Analysis of Non-linear Implicit Solutions to the Euler Equations for Hypersonic and Shocked Flows

Prof. Matthew Smith
Department of Mechanical Engineering, NCKU

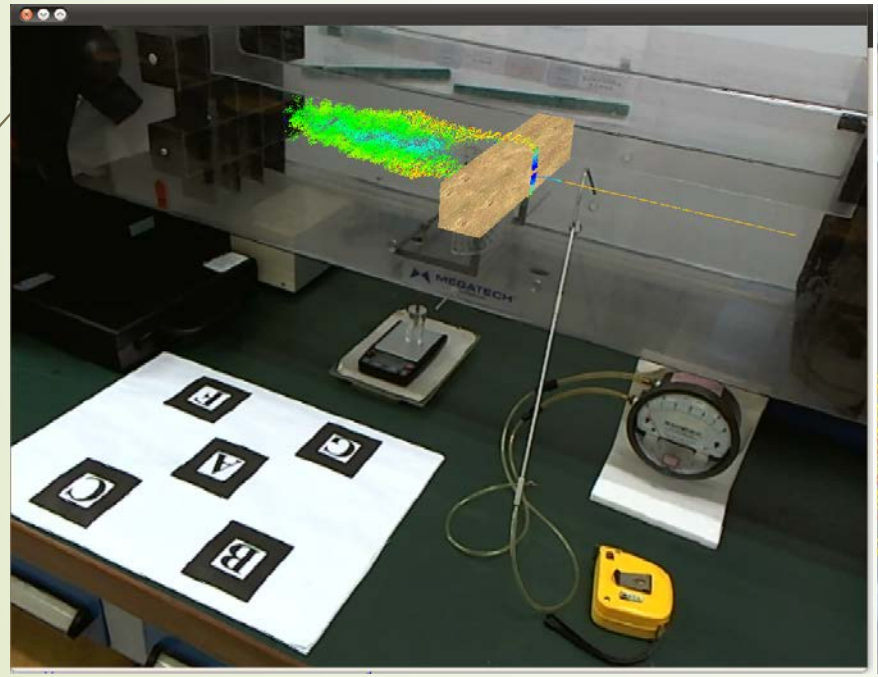
The first annual meeting of applied mathematics: Frontier aspects of applied mathematics, NTU, 6th Dec. 2015.

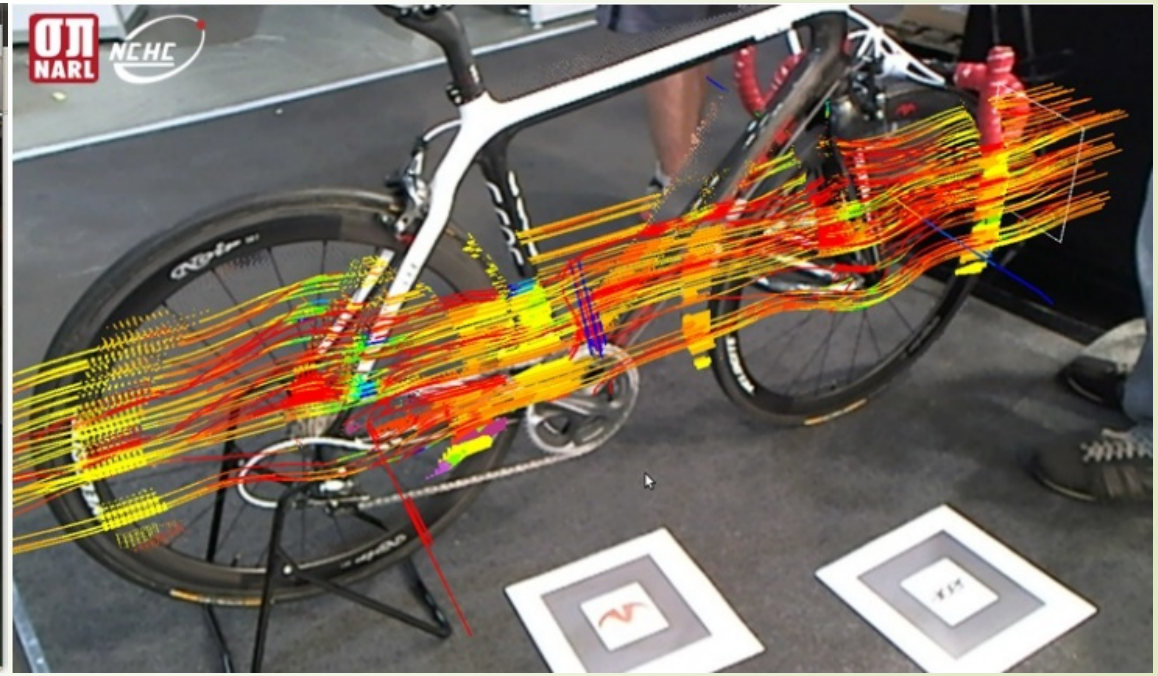
Acknowledgements

- I’d like to thank the organizers for the invitation to present my work.

Background

- Computational Fluid Dynamics (CFD) forms one of the core disciplines of modern engineering:





Background

- The Finite Volume Method (FVM) is a conservative approach which aims at solving equations based on their integral form:

$$\frac{\partial}{\partial t}\oint U\, dV + \oint F \cdot n\, dS = 0$$

Where U is a vector of conserved quantities, and F is the associated vector of Fluxes of U across surface dS.

- The core of the approach lies on the computation of fluxes at cell interfaces → these are then used to update the average conserved quantity in each control volume (or cell):

$$\frac{\partial \overline{U}}{\partial t} = -\frac{1}{V}\sum_{i=1}^{NF} F_i \cdot n_i\, A_i$$

Background

- For the purpose of today's presentation, we will restrict our investigations to the Euler Equations – for the compressible flow of an ideal, inviscid gas.

- These can be written in PDE form as:

$$\frac{d}{dt}[U] + \frac{d}{dx}[F] = \frac{d}{dt}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix} + \frac{d}{dx}\begin{bmatrix}\rho u\\ \rho u^2 + P\\ \rho uv\\ u(E+P)\end{bmatrix} + \frac{d}{dy}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2 + P\\ v(E+P)\end{bmatrix} = 0$$

- In order to apply the Finite Volume Method, we need to find a way to compute the fluxes F across each cell interface ➔ EFM.

Background

- Today's talk focuses on the application of a kinetic-theory based method to the application of a transient FVM solver.
- The method is Pullin's **Equilibrium Flux Method**, or EFM, which computes fluxes across cell interfaces by taking moments around the Maxwell-Boltzmann equilibrium particle distribution function.
- Since this time, several researchers have proposed advancements of this technique – however, we will see that EFM serves as a useful start point for our investigation.

Background

- The Maxwell-Boltzmann probability distribution function governing molecular velocities of a gas in thermal equilibrium can be written in 1D form as:

$$f(v) = \frac{1}{\sqrt{2\pi s}} \exp\left[\frac{-(v-\bar{v})^2}{2s^2}\right]$$

- The flux of any conserved quantity can be computed by taking moments around this distribution function:

$$F = \int_{-\infty}^{\infty} f(v_n) v_n Q(U) dv_n \qquad Q = \{\rho, \rho v_n, \rho v_p, \rho(0.5V^2 + E_{in})\}^T$$

- The EFM approach starts by splitting F into a forward and backward part:

$$F = F^- + F^+ = \int_{-\infty}^{0} f(v_n) v_n Q(U) dv_n + \int_{0}^{\infty} f(v_n) v_n Q(U) dv_n$$

Background

- We can see that F^+ describes fluxes due to particles with a positive velocity, while F^- describes fluxes due to particles moving with a negative velocity.
- Pullin proposed that, if particles were to move in free flight over the course of a time step, that positive moving particles must originate from the left hand side of an interface (and vice versa for negative moving particles)
- This gives us the split fluxes as:

$$F^+ = \int_0^\infty f(v_n) v_n Q(U_L) dv_n \qquad F^- = \int_\infty^0 f(v_n) v_n Q(U_R) dv_n$$

Background

- Note: This means that the EFM approach does not allow the evolution of the particle velocity distribution over the time which particles travel.
- This results in excessive numerical diffusion – particles are allowed to “fly” unhindered over a timestep.
- Several authors have proposed fixes to this problem by using the BGK / Boltzmann Equations to evolve the particle distribution functions over time (and space) to obtain a better result (not investigated here).

Discretization of the FVM

- Examine for a second this discretization:

$$\frac{\partial \overline{U}}{\partial t} = -\frac{1}{V}\sum_{i=1}^{NF} F_i \cdot n_i \, A_i$$

- We can solve for our value of $\overline{U}$ at time level k+1 by:
  - Using values of both $\overline{U}$ and F at time levels k → This leads to an **explicit approach**.
  - Using values of $\overline{U}$ at time level k and F at time levels k+1 → This leads to an **implicit approach**.

Discretization of the FVM

- Let's write the equations out for clarity – in 1D, we may write:
  - Explicit Form: $$\frac{\overline{U}_i^{k+1} - \overline{U}_i^{k}}{\Delta t} + \frac{F^k_{i+1/2} - F^k_{i-1/2}}{\Delta x} = 0$$
  - Implicit Form: $$\frac{\overline{U}_i^{k+1} - \overline{U}_i^{k}}{\Delta t} + \frac{F^{k+1}_{i+1/2} - F^{k+1}_{i-1/2}}{\Delta x} = 0$$
- Fluxes at cell interfaces (i.e. i+1/2) are reconstructed:

$$F^+_{i+x} \approx F^+_i + x\left(\frac{dF^+}{dx}\right)_{i,eff} + H.O.T \qquad F^-_{i+x} \approx F^-_i + x\left(\frac{dF^-}{dx}\right)_{i,eff} + H.O.T$$

- Evaluations of (dF^+/dx) and (dF^-/dx) use slope limiting functions to maintain positivity.

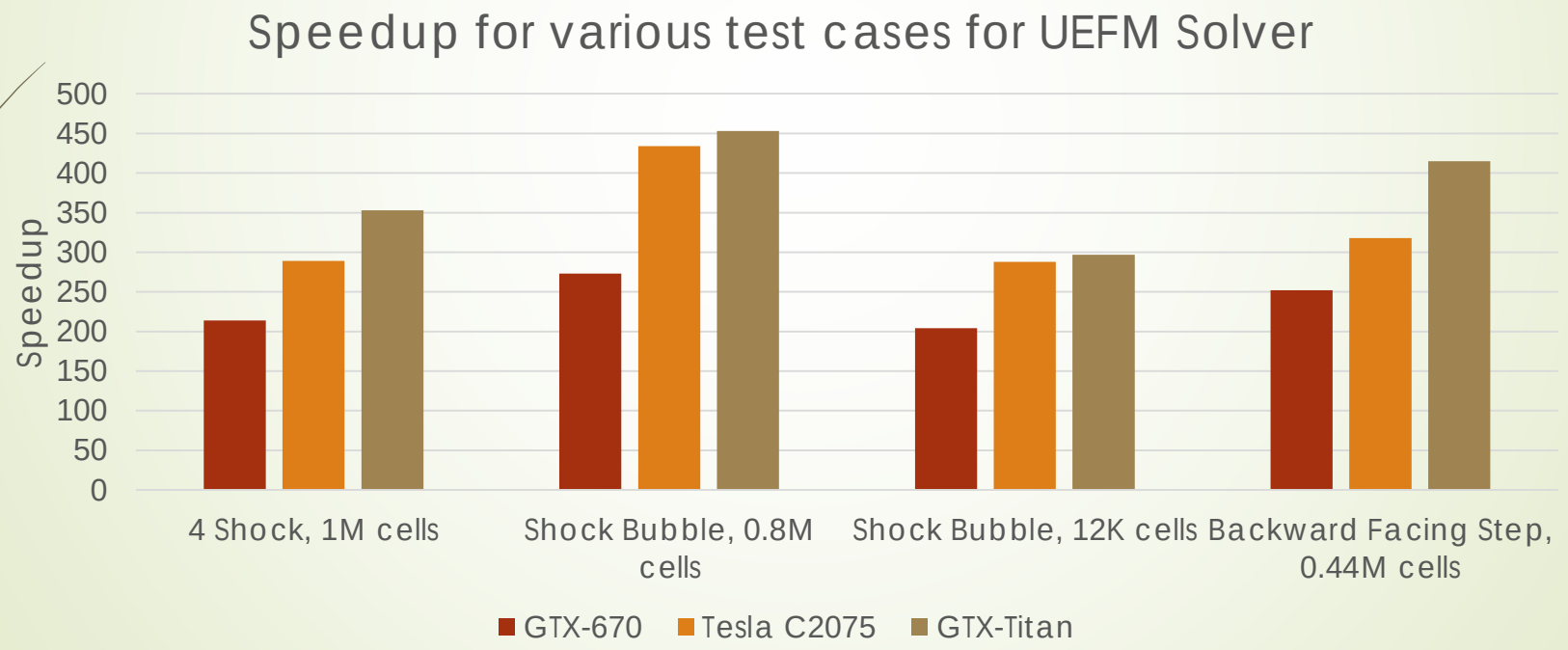
Application of Explicit FVM

- In the past, we have employed explicit techniques for application to parallel computing due to the ease of parallelization.
- The result – a family of efficient solvers which run on both the Intel Xeon Phi and the GPU (AMD and Nvidia devices)

This simulation required less than 3 minutes – from start to finish – using a GTX-Titan.

