Algorithms for Nonlinear and Transient Problems

David E. Keyes

Department of Applied Physics & Applied Mathematics
Columbia University

Institute for Scientific Computing Research
Lawrence Livermore National Laboratory

Recall Newton methods

- Given $F(u) = 0,\ F : \Re^n \to \Re^n$ and iterate u^0 we wish to pick u^{k+1} such that

$$F(u^{k+1}) \approx F(u^k) + F'(u^k)du^k = 0$$

where $du^k = u^{k+1} - u^k,\ k = 0, 1, 2, \ldots$

- Neglecting higher-order terms, we get

$$du^k = -[J(u^k)]^{-1} F(u^k)$$

where $J = F'(u^k)$ is the Jacobian matrix, generally large, sparse, and ill-conditioned for PDEs

- In practice, require $\| F(u^k) + J(u^k)du^k \| < e$
- In practice, set $u^{k+1} = u^k + l\, du^k$ where l is selected to minimize $\| F(u^k + l\, du^k) \|$

Newton's method: pros and cons

- Locally quadratically convergent (if Jacobian is nonsingular at the solution)
  - number of significant digits doubles asymptotically at each step
  - not globally convergent from arbitrary initial iterate
- Requires Jacobian evaluation at each iteration
  - may be nontrivial for user to supply derivatives
  - may require large fraction of code size and execution time
  - if exact derivative information is sacrificed, so if quadratic convergence
- Requires solution of linear system with Jacobian at each iteration
  - bottleneck when ill-conditioned

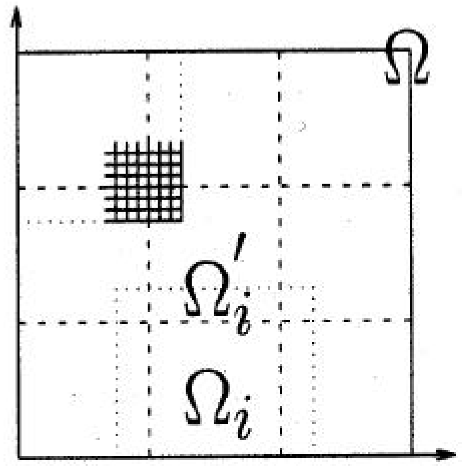
Recall Krylov methods

- **Given $Ax = b, A \in \Re^{n\times n}$ and iterate x^0, we wish to generate a basis $V = \{v_1, v_2, \ldots, v_k\} \in \Re^{n\times k}$ for x ($x \approx Vy$) and a set of coefficients $\{y_1, y_2, \ldots, y_k\}$ such that x^k is a best fit in the sense that $y \in \Re^k$ minimizes $\| AVy - b \|$**
- **Krylov methods define a complementary basis**

  $W = \{w_1, w_2, \ldots, w_k\} \in \Re^{n\times k}$ so that $W^T(AVy - b) = 0$ may be solved for *y*
- **In practice $\boldsymbol{k << n}$ and the bases are grown from seed vector $r^0 = Ax^0 - b$ via recursive multiplication by A and Gram-Schmidt**
- **Does not require inverse of A**

Recall Schwarz preconditioning

- Given $A x = b$, partition x into subvectors, corresp. to subdomains Ω_i of the domain Ω of the PDE, nonempty, possibly overlapping, whose union is all of the elements of $x \in \Re^n$

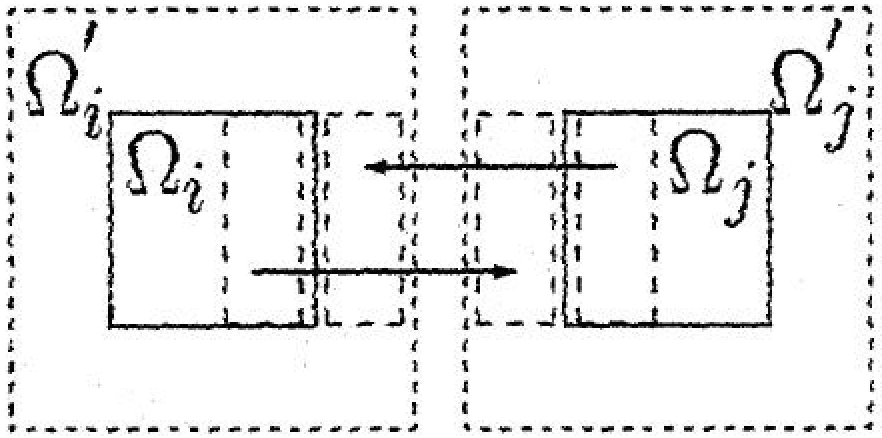


- Let Boolean rectangular matrix R_i extract the i^{th} subset of x :

$$x_i = R_i x$$

- Let $A_i = R_i A R_i^T$

$$B^{-1} = \sum_i R_i^T A_i^{-1} R_i$$



The Boolean matrices are gather/scatter operators, mapping between a global vector and its subdomain support

Newton-Krylov-Schwarz

Popularized in parallel Jacobian-free form under this name by Cai, Gropp, Keyes & Tidriri (1994)

Newton

nonlinear solver

asymptotically quadratic

Krylov

accelerator

spectrally adaptive

Schwarz

preconditioner

parallelizable

Jacobian-Free Newton-Krylov Method

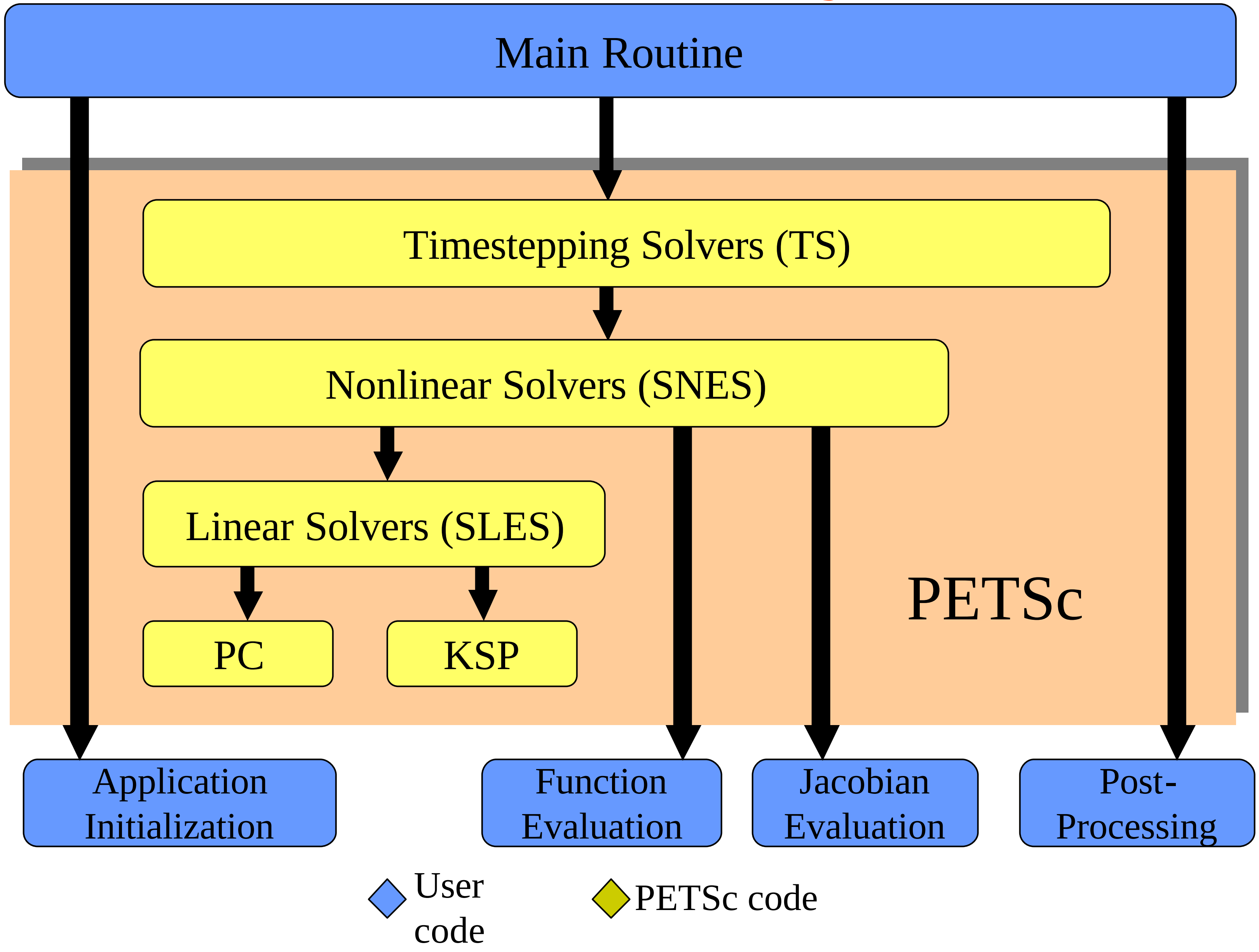
- In the Jacobian-Free Newton-Krylov (JFNK) method, a Krylov method solves the linear Newton correction equation, requiring Jacobian-vector products
- These are approximated by the Fréchet derivatives

