Eilmer4 Steady-State Accelerator
Progress Update

K. A. Damm

Centre for Hypersonics, School of Mechanical and Mining Engineering,
University of Queensland, Brisbane, Queensland 4072, Australia

6th May 2021

Overview

- Motivation:
  - **"why so slow?"** (*Eilmer User*, circa 1990-2021)
- Numerical Methods
  - Governing Equations
  - Spatial Discretisation
  - Temporal Discretisation:
    - Explicit vs Implicit
    - Newton-Krylov method
- Development History & Demonstration Cases

Motivation

- Why a **steady-state solver**?
  - Supplement design and analysis of steady flow experiments
  - Vehicle design work requires analysis of many candidate designs
- Benefits of in-house code development:
  - **Open-source code:** no hidden fudge factors
  - **Full control:** features on demand (including bug fixes)
  - **Direct communication:** trouble shooting problematic simulations

Compressible Flow Governing Equations

Conservation of mass:

$$\frac{\partial}{\partial t}\rho + \nabla \cdot \rho \mathbf{u} = 0$$

Conservation of species mass:

$$\frac{\partial}{\partial t}\rho_i + \nabla \cdot \rho_i \mathbf{u} = -\left(\nabla \cdot \mathbf{J}_i\right) + \dot{\omega}_i$$

Conservation of momentum:

$$\frac{\partial}{\partial t}\rho \mathbf{u} + \nabla \cdot \rho \mathbf{u}\mathbf{u} = -\nabla p - \nabla \cdot \left\{-\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\dagger}) + \tfrac{2}{3}\mu(\nabla \cdot \mathbf{u})\boldsymbol{\delta}\right\}$$

Conservation of total energy:

$$\frac{\partial}{\partial t}\rho E + \nabla \cdot (e + \frac{p}{\rho})\mathbf{u} = \nabla \cdot [k\nabla T + \sum_{s=1}^{N_v} k_{v,s}\nabla T_{v,s}] + \nabla \cdot \left[\sum_{i=1}^{N_s} h_i \mathbf{J}_i\right]$$
$$- \left(\nabla \cdot \left[\left\{-\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\dagger}) + \tfrac{2}{3}\mu(\nabla \cdot \mathbf{u})\boldsymbol{\delta}\right\} \cdot \mathbf{u}\right]\right) - Q_{\text{rad}}$$

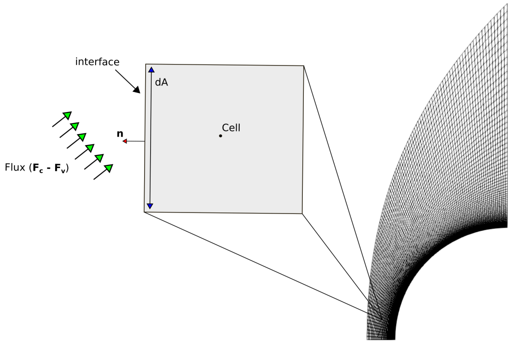
Conservation of vibrational energy:

$$\frac{\partial}{\partial t}\rho_i e_{v,i} + \nabla \cdot \rho_i e_{v,i}\mathbf{u} = \nabla \cdot [k_{v,i}\nabla T_{v,i}] - \nabla \cdot e_{v,i}\mathbf{J}_i + Q_{T-V_i} + Q_{V-V_i} + Q_{\text{Chem}-V_i} - Q_{\text{rad}_i}$$

***Coming soon**

Spatial Discretisation

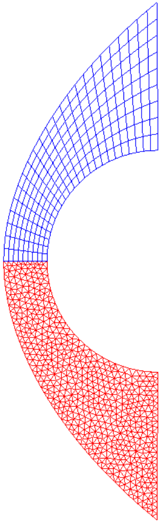
- Finite Volume Method



$$\frac{\partial \boldsymbol{U}}{\partial t} = \boldsymbol{R}(\boldsymbol{U}) = -\frac{1}{V}\sum_{faces}(\overline{\mathbf{F}_c} - \overline{\mathbf{F}_v}) \cdot \hat{\mathbf{n}}\ dA + \mathbf{S}$$

