Accelerated Solvers for CFD
Co-Design of Hardware/Software for Predicting MAV Aerodynamics

Eric de Sturler, Virginia Tech – Mathematics
Email: sturler@vt.edu
Web: http://www.math.vt.edu/people/sturler

Co-design Workshop, Virginia Tech, February 7, 2014
People

- Faculty
  - Eric de Sturler, Chris Roy, Adrian Sandu, Danesh Tafti

- Postdocs
  - Xiao Xu
  - Amit Amritkar

- Graduate Students
  - Katarzyna Swirydowicz,
  - Arielle Grim McNally
Overview

- Current Efforts
- Long Term Plan
  - Parallel, Accelerated Solvers and Preconditioners for CFD Applications
- Quick Intro to Krylov Methods and Preconditioners
- Recycling Krylov Subspaces for GenIDLEST
  - (but results for an acoustics problem)
- GPU Preconditioners for SENSEI (LDC)
- Conclusions and Future Work
Current Efforts

- Integrating innovative iterative solvers and preconditioners in CFD codes
  - GENIDLEST (Tafti) – Recycling Solvers rGCROT (+ rBiCGStab)
  - SENSEI (Roy) – Fast Preconditioning (plus recycling)
- Faster Krylov-based time integrators (Sandu)
- Solvers that have better convergence, especially for sequences of problems – Krylov recycling
- GPU Acceleration, especially preconditioners
- New solvers with better opportunities to optimize multiple matvecs, precvecs, orthogonalizations in addition to faster convergence
- Updating preconditioners and further efficient variants of preconditioners
Long Term Plan (including CFD appl.s)

- Basic issue is “#iterations vs cost per iteration”
- All methods consist of different arrangements of matvecs, precvecs, dots, daxpy (and computing preconditioners)

Faster Preconditioners on GPUs

- Preconditioners with high level of fine grained parallelism and little data movement
  - Often not as effective (more iterations)
- Precludes ILU and preconditioners related to/based on it
  - Domain decomposition has local (approx) solve
- SAI very fast on GPUs (matvec); can improve convergence by multilevel extension
- FSAI promising too (matvec-like but more effective than SAI)
- Multigrid also promising if smoother fast
Long Term Plan

Solvers and Preconditioners

• Many problems require solving slowly changing systems for many parameters, many rhs, in nonlinear iteration or optimization, etc: **Improve convergence across systems**

• Recycling Krylov subspaces (select and reuse)

• Recycling preconditioners (update and reuse)

• Faster solver allows weaker preconditioner for cheaper iterations

• Solver variants that allow substantially faster implementations of main kernels by rearranging parts of algorithm (possibly over multiple iterations) – recycling solvers have advantageous over standard solvers

• Use model reduction to solve multiple systems much faster
Important Trends

- Simulations increasingly part of larger analysis, including design, uncertainty/reliability, inverse problems
- Simulations often involve parameters/parameter space
- Simulations involve wide ranges of scales and multi-physics. Drastically reduce effective number of unknowns: model reduction, parameterizing problems, adaptive meshing
- Move from generic models with idealized properties to realistic models individualized by parameterization (with uncertainty) – models first calibrated and then simulated
- Simulation also used to find parameters that cannot be measured directly
- New architectures for HPC require new algorithms, but significant support for solving many related problems
Krylov Methods Crash Course

Solve $Ax = b$, initial solution $x_0$, residual $r_0 = b - Ax_0$

Solve for error, $Ae_0 = r_0$; find update from search space

Generate space: $K_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$

Find update $z_m \in K_m(A, r_0)$ by minimizing

- error in suitable norm (typically special matrices only)
- residual in suitable norm (e.g., GMRES)

Implemented through orthogonal projection (in suitable inner product) – can be expensive.

Alternatively, give up on minimization and compute a cheap projection. Fast but possible robustness problems, e.g, BiCGStab.
Preconditioning

What if convergence slow? Precondition the system.

