Applications of Anderson Accelerated Fixed Methods to DOE Applications

Presented to: TAMU Seminar on Industrial and Applied Mathematics
Mar. 7, 2016

Carol S. Woodward
Outline

- Overview of DOE lab system
- Comments on mathematician’s job at a DOE lab
- Needs for nonlinear solvers
- Newton-Krylov methods
- Fixed Point and Picard methods
- Anderson acceleration
- Results in applications
  - Subsurface flow
  - Dislocation dynamics
- Conclusions
The DOE mission is to ensure America’s security and prosperity by addressing its energy, environmental and nuclear challenges through transformative science and technology solutions.

- Maintain a vibrant U.S. effort in science and engineering as a cornerstone of our economic prosperity
  - Biological science
  - Chemical science
  - Computing
  - Environmental science
  - Materials science
  - Physics
- Enhance nuclear security through defense, nonproliferation, and environmental efforts

Two main types of DOE labs:
- National Nuclear Security Administration: Lawrence Livermore, Los Alamos, Sandia [2 locations], …
- Office of Science: Argonne, Oak Ridge, Lawrence Berkeley, Pacific Northwest, …
The DOE is composed of many offices

Off of Sci. Programs:
- Advanced Scientific Computing Research
- Biological and Environmental Research
- Basic Energy Sciences
- High Energy Physics
- Nuclear Physics
- Fusion Energy Sciences

Additional offices include:
- Applied Mathematics
- Computer Science
- Office of Electrical Delivery and Reliable Energy
- Office of Environmental Management
- Office of Energy Efficiency & Renewable Energy
- Office of Fossil Energy
- And many more…
Many important U.S. national missions depend on high-performance computing and simulation:

- Nuclear and conventional weapons
- Global and regional climate
- Nonproliferation
- Counter-terrorism
- Cryptanalysis
- Energy Security
- Drug design
- Engineering design for economic competitiveness
- Data management for massive experiments
- Cyber security
Foundational mathematics and computer science research support our advances in high-performance computing

Scalable numerical algorithms for petascale scientific simulation

New tools and programming paradigms for portable performance on hundreds of thousands of processors

Interactive data exploration tools for petabyte-sized data sets
CASC furthers the lab mission through advances in mathematics, computer science and data science

- Advancing physics understanding by increasing simulation fidelity
  - Multiple physics models
  - Multiple scales
  - Uncertainty quantification

- Extreme scale algorithms and tools
  - Many-core HPCs
  - Manage software complexity
  - Testing, debugging, understanding

CASC R&D supports a wide range of applications: atomistic to complex multi-physics

CASC Statistics
- 104 researchers (22 Postdocs, 30 FNs)
- Founded in 1996
- Total Budget is approximately $45M

Increased insight into massive data sets
- Data compression and visualization
- Text and graph analysis
What does a mathematician do at a DOE lab?

- Enable new science understanding through modeling and simulation on high performance computers
  - Innovation in algorithms – mostly numerical analysis
  - Innovative implementations
- Incorporate new algorithms into scientific codes
  - Contribute to mathematical software packages
  - Interface packages to scientific simulation codes
- Work in multidisciplinary teams
  - Listen to scientists to understand their needs
  - Interface with computer scientists to understand impacts of computer architectures on algorithms
- Interface with the research community – conferences and papers
- Develop new directions for computational mathematics work
  - Participate in workshops and report-writing to shape DOE investments
  - Write proposals
What do we look for in computational scientists?

