Interactions

- Implicit SENSEI-LDC code (with Roy group)
  - GPU-based solver and preconditioner development for the SENSEI package

- Recycling solver in GENIDLEST code (with Tafti group)
  - Krylov subspace recycling based solvers in GENIDLEST, including new hybrid methods (starting simulation with rGCROT, then switching to rBiCGStab with rGCROT recycle space)
2. Publications

- All in preparation:
Plans for Next Year (and beyond)

- Finish 3 papers mentioned on previous slide
- GPU testing/tuning of multilevel SAI preconditioners, AINV preconditioners, and develop other variations
- Develop preconditioner updates in SENSEI/GENIDLEST (recycle preconditioners from one system to next)
- Collaborate with CS group in library development for key components in solvers and preconditioners on GPUs
- Port recycling solvers rGCROR, rGCRODR, rBiCGSTAB to GPUs
- Collaborate with Hong and Edward groups on preconditioners
- Parameter analysis for rBiCGSTAB for further development
- Explore optimizations on GPU of computational kernels for recycling solvers
- Analyze solver and preconditioner components with respect to the computational dwarves
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

NCSU/VT Meeting, NC State, July 23, 2014
People

- Faculty
  - Eric de Sturler, Chris Roy, Adrian Sandu, Danesh Tafti
- Postdocs
  - Amit Amritkar, Xiao Xu (until May 2014)
- Graduate Students
  - Katarzyna Swirydowicz, Arielle Grim McNally, Joe Derlaga

- SENSEI Solvers, GPU Preconditioners – Kasia, Xiao, Joe, Chris, EdS
- GENIDLEST Recycling Solvers – Amit, Kasia, Danesh, EdS
- Informal collaboration with Wu Feng, Tom Scogland, …
Overview

- **Goal:** Develop Fast Parallel Iterative Solvers and Preconditioners for CFD Applications
  - **Short Term:** GPU Acceleration
  - **Longer Term:** Add Coarse Grain Parallelism (DD)

- Quick Intro to Krylov Methods and Preconditioners and Current Trends
- Preconditioners for SENSEI (LDC) on CPUs
- Recycling Krylov Subspaces for GenIDLEST
- Updating preconditioners
- Conclusions and Future Work
Iterative Solvers and Preconditioners for CFD

- Solution of linear systems often dominates run time
- All Krylov subspace solvers have same components
  - matvecs, dot products, vector updates (axpy)
  - preconditioner computation, precvecs
- Balance number of iterations vs cost per iteration
- Solve many systems:
  - Time steps
  - Nonlinear iteration
  - Parameter studies
  - … (and all of these combined)
- Matrix sometimes fixed, sometimes changes slowly
- Exploit for faster solution time
Important Trends

- Simulations increasingly part of larger analysis, including design, uncertainty/reliability, inverse problems
  - Many solutions/simulations of slowly varying problems
  - Time-dependent, nonlinear, or inverse problems, parameter dependence, uncertainty
- Want to solve problems faster: faster solvers
  - Make each iteration cheaper
  - Reduce number of iterations (across all solutions)
- New architectures for HPC require new algorithms
- Adapt solvers to new architectures (GPUs, multicore)
  - Focus: sparse matvecs, preconditioners, inner products
  - On GPUs sparse matvec and precvec bottleneck
  - Exascale machines: inner products
- Opportunities in solving many related problems
Consider $Ax = b$ (or prec. system $PAx = Pb$)

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

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

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

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

$$AK_m \zeta \approx r_0 \quad \Leftrightarrow \quad \left[ Ar_0 \ A^2r_0 \ \cdots \ A^{m}r_0 \right] \zeta \approx r_0$$

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 $r_k = b - Ax_k$
BiCGStab

$x_0$ is an initial guess; $r_0 = b - Ax_0$
Choose $\tilde{r}$, for example, $\tilde{r} = r_0$
for $i = 1, 2, ...$
\[ \rho_{i-1} = \tilde{r}^T r_{i-1} \]
\text{if } \rho_{i-1} = 0 \text{ method fails}
\text{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
\[ v_i = Ap_i; \]
\[ \alpha_i = \rho_{i-1}/\tilde{r}^Tv_i \]
\[ s = r_{i-1} - \alpha_i v_i \]
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
Preconditioning

What if convergence slow? Precondition the system.