$$J(u)v \approx \frac{1}{e}[F(u+ev)-F(u)]$$

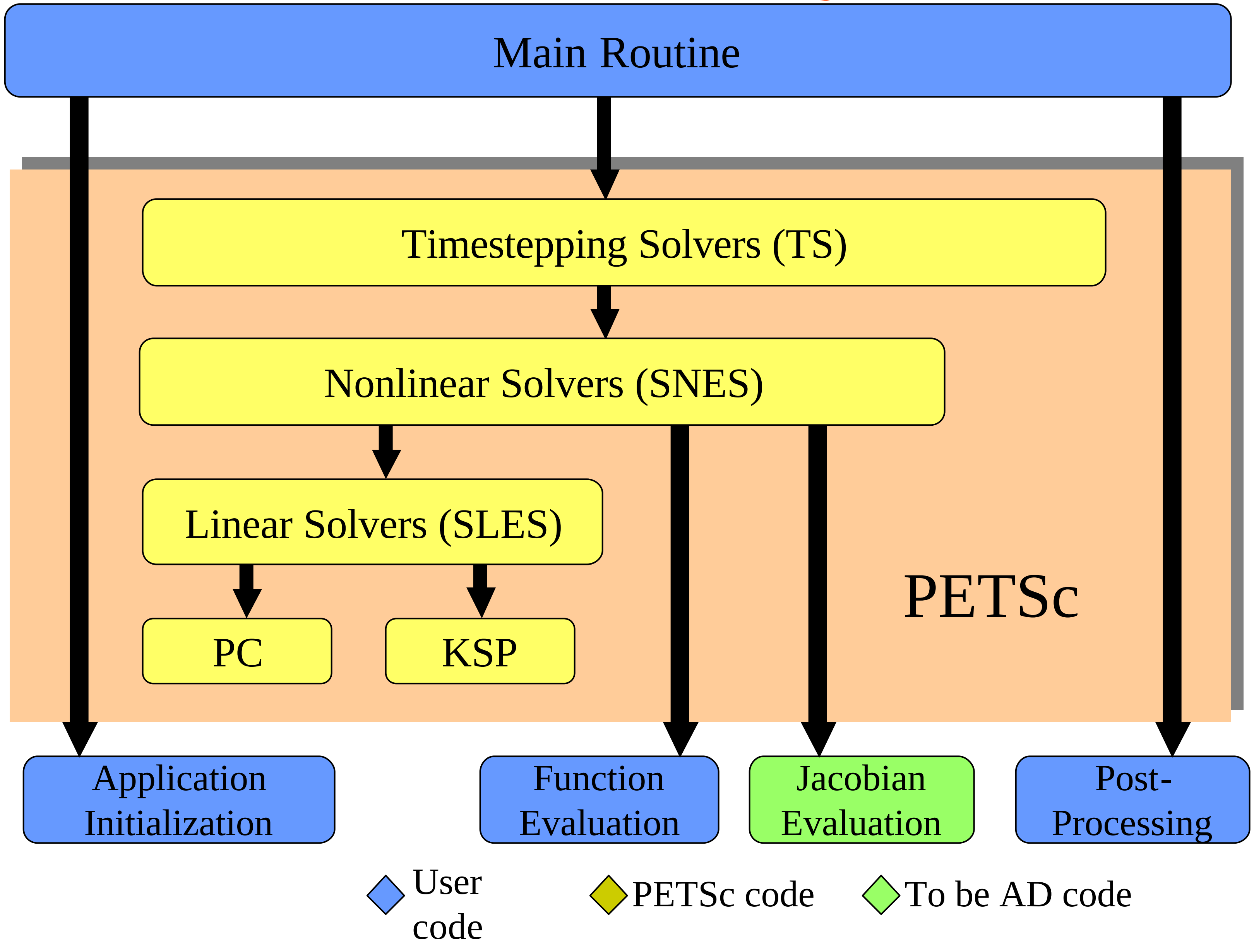
so that the actual Jacobian elements are never explicitly needed, where e is chosen with a fine balance between approximation and floating point rounding error

- Schwarz preconditions, using approximate elements

User Code/PETSc Library Interactions



User Code/PETSc Library Interactions



Philosophy of Jacobian-free NK

- To *evaluate* the linear residual, we use the true $F'(u)$, giving a true Newton step and asymptotic quadratic Newton convergence
- To *precondition* the linear residual, we do anything convenient that uses understanding of the dominant physics/mathematics in the system and respects the limitations of the parallel computer architecture and the cost of various operations:
  - combinations of operator-split Jacobians (for reasons of physics or reasons of numerics)
  - Jacobian of related discretization (for "fast" solves)
  - Jacobian of lower-order discretization (for more stability, less storage)
  - Jacobian with "lagged" values for expensive terms (for less computation per degree of freedom)
  - Jacobian stored in lower precision (for less memory traffic per preconditioning step)
  - Jacobian blocks decomposed for parallelism

Philosophy of Jacobian-free NK, cont.

- These motivations are not new; most large-scale application codes *also* take "short cuts" on the approximate Jacobian operator to be inverted – showing physical intuition
- The problem with many codes is that they do not anywhere have an accurate global Jacobian operator; they use *only* the weak Jacobian
- This leads to a weakly nonlinearly converging "defect correction method"
  - **Defect correction:**

$$B\, du^k = -F(u^k)$$

  - **in contrast to preconditioned Newton:**

$$B^{-1} J(u^k)\, du^k = -B^{-1} F(u^k)$$

Jacobian-free NKS

- In the Jacobian-free Newton-Krylov (JFNK) framework, any standard nonlinear solver, which maps a residual into a correction, can be regarded *as a preconditioner*
- The true Jacobian is never formed yet the time-implicit nonlinear residual at each time step can be made as small as needed for nonlinear consistency in long time integrations

Using Jacobian of lower order discretization

- **Orszag popularized the use of linear finite element discretizations as preconditioners for high-order spectral element discretizations in the 1970s; both approach the same continuous operator**
- **It is common in CFD to employ first-order upwinded convective operators as approximate inversions for higher-order operators:**
  - **better factorization stability**
  - **smaller matrix bandwidth and complexity**
- **With Jacobian-free NK, we can have the best of both worlds – a stable factorization/cheap solve *and* a true Jacobian step**

Using Jacobian with lagged terms

- Newton-chord methods (e.g., papers by Smooke et al.) “freeze” the Jacobian matrices:
  - saves Jacobian evaluation and factorization, which can be up to 90% of the running time of the code in some apps
  - however, nonlinear convergence degrades to linear rate
- In Jacobian-free NK, we can “freeze” some or all of the terms in the Jacobian preconditioner, while always accessing the action of the true Jacobian for the Krylov matrix-vector multiply:
  - still saves Jacobian work
  - maintains asymptotically quadratic rate for nonlinear convergence
- See (Knoll-Keyes ’03) for example with coupled edge plasma and Navier-Stokes, showing five-fold improvement over full Newton with constantly refreshed Jacobian on LHS, versus JFNK with preconditioner refreshed once each ten timesteps

Using Jacobian with lower precision elements

- Memory bandwidth is the critical architectural parameter for sparse linear algebra computations
- Storing the preconditioner elements in single precision effectively doubles memory bandwidth (and potentially halves runtime) for this critical phase
- We still form the Jacobian-vector product with full precision and “zero-pad” the preconditioner elements back to full length in the arithmetic unit, so the numerical quality of the Krylov subspace does not degrade

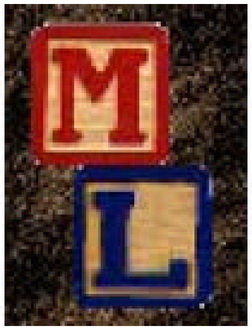
Memory BW bottleneck revealed via precision reduction

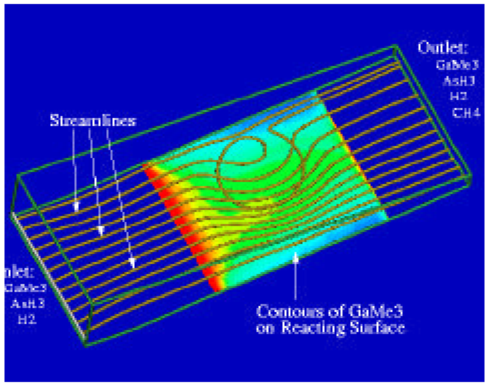
Execution times for unstructured NKS Euler Simulation on Origin 2000: double precision matrices versus single precision preconditioner

Number of Processors	Computational Phase			
	Linear Solve		Overall	
	Double	Single	Double	Single
16	223s	136s	746s	657s
32	117s	67s	373s	331s
64	60s	34s	205s	181s
120	31s	16s	122s	106s

Note that times are nearly halved, along with precision, for the BW-limited linear solve phase, indicating that the BW can be at least doubled before hitting the next bottleneck!