- **Deep knowledge in a specific area**
- **Science knowledge:** Researchers should have a working knowledge of 2-3 science areas.
- **Methods knowledge:** strong understanding of the convergence and accuracy of numerical methods used for the applications in which they work
- **Coding knowledge:** at least one compiled language
- **Software Engineering:** algorithms and data structures, software architecting and design, parallel programming models, and software quality assurance concepts
- **Computer architecture:** be able to adapt algorithms to achieve good performance on new machines
- **Tools:** build systems, source code control systems, visualization systems, performance analysis and memory checking tools, scripting languages, debuggers, and efficient input/output formats
- **Social skills:** we expect researchers to be able to interact productively with other researchers from a variety of backgrounds
Outline

- Overview of DOE lab system
- Comments on mathematician’s job at a DOE lab

- Needs for nonlinear solvers
- Newton-Krylov methods
- Fixed Point and Picard methods
- Anderson acceleration
- Results in applications
  - Subsurface flow
  - Dislocation dynamics
- Conclusions
Implicit approaches can offer benefits over explicit methods

- Multirate problems may involve slowly varying phenomena with rapid (but unimportant) transients:
  - Fast magnetosonic wave in magnetohydrodynamics
  - Core physics in supernovae
  - Surface gravity waves in climate
  - Fast chemical reactions among intermediates

- Explicit methods: resolve fastest phenomena for stability; simulation purpose can dictate larger step

- Implicit solvers offer stability and accuracy with larger step sizes in multirate problems

- Implicit methods require effective solvers
  - Time steps are more expensive
  - Need efficient nonlinear solver for each step
Newton-Krylov methods are heavily used for solving large-scale nonlinear systems

1. Starting with $U^0$, want $U^*$ such that $F(U^*) = 0$

2. Repeat for each $k$ until
   \[ \|F(U^{k+1})\| \leq tol \]
   a. Solve (approximately) with Krylov method
      \[ \|J(U^k)s^k + F(U^k)\| \leq \eta_k \|F(U^k)\| \]
   b. Update, $U^{k+1} = U^k + \lambda s^k$

- Combine Newton nonlinear solver with Krylov linear solver
- Approximate matrix-vector products allow for easy first implementation

\[ J(u)v \approx \frac{F(u + \sigma v) - F(u)}{\sigma} \]

*courtesy of D. Reynolds (SMU)*
We employ Newton’s iteration (NI) because of its fast quadratic convergence

- Assuming proper choices of $\eta_k$, NI is \textit{q-quadratic}

- For constant $C$:
  \[ \left\| U^{k+1} - U^* \right\| \leq C \left\| U^k - U^* \right\|^2 \]

- Well-developed convergence theory results in a number of considerations:
  - Theory is local: current iterate must be close to the solution for fast convergence
  - Requires sufficient continuity of $F$ and $J$, finite difference approx exacerbates this
  - Must have a good preconditioner for the linear solves
  - Need derivative information or suffer possible loss of nonlinear solver speed

![Types of Nonlinear Convergence diagram courtesy of D. Reynolds (SMU)](chart.png)
Fixed point iteration is an alternative to Newton with only linear convergence

- Define an iterative scheme to solve \( F(u) = u - G(u) = 0 \) as,

\[
\text{Starting with } U^0, \text{ want } U^* \text{ such that } F(U^*) = 0
\]

Repeat for each \( k \) until
\[
\left\| F(U^{k+1}) \right\| \leq tol
\]

Set \( U^{k+1} = G(U^k) \)
End

- Alternative to Newton: no derivatives!
- This is an easy method to implement, but convergence can be slow
- Requires \( G \) to be a contraction: \( \left\| G(x) - G(y) \right\| \leq \gamma \left\| x - y \right\|, \quad \gamma < 1 \)
- The closer \( \gamma \) is to 1, the slower the convergence
- Fixed point iteration has a global but linear convergence theory
A Picard iteration can be set up to use fixed point

A Picard iteration is a fixed point method formed from writing $F$ as the difference of a linear, $L_\text{u}$, and a nonlinear, $N(u)$, operator:

$$F(u) = L_\text{u} - N(u); \quad L^{-1}N(u) = u - L^{-1}F(u) \equiv G(u)$$

Then, making the approximation $u^{k+1} \approx u^k - L^{-1}F(u^k)$ gives the fixed point iteration:

$$u^{k+1} = G(u^k)$$

This is like Newton with $L$ approximating $J$

Convergence depends on spectral radius $\sigma = \rho(I - L^{-1}J(u^k)) < 1$. The smaller $\sigma$ is, the faster convergence will be.