Replace $Ax = b$ by $P_1AP_2\tilde{x} = P_1b$ and $x = P_2\tilde{x}$

Where

1. Fast convergence for $P_1AP_2$ 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 $L^{-1}AU^1$ 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. \(\left\|AM - I\right\|_F\)
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
Solvers for Nonsymmetric Matrices

- Two types of methods
  - **Optimal** – minimum number of iterations (e.g., GMRES)
    - Full orthogonalization of search space
    - $m$ iterations: $\sim \frac{1}{2} m^2$ orthogonalizations ($2m^2N$), $m$ matvecs+precvecs (some further vector operations)
    - $O(mN)$ storage (grows with iteration count)
    - Typically restarted, increases iterations (better strat.s)
  - **Nonoptimal**, short recurrences (e.g., BiCGStab)
    - More iterations, possible breakdown (rare)
    - Few (fixed $k$) orthogonalizations per iteration
    - $m$ iterations: $km$ orthogonalizations ($4kmN$), $m$ matvecs+precvecs (some further vector operations)
    - no restarts, occasional accuracy problems
Faster Preconditioners on GPUs

- Efficiency requires preconditioners with high level of fine grained parallelism and little data movement
  - Often not as effective (more iterations)
- Precludes ILU and related preconditioners (slow)
  - Domain decomposition has local (approx) solve
- SAI very fast on GPUs (matvec), but more iterations
  - Improve convergence by multilevel extension
    - all matvec type operations
    - not needed for all problems
- FSAI/AINV promising too (but needs GPU testing)
  - LU like but sparse approximations to inverse factors
- Multigrid also promising if smoother is fast (matvec-like)
LDC Problem

Incompr. Navier-Stokes with Artificial Viscosity

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

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

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

- Matrix format determines performance
  - CSR, DIA, JAD, etc.
- JAgged Diagonal (JAD) most efficient for LDC
Average Iterations & Solution Times for LDC Step

- Faster precvec reduces run times, even with increased iterations (3x to 5x)
- Full orthogonalization dominates for GMRES/SAI
- With SAI use cheaper solver: BiCGStab
- More iterations but further run time improvement

Proper combination solver/prec. matters:
  - BiCGStab/ILUT slower than GMRES/ILUT

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40)</th>
<th>GMRES(40)</th>
<th>Bicgstab</th>
<th>Bicgstab</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ILUT</td>
<td>SAI</td>
<td>ILUT</td>
<td>SAI</td>
</tr>
<tr>
<td></td>
<td>ms</td>
<td>ms</td>
<td>ms</td>
<td>ms</td>
</tr>
<tr>
<td></td>
<td>MV</td>
<td>MV</td>
<td>MV</td>
<td>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>

Iterations in (preconditioned) matrix-vector products
<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>
## GMRES Runtime - Breakdown

### GMRES with ILUT preconditioner

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40) ms</th>
<th>#PMV</th>
<th>PrecVec Mult ms</th>
<th>%</th>
<th>GS orthog. ms</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>30.6K</td>
<td>207</td>
<td>183</td>
<td>89</td>
<td>17.9</td>
<td>7</td>
</tr>
<tr>
<td>151</td>
<td>68.4K</td>
<td>392</td>
<td>360</td>
<td>92</td>
<td>25.3</td>
<td>5</td>
</tr>
<tr>
<td>251</td>
<td>190K</td>
<td>769</td>
<td>721</td>
<td>94</td>
<td>36.4</td>
<td>5</td>
</tr>
<tr>
<td>301</td>
<td>272K</td>
<td>1.18e3</td>
<td>1.12e3</td>
<td>95</td>
<td>43.5</td>
<td>4</td>
</tr>
</tbody>
</table>