NKS for transport modeling





CVD of GaAs in 3D horizontal rotating disk reactor

- **Aztec: efficient parallel linear solvers**
  - **Krylov methods + preconditioners**
  - **variable overlap Schwarz**
  - **subdomain “solvers”: ILU, MILU, ILUT, BILU, LU, Krylov, …**

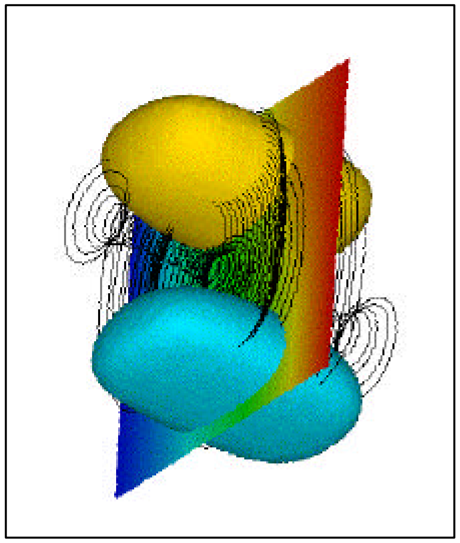
  www.cs.sandia.gov/CRF/aztec1.html
- **ML: parallel multigrid linear solvers**
  - **Algebraic : classical, smoothed aggregation, H-curl**
  - **Geometric: FE basis domain decomp., grid refinement**

  www.cs.sandia.gov/~tuminaro/ML_Description.html
- **MPSalsa: parallel transport / reaction system simulator**
  - **GLS FE formulation; variable density fluid flow, heat and mass transfer with non-equilibrium chemical reactions**
  - **Fully-coupled Newton/Krylov iterative solution methods**
  - **CVD, catalytic reactors, combustion, chemical detectors**

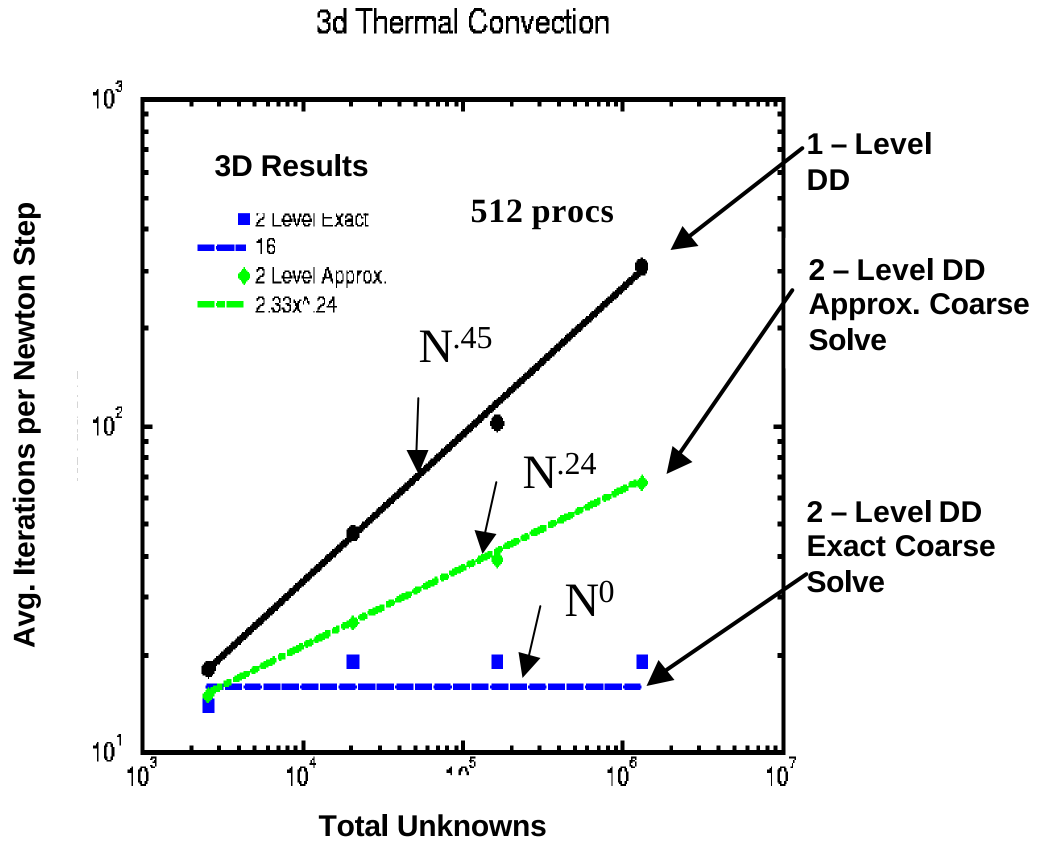
  www.cs.sandia.gov/CRF/MPSalsa

Algorithmic scaling of 1- and 2-level DD preconditioners

Thermal Convection Problem (Ra = 1000)



Temperature iso-lines on slice plane, velocity iso-surfaces and streamlines in 3D



c/o J. Shadid and R. Tuminaro

Newton-Krylov solver with Aztec non-restarted GMRES with 1 – level domain decomposition preconditioner, ILUT subdomain solver, and ML 2-level DD with Gauss-Seidel subdomain solver. Coarse Solver: “Exact” = Superlu (1 proc), “Approx” = one step of ILU (8 proc. in parallel)

Nonlinear Robustness

- **Problem:**
  - Attempts to handle nonlinear problems with nonlinear implicit methods often encounter stagnation failure of Newton away from the neighborhood of the desired root
- **Algebraic solutions:**
  - Linesearch and trust-region methods
  - "Forcing terms"
- **Physics-based solutions:**
  - Mesh sequencing
  - Continuation (homotopy) methods for directly addressing this through the physics, e.g., pseudo-transient continuation
  - Transform system to be solved so that neglected curvature terms of multivariate Taylor expansion truncated for Newton's method are smaller (nonlinear Schwarz)

Standard robustness features

- PETSc contains in its nonlinear solver library some standard algebraic robustness devices for nonlinear rootfinding from Dennis & Schnabel, 1983
- Line search
  - Try to ensure that *F*(*u*) is strictly monotonically decreasing
  - Parameterize reduction of |*F*(*u* + *?du*)| along Newton step *du*
  - Solve scalar minimization problem for *?*
- Trust region
  - Define a region about the current iterate within which we trust a model of the residual
  - Approximately minimize the model of the residual within the region (again with low-dimensional parameterization of convex combination of descent direction and Newton direction)
  - Shrink or expand trust region according to history

Standard robustness features

- PETSc contains in its nonlinear solver library standard algebraic robustness devices for nonlinear rootfinding from Eisenstat & Walker (1996)
  - EW'96 contains three heuristics for the accuracy with which a Newton step should be solved
  - relies intrinsically on iterative solution of the Newton correction equation
  - tolerance for linear residual ("forcing factor") computed based on norms easily obtained as by-products of the rootfinding computation – little additional expense
  - tolerance tightens dynamically as residual norm decreases during the computation
  - "oversolving" not only wastes execution time, but may be less robust, since early Newton directions are not reliable

Time-implicit Newton-Krylov-Schwarz

For accommodation of unsteady problems, and nonlinear robustness in steady ones, NKS iteration is wrapped in time-stepping:

