EA-AMG - software for parallel solution of elasticity problems using aggregation.

Roman Kohut, 8.1.2015
Institute og Geonics, Academy of Sciences of the Czech Republic
kohut@ugn.cas.cz

1 Introduction

Software EA-AMG contains two codes - SOLVER_AG and AGGREG. The specific feature of this sofrware is the using of regular structured grids which, simply speaking, are the grids arising as a result of deformation of some regular rectangular grid (the pattern grid). The discretization is given by the decomposition of the domain into eight-node bricks and each brick is consequently decomposed into 6 tetrahedral finite elements. Hence the corresponding matrix has all nonzero entries within a 27 node regular stencil. Supposing the symmetry of the matrix, we can additionally store only the upper triangular part of the matrix. It can be done row-by-row by using regular 42 element stencil (the displacements in three directions x,y,z correspond to each node) for the storage of the nonzero matrix entries.

The code **SOLVER_AG** is designed for the solution of large linear systems arising from the finite element analysis of 3D boundary value problems of elasticity. The corresponding linear system of algebraic equations is solved by the preconditioned conjugate gradient method with the additive two level overlapping Schwarz preconditioner. The domain is decomposed along the z direction into several non-overlapping subdomains, which are then extended, so that adjacent subdomains have usually the minimal overlap (the number of overlapping layers is optional).

The code AGGREG serves for preparation of an aggregated matrix. The code AGGREG uses a new agregation technique which presents the generalization of the aggregation for fully compatible FE spaces.

The codes are written using Fortran 95, for parallelization OpenMP paradigm is used. The code was tested on Super Micro computer (symmetric multiprocessor) with 8 AMD Opteron 8380 processors under UNIX operating system.

2 Code distribution

Software **SOL_AG** is available from the web site

http://www.ugn.cas.cz/,

questions can be sent to *roman.kohut@ugn.cas.*

The software **SOLV_AG** is free software distributed under the terms of the GNU General Public License as published by the Free Software Foundation.

3 Numerical methods

Numerical solution of the elasticity problem is based on the discretization of a studied domain Ω by a regular grid and on the application of the finite element method arising from the variational form and leading to the solution of a large linear system

$$Au = b,\ u, b \in R^n,$$

with symmetric $(n \times n)$ positive definite stiffness matrix A. For the solution we use the preconditioned conjugate gradient method with two-level overlapping additive Schwarz preconditioner and aggregated coarse matrix. In our code the domain is divided into m subdomains Ω_k in z direction. The linear system is solved in parallel and the number of subproblems corresponds to the number of used processors. The solution of subproblems is replaced by the incomplete factorization of the block-diagonal matrices arising by the displacement decomposition of the subproblem matrices. The solution of the coarse problem is solved approximately by an inner PCG method with appropriate accuracy ε_{in}. If the solution of the coarse problem is not exact, the orthogonality of a searched direction is violated and it is necessary to modify a new direction by the orthogonalization procedure. Both Dirichlet and Neumann boundary conditions can be applied. In the second case, arising singular system can be efficiently solved making use of projections.

The matrix corresponding to a coarse problem is constructed using an aggregation technique. In the case of fully compatible FE spaces we can derive formulae for the relations between the coarse matrix elements and the fine matrix elements. In general case the fine grid is so complicated that the fully compatible coarse grid space doesn't exist. We use structural grids in our software, it means that the position of each node is given

by the triplet of indices (i, j, k) and the corresponding nodal coordinates $(x(i, j, k), y(i, j, k), z(i, j, k))$. The general structural grid with $n_x \times n_y \times n_z$ nodes corresponds to the rectangular uniform "index" grid where the node in the position (i, j, k) has the coordinates (i, j, k). The mapping

$$\Phi : (i, j, k) \longrightarrow (x(i, j, k), y(i, j, k), z(i, j, k))$$

presents the isomorphism between nodes of the "index" grid and the corresponding general structural grid. Now we can modify the relations between matrix elements derived for fully compatible spaces using index coordinates instead of the real coordinates for the determinations of coefficients in the relations. In the case of noncompatible "index" grids we use some approximation of these coefficients.