Replace $Ax = b$ by $P_1 A P_2 \tilde{x} = P_1 b$ and $x = P_2 \tilde{x}$

Where

1. Fast convergence for $P_1 A P_2$ and
2. Products with $P_1$ and $P_2$ is cheap
3. Computing $P_1$ and $P_2$ not too expensive

Often $A \approx LU$ (ILU) and use $L^{-1} A U^1$ or $U^{-1} L^{-1} A$

Forward-backward solve often slow on GPUs
Generally problematic for parallelism – do only for diagonal blocks (subdomain or grid line, etc)
Sparse Approx. Inverse Preconditioners

Preconditioners are matvec like (no solves)
Consider $Ax = b \rightarrow AM\tilde{x} = b$

(1) Sparse Approximate Inverse – SAI / SPAI
Pick sparsity pattern of $M$ and min. $\|AM - I\|_F$
Embarrassingly parallel, many tiny LS problem
Pattern often subset of $A^k$ (dynamic possible)

(2) Factorized Sparse Approximate Inverse – FSAI
Compute $A^{-1} \approx ZDW^T$ (biconjugation process)
with $Z, W$ sparse, uppertriangular, $D$ diagonal
Consider $Ax = b$ (or prec. system $PAx = Pb$)

Given $x_0$ and $r_0 = b - Ax_0$, find optimal update $z_m$ in

$$K^m(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{m-1}r_0\}:$$

$$\min_{z \in K^m(A, r_0)} \|b - A(x_0 + z)\|_2 \iff \min_{z \in K^m(A, r_0)} \|r_0 - Az\|_2$$

Let $K_m = \begin{bmatrix} r_0 & Ar_0 & A^2r_0 & \cdots & A^{m-1}r_0 \end{bmatrix}$, then $z = K_m \zeta$,

and we must solve the least squares problem

$$AK_m \zeta \approx r_0 \iff \begin{bmatrix} Ar_0 & A^2r_0 & \cdots & A^{m}r_0 \end{bmatrix} \zeta \approx r_0$$

Set up and solve in elegant, efficient, and stable way:

GCR – Eisenstat, Elman, and Schulz '83
GMRES – Saad and Schulz '86
Minimum Residual Solutions: GMRES

Solve $Ax = b$: Choose $x_0$; set $r_0 = b - Ax_0$;
$v_1 = r_0 / \|r_0\|_2$, $k = 0$.

while $\|r_k\|_2 \geq \varepsilon$ do

$k = k + 1; \tilde{v}_{k+1} = Av_k$;

for $j = 1 \ldots k$,

$h_{j,k} = v_j^* \tilde{v}_{k+1}; \tilde{v}_{k+1} = \tilde{v}_{k+1} - h_{j,k} v_j$;

end

$h_{k+1,k} = \|\tilde{v}_{k+1}\|_2; v_{k+1} = \tilde{v}_{k+1} / h_{k+1,k}$; ($AV_k = V_{k+1} H_k$)

Solve/Update LS $\min_\zeta \|\eta_1\| r_0 - H_k \zeta\|_2$

end

$x_k = x_0 + V_k \zeta$;

$r_k = r_0 - V_{k+1} H_k \zeta$ or $r_k = b - Ax_k$
Convergence restarted GMRES

Test problem on unit square: 202 x 202 grid points
Interior: \(-\nabla \cdot (\nabla u) = 0\)  
Boundary: \(u = 1\) for \(x = 0\) and \(y = 1\)  
\(u = 0\) elsewhere

\[ \log_{10} \| r \|_2 \]

### GMRES(m) 200 x 200 unknowns

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>72.888</td>
<td>587</td>
</tr>
<tr>
<td>100</td>
<td>40.256</td>
<td>1851</td>
</tr>
<tr>
<td>50</td>
<td>41.087</td>
<td>3043</td>
</tr>
<tr>
<td>20</td>
<td>63.604</td>
<td>6985</td>
</tr>
<tr>
<td>10</td>
<td>111.26</td>
<td>13761</td>
</tr>
<tr>
<td>5</td>
<td>199.42</td>
<td>27451</td>
</tr>
</tbody>
</table>
**BiCGStab**

