Tutorial/Report in OpenFoam Course 2008

Implementing third order compressible flow solver for hexahedral meshes in OpenFoam

Martin Olausson,
Chalmers University of Technology, SE-412 96 Gothenburg, Sweden

Abstract

A new solver for unsteady compressible flow is implemented in OpenFoam. The solver, called "g3dFoam", is based on the G3D family of codes and uses standard third order upwind scheme for the inviscid flux and a three stage Runge-Kutta technique to update the solution. This method has successfully been used for high speed aeroacoustic computations in turbomachinery applications. The implementation is done using the mesh class functions to loop over faces and cells to compute fluxes. The limitation of this new solver is to hexahedral meshes where an opposing face can be found for all faces in a cell. Results from the "shockTube" case is presented and compared with existing conservative compressible solvers in OpenFoam.

Introduction

A need for higher order schemes for compressible flow calculations in OpenFoam has been identified. The main reason for this need is a possibility to do aeroacoustic calculations, i.e. noise predictions of turbomachinery and jets etc, where lower order schemes require too large cell density to capture the higher frequency pressure waves. A lot of aeroacoustic work has been done with the G3D family of codes.[1, 2, 3, 4, 5] These are written in Fortran and handles only block structured meshes, but are still state of the art in many aeroacoustic applications. Here there is a need for a more modern platform, and this work can be seen as an attempt to transfer technology from G3D to a more modern platform like OpenFoam.

A compressible flow solver for high speed air flow of Mach numbers around 0.5 to 1.5 must also be able to capture shocks in a correct way. Therefore it is crucial that the equations are of conservative form. The available solvers in OpenFoam that are of conservative form are "rhoSonicFoam" and "rhopSonicFoam" and they are used as reference. Both solver looks very much alike in the beginning of the code, where the continuity equation (1), the momentum equation (2), and the energy equation (3) are solved using the function "solve" for each equation.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \tag{1}$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot (\rho U U) = -\nabla p \tag{2}$$

$$\frac{\partial (\rho e_0)}{\partial t} + \nabla \cdot (\rho U e_0) = -\nabla \cdot (pU) \tag{3}$$

These are the Euler-equations, i.e. the Navier-Stokes equations without viscosity. The difference between the two solvers is that "rhopSonicFoam" uses a more advanced so called "MUSCL"-scheme, with a lot of corrections and limiters etc. The ideal gas law (4) is used to relate pressure with density and temperature.

$$p = \rho R T \tag{4}$$

The energy equation solves for ρ times total energy, and the definition is (5).

$$\rho e_0 = \rho C_v T + \frac{\rho |U|^2}{2} \tag{5}$$

Equation (6) holds for a thermally perfect gas, but C_p is assumed to be constant, and therefore the fluid is a calorically perfect gas.

$$C_p - C_v = R \tag{6}$$

The newly implemented solver called "g3dFoam" uses the same assumptions and solves the same equations as described above.

g3dFoam.C

The G3D family of codes, that this new solver is based on, solves the compressible flow equations in conservative form on a boundary-fitted, curvilinear non-orthogonal multi-block mesh.[6] It typically uses two cells on each side of a face to do a reconstruction of the inviscid flux over the face, i.e. a total of four cells for each face as shown in figure 1. This is easily done in a structured mesh since indices (i,j,k) tells you where you are and where your neighbours are in each direction. OpenFoam on the other hand uses unstructured meshes and the index of a cell or a face does not tell you where you are directly. In OpenFoam each face has an owner and a neighbour cell and they are easily found by using the functions "mesh.faceOwner()" and "mesh.faceNeighbour()". The cells next to the owner and neighbour cell can be located by the "opposingFaceLabel"-function. First the opposing face is found and then either the owner or neighbour of this face is the cell of interest. This is done before the calculations start and the data is saved in a "labelList", i.e. a face-owner-owner and a face-neighbour-neighbour list. It requires that an opposing face can be identified, and this can be done for an unstructured mesh if it only consist of hexahedrons.

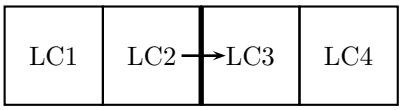

Figure 1: The flux over each cell face is calculated using four cells, two on each side. They are named LC1, LC2, LC3, LC4 in the code, and correspond to face-owner (LC2), face-neighbour (LC3), face-owner-owner (LC1), and face-neighbour-neighbour (LC4).

