PATC Elmer FEM – Linear Solver Tutorial

In this tutorial we will try to solve a linear elasticity (or heat equation) problem in a cheese- like domain (a 3D block with a number of holes). The idea is to have a linear case of which complexity can be easily increased by adding more holes to the cheese. Otherwise the smooth solutions may in large cases give too optimistic view of the scalability.

Linear Solvers in Elmer

Elmer includes large number of linear solvers and preconditioners. Some of them are only serial, some only parallel. Some are part of Elmer code but many are also third party external libraries. The optimal solver depends on the size of the problems, number of partitions, type of linear system etc.

Linear Solver	Serial	Parallel	Optional
Umfpack (direct)	X		
MUMPS (direct)		X	X
Elmer Krylov method	X	X	
Elmer Preconditioners (ILU, BILU, diag., vanka, BP)	X	X	
Elmer GMG	X	X	
Elmer AMG (Ruge-Stueben & Agglomeration)	X		
Elmer FETI		X	
Hypre (Krylov methods, Boomer AMG, Parasails, ILU)		X	X
Trilinos (Krylov methods & ML)		X	X
Pardiso (direct)		X	X

Mesh definition

The geometry has been created by a Matlab script (where the generation of holes is randomized) and meshes with netgen mesh generator. Three predefined meshes at different resolutions are provided in `holes10/mesh1` (coarsest mesh) through `holes10/mesh3` (finest). These meshes have 8-fold increase in number of elements.

Mesh density can be increased also by mesh multiplication may be done a posteriori by the "Mesh Levels". This does not increase the geometric accuracy though but provides easy means of making huge meshes in the parallel case. So using mesh3 is roughly as mesh2 with "Mesh Levels = 2".

Mesh partitioning

Start out by partitioning the meshes for your parallel computations. For instance:

```
>cd holes10/
>ElmerGrid 2 2 mesh1 -partition 2 2 1
```

to partition the coarsest mesh into 4 parts uniformly and

```
>ElmerGrid 2 2 mesh1 -partdual -metis 4 4
```

uses the Metis graph partitioning on the dual mesh and algorithm “4” to yield 4 parts.

Running the case

When running the case make sure that the ELMERSOLVER_STARTINFO refers to the command file you want to study and that partitioning has been made for the number of partitions that you request.

The relevan command file for the case is `cheese_stress.sif` (or `cheese_poisson.sif`). The linear system settings in the command file are inserted from file `linsys.sif` that is inserted when the file is read. Default settings in solve the coarsest problem using BiCGStab, preconditioned by an ILU with fill-level 1.

Run the solver in serial mode run the solver by:

```
>ElmerSolver
```

And similarly in parallel mode:

```
>mpirun -np 4 ElmerSolver_mpi
```

Time usage is saved to the file with suffix `.dat` (and metadata to file `.dat.names`) and you may study them there using Matlab, Octave etc.

Try solving the system on 1, 2 and 4 processors. As you can see, this problem cannot be solved straight-forwardly in parallel and we will have to find better strategies. The problem with the ILU preconditioners in Elmer is that they are only done partition-wise and therefore provide inferior convergence in parallel.

To run through all the linear system settings (see Appendix A) in directory `linsys` you can (in serial) use a simple script like:

```
#!/bin/bash
for s in linsys/*.sif; do
    cp $s linsys.sif
    ElmerSolver
done
```

You can mask certain families, e.g. `hypre` by selecting files `hypre*.sif`.

Parallel ILU Preconditioners in HYPRE and Trilinos

HYPRE and **Trilinos** interfaces are only available in parallel, so run these tests on at least two cores using `ElmerSolver_mpi` (There are a number of different predefined strategies in the directory linsys).

The HYPRE library contains the parallel ILU preconditioner Euclid. To use it you need to add the line

```
Linear System Use HYPRE = True
```

to the Solver section of your sif-file. The linear solver settings are taken from the sif-file so you do not have to change anything else. It is recommended to use ILU0 or ILU1 here because the Euclid ILU is fairly slow to compute. Observe how the number of iterations is now no longer increasing drastically with the number of cores. What about the speed-up?

Trilinos does not contain a parallel ILU preconditioner, but parallelizes ILUs by the Additive Schwarz method. Overlap between the subdomains can be added to achieve better numerical scalability. Play a little with the Trilinos package ***ifpack*** by setting

```
Linear System Use Trilinos = Logical True
Trilinos Parameter File = String ifpack.xml
```

In the file `ifpack.xml` you can adjust the ll-level of the ILU to achieve faster convergence and the amount of overlap to achieve a slower increase in iterations when using more processors. As there is currently no BiCGStab solver in Trilinos, we use GMRES instead. Check the numerical scalability (increase of iterations) on up to 16 MPI tasks (note that there are only 4 cores, so an actual speed-up cannot be expected).

Try the fastest method(s) you have found so far on the finer grids (`mesh2` and `mesh3`). What happens?

