Elastodynamics Tutorials

Mohd Afeef Badri

Abstract

This document details some tutorials of elastodynamics module of PSD. These tutorials are not verbose, but does instead give a kick start to users/developers for using PSD's elastodynamics module.

Parallel 2D

The problem of interest is a single Dirichlet condition (clamped end bar) and traction loading. For this example we use Newmark-β time discretization. Additionally postrocessing is demanded for displacement, acceleration, and velocity (u, a, v).

```
PSD_PreProcess -dimension 2 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization newmark_beta -postprocess uav
```

Once the step above has been performed, we solve the problem using two MPI processes, with the given mesh file bar-dynamic.msh.

```
PSD_Solve -np 2 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0
```

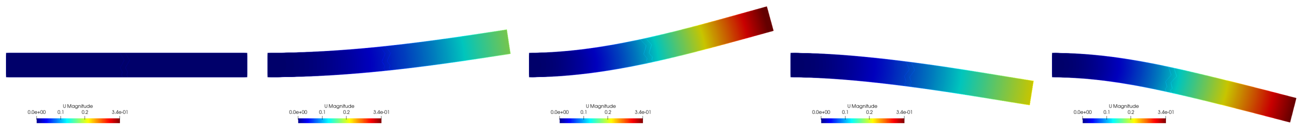


Figure 1: Finite element displacement field on warped mesh shown at different time steps.

Using ParaView for postprocessing the results that are provided in the VTUs... folder, results such as those shown in figure~1 can be extracted.

Parallel 3D

The problem of interest is a single Dirichlet condition (clamped end bar) and traction loading. For this example we use Newmark-β time discretization. Additionally postrocessing is demanded for displacement, acceleration, and velocity (u, a, v).

```
PSD_PreProcess -dimension 3 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization newmark_beta
```

Once the step above has been performed, we solve the problem using four MPI processes, with the given mesh file bar-dynamic.msh.

```
PSD_Solve -np 4 Main.edp -mesh ./../Meshes/3D/bar-dynamic.msh -v 0
```

Sequential problems

To the same problems above Add -sequential flag to PSD_PreProcess for sequential solver, but remember to use PSD_Solve_Seq instead of PSD_Solve. So the work flow for the 2D problem would be:

```
PSD_PreProcess -dimension 2 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization newmark_beta -postprocess uav -sequential
```

Once the step above has been performed, we solve the problem using the given mesh file bar-dynamic.msh.

```
PSD_Solve_Seq -np 2 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0
```

Similarly try out the 3D problem as well.

Different time discretization

PSD offers different time discretization techniques for solving time dependent problems. For this example instead of using Newmark-β time discretization let us switch to more advanced Generalized-α one. This can be done by -timediscretization generalized_alpha, so for example for a 2D problem we use:

```
PSD_PreProcess -dimension 2 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization generalized_alpha -postprocess uav
```

Once the step above has been performed, we solve the problem using three MPI processes, with the given mesh file bar-dynamic.msh.

```
PSD_Solve -np 3 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0
```

Similarly try out the 3D problem as well.

Comparing CPU time

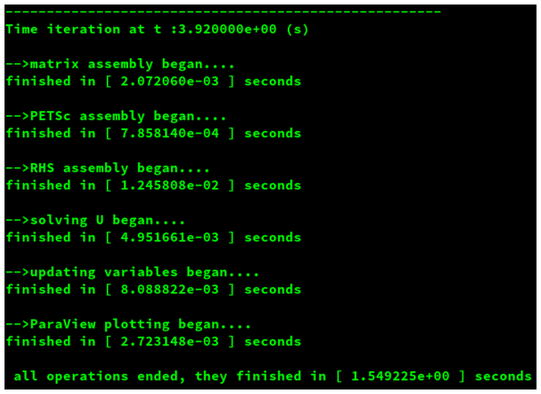
PSD provides mean to time log your solver via -timelog flag. What this will do when you run your solver, on the terminal you will have information printed on what is the amount of time taken by each step of your solver. Warning, this will make your solver slower, as this action involves MPIbarrier routines for correctly timing operation.