Application of Explicit FVM

- When Cartesian grids where employed, the speedup – ratio of computational times for a single CPU core to that of the GPU – was computed to be very large.



Drawbacks of Explicit Approach

- There are disadvantages to using Explicit time-stepping approaches:
  - The size of the time step of limited based on the stability requirement of the smallest cell in the flow field – which is often several orders of magnitude smaller than the average cell size when using adaptive grids.
  - We cannot solve for a steady flow*.
- One possible way we can avoid these problems is to use an implicit time-stepping procedure to solve the governing equations.

Implicit formulation of EFM

- A casual inspection of our governing equations reveals they are fundamentally non-linear:

$$\frac{d}{dt}[U]+\frac{d}{dx}[F]=\frac{d}{dt}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix}+\frac{d}{dx}\begin{bmatrix}\rho u\\ \rho u^2+P\\ \rho uv\\ u(E+P)\end{bmatrix}+\frac{d}{dy}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+P\\ v(E+P)\end{bmatrix}=0$$

- In addition, our split fluxes – which are used to compute net fluxes in our discretization – contain erf(x) and exp(-x^2) terms, which are also non linear.
- This gives us – for example, in a 1D problem:

$$\frac{\overline{U}_i^{k+1}-\overline{U}_i^{k}}{\Delta t}+\frac{\left[\left(F_i^{+,k+1}+F_{i+1}^{-,k+1}\right)-\left(F_i^{-,k+1}+F_{i-1}^{-,k+1}\right)\right]}{\Delta x}=0$$

$$F_i^{+}\approx A.erf(M_i)+B.exp\left(-M_i^2\right) \qquad F_i^{-}\approx C.erf(M_i)+D.exp\left(-M_i^2\right)$$

Implicit formulation of EFM

- It should be easy to see that a linear decomposition in this case is impossible.

- Rewrite the governing equations into residual form:

$$R = \frac{\partial \overline{U}}{\partial t} + \frac{1}{V}\sum_{i=1}^{NF} F_i \cdot n_i \, A_i$$

- We need an approximation for the time derivative. One way might be:

$$R = \left(\frac{\overline{U^{k+1}} - \overline{U^k}}{\Delta t}\right) + \frac{1}{V}\sum_{i=1}^{NF} F_i\left(\overline{U^{k+1}}\right) \cdot n_i \, A_i$$

where k indicates the time-level of the solution and A, V do not change in time.

Implicit formulation of EFM

- We need to solve our unknowns – which, interesting enough, aren't the new conserved quantity, but the primitive values.

$$x = \begin{bmatrix} \rho \\ u \\ T \end{bmatrix} \qquad R(x) = \left(\frac{\overline{U(x)^*} - \overline{U(x)}}{\Delta t} \right) + \frac{1}{V} \sum_{i=1}^{NF} F_i(x^*) \cdot n_i \, A_i$$

- For spatially higher order implementations, we have:

$$R(x) = \left(\frac{\overline{U(x)^*} - \overline{U(x)}}{\Delta t} \right) + \frac{1}{V} \sum_{i=1}^{NF} \left[F_i(x^*) + r_i \frac{dF_i(x^*)}{dr} \right] \cdot n_i \, A_i$$

where F + r.dF/dr is the reconstructed value at the center of the cell face.

Implicit formulation of EFM

- For large CFL numbers, our first order treatment in time is inappropriate. We employ the **second order Adams-Moulton formula** to allow 2nd order in time and space:

$$R(x) = \left(\frac{\overline{U(x)^*} - \overline{U(x)}}{\Delta t}\right) + \frac{1}{2}\left[\frac{1}{V}\sum_{i=1}^{NF} F_i(x)\cdot n_i\, A_i\right] + \frac{1}{2}\left[\frac{1}{V}\sum_{i=1}^{NF} F_i(x^*)\cdot n_i\, A_i\right]$$

Saved from previous time step

Implicitly computed

- The 2nd order Adams-Moulton scheme is much more accurate in time than the (explicit) 2nd order Adams-Bashforth technique while incurring very little additional computational expense.

Implicit Formulation of EFM

- Hence, for a 1D problem with N cells, x – our solution vector - will contain 3N elements.

- Since our system is non-linear, we'll use a Newton-Raphson type solver to evolve our solution x:

$$x^* = x - J^{-1}R(x)$$

where J is the Jacobian of R(x) and x* is our new estimate of x.

- It turns out this approach doesn't always work – but we will return to this point later on.

Implicit Formulation of EFM

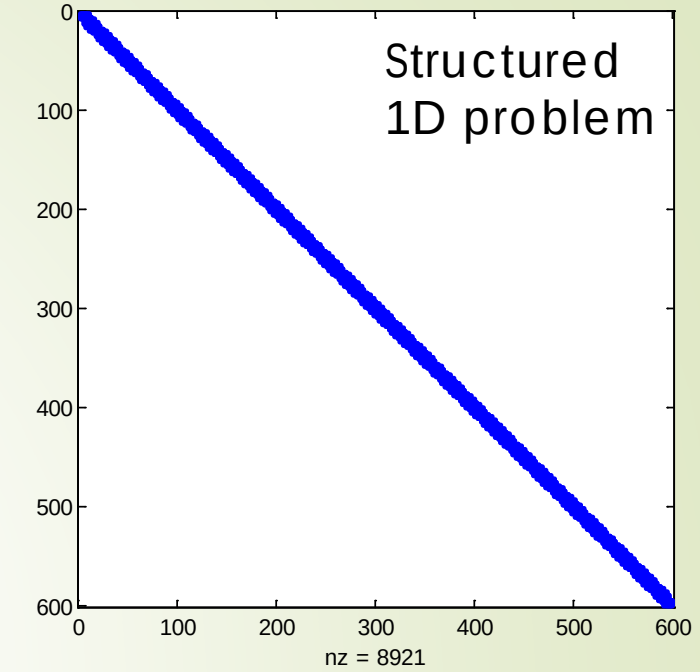
- The Jacobian J can be written as:

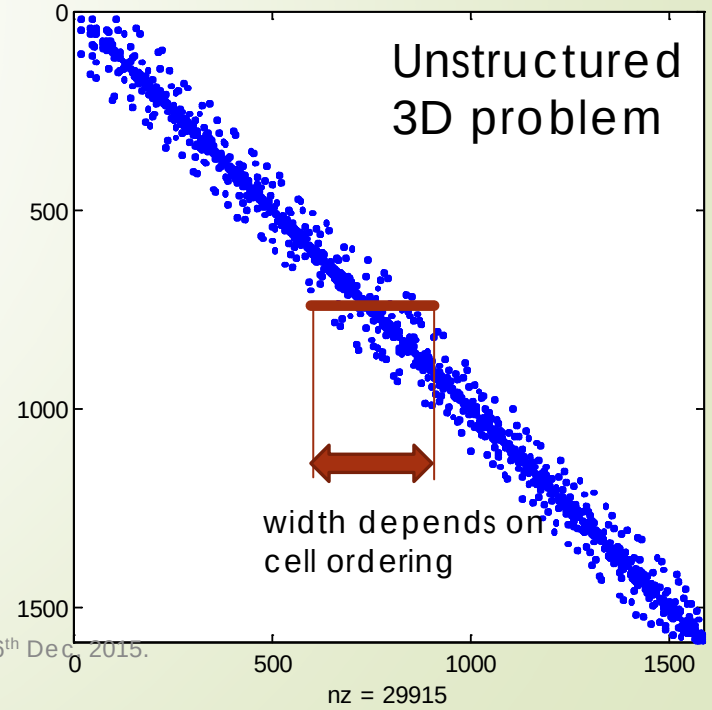
$$J = \begin{bmatrix} \frac{dR_1}{dx_1} & \cdots & \frac{dR_1}{dx_M} \\ \vdots & \ddots & \vdots \\ \frac{dR_M}{dx_1} & \cdots & \frac{dR_M}{dx_M} \end{bmatrix}$$

- There are two methods we might evaluate dR_i/dx_j:
  - Numerically – using a finite difference evaluation of R and x.
  - Analytically – using the analytical value of dR/dx derived from our governing equations.

Implicit Formulation of EFM

- On flows with unstructured grids and/or multiple unpredictable boundary types, the analytical form of J is too troublesome.
- We can see this by examining the shape of J (RHS).
- Hence, this work focuses on the application of a finite difference method to solve for dR/dx – hence, this is a **pseudo**-Newton Raphson method.





Implicit Formulation

- The computation of $J^{-1}R(x)$ does not require the actual inversion of the J matrix. We simply require the solution to the $J\Delta x = R$ systems of equations.

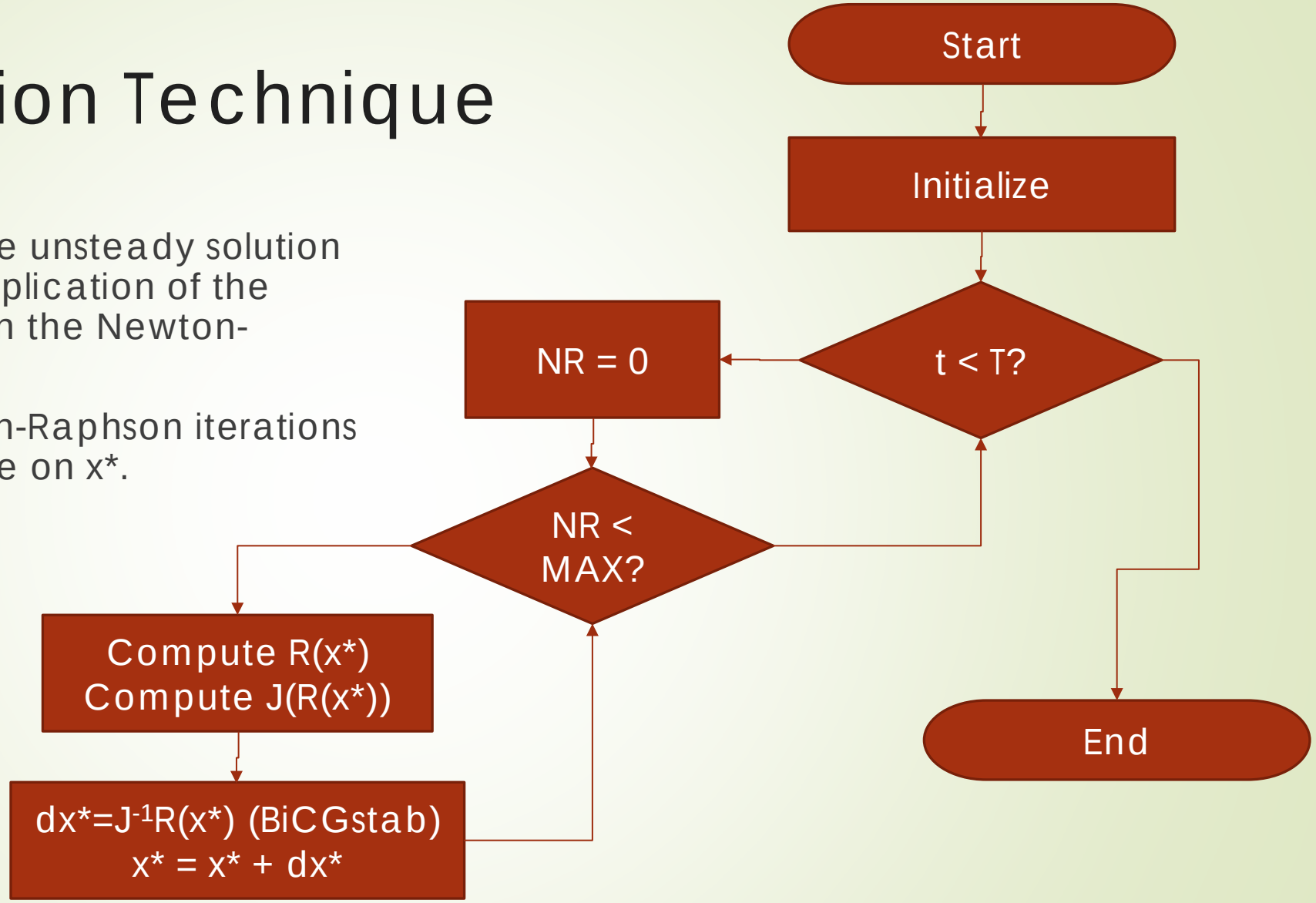
- An iterative approach may be employed: the J matrix is positive definite but is not symmetrical. Hence, we have a large number of options, but we have decided on the BiCG and BiCGstab methods.

