PRACE Workshop, Worksheet 2

Stockholm, December 3, 2013.

0 Download files

http://csc.kth.se/∼rvda/prace_files_ws2.tar.gz.

1 Introduction

In this exercise, you will have the opportunity to work with a real research problem faced by our group, where the use of parallel computers was the cornerstone of the solution process.

The exercise is divided into four parts:

- pre-processing [1], where you will need to set up the solver and its main parameters (such as e.g. length of simulation, boundary conditions, etc);
- the actual "simulation part", where you will run a small sample of the total problem with the flow solver Unicorn, [1], on the supercomputer Lindgren at the Center for High Performance Computing (PDC);
- post-processing, where key quantities will be extracted from the solution files, using the visualization softwares Paraview and Visit.
- formulation and solution of an adjoint problem for automatic mesh generation.

The three first items represent the usual workflow of a CFD engineer. In the last item, a key feature of Unicorn is highlighted and you will have the oppotunity to get acquainted with the adjoint problem formulation for automatic mesh refinement.

[1] In the pre-processing stage, it is assumed that the mesh has already been generated.

2 Problem Statement, the “Rudimentary Landing Gear” (RLG)

The RLG geometry was developed at Boeing Commercial Airplanes by Philippe Spalart and is a non-proprietary, highly simplified version of a four-wheel landing gear, Figure 1. The geometry was used to formulate one of the problems, [2], in the BANC-I NASA/AIAA workshop. The RLG is non-proprietary and allows for unrestricted sharing of data and open publication of results.

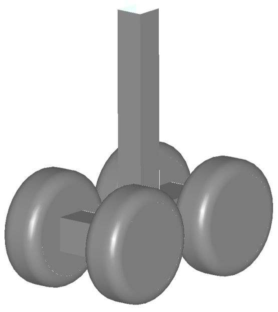

Figure 1: Rudimentary Landing Gear (RLG).

The objective in the workshop was to calculate surface pressure distributions, both average and “rms”, to compute the total aerodynamic forces on the RLG, and to visualize the flow dynamics on the RLG surface and in the wake. For a description of the simulation results obtained with Unicorn, and detailed comparisons of these with experiments, see [3].

3 To-do

In the following paragraphs, you will find a number of tasks that simulate the complete workflow when using Unicorn as a CFD solver.

3.1 Pre-processing

Task 1: Compile the code by typing “make” under **.../icns/rlg/**. If your environment is correctly configured, the code should compile with no errors. If you run this simulation, however, your code will not work, since boundary conditions (BC) have been wrongly specified (on purpose) in the file **main.cpp**.

Start by setting boundary conditions. Figure 2 shows a sketch representation of the domain, where you may find the geometrical features needed to define inflow, outflow, and the remaining BC. Notice that the geometry has been scaled with the wheel diameter, D (the mesh file you will work with, **rlg_mesh.xml**, has also been scaled).

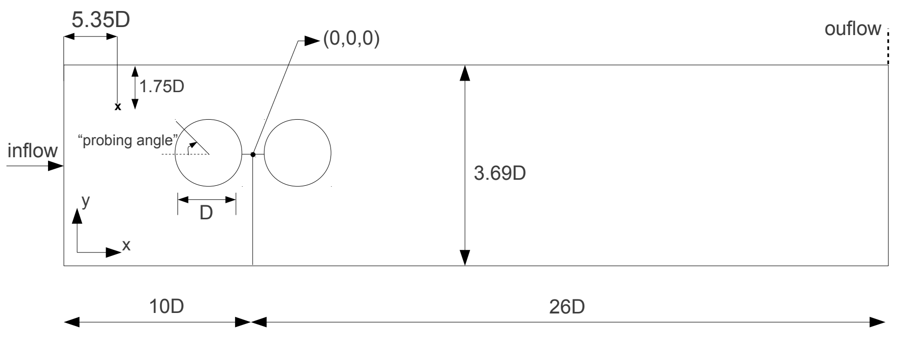

Figure 2: Geometry sketch, the RLG is mounted upside down on the wind-tunnel floor.

Task 2: Modify the file **.../icns/rlg/main.cpp** to add in- and outflow BC. To do so, look how these are implemented in **.../icns/cylinder/main.cpp**, where a similar problem is formulated for a circular cylinder inside a wind-tunnel.

