Wave simulation performance report

Raivo Laanemets, rlaanemt@ut.ee

Implementation of simulation

Part of algorithm	**Implementing Python class**
Laplacian generation	laplacian.LaplacianGenerator
Simulation loop and step	WaveSimulation
Initial conditions (initial u, u_i values)	InitialConditions
CG algorithm	cg.ConjugateGradientSolver
Sparse matrix	util.SparseMatrix
Sparse matrix with vector multiplication Ax in Fortran	util.SparseMatrixFortran
Python SGS precoditioner	cg.prec.PythonFastPreconditioner
Optimized SGS preconditioner	cg.prec.FortranZeroPreconditioner
Optimized SGS preconditioner with the original Fortran subroutine	cg.prec.FortranPreconditioner
Visualization	vis.EasyvizWaveVisualizator

Test case 1

In this test case we compare the number of iterations for Conjugate Gradient (CG) method with and without Symmetric Gauss-Seidel preconditioner (SGS). The preconditioner class for this test case is *SGSFortranZeroPreconditioner*, which is a Fortran-optimized implementation.

Wave simulation parameters:

Space steps (N)	50
Time steps (S)	300
Simulation time (T)	3.0
CG tolerange	1E-10 (default)

Command line for simulation without SGS:

```
python Main.py -O 3 -T 3.0 -N 50 -S 300 2>&1 | sed -n
's/DEBUG:cg:iterations: \([0-9]*\)/\1/p' | awk '{ s += $1 } END
{ print "total: ", s, " average: ", s/NR, " samples: ", NR }'
```

Command line for simulation with SGS (with k iterations):

```
python Main.py -O 3 -T 3.0 -N 50 -S 300 --sgs=k 2>&1 | sed -n
's/DEBUG:cg:iterations: \([0-9]*\)/\1/p' | awk '{ s += $1 } END
{ print "total: ", s, " average: ", s/NR, " samples: ", NR }'
```

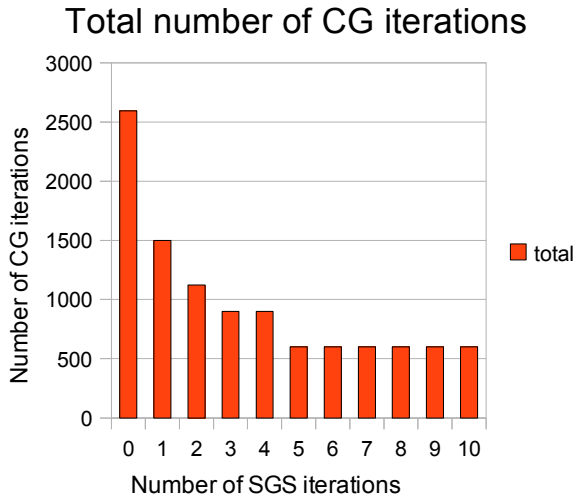
Command line for running time:

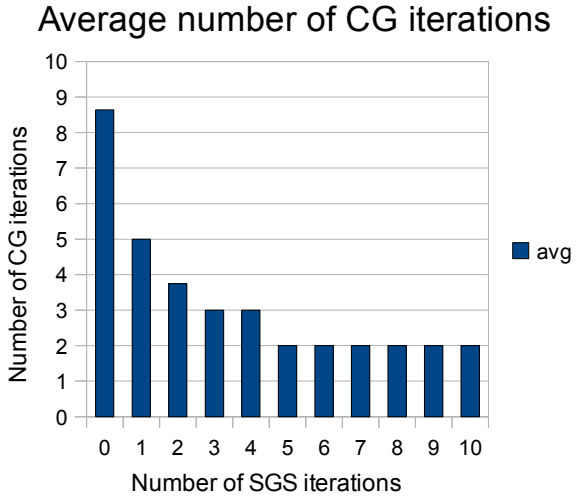
```
python Main.py -O 3 -T 3.0 -N 50 -S 300  2>&1 | sed -n 's/Wall time:
\([0-9]*\)/\1/p'
```

