3 – Efficient Solvers and Time Integration

Per-Olof Persson

Department of Mathematics, University of California, Berkeley
Mathematics Department, Lawrence Berkeley National Laboratory

Summer School on Discontinuous Galerkin Methods
International Center for Numerical Methods in Engineering
Barcelona, Spain

University of California Berkeley

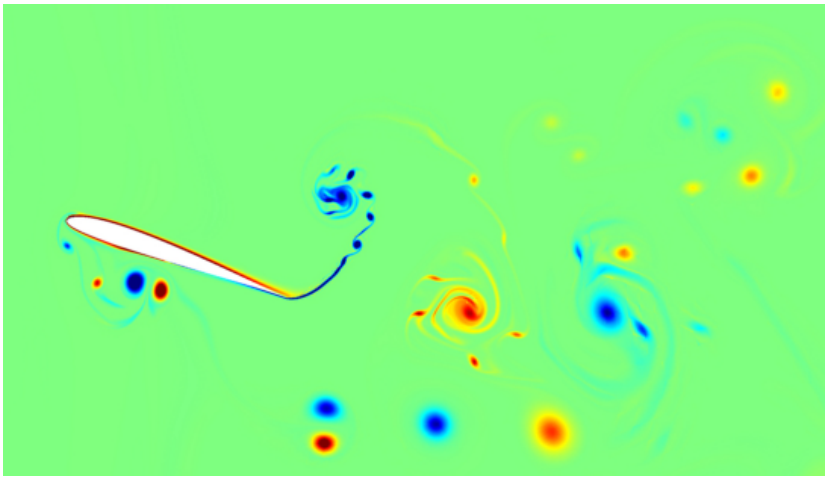
June 11-15, 2012

Berkeley Lab

Outline

Implicit Time Integration

- Implicit solvers typically required because of CFL restrictions from viscous effects, low Mach numbers, and adaptive/anisotropic grids
- Jacobian matrices are large even at $p = 2$ or $p = 3$, however:
  - They are required for non-trivial preconditioners
  - They are very expensive to recompute
- Therefore, we consider matrix-based Newton-Krylov solvers
- In [Persson/Peraire, SISC'08], we proposed efficient preconditioners with block-ILU(0) and automatic element ordering
- Distributed parallel versions developed in [Persson '09]



Problem Formulation

- Consider time-dependent systems of conservation laws:

$$\frac{\partial u}{\partial t} + \nabla \cdot F(u, \nabla u) = S(u, \nabla u)$$

- Standard DG discretization using degree p simplex elements
- Approximate Riemann solver for inviscid fluxes, the Compact Discontinuous Galerkin (CDG) method for viscous fluxes [Peraire/Persson, SISC'08]
- Leads to system of ODEs of the form $\boldsymbol{M}\dot{\boldsymbol{u}} = \boldsymbol{r}(\boldsymbol{u})$
- Implicit BDF or DIRK methods with Newton's method lead to linear systems with matrices of the form

$$\boldsymbol{A} \equiv \alpha_0 \boldsymbol{M} - \Delta t \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}} \equiv \alpha_0 \boldsymbol{M} - \Delta t \boldsymbol{K}$$

- Strong dependence on the timestep Δt

Diagonally Implicit Runge-Kutta (DIRK) Methods

- General form of Runge-Kutta methods

$$k_i = f\left(t_n + c_i h, u_n + h\sum_{j=1}^{s} a_{ij}k_j\right), \quad i = 1, \ldots, s$$

$$u_{n+1} = u_n + h\sum_{j=1}^{s} b_j k_j$$

- DIRK schemes:

  Lower triangular

  RK tablaux

c_1	a_{11}		
$\vdots$	$\vdots$	$\ddots$	
c_s	a_{s1}	$\cdots$	a_{ss}
	b_1	$\cdots$	b_s

- Solve with Newton's method

$$F_1(k_1) = k_1 - f(t_n + c_1 h, u_n + h a_{11} k_1)$$

$$F_2(k_2) = k_2 - f(t_n + c_2 h, u_n + h a_{21} k_1 + h a_{22} k_2)$$

$$\vdots$$

for s systems of N equations.

Examples of DIRK Schemes

2-stage, 2nd order DIRK, L-stable

$$\begin{array}{c|cc} \alpha & \alpha & 0 \\ 1 & 1-\alpha & \alpha \\ \hline & 1-\alpha & \alpha \end{array}$$

where $\alpha = 1 - \frac{\sqrt{2}}{2}$

2-stage, 3rd order DIRK, A-stable, not L-stable

$$\begin{array}{c|cc} \alpha & \alpha & 0 \\ 1-\alpha & 1-2\alpha & \alpha \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

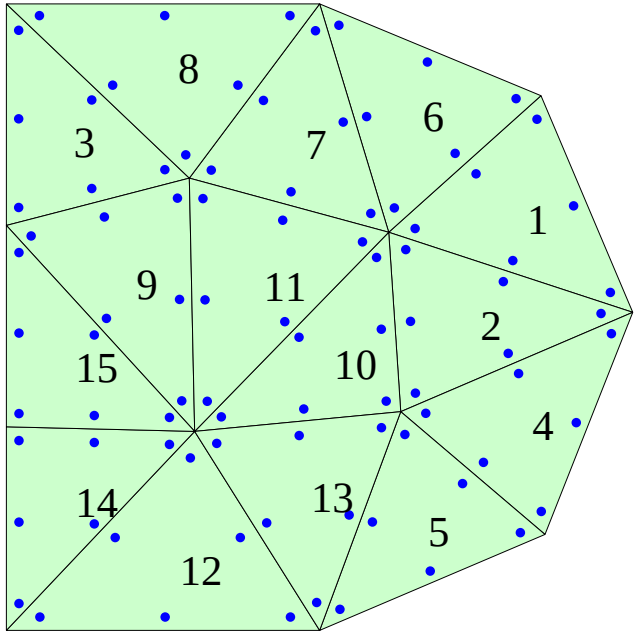
where $\alpha = (3 + \sqrt{3})/6$.

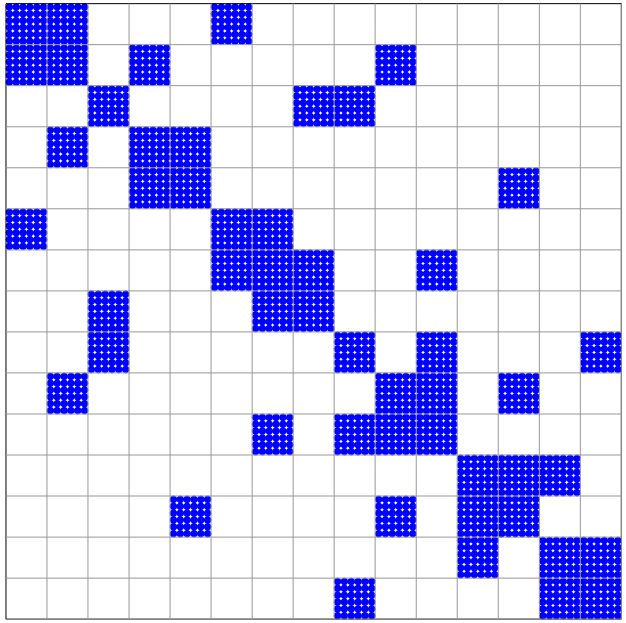
3-stage, 3rd order DIRK, L-stable

$$\begin{array}{c|ccc} 0.43586652 & 0.43586652 & 0 & 0 \\ 0.71793326 & 0.28206673 & 0.43586652 & 0 \\ 1 & 1.2084966 & -0.64436317 & 0.43586652 \\ \hline & 1.2084966 & -0.64436317 & 0.43586652 \end{array}$$

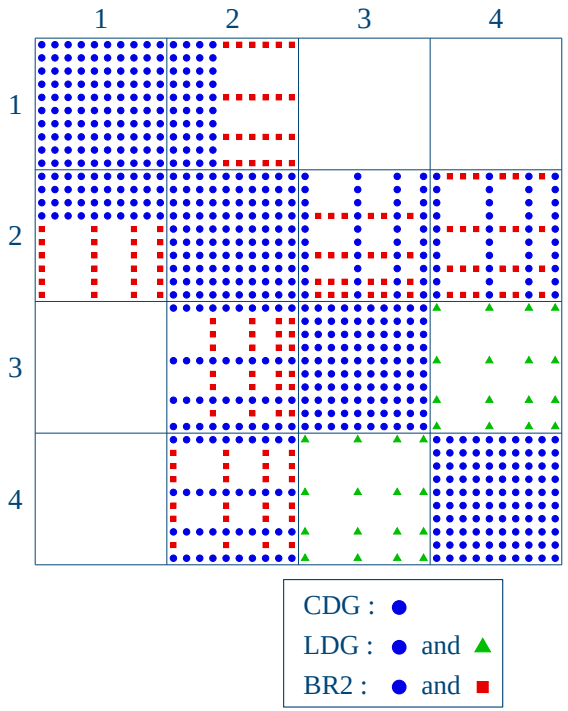
Jacobian Matrix Sparsity Structure

- Dual mesh connectivity, with each entry a large complete graph $\boldsymbol{A}_{ij}$
- Off-diagonal blocks actually sparser with CDG, but assume dense for simplicity
- Size N of submatrices $\boldsymbol{A}_{ij}$ is often > 100
- Block-based storage format essential for high performance using BLAS routines
- For other structures (e.g. elimination matrices), use block-wise compressed column





Computational Cost – DG Jacobian Matrices



- Number of non-zeros in Jacobian matrix for nodal DG (C solution components, T simplex elements, degree p, dimension D)
- Nodes per element $S = \binom{p+D}{p} = \mathcal{O}(p^D)$, nodes per face $s = \binom{p+D-1}{D-1} = \mathcal{O}(p^{D-1})$
- Example: 3-D Navier-Stokes, $p = 3$, $T = 100,000$:

Quantity	Size	Example storage
Solution vector	SCT	80 MB
Inviscid Jacobian	$(S^2 + (D+1)s^2)C^2T$	16 GB
CDG Jacobian	$(S^2 + (D+1)Ss)C^2T$	24 GB
BR2/IP Jacobian	$(S^2 + (D+1)(S+s)s)C^2T$	32 GB
Full block Jacobian	$(D+2)S^2C^2T$	40 GB

Outline

Data Structures for Sparse Matrices

Full:

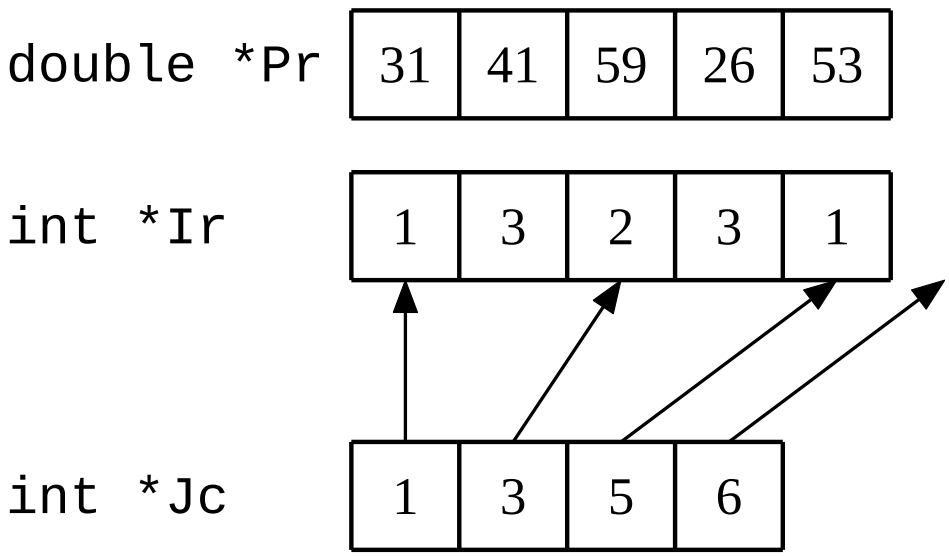
- Storage: Array of real (or complex) numbers
- Memory: `nrows*ncols`

31	0	53
0	59	0
41	26	0

`double *A`

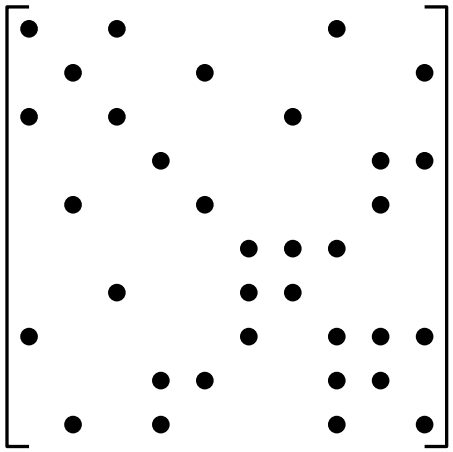
Sparse:

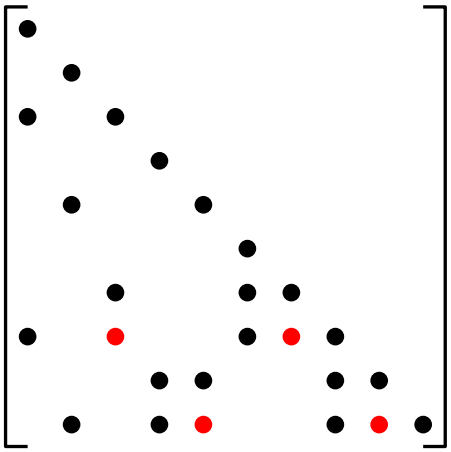
- Compressed column storage
- Memory: About `1.5*nnz+.5*ncols`