The inflow condition is a *Dirichlet* BC on the velocity, $U = 1$, while the outflow is a *Neumann* BC for the stress, $\nu \nabla U \cdot n - pn = 0$, where ν is the kinematic viscosity, n is the surface outward normal and p is the manometric pressure inside the wind-tunnel. For the outflow, it is assumed that there are no significant velocity gradients in the flow direction, and the Neumann condition reduces to a homogeneous Dirichlet condition for the pressure, $p = 0$.

The remaining BC, for the tunnel walls and the RLG surface, are set to a no-penetration, slip with friction BC. For more information on this type of BC, see, e.g., section 28.13 in [4].

Task 3: Set simulation parameters (such as total simulation time, number of solution samples and inflow velocity) in the file **.../icns/rlg/parameters**.

If you are interested in dynamic quantities that change rapidly with time (e.g., vorticity, turbulent structures), you will need to have a relatively high sample rate. It is not unsual that the number of samples for the entire simulation is set to $> 1,000$ or even $> 10,000$. This is particularly useful if a movie will be produced to visualize the dynamics in the result. With the use of *MPI/IO* it becomes extremely cheap to write solution files to the disk and sampling frequently does not slow down the simulation.

One important parameter in the **parameters**-file is the output format of the simulation. You can set it either to **vtk** or **binary**. The **binary**-format is strongly recommended, especially for simulations running on a large core-count. Why? Because writing files in **vtk**-format in parallel requires one file *per core, per sample*, while the **binary**-format requires only one file *per sample.* For example, if you are running a simulation on $1,200$ cores with a total of $5,000$ samples, you will end up with $6,000,000$ files, if using **vtk**!

Task 4: Make sure that the parameter **output_format** is set to **binary** in **.../icns/rlg/parameters**.

3.2 Simulation

Now, to run the RLG simulation, if you have been working on your local computer, you will need to transfer your files and login to Lindgren. See **http://www.pdc.kth.se/resources/computers/lindgren** for instructions.

Task 5: Now that you have edited your **main.cpp**-file, compile the code again. This is sometimes a trial and error task; if the code does not compile, try to add one boundary at a time, e.g., first add the inflow, and then compile. If it works, then add the outflow and compile once more.

Once the code is correctly compiled on Lindgren, run it by typing (from the directory where you have your executable, your mesh file and your **parameters**-file):

```
> aprun -n 96 ./rlg -p parameters -m rlg_mesh.xml
```

The output printed during execution shows information that helps the user to understand the current status of the simulation, e.g., the time-step size, residuals, etc. You may, alternatively, print the output to a **log**-file, and print its content with the command **tail**:

```
> aprun -n 96 ./rlg -p parameters -m rlg_mesh.xml > log &
> tail -f log
```

During the execution, the result files are saved under the directory **.../iter_0**. Check its content to see if you get any samples.

Suppose now you have been running a simulation for a few hours and the computer, for unexpected reasons, stops working. Or, an even more common situation, suppose you need to run a simulation for a longer time than the maximum time allowed by the administrators of the batch queue. There is a standard solution for both these cases in Unicorn [2]. It is called *checkpoint restart.*

Task 6: Check the contents of the directory **.../iter_0**. Do you see any file with the extension **.chkp**? If so, restart the simulation from one of the files by changing the argument **-m rlg_mesh.xml** to **-c iter_0/primal0** (or **-c iter_0/primal1**) in the **aprun**-call.

3.3 Post-processing

There are several interesting quantities one can extract from the simulation output, but first you will need to make it "readable" to your post-processing tool (if the the **binary** format was used, otherwise you can read your **vtk**-files directly with Visit or Paraview).

The conversion is simply done using the program **dolfin_post**, which is available on Lindgren. As an example, suppose you have a total of $5,000$ **binary**-files you want to convert to **vtk**-files. From under the directory where you have the files, type:

```
> dolfin_post -m mesh.bin -t vtk -n 5000 -s primal_solution
```

Since **dolfin_post** in executed in serial, it produces only one **vtk**-file per sample. Once the conversion process is finished, you are ready to produce some beautiful and insightful visualizations.