- Both of these have varying suitability to GPU computation.

Solution Technique

- The computation of the unsteady solution requires the nested application of the BiCG / BiCGstab within the Newton-Raphson algorithm.
- In practice, 3-5 Newton-Raphson iterations is sufficient to converge on x*.

It helps that the **initial guess** for both the solution to x* (for each NR iteration) and $J^{-1}R$ is based on the **previous iterations solution**.



Solution Technique

- The BiCGStab method is chosen to find the solution to our solution change (dx*) within each Newton-Raphson iteration.
- This approach was selected primarily due to the associated advantages which arise when performing parallelization of the scheme using heterogeneous devices.
- We'll most likely want a preconditioner (as shown here) – we will visit this later as well.

Algorithm 1: Right preconditioned BiCGStab

$r_o = b - Ax_o$; x_o *taken from previous time step's solution*, $\bar{r}_0 = r_o$

$p_o = r_o$

for $j = 0,1$ until convergence **or** maximum iterations, **do**

$\tilde{p}_j = M^{-1} p_j$

$\propto_j = (r_j, \bar{r}_0) / (A\tilde{p}_j, \bar{r}_0)$

$s_j = r_j - \propto_j A\tilde{p}_j$

$\tilde{s}_j = M^{-1} s_j$

$w_j = (A\tilde{s}_j, s_j) / (A\tilde{s}_j, A\tilde{s}_j)$

$x_{j+1} = x_j + \propto_j \tilde{p}_j + w_j \tilde{s}_j$

$r_{j+1} = s_j - w_j A\tilde{s}_j$

$\beta_j = (r_{j+1}, \bar{r}_0) / (r_j, \bar{r}_0) \cdot \propto_j / w_j$

$p_{j+1} = r_{j+1} + \beta_j (p_j - w_j A\tilde{p}_j)$

end for

Parallelization

- The algorithm has been parallelized using OpenMP for conventional CPU and the Intel Phi Coprocessor, in addition to Graphics Processing Units using CUDA.
- Today's talk will focus on CUDA (Compute Unified Device Architecture) which allows general purpose computation on NVIDIA GPU devices.
- While our lab has many various GPU's, today's talk will focus on the GTX-Titan:

Number of CUDA cores: 2688 cores
Core frequency: 0.83 GHz
Memory bandwidth: 288 GB/sec
Amount of DDR5 ram: 6 GB

Parallelization

- The parallelization strategy for CUDA is performed through the application of kernels for each key part of the algorithm:
  - Compute_Flux<<< >>> → Computes the flux across each cell interface.
  - Compute_Residual<<< >>> → Computes the residual within each cell based on the fluxes.
  - Compute_J<<< >>> → Computes the Jacobian for each cell based on finite difference approximations for each variable within the cell and its attached neighbours.
  - BiCG and BiCGstab kernels → Numerous kernels for computing matrix-vector, vector-vector and dot product computations.

- A device function – callable only by GPU kernels – called GPUCalcFlux() is employed to compute the flux of conserved quanties by the Compute_Flux and Compute_Residual kernels.

Parallelization

- Each of these functions has an accompanying parallel efficiency:
  - Compute_Flux<<< >>> → Embarrassingly parallel, but requires poorly coalesced memory access.
  - Compute_Residual<<< >>> → Embarrassingly parallel, but requires poorly coalesced memory access.
  - Compute_J<<< >>> Embarrassingly parallel, but requires poorly coalesced memory access.
  - BiCG and BiCGstab kernels → A large number of kernels make up the computation. For BiCGstab, most are easily parallelized. However, BiCG relies upon the dot product, which requires parallel reduction (i.e. poor parallel efficiency).

Now for the real questions...

- There are several issues which will pertain to the success of this implementation:
  - Will the Pseudo-Newton Raphson implementation shown here behave as we expect? (Or, will it give us the solution we want?)
  - The Condition number of J will influence our solution time and general ease of computation. Which controllable simulation parameters influence J?
  - Does the size of our timestep / CFL influence the physical properties of the solution? In what way?

Newton-Raphson Solver

- Review our equation for our pseudo NR solver:

$$x^* = x - J^{-1}R(x)$$

- In our case, x* must remain positive (for density and temperature) – the EFM fluxes cannot be computed for negative temperatures, for example. And besides, they are non-physical (separate issue).

- The Newton-Raphson does not care about your need to keep a positive x. There are no active controls on the solver, and nothing stopping the solver taking a trip through negative x space on its way to the solution.

Newton-Raphson Solver

- Hence, we need to modify the original N-R algorithm slightly to prevent it from computing non-physical solutions on its way to the physical solution.

- We rewrite our equation as:

$$x^* = x - \alpha(J^{-1}R(x))$$

- Where alpha is computed as:

$$\alpha = \frac{|x(I_{MAX})|}{2|\Delta X_{MAX}|}$$

Algorithm 2: Implicit FVM using EFM and BiCGstab

1: $X\ set\ from\ initial\ conditions, X_{old} = X$
2: **for** *i = 0,1* until desired time has been reached, **do**
4: $X_{old} = X$
5: $Compute\ \nabla F(X_{old})$
6: **for** j = 0, 1,..until maximum no. of N-R iterations, **do**
7: $Compute\ R(X) = \frac{(U(X) - U(X_{old}))}{\Delta t} + 0.5\nabla F(X) + 0.5\nabla F(X_{old})$
8: $Evaluate\ J_j\ numerically\ over\ all\ M\ variables$
9: $Compute\ preconditioner\ M_j = diag(J_j)$
10: ➔ $Use\ Algorithm\ 1\ (BiCGstab)\ to\ iteratively\ solve\ for\ \Delta X_j$
11: $[\Delta X_{max}, I_{max}] = \max(|\Delta X_j|)$
12: $\propto_j = |X(I_{max})| \ / \ 2\ |\Delta X_{max}(I_{max})|$
13: if $j = 1, \propto_j = 0.01\ else\ \ \propto_j = \min(\propto_j, 1.0)$
14: $X_{j+1} = X_J + \propto_j \Delta X_j$
15: **if** $\Delta X_{max} \ / \ X(I_{max}) \ < 10^{-4}$ ***break***
16: **end for**
17: **end for**

Newton-Raphson Solver

- Explain a little:

$$x^* = x - \alpha(J^{-1}R(x)) = x - \alpha\Delta x$$

- To prevent NR from causing negative x values, the value of Δx must be **less than x**.

- Hence, we find the largest Δx in the solution (inside element I_{max}) and use a safety factor of 2, just to be sure.

Algorithm 2: Implicit FVM using EFM and BiCGstab

1: $X\ set\ from\ initial\ conditions, X_{old} = X$
2: **for** *i = 0,1* until desired time has been reached, **do**
4: $X_{old} = X$
5: $Compute\ \nabla F(X_{old})$
6: **for** j = 0, 1,..until maximum no. of N-R iterations, **do**
7: $Compute\ R(X) = \frac{(U(X) - U(X_{old}))}{\Delta t} + 0.5\nabla F(X) + 0.5\nabla F(X_{old})$
8: $Evaluate\ J_j\ numerically\ over\ all\ M\ variables$
9: $Compute\ preconditioner\ M_j = diag(J_j)$
10: ➔ $Use\ Algorithm\ 1\ (BiCGstab)\ to\ iteratively\ solve\ for\ \Delta X_j$
11: $[\Delta X_{max}, I_{max}] = \max(|\Delta X_j|)$
12: $\propto_j = |X(I_{max})| \,/\, 2\,|\Delta X_{max}(I_{max})|$
13: if $j = 1, \propto_j = 0.01\ else\ \propto_j = \min(\propto_j, 1.0)$
14: $X_{j+1} = X_J + \propto_j \Delta X_j$
15: **if** $\Delta X_{max} \,/\, X(I_{max}) < 10^{-4}$ ***break***
16: **end for**
17: **end for**

Newton-Raphson Solver

- We still need to be careful – as we get closer to the solution, the value of alpha explodes:

$$\alpha = \frac{|x(I_{MAX})|}{2|\Delta X_{MAX}|} \qquad x(I_{MAX}) \gg \Delta X_{MAX}$$

- Hence, we need to limit alpha to a maximum value – the safest and most mathematically correct limit is 1.
- Practice with this solver shows us this is the best choice (not > 1).

Algorithm 2: Implicit FVM using EFM and BiCGstab

1: $X\ set\ from\ initial\ conditions, X_{old} = X$
2: **for** *i = 0,1* until desired time has been reached, **do**
4: $X_{old} = X$
5: $Compute\ \nabla F(X_{old})$
6: **for** j = 0, 1,..until maximum no. of N-R iterations, **do**
7: $Compute\ R(X) = \frac{(U(X) - U(X_{old}))}{\Delta t} + 0.5\nabla F(X) + 0.5\nabla F(X_{old})$
8: $Evaluate\ J_j\ numerically\ over\ all\ M\ variables$
9: $Compute\ preconditioner\ M_j = diag(J_j)$
10: ➔ $Use\ Algorithm\ 1\ (BiCGstab)\ to\ iteratively\ solve\ for\ \Delta X_j$
11: $[\Delta X_{max}, I_{max}] = \max(|\Delta X_j|)$
12: $\propto_j = |X(I_{max})| \ / \ 2\ |\Delta X_{max}(I_{max})|$
13: if $j = 1, \propto_j = 0.01\ else\ \propto_j = \min(\propto_j, 1.0)$
14: $X_{j+1} = X_J + \propto_j \Delta X_j$
15: **if** $\Delta X_{max} \ / \ X(I_{max}) < 10^{-4}$ ***break***
16: **end for**
17: **end for**

Influence of CFL

- The first mission of this research was to determine (i) the influence of CFL number of the physical results, and (ii) the influence of CFL number on the Condition number of J, which in-turn has consequences for computational time.
- To demonstrate this influence, we will look at solutions to Sod's 1D shock problem ($\gamma = 1.4$).

$[\rho, u, T] = [10,0,1]$	$[\rho, u, T] = [1,0,1]$

Influence of CFL

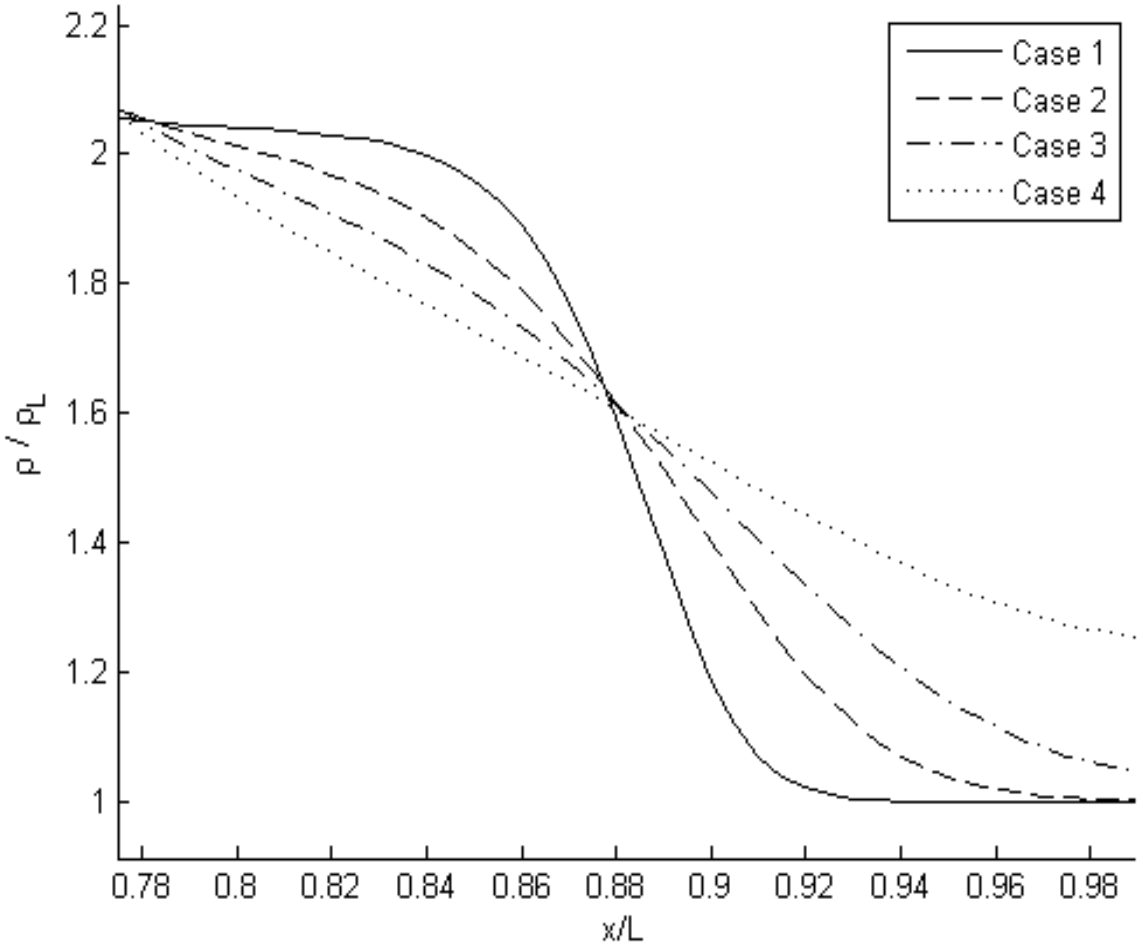
- We ran a large number of cases, over varying orders of spatial and temporal accuracy for various CFL numbers.
- There were – roughly speaking – 4 CFL numbers used: 0.5, 2.5, 5 and 10 (approximately).