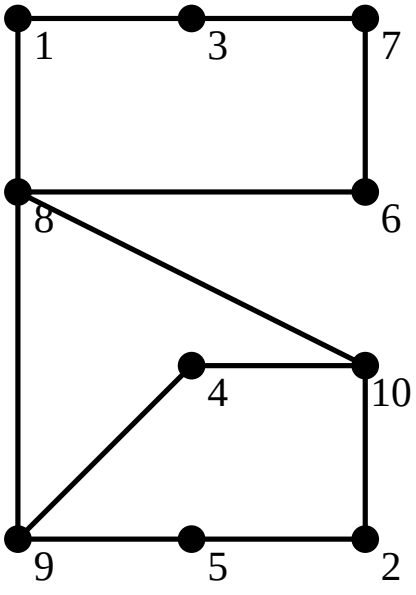
- Element look-up: $\mathcal{O}(\log$ #elements in column$)$ time
- Insertion of new nonzero very expensive
- Sparse vector = Column vector (not Row vector)

Graphs and Sparsity: Cholesky Factorization

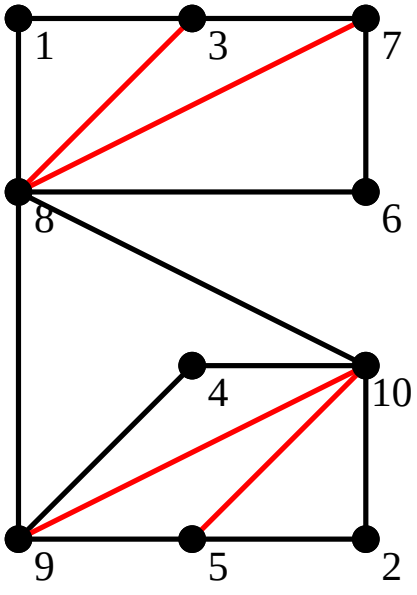




Fill: New nonzeros in factor



$G(A)$



$G^+(A)$

Symmetric Gaussian Elimination:

for $j = 1$ **to** N

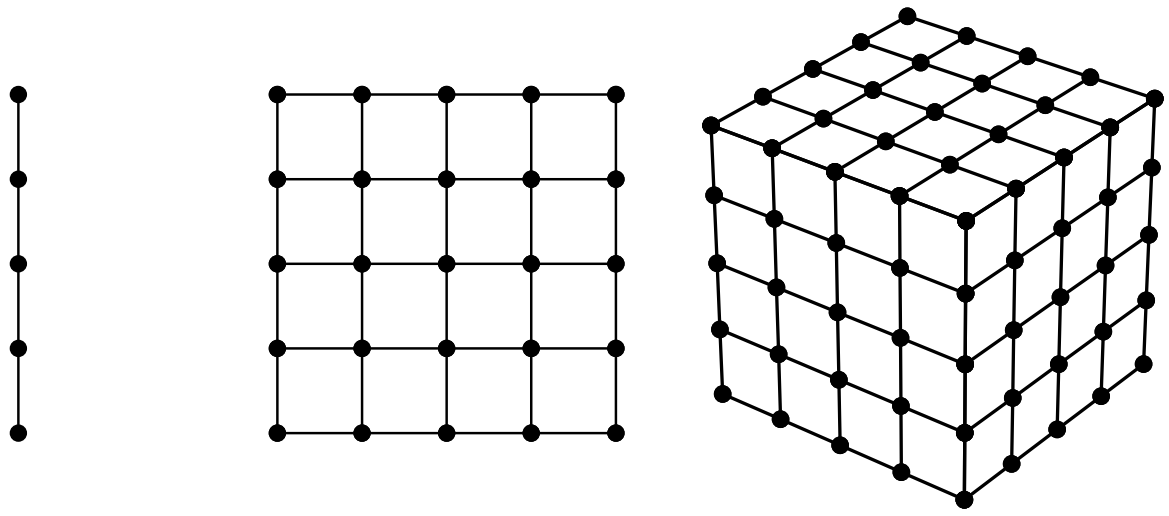
Add edges between j's higher-numbered neighbors

Heuristic Fill-Reducing Matrix Permutations

- Banded orderings (Reverse Cuthill-McKee, Sloan, etc):
  - Try to keep all nonzeros close to the diagonal
  - Theory, practice: Often wins for "long, thin" problems
- Minimum degree:
  - Eliminate row/col with fewest nonzeros, add fill, repeat
  - Hard to implement efficiently – current champion is "Approximate Minimum Degree" [Amestoy, Davis, Duff]
  - Theory: Can be suboptimal even on 2-D model problem
  - Practice: Often wins for medium-sized problems
- Nested dissection:
  - Find a separator, number it *last*, proceed recursively
  - Theory: Approximately optimal separators $\Longrightarrow$ approximately optimal fill and flop count
  - Practice: Often wins for very large problems
- The best modern general-purpose orderings are ND/MD hybrids

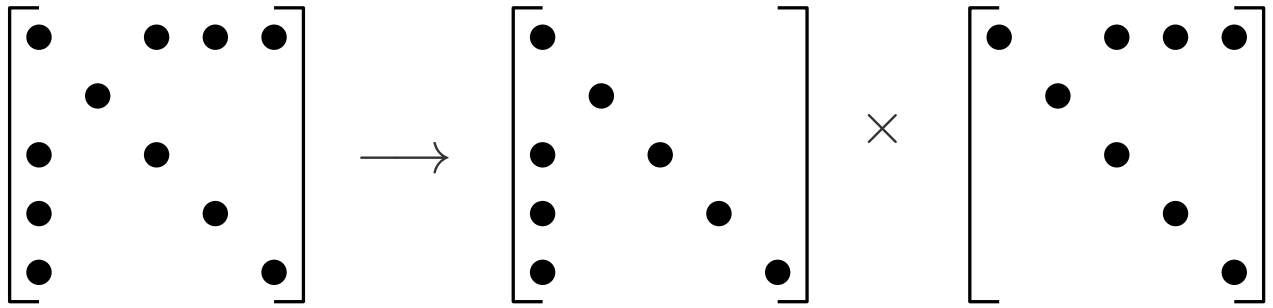
Complexity of Direct Methods

- Time and space to solve any problem on any well-shaped finite element mesh with N nodes:



	1-D	2-D	3-D
Space (fill):	$\mathcal{O}(N)$	$\mathcal{O}(N \log N)$	$\mathcal{O}(N^{4/3})$
Time (flops):	$\mathcal{O}(N)$	$\mathcal{O}(N^{3/2})$	$\mathcal{O}(N^{2})$

Incomplete Cholesky Factorization (IC, ILU)

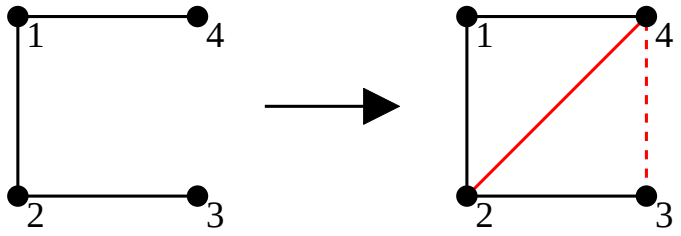


A $\qquad$ R^T $\qquad$ R

- Compute factors of A by Gaussian elimination, but ignore fill
- Preconditioner $B = R^T R \approx A$, not formed explicitly
- Compute $B^{-1}z$ by triangular solves in time $O(\texttt{nnz(A)})$
- Total storage is $O(\texttt{nnz(A)})$, static data structure
- Either symmetric (IC) or nonsymmetric (ILU)

Incomplete Cholesky and ILU: Variants

- Allow one or more "levels of fill"
  - Unpredictable storage requirements



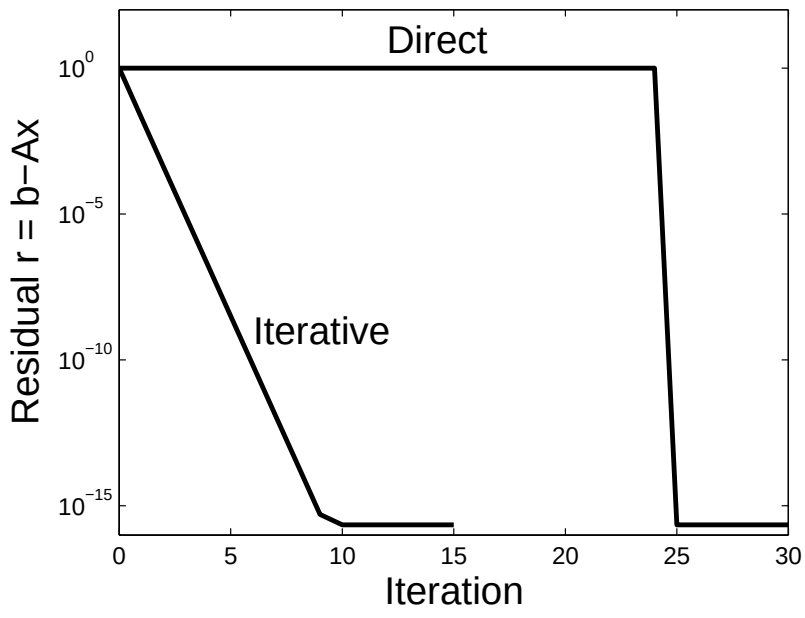
- Allow fill whose magnitude exceeds a "drop tolerance"
  - May get better approximate factors than levels of fill
  - Unpredictable storage requirements
  - Choice of tolerance is ad hoc
- Partial pivoting (for nonsymmetric A)
- "Modified ILU" (MIC): Add dropped fill to diagonal of U (R)
  - A and $R^T R$ have same row sums
  - Good in some PDE contexts

Incomplete Cholesky and ILU: Issues

- Choice of parameters
  - Good: Smooth transition from iterative to direct methods
  - Bad: Very ad hoc, problem-dependent
  - Trade-off: Time per iteration vs # of iterations (more fill $\rightarrow$ more time but fewer iterations)
- Effectiveness
  - Condition number usually improves (only) by constant factor (except MIC for some problems from PDEs)
  - Still, often good when tuned for a particular class of problems
- Parallelism
  - Triangular solves are not very parallel
  - Reordering for parallel triangular solve by graph coloring

Iterative Methods for Linear Systems

- *Direct methods* for solving $Ax = b$, e.g. Gaussian elimination, compute an exact solution after a finite number of steps (in exact arithmetic)
- *Iterative algorithms* produce a sequence of approximations $x^{(1)}, x^{(2)}, \ldots$ which hopefully converges to the solution, and
  - may require less memory than direct methods
  - may be faster than direct methods
  - may handle special structures (such as sparsity) in a simpler way



Two Classes of Iterative Methods

- *Stationary methods* (or classical iterative methods) finds a splitting $A = M - K$ and iterates $x^{(k+1)} = M^{-1}(Kx^{(k)} + b) = Rx^{(k)} + c$
  - Jacobi, Gauss-Seidel, Successive Overrelaxation (SOR), and Symmetric Successive Overrelaxation (SSOR)
- *Krylov subspace methods* use only multiplication by A (and possibly by A^T) and find solutions in the Krylov subspace $\{b, Ab, A^2b, \ldots, A^{k-1}b\}$
  - Conjugate Gradient (CG), Generalized Minimal Residual (GMRES), BiConjugate Gradient (BiCG), etc

Krylov Subspace Algorithms

- Create a sequence of *Krylov subspaces* for $Ax = b$:

$$\mathcal{K}_n = \left\langle b, Ab, \ldots, A^{n-1}b \right\rangle$$

  and find approximate solutions x_n in $\mathcal{K}_n$
- Only matrix-vector products involved
- For SPD matrices, the most popular algorithm is the *Conjugate Gradients* method [Hestenes/Stiefel, 1952]
  - Finds the best solution $x_n \in \mathcal{K}_n$ in the norm $\|x\|_A = \sqrt{x^T A x}$
  - Only requires storage of 4 vectors (not all the n vectors in $\mathcal{K}_n$)
  - Remarkably simple and excellent convergence properties
  - Originally invented as a direct algorithm! (converges after m steps in exact arithmetic)

The Conjugate Gradients Method

Algorithm: Conjugate Gradients Method

$x_0 = 0, r_0 = b, p_0 = r_0$

for $k = 1, 2, 3, \ldots$

$\alpha_n = (r_{n-1}^T r_{n-1})/(p_{n-1}^T A p_{n-1})$	step length
$x_n = x_{n-1} + \alpha_n p_{n-1}$	approximate solution
$r_n = r_{n-1} - \alpha_n A p_{n-1}$	residual
$\beta_n = (r_n^T r_n)/(r_{n-1}^T r_{n-1})$	improvement this step
$p_n = r_n + \beta_n p_{n-1}$	search direction

- Only one matrix-vector product Ap_{n-1} per iteration
- Operation count $O(m)$ (excluding the matrix-vector product)

Properties of the Conjugate Gradients Method

- The errors $e_n = x_* - x_n$ are minimized in the A-norm
- Monotonic: $\|e_n\|_A \leq \|e_{n-1}\|_A$, and $e_n = 0$ in $n \leq m$ steps
- CG also minimizes the quadratic function $\varphi(x) = \frac{1}{2}x^T Ax - x^T b$
- Conjugate Gradients finds an optimal polynomial $p_n \in P_n$ of degree n with $p(0) = 1$, minimizing $\|p_n(A)e_0\|$ with initial error $e_0 = x_*$
- Important convergence results can be obtained from the polynomial approximation:
  1. If A has n distinct eigenvalues, CG converges in at most n steps
     Proof. The polynomial $p(x) = \Pi_{j=1}^{n}(1 - x/\lambda_j)$ is zero at $\Lambda(A)$
  2. If A has 2-norm condition number κ, the errors are

$$\frac{\|e_n\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n \approx 2\left(1 - \frac{2}{\sqrt{\kappa}}\right)^n$$