Each iteration requires the solution of a linear system
- More expensive than regular fixed point
- Choice of $L$ can result in no derivative requirements
Anderson Acceleration (AA) uses prior residuals to accelerate fixed point

Initialize $u^0$.  
Set $u^1 = G(u^0)$.  
For $k = 1, 2, \ldots$, until $\|F(u^k)\| < \tau$  
   Set $m_k = \min\{m, k\}$.  
   Set $F_k = (f_{k-m_k}, \ldots, f_k)$, where $f_i = G(u^i) - u^i$.  
   Solve $\min_\alpha \left\| F_k \alpha \right\|_2$ s.t. $\sum_{i=0}^{m_k} \alpha_i = 1$.  
   Set $u^{k+1} = \sum_{i=0}^{m_k} \alpha_i^{(k)} G(u^{k-m_k+i})$.  
end

Like a nonlinear GMRES

For linear problem, $u_{k+1} = G(u_{\text{min}})$ where $u_{\text{min}}$ has minimal fixed point residual over space spanned by prior $m_k$ iterates

Have a choice as to the number of prior residuals to save, $m$

New cost: size $m_k$ least squares solve

Anderson, 1965; Walker and Ni, 2011
Anderson acceleration has been reinvented by many authors unaware of each other.

Europe

FSI Interface Quasi-Newton

- Brummelen et al. 2005, I-GMRES
- Vierendeels et al. 2007, IBQN-LS
- Degroote et al. 2009, IQN-ILS
- Ueckermann et al. 2013, V-IQN-ILS
- Bogaers et al. 2014, MVQN

US

Fixed-Point Acceleration

- Anderson Acceleration 1965
- Fang & Saad 2007, Theory: AA & GB
- Walker et al. 2010, Theory: AA
- Lott et al. 2012, AA groundwater flow

Others: Marks and Luke (08), Washio and Osterlee (97), Carlson and Miller (98)
Theory: Toth and Kelley (15), Rohwedder and Schneider (11)

Figure courtesy of B. Ueckermann (TUM)
Full convergence has not yet been shown

- Eyert (1996) showed relationship to a quasi-secant update
- Fang and Saad (2007) showed equivalence of an Anderson Family of methods to a Broyden-type update
- Walker and Ni (2011) showed equivalence of AA to GMRES for linear problems (iterates of one can be obtained from the other)
  - Truncated AA (with $m_k < m$) is like truncated GMRES
  - Restarted AA is like restarted GMRES
  - Gave some perspectives on efficient implementation
- Toth and Kelley (2015) showed a *local convergence* proof of r-linear convergence and contraction properties of resulting accelerated iterates assuming fixed point function is a contraction
We follow implementation considerations outlined by Walker and Ni

- Pose least squares problem as unconstrained minimization problem formulated by Fang & Saad and advocated by Walker & Ni

\[
\min_{\gamma=(\gamma_0,\ldots,\gamma_{m_k-1})^T} \left\| f_k - H_k \gamma \right\|_2
\]

\[
H_k = (\Delta f_{k-m} \ldots \Delta f_{k-1}), \quad \Delta f_i = f_{i+1} - f_i
\]

\[
\alpha_0 = \gamma_0, \quad \alpha_i = \gamma_i - \gamma_{i-1}, \quad \alpha_{m_k} = 1 - \gamma_{m_k-1}
\]

- Easier to handle successive solutions
- Some computational results indicate this may be better conditioned than prior form

- Solve least squares problem with QR factorization and sliding window
  - Add/drop one column each iteration and use QR updating techniques
  - Drop columns to maintain conditioning if needed
We looked at effectiveness of AA for variably saturated subsurface flow problems

