Interactions

- Implicit SENSEI-LDC code (with Roy group)
  - GPU-based solver and preconditioner development for the SENSEI package
  - Effective Solvers and Preconditioners on GPUs for CFD Applications, in preparation (Parallel Computing), K. Swirydowicz, E. de Sturler, X. Xu, and C.J. Roy.
  - Effective Parallel Preconditioners for CFD Applications, SIAM Annual Mtg 2014, K. Swirydowicz, E. de Sturler, X. Xu, and C.J. Roy.

- 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)
  - Recycling Krylov subspaces for CFD Applications, in preparation (Computer Methods in Applied Mechanics and Engineering), A. Amritkar, E. de Sturler, K. Swirydowicz, D. Tafti, K. Ahuja.

2. Publications

- All in preparation:
  - Effective Solvers and Preconditioners on GPUs for CFD Applications, in preparation (Parallel Computing), K. Swirydowicz, E. de Sturler, X. Xu, and C.J. Roy.
  - Recycling Krylov subspaces for CFD Applications, in preparation (Computer Methods in Applied Mechanics and Engineering), A. Amritkar, E. de Sturler, K. Swirydowicz, D. Tafti, K. Ahuja.
  - Preconditioning Parameterized Linear Systems, in preparation (SIAM J. Scientific Computing), A.K. Grim McNally, M. Li, E. de Sturler, S. Gugercin.

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 rGCROT, 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 GPUs
- 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

Krylov Methods Crash Course

Consider $Ax = b$ (or prec. system $PAx = Pb$)
Given x_0 and $r_0 = b - Ax_0$, find optimal update z_m in

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

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

Let $K_m = \left[r_0 \; Ar_0 \; A^2 r_0 \cdots A^{m-1} r_0 \right]$, then $z = K_m \zeta$,

and we must solve the least squares problem

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

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

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

  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

  $h_{k+1,k} = \|\tilde{v}_{k+1}\|_2$; $v_{k+1} = \tilde{v}_{k+1} / h_{k+1,k}$; $\quad (A V_k = V_{k+1} \underline{H}_k)$

  Solve/Update LS $\min_\zeta \left\| \eta_1 \|r_0\|_2 - \underline{H}_k \zeta \right\|_2$

end

$x_k = x_0 + V_k \zeta$;

$r_k = r_0 - V_{k+1} \underline{H}_k \zeta$ or $r_k = b - A x_k$

BiCGStab

van der Vorst '92

x_0 is an initial guess; $r_0 = b - Ax_0$
Choose $\widetilde{r}$, for example, $\widehat{r} = r_0$
for $i = 1, 2, ...$
$\quad \rho_{i-1} = \widetilde{r}^T r_{i-1}$
$\quad$ **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**
$\quad v_i = Ap_i;$
$\quad \alpha_i = \rho_{i-1}/\widetilde{r}^T v_i$
$\quad s = r_{i-1} - \alpha_i v_i$
$\quad$ check $\|s\|_2$, if small enough: $x_i = x_{i-1} + \alpha_i p_i$ and stop
$\quad t = As$, $\omega_i = t^T s / t^T t$
$\quad x_i = x_{i-1} + \alpha_i p_i + \omega_i s$
$\quad r_i = s - \omega_i t$
$\quad$ check convergence; continue if necessary
$\quad$ for continuation necessary that $\omega_i \neq 0$
end

from: Iterative Solvers for Large Linear Systems, H.A. van der Vorst Cambridge University Press

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. $\left\| AM - I \right\|_F$

Embarrasingly 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: $\sim {}^{1}/_{2}\, m^2$ 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)