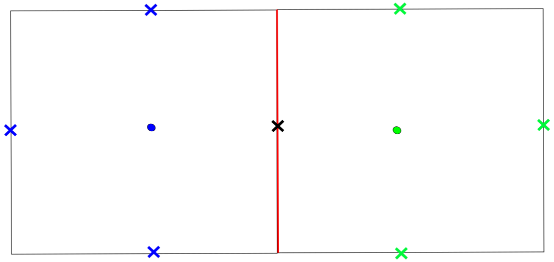
Spatial Discretisation

- Convective Fluxes
  - Flux calculators
    - EFM, AUSMDV, HLLC, LDFSS,
    - Hanel, HLLE, Roe, ASF
  - Structured Grids
    - Piecewise parabolic reconstruction — $\mathcal{O}(h^3)$
    - Modified Van Albada limiter
  - Unstructured Grids
    - Least-squares reconstruction — $\mathcal{O}(h^2)$
    - Venkatakrishnan limiter
    - Limiter freezing available



Spatial Discretisation

- Viscous Fluxes
  - Augmented-face face-tangent method
    - Least-squares method to reconstruct gradients
    - Special averaging using gradients, flowstates, and cell geometry
    - Available with structured and unstructured grids
    - Retains high spatial order for multi-block simulations



Temporal Discretisation

- **Explicit** Euler Time Integration

Recall semi-discrete form of residual

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{R}(\mathbf{U})$$

Using a forward difference discretization, the scheme is written as

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^{n}}{\Delta t} = \mathbf{R}(\mathbf{U})^{n}$$

where Δt is the discretised time increment.
Rearranging recovers the explicit Euler time marching iterate

$$\mathbf{U}^{n+1} = \mathbf{U}^{n} + \Delta t \mathbf{R}(\mathbf{U})^{n}$$

Temporal Discretisation

- **Explicit** Euler Time Integration continued...

Stability of the explicit Euler scheme is dependent on the CFL condition:

$$CFL = \left[\frac{\Lambda_i + C\Lambda_v}{\Delta x}\right]\Delta t \leq CFL_{\max},$$

where,

$$\Lambda_i = \left(|u(x)| + a(x)\right) \; ,$$
$$\Lambda_v = \frac{(\mu + \mu_T)\gamma}{\rho Pr \Delta x} \; .$$

In practice we know $CFL_{\max}$ (e.g. $CFL_{\max} = 1$ for Euler schemes)

$$\Delta t = \frac{\Delta x}{\Lambda_i + C\Lambda_v} CFL_{\max}.$$

Temporal Discretisation

- **Implicit** Time Integration

Again recall semi-discrete form of residual

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{R}(\mathbf{U})$$

written in fully discrete form

$$\frac{\Delta \mathbf{U}^n}{\Delta t} = \mathbf{R}(\mathbf{U})^{n+1}$$

where,

$$\Delta \mathbf{U}^n = \mathbf{U}^{n+1} - \mathbf{U}^n$$

and Δt is the discretised time increment.

Temporal Discretisation

- **Implicit** Time Integration continued...

Since we do not know $\mathbf{R}(\mathbf{U})^{n+1}$, we linearise in time

$$\frac{\Delta \mathbf{U}^n}{\Delta t} = \mathbf{R}^n + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}} \Delta \mathbf{U}^n$$

this is then rearranged to recover the implicit-Euler time marching iterate

$$[\mathbf{A}]^n \Delta \mathbf{U}^n = \left\{ \frac{1}{\Delta t} \mathbf{I} - \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}} \right\} \Delta \mathbf{U}^n = \mathbf{R}^n$$

Temporal Discretisation

- Rowan's implicit algorithm wish list from 2017:
  - able to treat $\mathbf{R}(\mathbf{U})$ as a black box
  - good for high speed flows (ie. grids with high aspect ratio cells)
  - works for both structured and unstructured grids
  - avoid the need to derive and code implicit boundary conditions
  - easily parallelized and scales well in parallel
  - efficient on memory, particularly in 3D

