Division of Scientific Computing
Department of Information Technology
Uppsala University

---

Parallel algorithms for Scientific Computing
May 28, 2013

Hands-on and assignment
solving numerical PDEs: experience with *PETSc*, *deal.II* and *Trilinos*

The task is to try some basic functionalities of the packages *PETSc*, *deal.II* and *Trilinos*, collect data for studying the parallel performance of the methods and then analyse it, in alignment with the topics discussed within the lectures.

To play with, you are provided with some ready makefiles and codes in C++ (serial and parallel). These are to be downloaded from `http://user.it.uu.se/~maya/Courses/NLA_Parallel/Files_2013/`.

# 1 Introduction

We consider the classical model Poisson's problem in two dimensions:
Find $u(x, y)$ such that

$$\begin{aligned} -\Delta u &= f, \quad in\, \Omega \subset \mathbb{R}^2 \\ u &= 0, \quad on\, \partial\Omega \end{aligned}$$

where $\Omega = [-1, 1]^2$, and $f = 1$.

We discretize the problem using a quadrilateral mesh (square in this case) and bilinear basis functions.

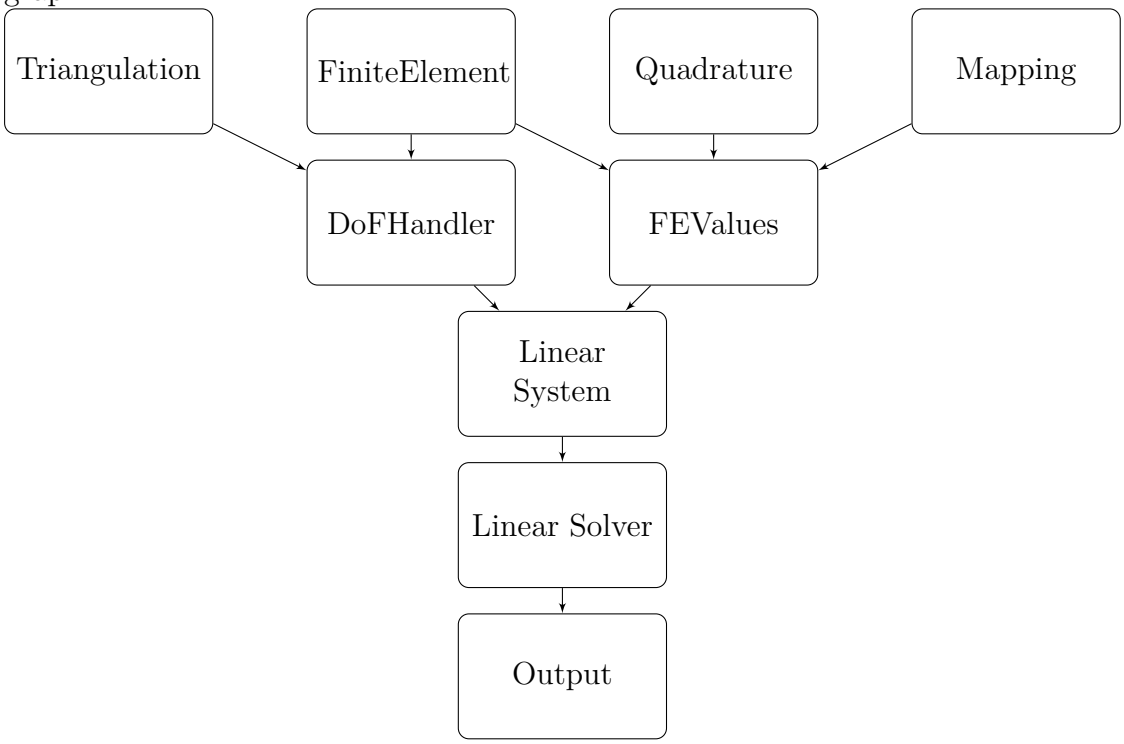
# 2 The blocks of a finite element program

The basic package is deal.ii, which creates the system matrix and the right-hand-side vector. Then, depending on the test case, the solution is dome via PETSc, deal.ii or Trilinos.
The basic structure of a general finite element program to compute the solution of the problem numerically is as follows:

- Create a mesh
- Set up the linear system, i.e., assemble the matrix and the right hand side vector
- Solve the linear system

- Output the solution and some performance results - execution time and iteration counts.