Now when the owners owner and the neighbours neighbour are located for all the faces in the mesh the computation can start. First some fields are created to collect the fluxes of the conservative variables: Drho, DrhoU, and DrhoE. These are cell time derivatives of ρ, ρU, and ρe_0. Then the temporary storage for the conservative variables that is used in the Runge-Kutta loop is created: rho0, rhoU0, and rhoE0. The time loop that follows consist of the three stage Runge-Kutta cycle where a file named "flux.H" is included three times. In that file the flux over the faces is calculated and the result is put in the time derivative fields.

The flux in the interior is calculated first. That means all faces that have two cells on each side inside the domain. The faces on the boundary and one step into the domain has to be treated specially. The file "flux.H" begins with a loop over all internal faces, and inside that loop a file named "conv_interior.H" is included. This is where the flux over an internal face is calculated. A check has to be done since internal faces in OpenFoam are all faces that don't belong to a boundary. If a face has an opposing face at a boundary, that face is skipped. It will be treated later as a boundary anyway. Inside "conv_interior.H" all the variables needed to do a third order upwind biased reconstruction on the face is extracted. The "mesh.Sf()"-function is used to get the face area normal vector, and "mesh.magSf()" is used to get the area. Then the file named "cflux1.H" is included, and this is where the reconstruction is done.

The reconstruction starts with calculating left- and right-biased ρ, U, and p at the face using all four cells and the user defined coefficients. Then an average of ρ, U, and c (speed of sound) is calculated using only the face owner and neighbour. These average values is used to calculate

the eigenvalues (actually only the sign of the eigenvalues), to determine in which direction the information goes. The sign of the five eigenvalues indicates in which direction the corresponding waves travel. The first one is the entropy wave, the second and third are vorticity waves, and the fourth and fifth are pressure waves. Then either the left or right biased ρ, U, and p is selected to calculate the characteristic variables on the face depending on if the information comes from the left or from the right. When the characteristic variables on a face are obtained they are transformed back to primitive variables (ρ, U, and p). After that, pressure and density controlled damping is calculated. This is needed when strong shocks and contact discontinuities are present in the solution. The fluxes are then calculated and added to the time derivative fields after multiplying with the inverse of the volume, using "mesh.V()" to get the volume of the cell.

Going back to the file "flux.H", the next thing to do after calculating the interior flux is to handle boundaries. So far only wall b.c. is implemented. It ensures that the normal velocity is zero on the wall and that the gradient of pressure and temperature is zero by setting ghost cells to corresponding values and then using "cflux1.H" to obtain the flux.

Running g3dFoam on the shockTube case

- First you should copy/extract the "g3dFoam"-folder to your local OpenFoam "applications" directory, and then type "wmake" to compile g3dFoam. When compiled, g3dFoam should be available to use as a solver.
- Now copy/extract the "shockTube"-folder to your local OpenFoam "run" directory. This is the standard shockTube case with some minor changes.
- Run "blockMesh" on the "shockTube" case to create the mesh. In the "boundary" file, located in "shockTube/constant/polyMesh/", the "physicalType" of the sides are set to "wall". This is what g3dFoam is looking for to find the walls in the domain. In the "blockMeshDict" file, located in the same directory, the number of cells in the domain is changed from 1000 to 200, to better see how the solver works.
- Run "setFields" on the "shockTube" case. This sets the piecewise constant initial solution, as specified in the file "setFieldsDict" located in "shockTube/system/". The files "p" and "T", located in the "shockTube/0/" directory are changed.
- Run "g3dFoam" on the "shockTube" case. The new solver "g3dFoam" now solves the shocktube case. Time step size and end time etc. are specified in the file "controlDict", located in the "shockTube/system/" directory. The compressible CFL-number should not be larger than unity for this solver to be stable, and the specified Δt does not yield a too large value. The coefficients for the numerical scheme are read from the file "G3Dcoefficients" located in the folder "shockTube/system/". The four specified coefficients gives a standard third order upwind biased scheme. The other two variables controls extra pressure and density based numerical dissipation. These can be set to zero if no shocks or contact discontinuities are present in the solution.
- Run "magU" on the "shockTube" case. This adds magU to the solution.
- Run "sample" on the "shockTube" case. This samples the solution and puts the result in the "shockTube/samples/" folder.
- Plot the solution using "xmgrace -nxy shockTube/samples/0.007/data_T_magU_p.xy". This plots temperature, velocity, and pressure in the same window.