$x_0$ is an initial guess; $r_0 = b - Ax_0$

Choose $\tilde{r}$, for example, $\tilde{r} = r_0$

for $i = 1, 2, ...$

\[
\rho_{i-1} = \tilde{r}^T r_{i-1}
\]

if $\rho_{i-1} = 0$ method fails

if $i = 1$

\[
p_i = r_{i-1}
\]

else

\[
\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})
\]

\[
p_i = r_{i-1} + \beta_{i-1}(p_{i-1} - \omega_{i-1}v_{i-1})
\]

endif

$v_i = Ap_i$;

\[
\alpha_i = \rho_{i-1}/\tilde{r}^T v_i
\]

$s = r_{i-1} - \alpha_i v_i$

check $\|s\|_2$, if small enough: $x_i = x_{i-1} + \alpha_i p_i$ and stop

$t = As$, $\omega_i = t^T s / t^T t$

\[
x_i = x_{i-1} + \alpha_i p_i + \omega_i s
\]

\[
r_i = s - \omega_i t
\]

check convergence; continue if necessary

for continuation necessary that $\omega_i \neq 0$

end
Solving Sequences of Linear Systems

- Many applications involve a sequence/group of systems with small or localized changes in space or structure
  - Time-dependent/time-like problems, nonlinear problems and optimization, adaptive discretizations
  - Systems depend (nonlinearly) on multiple parameters
  - Inverse problems, parameter estimation, Monte Carlo and MCMC methods, design, model reduction
  - Uncertainty quantification, reliability (with design)
- Application requires solution of hundreds to thousands of large, sparse, linear systems (millions for MCMC)
- Recycle previously computed results for faster solution
  - Update old solutions
  - Update and reuse search spaces – Krylov recycling
  - Update preconditioners
What to Recycle?

- Krylov methods build search space; solution by projection
- Building search space often dominates cost
- Initial convergence often poor, reasonable size search space needed, then superlinear convergence
- Get fast convergence rate and good initial guess immediately by recycling selected search spaces from previous systems
- How to select the right subspace to recycle?
  - Approximate invariant subspaces
  - Canonical angles between successive spaces
  - Subspace from previous solutions
How to Recycle?  

(GCRO, dS’95)

Solve $Ax = b$ with recycled subspace $\tilde{U}$ (for new $A$):

Compute $A\tilde{U} = \tilde{C}$, $CR = \tilde{C}$ (QR), $U = \tilde{U}R^{-1}$ (implicit)

Now $AU = C$ and $C^*C = I$

Set $r_0 = (I - CC^*) b$, $x_0 = UC^* b$, and $v_1 = r_0 / \|r_0\|$

Augmented Arnoldi: $AV_m = CB + V_{m+1} H_m$

Minimize:

$$\|b - A(x_0 + Uz + V_m y)\| = \|V_{m+1}(e_1 \|r_0\| - H_m y) - C(z + By)\|$$

Solve $H_m y \approx e_1 \|r_0\|$ and set $z = -By$ (optimal)

$x_m = x_0 + Uz + V_m y$ and $r_m = V_{m+1}(e_1 \|r_0\| - H_m y)$

Multiple matvecs/precvecs at once, orthogonalizations not in lock-step (GMRES), $m$ small more $U / C$ vectors
Example: Acoustics Problem

Real experiment

Part of the acoustic FE mesh

Acoustic FE/IFE mesh with solution

Details small model problem:
• 2nd order acoustic Finite Elements
• 6th order acoustic Infinite Elements
• ~10,000 degrees of freedom
• about 150 frequencies to be evaluated

On large realistic problem factor 6 to 10 speedup in time
Discretization

Variational form and resulting matrix components:

\[ \int_{\Omega} (\nabla p \cdot \nabla \tilde{q} - k^2 p \tilde{q}) \, dV - \int_{S_N} i \rho \omega v_n^o p \tilde{q} \, dS - \int_{S_R} i \kappa \alpha p \tilde{q} \, dS = 0 \]

\[ K_{ij} = \int_{\Omega^e} \nabla N_i \nabla N_j \, dV, \quad K_{ij} = \int_{\Omega^e} (\nabla \Phi_i + \nabla \Phi_j) \nabla \Phi_j \, dV, \]

\[ M_{ij} = \frac{1}{c^2} \int_{\Omega^e} N_i N_j \, dV, \quad M_{ij} = \frac{1}{c^2} \int_{\Omega^e} (1 - (\nabla \mu \nabla \mu)) \Phi_i \Phi_j D \, dV, \]

\[ C_{ij} = \rho \int_{S_R^e} \alpha N_i N_j \, dS, \quad C_{ij} = \frac{1}{c} \int_{\Omega^e} (\nabla \mu \nabla \Phi_j) D \Phi_i - (\nabla D \nabla \mu) \Phi_i \Phi_j - (\nabla \Phi_i \nabla \mu) D \Phi_j \, dV, \]

\[ f_i = -i \omega \int_{S_N^e} \rho v_n^o N_i \, dS. \]
Tire Rolling Noise Modeling