Temporal Discretisation

- Globalised **Newton-Krylov** method

$$[\mathbf{A}]^n \Delta \mathbf{U}^n = \left\{ \frac{1}{\Delta t}\mathbf{I} - \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}} \right\} \Delta \mathbf{U}^n = \mathbf{R}^n.$$

  - note: as $1/\Delta t$ approaches 0, **Newton's** method is recovered
  - **Krylov** term comes from the use of a Krylov subspace linear solver:
    - We use **GMRES** which only requires matrix-vector products
    - No need to form [**A**] matrix!*

$$\mathbf{Jv} \approx \frac{[\mathbf{R}(\boldsymbol{U} + h\mathbf{v}) - \mathbf{R}(\mathbf{U})]}{h}$$

Temporal Discretisation

- Looking back at Rowan's implicit algorithm wish list from 2017:
  - **able to treat $\mathbf{R}(\mathbf{U})$ as a black box**
  - good for high speed flows (ie. grids with high aspect ratio cells)
  - works for both structured and unstructured grids
  - avoid the need to derive and code implicit boundary conditions
  - easily parallelized and scales well in parallel
  - efficient on memory, particularly in 3D

The Frechet derivative allows the solver to feel the "full" Jacobian. This is in contrast to methods that might approximate the Jacobian. This method makes no assumptions about the flow gradient directions and full information from the Jacobian is transmitted to the solution.

Temporal Discretisation

- Looking back at Rowan's implicit algorithm wish list from 2017:
  - **able to treat R(U) as a black box**
  - good for high speed flows (ie. grids with high aspect ratio cells)
  - **works for both structured and unstructured grids**
  - avoid the need to derive and code implicit boundary conditions
  - easily parallelized and scales well in parallel
  - efficient on memory, particularly in 3D

Using a Krylov method, which only requires the result of a matrix-vector product, and the Frechet derivative means that an explicit Jacobian is never formed. This means I don't need to know about the details of how R(U) is constructed.

Temporal Discretisation

- Looking back at Rowan's implicit algorithm wish list from 2017:
  - able to treat $\mathbf{R}(\mathbf{U})$ as a black box
  - good for high speed flows (ie. grids with high aspect ratio cells)
  - works for both structured and unstructured grids
  - **avoid the need to derive and code implicit boundary conditions**
  - easily parallelized and scales well in parallel
  - efficient on memory, particularly in 3D

The numerical differentiation involved with the Frechet derivative takes the hassle out of forming implicit boundary conditions.

Temporal Discretisation

- Looking back at Rowan's implicit algorithm wish list from 2017:
  - able to treat $\mathbf{R}(\mathbf{U})$ as a black box
  - good for high speed flows (ie. grids with high aspect ratio cells)
  - works for both structured and unstructured grids
  - avoid the need to derive and code implicit boundary conditions
  - **easily parallelized and scales well in parallel**
  - **efficient on memory, particularly in 3D**

The Newton-Krylov methods have been designed to scale well for use on large supercomputers. The present implementation was extended to shared-memory (and now MPI) parallel mode with very little extra code.

2016/2017

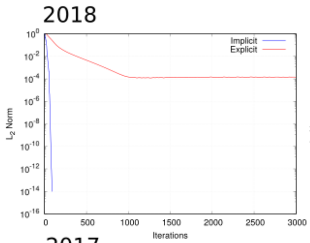
- Rowan begins steady-state solver development
- Features included:
  - Shared memory
  - Structured grid numerics
  - Navier-Stokes flows (ie. laminar flows)
  - Unpreconditioned GMRES
- Rowan presents steady-state solver at CfH (06-04-2017)
  - Classic Cone20 test case
  - Laminar flow over a flat plate