4 The storage of the files

As was written in the first part, a specific feature of our code is the using of regular structured grids. If nx, ny, nz represent the numbers of nodes in corresponding directions, the nodes can be easily indexed by the triples (i,j,k),

$$1 \leq i \leq Nx, \quad 1 \leq j \leq Ny, \quad 1 \leq k \leq Nz$$

or enumerated by 1D numbers

$$indn = i + (j-1) * Nx + (k-1) * Nx * Ny.$$

It means that enumeration is done first in the direction x, then in direction y and finally in direction z.

a) matrix storage

For 3D elasticity problem we have aditionally 3 degrees of freedom (DOF) in each node. These DOF correspond to the displacement in the directions x, y, z. The overall number of DOF is $nd = nx \times ny \times nz$ and the degrees of freedom at the node with index $indn$ can be indexed by the numbers

$$\begin{aligned} indx &= 3*indn - 2, \\ indy &= 3*indn - 1, \\ indz &= 3*indn. \end{aligned}$$

The row numbers are given by the order of the records, the column numbers have not to be stored due to the regular stencil. More exactly, the column numbers for the regular stencil and for the row index i will be the following:

If $i = indx$ for some DOF, then the column numbers used in the 42 element stencil are subsequently:

$$j =, i, i+1, ..., i+5$$

and futher

$$j = k, k+1, ..., k+8 \quad for \quad k = i+k1, i+k2, i+k3, i+k4,$$

where

$$k1 = 3 * nx - 3, \quad k2 = 3 * nx * ny - 3 * nx - 3,$$

$$k3 = 3 * nx * ny - 3, \quad k4 = 3 * nx * ny + 3 * nx - 3.$$

If $i = indy$ for some DOF, then the last entry in the stencil is not used and for the remaining we have

$$j =, i, i+1, ..., i+4$$

and futher

$$j = k, k+1, ..., k+8 \quad for \quad k = i-1+k1, i-1+k2, i-1+k3, i-1+k4.$$

If $i = indz$ for some DOF, then the last two entries in the stencil are not used and

$$j =, i, i+1, ..., i+3$$

and futher

$$j = k, k+1, ..., k+8 \quad for \quad k = i-2+k1, i-2+k2, i-2+k3, i-2+k4.$$

The stifness matrix is stored in $real * 4$. Each record corresponds to some node and contains three 42 element stencils for directions x, y, z. For opening and reading file with stored matrix we use folowing instructions:

open(1,file='fkbc.g32', access='direct', recl=504),

The file is read as stored to the matrix in the memory as follows:

```
do i=1,nn
   read(1,rec=i) aa
   do j=1,126
      a(j+(i-1)*126)=aa(j)
   enddo
enddo
```

where aa is the vector of the length 126 and a is the vector (stifness matrix) of the length $nn * 126$ where nn is the number of the nodes.

b) Vector storage

Vectors contain nodal values (the triplet values for each node) corresponding to the triplet of indices (i,j,k). For the node with index $indn$ we have three values of vector v:

$$v(3 * indn - 2), \quad v(3 * indn - 1), \quad v(3 * indn).$$

Vectors are stored in $real * 4$. For opening and reading files with stored vectors we use following instructios:

```
open(1,file=filename, form='unformatted'),
read(1) (v(i)=1,n),
```

where $n = 3 * nn$ is the number of unknowns (DOF), $filename$ is the name a file (e.g. the displacemet vector is stored in the file $fu.g32$).

c) Nodal coordinates storage

The coordinates are stored in the file $fx.g32$ according to the numbering of nodes.

```
open(1,file='fx.g32', form='unformatted'),
do i=1,nn
   read(1) x(i),y(i),z(i)
enddo
```

Here $(x(i), y(i), z(i))$ are coordinates of the node with the index number i.

d) Parameters of task storage

These parameters are stored in the formatted text file $fv.g32$.
open(1,file='fv.g32'),

READ(1,11) name,date	- the name (CHAR*50),date (CHAR*8)
READ(1,12) nx,ny,nz	- the numbers od the nodes in the directions x, y, z
READ(1,13) xmin,xmax	- the size in the direction x
READ(1,13) ymin,ymax	- the size in the direction y
READ(1,13) zmin,zmax	- the size in the direction z
READ(1,14) nmat	- the number of materials
READ(1,15) nn	- the number of nodes
READ(1,15) nd	- the number of unknowns(DOF)
READ(1,15) nc	- the numbrer of hexahedral bricks
READ(1,15) nel	- the number of tetrahedral elements (nel=nc*6)

11 FORMAT(T7, A50, 1X, T64, A8)
12 FORMAT(T40, I3, 1X, I3, 1X, I3)
13 FORMAT(T40, F10.4, 1X, F8.2)
14 FORMAT(T40, I3)
15 FORMAT(T40, I8)

e) Input data storage

Files containing input data are text files with a free format.

