TAU-Code: Basic Installation and Examples

March 22, 2006

Contents

1 TAU-Code Basics

This user-guide is aimed at describing the basic installation procedure for the DLR TAU-Code, and to provide a short introduction to the handling of the code by way of some simple examples.

1.1 System Preparation

The following software should be installed on the target system in order to successfully compile the TAU-Code.
Required:

- MPICH (version 1.2.4 or higher)
- NetCDF (version 3.4 or higher, version 3.6.0 or higher required for large-file (>2GB) support)

Optional (required for TauPython):

- Python (version 2.3 or higher)
- SWIG (version 1.3.25)

1.2 Installation

1.2.1 .taudef-file

During the TAU-Code compilation the paths to the libraries and header-files of the software listed above need to be defined. The most common way to handle this is to create a file called *.taudef* in the home-directory, where all of the relevant paths are specified, as shown here:

```
NETCDF_INC      = /home/ea44/local/include
NETCDF_LIB      = /home/ea44/local/lib
PARALLEL_INC    = /home/ea44/local/include
PARALLEL_LIB    = /home/ea44/local/lib

PYTHON_INC_FILE = /home/ea44/local/include/python2.4
PYTHON_BIN_FILE = /home/ea44/local/bin/python2.4
SWIG_BIN_FILE   = /home/ea44/local/bin/swig
```

1.2.2 Makefile

In the top-level Makefile the appropriate platform for which the code is to be compiled has to be selected by un-commenting the given line (shown here for a Linux Opteron system):

```
#PLATFORM = LINUX_486_MPI
#PLATFORM = LINUX_586_MPI
PLATFORM = LINUX_opteron_MPI
```

Once the platform has been selected the most straightforward way of creating the TAU-Code binaries is to execute the *make all* command; this creates executables for all of the standard TAU-Code modules in the *bin/* directory.
In order to create the TauPython bindings the command *make main* is used.

2 Example Simulations

The basic process-chain when running a simulation is as follows:

- Create a dualgrid with the Preprocessor (ptau3d.preprocessing <para-file> <log-file>)
- Run the simulation with the appropriate Solver (ptau3d.el <para-file> <log-file>)
- Visualize the results (tau2plt <para-file>)

All parameter and grid files needed to run the examples are available in the tar-file **TAU-basics.tar.gz**.

2.1 euler

In this example a steady, transonic simulation of a 2D, Euler, NACA0012 airfoil is specified. For this simulation the boundary conditions are as follows:

- Reference Mach number: 0.755
- Angle alpha: 2 degrees

A multigrid cycle of 4w is specified, and the maximum number of iterations is set at 200. Figure 1 shows the convergence for this simulation, and Figure 2 shows the iso-lines of density.

2.2 navier

In this example a steady, subsonic simulation of a 2D, Navier-Stokes, NACA64A010 airfoil is specified. For this simulation the boundary conditions are as follows:

- Reference Mach number: 0.3
- Reynolds number: 7500000

A multigrid cycle of 4w is specified, and the maximum number of iterations is set at 200. The default values for the 1-equation Spalart-Allmaras turbulence model are used. Figure 3 shows the convergence for this simulation, and Figure 4 shows the iso-lines of density.

2.3 parallel

This example is based off of the *navier* simulation, using exactly the same parameters except for the number of domains used. Since we want to run both the Preprocessor and the Solver in parallel, the process-chain now looks like the following:

- Partition the primary grid (ptau3d.subgrids <para-file> <log-file>)
- Create a dualgrid with the Preprocessor (mpirun -np 2 ptau3d.preprocessing <para-file> <log-file> usempi)
- Run the simulation with the appropriate Solver (mpirun -np 2 ptau3d.turb1eq <para-file> <log-file> usempi)
- Visualize the results (tau2plt <para-file>)

Figure 5 shows the convergence for this simulation.

2.4 unsteady-euler

In this example an unsteady, pitching-oscillation simulation of the *euler* case is specified. The unsteady simulation uses the results of the steady simulation as a restart condition. Figure 6 shows the history of the lift-coefficient against the angle-of-attack.

2.5 deformation

In this example a flight-shape simulation is done using the AMIF file-format to exchange data (in ASCII format) between the 'Deformation-Preprocessor-Solver' chain and a *very* low-fidelity surface deformation function. The grid is a fairly coarse Euler-grid of a delta-wing with a sting. This example requires TauPython and 'tau2plt' in order to function, and a TAU-Code release 2006.1.0 or newer. The process-chain for this example is as follows:

- Preprocessor creates dualgrid from initial primary grid
- Solver creates an initial steady solution as start-condition for the unsteady loop
- tau2plt writes ASCII file containing point-ids, coordinates and forces
- Unsteady simulation-loop begins with Deformation reading the ASCII file, followed by Preprocessing of the new primary grid and a Solver run using the new dualgrid information, and tau2plt creating a new ASCII file based off of the latest results.