Richards’ Equation: \[ \frac{\partial \theta(h)}{\partial t} - \nabla \cdot (Kk_r(h) \nabla (h + z)) = q, \]

Finite difference discretization:

\[ F_i(h^n) = \Delta z(\theta(h^n_i) - \theta(h^{n-1}_i)) - \Delta t(u_{i+1/2} - u_{i-1/2}) - \Delta t\Delta zq_i = 0, \]

\[ u_{i+1/2} = [K(z)k_r(h^n)]_{i+1/2} \left( \frac{h^n_{i+1} - h^n_i}{\Delta z} + 1 \right) \]

Vector Form:

\[ F(h^n) = \Delta z(\theta(h^n) - \theta(h^{n-1})) + \Delta t(A(h^n)h^n) - \Delta t\Delta zq = 0 \]
Modified Picard Iteration (MPI) combines properties of Newton and Picard
(Celia,Ahuja,Pinder,1987,Zarba,1988)

Approximate

\[ A(h^{k+1})h^{k+1} \approx A(h^k)h^{k+1}, \]
\[ \theta(h^{k+1}) \approx \theta(h^k) + J_\theta(h^k)\Delta h^{k+1} \]

\[ (\Delta z J_\theta(h^k) + \Delta t A(h^k))\Delta h^{k+1} = -F(h^k) \]

\[ h^{k+1} = h^k - (\Delta z J_\theta(h^k) + \Delta t A(h^k))^{-1} F(h^k) \equiv G(h^k) \]

Each evaluation of G only requires a symmetric linear solve
Modified Picard Iteration (MPI) convergence will be linear due to handling of relative permeability
We compared Newton solvers with the Anderson accelerated modified Picard method

- Newton’s method (NI)
- (Modified) Picard (or fixed-point) iteration (MPI)
- Anderson Acceleration of (Modified) Picard (AA)

Experiments attempted to equilibrate costs while using most efficient parameters

- Newton solves through Newton-Krylov package, KINSOL (part of the SUNDIALS suite from LLNL)
  - Line search was Goldstein-Armijo
  - GMRES with 50 vectors for linear solves, 3 restarts allowed
- Modified Picard and Anderson Acceleration implemented on same framework as Newton
  - Conjugate Gradient linear solve
- Linear solves preconditioned with PFMG semicoarsening multigrid from hypre package
- Matrix-vector multiplies use analytic linear system, no finite difference approximations for Newton-GMRES
We compared the methods on a steady state problem isolating the nonlinearity.

Steady-state elliptic problem (Yavneh and Dardyk, 2006):

\[- \frac{\partial}{\partial x} \left( g(u) \frac{\partial u}{\partial x} \right) - \frac{\partial}{\partial z} \left( g(u) \frac{\partial u}{\partial z} \right) = 0, \quad x, z \in (0,1)\]

\[g(u) = \begin{cases} 
1, & u \geq 0, \\
k_r(|u|), & u < 0,
\end{cases}\]

\[u(x,0) = u(x,1) = x + 2(x - 1); \quad u(0,z) = -2; \quad u(1,z) = 1.\]

Initial iterate: Linear between -2 and 1
1,024 x 1,024 cells
Stop on \(||F|| < 10^{-7}\) for all methods
Arithmetic averages of $k_r$ at cell interfaces
Common Van Genuchten parameterization may not satisfy continuity requirements

We use the Van Genuchten parameterization:

\[ k_r(h) = (1 + (\alpha|h|)^n)^{-m/2} \left(1 - \frac{(\alpha|h|)^{n-1}}{(1 + (\alpha|h|)^n)^m}\right)^2, \quad m = 1 - 1/n. \]

Problem gets more difficult as \( n \) decreases and \( \alpha \) increases

For \( n < 2 \), \( k'_r \) is no longer continuous at \( h = 0 \)

As \( n \to 1 \), \( k_r \) approximates step fcn, derivative \( \to \infty \) at \( h = 0 \)

Slightly Loamy Sand, alpha = 0.0599; \( n = 1.51 \)