An outline of how the primary groups of classes in deal.II interact is given by the following graph:




To ease the implementation of such a problem, *deal.II*provides a number of building blocks (modules), namely,

- Triangulation: Triangulation objects are used to define and create mesh. The triangulation stores geometric and topological properties of a mesh.
- Finite Element: Finite element classes describe the properties of a finite element space as defined on the unit cell.
- Quadrature: Quadrature objects are defined on the unit cell and only describe the location of quadrature points on the unit cell, and the weights of quadrature points thereon.
- DoFHandler: DoFHandler objects are the confluence of triangulations and finite elements. These enumerate all the degrees of freedom on the mesh and manage which degrees of freedom live where.
- Mapping: Mappings between reference and real cell.

- FEValues: The FEValues classes offer exactly this information: Given finite element, quadrature, and mapping objects.
- Linear Systems: In this module, classes are used to store and manage the entries of associated matrices, vectors, and the solution of linear systems.
- Linear Solvers: This module groups iterative and direct solvers, eigenvalue solvers, and some control classes.
- Output: deal.II generates output files in a variety of graphics formats understood by widely available visualization tools.

More details about these classes can be found at
`http://www.dealii.org/developer/doxygen/tutorial/index.html`

# 3 Tasks

## 3.1 Get started

Log in to the parallel computer tintin and use one of the the projects `p2009040, p2011076, g2013100`. The provided test programs are 'seial.cc' (the sequential code) and 'trilinos.cc' (the parallel code using trilinos) and 'petsc.cc' (the parallel code using PETSc).

## 3.2 Check the *.bashrc* file

The *.bashrc* file has to contain the following lines:

```
module unload pgi
module load gcc/4.7.2
module load openmpi/1.6

export PETSC_DIR=/bubo/home/h28/alid/sw/petsc-3.3-p6
export PETSC_ARCH=arch-linux2-c-debug
export SLEPC_DIR=/bubo/home/h28/alid/sw/slepc-3.3-p3
export METIS_DIR=/bubo/home/h28/alid/sw/metis-5.1.0/

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/bubo/home/h28/alid/sw/deal.II/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/bubo/home/h28/alid/sw/trilinos-11/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/bubo/home/h28/alid/sw/petsc-3.3-p6/$PETSC_ARCH/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/bubo/home/h28/alid/sw/slepc-3.3-p3/arch-linux2-c-debug/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/bubo/home/h28/alid/sw/metis-5.1.0/lib
```

If this is not included in your *.bashrc* add it and restart your connection to tintin.

## 3.3 Compile and run programs

Study the Makefile to see how to compile programs. calling `make` or `make all` will compile all three codes.

### 3.3.1 Serial runs

- Compile the program 'serial.cc',

  ```
  make serial
  ```

  Run the program with $n$ times of mesh refinements,

  ```
  ./serial n
  ```

  Study the program source file and modify the program so that the linear system can also be solved by direct solver and CG solver with AMG preconditioner. Re-run the program and compare with the output.

- Test the behaviour of the three solvers for a different size of the problem. Write down the timing results and compare those with the timings below.

### 3.3.2 Parallel runs

- Compile the program 'trilinos.cc' and 'petsc.cc'

  ```
  make trilions
  make petsc
  ```

  Notice that we use mpicc and mpicxx. The runs are then handled with the command

  ```
  mpirun -n x ./trilinos y
  mpirun -n x ./petsc y
  ```

  requesting $x$ processes to run the program with $y$ times of mesh refinements.

- For the parallel runs we want to see the strong scalability and the weak scalability of the problem. How would you do that?

# 4 Visualization

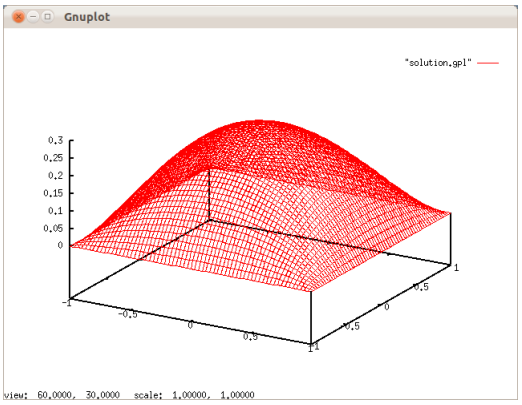
The program 'serial.cc' generates the output file solution.gpl, which is in GNUPLOT format. It can be viewed by typing the following commands:
invoke GNUPLOT:

```
gnuplot
```

type commands at GNUPLOT prompt:

```
gnuplot> set style data lines
gnuplot> splot "solution.gpl"
```