Pseudo-time loop

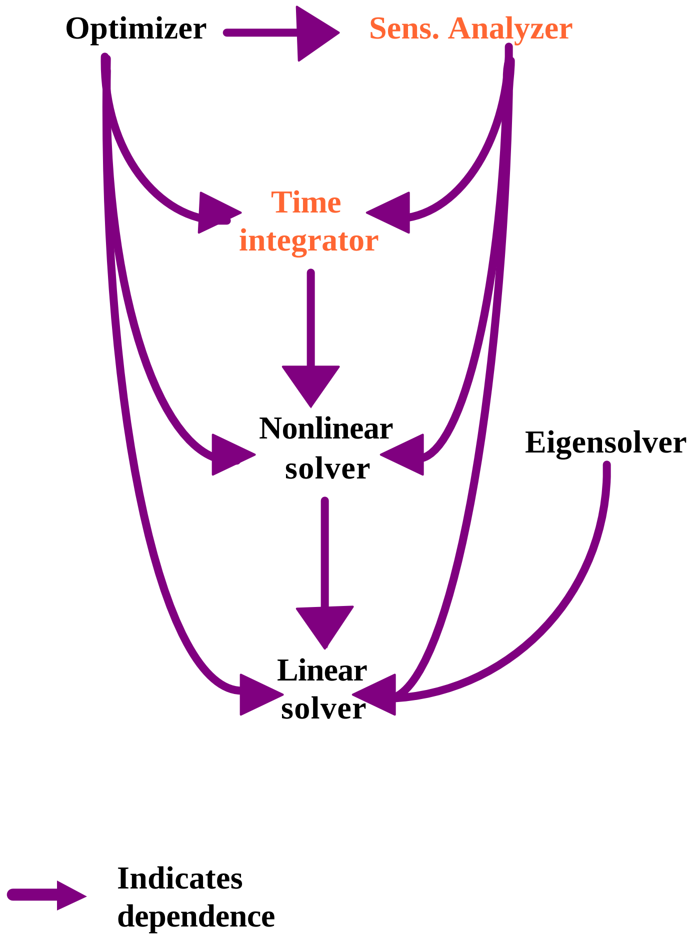
NKS loop

```
for (l = 0; l < n_time; l++) {
    select time step
    for (k = 0; k < n_Newton; k++) {
        compute nonlinear residual and Jacobian
        for (j = 0; j < n_Krylov; j++) {
            forall (i = 0; i < n_Precon ; i++) {
                solve subdomain problems concurrently
            } // End of loop over subdomains
            perform Jacobian-vector product
            enforce Krylov basis conditions
            update optimal coefficients
            check linear convergence
        } // End of linear solver
        perform DAXPY update
        check nonlinear convergence
    } // End of nonlinear loop
} // End of time-step loop
```

Time integrators w/ sensitivity analysis

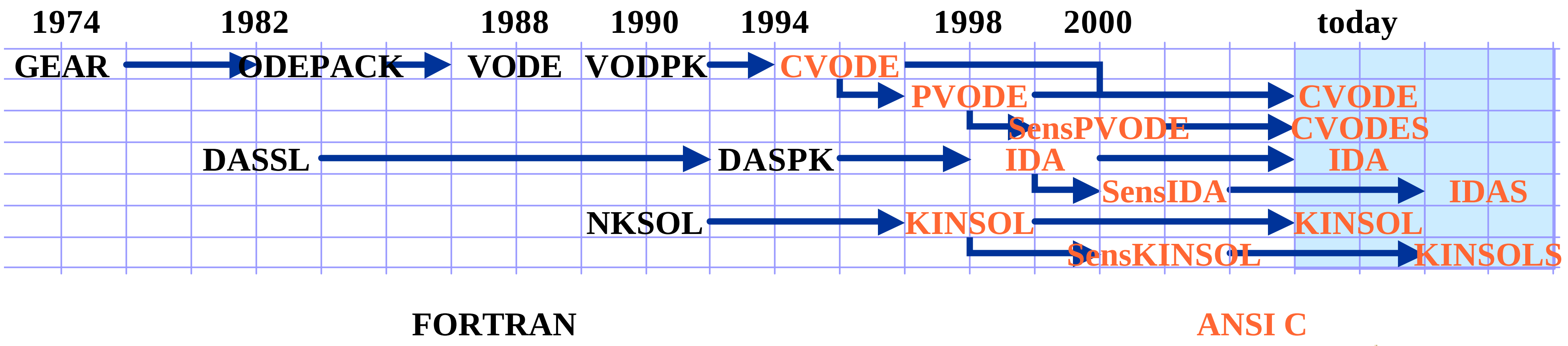
- Transient multirate problems require stiff integrators, a known art, assuming a powerful nonlinear solver capability
- SUNDIALS and PETSc both implement the PVODE backward differentiation schemes for temporal discretization
- PETSc supplies a variety of distributed data structures
- Users who want to use their own data structures, or to utilize built-in sensitivity estimation may prefer SUNDIALS
- Especially recommended for parameterized applications, requiring uncertainty quantification

$$f(\dot{x}, x, t, p) = 0$$



Integrators progress

- PVODE, IDA, and KINSOL (an NK solver) now wrapped together in SUNDIALS and augmented with forward and adjoint sensitivity analysis capabilities
- Embodies decades of work in variable-order, variable-timestep method-of-lines and Newton-Krylov solvers at LLNL



SUNDIALS

SUite of Nonlinear and DIfferential/ALgebraic equation Solvers

Pseudo-transient continuation (Ytc)

- Solve *F(u)=0* through a series of problems derived from method of lines model

$$f^{\ell}(u) = \frac{u - u^{\ell-1}}{t^{\ell}} + F(u) = 0, \ \ \ell = 1,2,\cdots \qquad (*)$$

- t^{ℓ} is advanced from $t^{0} << 1$ to ¥ as $\ell \to \infty$ so that u^{ℓ} approaches the root
- With initial iterate for u^{ℓ} as $u^{\ell-1}$, the first Newton correction for (*) is

$$u^{\ell} = u^{\ell-1} - [\frac{1}{t^{\ell}} I + F'(u^{\ell-1})]^{-1} F(u^{\ell-1})$$

- Note that ||*F(u)*|| can climb hills during Ytc
- Can subcycle inside physical timestepping

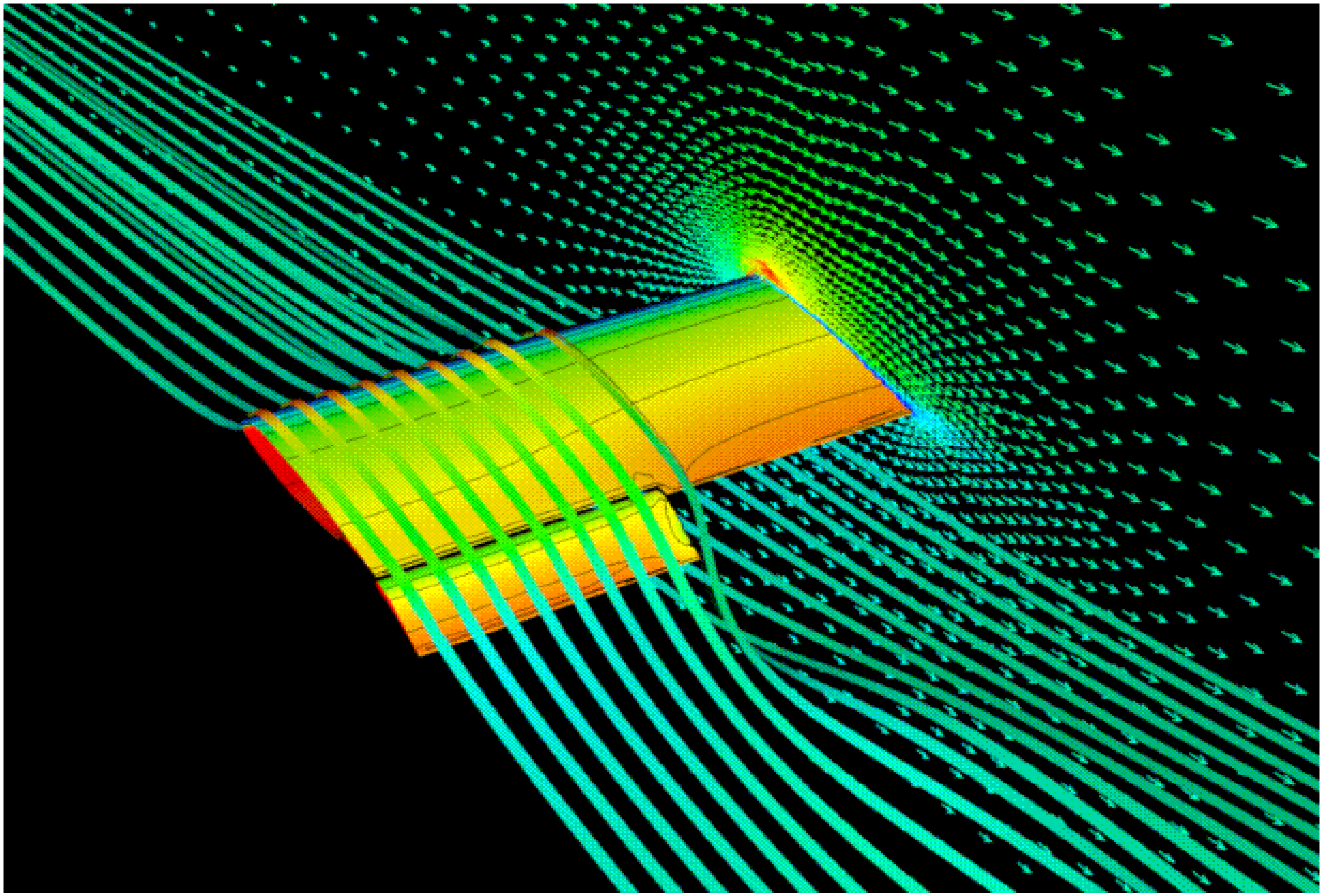
Algorithmic tuning - continuation parameters