Derivative is not Lipschitz continuous for \( u=0 \)
We conducted a “head-on” comparison of best in class on a large test suite

\[ \alpha \]

\begin{array}{cccccc}
0.01 & 0.03 & 0.05 & 0.1 \\
0.25 & 0.4 & 0.6 & 0.8 & 1.0 \\
1.2 & 1.4 & 1.6 & 1.8 & 2.0 \\
2.2 & 2.4 & 2.6 & 2.8 & 3.0 \\
3.2 & 3.4 & 3.6 & 3.8 & 4.0 \\
\end{array}

\[ n \]

\begin{array}{cccccc}
1.1 & 1.3 & & & & \\
1.2 & 1.4 & 1.6 & 1.8 & 2.0 \\
2.2 & 2.4 & 2.6 & 2.8 & 3.0 \\
3.2 & 3.4 & 3.6 & 3.8 & 4.0 \\
\end{array}

Used all combinations
Newton performed well for region $n > 1.6$ and $\alpha < 2$

MPI succeeded only for $n > 1.2$, $\alpha < 1$; more $n$ values than INI

MPI+AA succeeded for $n > 1.4$ for all values of $\alpha$. Significantly enhanced robustness for lower $n$ values.

Failure shows a run time of 30 minutes (dark red)
We also tested with Richards’ equation, 1D test case (Celia, Bouloutas, Zarba, 1990)

\[
\frac{\partial \theta(h)}{\partial t} - \nabla \cdot (K h_r(h) \nabla (h + z)) = q, \quad \theta(h) = \frac{\theta_s - \theta_r}{(1 + (\alpha |h|)^p)^q} + \theta_r
\]

\(\theta_s = 0.368; \theta_r = 0.102\)
\(K = 0.00922 \text{ cm/s}\)
\(h(z,0) = -1000 \text{ cm}\)
\(h(0,t) = -75 \text{ cm}\)
\(h(60 \text{ cm}, t) = -1000 \text{ cm}\)
\(\Delta t = 60 \text{ min}, \Delta z = 2.5 \text{ cm}\)

New Mexico data

**AMG preconditioner (no special savings due to symmetric systems)**

Linear solves to 10^{-3} tol.
Run times (s) show benefit of acceleration

<table>
<thead>
<tr>
<th>1,024 zones</th>
<th>$\Delta t = 3600$ (s)</th>
<th>$\Delta t = 720$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solver / NL Tol</strong></td>
<td>$10^{-7}$</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td><strong>INI</strong></td>
<td>fail</td>
<td>fail</td>
</tr>
<tr>
<td><strong>MPI</strong></td>
<td>3.98</td>
<td>2.58</td>
</tr>
<tr>
<td><strong>MPI+AA(1)</strong></td>
<td>3.35</td>
<td>2.21</td>
</tr>
<tr>
<td><strong>MPI+AA(2)</strong></td>
<td>2.22</td>
<td>1.53</td>
</tr>
<tr>
<td><strong>MPI+AA(3)</strong></td>
<td>2.26</td>
<td>1.60</td>
</tr>
<tr>
<td><strong>MPI+AA(4)</strong></td>
<td>1.99</td>
<td>1.47</td>
</tr>
<tr>
<td><strong>MPI+AA(5)</strong></td>
<td>2.07</td>
<td>1.63</td>
</tr>
</tbody>
</table>

INI time ~ 75% of MPI+AA; exploit symmetry => MPI+AA competitive

Tight tolerances show more benefit of AA due to linear convergence of MPI
AA never added runtime to MPI
We investigated AA in the Parallel Dislocation Simulator (ParaDiS) which models dislocations

- Dislocation lines discretized in line segments ended by nodes
- Uses local and Fast Multipole methods for force calculations
- MPI and openMP parallel
- Adaptive grid
- Topology changes each step

