Short reference SPC-ATPo3

Frank Milde
Fakultät für Mathematik, TU Chemnitz

December 19, 2000

1 Intro

SPC-ATPo3 is an adaptive, parallel FEM program solving either the Poisson or Lamé boundary value problem on (linear or quadratic) tetrahedral meshes. It is based on SPC-PM Po 3D v4.0 [1] (and previous versions [2]), but it now includes adaptive mesh refinement, error estimation, load balancing, and extended material parameters. The program solves the following equations:

Poisson problem (1 degree of freedom):

$$-\mathrm{div}(\mathbf{A}\nabla u) + c\,u = f \quad ,$$

with $\mathbf{A} = \mathrm{diag}(a_1, a_2, a_3)$ being the anisotropic material constants.

Lamé problem (3 degrees of freedom):

$$-\mu\nabla \underline{u} + (\lambda+\mu)\mathrm{grad}\,\mathrm{div}\,\underline{u} = \underline{f} \quad .$$

The right hand side f or $\underline{f}$ is either

- Zero (default).
- Given by the Material data in the mesh file, see 1.1.
- Defined in `bsp.f` (`UseBspF`=1, see 1.2).

1.1 New `MATERIAL` parameters (standard file)

The present program version can handle anisotropic material constants and volume forces. Such material properties can be defined in the `MATERIAL` section of the mesh file (type: standard file). Up to 5 real parameters $\underline{C} = (C_1, C_2, ..., C_5)$ are used in the program which have the following meaning.

Poisson problem $\underline{C} = (a_1, a_2, a_3, c, f)$, at least a_1 must be given. Default (standard material) is $\underline{C} = (1, 1, 1, 0, 0)$.

Lamé problem $\underline{C} = (E, \nu, f_1, f_2, f_3)$, at least E and ν must be given. Default (standard material) is $\underline{C} = (2e5, 0.3, 0, 0, 0)$.

If only the first 1 or 2 elements of the vectors $\underline{a} = (a_1, a_2, a_3)$ or $\underline{f} = (f_1, f_2, f_3)$ are given, the remaining ones are set equal to the first.

1.2 Control Parameters in `control.adapt`

Most of the behaviour of the program can be controlled by parameters in the file `control.adapt` as explained below. In the following *elements* stands for *volumes*, *faces* or *edges* (N_{el} pieces), N_{vol} for the number of volumes. η_T is the error estimation in an element, N_{proc} the number of processors, and M the amount of memory used locally on each processor.

new parameters		
parameter	value	description
`MarkVar`	-1	global refinement
	0	markers by hand (for simple test purposes)
	1	markers as defined in `geo_mark.f`
	2	markers according to the error estimation
`MarkLog`	0/1	(if `MarkVar=0`) log/don't log marking to file `LOG_MARK`
`alpha`	$\geq 0.$	(if `MarkVar=2`) mark elements, if $\eta_T >$ `alpha` $* \eta_{max}$
`MinMark`	$0. - 1.$	(if `MarkVar=2`) mark at least `MinMark` $* N_{el}$ elements
`NelCoars`	≥ 0	the coarse mesh has at least `NelCoars` $* N_{proc}$ volumes
	< 0	the coarse mesh has at least |`NelCoars`| volumes
`MemSplit`	0.	distribute the coarse mesh to *all* processors
	$> 0.$	split the mesh on one processor into two parts, if $M >$ `MemSplit` $* M_{max}$, send one half to a free node*
`MaxInbal`	$\geq 0.$	redistribute parts of the coarse mesh (load balancing), if $\frac{\max N_{vol}}{\min N_{vol}} - 1. >$ `MaxInbal`
`ZeroSVec`	0	use (interpolated) old solution as starting vector
	1	use always **0** as starting vector
`RepartVar`**	1–4	different repartition routines from `ParMetis`
`FWeights`	0/1	use no/ use face weights for repartitioning
`SendEMat`	0/1	split or balancing: recompute/send element matrices
`UseBspF`	0/1	use function F from `bsp.f` for right hand side or not
`BatchMode`	-1	normal operation
	$\geq 0.$	"autopilot": no input after filename, perform `BatchMode` global refinements first

*) After the marking of the elements, the new number of volumes is computed and the new local memory usage is estimated by $M_{new} = M * N_{vol}^{new} / N_{vol}$.