The simulation is launched with the following command-line arguments:
tau.py script_deformation.py para_delta log

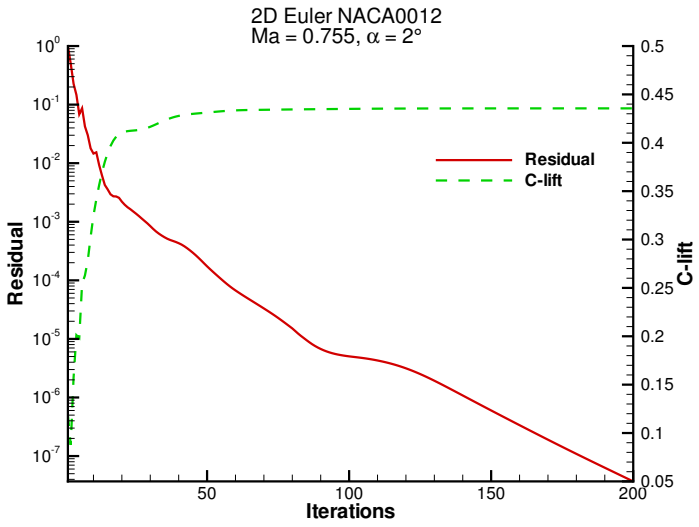

Figure 1: 2D Euler NACA0012: Convergence history

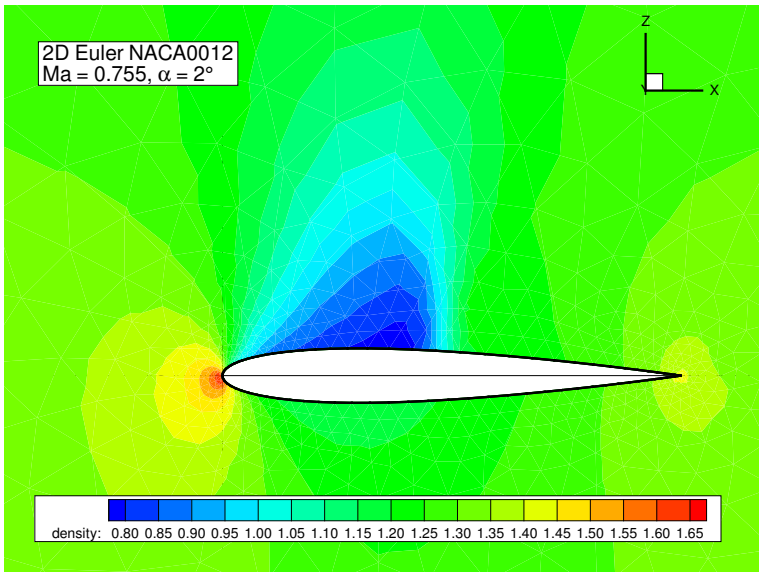

Figure 2: 2D Euler NACA0012: Density iso-lines

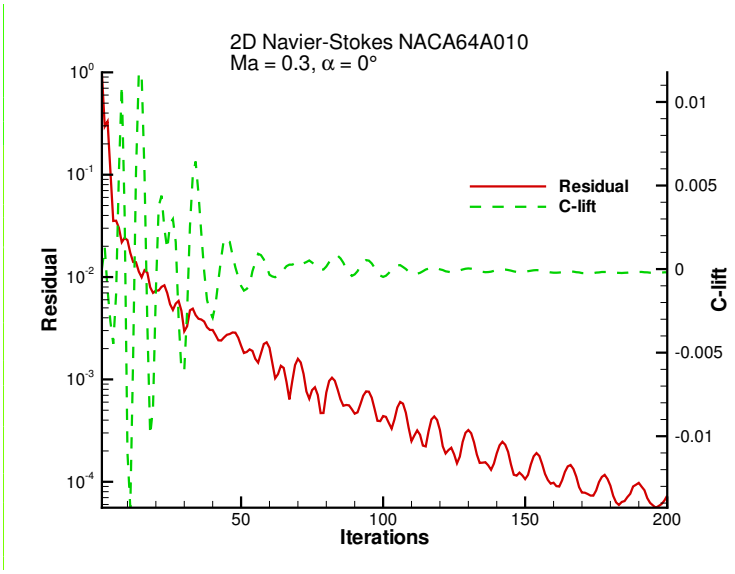

Figure 3: 2D Navier-Stokes NACA64A010: Convergence history