The result looks as follows:
The program 'trilinos.cc' also generates the output file in GNUPLOT format. All processors will write their own files. We could visualize them individually in GNUPLOT. And there is also a whole set of solution. If we test with following command:




```
mpirun -n 2 trilinos 6
```

We will get two individual files written by two processors, namely 'solution-00.00.gpl' and 'solution-00.01.gpl', and a whole solution file, namely 'solution-00.gpl'.
The result looks as follows:

# 5 Assignment

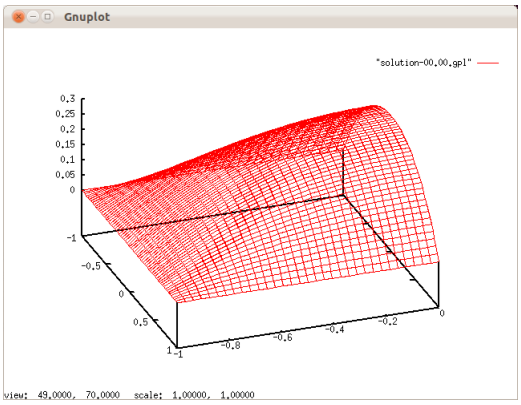
Run all the methods for at least four consecutive refinements in serial and parallel. Choose properly the problem sizes as well as the number of cores to run onto, in order to be able to obtain meaningful results to analyse the performance of the algorithms and the computer. The assignment consists of the following tasks.

1. Study and document the performance of the direct solver. Comment on the observed performance and compare with the theoretically predicted one as described during the lecture on sparse direct solvers. Which solver is used, is there any reordering done? Consult with the documentation for PETSc and deal.ii.

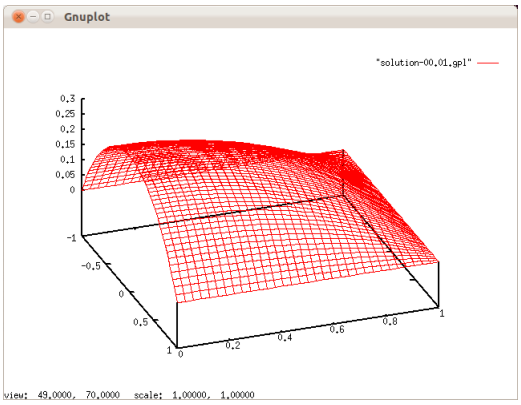
2. Study and document the performance of the unpreconditioned CG using PETSc and deal.ii. Perform some runs in order to show fixed-size and weak scalability figures. Are they as expected? Comment on the condition number of the matrices, what is the theoretical expectation for the convergence of the unpreconditioned CG. Estimate the time for one CG iteration for the various problem sizes. Is it linear in the number of degrees of freedom? Is there a difference between PETSc and deal.ii. Why? Give your arguments.

3. Study and document the performance of the AMG-CG using deal.ii&Trilinos. Do we get the optimal performance of the method? What is the time per AMG-CG iteration, compared with that of one CG iteration? How this ratio varies with the problem size?

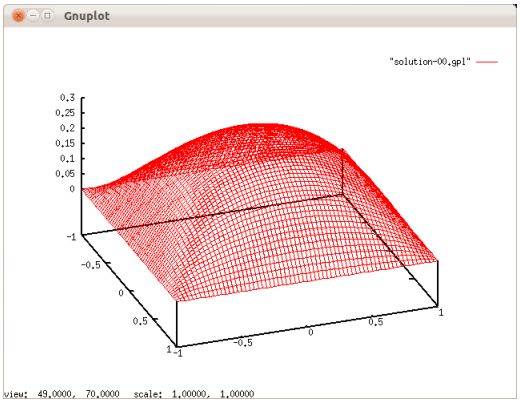
4. Is there any difference in the perforance and scalability when more than one node is used?