**) The following routines from the `ParMetis` library are used in order to compute the new partition of the coarse tetrahedra, which balances the load (number of volumes on the processors) while minimizing the edgecut (number of (coarse) faces which couple to other processors). The partitions from the first two routines typically have higher edgecut but lower migration cost compared to those from the last two routines.

	name	description
1	`ParMETIS_RepartLDiffusion`	diffusive scheme, local imbalance info
2	`ParMETIS_RepartGDiffusion`	diffusive scheme, global imbalance info
3	`ParMETIS_RepartRemap`	new partition & remap to old, local i.i.
4	`ParMETIS_RepartMLRemap`	new partition & remap, global i.i.

The following parameters in `control.adapt` are already known from SPC-PM Po 3D v3.3, for more information see [2].

old parameters		
parameter	value	description
`lin_quad`	1/2	kind of shape functions (linear / quadratic)
`loesvar`	1	Jacobi preconditioning
	2	Yserentant preconditioning
	3	Yserentant with coarse grid solver
	4	BPX preconditioning
	5	BPX with coarse grid solver
`femakkvar`	2/3	variant of kette accumulation
`nint2ass`		number of quadrature formulas
`nint2error`		number of quadrature formulas
`nint3ass`		number of quadrature formulas
`nint3error`		number of quadrature formulas
`ion`	≥ 0	number of iterations between output lines
`iter`	> 0	maximum number of iterations
`epsilon`	$> 0.$	accuracy of the solver
`verf`	$> 0.$	special node displacement for `amw` meshes
`tet_ord`	0/1	reorder face names in volumes

1.3 Output

After starting the program two output switches can be set (see next table). Afterwards, one decides what should be displayed later graphically: solution and/or error estimation and/or marking. Note: error estimation and marking is shown on the first degree of freedom u_x.

key	value	description
`i`*	0 – 5	amount of terminal output, increasing from 0 to 5
`o`		write mesh to `GraphTet.dat` (for `fem_ogl`[3])

*) Particularly useful are levels 2 (default) and 0 (gives a table after definition of the *coarse mesh*). Higher output levels give lots of debug information. In level 0 (table) one can control the program run by the following *secret keys*:

key	description
t	global refinement
0	no refinement, solve same mesh again
n	no further step
<**enter**>	next step

During the run of the program two files are written:

`LOGFILE` includes the filenames being processed, all parameters and two tables with all times and interesting numbers as in info level 0.

`LOG.dat` a shortened version of `LOGFILE` directly usable for graphic programs like `gnuplot`

1.4 Program cycle

The program works (typically) as follows:

0. define useful parameters in `control.adapt`
1. set output level, which graphics, mesh file
2. read in the mesh, solve, estimate the error
3. perform a number of mark/solve/estimate cycles until the mesh is large enough to define the coarse mesh
4. set the coarse mesh, compute coarse grid matrix

Until now, only processor 0 works and Jacobi preconditioning is used. From now on, the coarse mesh tetrahedra can be distributed to other nodes and preconditioning as defined by `loesvar` works.

5. distribute the mesh to all processors (if `MemSplit=0.`)
6. solve, estimate the error
7. perform a number of mark/solve/estimate cycles as follows

   7.1 initial marking according to `MarkVar`

   7.2 extend marking for edges/faces/nodes (green closure etc.)

   7.3 estimate the new load; if all processors have data balance load (depending on `MaxInbal`), otherwise (depending on `MemSplit`) split the mesh among (pairs of) full and empty nodes

   7.4 refine the mesh

   7.5 compute the element matrices of new tetrahedra.

   7.6 solve (using the element matrices)

   7.7 estimate the error of the solution

References

[1] T. Apel, F. Milde, and U. Reichel, *SPC-PM Po 3D V4.0 Programmer's Manual*, preprint SFB 393/99-37.

[2] T. Apel and U. Reichel, *SPC-PM Po 3D V3.3 User's Manual*, preprint SFB 393/99-06.

[3] R. Ruhmer, "3D-Finite-Elemente-Rechnungen und ihre Visualisierung mit Hilfe von OpenGL", Diplomarbeit, Fak. f. Mathematik, TU Chemnitz (2000).