- In general: CG performs well with clustered eigenvalues

Preconditioners for Linear Systems

- Main idea: Instead of solving

$$Ax = b$$

  solve, using a nonsingular $m \times m$ *preconditioner* M,

$$M^{-1}Ax = M^{-1}b$$

  which has the same solution x

- Convergence properties based on $M^{-1}A$ instead of A
- Trade-off between the cost of applying M^{-1} and the improvement of the convergence properties. Extreme cases:
  - $M = A$, perfect conditioning of $M^{-1}A = I$, but expensive M^{-1}
  - $M = I$, "do nothing" $M^{-1} = I$, but no improvement of $M^{-1}A = A$

Preconditioned Conjugate Gradients

- To keep symmetry, solve $(C^{-1}AC^{-*})C^*x = C^{-1}b$ with $CC^* = M$
- Can be written in terms of M^{-1} only, without reference to C:

Algorithm: Preconditioned Conjugate Gradients Method

$x_0 = 0, r_0 = b$

$p_0 = M^{-1}r_0, z_0 = p_0$

for $n = 1, 2, 3, \ldots$

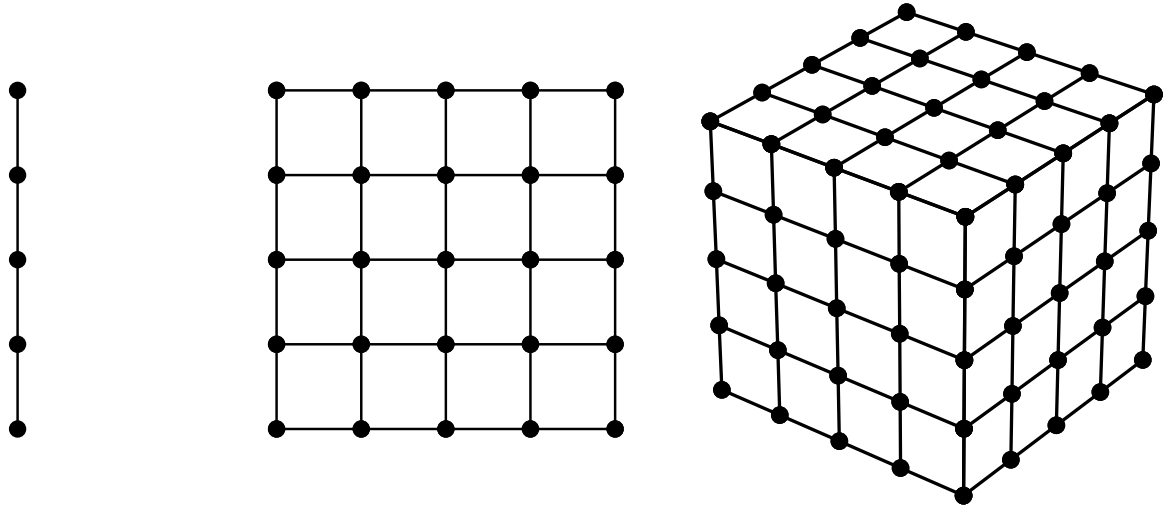
$\alpha_n = (r_{n-1}^T z_{n-1})/(p_{n-1}^T A p_{n-1})$	step length
$x_n = x_{n-1} + \alpha_n p_{n-1}$	approximate solution
$r_n = r_{n-1} - \alpha_n A p_{n-1}$	residual
$z_n = M^{-1} r_n$	preconditioning
$\beta_n = (r_n^T z_n)/(r_{n-1}^T z_{n-1})$	improvement this step
$p_n = z_n + \beta_n p_{n-1}$	search direction

Commonly Used Preconditioners

- A preconditioner should "approximately solve" the problem $Ax = b$
- **Jacobi preconditioning** - $M = \mathrm{diag}(A)$, very simple and cheap, might improve certain problems but usually insufficient
- **Block-Jacobi preconditioning** - Use block-diagonal instead of diagonal. Another variant is using several diagonals (e.g. tridiagonal)
- **Classical iterative methods** - Precondition by applying one step of Jacobi, Gauss-Seidel, SOR, or SSOR
- **Incomplete factorizations** - Perform Gaussian elimination but ignore fill, results in approximate factors $A \approx LU$ or $A \approx R^T R$
- **Coarse-grid approximations** - For a PDE discretized on a grid, transfer the residual to a coarser grid, solve a smaller problem, then transfer back an error correction (*multigrid*)

Complexity of Linear Solvers

- Work to solve a Poisson problem on regular mesh with N nodes:



Solver	1-D	2-D	3-D
Sparse Cholesky	$O(N)$	$O(N^{1.5})$	$O(N^2)$
CG, exact arith.	$O(N^2)$	$O(N^2)$	$O(N^2)$
CG, no precond.	$O(N^2)$	$O(N^{1.5})$	$O(N^{1.33})$
CG, modified IC	$O(N^{1.5})$	$O(N^{1.25})$	$O(N^{1.17})$
Multigrid	$O(N)$	$O(N)$	$O(N)$

Outline

Preconditioners for DG

- Preconditioner critical for good performance with Krylov methods
- Various standard methods for block matrices:
  - Block diagonal: $\boldsymbol{A} \approx \mathrm{blockdiag}(\boldsymbol{A})$
    - Poor in general
  - Block Incomplete LU: $\boldsymbol{A} \approx \tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}}$
    - Good for convection (with the right ordering, more later)
  - p-multigrid (low-degree) preconditioner
    - Good for diffusion
- Highly efficient combination [Persson/Peraire 06,08]
  - Use block ILU(0) as post-smoother for coarse scale correction
  - Combines advantages of ILU and low-degree preconditioners
  - Cheap general purpose preconditioner

Incomplete LU Factorization

- For our low-degree dual meshes, it is uncommon that an element k is a neighbor of both element j and i when j is a neighbor of i
- This simplifies the Block-ILU(0) algorithm:

$\tilde{\boldsymbol{U}} \leftarrow \boldsymbol{A}, \tilde{\boldsymbol{L}} \leftarrow \boldsymbol{I}$

for $j = 1$ **to** $n - 1$

  for neighbors $i > j$ of j

$\tilde{\boldsymbol{L}}_{ij} = \tilde{\boldsymbol{U}}_{ij} \tilde{\boldsymbol{U}}_{jj}^{-1}$

$\tilde{\boldsymbol{U}}_{ii} \leftarrow \tilde{\boldsymbol{U}}_{ii} - \tilde{\boldsymbol{L}}_{ik} \tilde{\boldsymbol{U}}_{ki}$

  end for

end for

- This means $\tilde{\boldsymbol{U}}_{ij} = \boldsymbol{A}_{ij}$ when $j > i$, or $\tilde{\boldsymbol{U}}$ only differs from $\boldsymbol{A}$ in the diagonal blocks
- Store only diagonal blocks of $\tilde{\boldsymbol{U}}$!

Computational Cost

- Pre-calculation cost (block-size N, dimension D, ne elements):

Operation	Flop count
Jacobi Factorization $\tilde{\boldsymbol{A}}^{J}$	$(2/3)N^3ne$
Gauss Seidel Factorization $\tilde{\boldsymbol{A}}^{GS}$	$(2/3)N^3ne$
ILU(0) Factorization $\tilde{\boldsymbol{A}}^{ILU}$	$(2D+8/3)N^3ne$

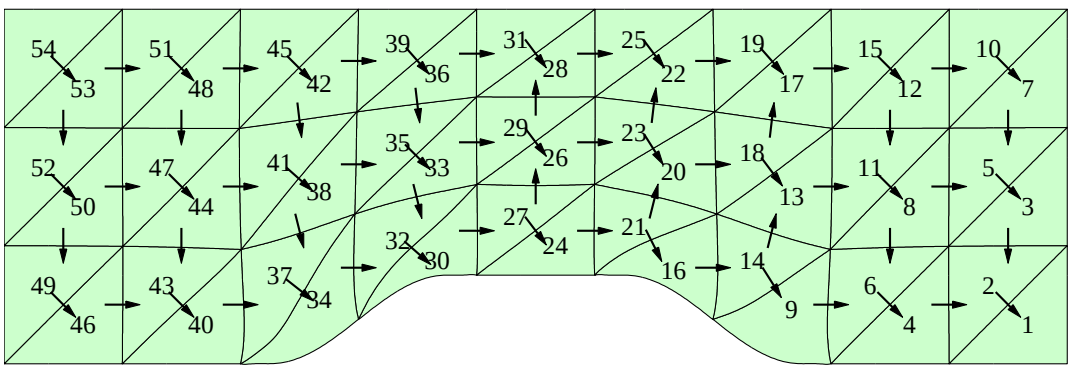
- This assumes full blocks (CDG is significantly cheaper)
- Cost of assembling a nonlinear $\boldsymbol{A}$ is $\mathcal{O}(N^3ne)$ with large constant
- Cost per iteration:

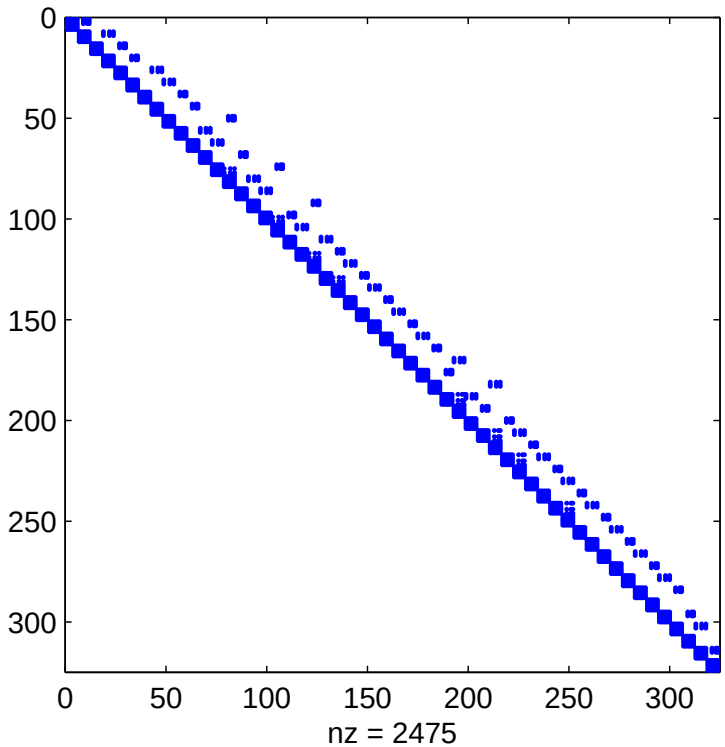
Operation	Flop count
Jacobi solve $(\tilde{\boldsymbol{A}}^{J})^{-1}p$	$(2)N^2ne$
Gauss Seidel solve $(\tilde{\boldsymbol{A}}^{GS})^{-1}p$	$(D+3)N^2ne$
ILU(0) solve $(\tilde{\boldsymbol{A}}^{ILU})^{-1}p$	$(2D+4)N^2ne$
Matrix-vector product $\boldsymbol{Ax}$	$(2D+4)N^2ne$

- Only factor of 2 between ILU + MATVEC and Jacobi + MATVEC

Element Order Dependency

- Properties of Gauss Seidel and ILU preconditioners highly dependent on the ordering of the elements
- For *upwinded scalar convection*, “ordering by lines” gives an optimal upper triangular matrix
- But for viscous or multivariate problems, best ordering not clear
- Matrix-approach: Minimize error in the approximations, rather than using physical observations





Minimum Discarded Fill Element Ordering

- Greedy algorithm for element ordering [Persson/Peraire '08]:
  At step j, if j' is chosen next, we would discard the fill

$$\Delta\tilde{\boldsymbol{U}}_{ik}^{(j,j')} = -\tilde{\boldsymbol{U}}_{ij'}\tilde{\boldsymbol{U}}_{j'j'}^{-1}\tilde{\boldsymbol{U}}_{j'k}, \quad \text{for neighbors } i \geq j, k \geq j \text{ of element } j'$$

- Choose the j' that minimizes the norm of the discarded fill

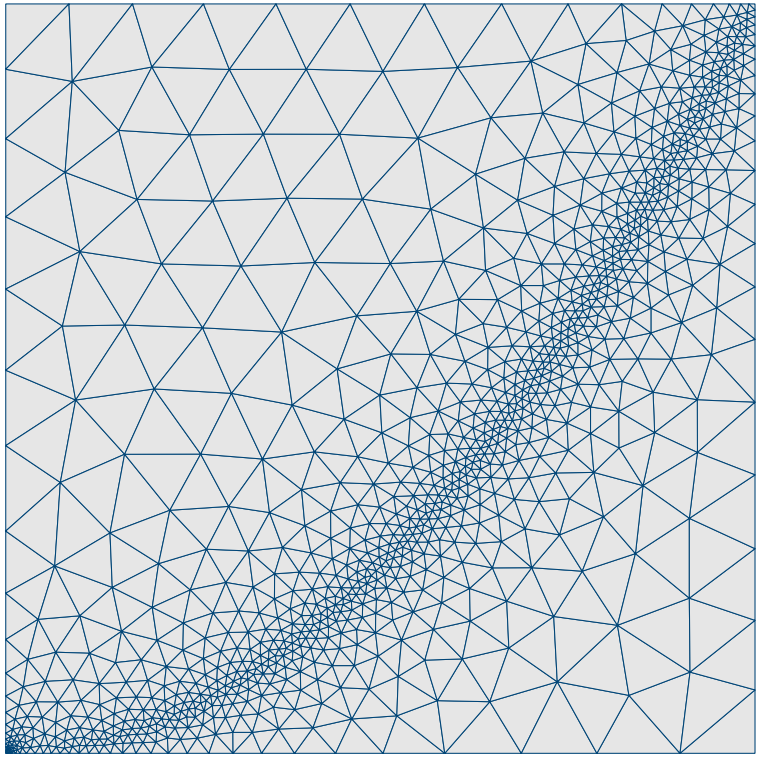
$$w^{(j,j')} = \|\Delta\tilde{\boldsymbol{U}}^{(j,j')}\|_{\mathrm{F}}$$