(a) processer 1




(b) processer 2

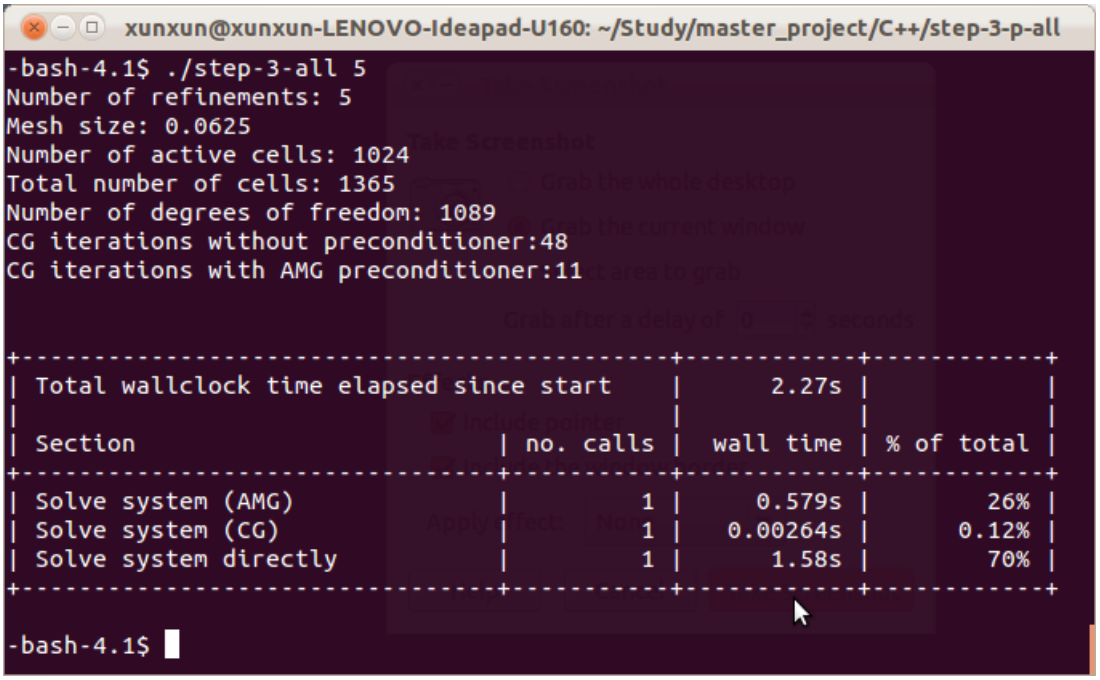



(c) whole solution

A written report on the above should be prepared and sent to Maya Neytcheva not later than June 12, 2013.

# 6 Old results on Kalkul from 2011, for comparisons

The output of the program looks as follows:
In 2011, the same tasks were executed on Kalkul. For the sequential code, the direct solver was the sparse direct solver UMFPACK, using the Unsymmetric-pattern MultiFrontal method and direct sparse LU factorization.
You can use these results for comparisons with the results you obtain on Tintin, where the software packages have been reinstalled and you have the access to their lates versions.
In the parallel code, the direct solver is theKLU direct solver, provided by Trilinos.




| | | Direct | CG | | AMG | |
|---|---|---|---|---|---|---|
| DOF | Mesh Size $h$ | time (s) | time (s) | iteration | time (s) | iteration |
| 4225 | 0.03125 | 0.0438 | 0.00771 | 96 | 0.0346 | 13 |
| 66049 | 0.0078125 | 0.788 | 0.336 | 380 | 0.427 | 13 |
| 1050625 | 0.00195312 | 26.9 | 32.7 | 1446 | 7.07 | 14 |

Table 1: Sequential code

| | | Direct | CG | | AMG | |
|---|---|---|---|---|---|---|
| DOF | Cores $n$ | time (s) | time (s) | iteration | time (s) | iteration |
| 66049 | 1 | 1.57 | 0.734 | 380 | 0.385 | 13 |
| 263169 | 1 | 15.5 | 6.62 | 751 | 1.61 | 13 |
| 1050625 | 1 | 134 | 57.9 | 1446 | 6.89 | 14 |

Table 2: Results obtained from the parallel code on one core (mpirun -n 1)

| | | Direct | CG | | AMG | |
|---|---|---|---|---|---|---|
| DOF | Cores $n$ | time (s) | time (s) | iteration | time (s) | iteration |
| 66049 | 1 | 1.57 | 0.734 | 380 | 0.385 | 13 |
| 263169 | 4 | 15.2 | 2.76 | 751 | 0.484 | 13 |
| 1050625 | 8 | 137 | 22.6 | 1446 | 1.62 | 13 |

Table 3: Results obtained from the parallel code