Case`	Time Step Size	Spatial Order of Accuracy	Temporal Order of Accuracy	Maximum CFL
1	1e-3	1	1	0.47
2	5e-3	1	1	2.34
3	1e-2	1	1	4.62
4	2e-2	1	1	8.97
5	1e-3	2	1	0.47
6	5e-3	2	1	2.36
7	1e-2	2	1	4.68
8	2e-2	2	1	9.01
9	1e-3	2	2	0.47
10	5e-3	2	2	2.36
11	1e-2	2	2	4.92
12	2e-2	2	2	10.18

Influence of CFL - Results

Case`	Maximum CFL
1	0.47
2	2.34
3	4.62
4	8.97

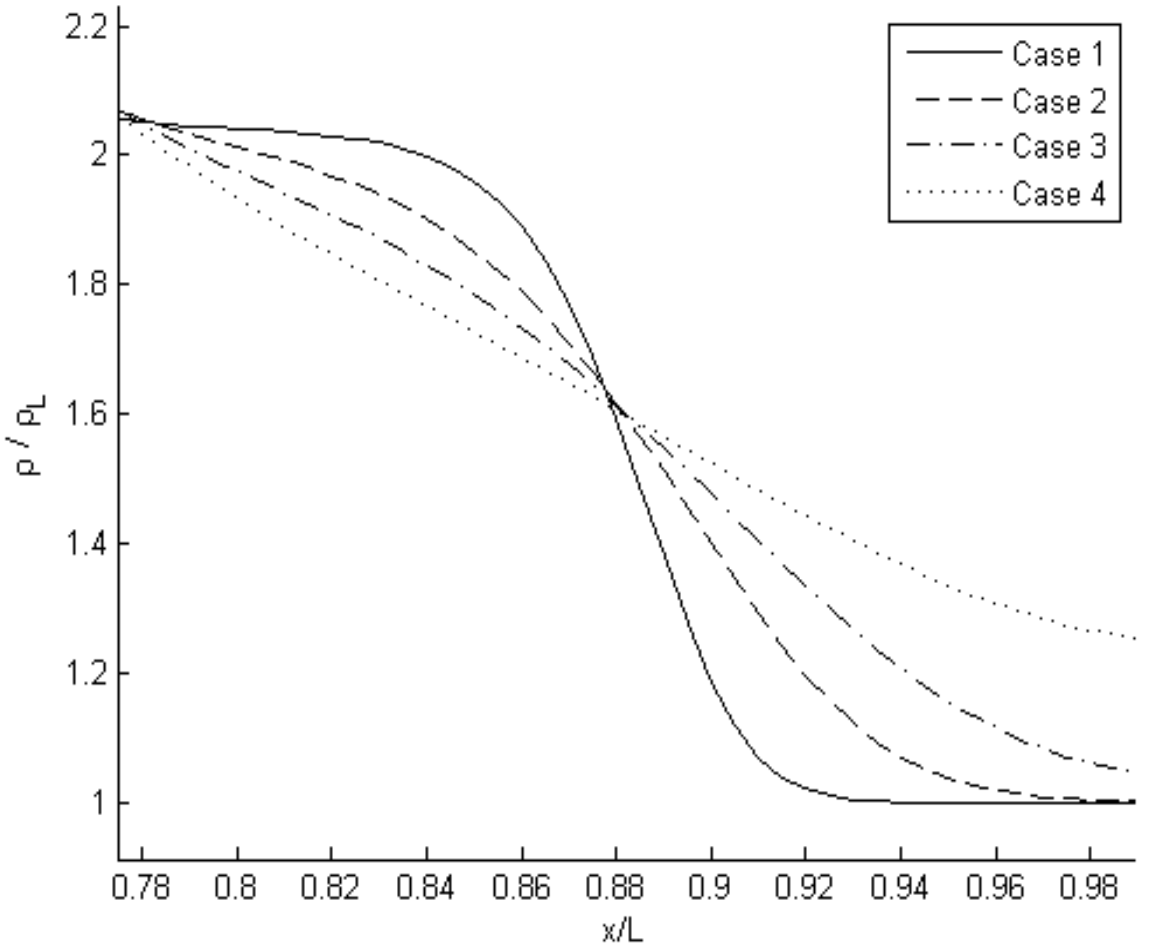
- Considering the first 4 cases – first order accurate in both time and space – and we obtain the following results.
- We can see that larger CFL numbers result in excessive diffusion.
- This is a result of the free flight assumption which forms the core of the EFM approach – or is it?



Influence of CFL - Results

Case`	Maximum CFL
1	0.47
2	2.34
3	4.62
4	8.97

- As the CFL increases, particles – these are imaginary particles, of course – are allowed to travel further and further.
- Hence, these results actually approach a result which might be obtained for increasingly rarefied flows.
- We might be mistaken into thinking that this result is due to the physical nature of the solver.... (**mistake**)



Influence of CFL - Results

- We also see that the CFL number has an impact on the computational time required – through the Condition number of the Jacobian matrix J.
- Increasing the CFL from ~0.5 to ~9 leads to an average Condition number increase of ~x8 times. (A seemingly good deal)
- Let's have a look at the convergence properties....

Case`	Maximum CFL
1	0.47
2	2.34
3	4.62
4	8.97

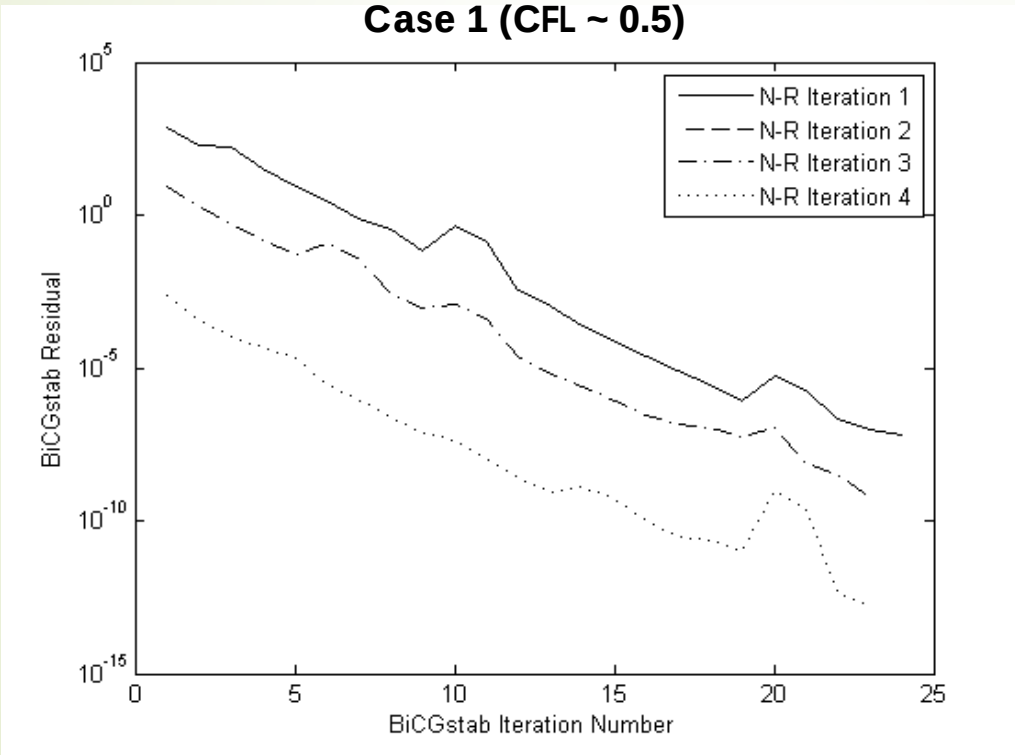
	Case 1	Case 2	Case 3	Case 4
Normalized Average Condition Number [i.e. cond($M^{-1}J$)] Accuracy - O(1) (time) / O(1) (space)	1	2.45	4.29	7.96

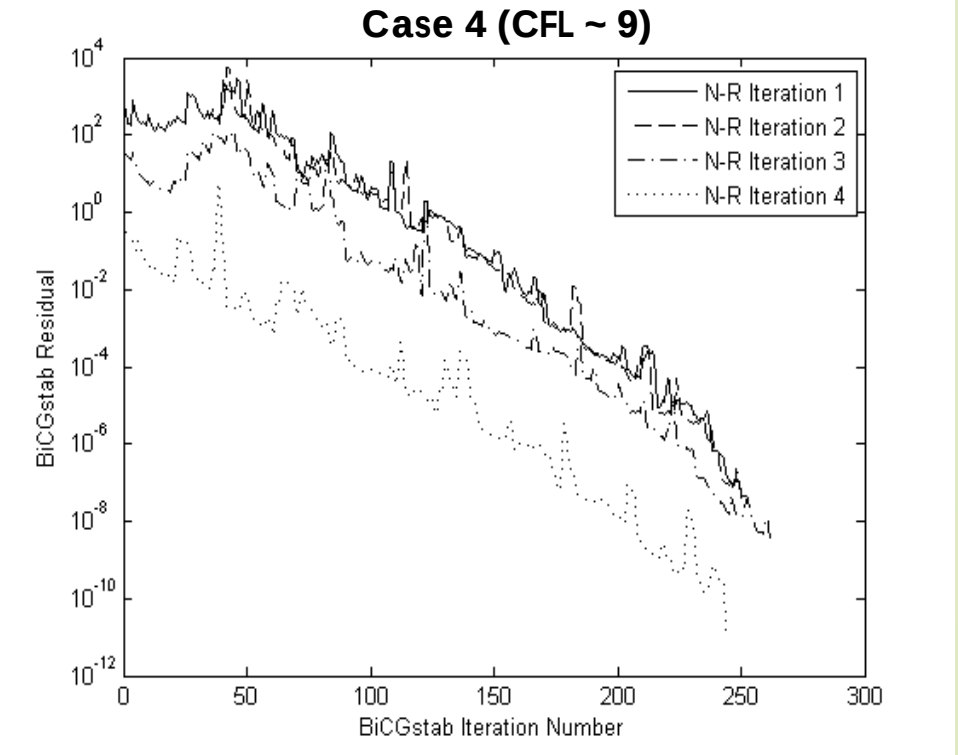
Influence of CFL - Results

- We see that there is approximately a 10x in the number of BiCGstab iterations required for the larger CFL.

Here I neglected to mention that I am using BiCGStab with a Jacobi preconditioner.