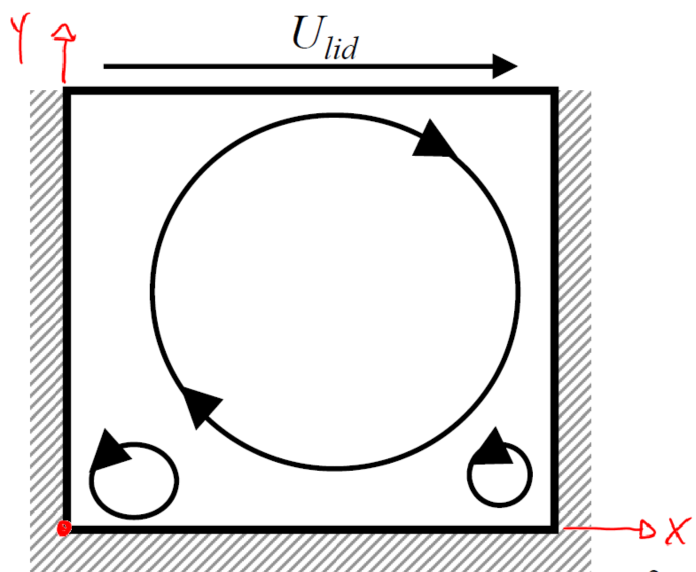
LDC Problem

Incompr. Navier-Stokes with Artificial Viscosity

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

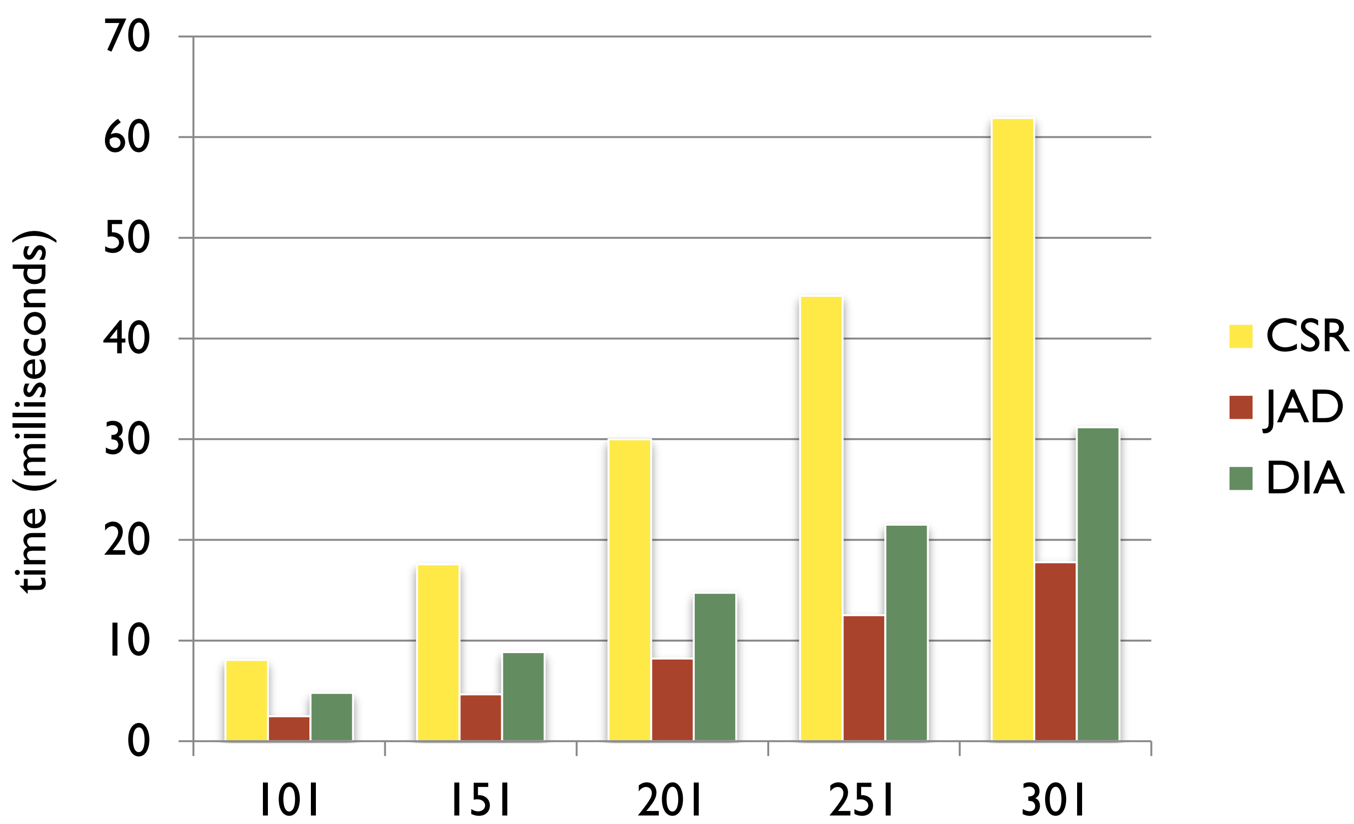
$$\rho\frac{\partial u}{\partial t} + \rho u\frac{\partial u}{\partial x} + \rho v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = \mu\frac{\partial^2 u}{\partial x^2} + \mu\frac{\partial^2 u}{\partial y^2}$$