- Some simplifications, min-heap data structure $\Longrightarrow \mathcal{O}(n \log n)$ cost
- Increased locality: Consider only neighbors for j'

Effect of Ordering on Convection-Diffusion

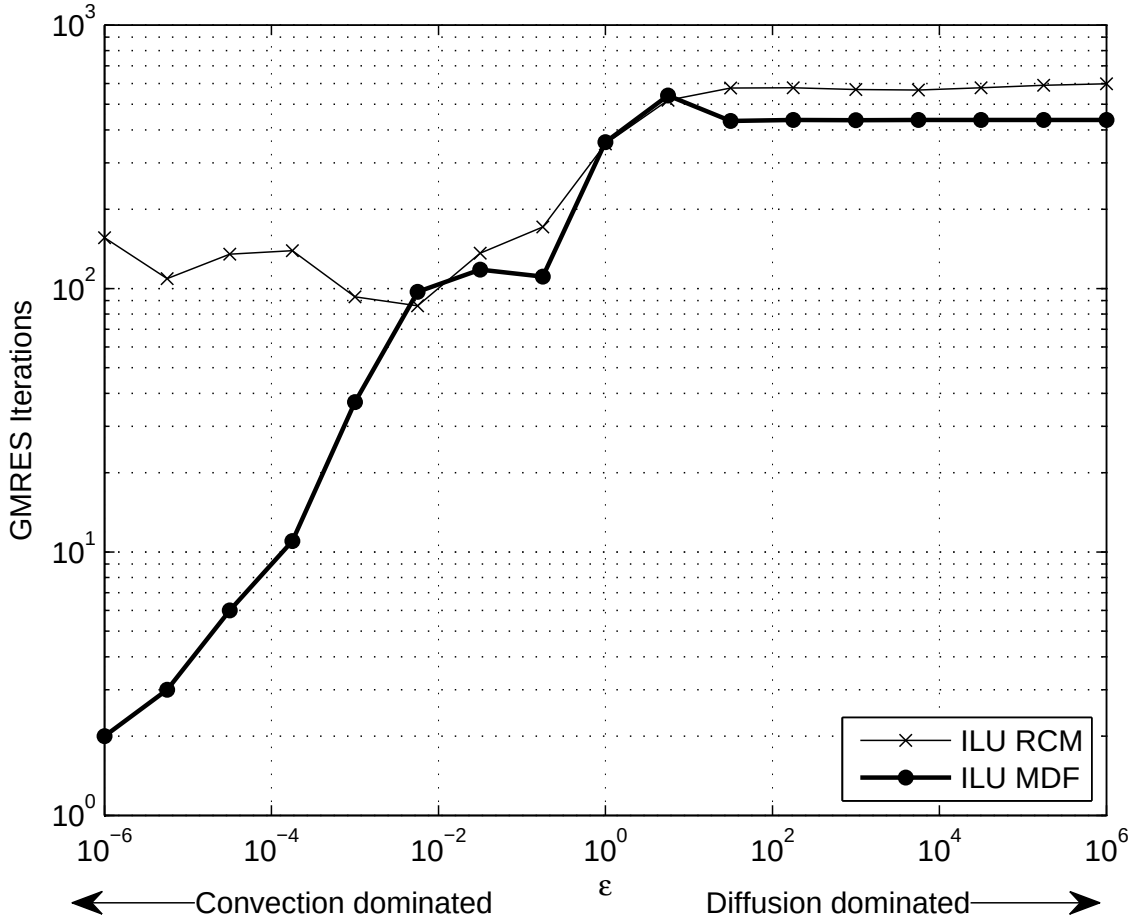
- Convection-Diffusion model problem, with $(\alpha, \beta) = (1, 2x)$, $\varepsilon \geq 0$:

$$\frac{\partial u}{\partial t} + \nabla \cdot \begin{bmatrix} \alpha u \\ \beta u \end{bmatrix} - \nabla \cdot \begin{bmatrix} \varepsilon u_x \\ \varepsilon u_y \end{bmatrix} = 0$$



Effect of Ordering on Convection-Diffusion

- Reverse Cuthill-McKee vs. Minimum Discarded Fill ordering
- MDF makes ILU perfect for convection
- Ordering less important for diffusion, ILU remains poor



The ILU(0)-p1 Preconditioner

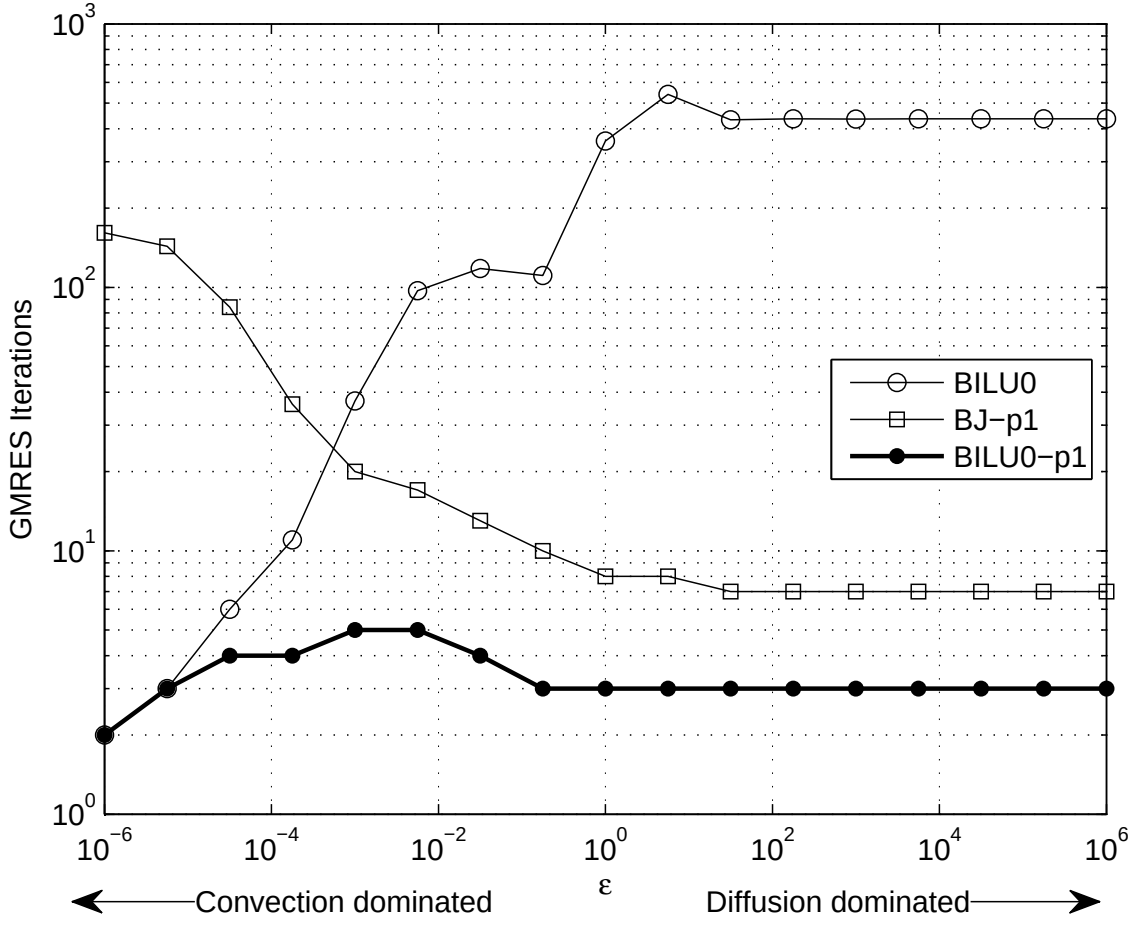
- Combination of block ILU and multigrid [Persson/Peraire 06,08]
- Coarse scale correction + post-smoothing by $\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}}$:

0. $\boldsymbol{A}^{(0)} = \boldsymbol{P}^T\boldsymbol{A}\boldsymbol{P}$	*Precompute coarse operator, block wise*
1. $\boldsymbol{b}^{(0)} = \boldsymbol{P}^T\boldsymbol{b}$	*Restrict residual element/component wise*
2. $\boldsymbol{A}^{(0)}\boldsymbol{u}^{(0)} = \boldsymbol{b}^{(0)}$	*Solve coarse scale problem*
3. $\boldsymbol{u} = \boldsymbol{P}\boldsymbol{u}^{(0)}$	*Prolongate solution element/component wise*
4. $\boldsymbol{u} = \boldsymbol{u} + \alpha\tilde{\boldsymbol{A}}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{u})$	*Apply smoother $\tilde{\boldsymbol{A}}$ with damping α*

- Restriction/prolongation operator $\boldsymbol{P}$ block diagonal, based on orthogonal Koornwinder polynomials
- Coarse scale problem solved directly (2-D problems, serial) or iteratively by GMRES or h-multigrid

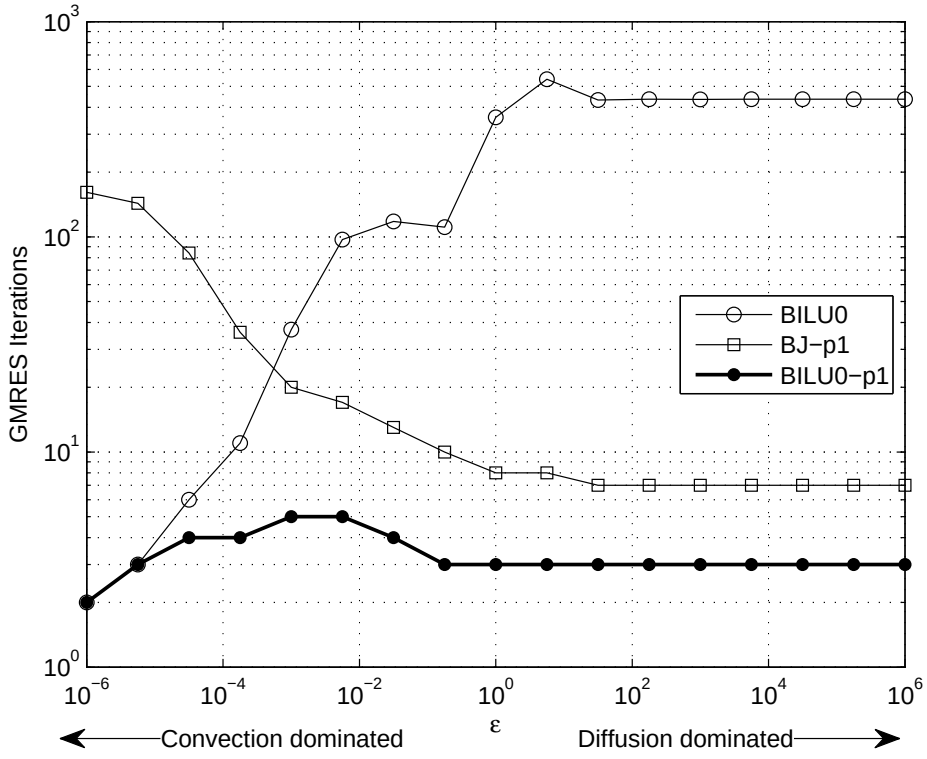
The ILU-p1 Preconditioner - Convection-Diffusion

- Block ILU perfect for convection, multigrid perfect for diffusion
- Block ILU-smoothed multigrid (BILU0-p1) almost perfect for any ε



Effect of Ordering on ILU

- MDF ordering makes block-ILU0 with coarse grid correction almost perfect for convection-diffusion
- Good element ordering critical for Navier-Stokes as well:



Problem	Element Ordering		
	Random	RCM	MDF
Inviscid	51	27	12
Laminar, Re=1,000	200	135	12
Laminar, Re=20,000	197	139	27
RANS, Re=10^6	98	99	18

Convergence – Model Navier-Stokes Problem

$\times$ = No convergence after 1,000 iterations, from [Persson/Peraire '08]

Problem	Parameters		Preconditioner/Iterations					
			Block Jacobi		Block G-S		Block ILU0	
	Δt	M	BJ	BJ-p1	BGS	BGS-p1	BILU0	BILU0-p1
Inviscid	10^{-3}	0.2	24	19	14	11	5	4
	10^{-1}	0.2	187	85	73	49	12	6
	∞	0.2	840	142	456	72	40	9
	10^{-3}	0.01	200	112	111	67	15	6
	10^{-1}	0.01	$\times$	$\times$	$\times$	532	94	10
	∞	0.01	$\times$	$\times$	$\times$	$\times$	374	16
Laminar	10^{-3}	0.2	50	34	25	19	4	4
Re=1,000	10^{-1}	0.2	$\times$	597	477	225	11	5
	∞	0.2	$\times$	$\times$	$\times$	$\times$	37	7
	10^{-3}	0.01	98	66	51	33	8	5
	10^{-1}	0.01	$\times$	619	$\times$	207	27	9
	∞	0.01	$\times$	$\times$	$\times$	748	135	12