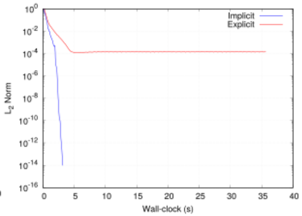
2018

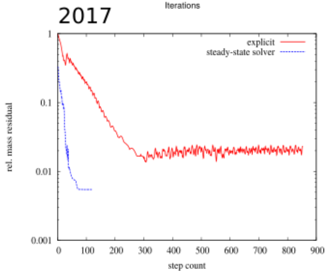
- Kyle takes over steady-state solver development
- Initial features implemented:
  - Operate on unstructured grid numerics
  - Complexified Eilmer
  - Preconditioned GMRES
    - Jacobian formed via complex-step differentiation
    - Complex-step Fréchet derivative
- Revisit: Cone20 & Laminar flow over a flat plate

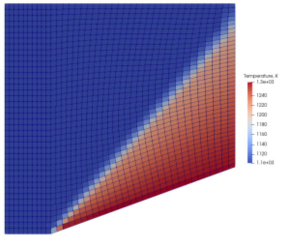
2018

- Classic Cone20 (Mach 1.5 air flow over a cone)



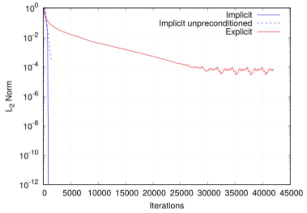


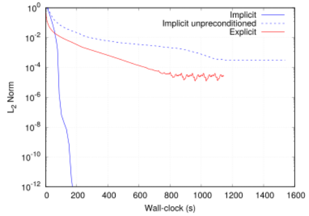


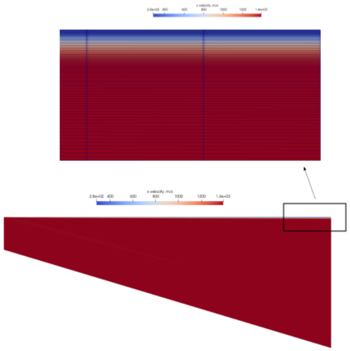


2018

- Mach 4.0 laminar air flow over a flat plate





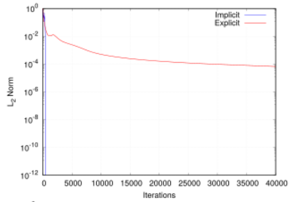


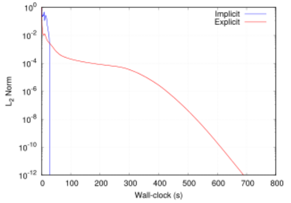
2019

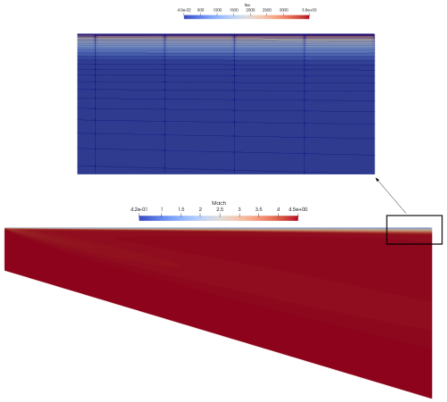
- Kyle extends solver to operate on the RANS equations
  - k-ω turbulence model
- Nick adds turbulence model class to Eilmer4 (huzzah!)
  - Implements Spalart-Allmaras model (vapourware no longer!)
  - Works out of the box for steady-state solver!

2019

- Mach 4.5 turbulent (k-ω) air flow over a flat plate ($y+ <= 5$)

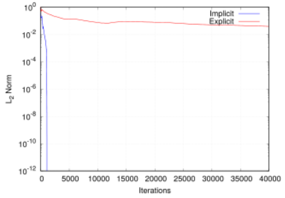


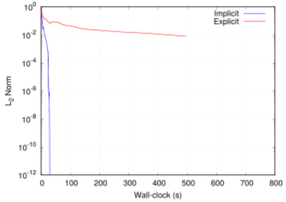


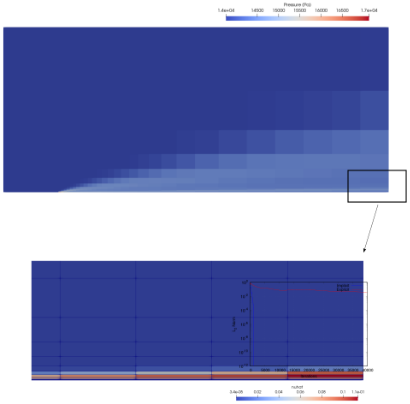


2019

- Mach 5.0 turbulent (SA) air flow over a flat plate ($y+ <= 1$)