Results

The new "g3dFoam" solver was compared to the two existing conservative compressible solvers "rhoSonicFoam" and "rhopSonicFoam", using the same mesh and the same time step size for the "shockTube" case. This is a one dimensional unsteady case where a discontinuity, normally in pressure, is introduced in the middle of the domain. Many important flow-features of compressible flow are obtained shortly after the computation has started. A shock will form and move into the low-pressure region, an expansion fan will move into the high-pressure region, and a contact discontinuity will form in the middle. This can be solved analytically, and therefore it's a good reference case.

The setup and execution time is shown in Table 1. It can be seen that "rhoSonicFoam" and "g3dFoam" has approximately the same execution time, but "rhopSonicFoam" is much slower for the same setup.

	g3dFoam	rhoSonicFoam	rhopSonicFoam
number of cells	200	200	200
Δt	$4 * 10^{-5} s$	$4 * 10^{-5} s$	$4 * 10^{-5} s$
CFL number	~ 0.6	~ 0.6	~ 0.6
end time	$0.007 s$	$0.007 s$	$0.007 s$
execution time	~ 0.51	~ 0.51	~ 1.3

Table 1: Setup and comparison between the solvers.

The results from the simulations are shown in Figure 2. It can be seen that there are some fluctuations around the shock and the contact discontinuity for "g3dFoam", but the expansion wave agrees well with the exact solution. For "rhoSonicFoam" on the other hand there are fluctuations around the shock and the tail of the expansion wave, and the contact discontinuity is not as sharp as in the "g3dFoam" solution. In the solution from the solver "rhopSonicFoam" there are no fluctuations but instead it misses a bit on the head and tail of the expansion wave compared to the exact solution.

Conclusions and future work

This implementation of a new solver is not really done in an OpenFoam-ish way, but it works and shows promising results both in terms of accuracy and speed. It lacks necessary boundary conditions such as inflow/outflow, and it has not been tested for 2d and 3d cases or parallel run. A viscous model is also needed but that can probably be copied from an existing compressible viscous solver.

The overall impression of OpenFoam is that it is relatively easy to implement a new solver this way, but to make it fit in the template-class-hierarchy-"jungle" seems to be a lot more work. It would of course be good to implement this scheme the way other schemes are implemented in OpenFoam, because then it would be much easier to reuse it and modify it to fit other cases. The final goal with this work would be to have a fully unstructured solver, but then the reconstruction has to be done in a different way.

Another impression is that OpenFoam is mostly adapted to incompressible solvers, where the equations can be separated. This is obvious when looking into how boundary conditions are set. The initial solution and the boundaries are specified in the same file. In many compressible simulations the boundary conditions are not specified the same way as the initial solution, i.e. an inlet can be specified with total temperature and total pressure while the initial condition might be static temperature, static pressure, and velocity.