- "Switched Evolution-Relaxation" (SER) heuristic

$$N_{CFL}^{l} = N_{CFL}^{0}\left(\frac{\|f(u^{0})\|}{\|f(u^{l-1})\|}\right)^{p}$$

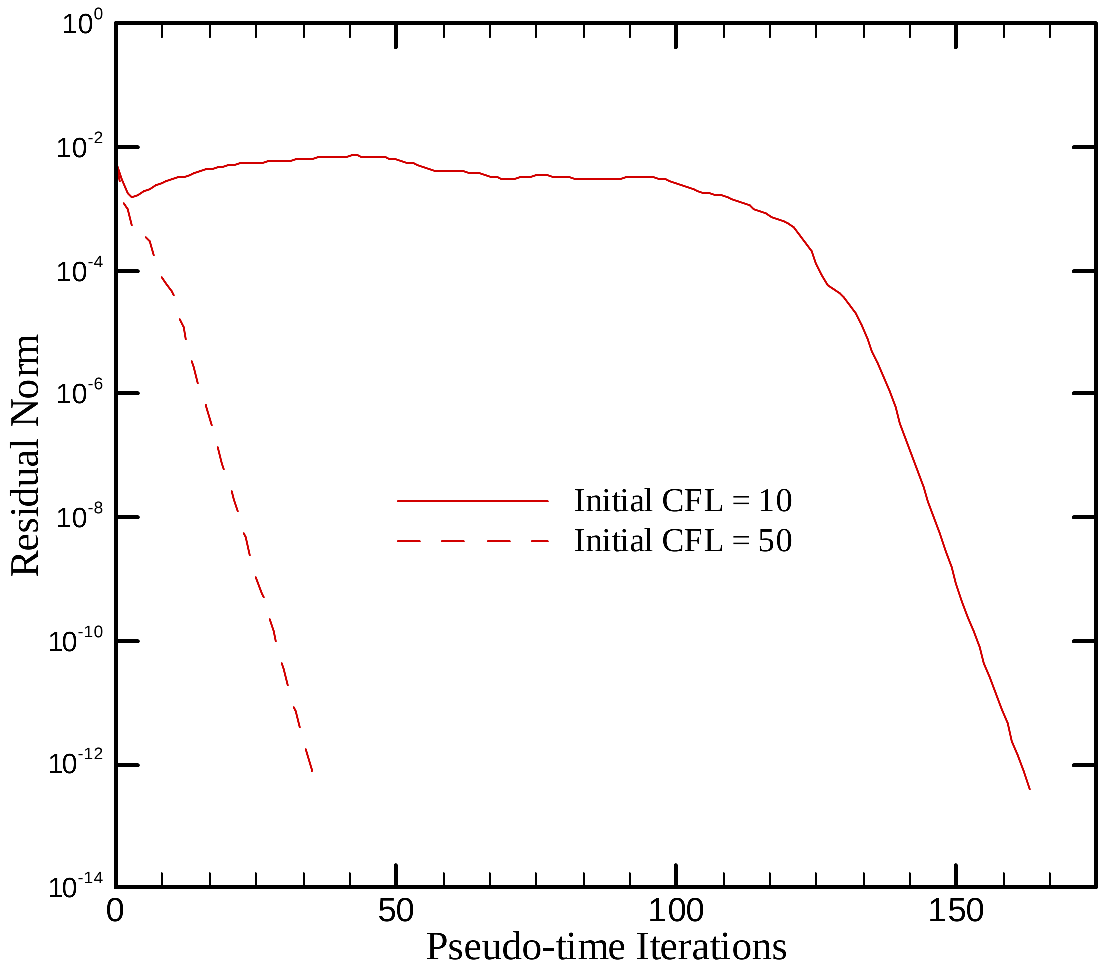
- Analysis in SIAM papers by Kelley & Keyes (1999 for parabolized, 2002 for mixed elliptic/parabolized)
- Parameters of interest:
  - Initial CFL number
  - Exponent in the Power Law
    - = 1 normally
    - > 1 for first-order discretization (1.5)
    - < 1 at outset of second-order discretization (0.75)
  - Switch-over ratio between FO and SO