Sorry. ☺

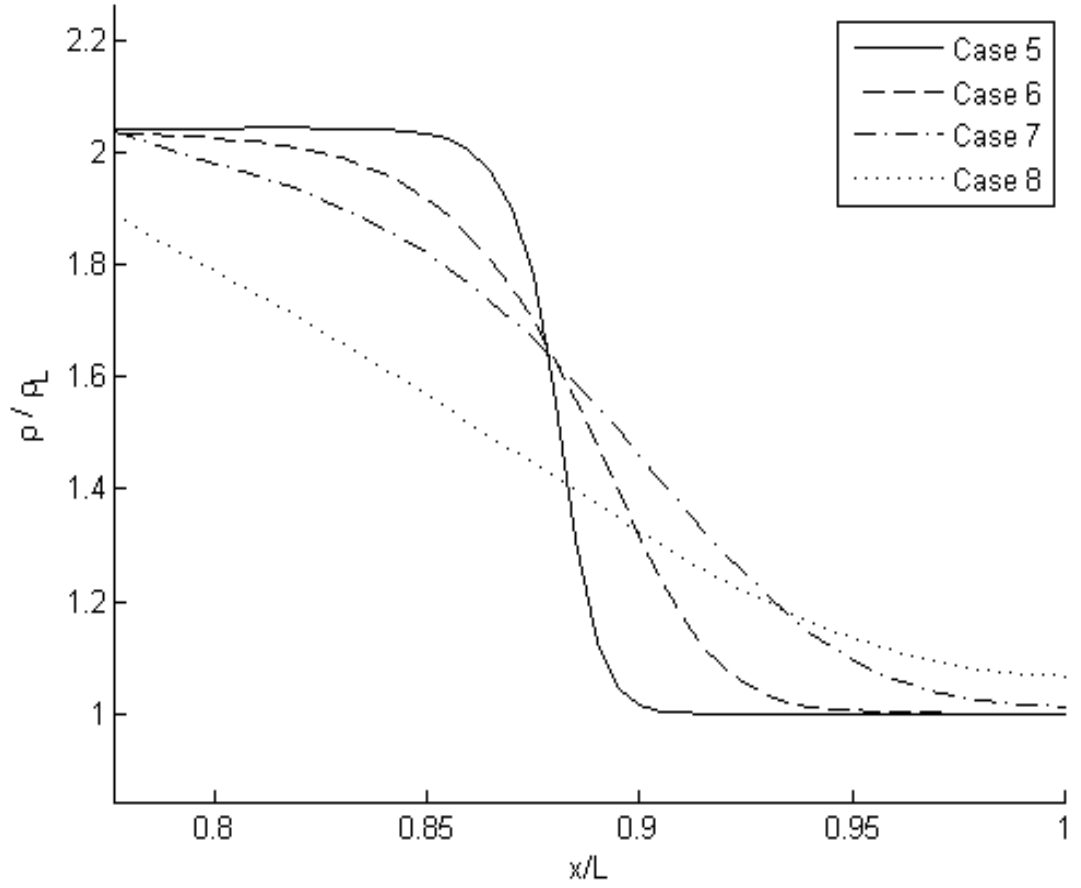




Influence of Spatial Accuracy

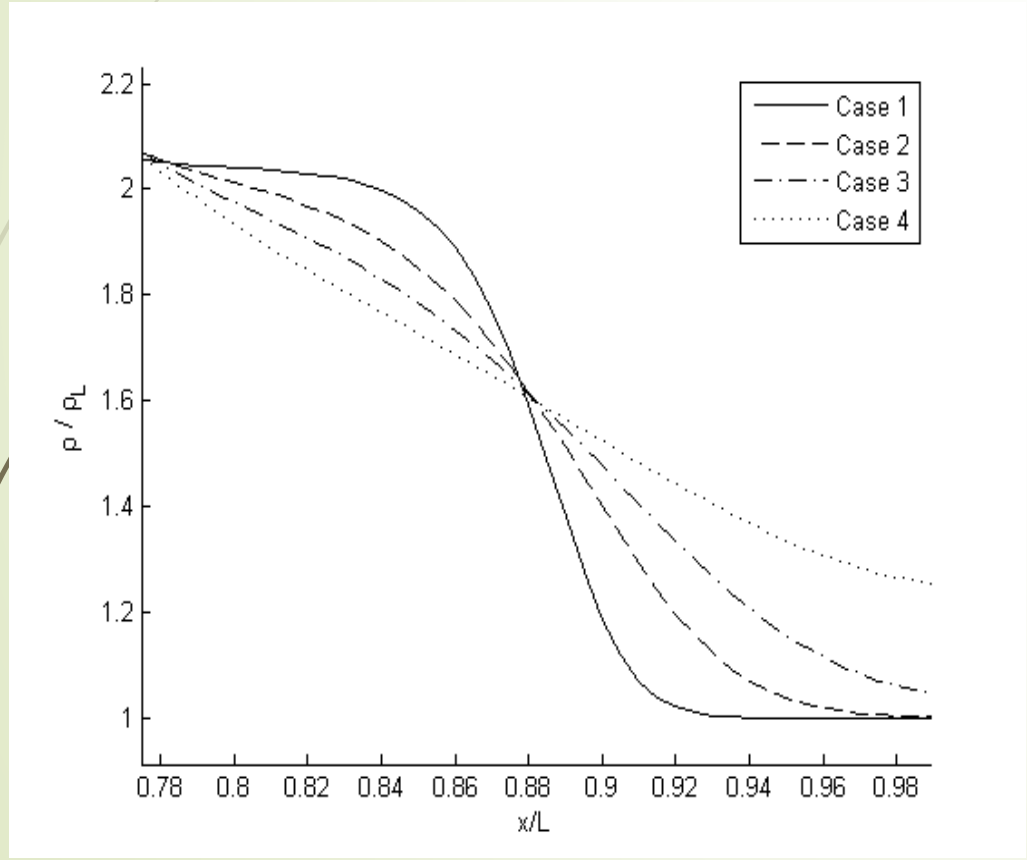
5	0.47
6	2.36
7	4.68
8	9.01

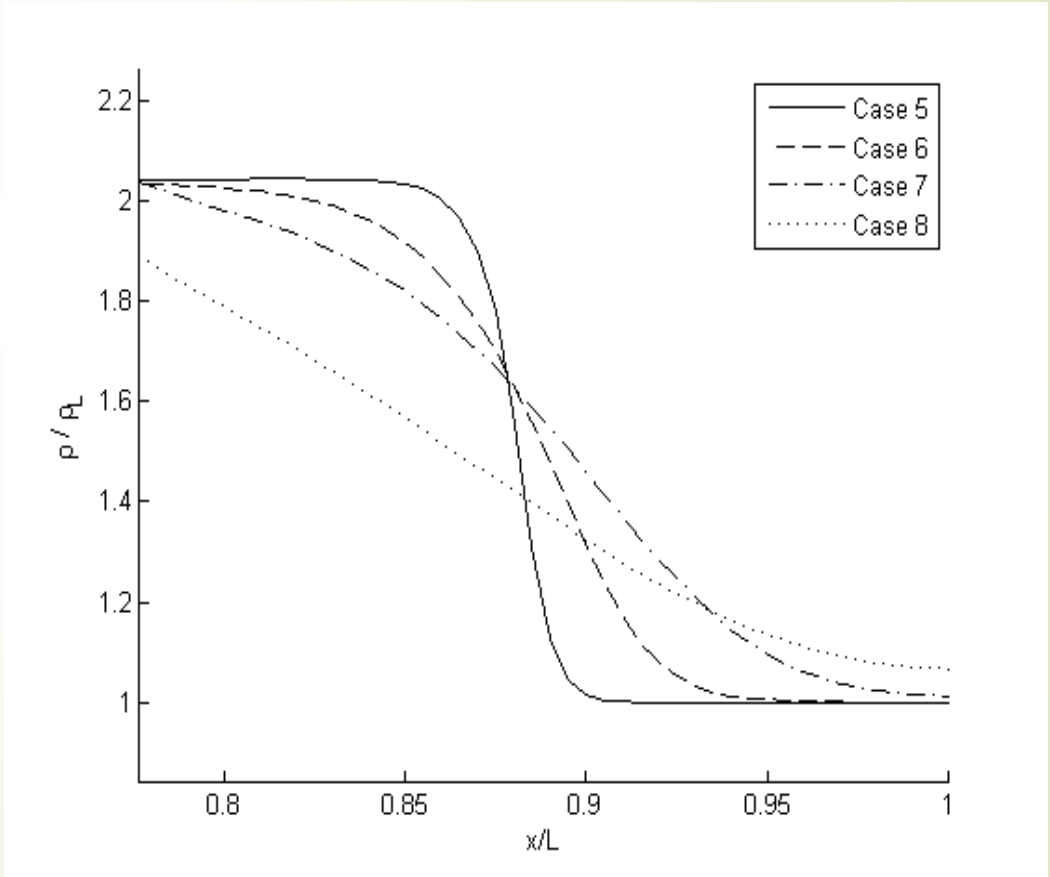
- Very few people employ a (spatially) first order accurate solver.
- Hence, we also need to investigate the influence the 2nd order extension (in space) plays on the results and convergence.
- Let's have a look at 1st order in time, 2nd order in space results.



Influence of Spatial Accuracy

- We can see that increasing the spatial accuracy while maintaining the temporal accuracy does help – but not for larger CFL numbers.





Influence of Spatial Accuracy

- We also can see that the extension to higher spatial accuracy – through reconstruction of split fluxes F – has a negative influence on the size of the Condition number of J.

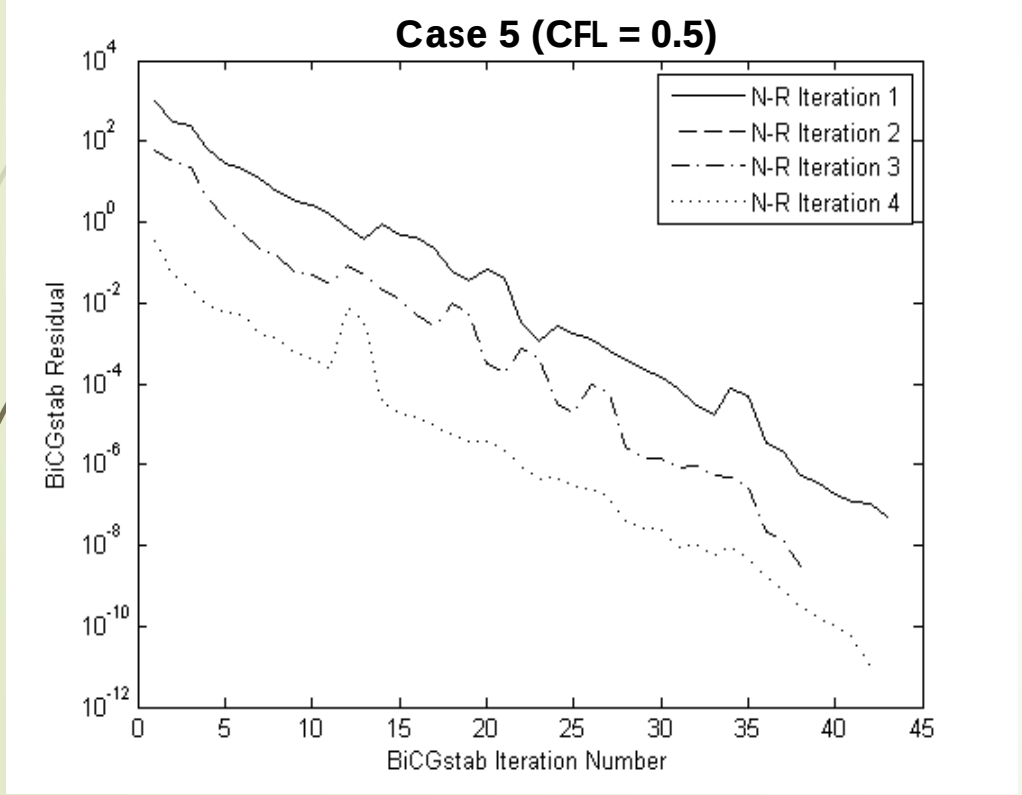
- Here, going from a CFL number of 0.5 to ~9, we see an increase in the Condition number of ~25x. Again, this is after we apply a preconditioner to the problem.