Equations interior and exterior acoustics simulation

\[ A(\omega) p = (K + i\omega C - \omega^2 M) p = f(\omega) \]

RHS depends on excitation frequency (from road texture)

Problem to be solved for \( \omega = 100, \ldots, 1500 \) and \( \Delta \omega = 10 \)

Must solve 140 linear systems (for small model problem)
For full problem up to 500 frequencies

Matrix components from interior domain are symmetric;
the components from exterior domain are nonsymmetric

In general, the exterior domain component is not a low rank update
Acoustics – rGCRot vs BiCGStab in # Matvecs

Comparison of Matrix-vector Products

- BiCGStab
- Recycling GCRot
Preconditioning in SENSEI (LDC)

- Standard ILU type preconditioners require mostly sequential forward/backward solve
  - Mitigated by Block ILU (#blocks - #threads)
- Parallelizes poorly on GPUs (not high level of fine-grained parallelism)
- In contrast sparse MatVec very fast
- Replace by other preconditioners that are more sparse MatVec like
  - Variants of Sparse Approximate Inverses (vary solver)
  - Next, combine with multilevel acceleration for convergence
    - Block Jacobi (small blocks)
- Solver GMRES, BiCGStab, recycling GCROT (almost)
LDC Problem

Incompr. Navier-Stokes with Artificial Viscosity

\[
\frac{1}{\beta^2} \frac{\partial p}{\partial t} + \rho \frac{\partial u}{\partial x} + \rho \frac{\partial v}{\partial y} = -C_1(u,v) \frac{\partial^4 p}{\partial x^4} - C_2(u,v) \frac{\partial^4 p}{\partial y^4}
\]

\[
\rho \frac{\partial u}{\partial t} + \rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = \mu \frac{\partial^2 u}{\partial x^2} + \mu \frac{\partial^2 u}{\partial y^2}
\]

\[
\rho \frac{\partial v}{\partial t} + \rho u \frac{\partial v}{\partial x} + \rho v \frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} = \mu \frac{\partial^2 v}{\partial x^2} + \mu \frac{\partial^2 v}{\partial y^2}
\]
**Comparison Iterative Solution in LDC**

Average iterative solution times for LDC step

- Solution time in ms
- Iterations in (preconditioned) matrix-vector products (BiCGStab – 2/iteration)

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40) ILUT Ms</th>
<th>GMRES(40) SAI Ms</th>
<th>Bicgstab ILUT Ms</th>
<th>Bicgstab SAI Ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ILUT MV</td>
<td>SAI MV</td>
<td>ILUT MV</td>
<td>SAI MV</td>
</tr>
<tr>
<td>101 (30.6K)</td>
<td>207 (21)</td>
<td>189 (99)</td>
<td>213 (23)</td>
<td>47.4 (116)</td>
</tr>
<tr>
<td>151 (68.4K)</td>
<td>392 (24)</td>
<td>186 (83)</td>
<td>423 (27)</td>
<td>48.3 (96)</td>
</tr>
<tr>
<td>251 (190K)</td>
<td>769 (23)</td>
<td>253 (73)</td>
<td>807 (25)</td>
<td>75.9 (80)</td>
</tr>
<tr>
<td>301 (272K)</td>
<td>1.18e3 (23)</td>
<td>293 (72)</td>
<td>1.27e3 (26)</td>
<td>93.3 (80)</td>
</tr>
</tbody>
</table>
# Speedups