5 Routines used in codes

Routines used in the code **SOLV_AG** are inserted in folowing files:

itera_el_z.for	- the main routine for controlling the running of the code.
pcg_el_z.for	- subroutine for realization of preconditioned conjugate gradients.

pcg_inner_IS.for	- the subroutine for realization of preconditioned conjugate gradients on a coarse problem (inner iterations.)
mxv_el_z.for	- the subroutine for the multiplication of matrix A by vector v.
pcond_el_z.for	- two subroutines for the preconditioning based on the additive overlapping Schwarz methods.
rfv.for	the subroutine for reading of parameters for the task.
agr_vek_el_z.for	- the subroutine for the aggregation of vectors.
int_vek_el_z.for	- the subroutine for the interpolation of vectors.
agr_koef_el_z.for	- the computing of the coeficients for the aggregation.
r_agr_el_z.for	- the reading of the parameters for the aggregation.
geore_el_z.for	- subroutine for modification of matrices and rhs according to boundary conditions.
diag_hex_el_z.for	- the preparation of a vector for the lumping of the stiffness matrix.
lin_par_o3.mki	- information for compiler
solver_el_z.mk	- Makefile

Routines used in the code **AGGREG** are inserted in folowing files:

agregace_el_z	- the main routine for controlling the runing of the code.
rfv.for	- the subroutine for reading of parameters for the task.
agr_vek_el_z.for	- the subroutine for the aggregation of vectors.
agr_mat_el_z.for	- the subroutine for the aggregation of the matrix.
agr_koef_el_z.for	- the computing of the coeficients for aggregation.
r_agr_el_z.for	- the reading of the parameters for the aggregation.
rmat.for	- the reading of the matrix.
wmat.for	- the writing of the aggregated matrix.
lin.mki	- information for compiler
agregace_el_z.mk	- Makefile

6 Makefile for compilation

The compilation is done in two steps:

1.step: **make -f solver_el_z.mk.mk clean**
The files $*.f, *.fo, *.o$ are deleted.

2. step **make -f solver_el_z.mk.mk**
The compilation is done.

The same steps are doing with agregace_el_z.mk.

7 Input and output files for the code AGGREG

The code AGGREG serves for preparing aggregated matrix and aggregared vectors which are necessary for running of the solver SOLV_AG. The code AGGREG needs following files:

$rfv.g32$	- the text file with parameters
$fkbc.g32$	- the stiffness matrix
$frw.g32$	- the vector of the nodal forces, which are equivalent statically to the boundary stresses and distributed loads (the weight of material)
$fbc.g32$	- the vector corresponding to a geometric boundary conditions in nodes.
$r\_agr.in$	- the text file for the controlling of the aggregation
$fx.g32$	- the file of the coordinates of the nodes

The structure of these files was described in Section 4. Note that rows of the matrix corresponding to nodes in emty area (nodes inside holes, tunnels) have all elements equal to zero. The code *SOLV_AG* (subroutine geore_el_z.for) replaces corresponding zero element in position of diagonal element with value one.

The Dirichlet boundary conditions are saved in file *fbc.g32*. The file vector contains following values in positions correponding to node with index i:

bc($3*i-k$*), k=2,1,0*	- $v(3*i-k)$ for *node_i* with prescribed value in the corresponding direction
bc($3*i-k$*), k=2,1,0*	- $1e9$ for free node i in the corresponding direction

The input file *r_agr.in* contains parameters for the aggregation. For the first start of *AGGREG* this file doesn't exist and is generated during the running of the code. The file is text file with data in free format. The data are stored in following order:

tag	- the type of aggregation (0 - the clustering of neighbouring nodes, 2 - the tetrahedra coarsening, 3 - the hexahedra coarsening)
itp	- the type of coarsening (0 - regular, 1 - nonregular)
if itp=0	
nxh,nyh,nzh	- the number of nodes for a coarse grid
if itp=1	
ix(i),ixh(i)	- sequentially indices of a fine grid node and corresponding coarse grid node in the direction x
..	..
iy(i),iyh(i)	- sequentially indices of a fine grid node and corresponding coarse grid node in the direction y
..	..
iz(i),izh(i)	- sequentially indices of a fine grid node and corresponding coarse grid node in the direction z

The output files are the following:

fkbc_ag.g32	- the agrregated stiffness matrix
frw_ag.g32	- the aggregated vector of the nodal forces
fbc_ag.g32	- the aggregated vector corresponding to a geometric boundary conditions in nodes.
fx_ag.g32	- a vector of nodes coordinates for a coarse grid.