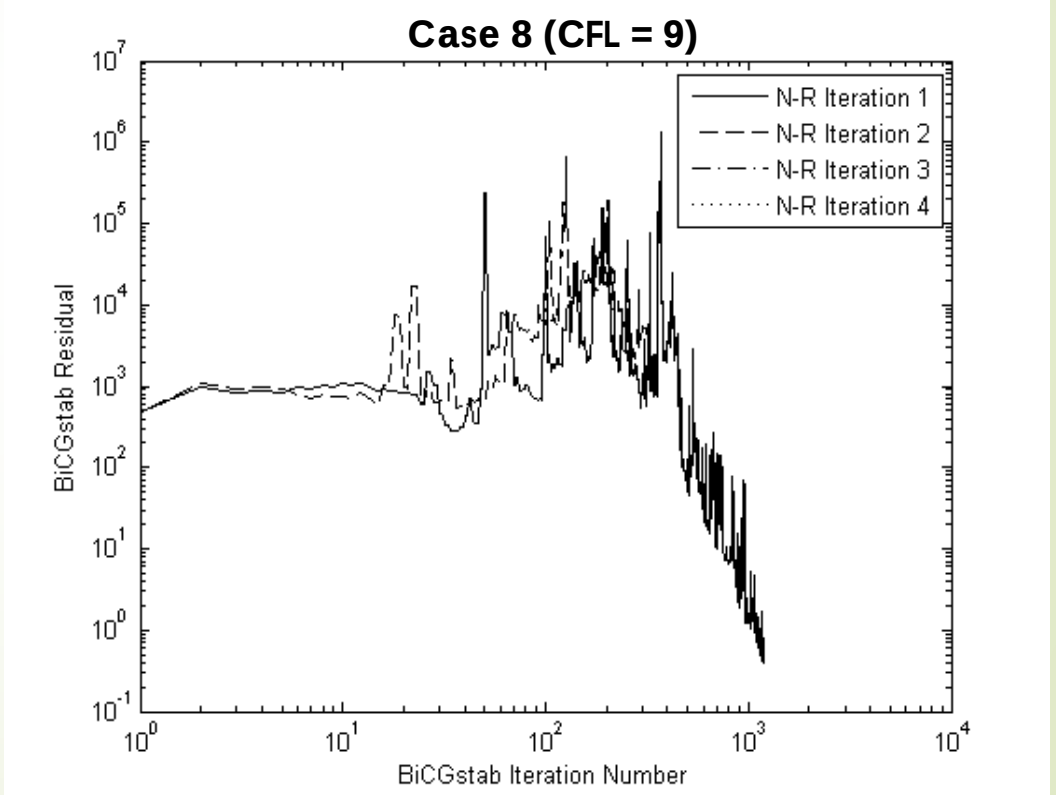
5	0.47
6	2.36
7	4.68
8	9.01

	Case 5	Case 6	Case 7	Case 8
Normalized Average Condition Number [i.e. cond(M-1J)] Accuracy - O(1) (time) / O(2) (space)	1.08	3.26	6.31	25.11

Influence of Spatial Accuracy

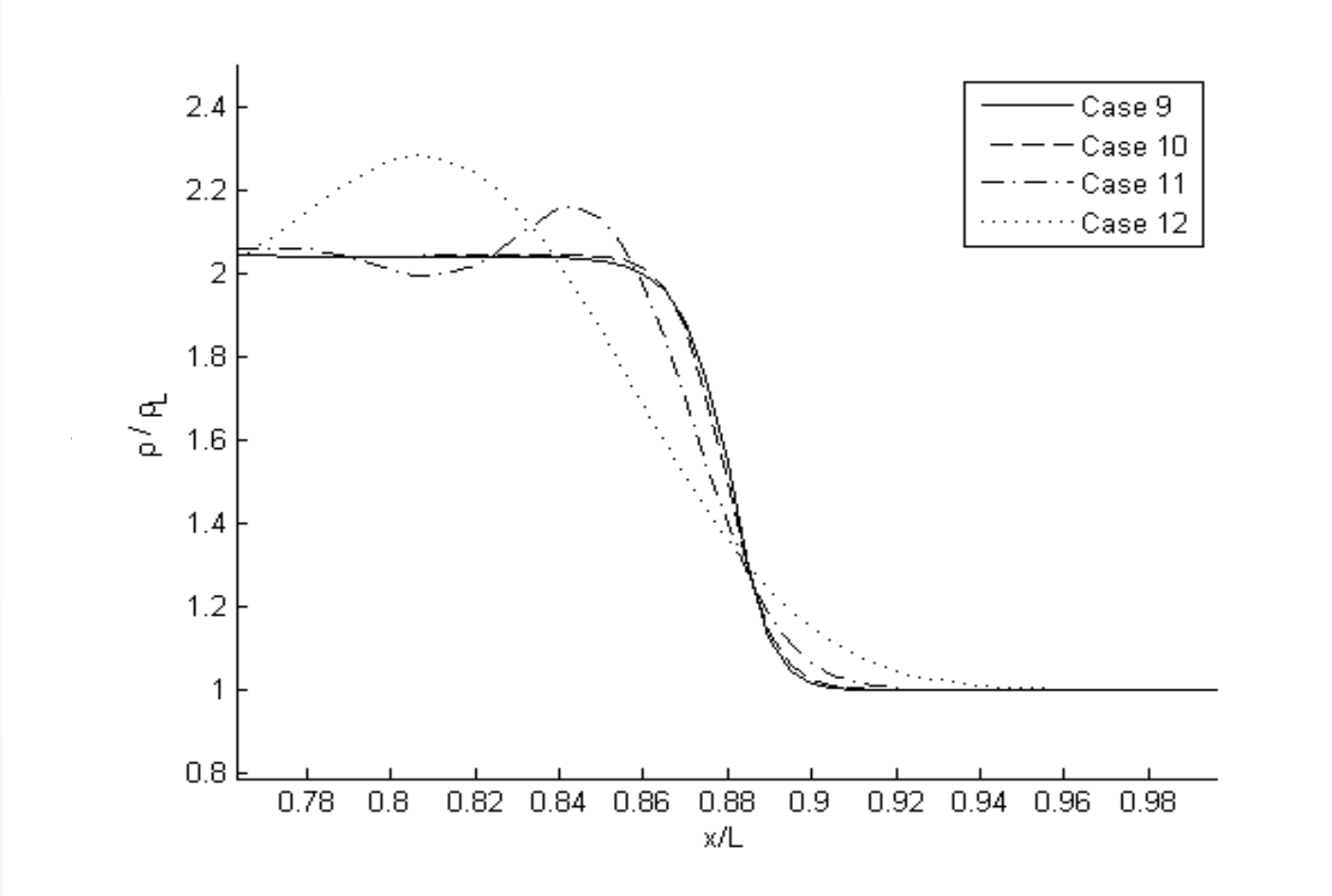
- Under severe conditions, the BiCGstab method does not converge when a 2nd order (space) / 1st order (time) solver is applied.





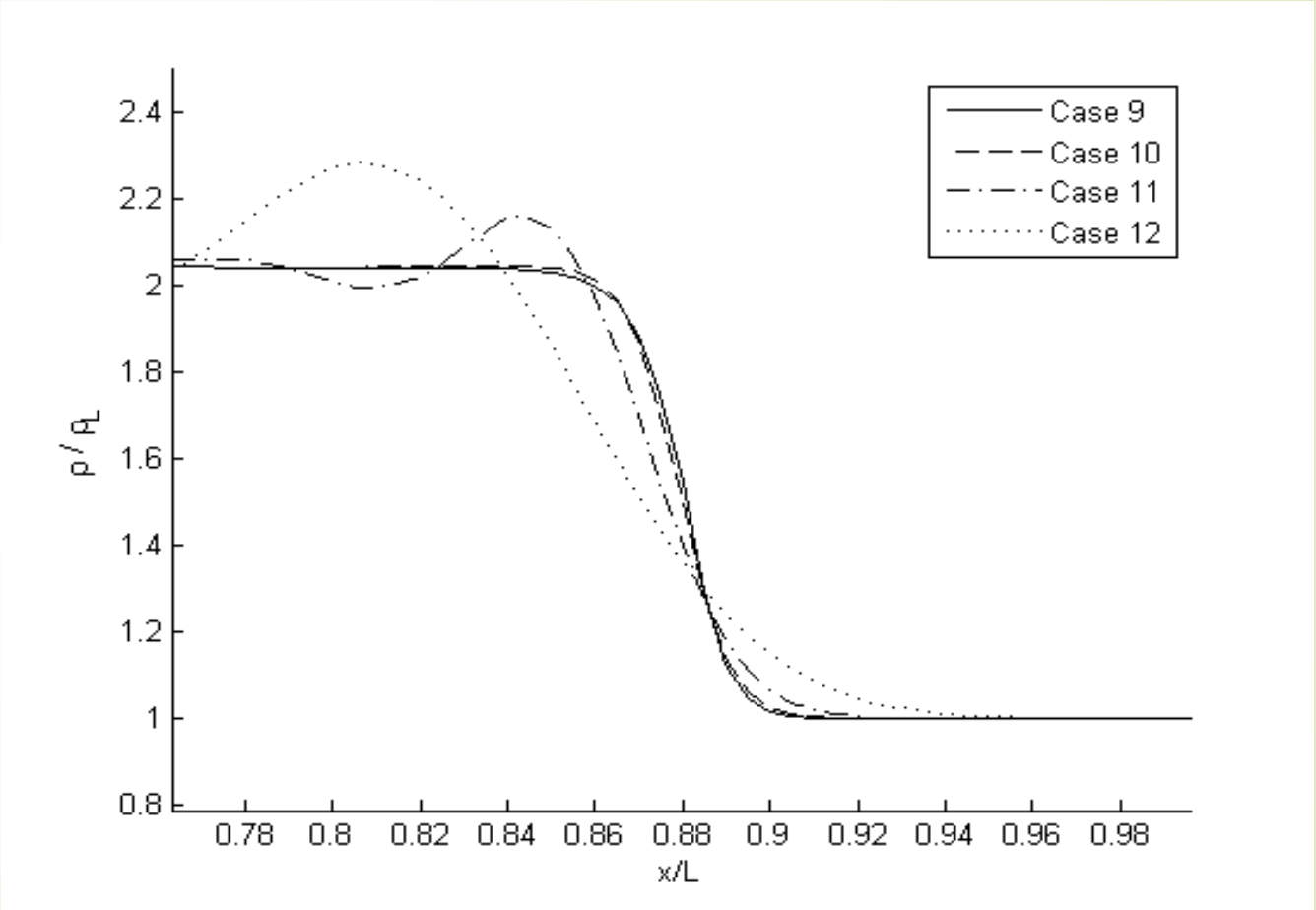
Influence of Temporal Accuracy

- Until now, the presented results employ a 1st order accurate time discretization (for the sake of analysis).
- Let's see the influence of the 2nd order accuracy in time implementation.