**Nodal force**
\[ f_{i}^{\text{tot}} = f_{i}^{\text{self}} + f_{i}^{\text{seg}} + f_{i}^{\text{ext}} \]

**Nodal velocity**
\[ v_{i} = \frac{dx_{i}}{dt} = M_{ij}f_{j} \]

**Numerical integrator**
\[ r_{i}(t + \Delta t) = r_{i}(t) + v_{i}(t)\Delta t \]

**Topological changes**
- Insert a node, merge two nodes

Mobility law: material inputs
ParaDiS integrates a stiff nonlinear system in time

- Original integrator was trapezoid with a predictor:
  
  a) Explicit predictor: $x_{i,\text{new},1} = x_{i,\text{old}} + v_{i,\text{old}} \Delta t$
  
  b) Update: $v_{i,\text{new},1} = v_i(x_{i,\text{new},1})$
  
  c) Implicit corrector: $x_{i,\text{new}} = x_{i,\text{old}} + 0.5(v_{i,\text{new}} + v_{i,\text{old}}) \Delta t$

- Corrector: a nonlinear system for node position consistent w/ velocity

- Current method is fixed point: $x_{i,\text{new},k} = x_{i,\text{old}} + 0.5(v_{i,\text{new},k-1} + v_{i,\text{old}}) \Delta t$

- Convergence can be slow, leading to smaller time steps than desired

*We investigated whether the Anderson accelerated fixed point method would be effective for ParaDiS*
We tested accelerated fixed point and Newton’s method in ParaDiS

- KINSOL
  - Fixed point solver with acceleration
  - Newton-Krylov method with no preconditioning
    - Finite difference approximations to the matrix-vector product
    - Requires extra force calculations

- Newton implementation in ParaDiS may be made faster with full Jacobian formulation and preconditioner

We explored algorithm parameters on LLNL’s Cab system

- Ran MPI parallel on 16 cores
- Linux cluster system
- 1,296 nodes each with 16 cores (2 x Intel Xeon 8-core E5-2670) and 32 GB memory
- CPUs = 2.6 GHz
- Initiated with ~2,850 nodes
- Start at 3.3 \( \mu \)s; end at 4.4 \( \mu \)s
- Temp = 600K; Pres. = 0 GPa
- Body-centered crystal, 4.25 \( \mu \)m\(^3\)
- Strain rate = 1.0e3 /s in [0 0 1]
- Tolerance = 0.5

http://damask.mpie.de/pub/Documentation/BCC/BCC_crystal_structure.png
Results show benefit from acceleration

- Allowing native solver to take more iterations does not help
- AA with 7 iterations and 6 saved residuals gave the fastest result
- Newton-Krylov showed speedup but only for careful choices of the linear system tolerance
We tested a larger-scale problem of a material at high pressure & temperature

- Initially 1,094,620 dislocation nodes
- Start :1.286713e-05 µs, end:1.286848e-05 µs
- Temp = 600K; Pressure = 1 Gpa
- Strain = 1.0e4 /s
- Tolerance = 1.25; Domain: 1.0 µm³
- Run on 4,096 cores of LLNL Vulcan machine:
  - IBM Blue Gene/Q system
  - 5 Petaflop system: 24,576 nodes each w/ 16 cores and 16 GB memory
- Tested MPI + OpenMP parallelization with Trapezoid using fixed point iteration and AA + fixed point
Results show speedup from threading on 4,096 cores of Vulcan

ParaDiS with OpenMP threading gives speedup over MPI-only

KINSOL (6 Its, 5 Vecs) with OpenMP threading shows little benefit unless ParaDiS is also threaded, ~25% speedup over openMP threaded ParaDiS
Results on Sequoia demonstrate advantages and disadvantages of solver

- Test case with 55,456,000 segment nodes at initial time
- Start at 1.269053578987361e-09 µs, end at 1.36458e-09 µs
- Temp = 300K, pressure = 0 Gpa, Strain rate = 1.0 /s, domain = 1.0 µm³
- 262,144 cores of Sequoia, OpenMP threading with 4 threads per core
- Tolerance = 1.25, AA with I6, V5

12% speedup

Speedup limited by failure of code to handle larger time steps in the model.