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



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

Problem	GMRES(40) ILUT ms	MV	GMRES(40) SAI ms	MV	Bicgstab ILUT ms	MV	Bicgstab SAI ms	MV
101 (30.6K)	207	(21)	189	(99)	213	(23)	47.4	(116)
151 (68.4K)	392	(24)	186	(83)	423	(27)	48.3	(96)
251 (190K)	769	(23)	253	(73)	807	(25)	75.9	(80)
301 (272K)	1.18e3	(23)	293	(72)	1.27e3	(26)	93.3	(80)

Iterations in (preconditioned) matrix-vector products

Speedups

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

GMRES Runtime - Breakdown

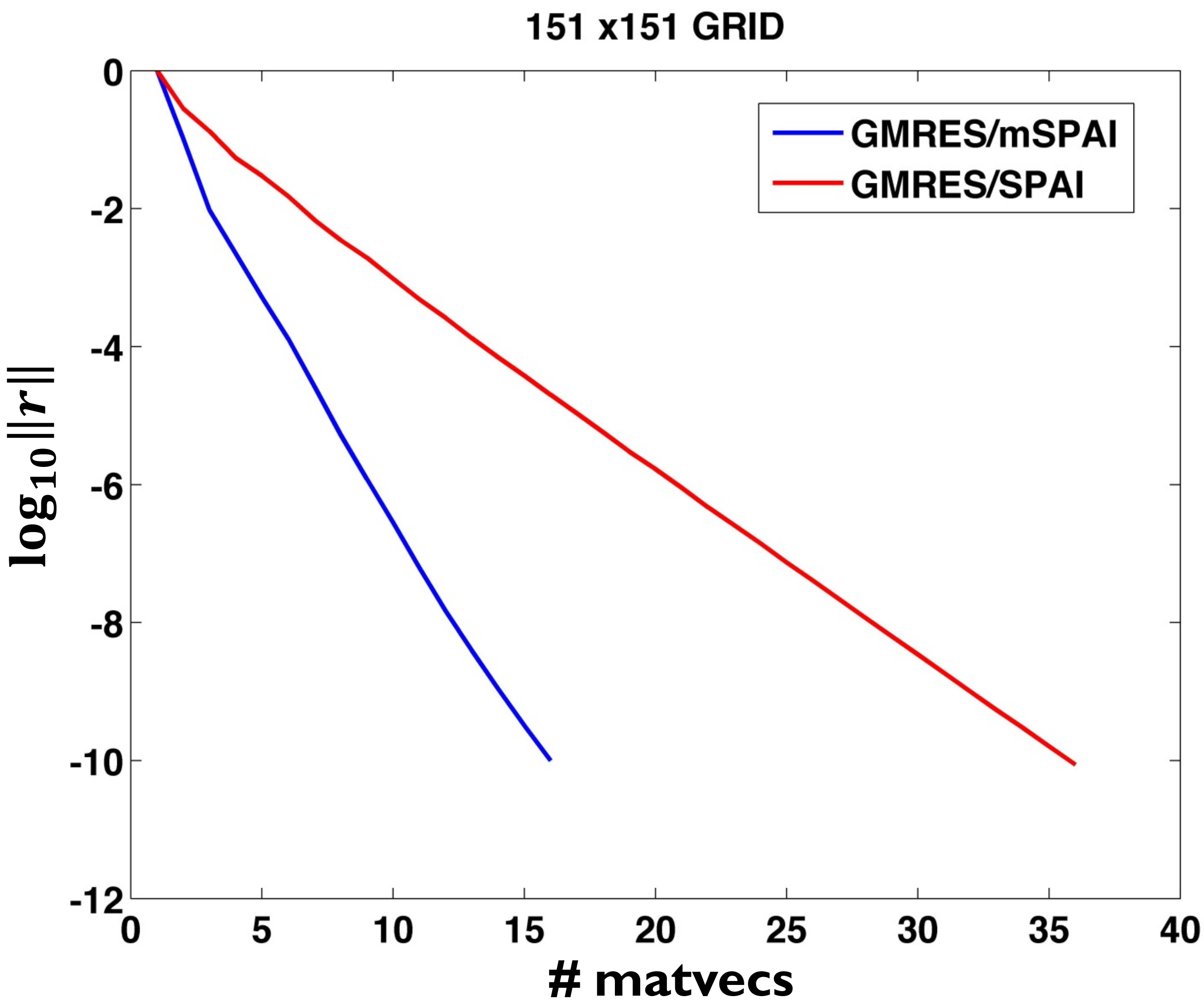
GMRES with ILUT preconditioner

Problem		GMRES(40)		PrecVec Mult		GS orthog.	
		ms	#PMV	ms	%	ms	%
101	30.6K	207	20.6	183	89	17.9	7
151	68.4K	392	23.8	360	92	25.3	5
251	190K	769	23.2	721	94	36.4	5
301	272K	1.18e3	23.4	1.12e3	95	43.5	4