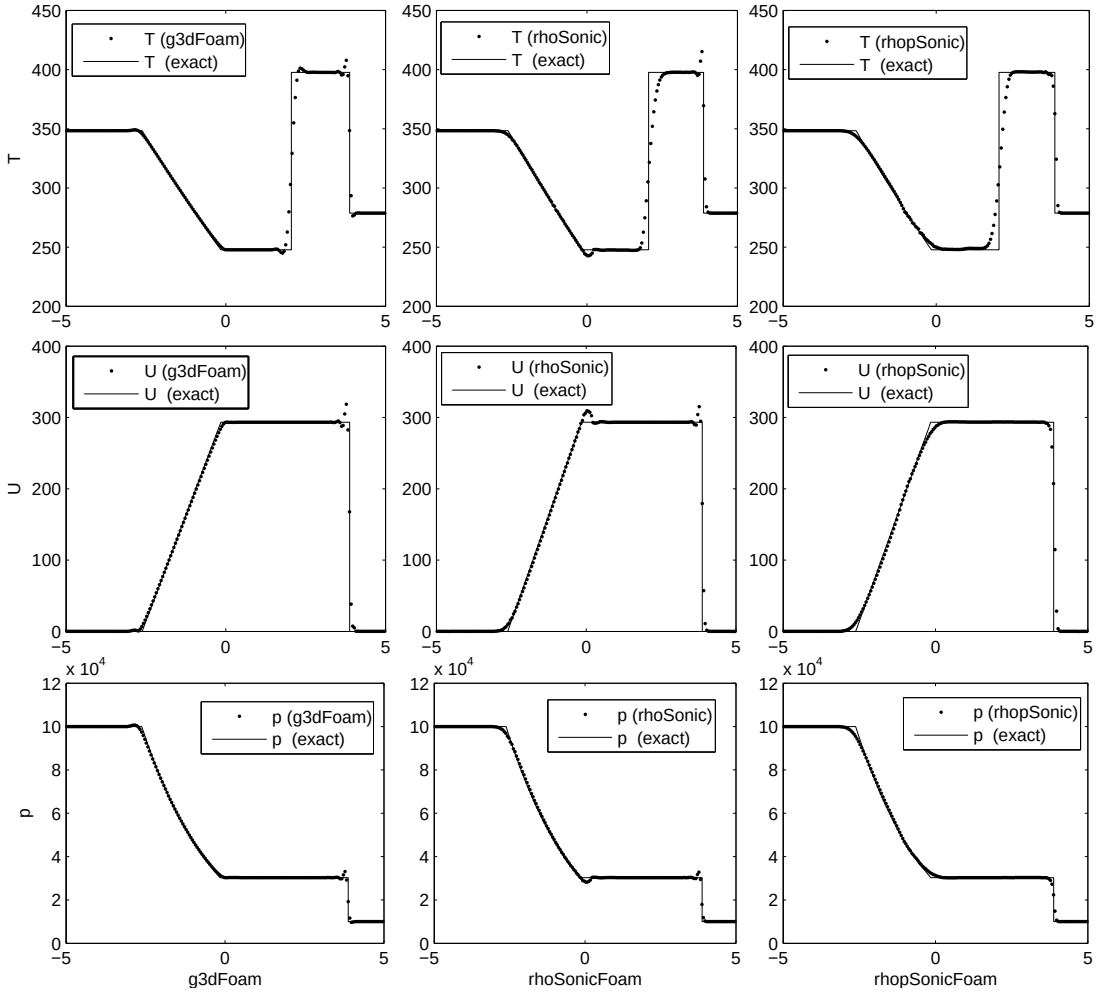

Figure 2: Comparison between g3dFoam (left), rhoSonicFoam (middle), and rhopSonicFoam (right) for the shockTube case, looking at temperature (top), velocity (middle), and pressure (bottom), at time 0.007 s. The number of cells are 200 and the compressible CFL number are about 0.6 for all simulations. An expansion wave is located to the left and a shock to the right. The contact discontinuity can be seen in the middle in the temperature plots.

References

[1] Baralon, S., Eriksson, L.-E., Billson, M., and Andersson, N., "Evaluation of Advanced Prediction Methods for Aero Engine Exhaust Noise," No. 1190, The 17th ISABE meeting, Munich, Germany, 2005.

[2] Burak, M. and Eriksson, L.-E., "Flow and noise predictions for a mixer-ejector engine configuration using LES," The 13th AIAA/CEAS Aeroacoustics Conference, Rome, Italy, 2007.

[3] Olausson, M., Eriksson, L.-E., and Baralon, S., "Nonlinear Rotor Wake/Stator Interaction Computations," No. 1307, The 18th ISABE meeting, Beijing, China, 2007.

[4] Andersson, N., *A Study of Subsonic Turbulent Jets and Their Radiated Sound Using Large-Eddy Simulation*, Ph.D. thesis, Division of Fluid Dynamics, Chalmers University of Technology, Gothenburg, 2005.

[5] Billson, M., *Computational techniques for turbulence generated noise*, Ph.D. thesis, Division of Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 2004.

[6] Eriksson, L.-E., "Development and Validation of Highly Modular Flow Solver Versions in G2DFLOW and G3DFLOW," Internal report 9970-1162, Volvo Aero Corporation, Sweden, 1995.