With step capped, have run on more than 520,000 cores

<table>
<thead>
<tr>
<th></th>
<th>ParaDiS</th>
<th>KINSOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>1,774</td>
<td>1,566</td>
</tr>
<tr>
<td># Steps</td>
<td>551</td>
<td>394</td>
</tr>
<tr>
<td>Avg. Step</td>
<td>2.23e-11 s</td>
<td>3.24e-11 s</td>
</tr>
</tbody>
</table>

Always see a benefit with AA solver
The AA solver works extremely well for annealing simulations

- **Annealing** is when a metal is heated and then allowed to cool in order to remove internal stresses and increase toughness – dislocation density decreases with time

- AA gives a ~50% increase in simulated time over 12 wall clock hours
Annealing Simulation

- ~425,000 nodes to ~10,000 nodes
- 12 hour run on 256 cores of LLNL Ansel system
The AA algorithm still contains many decisions points with tradeoffs

- Can blocking the linear algebra (as in communication-avoiding GMRES) help?
- Can we apply restarting?
- How does AA perform in a GPU environment?
- Can we delay the start of acceleration?
- Does the conditioning of the QR problem affect results?
We investigated whether blocking the QR factorization could reduce costs

- Applied $m$ fixed point iterations, then Anderson acceleration with the block of $m$ vectors every $m$th iteration
  - Fewer reduction operations
  - Allows for blocked linear algebra
- Problem from Walker-Ni, 2011
  - Expectation-maximization (EM)

Blocking can have a modest impact on convergence!
QR cost savings will not justify algorithm slow down since MPI time is small overall

- GS: Gramm-Schmidt, from KINSOL – factorize one column at a time
- Blocked: factorize many columns at once using Tall Skinny QR (TSQR) code (Solomonik, et al.)

- Dummy problem, block size 16; 1,000 nodes on Vulcan
- Comm. time small relative to comp. for both
- TSQR always faster
- TSQR local work leverages optimized LAPACK library which KINSOL does not use

Cost savings is from computation not communication
Restarting slows convergence significantly

- As with GMRES can restart every mth iterate – reduces cost and memory
- Both restarting and sliding slow convergence

<table>
<thead>
<tr>
<th>Type</th>
<th>m</th>
<th>Iters</th>
<th>Vulcan Time(s)</th>
<th>Cab Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>30</td>
<td>30</td>
<td>0.07</td>
<td>0.03</td>
</tr>
<tr>
<td>Sliding</td>
<td>5</td>
<td>47</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>Sliding</td>
<td>10</td>
<td>33</td>
<td>0.07</td>
<td>0.03</td>
</tr>
<tr>
<td>Sliding</td>
<td>15</td>
<td>32</td>
<td>0.09</td>
<td>0.04</td>
</tr>
<tr>
<td>Restarted</td>
<td>5</td>
<td>71</td>
<td>0.06</td>
<td>0.02</td>
</tr>
<tr>
<td>Restarted</td>
<td>10</td>
<td>49</td>
<td>0.07</td>
<td>0.02</td>
</tr>
<tr>
<td>Restarted</td>
<td>15</td>
<td>42</td>
<td>0.09</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Restarting is cheaper per iteration but slows convergence more than sliding; gives small benefit on Cab

128² LaPlace; f = -10, 4x4 subdoms & 3 cell overlap AS (Walker-Ni) on 16 nodes
Condition Number Monitoring

- F matrix can become ill-conditioned. By $F = QR$, we can cheaply compute the condition number of $R$
  - If condition number($R$) > drop tolerance, drop the oldest column(s).