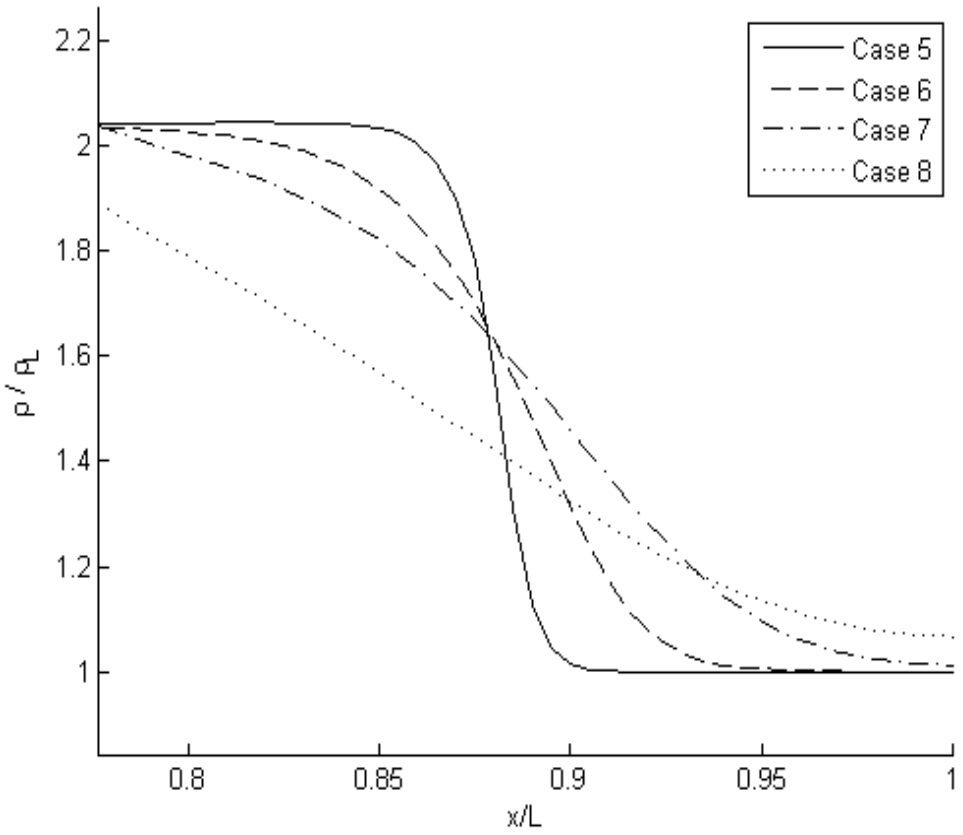


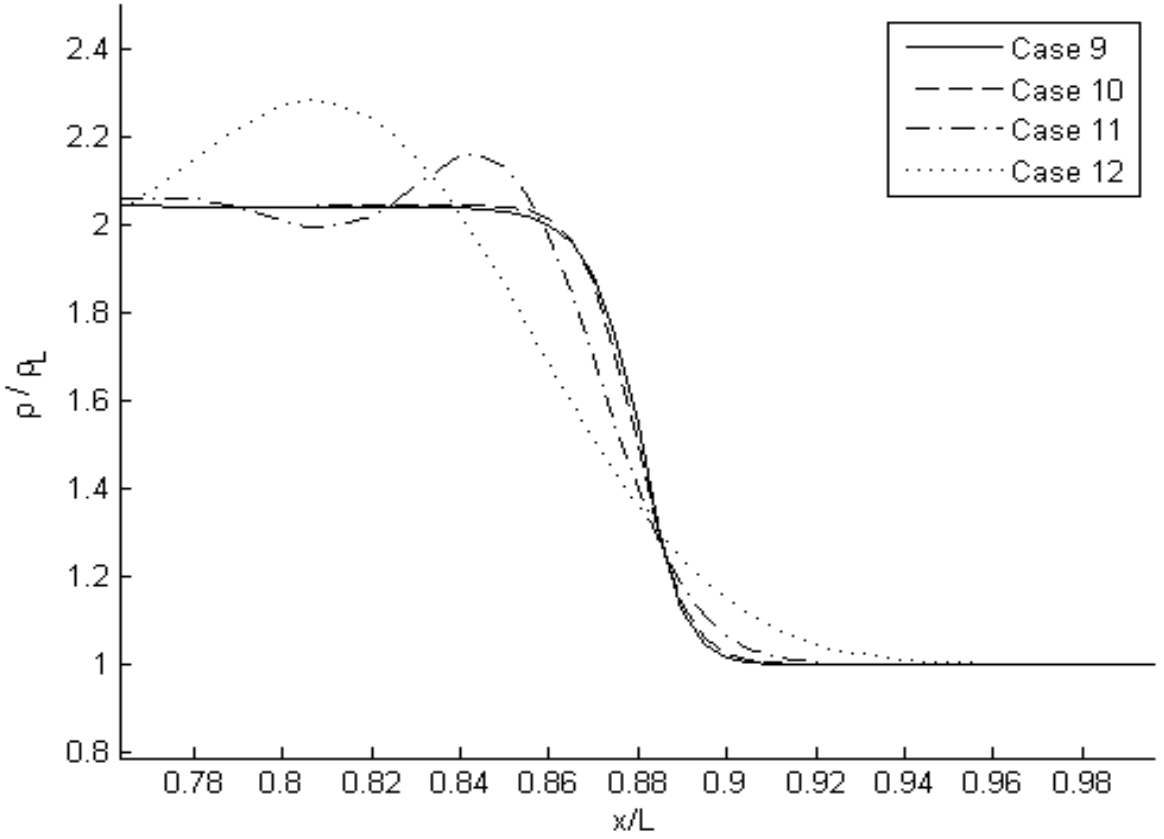
Influence of Temporal Accuracy

- We can see that the solution is much better behaved – but we have introduced some stable (non-growing) overshoot behavior similar to Gibbs phenomenon.
- The amount of introduced (numerical) diffusion from the use of larger CFL numbers has been negated.
- Hence, the diffusion observed earlier was not (only) due to the physical nature of the solver.



Influence of Temporal Accuracy





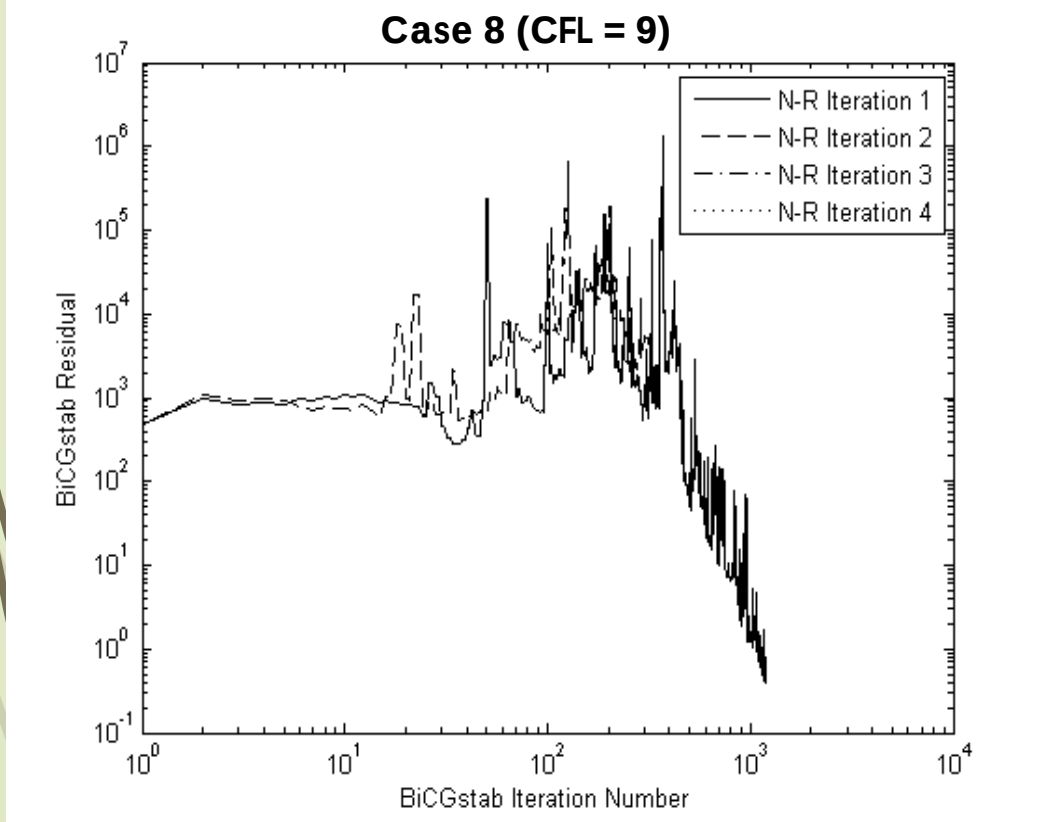
Influence of Temporal Accuracy

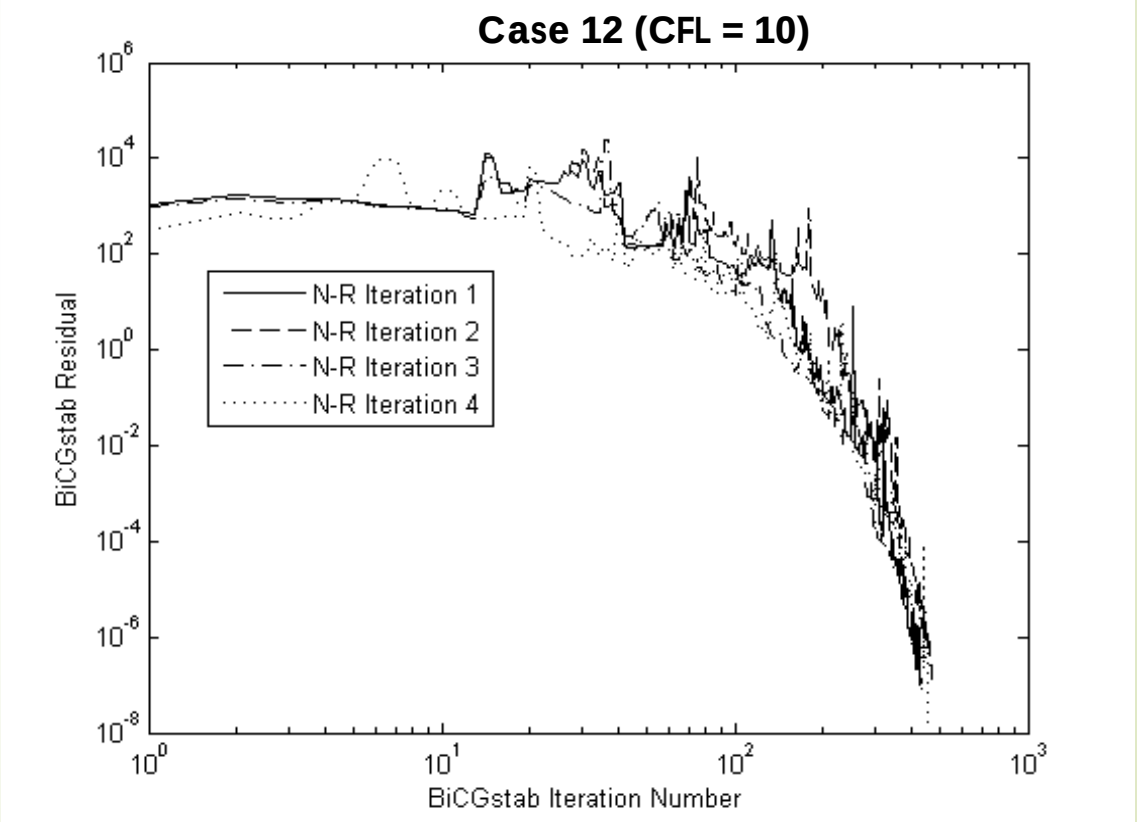
- The increase in temporal accuracy has another benefit – we decrease the condition number of the J matrix.
- Were, we have moved from a 25x increase to a ~6.5x increase!
- As a result, we can see that implementing a 2nd order in time solver leads to a significant decrease in computational time when compared to a first order in time implementation.

	Case 9	Case 10	Case 11	Case 12
Normalized Average Condition Number [i.e. cond(M-1J)] Accuracy - O(2) (time) / O(2) (space)	0.84	1.86	3.27	6.24

Influence of Temporal Accuracy

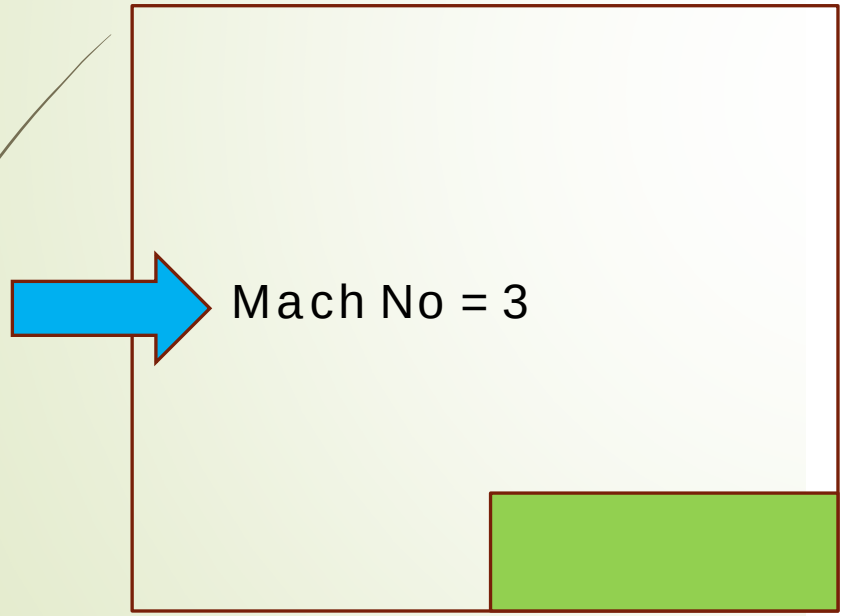
- Compare the convergence properties:

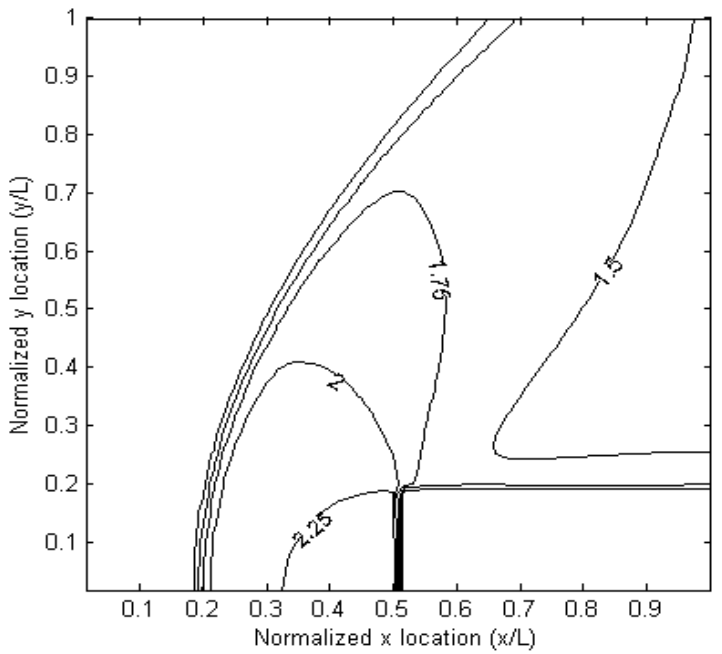




Steady 2D Flow - Test

- To test the application of this approach to steady flows, I trust 2D test cases most.
- Here, we use hypersonic (M=3) flow over a forward facing step.