Application Domain: Computational Aerodynamics

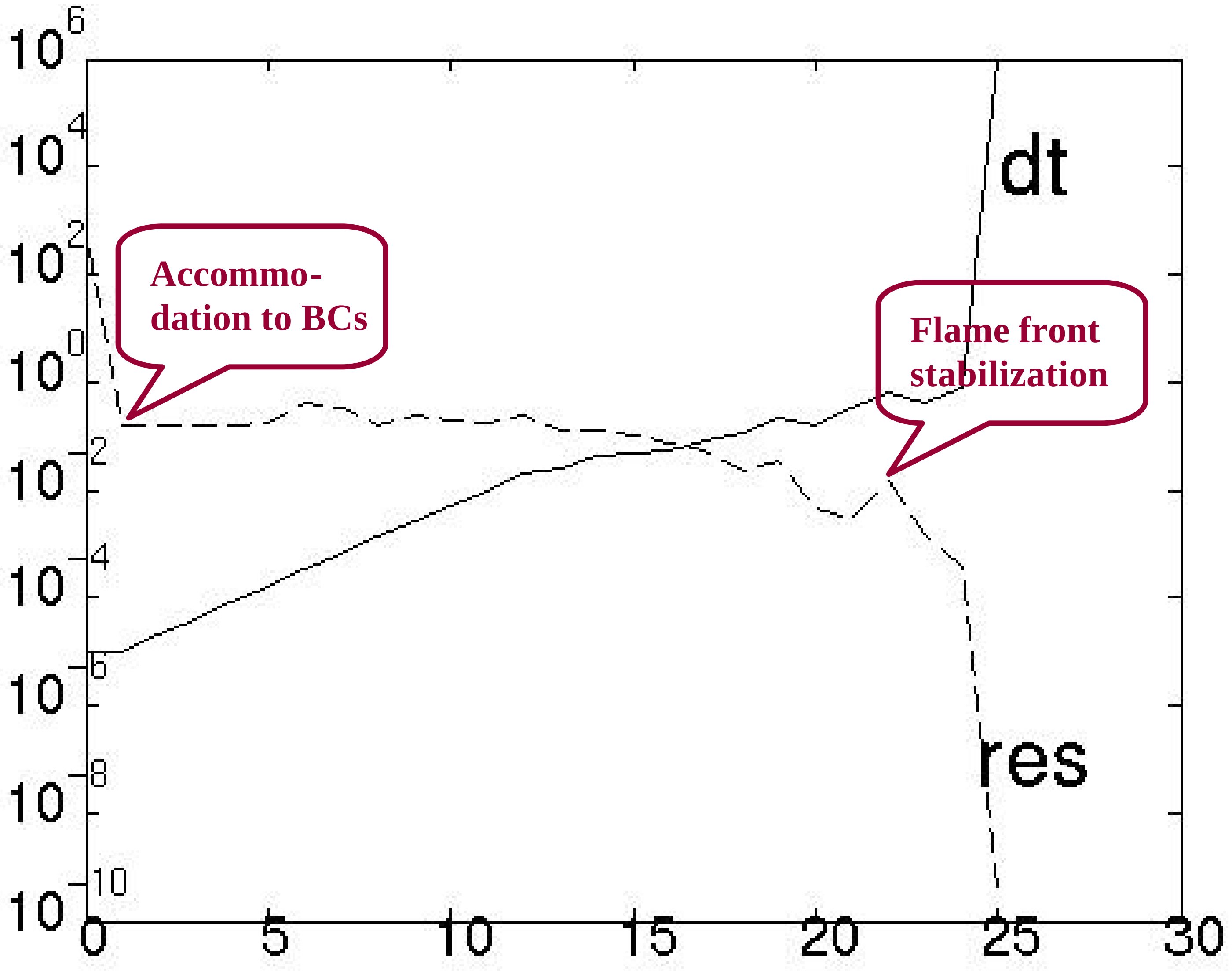


Effect of initial CFL number

ONERA M6 aerodynamics problem on grid of 2.8M vertices



Ytc in combustion application



Velocity-vorticity governing equations

x-velocity

$$-\nabla^2 u - \frac{\partial w}{\partial y} = 0$$

y-velocity

$$-\nabla^2 v + \frac{\partial w}{\partial x} = 0$$

vorticity

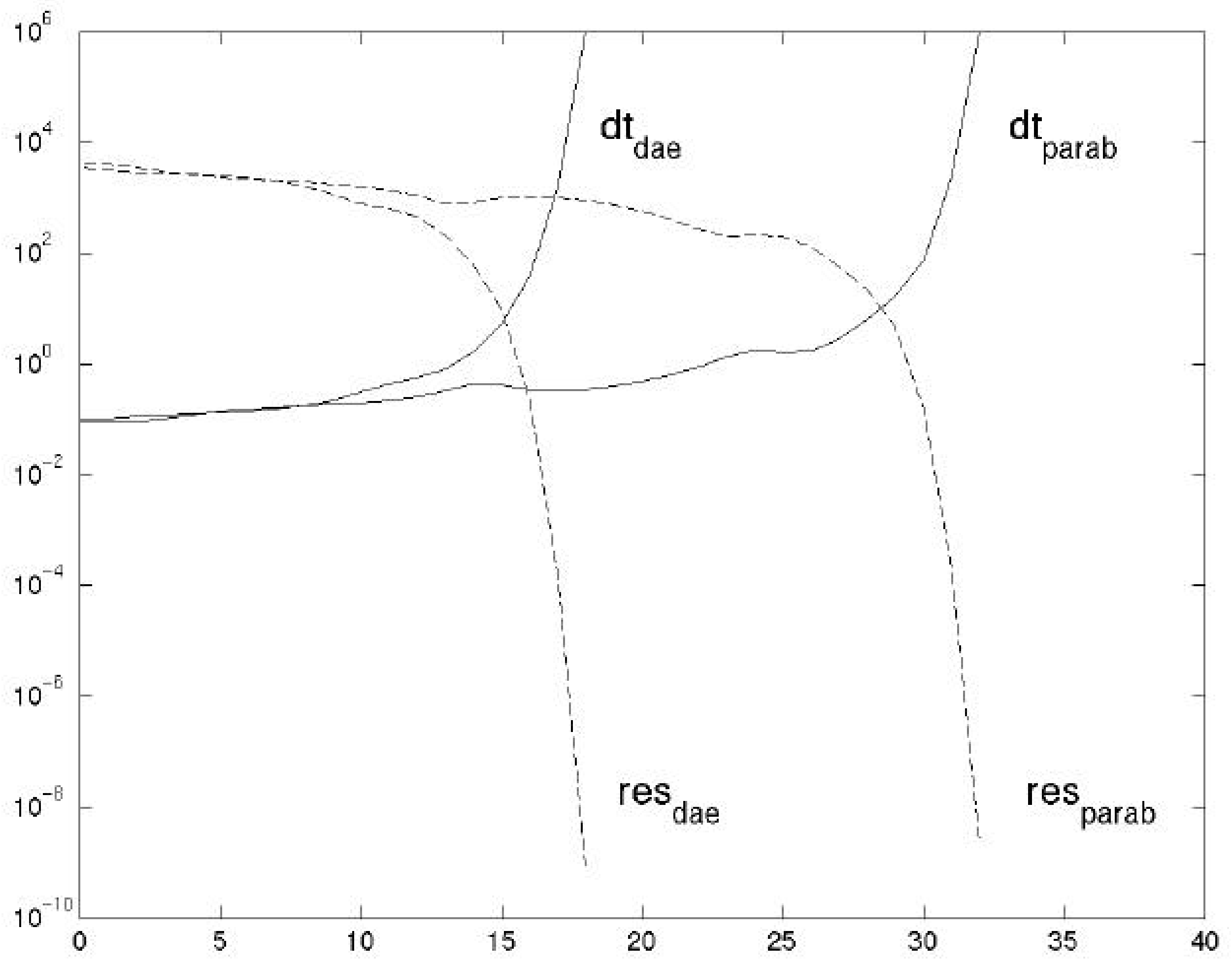
$$\frac{\partial w}{\partial t} - \nabla^2 w + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} - \mathrm{Gr}\frac{\partial T}{\partial x} = 0$$

internal energy

$$\frac{\partial T}{\partial t} - \nabla^2 T + \mathrm{Pr}(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}) = 0$$

Extension to DAE systems

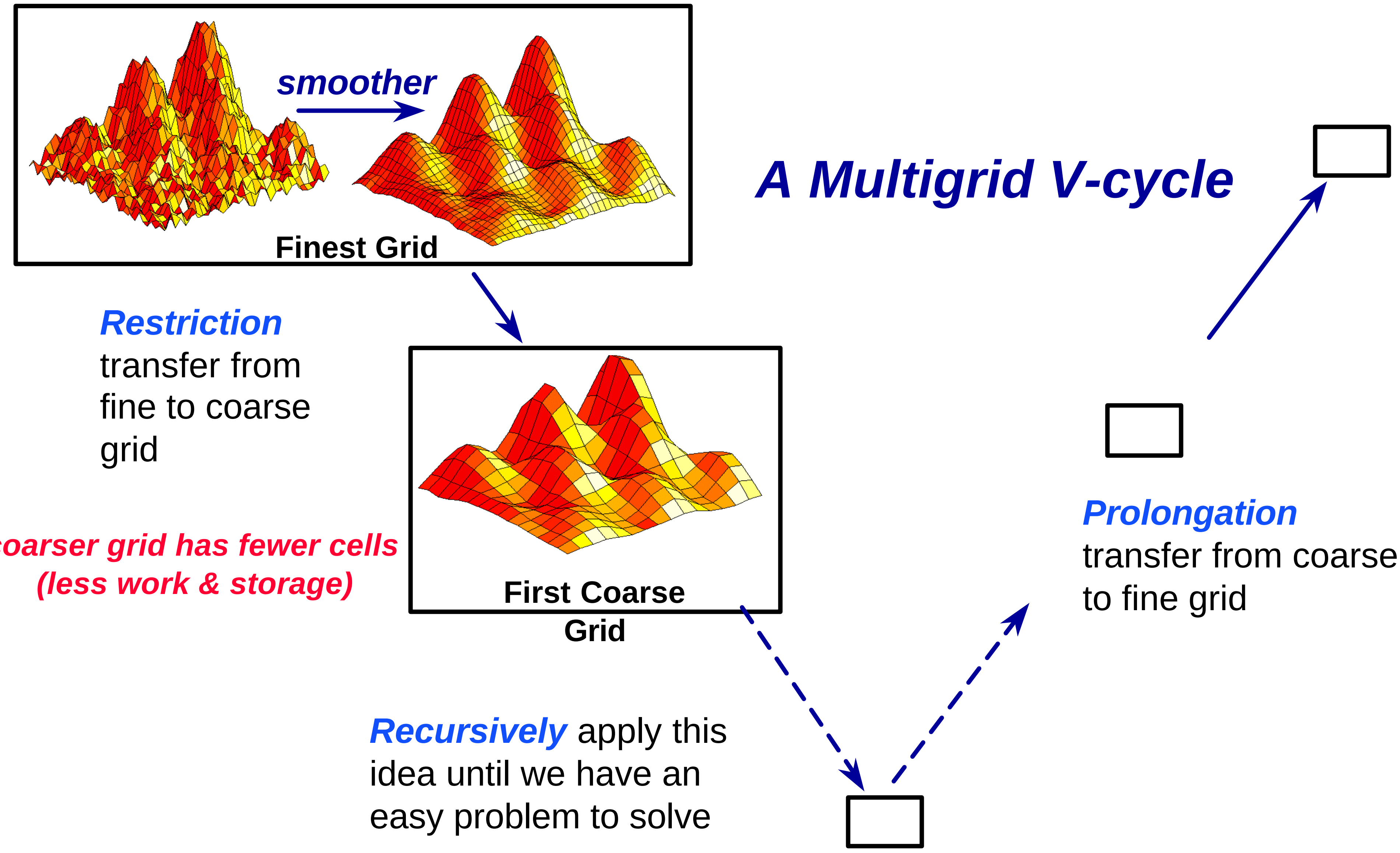
- Some PDEs act as elliptic constraints on the others and should not be parabolized, e.g., incompressible flow (continuity, streamfunction-vorticity, velocity-vorticity)
- Fast-converging results enforced incompressibility; slower (fully parabolized) did not