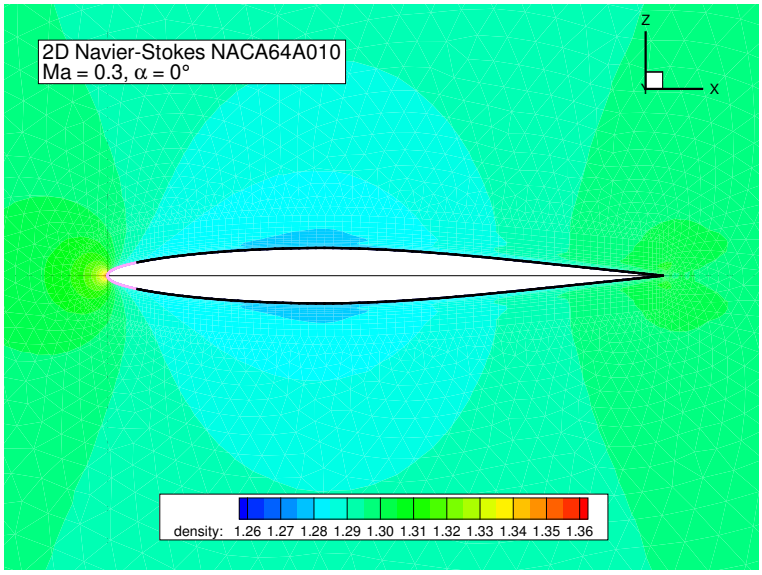

Figure 4: 2D Navier-Stokes NACA64A010: Density iso-lines

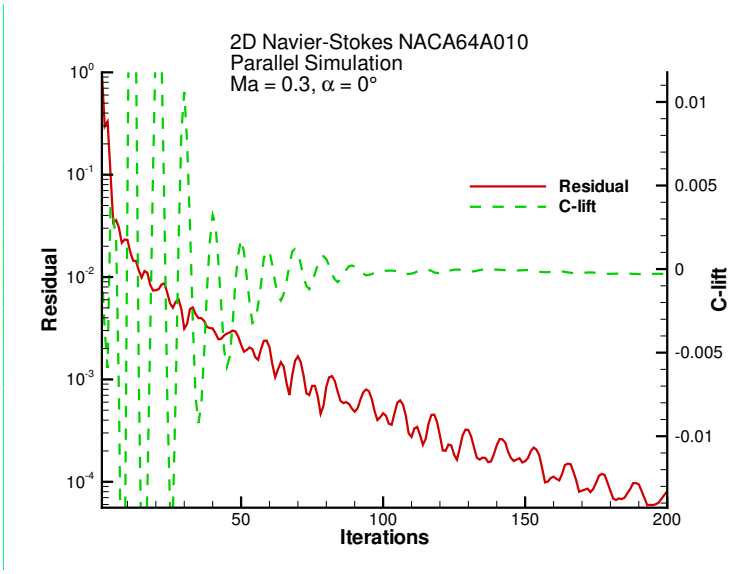

Figure 5: 2D Navier-Stokes NACA64A010 - Parallel: Convergence history

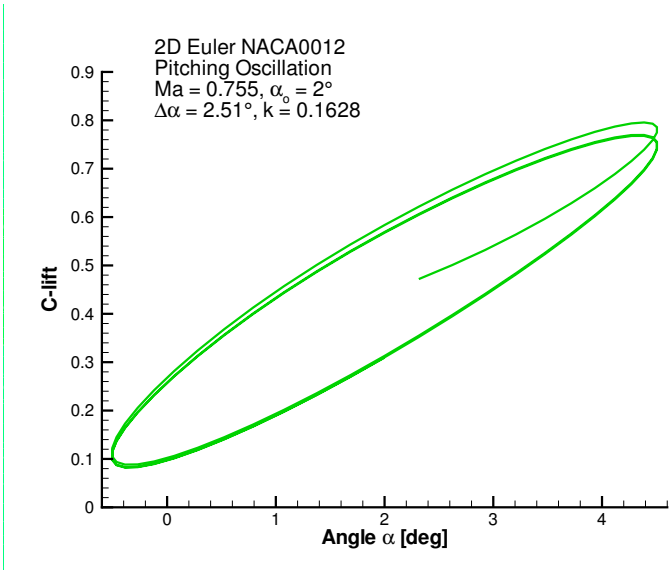

Figure 6: 2D Euler NACA0012 - Pitching Oscillation: Lift-coefficient vs. angle-of-attack