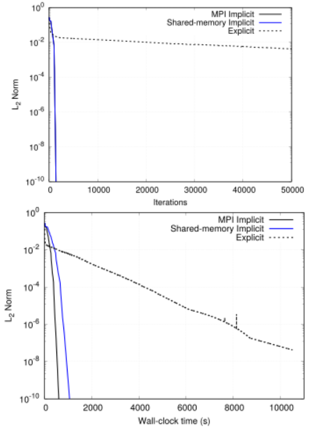


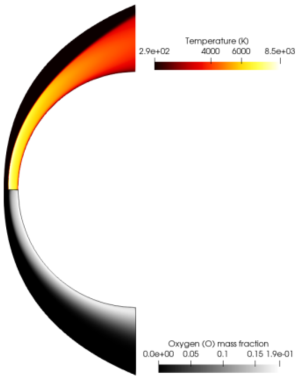
2020

- Kyle extends steady-state solver capability for reacting flows:
  - Thermally perfect gases
  - Multi-species
  - Finite rate chemistry
- Rowan adds MPI capability to Newton-Krylov solver
- Kyle also extends conjugate heat transfer solver capability:
  - 3D domains
  - Spatial gradients using augmented-face face-tangent method
  - MPI capability
  - More on the applicability of this later...

2020

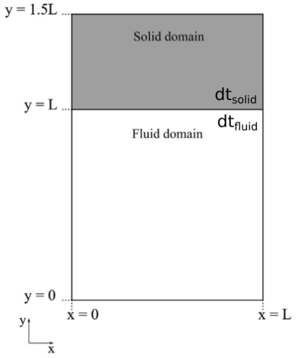
- Simulation of Lobb experiment
  - Mach 14 reacting air (5 species) flow over a sphere
  - Axisymmetric simulation





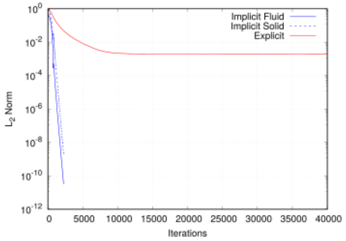
2021

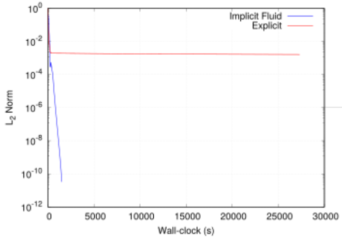
- Extensions to steady-state solver:
  - Added preconditioning capability for structured grid numerics
  - Extended to operate on solid domains
    - Enabled implicit Conjugate Heat Transfer calculations (CHT)
    - Loosely coupled implementation

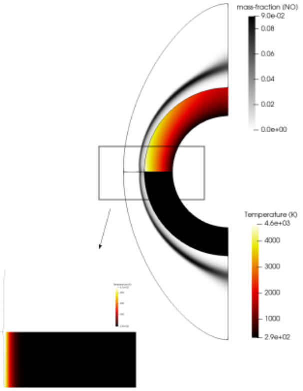


2021

- Reacting, turbulent flow over a hollow sphere with CHT:

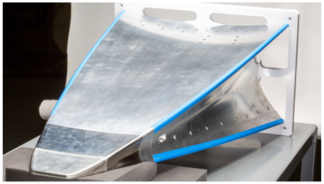
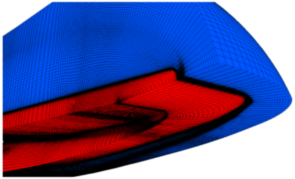
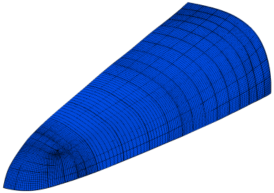
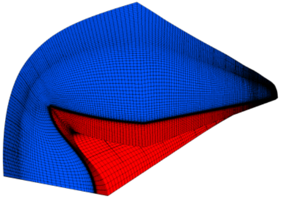