Mesh sequencing

• Technique for robustifying nonlinear rootfinding for problems based on continuum approximation
• Relies on several levels of refinement from coarse to fine
• Theory exists showing (for nonlinear elliptic problems) that, asymptotically, the root on a coarser mesh, appropriately interpolated onto a finer mesh, lies in the domain of convergence of Newton’s method on the finer grid

Multilevel preconditioning



A Multigrid V-cycle

Restriction transfer from fine to coarse grid

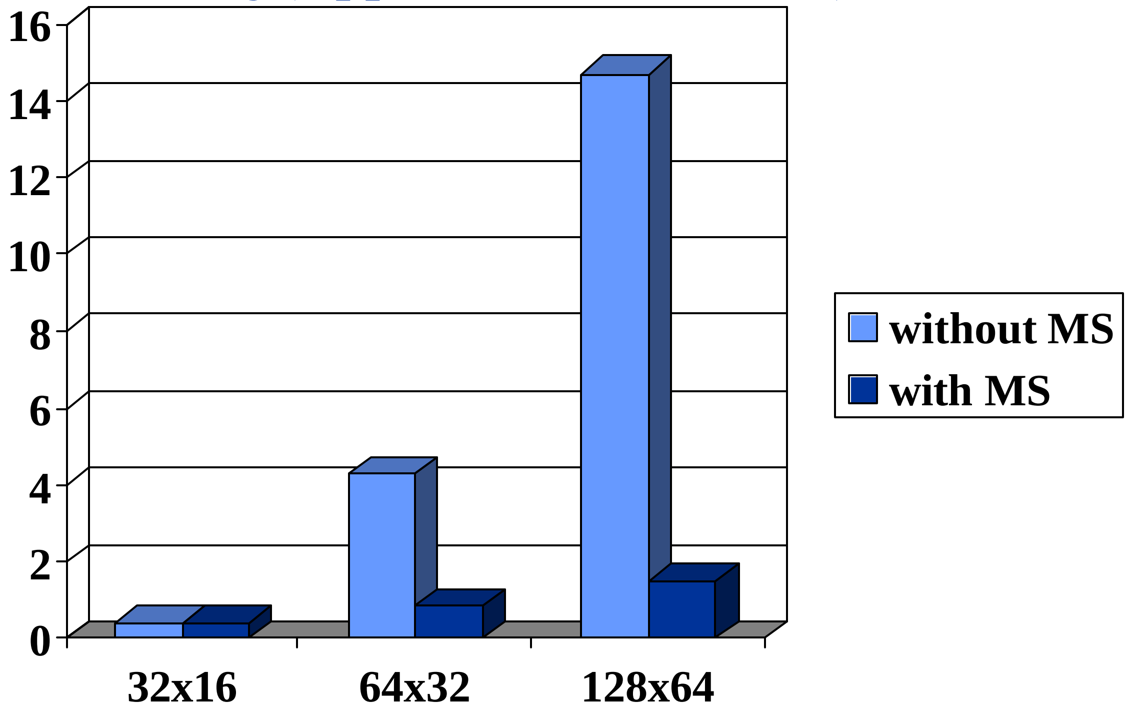
coarser grid has fewer cells (less work & storage)

Prolongation transfer from coarse to fine grid

Recursively apply this idea until we have an easy problem to solve

Mesh sequencing example

- **From Knoll & McHugh (SIAM J. Sci. Comput., 1999) summarized in Ref [1]**
- **Execution times for 8-equation 2d BVP steady-state coupled edge plasma/Navier-Stokes problem**
- **Each grid in sequence is solved from a "cold" initial iterate *or* initialized for Newton's method by the solution on the previous coarse grid**
- **See Smooke & Mattheij (Appl. Num. Math, 1985) for BVP theory**



Other continuation methods

- There is often a physical "knob," such as Reynolds number, that can be varied to "sneak up" on a hard problem
- Let the parameter at which the solution is sought be p and let the solution at a value p^0 be such that $F(u, p^0)=0$ be "easy" (e.g., linear)
- By implicit differentiation of $F(u, p)=0$, we get

$$\frac{\partial F}{\partial u}\frac{\partial u}{\partial p}+\frac{\partial F}{\partial p}=0 \quad \text{or} \quad \boxed{\frac{\partial u}{\partial p}}=-\left(\frac{\partial F}{\partial u}\right)^{-1}\frac{\partial F}{\partial p}$$

- By Taylor expansion

$$u^{\ell} \approx u^{\ell-1}+\boxed{\frac{\partial u}{\partial p}(p^{\ell-1})}\,(p^{\ell}-p^{\ell-1})$$

- This allows bootstrapping with a series of Newton problems

Nonlinear Schwarz preconditioning

- Nonlinear Schwarz has Newton both *inside* and *outside* and is fundamentally Jacobian-free
- It replaces $F(u)=0$ with a new nonlinear system possessing the same root, $\Phi(u)=0$
- Define a correction $d_i(u)$ to the i^{th} partition (e.g., subdomain) of the solution vector by solving the following local nonlinear system:

$$R_i F(u+d_i(u))=0$$

  where $d_i(u)\in\Re^n$ is nonzero only in the components of the i^{th} partition
- Then sum the corrections: $\Phi(u)=\sum_i d_i(u)$

Nonlinear Schwarz, cont.

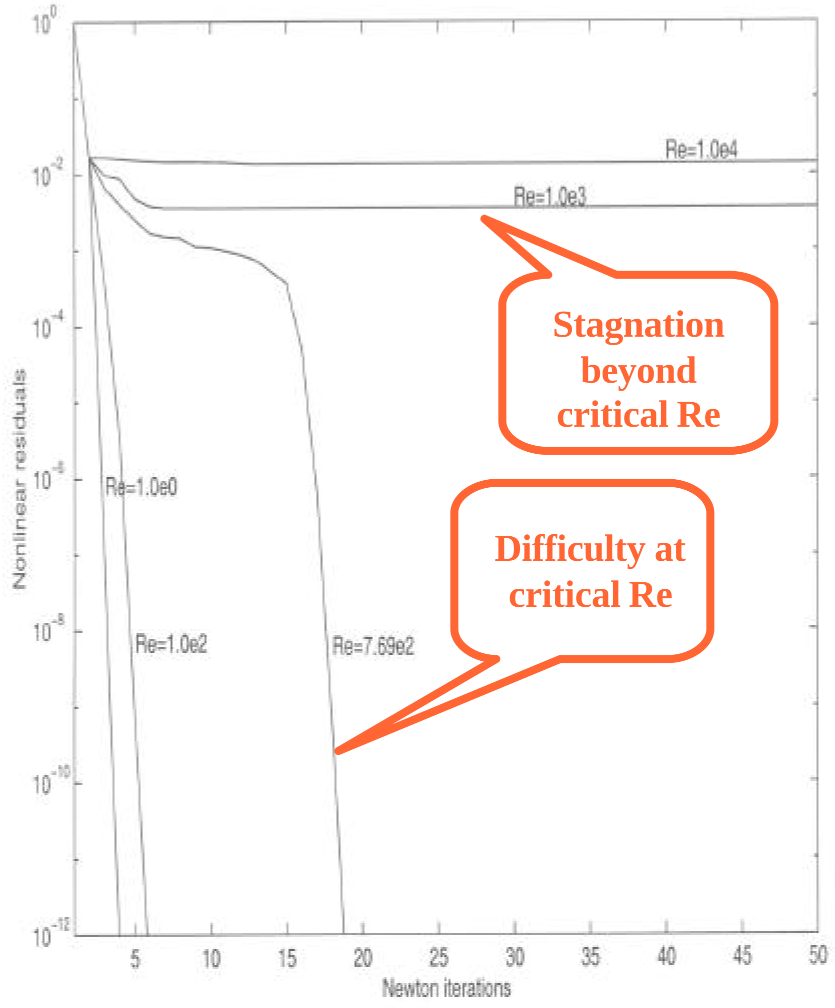
- It is simple to prove that if the Jacobian of $F(u)$ is nonsingular in a neighborhood of the desired root then $\Phi(u)=0$ and $F(u)=0$ have the same unique root
- To lead to a Jacobian-free Newton-Krylov algorithm we need to be able to evaluate for any $u,v \in \Re^n$:
  - The residual $\Phi(u)=\sum_i d_i(u)$
  - The Jacobian-vector product $\Phi(u)'v$
- Remarkably, (Cai-Keyes, 2000) it can be shown that