GMRES with SAI preconditioner

Problem		GMRES(40)		PrecVec Mult		GS orthog.	
		ms	#PMV	ms	%	ms	%
101	30.6K	189	99.3	11.1	6	157	83
151	68.4K	186	83.5	12.7	7	160	85
251	190K	253	73.4	21.5	8	200	79
301	272K	293	71.9	26.4	9	229	78

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

10 time steps	BiCGSTAB	rGCROT
Total time (s)	44	157.5
Average number of iterations	62 iterations	8 iterations

- Reuse the Krylov subspace as the matrix doesn't change
- Use BiCGSTAB like solver
  - Faster

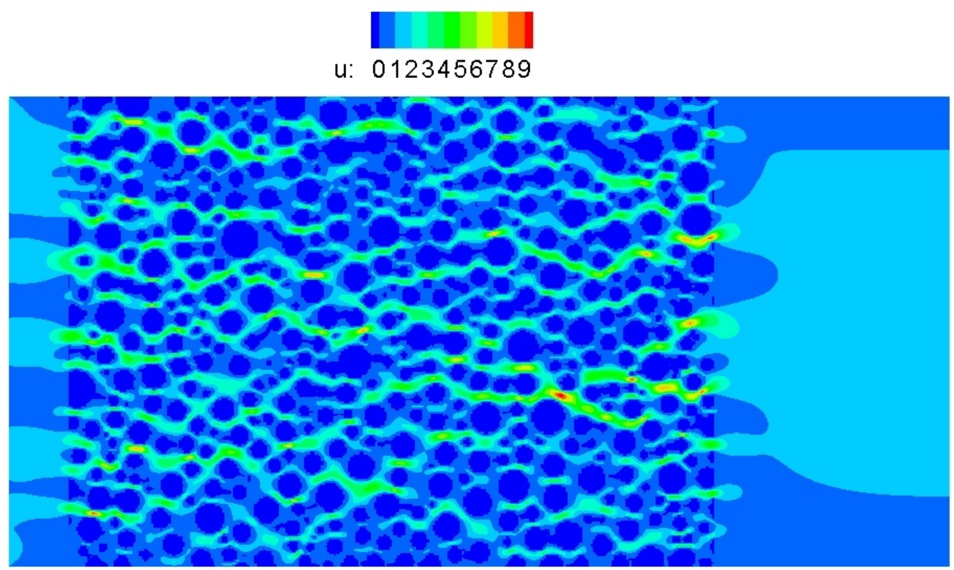
Hybrid approach

	Pros	Cons
BiCGSTAB	Faster	Irregular convergence for stiff problems
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)