### GMRES with SAI preconditioner

<table>
<thead>
<tr>
<th>Problem</th>
<th>GMRES(40) ms</th>
<th>#PMV</th>
<th>PrecVec Mult ms</th>
<th>%</th>
<th>GS orthog. ms</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>30.6K</td>
<td>189</td>
<td>11.1</td>
<td>6</td>
<td>157</td>
<td>83</td>
</tr>
<tr>
<td>151</td>
<td>68.4K</td>
<td>186</td>
<td>12.7</td>
<td>7</td>
<td>160</td>
<td>85</td>
</tr>
<tr>
<td>251</td>
<td>190K</td>
<td>253</td>
<td>21.5</td>
<td>8</td>
<td>200</td>
<td>79</td>
</tr>
<tr>
<td>301</td>
<td>272K</td>
<td>293</td>
<td>26.4</td>
<td>9</td>
<td>229</td>
<td>78</td>
</tr>
</tbody>
</table>
Convergence Results with Multilevel SAI

Efficient implementation on GPUs is in progress
Solving Sequences of Linear Systems

- **GENIDLEST/SENSEI** solve sequence of slowly changing linear systems (sometimes matrix constant)
- Common feature of many applications
- Application requires solution of hundreds to thousands of large, sparse, linear systems (millions for MCMC)
- Improve convergence across systems
- Recycle previously computed results for faster solution
  - Update old solutions (standard)
  - Update & reuse search spaces: Krylov subspace recycling
  - Update preconditioners (Arielle)
- Faster kernels by rearranging parts of algorithm, possibly over multiple iterations – recycling solvers have advantages over standard solvers (future work)
Krylov Subspace Recycling

- Krylov methods build search space; pick solution by projection
- Building search space often dominates cost
- Initial convergence often poor, reasonable dimension search space needed, then superlinear convergence
- Get fast convergence rate and good initial guess immediately by recycling selected search spaces from previous systems
- Recycling reduces iterations, but overhead per iteration
- Several ways to select the right subspace to recycle
  - Approximate invariant subspaces, canonical angles between successive spaces, subspace from previous solutions, …
rGCROT Performance

- CPU performance
  - Turbulent channel flow case (64x64x64)
  - Intel i5-2400 CPU @ 3.1 GHz
  - Solution of pressure Poisson equation

<table>
<thead>
<tr>
<th>10 time steps</th>
<th>BiCGSTAB</th>
<th>rGCROT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time (s)</td>
<td>44</td>
<td>157.5</td>
</tr>
<tr>
<td>Average number of iterations</td>
<td>62 iterations</td>
<td>8 iterations</td>
</tr>
</tbody>
</table>

- Reuse the Krylov subspace as the matrix doesn’t change
- Use BiCGSTAB like solver
  - Faster
Hybrid approach

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiCGSTAB</td>
<td>Faster</td>
<td>Irregular convergence for stiff problems</td>
</tr>
</tbody>
</table>
| rGCROT        | • Reuse of selected Krylov subspace
• Monotonic residual decrease | Higher cost per iteration due to orthogonalizations         |

- Recycling BiCGSTAB (rBiCGSTAB) after rGCROT
  - Best of both worlds
  - Building the outer subspace initially
    - Using rGCROT, then switch to rBiCGSTAB
    - Other approaches possible (rGCRODR or First time step with BiCGSTAB)
Flow through porous media

- Immersed Boundary Methods with stochastic reconstruction for porous media
- Background mesh of 2.56 million cells (100x800x32)
- 10 time steps
  - relative tolerance of $1 \times 10^{-10}$
- 16 CPU cores and 46 GB memory
Convergence for different solvers (10th time step)

- **New run**
  - Point Jacobi prec
  - Recycling is effective

- **Restarted run**
  - Point Jacobi prec
  - Reduced recycling effects
Time to solution

- rBiCGSTAB converges fastest in time
- Further potential for performance improvement
  - Size of the outer vector space
  - Algorithm for building the outer vector space
  - Improving efficiency of kernels

<table>
<thead>
<tr>
<th>Solver</th>
<th>Iterations</th>
<th>Time to solution for 10th time steps (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiCGStab</td>
<td>1000</td>
<td>417.6</td>
</tr>
<tr>
<td>rBiCGStab</td>
<td>228</td>
<td>208.5</td>
</tr>
<tr>
<td>GMRES(25)</td>
<td>73</td>
<td>631.2</td>
</tr>
<tr>
<td>rGCR(10)</td>
<td>41</td>
<td>213.3</td>
</tr>
</tbody>
</table>
Effect of Preconditioner