Convergence – Model Navier-Stokes Problem

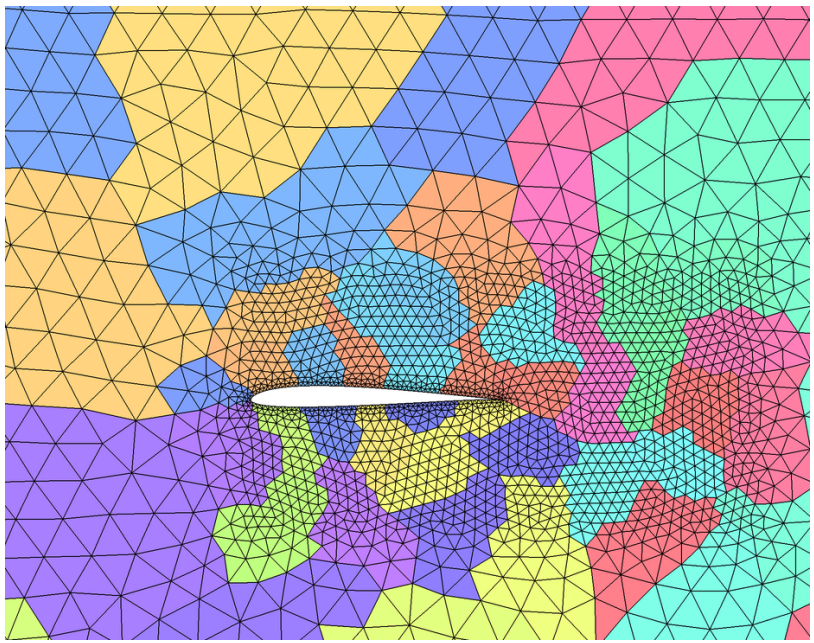
$\times$ = No convergence after 1,000 iterations, from [Persson/Peraire '08]

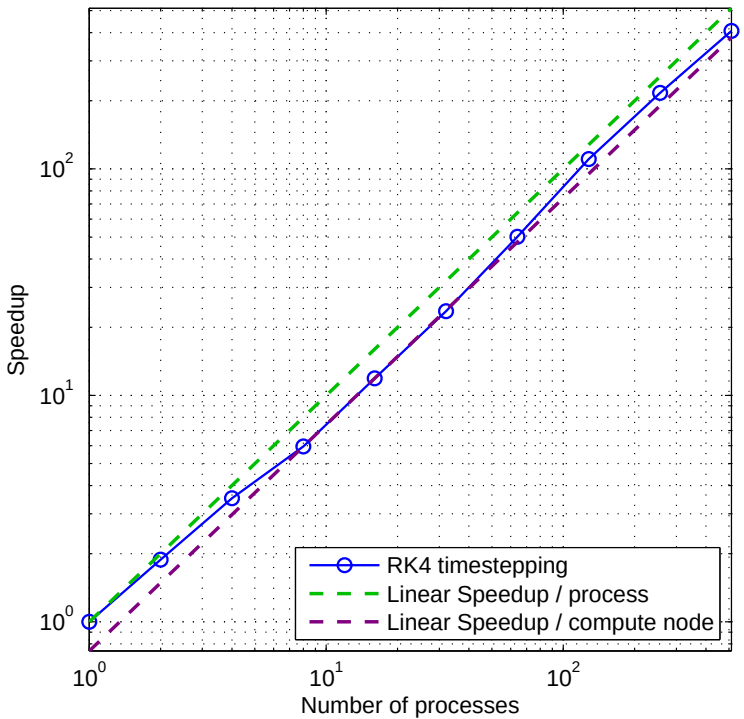
Problem	Parameters		Preconditioner/Iterations					
			Block Jacobi		Block G-S		Block ILU0	
	Δt	M	BJ	BJ-p1	BGS	BGS-p1	BILU0	BILU0-p1
Laminar	10^{-3}	0.2	26	19	14	11	4	4
Re=20,000	10^{-1}	0.2	456	220	219	113	16	8
	∞	0.2	$\times$	$\times$	$\times$	$\times$	236	20
	10^{-3}	0.01	160	90	61	38	12	6
	10^{-1}	0.01	$\times$	$\times$	$\times$	735	80	16
	∞	0.01	$\times$	$\times$	$\times$	$\times$	$\times$	35
RANS	10^{-3}	0.2	76	56	33	28	8	7
Re=10^6	10^{-1}	0.2	$\times$	$\times$	$\times$	$\times$	35	25
	∞	0.2	$\times$	$\times$	$\times$	$\times$	70	18
	10^{-3}	0.01	411	231	174	110	14	9
	10^{-1}	0.01	$\times$	$\times$	$\times$	$\times$	46	16
	∞	0.01	$\times$	$\times$	$\times$	$\times$	132	28

Outline

Parallelization of Residual Evaluation

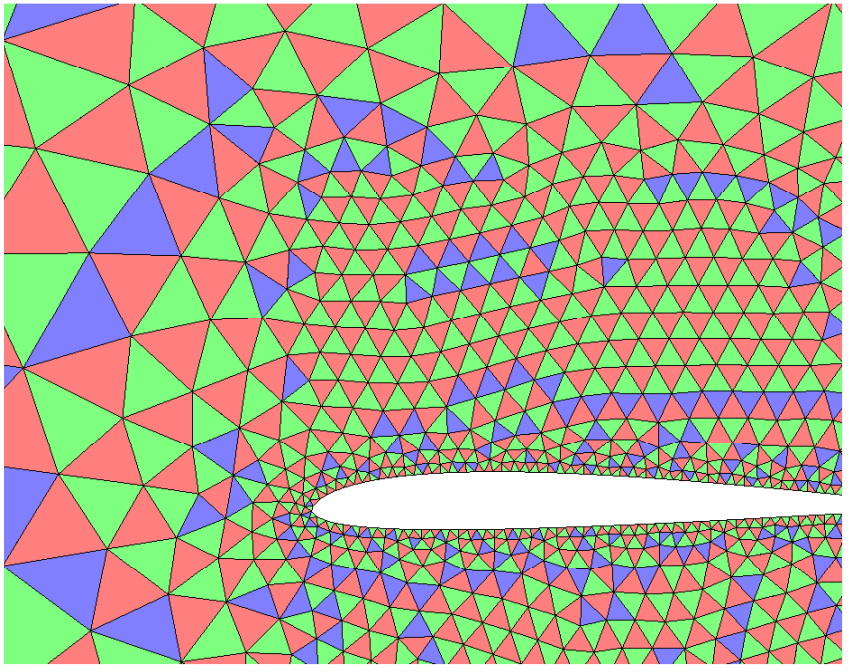
- Two critical properties for parallelization:
  - High computational cost per degree-of-freedom
  - Memory accessed in localized chunks
- Essentially perfect speed-up of residual calculation in MPI
- Communication not an issue with overlapping computations
- However, memory bandwidth limitations for 8 cores on single node





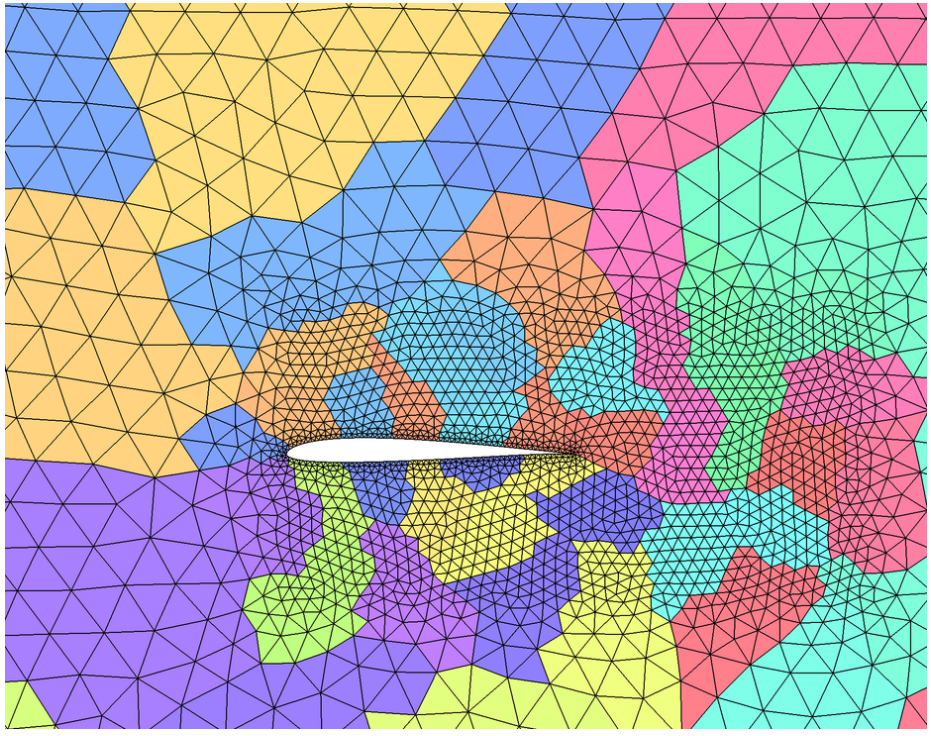
ILU Parallelization – Coloring

- Serial nature of Gaussian elimination – Hard to parallelize ILU(0)
- Standard approach: *Color* the graph to find independent sets
- Elements of the same color can be updated independently
- But this is typically a poor ILU-ordering for convective problems!
- The coloring eliminates the sequential dependencies found by the MDF ordering



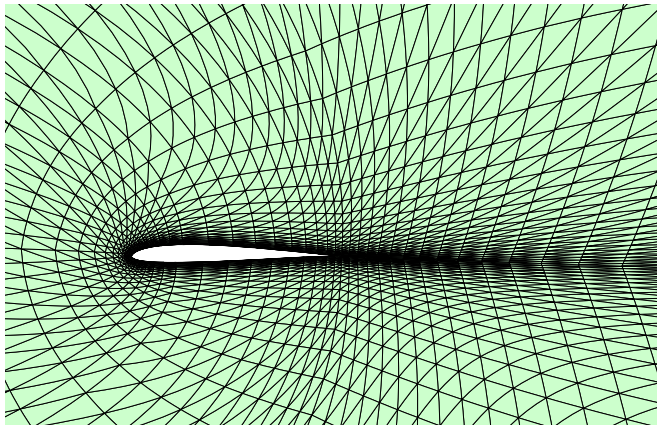
ILU Parallelization – Domain Decomposition

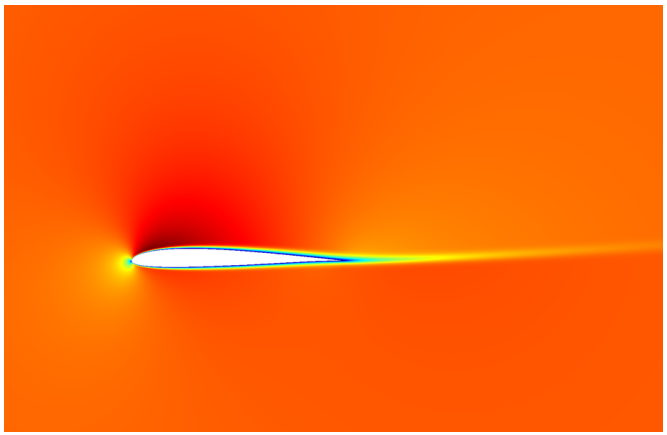
- Better approach: Partition-wise ILUs with MDF ordering
- Partition using the weights $C_{ij} = \|\boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{ij}\|_F$
- Essentially a “non-overlapping Schwartz preconditioner with incomplete solutions”
- Approaches Jacobi as # partitions $\rightarrow$ # elements
- Good option for many problems – GMRES iterations cheap compared to matrix creation

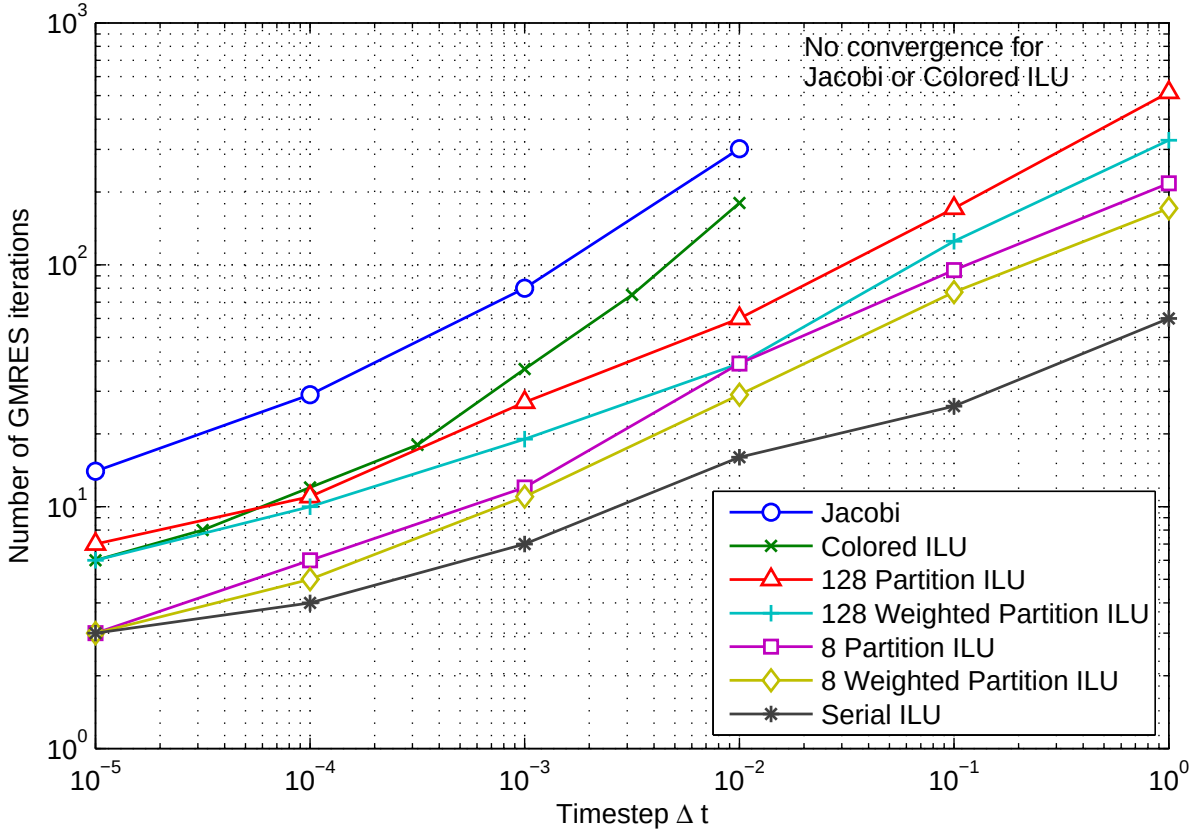


Iterative Solver Convergence Properties

- SD7003 test problem: $\mathrm{Re} = 8000$, $M = 0.1$, CFL $\Delta t < 10^{-6}$
- Coloring essentially destroys the excellent convergence of MDF
- Partition-wise ILUs scale better, only 2-3 times slower for large Δt