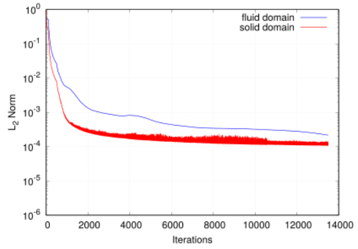
Demonstration Case I

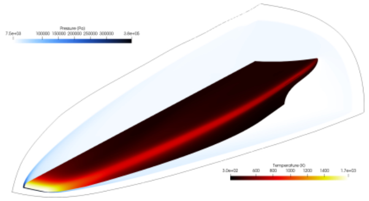
- BoLT-II
  - CHT simulation (fluid & solid domains)
  - 1.2 million cell (GridPro) structured grid (c/o Damian Curran)
  - 800k cells in fluid domain & 400k cells in solid domain

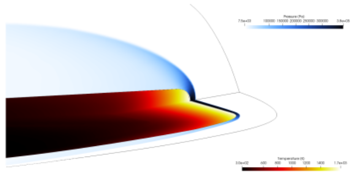





Demonstration Case I

- BoLT-II
  - Mach 6 tunnel condition
  - Structured grid numerics (nominally 3rd order reconstruction)
  - 24 hours on 96 cores (2 nodes of Gadi)

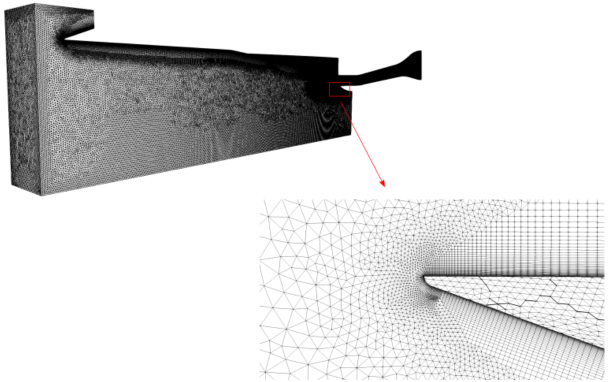






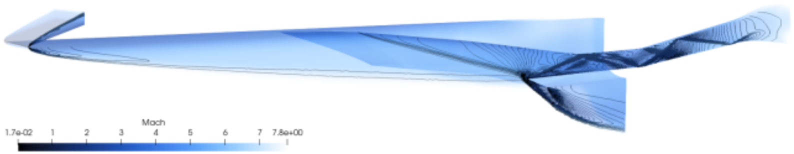
Demonstration Case II

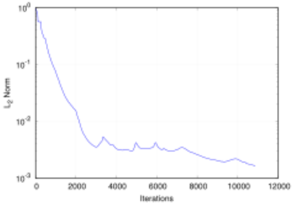
- HIFiRE-7
  - Eilmer4 challenge problem
  - 45 million cell (Pointwise) unstructured grid (c/o NASA)
  - Grid partitioned into 768 blocks using Eilmer4 METIS wrapper



Demonstration Case II

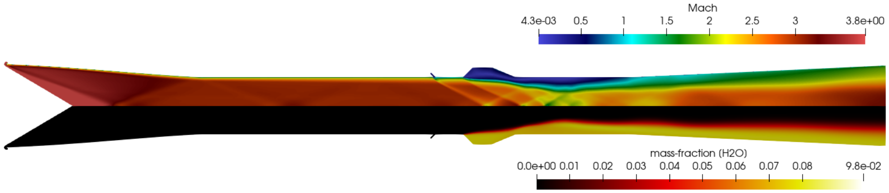
- HIFiRE-7
  - Mach 7.8448 tunnel condition
  - Current results: Euler simulation with first order reconstruction
  - 16 hours on 768 cores (16 nodes of Gadi)





Future Work

- Extend to include **two-temperature modelling**
- Improve robustness & efficiency
- Explore application to more complex flows...


Mach
4.3e-03
3.8e+00
mass-fraction (H2O)
0.0e+00
9.8e-02