$$\Phi'(u)v \approx \sum_i (R_i^T J_i^{-1} R_i) Jv$$

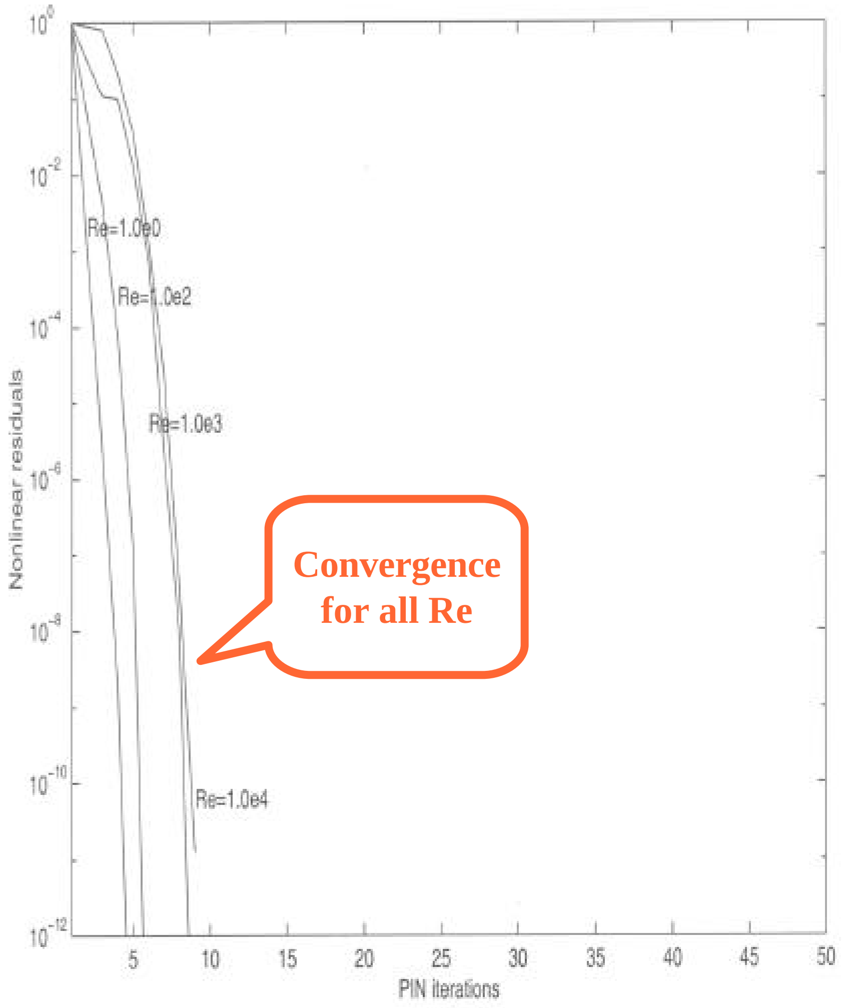
where $J = F'(u)$ and $J_i = R_i J R_i^T$

- All required actions are available in terms of $F(u)$!

Experimental example of nonlinear Schwarz



Newton's method

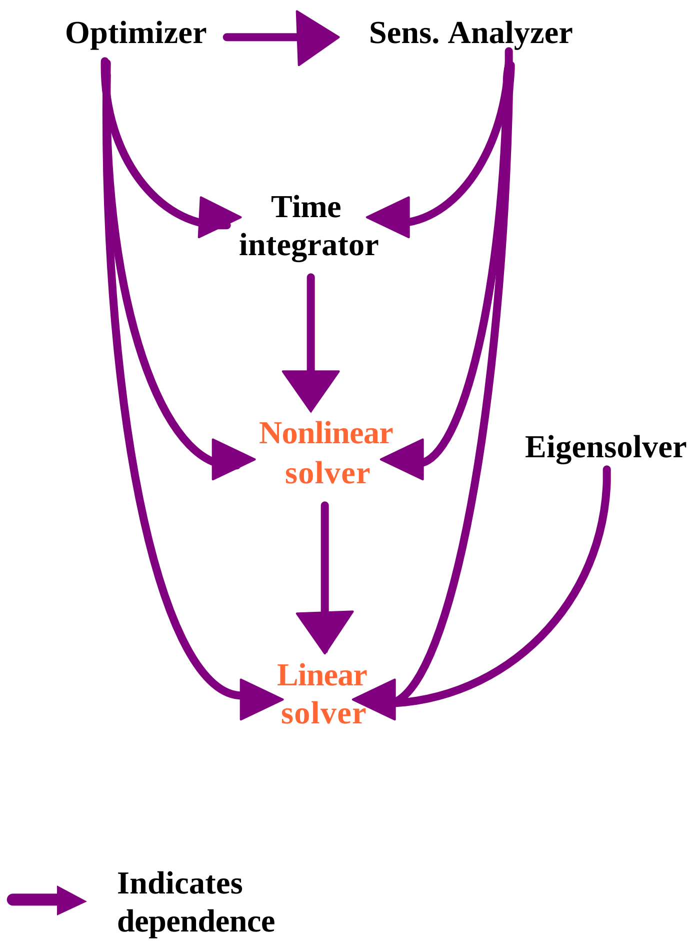


Additive Schwarz Preconditioned Inexact Newton (ASPIN)

Common software infrastructure for nonlinear PDE solvers

- User codes to the problem they are solving, not the algorithm used to solve the problem
- Implementation of various algorithms reuse common concepts and code when possible, without losing efficiency

$$Ax = b$$



Encompassing …

- **Newton's method**
  - **Direct solvers**
  - **Matrix-based preconditioned solvers**
  - **Matrix-free methods**
  - **Multigrid linear solvers (Newton-MG)**
    - **Matrix-based and matrix-free**
- **Nonlinear multigrid**
  - **a.k.a. Full approximation scheme (FAS)**
  - **a.k.a. MG-Newton**

Software engineering ingredients

- Standard solver interfaces
- Solver libraries
- Automatic differentiation (AD)
- Code generation

Algorithm review

$F(u) = 0$, **Jacobian** $A(u)$

Newton

$$u \leftarrow \bar{u} - A^{-1}(\bar{u})F(\bar{u})$$

Newton – SOR (1 inner sweep)

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(\bar{u})\{F_i(\bar{u}) - \sum_{j<i} A_{ij}(\bar{u})[\bar{u}_j - u_j]\}$$

SOR-Newton (1 inner sweep)

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(u)F_i(u)$$

Cute observation

SOR-Newton

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(u) F_i(u)$$

With approximations

$$A_{ii}(u) \approx A_{ii}(\bar{u})$$

$$F_i(u) \approx F_i(\bar{u}) + \sum A_{ij}(\bar{u})[u_j - \bar{u}_j]$$

Gives Newton-SOR

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(\bar{u})\{F_i(\bar{u}) - \sum_{j<i} A_{ij}(\bar{u})[\bar{u}_j - u_j]\}$$

Þ Matrix-free linear relaxation

(Gauss-Seidel)

is almost identical to nonlinear relaxation

Function and Jacobian evaluation

- **FAS requires pointwise**
- **Newton desires global**
- **Newton-MG desires both**

Automatic Differentiation

- Given code for $F(u)$ can compute
  - $A(u)$ and
  - $A(u)*w$ efficiently
- Given code for $F_i(u)$ can compute
  - $A_{ii}(u)$ and
  - $\sum_j A_{ij}(u)w_j$ efficiently

Code generation (in-lining ☺)

- **Inside the small dimensional Newton methods is a user-provided function and (AD) Jacobian**
- **Big performance hit if handled directly with components**

Coarse grid correction is not an issue ☺

- **Newton-MG**

$$A_H(\overset{\frown}{R}\bar{u})c_H = RF(\bar{u})$$

$$u \leftarrow u - R^T c_H$$

- **MG-Newton**

$$F_H(\overset{\frown}{R}\bar{u} + c_H) - F_H(\overset{\frown}{R}\bar{u}) + RF(\bar{u}) = 0$$

Conclusion

The algorithmic/mathematical building blocks for Newton-MG and MG-Newton are essentially the same

Thus the software building blocks should be also (and they will be in the next release of PETSc).