets BCs, solves local problems, and extracts pressure, velocity and flux AND a call to swapbdry which computes the jump of the flux.
- Make the existing variables Debug and Verbosity global in utproj/proj/masterdefs.h and use them to output various information:
  - Verbosity = 0: no extra information is outputted
  - Verbosity = 2: balSlvType, warning in case of a small eigenvalue, and the out-of-symmetry and out-of-balance amounts
  - Verbosity = 4: previous plus number of iterations for subdomain solves
  - Verbosity = 6: previous plus balancing matrix and its Cholesky factorization or eigen-decomposition, and for global Neumann problem, the mean value of the flux after rmBdryFluxesG
  - Debug = 0: no extra information is outputted
  - Debug = 2: some scalar variables like α, β, ρ in CG algorithms
  - Debug = 4: previous plus r, z in cg
  - Debug = 6: all debug outputs
- In function tecplot in file utproj/proj/solnMgr.C combine all output zone files into one file named solution.tec and fix the declaration of the function in utproj/proj/solnMgr.h. Introduce another flag Output in input file file to specify if we want to get a tecplot file with the solution.

5 Numerical tests

All examples are on the unit cube with uniform prismatic meshes that are refined four times. The initial mesh has each of the intervals in x,y,z direction split in 2 equal parts (2x2x2) and we double that number when refine: 4x4x4, 8x8x8, 16x16x16, 32x32x32 so we quadruple the number of d.o.f for the interface problem. We consider the following cases of domain decomposition:

1. 2 subdomains along x-direction (2 x 1 x 1)
2. 4 subdomains: 2 along each x- and y-direction (2 x 2 x 1)
3. 8 subdomains: 2 along each of the three directions (2 x 2 x 2)

4. 27 subdomains: 3 along each of the three directions (3 x 3 x 3)

and compare the number of interface-cg iterations with diagonal and balancing preconditioning when the mesh is refined. The interfaces between the subdomains are flat.

We did numerical tests on two examples. The problem in Example 1 has analytical solution pressure $p(x, y, z) = x^3y^4 + \sin(xy)\cos(y)$, permeability tensor

$$K = \begin{pmatrix} (x+1)^2 + y^2 & 0 & 0 \\ 0 & (x+1)^2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and velocity field $U = K\,\nabla p$ (RHS = $-\nabla \cdot U$, see [4]). The boundary conditions are Dirichlet (from the true pressure) on x = 0, x = 1, and noflow otherwise. The number of iterations with balancing and diagonal preconditioning for the three cases are given in Table 1.

$1/h$	*2 x 1 x 1 dom*		*2 x 2 x 1 dom*		*2 x 2 x 2 dom*		*3 x 3 x 3 dom*	
	BalCG	*CG*	*BalCG*	*CG*	*BalCG*	*CG*	*BalCG*	*CG*
2	1	2	1	4	3	8		
4	2	4	3	8	5	15	7	23
8	2	7	4	14	7	22	8	31
16	2	11	4	19	8	32	10	42
32	2	17	5	28	10	46	11	58

Table 1: Number of iterations for Example 1

To reproduce these result, one can use the input and velocity files that are stored in directory utproj/utproj/work/tests/example1. First one needs to copy each of the velocity files for the chosen test in file utproj/utproj/work/PE????/data.3d before running the code. For example, to use mesh 16x16x16 on 2x1x1 domains, type the following commands from directory utproj/utproj/work (assuming directories PE0000 and PE0001 already exist):

cp tests/example1/pe0_ex1_16x16x16_2x1x1dom.3d PE0000/data.3d
cp tests/example1/pe1_ex1_16x16x16_2x1x1dom.3d PE0001/data.3d
mpirun -np 2 utproj tests/example1/mesh_ex1_16x16x16_2x1x1dom.3d

Alternatively, a user can submit batch jobs using the same convention in naming the input files (see file utproj/README).

The problem in Example 2 has permeability tensor $K = I$ and velocity field

$$U(x, y, z) = \begin{pmatrix} -\frac{\pi}{2}\sin(\frac{\pi}{2}x)\cos(\frac{\pi}{2}y) \\ -\frac{\pi}{2}\cos(\frac{\pi}{2}x)\sin(\frac{\pi}{2}y) \\ z \end{pmatrix}$$

The boundary conditions are Neumann everywhere. The number of iterations with balancing and diagonal preconditioning for the three cases are given in Table 2.

References

[1] GERGINA PENCHEVA AND IVAN YOTOV, *Balancing domain decomposition for mortar mixed finite element methods*, Numer. Linear Algebra Appl. 2003; **10**:159-180.

$1/h$	*2 x 1 x 1 dom*		*2 x 2 x 1 dom*		*2 x 2 x 2 dom*	
	BalCG	*CG*	*BalCG*	*CG*	*BalCG*	*CG*
2	2	2	1	3	2	8
4	2	7	3	9	4	13
8	2	10	3	14	6	17
16	2	15	3	21	8	24
32	2	20	4	30	10	35

Table 2: Number of iterations for Example 2

[2] J. MANDEL, *Balancing domain decomposition*, Communications in Numerical Methods in Engineering **9** (1993), no. 3, 233–241.

[3] L. C. COWSAR, J. MANDEL, AND M. F. WHEELER, *Balancing domain decomposition for mixed finite elements*, Mathematics of Computation **64** (1995), 989–1015.

[4] S. CHIPPADA, C. N. DAWSON, M. L. MARTÍNEZ, AND M. F. WHEELER, *A projection method for constructing a mass conservative velocity field*, Comput. Methods Appl. Mech. Engrg. **157** (1998), 1–10.