Theoretical studies and numerical experiments suggest that high-order methods for unstructured grids can solve hitherto intractable fluid flow problems in the vicinity of complex geometrical configurations. The flux reconstruction (FR) approach, developed by Huynh [1], is a simple yet efficient high-order scheme that is particularly amenable to the requirements of modern hardware architectures—including GPUs. Using the FR approach it is possible to unify various popular high-order methods such as nodal discontinuous Galerkin (DG) and spectral difference schemes. In 2011 Vincent et al. identified an infinite range of FR schemes which are linearly stable [2];

The FR Approach

- Consider the 1D scalar conservation law
  \[ \frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0 \]
  on a domain \([a,b]\).
- Divide the domain up into elements (of any length) and apply a linear transformation to these elements into standard elements in \([-1,1]\); inside each standard element we store the approximate solution at a set of \(k+1\) points termed solution points.
- The FR approach then proceeds as follows.
  - (a) Construct a degree \(k\) Lagrange interpolating polynomial through the solution points.
  - (b) Evaluate the flux at each solution point and form a second-degree \(k\) interpolating polynomial.
  - (c) Evaluate the solution point polynomial at \(x = \pm 1\) and \(r \pm 1\) and solve the Riemann problem resulting at each element interface to yield a common flux.
  - (d) Define a left correction function \(g_1\) such that \(g_1(0) = 1\) and \(g_1(1) = 0\).
  - (e) Scale this function by the jump between the common and interpolated fluxes.
  - (f) Add this scaled correction function to the flux polynomial.
  - (g) Repeat steps (d)-(f) for the right hand side with \(g_2(0) = g_1(0)\).
  - (h) Take the derivative of the corrected flux polynomial at each solution point. Once suitably transformed this flux derivative can be fed into a time marching algorithm, e.g. RK4.

Implementation

- Operations in FR reduce down to evaluating
  - polynomials and their derivatives;
  - point-wise functions, e.g. the flux;
  - common fluxes at interfaces.
- Majority of operations are therefore element local.
- Possible to cast polynomials evaluations as matrix multiplications which are extremely efficient on today’s many-core computing platforms.

PyFR

- Capable of solving the compressible Euler/Navier-Stokes equations on unstructured grids of tensor-product elements (quads and hexes).
- Written in Python and utilizes a backend architecture to target multiple platforms
  - backends currently exist for C/C++, OpenMP and CUDA (PyCUDA);
- Matrix multiplications are optimized for C++.
- Point-wise functions and other bespoke operations are templated using Makos;
  - code is generated, compiled, linked and loaded at runtime;
  - allows PyFR to readily exploit platform-specific instruction sets, e.g. XOP/XAVI/...;
  - Multiple nodes supported through C++
  - accomplished via the mako.py library.
- Current statistics:
  - 4000 lines of Python;
  - 800 lines of Makos/CUDA;
  - 800 lines of Makos/C.
- Released under a three-clause ‘new style’ BSD license.

Sample Code Fragments

- Use of SymPy’s symbolic manipulation capabilities for constructing the various polynomial operators.
- Template kernel generation.

Future Plans

- Support for simplex elements including triangles, prisms and tetrahedra.
- Implement adaptive time-stepping algorithms and the viability of semi-implicit schemes.
- An OpenCL backend to allow PyFR to target AMD’s 16000 and Intel Xeon Phi accelerators.
- Investigate means of incorporating shock capturing into FR.

Acknowledgements

Our work is supported by the Engineering and Physical Sciences Research Council (EPSRC) and NVIDIA. Conference participation aided by the Royal Commission for the Exhibition of 1851, the Imperial College General Trust and the Old Courtiers’ Trust.

References