- **Preconditioners**
  - Point Jacobi smoothening
  - Symmetric Successive Over-Relaxation

- **Convergence**
  - Faster with SSOR

- **Why use Point Jacobi?**

<table>
<thead>
<tr>
<th>On GPU</th>
<th>BiCGSTAB (Jacobi)</th>
<th>BiCGSTAB (SSOR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>0.16</td>
<td>0.26</td>
</tr>
</tbody>
</table>
## Effect of number of time steps with rGCROT

Time steps with rGCROT before switching to rBiCGStab

<table>
<thead>
<tr>
<th>Time steps with rGCROT</th>
<th>Solution time for 10(^{th}) time step (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unstable</td>
</tr>
<tr>
<td>2</td>
<td>232.2</td>
</tr>
<tr>
<td>3</td>
<td>231.7</td>
</tr>
<tr>
<td>4</td>
<td>232.2</td>
</tr>
<tr>
<td>5</td>
<td>unstable</td>
</tr>
<tr>
<td>6</td>
<td>240.3</td>
</tr>
<tr>
<td>7</td>
<td>235.9</td>
</tr>
<tr>
<td>8</td>
<td>250.7</td>
</tr>
<tr>
<td>9</td>
<td>259.4</td>
</tr>
</tbody>
</table>
Updating Preconditioners – Key Idea

Sequence of systems $A_1 x_1 = b_1, A_2 x_2 = b_2, ...$ (with small changes)

Very good preconditioner, $P_1$, for $A_1$: fast convergence for $A_1 P_1$

Want cheap updates for $P_1$ such that $A_1 P_1 = A_2 P_2 = A_3 P_3 = ...$

Fast solution for all systems, low cost for updating preconditioners

Flexible: map $A_k$ to $A_1$: $\min \| A_k M_k - A_1 \| \rightarrow A_k M_k P_1 \approx A_1 P_1$

So, preconditioner $P_k = M_k P_1$ gives fast convergence for $A_k$

SAM (sparse approximate map) for fixed simple nonzero pattern
- SAI-like (sparse approximate inverse)
- cheap to compute
- easy to update for individual columns (even cheaper)
- update independent of type of preconditioner $P$
Results for model reduction problem

Thruster 80K, 80 nonzeros/row
Note ILUTP compiled vs computing SAI m-file!

ILUTP each system (4 systems)

<table>
<thead>
<tr>
<th>droptol</th>
<th>Total (s)</th>
<th>ILUTP</th>
<th>GMRES</th>
<th># its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.e-6</td>
<td>4083</td>
<td>1011</td>
<td>5.81</td>
<td>12, 12, 12, 11</td>
</tr>
<tr>
<td>1.e-4</td>
<td>733.4</td>
<td>163.4</td>
<td>19.62</td>
<td>176, 211, 179, 230</td>
</tr>
</tbody>
</table>

ILUTP & SAI update (4 systems), nz(P) = nz(A0)

<table>
<thead>
<tr>
<th>droptol</th>
<th>Total (s)</th>
<th>ILUTP</th>
<th>SAI upd</th>
<th>GMRES</th>
<th># its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.e-6</td>
<td>1301</td>
<td>1039</td>
<td>77.4</td>
<td>5.11</td>
<td>12, 12, 11, 13</td>
</tr>
<tr>
<td>1.e-4</td>
<td>471.7</td>
<td>165</td>
<td>74.5</td>
<td>18.9</td>
<td>176, 181, 178, 170</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Good insight cost issues of solvers/preconditioners on GPUs
  - Analyzed range of preconditioners
- Much better GPU performance for solver and preconditioner (BiCGSTAB / SAI) – factor 12
- Multilevel correction to SAI yields better convergence
- Good results for GENIDLEST with recycling
  - Switching methods new, efficient approach

- Move rBiCGSTAB and other recycling solvers to GPU
- Parameter analysis with rBiCGSTAB
- GPU testing/tuning of multilevel SAI preconditioner
- Test AINV preconditioner, typically more effective than SAI, also multiplicative. Needs testing and GPU tuning.
- Explore optimizations for recycling solvers
- Preconditioner updates in SENSEI/GENIDLEST
Nice Fluid Flow Picture