8 Input and output files for the code SOLV_AG

The code *SOLV_AG* serves for the solution of system of linear equations generated by the using of FEM to linear elasticity problems. The code *SOLV_AG* needs following input files:

$rfv.g32$	- the text file with parameters
$fkbc.g32$	- the stiffness matrix
$fkbc\_ag.g32$	- the aggregated stiffness matrix
$frw.g32$	- the vector of the nodal forces, which are equivalent statically to the boundary stresses and distributed loads (the weight of material)
$frw\_ag.g32$	- the aggregated vector of the nodal forces
$fbc.g32$	- the vector corresponding to a geometric boundary conditions in nodes.

$fbc.g32$	- the aggregated vector corresponding to $fbc.g32$
$r\_agr.in$	- the text file for the controlling of the aggregation
$solver.in$	- the text file for the controlling of running of the solver
$n\_process.in$	- the number of processors used in the parallel computing

The files $*.g32$ and $r\_agr.in$ were described in the previous section.

The text file $solver.in$ serves for the controlling of the running of the solver. The parameters are stored in the following order:

eps	- the accuracy for the solution of linear system
ipc	- the type of preconditiong for solution on subdomains: ipc=1 (diagonal preconditiong),ipc=3 (the Cholesky incomplete factorization
itmax	- max. number of pcg iterations for the solution on the fine grid
iap	- the type of the initial approximation: iap=0 (the zero vector), iap=1(the incomplete factorization for the fine problem), iap=2 (the given vector $fu.g32$)
ipr	- for the Neumann problem the projection of the rhs vector is necessary: ipr=1 (the Neumann conditions are on all 6 walls), ipr=3 (the Neumann conditions are in the directions x and y - 4 walls), ipr=4 (the Neumann conditions are in the directions x and z), ipr=5 (the Neumann conditions are in the directions y and z)). The forces must be in equilibrium.
iort	- this parameter is usually used if the problem on coarse grid is solved using pcg iterations. If the small accuracy is used the searched directions in the pcg iterations on fine problems are not orthogonal and therefore must be orthogonalized. The parameter $iort$ presents the number of previous directions the new direction must be orhogonal to.
ag	- ag>0, the aggregation will be used, the coarse problem will be solved using inner pcg iterations.
epsin	- the accuracy for the solution of linear system for coarse problem
itmaxin	- max. number of pcg iterations for the solution of the coarse problem

iapin - the type of the initial approximation: iapin=0 (the zero vector), iapin=1(the incomplete factorization for the fine problem)

ipcin - the type of preconditiong for solution of a coarse problem: ipc=1 (diagonal preconditiong),ipc=3 (the Cholesky incomplete factorization

The text file *n_process.in* contains the number of used processors (it is equal to the numbers of subdomains) and the number of overlapping layers between neighbouring subdomains. The parameters are stored in the following order:

npro - the number of processors for a fine grid problem
npr - the number of overlapping layers for a fine problem
npro_ag - the number of processors for a coarse grid problem
npr_ag - the number of overlapping layers for a coarse problem

The output files are $fu.g32$ (the vector of displacements) and *solver.rep* (text file reporting on the course of iterations).

9 Conclusion

More information can be found in [1]. The paper is in preparation, the questions can be sent to roman.kohut@ugn.cas.cz. Some information about the domain decomposition used in the code can be founf in the description of free solver ISOL 1.45a documentation (see [2]).

References

[1] R. Kohut:Parallel Solution of Elasticity Problems using Aggregations, in preparation

[2] R. Blaheta, O. Jakl, J. Starý: Library of parallel PCG solvers. http://www.ugn.cas.cz/other/sw-gem/elpar-1.0.pdf