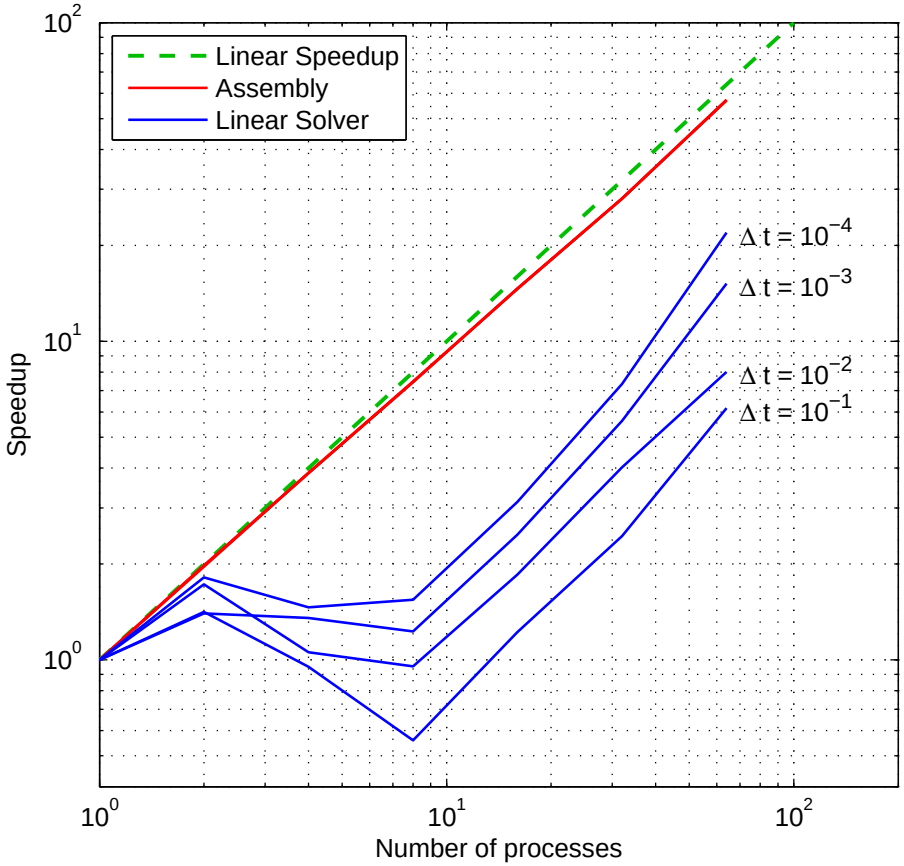


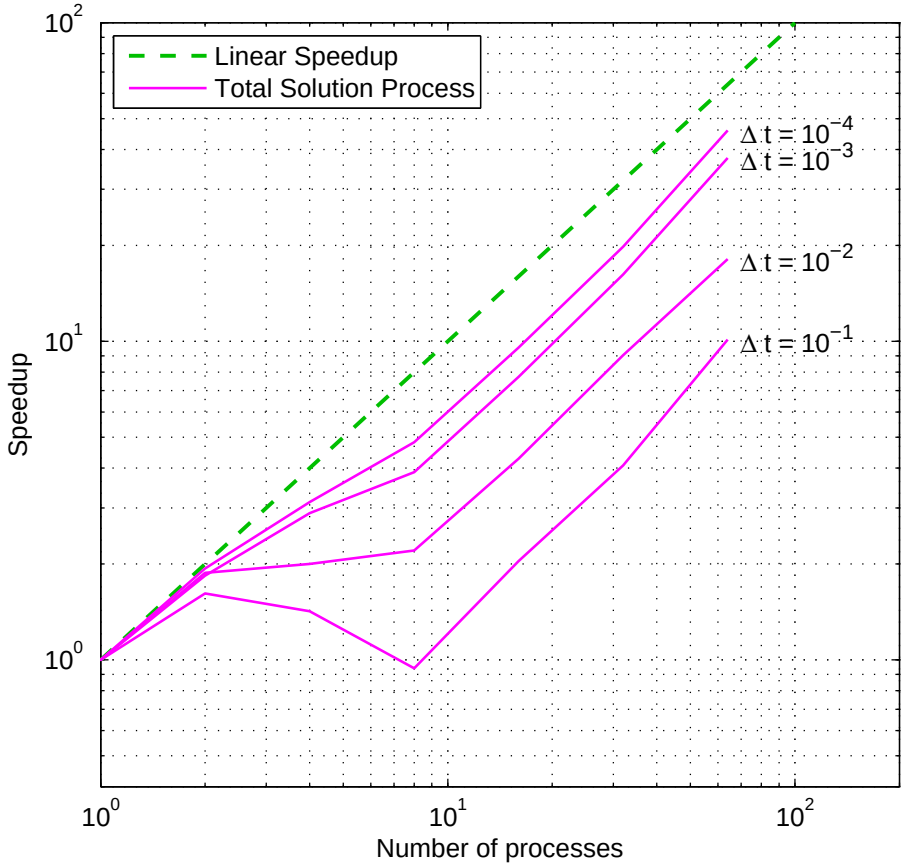




SD7003 Test Problem – Timings and Speedup

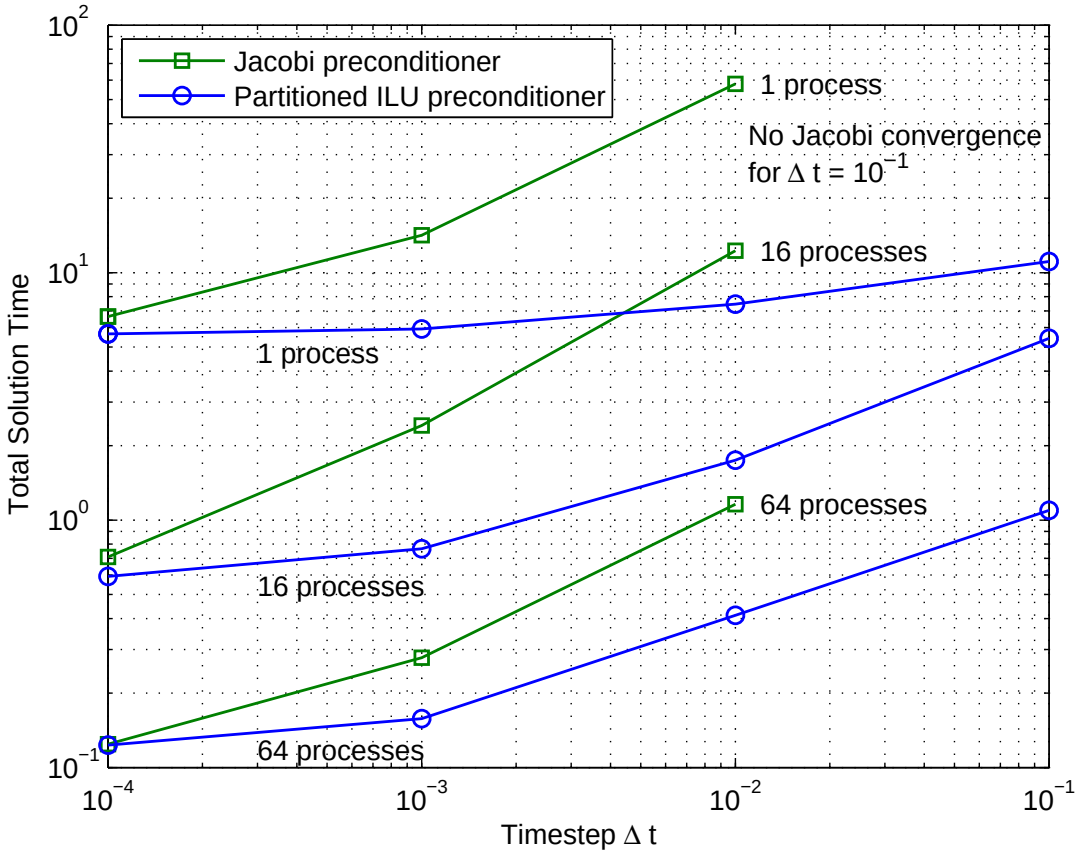
- Assembly scales almost perfectly, as expected
- Linear solver scales poorly up to the 8 cores in each compute node, due to increased iterations and limited memory bandwidth
- Beyond the 8 cores, the scaling per compute node is excellent





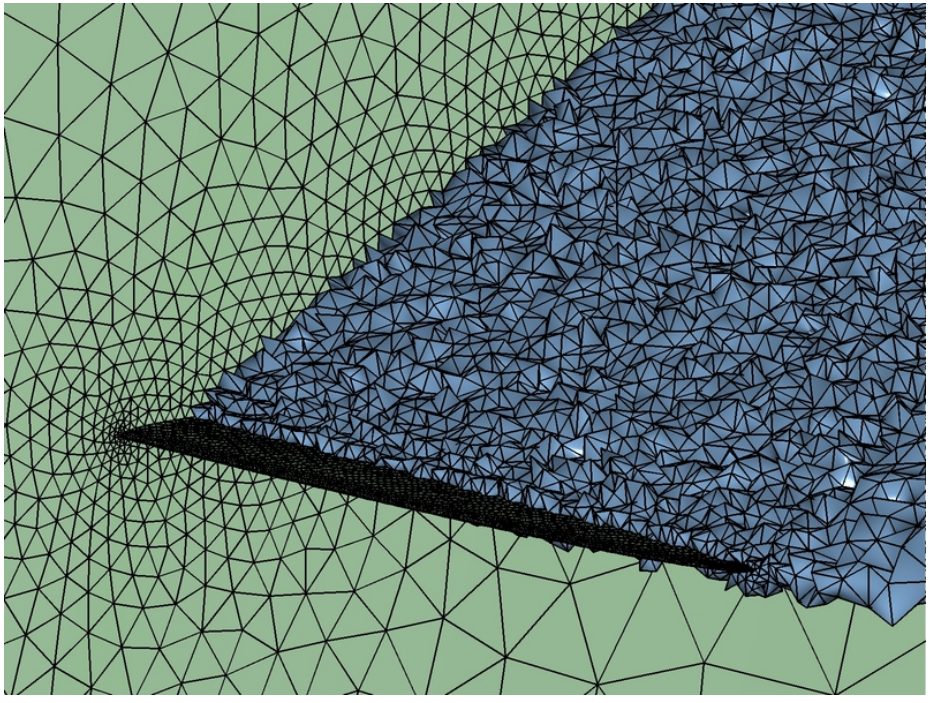
SD7003 Test Problem – Jacobi Comparison

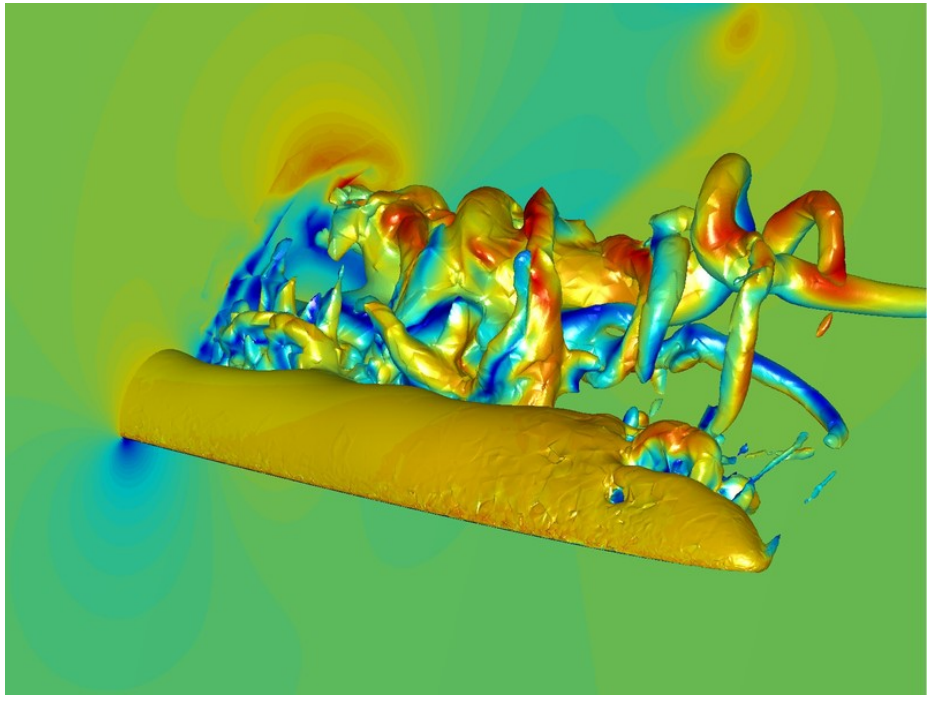
- Partition-wise block-ILU outperforms block-Jacobi even at moderate timesteps Δt
- Jacobi's good scalability does not compensate for its poor preconditioning properties