Block Preconditioning

We can write the system matrix of a linear elasticity problem in a component-wise (block-) form:

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

It can be shown that this system is spectrally equivalent to the block-diagonal matrix

$$\hat{A} = \begin{pmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{33} \end{pmatrix},$$

which means that if we use Â as a preconditioner for A we can expect a convergence rate independent of the (smallest) mesh-size h. As systems with a block diagonal matrix are much easier to solve, we will construct preconditioners base on $\hat{A}$ rather than A. There are several ways to do this:

Serial Block Preconditioning using Algebraic Multigrid

When using the Elmer ILUn preconditioners, simply replace e.g. ILU2 by BILU2. Note that this just makes the ILU cheaper to compute but the scalability issues remain. When using HYPRE, add the line

```
HYPRE Block Diagonal = Logical True.
```

Any HYPRE Preconditioner will then be based on $\hat{A}$. By using the general block preconditioning facility in Elmer. An example of such a file is given in `cheeseBlockSolveCMG.sif`, where the highly efficient (but sequential) clustering AMG solver is used to approximate the inverse of the diagonal blocks. The approach is rather general, which makes the sif-file more complicated. Any parameters related to the Krylov method for A receive a prefix `Outer:`'. GCR is used rather than BiCGStab because the method applied to the blocks may not be a constant operator.

The option `Block Gauss-Seidel = True` means that we use the upper triangular part of $\hat{A}$ rather than A, which does not add additional linear system solves but typically decreases the number of iterations by about a factor 2.

Parallel Block Preconditioning using Algebraic Multigrid

In the previous section we saw that the combination of block preconditioning with a few cycles of algebraic multi-grid (AMG) for the blocks gives a very efficient preconditioner for the linear elasticity problem at hand. The AMG method used was based on clustering (=aggregation), a parallel variant of which is avail-able in the Trilinos package ML. Adjust the following parameters to apply an ML cycle to each block:

```
Block 11: Linear System Solver = Iterative
```
(and likewise for 22, 33)

```
Linear System Use Trilinos = Logical True
Trilinos Parameter File = String ml.xml
```

The ML method has a very large number of parameters, but it provides sets of reasonable default values for some classes of problems. We will use the `smoothed aggregation' (SA) defaults (smoothed refers to the smoothing of the transfer operators here, not the smoother on each level. The clustering method in Elmer is an `unsmoothed aggregation' method).

Test this solver for `mesh2` and `mesh3` and see how it scales in terms of iterations and time to solution. Experiment with different smoothers, e.g. set "smoother: type" in `ml.xml` to "`Aztec`" (an ILU0 will be used on each sub-domain) or "`Chebyshev`" (a very scalable smoother as it only involves matrix-vector products).

Applying Algebraic Multigrid to full matrix

You may also Apply ML to the full system matrix A by starting out from cheese.sif and using the parameter file `gmres_ml.xml`, Compare with the parallel AMG solver from Hypre, called BoomerAMG, instead of ML. Start from cheese.sif and set

```
Linear System Use HYPRE = True
HYPRE Block Diagonal = Logical True
```
(optional)

```
Linear System Preconditioning = BoomerAMG
```

The BoomerAMG solver is based on the older `Ruge-Stuben' coarsening strategy which mimics geometric multigrid and also has a very large number of parameters that can be tuned (not all of them are available in Elmer).

Appendix A: Predefined linear system settings

Below is a non-exhaustive list of linear solver options in directory `linsys`. To use some of these definitions just copy it to the file `linsys.sif` that is inserted in the command file at runtime.

```
direct_MUMPS.sif
direct_umfpack.sif
elmer_GCR_GMG_jacobi_BiCGStab_none.sif
elmer_GCR_GMG_wjacobi_CG.sif
elmer_GMG_sgs_BiCGStab_none.sif
elmer_feti_mumps.sif
elmer_feti_mumps_10.sif
elmer_iter_BiCGStab2_BILU0.sif
elmer_iter_BiCGStab2_BILU1.sif
elmer_iter_BiCGStab2_ILU0.sif
elmer_iter_BiCGStab2_ILU1.sif
elmer_iter_BiCGStab2_none.sif
elmer_iter_BiCGStab2_vanka.sif
elmer_iter_BiCGStab4_ILU0.sif
elmer_iter_BiCGStab4_ILU1.sif
elmer_iter_BiCGStab4_none.sif
elmer_iter_BiCGStab_ILU0.sif
elmer_iter_BiCGStab_ILU1.sif
elmer_iter_BiCGStab_none.sif
elmer_iter_CG_ILU0.sif
elmer_iter_CG_none.sif
elmer_iter_CG_vanka.sif
elmer_iter_GCR_BP_CMG_SGS.sif
elmer_iter_GCR_GMG_BiCGStab_none.sif
hypre_BiCGStab_BoomerAMG.sif
hypre_BiCGStab_BoomerAMG_smoother3.sif
hypre_BiCGStab_BoomerAMG_smoother6.sif
hypre_BiCGStab_BoomerAMG_wo_scaling.sif
hypre_BiCGStab_ILU0.sif
hypre_BiCGStab_ILU1.sif
hypre_BiCGStab_Parasails.sif
hypre_BoomerAMG.sif
hypre_FlexGmres_BoomerAMG.sif
hypre_GMRes_ILU1.sif
hypre_GMRes_Parasails.sif
hypre_LGMRes_ILU0.sif
trilinos_ml_sgs.sif
trilinos_ml_sgs_dof3.sif
```