An example work flow of 2D solver with timelogging:

```
PSD_PreProcess -dimension 2 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization newmark_beta -postprocess uav -timelog
```

Once the step above has been performed, we solve the problem using two MPI processes, with the given mesh file bar-dynamic.msh.

```
PSD_Solve -np 2 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0
```

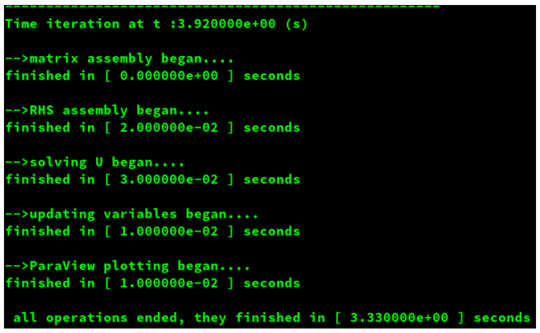


```
-----------------------------------------------------
Time iteration at t :3.920000e+00 (s)

-->matrix assembly began....
finished in [ 2.072060e-03 ] seconds

-->PETSc assembly began....
finished in [ 7.858140e-04 ] seconds

-->RHS assembly began....
finished in [ 1.245808e-02 ] seconds

-->solving U began....
finished in [ 4.951661e-03 ] seconds

-->updating variables began....
finished in [ 8.088822e-03 ] seconds

-->ParaView plotting began....
finished in [ 2.723148e-03 ] seconds

 all operations ended, they finished in [ 1.549225e+00 ] seconds
```

Figure 2: Time logging output produced for parallel run on 2 processes.

The figure~2 shows the time logging output produced for parallel run on 2 processes using -timelog flag. Similar output is produced for sequential solver of the same problem shown in figure~3. Take note of the speed up, which should be two folds - parallel solver solves the full problem in half the time (1.5 sec) than that of sequential solver (3.3 sec). This is due to the fact we used 2 MPI processes.

Also take note of timings produced for different operations of the solver. Note that in figures~2, 3, we only see the final time step of the solved problem.



```
------------------------------------------------------
Time iteration at t :3.920000e+00 (s)

-->matrix assembly began....
finished in [ 0.000000e+00 ] seconds

-->RHS assembly began....
finished in [ 2.000000e-02 ] seconds

-->solving U began....
finished in [ 3.000000e-02 ] seconds

-->updating variables began....
finished in [ 1.000000e-02 ] seconds

-->ParaView plotting began....
finished in [ 1.000000e-02 ] seconds

 all operations ended, they finished in [ 3.330000e+00 ] seconds
```

Figure 3: Time logging output produced for parallel run on 2 processes.

Exercise 1

You are encouraged to try out timelogging and find out if the code (parallel/sequential) is any faster when we use Newmark-β or Generalized-α. Read the documentation for other types of time discretizations that can be performed with PSD, try each one out with -timelog and compare.

Exercise 2

There is a solver run level flag for mesh refinement [1]. This flag is called -split [int] which splits the triangles (resp. tetrahedrons) of your mesh into four smaller triangles (resp. tetrahedrons). As such -split 2 will produce a mesh with 4 times the elements of the input mesh. Similarly, -split n where n is a positive integer produces 2^n times more elements than the input mesh. You are encouraged to use this -split flag to produce refined meshes and check, mesh convergence of a problem, computational time, etc. Use of parallel computing is recommended. You could try it out with PSD_Solve or PSD_Solve_Seq, for example:

```
PSD_Solve -np 2 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0 -split 2
```

for splitting each triangle of the mesh bar-dynamic.msh into 4.

Exercise 3

There is a preprocess level flag -debug, which as the name suggests should be used for debug proposes by developers. However, this flag will activate OpebGL live plotting of the problem, with displaced mesh. You are encouraged to try it out

```
PSD_PreProcess -dimension 2 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization newmark_beta -postprocess uav -timelog -debug
```

Then to run the problem we need aditional -wg flag

```
PSD_Solve -np 2 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0 -wg
```

Exercise 4

PSD comes with additional set of plugins/functions that are highly optimized for performing certain operations during solving. These operations are handled by GoFast Plugins (GFP) kernel of PSD (optimize C++ classes/templates/structures), by default this functionality is turned off and not used. You are encouraged to try out using GFP functions in a solver by using -useGFP flag flag to PSD_PreProcess For example, the PSD solver workflow for the first 2D example in this tutorial would be:

```
PSD_PreProcess -dimension 2 -problem elastodynamics -dirichletconditions 1 -tractionconditions 1 \
-timediscretization newmark_beta -postprocess uav -useGFP
```

Once the step above has been performed, we solve the problem using, with the given mesh file bar-dynamic.

```
PSD_Solve -np 2 Main.edp -mesh ./../Meshes/2D/bar-dynamic.msh -v 0 -wg
```

Try it out for other problems of this tutorial. -useGFP should lead to a faster solver, it might be a good idea to always use this option. To go one step further, use -timelog flag and determine if you have some speed up.

[1] Mesh refinement is performed after partitioning.