Elliptic Wing Problem

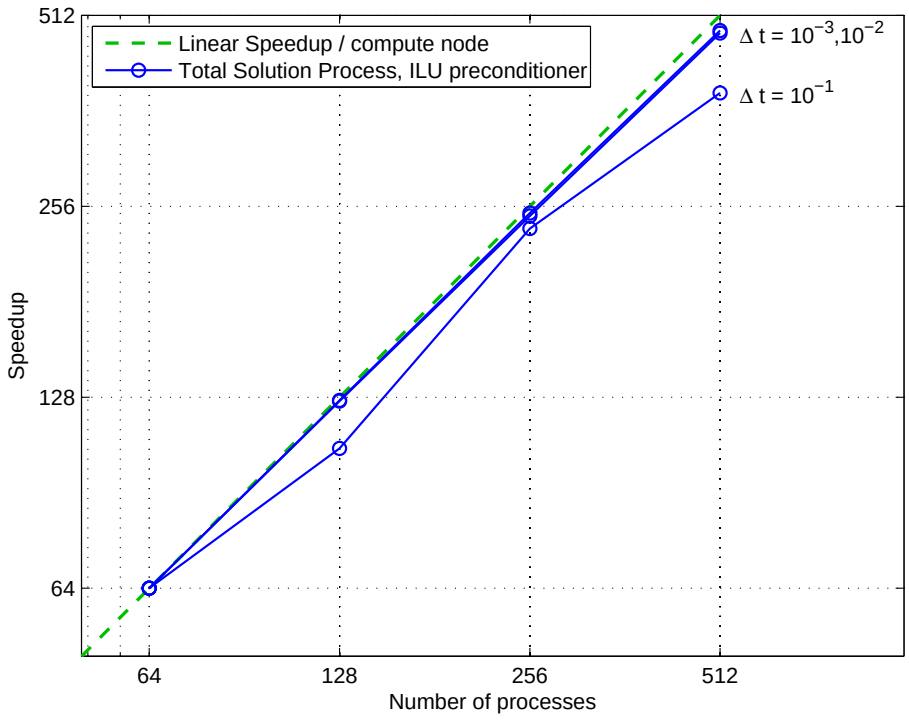
- Elliptic wing, $\mathrm{Re} = 2000$, $M = 0.3$, $\mathrm{AoA} = 30°$
- 190,000 tetrahedral elements, $p = 3$, about 19 million DOFs
- Time integration by 2-stage, 3rd order accurate DIRK scheme
- Jacobian storage 36GB, no comparison with serial possible

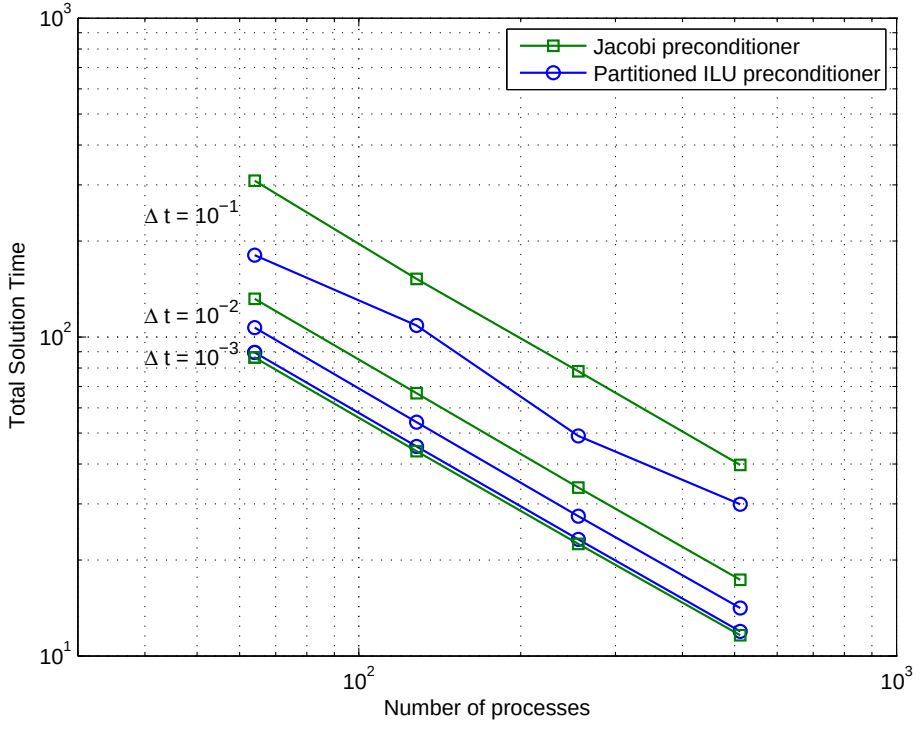




The Elliptic Wing Problem – Timings and Speedup

- Almost perfect speedup per compute node with 512 processes
- Solution time dominated by assembly for small Δt





Outline

Implicit-Explicit Runge-Kutta Methods

- [Persson 2011]: Combine implicit and explicit time stepping
- Based on a splitting $\frac{du}{dt} = f(u) + g(u)$ where $f(u)$ is considered nonstiff terms and $g(u)$ stiff terms
- Two Runge-Kutta schemes
  1. Diagonally Implicit Runge-Kutta (DIRK) scheme c, A, b for $g(u)$
  2. Explicit Runge-Kutta (ERK) scheme $\hat{c}, \hat{A}, \hat{b}$ for $f(u)$
- Combined scheme alternates explicit/implicit:

$\hat{k}_1 = f(u_n)$

for $i = 1$ **to** s

Solve for k_i in $k_i = g(u_{n,i})$, where $u_{n,i} = u_n + \Delta t \sum_{j=1}^{i} a_{i,j} k_j + \Delta t \sum_{j=1}^{i} \hat{a}_{i+1,j} \hat{k}_j$

Evaluate $\hat{k}_{i+1} = f(u_{n,i})$

end for

$u_{n+1} = u_n + \Delta t \sum_{i=1}^{s} b_j k_j + \Delta t \sum_{i=1}^{s+1} \hat{b}_j \hat{k}_j$

IMEX Schemes

IMEX1: 2-stage, 2nd order DIRK + 3-stage, 2nd order ERK

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cc} \alpha & \alpha & 0 \\ 1 & 1-\alpha & \alpha \\ \hline & 1-\alpha & \alpha \end{array} \qquad \begin{array}{c|c} \hat{c} & \hat{A} \\ \hline & \hat{b}^T \end{array} = \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \alpha & \alpha & 0 & 0 \\ 1 & \delta & 1-\delta & 0 \\ \hline & 0 & 1-\alpha & \alpha \end{array}$$

where $\alpha = 1 - \frac{\sqrt{2}}{2}$, $\delta = -2\sqrt{2}/3$. 2nd order, L-stable.

IMEX2: 2-stage, 3rd order DIRK + 3-stage, 3rd order ERK

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cc} \alpha & \alpha & 0 \\ 1-\alpha & 1-2\alpha & \alpha \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{array}{c|c} \hat{c} & \hat{A} \\ \hline & \hat{b}^T \end{array} = \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \alpha & \alpha & 0 & 0 \\ 1-\alpha & \alpha-1 & 2(1-\alpha) & 0 \\ \hline & 0 & \frac{1}{2} & \frac{1}{2} \end{array}$$

where $\alpha = (3 + \sqrt{3})/6$. 3rd order accurate, no L-stability.

IMEX Schemes

IMEX3: 3-stage, 3rd order DIRK + 4-stage, 3rd order ERK

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|ccc} 0.43586652 & 0.43586652 & 0 & 0 \\ 0.71793326 & 0.28206673 & 0.43586652 & 0 \\ 1 & 1.2084966 & -0.64436317 & 0.43586652 \\ \hline & 1.2084966 & -0.64436317 & 0.43586652 \end{array}$$

$$\begin{array}{c|c} \hat{c} & \hat{A} \\ \hline & \hat{b}^T \end{array} = \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 0.43586652 & 0.43586652 & 0 & 0 & 0 \\ 0.71793326 & 0.32127888 & 0.39665437 & 0 & 0 \\ 1 & -0.10585829 & 0.55292914 & 0.55292914 & 0 \\ \hline & 0 & 1.20849664 & -0.64436317 & 0.43586652 \end{array}$$

3rd order accurate, L-stable.

Mesh-Size Based Splitting of Residual

- Associate each component of the solution vector U with an equation in the residual $R(U)$
- Split based on stiff components U_{im} in mesh elements smaller than a given size, and the remaining nonstiff components U_{ex}:

$$R(u) = \begin{bmatrix} R_{\text{im}}(U) \\ R_{\text{ex}}(U) \end{bmatrix} = \begin{bmatrix} 0 \\ R_{\text{ex}}(U) \end{bmatrix} + \begin{bmatrix} R_{\text{im}}(U) \\ 0 \end{bmatrix} = f(U) + g(U)$$

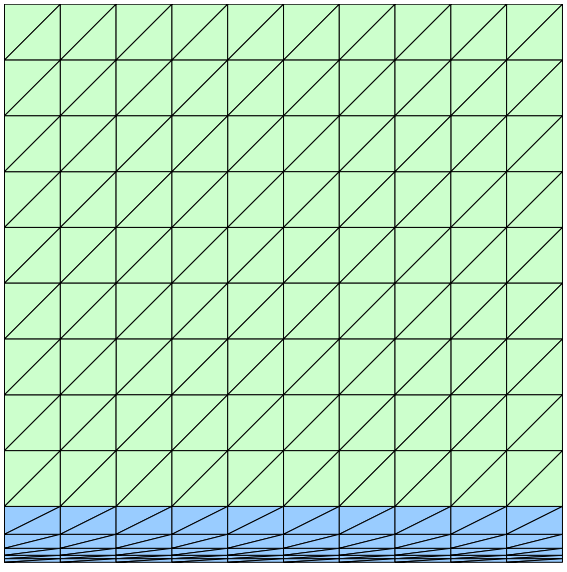
- Assumption: The CFL condition of the explicit scheme will not be affected by the small elements

Preconditioned Quasi-Newton-Krylov Methods

- Need to solve nonlinear systems of equations $k_i = g(u_{n,i})$
- Use quasi-Newton method with Jacobian $J = M - \alpha \Delta t dR/dU$
- Re-use J and its factorizations between iterations and between timesteps, as long as convergence acceptable
- Linear systems solved by preconditioned Newton-Krylov, with ILU-preconditioned CGS solvers and MDF element ordering

Absolute Stability – Flow Over a Flat Plate

- Simple flat plate model problem to evaluate IMEX approach
- Unit square, free-stream left/top, wall bottom, outflow right
- Mach number 0.2, Reynolds number 10,000
- Meshes created by n_{ref} anisotropic refinement of 10-by-10 Cartesian grid, with smallest element height $h_{\text{min}} = 0.1/2^{n_{\text{ref}}}$



Implicit/Explicit Mesh

Density

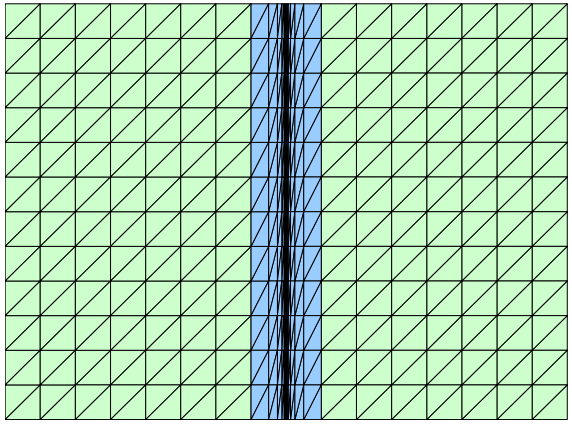
Absolute Stability – Flow Over a Flat Plate

- Determine largest stable timesteps $\Delta t_{\max}$ of ERK method for entire problem, with $\Delta t^0_{\max}$ on the initial mesh
- Confirm that the IMEX schemes are stable with timestep $\Delta t^0_{\max}$ regardless of the number of refinements
- Ratio close to 4 between successive $\Delta t_{\max}$ indicate CFL condition determined by viscous terms