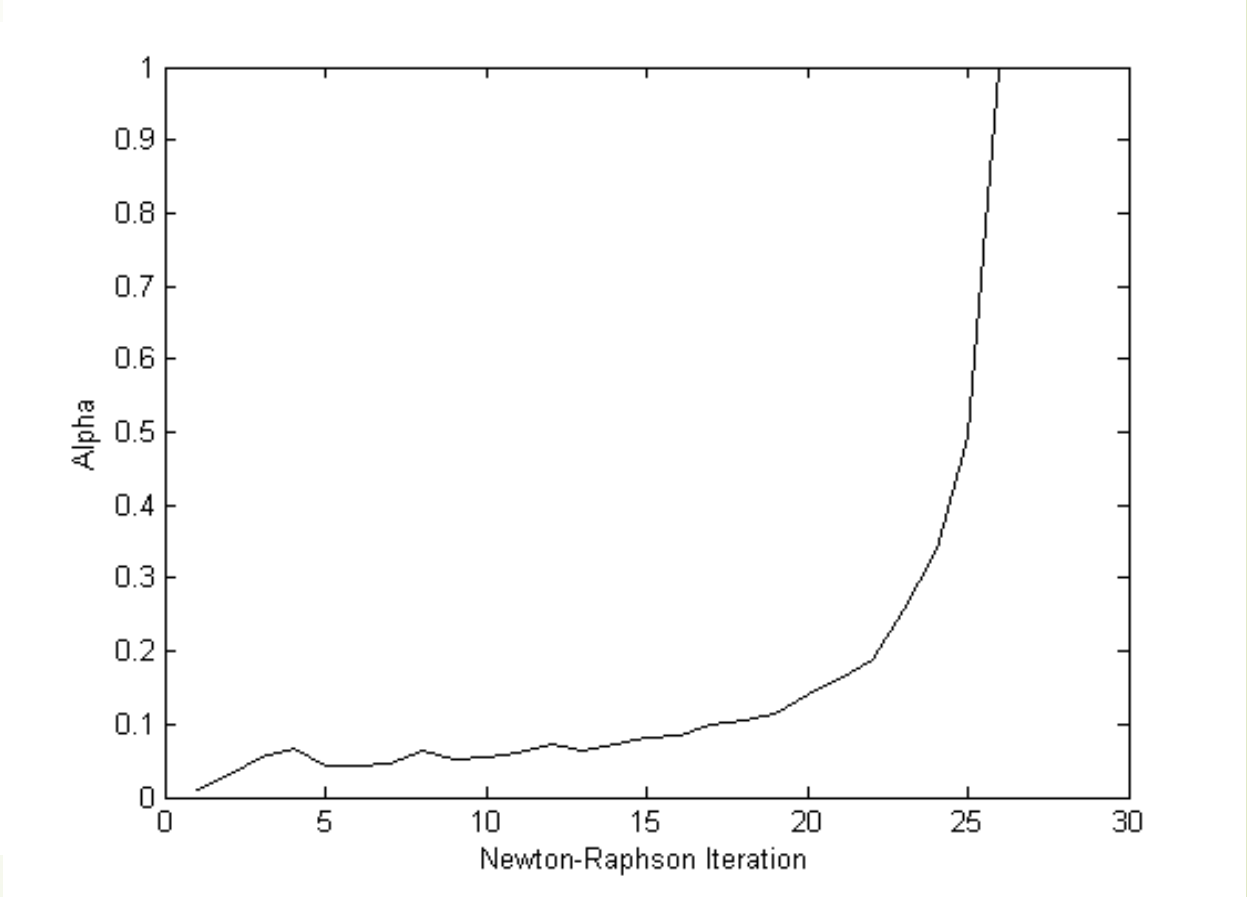




Steady 2D Flow - Test

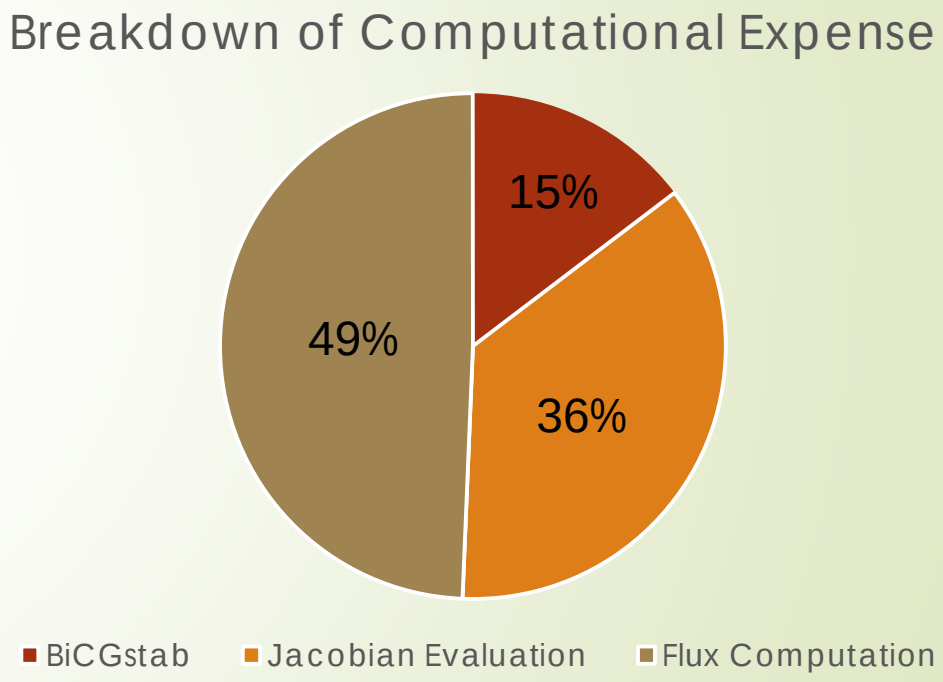
- The BiCGstab convergence behavior in this problem was very nice.
- Great care had to be taken, however – this was only due to the very small Newton-Raphson steps taken at the start.
- For larger CFL numbers in general – which includes steady flows – alpha is important in maintaining stability:

$$x^* = x - \alpha(J^{-1}R(x))$$



Results and Parallel Performance

- In order to make sense of the parallel performance, we need an estimate of the breakdown of computational expense.
- The flux computation phase represents the majority of the computational effort.
- The good news – the Jacobian Evaluation and Flux computation phases are embarrassingly parallel.



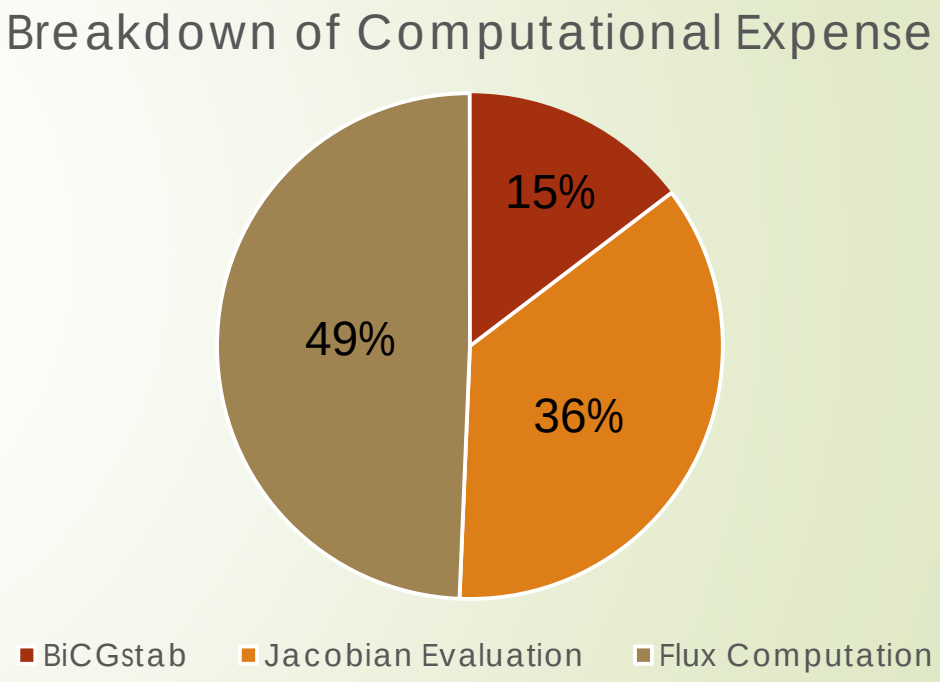
Results and Parallel Performance

- The overall computational speedup can be shown in terms of the speedup for each component:

Component	Flux	Jacobian	BiCGstab	Total
CPU time	21.2s	15.5s	6.3s	43s
GPU time	0.38s	0.28s	0.18s	0.85s
Speedup	~55x	~55x	~35x	~50x

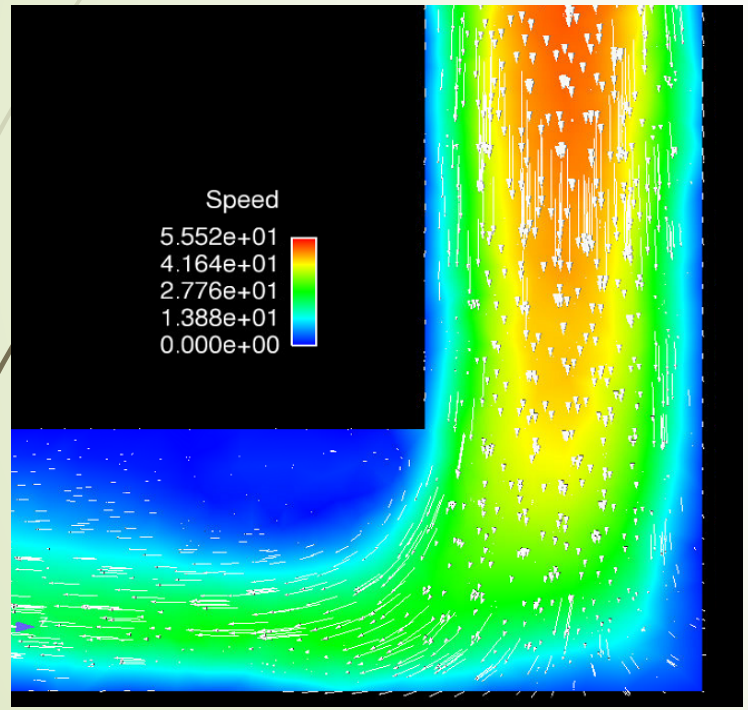
3D unstructured problem using GTX-Titan

- The performance increases further for structured grids.



Results and Parallel Performance

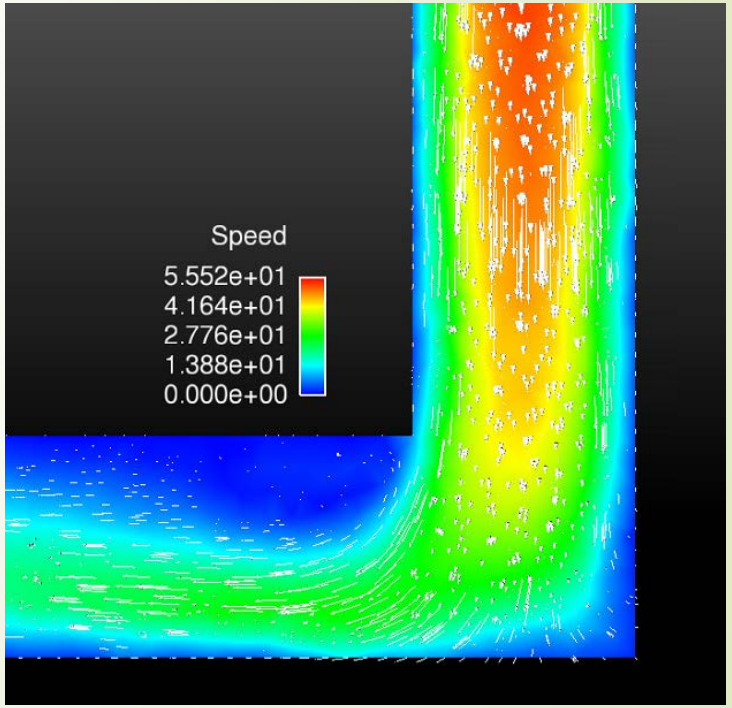
- Of course, we can apply this to other steady problems on different heterogeneous devices…



Intel PHI Parallelization

This results in the requirement for more NR steps and BiCGstab steps.

GPU Parallelization



Wrapping thing up..

- In this work we've developed a family of implicit and explicit FVM solvers for parallelization on various heterogeneous parallel computing architectures.
- The work presented today showed some of the things we've discovered about the EFM (KFVS) solver applied to Implicit computation – particularly in terms of the role the spatial and temporal order of accuracy plays on the computational effort required for the solution.
- We conclude that higher orders in spatial accuracy result in a J matrix with a higher Condition number, while higher orders in time result in lower condition numbers. (Influence of Time > Space, here).
- The goal – conclude, for application on GPU devices – which is a better approach – explicit or implicit.

Questions?

- Contact Details: Prof. Matthew Smith, NCKU, msmith@mail.ncku.edu.tw