- Verification Problem
  - Chandrasekhar H-Equation displayed poor conditioning
  - Ice Sheet conditioning on order of $1e3$ (not needed)

- Problem dependent performance
  - Possible improvement
  - # spikes close to solution can delay convergence

*Slide courtesy of R. Pawlowski (Sandia)*
Delayed Start of Acceleration (Ice Sheet)

- In some cases, delaying the start of AA improved performance.
  - Poor initial solution
  - Strong global convergence properties
- Storing the history of a poor initial solution inhibits convergence of AA
- Delaying the start of acceleration can improve performance

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Damping} & N & \text{Lag} & \text{Iters} & \text{Time (s)} \\
\hline
1 & 10 & \text{No} & 126 & 225 \\
1 & 10 & 5 & 115 & 207 \\
0.4 & 10 & \text{No} & 114 & 204 \\
0.4 & 10 & 5 & 89 & 161 \\
\hline
\end{array}
\]

Slide courtesy of R. Pawlowski (Sandia)
We conducted initial tests on Surface, a representative single GPU machine

- GPU implementation:
  - Main loop of FP with AA runs on CPU
  - Vector operations run on GPU using CuBLAS from Nvidia for all vector operations
  - Data for algorithm resides in GPU RAM (except the R matrix)

- Expect this to be memory bound

- Peak bandwidth of GPU on Surface is ~5x greater than CPU bandwidth

- Each call to BLAS incurs 10 $\mu$s latency
  - Expect GPU to be slow for small problems and faster for larger ones (where the vectors are longer)

- Compared with a CPU-only implementation using standard BLAS and keeping data in CPU RAM
See definite benefit from use of GPU

Run times for CPU and GPU (fcn cost not timed)

- For vectors less than 10,000, CPU versions take less time than GPU version
- CPU version costs remain ~constant until vector lengths reach 100
- GPU version cost is constant until vector is 10,000 – length at which the work per vector dominates overhead per vector op
- Times approach linear with vector length
- When both CPU and GPU versions are in linear regime, we expect ratio between timings to be ~ratio of bandwidth
- Threading reduces runtime on CPU
- GPU gives more benefit on large problem
SUite of Nonlinear and DIfferential-ALgebraic Solvers

Suite of time integrators and nonlinear solvers

- ODE (CVODE, ARKode) and DAE (IDA) time integrators with forward and adjoint sensitivity capabilities (CVODES and IDAS)
- KINSOL: nonlinear solver including Newton-Krylov, Fixed Point and Picard, both with acceleration
- Written in C with interfaces to Fortran and Matlab
- Designed to be incorporated into existing codes
- Modular structure allows users to supply their own data structures underneath and new linear solvers to easily be added
- Supplied with serial, MPI, pThreads, and openMP optional vectors (interfaces for PetSc and hypre vectors coming soon; solver wraps soon after)
- Freely available, released BSD license ( > 4 K downloads /yr); included in RedHat Project distribution

https://computation.llnl.gov/casc/sundials/main.html
Conclusions

- Anderson acceleration can significantly speed up fixed point convergence

- With acceleration, fixed point methods may be a viable alternative to Newton methods but more testing needs to be done within the context of stiff systems

- Some theory for AA exists but it is still not fully developed
  - Can we develop a nonlocal theory?
  - Can we show anything beyond linear convergence?

- AA has been shown to be very effective in applications where derivatives are numerically difficult or unavailable, even as nonlinear solver under an implicit ODE integrator

- There are still many variations to explore generally and within context of specific applications
  - Restarting, sliding, blocking, condition number monitoring
  - API for general purpose software for use on GPUs
Acknowledgements

LLNL ParaDiS Team

D. Gardner (LLNL; ParaDiS)

J. Loffeld (LLNL; GPU)

U. Yang (LLNL; Implementation of AA)

A. Lott (Dwave; Richards eqn)

D. Reynolds (SMU; ParaDiS)

H. Walker (WPI; AA Implementation, Richards’ eqn.)
References

Acknowledgements

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy Office of Advanced Scientific Computing Research and National Nuclear Security Administration.

 computation.llnl.gov/casc/sundials