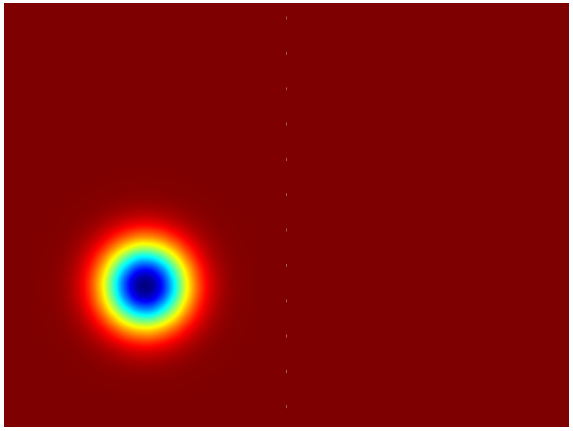
Scheme	ERK1		IMEX1	ERK2		IMEX2	ERK3		IMEX3
$\Delta t^0_{\max}$	$3.26 \cdot 10^{-4}$			$3.35 \cdot 10^{-4}$			$2.61 \cdot 10^{-4}$		
n_{ref}	$\frac{\Delta t_{\max}}{\Delta t^0_{\max}}$	Ratio		$\frac{\Delta t_{\max}}{\Delta t^0_{\max}}$	Ratio		$\frac{\Delta t_{\max}}{\Delta t^0_{\max}}$	Ratio	
0	1.0000		Stable	1.0000		Stable	1.0000		Stable
1	0.6612	1.51	Stable	0.6671	1.50	Stable	0.4958	2.02	Stable
2	0.1747	3.79	Stable	0.1762	3.79	Stable	0.1298	3.82	Stable
3	0.0457	3.82	Stable	0.0461	3.82	Stable	0.0337	3.85	Stable
4	0.0118	3.89	Stable	0.0119	3.89	Stable	0.0086	3.92	Stable
5	0.0032	3.62	Stable	0.0033	3.62	Stable	0.0023	3.72	Stable

Order of Accuracy – Euler vortex problem

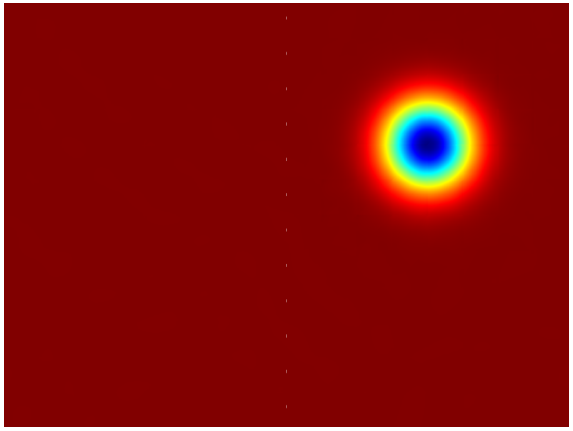
- Validate accuracy of IMEX scheme on unsteady model vortex problem
- Rectangular domain, periodic boundary conditions
- Mesh obtained by anisotropic refinement through center of the domain, a total of $n_{\text{ref}} = 5$ times



Implicit/Explicit Mesh



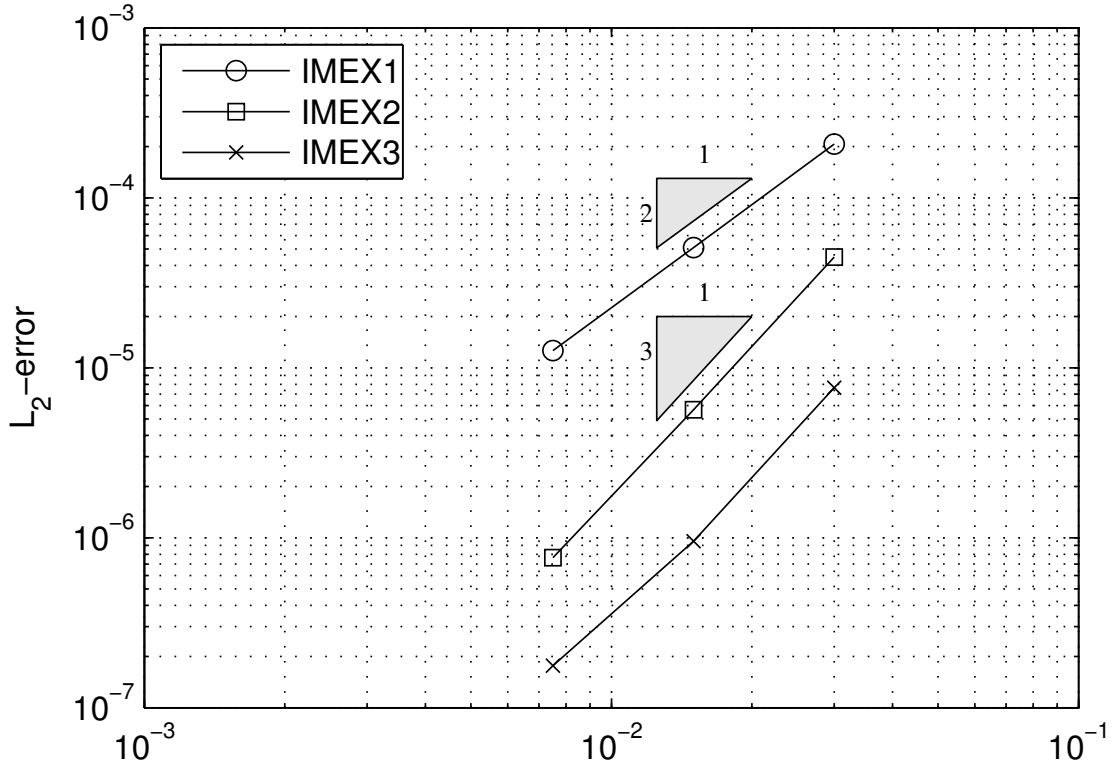
Initial density



Final density

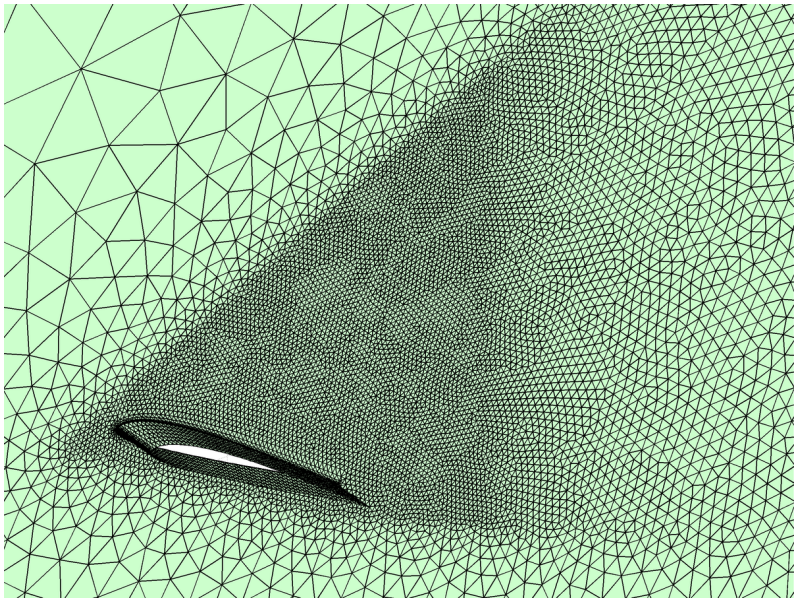
Order of Accuracy – Euler vortex problem

- $\Delta t^0_{\max}$ is again the explicit timestep limit for the unrefined mesh
- Compute L_2-errors for IMEX solutions for three timesteps $\Delta t^0_{\max}$, $\Delta t^0_{\max}/2$, and $\Delta t^0_{\max}/4$
- Expected orders of 2, 3, and 3 obtained, which is not entirely trivial since implicit part is far from the convergent regime



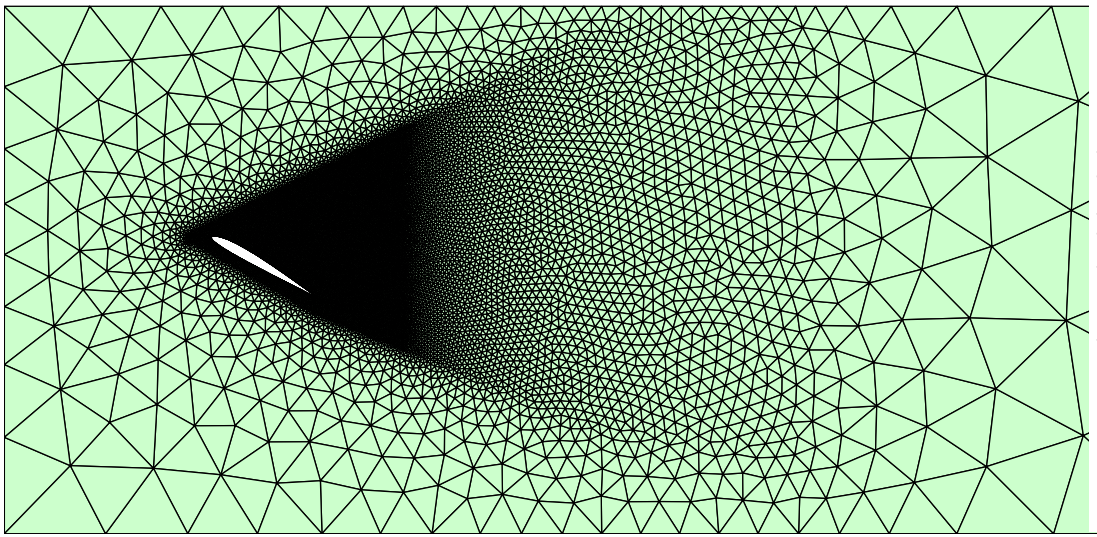
Large Eddy Simulation of Flow over Airfoil

- Consider the flow over an SD7003 airfoil at Reynolds number 100,000 and 30° angle of attack
- LES-type hybrid mesh, with DistMesh for unstructured triangles [Persson '04], each extruded as 6×3 prisms elements spanwise
- 312,000 elements, 31M DOFs, elements curved to align with boundaries using nonlinear elasticity approach [Persson '09].

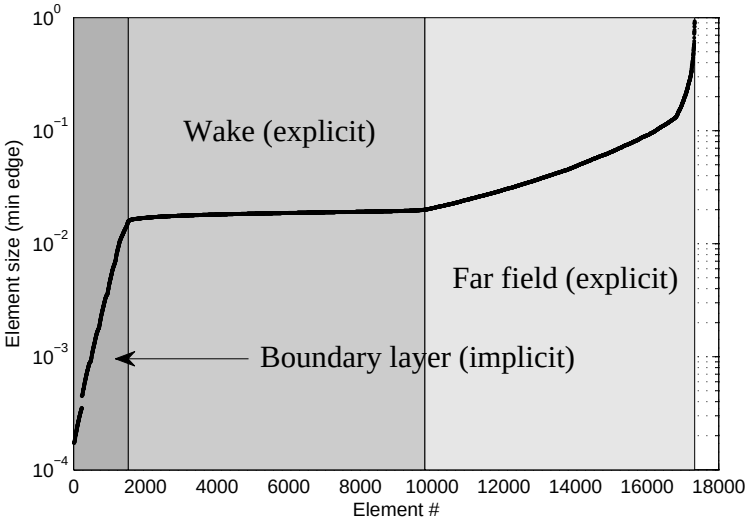


Large Eddy Simulation of Flow over Airfoil

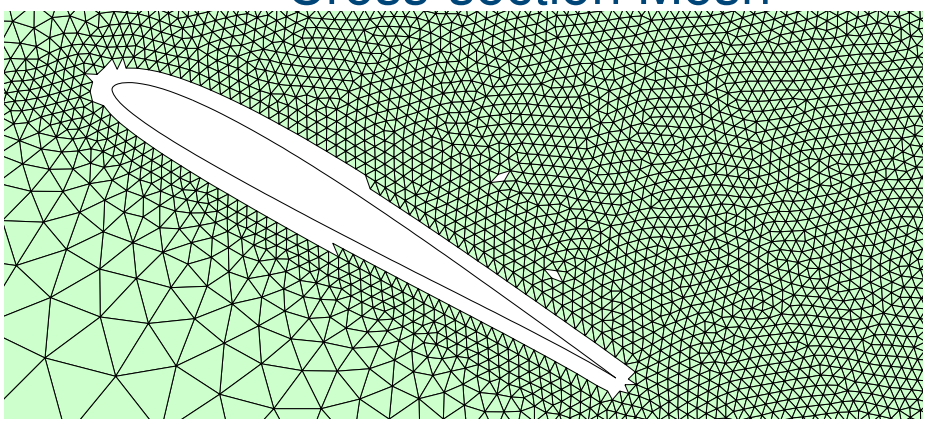
- Split based on smallest element edge sizes



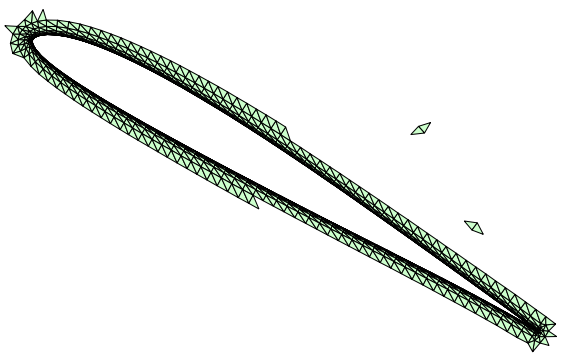
Cross-section Mesh



Element size distribution



Explicit elements



Implicit elements

Large Eddy Simulation of Flow over Airfoil

- Solve the flow problem using the **IMEX1** scheme
- ERK1 on explicit portion only: maximum $\Delta t = 2.0 \cdot 10^{-4}$
- ERK1 on entire mesh: maximum $\Delta t = 2.2 \cdot 10^{-8}$
- IMEX1 on entire mesh with splitting: Stable with $\Delta t = 2.0 \cdot 10^{-4}$
- Improvement of a factor 10,000, at the cost of solving non-linear equations involving 9% of the unknowns

Large Eddy Simulation of Flow over Airfoil

Visualization by Mach number on an isosurface of the entropy