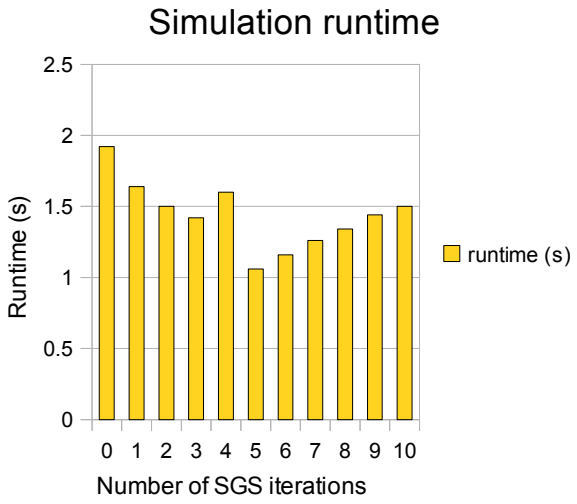
Table of results:

# CG	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
avg	8.64	5.0	3.74	3.0	3.0	2.0	2.0	2.0	2.0	2.0	2.0
total	2593	1500	1122	900	900	600	600	600	600	600	600
runtime (s)	1.92	1.64	1.5	1.42	1.6	1.06	1.16	1.26	1.34	1.44	1.5

With k=100 and k = 1000 the results are same as with k=10, except the running time which increases due to wasted SGS iterations.







Test case 1: conclusions

1. Optimal number of SGS iterations is 5. This minimizes both runtime and the number of needed CG iterations.
2. Using more than 5 SGS iterations is pointless because the number of CG iterations stays constant but the running time will increase due to wasted SGS iterations.

Test case 2

In this test case we measure simulation speedup by optimizing important non-vectorizable code in Fortran.

Wave simulation parameters:

Space steps (N)	50
Time steps (S)	300
Simulation time (T)	3.0

Case 2.1:

Use preconditioning	no
Optimize Ax	no

Command line for case 2.1:

```
python Main.py -O 0 -T 3.0 -N 50 -S 300 2>&1 | sed -n 's/Wall
time: \([0-9]*\)/\1/p'
```

Result: **132 seconds**

Case 2.2:

Use preconditioning	no
Optimize Ax	yes

Command line for case 2.2

```
python Main.py -O 2 -T 3.0 -N 50 -S 300 2>&1 | sed -n 's/Wall
time: \([0-9]*\)/\1/p'
```

Result: **1.9 seconds**

Case 2.3:

Use preconditioning	SGSFastPreconditioner (Python)
SGS iterations	5 (min. number of CG iterations found in test 1)
Optimize Ax	no

Command line for case 2.3:

```
python Main.py -O 0 -T 3.0 -N 50 -S 300 --sgs=5 2>&1 | sed -n
's/Wall time: \([0-9]*\)/\1/p'
```

Result: **340 seconds**

Case 2.4

Use preconditioning	SGSFastPreconditioner (Python)
SGS iterations	5 (min. number of CG iterations found in test 1)
Optimize Ax	yes

Command line for case 2.4:

```
python Main.py -O 2 -T 3.0 -N 50 -S 300 --sgs=5 2>&1 | sed -n
's/Wall time: \([0-9]*\)/\1/p'
```

Result: **316 seconds**

Case 2.5

Use preconditioning	SGSFortranZeroPreconditioner

SGS iterations	5 (min. number of CG iterations found in test 1)
Optimize Ax	no

Command line for case 2.5:

```
python Main.py -O 1 -T 3.0 -N 50 -S 300 --sgs=5 2>&1 | sed -n
's/Wall time: \([0-9]*\)/\1/p'
```

Result: **29 seconds**

Case 2.6

Use preconditioning	SGSFortranZeroPreconditioner
SGS iterations	5 (min. number of CG iterations found in test 1)
Optimize Ax	yes

Command line for case 2.6:

```
python Main.py -O 3 -T 3.0 -N 50 -S 300 --sgs=5 2>&1 | sed -n
's/Wall time: \([0-9]*\)/\1/p'
```

Result: **1.07 seconds**

Case 2.7: dense matrix

This cannot be run on command line. *common.DENSE* must be set *True* in the source code. All command line options about CG will be ignored.

Result: **877 seconds**

Test case 2: conclusions

1. Optimizing both Ax and SGS in Fortran gives the fastest algorithm (1.07 seconds). If we consider the case 2.3 as the base case (Ax and SGS not optimized) then optimizing only SGS gives better result than optimizing only Ax (29 vs. 316 seconds).
2. Running with preconditioner written in Python for the smallest number of CG iterations (2) takes more time than running without preconditioner at all (340 vs. 132 seconds).

Machine details

The tests were run on single-core 1.6Ghz laptop with 2GB of memory and 1MB of cache.