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40) ILUT ms</th>
<th>GMRES(40) SAI Speedup</th>
<th>Bicgstab ILUT Speedup</th>
<th>Bicgstab SAI Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>101 (30.6K)</td>
<td>207</td>
<td>1.10 (189)</td>
<td>.972 (213)</td>
<td>4.37 (47.4)</td>
</tr>
<tr>
<td>151 (68.4K)</td>
<td>392</td>
<td>2.11 (186)</td>
<td>.927 (423)</td>
<td>8.12 (48.3)</td>
</tr>
<tr>
<td>251 (190K)</td>
<td>769</td>
<td>3.04 (253)</td>
<td>.953 (807)</td>
<td>10.1 (75.9)</td>
</tr>
<tr>
<td>301 (272K)</td>
<td>1.18e3</td>
<td>4.03 (293)</td>
<td>.929 (1.27e3)</td>
<td>12.6 (93.3)</td>
</tr>
</tbody>
</table>
Observations

- Although SAI less effective preconditioner (iterations), much faster runtimes due to higher flop/s GPU
- For ILUT versions, most time spent in preconditioner
- GMRES expensive in orthogonalizations
  - dot product + vector update (axpy)
- Improvement for GMRES limited by cost of orthogonalizations
- BiCGStab more effective in spite of further increase in iterations
- Results depend on (problem dependent)
  - Convergence vs cost per iteration
  - Relative costs of SMV, PV, orthogonalizations
## GMRES Runtime - Breakdown

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40)</th>
<th>PrecVec Mult</th>
<th>GS orthog.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ms</td>
<td>#PMV</td>
<td>ms</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>101</td>
<td>30.6K</td>
<td>207</td>
<td>20.6</td>
</tr>
<tr>
<td>151</td>
<td>68.4K</td>
<td>392</td>
<td>23.8</td>
</tr>
<tr>
<td>251</td>
<td>190K</td>
<td>769</td>
<td>23.2</td>
</tr>
<tr>
<td>301</td>
<td>272K</td>
<td>1.18e3</td>
<td>23.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40)</th>
<th>PrecVec Mult</th>
<th>GS orthog.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ms</td>
<td>#PMV</td>
<td>ms</td>
</tr>
<tr>
<td>101</td>
<td>30.6K</td>
<td>189</td>
<td>99.3</td>
</tr>
<tr>
<td>151</td>
<td>68.4K</td>
<td>186</td>
<td>83.5</td>
</tr>
<tr>
<td>251</td>
<td>190K</td>
<td>253</td>
<td>73.4</td>
</tr>
<tr>
<td>301</td>
<td>272K</td>
<td>293</td>
<td>71.9</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Good insight into cost issues of solvers/preconditioners
- Much better GPU performance for solver and preconditioner (BiCGSTAB / SAI) – factor 12
  - and working on other preconditioners (BILUT, …)
  - add multilevel correction to SAI for better convergence
- Recycling solvers implemented for CFD codes, start testing and combine with appropriate preconditioners
  - Explore additional optimization space for these solvers
- Explore solver variants that allow faster (fused) implementations of kernels (across iterations?)
- Extract lessons for Computational Dwarfs
$x_0$ is an initial guess; $r_0 = b - Ax_0$
Choose $\tilde{r}$ (for example, $\tilde{r} = r_0$)
for $i = 1, 2, \ldots$
\quad $\rho_{i-1} = \tilde{r}^T r_{i-1}$
\quad if $\rho_{i-1} = 0$ method fails
\quad if $i = 1$
\quad \quad $p_i = r_{i-1}$
\quad else
\quad \quad $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
\quad \quad $p_i = r_{i-1} + \beta_{i-1}(p_{i-1} - \omega_{i-1}v_{i-1})$;
\quad endif
Solve $\hat{p}$ from $K\hat{p} = p_i$
$v_i = A\hat{p}$
$\alpha_i = \rho_{i-1}/\tilde{r}^T v_i$
$s = r_{i-1} - \alpha_i v_i$
if $\|s\|$ small enough then
\quad $x_i = x_{i-1} + \alpha_i \hat{p}$, quit
Solve $\hat{s}$ from $K\hat{s} = s$
$t = A\hat{s}$
$\omega_i = t^T s / t^T t$
$x_i = x_{i-1} + \alpha_i \hat{p} + \omega_i \hat{s}$
if $x_i$ is accurate enough then quit
$r_i = s - \omega_i t$
for continuation it is necessary that $\omega_i \neq 0$
end