Task 7: Using Paraview and/or Visit, try to reproduce the visualization in Figure 3 (you may find a sample dataset under **.../rlg_bin_files/**). In Paraview, e.g., you will need the filters *Extract Surface* and *Extract Cells by Region* to extract the RLG geometry, and the filter *Temporal Statistics* to compute the time-average. Make sure to convert the files with **dolfin_post**

[2]more precisely, in Dolfin-HPC, on which Unicorn relies.

before doing your visualization. If using the sample dataset, call **dolfin_post** with the options **-o 2050** and **-f 100**, since you want to start from sample $2,050$, and you only want to extract every 100^{th} file:

```
> dolfin_post -m mesh.bin -t vtk -n 10 -s primal_solution -o 2050 -f 100
```

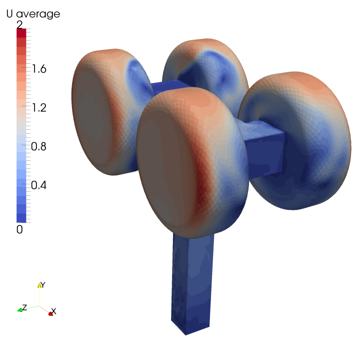

Figure 3: Average velocity field.

3.4 Adjoint based automated mesh generation

In Unicorn, an adjoint problem for automated mesh refinement is implemented for the Navier-Stokes equations. The default refinement target is the drag force, i.e., the mesh is successively refined such that the error in the computed drag (with respect to the discretization) is minimized. For a detailed description of the adjoint problem formulation, see chapter 33 in [4].

Task 8: A **main.cpp**-file where the adjoint problem for the RLG is formulated (and which contains the answers to "Task 2") may be found under **.../icns/rlg_adjoint/**. Browse through that file and see if you can find what the BC are for the adjoint problem. Moreover, identify the source term for that problem and see if there is any relation to the drag force.

Now you can compile and run the code again.

This code contains an implementation of a Navier-Stokes solver with its corresponding adjoint problem. When the program is executed, it will first solve the Navier-Stokes equations and then its adjoint. At the end of the solution of the adjoint problem, you will get a refined mesh saved under the directory **.../icns/rlg_adjoint/iter_1/**. This process constitutes an *adaptive iteration* and may be repeated several times, until mesh convergence is obtained. You can set the number of iterations by changing the parameter **adapt_iter** in the **parameters**-file.

Finally, the full *adaptive algorithm* for automated mesh generation reads:

1. For a given mesh, $\mathcal{T}_n$: compute the primal problem and the adjoint problem.
2. Mark the 10% of the elements with highest error indicator for refinement.
3. Generate the refined mesh, $\mathcal{T}_{n+1}$, and goto 1.

Task 9: Under **.../icns/rlg_adjoint/** you can find a coarse version of the RLG mesh. Change the parameters file and set the simulation time for the primal solver to 5.0 (parameter **T**) and the time for the adjoint solver to 1.0 (parameter **dual_T**). Now you should be able to run at least 1 iteration of the refinement algorithm above. Do that and look at the solution. Does the adjoint solution "flow" in the correct direction?
OBS! Sometimes the output generated by the code can be too fast (and difficult to read). In such case, try:

```
> aprun -n 96 ./rlg -p parameters -m rlg_mesh_coarse.xml > log&
> tail -f log |grep Fix-point
```

There are several interesting output quantities in the folder **.../iter_0/**, which are saved during (and after) the solution of the adjoint problem. Apart from the adjoint solution itself (saved to the files **dual*.bin**), you will also find the residuals (**residual*.bin**), the error indicators (**residual*.bin**) and the cells marked for refinement (**marked*.bin**). These quantities, although lacking a direct "physical" meaning, can give good insight in the physics of the problem... But this could be the topic for a whole new course!

Task 10: If you have time, try to visualize some of the outputs of the adjoint problem. The adjoint solution may be sometimes beautifully visualized with volume rendering...

References

[1] Hoffman, J., Jansson, J., Vilela de Abreu, R., Degirmenci, N. C., Jansson, N., Müller, K., Nazarov, M., and Spühler, J. H., "Unicorn: Parallel adaptive finite element simulation of turbulent flow and fluid-structure interaction for deforming domains and complex geometry," *Computers & Fluids*, Vol. 80, No. SI, 2013, pp. 310–319.

[2] Spalart, P. R. and Mejia, K., "Analysis of Experimental and Numerical Studies of the Rudimentary Landing Gear," *Proceedings for 49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Orlando, Florida*, 2011.

[3] Vilela de Abreu, R., Jansson, N., and Hoffman, J., "Adaptive Computation of Aeroacoustic Sources for a Rudimentary Landing Gear," *International Journal for Numerical Methods in Fluids*, Vol. Published online in Wiley Online Library, DOI: 10.1002/fld.3856, 2013, http://dx.doi.org/10.1002/fld.3856.

[4] Hoffman, J. and Johnson, C., *Computational Turbulent Incompressible Flow: Applied Mathematics Body and Soul Vol 4*, Springer-Verlag Publishing, 2006.