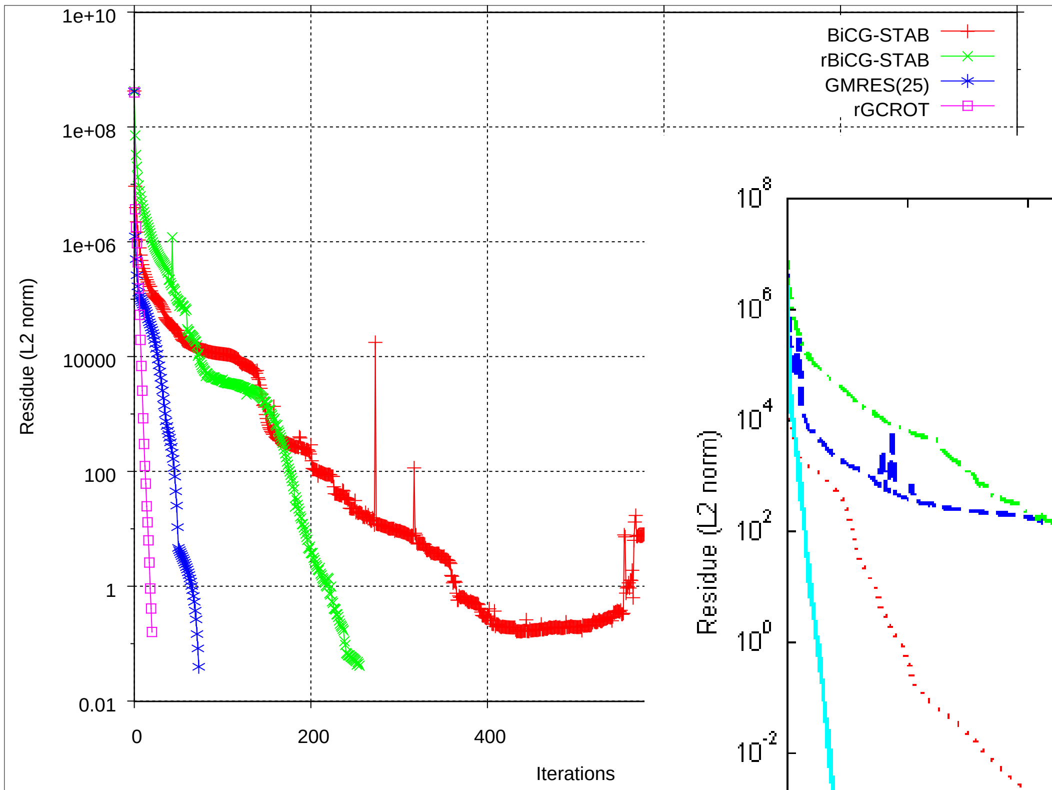
Flow though porous media

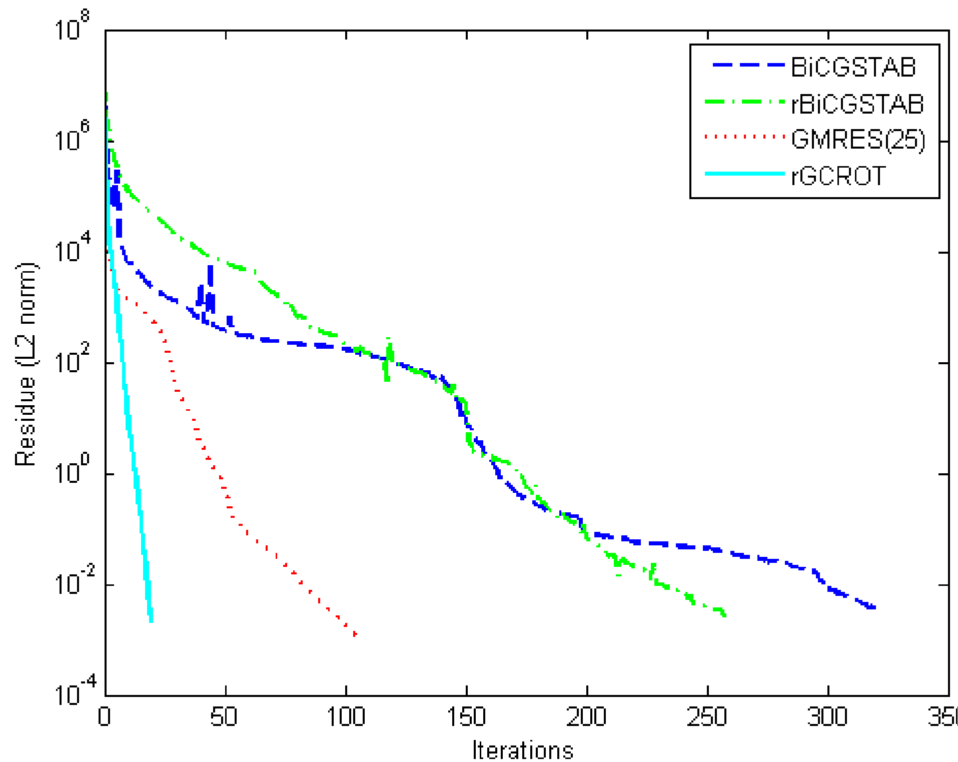
- Immersed Boundary Methods with stochastic reconstruction for porous media
- Background mesh of 2.56 million cells (100x800x32)
- 10 time steps
  - relative tolerance of 1×10^{-10}
- 16 CPU cores and 46 GB memory



Convergence for different solvers (10th time step)

- New run
  - Point Jacobi prec
  - Recycling is effective
- Restarted run
  - Point Jacobi prec
  - Reduced recycling effects





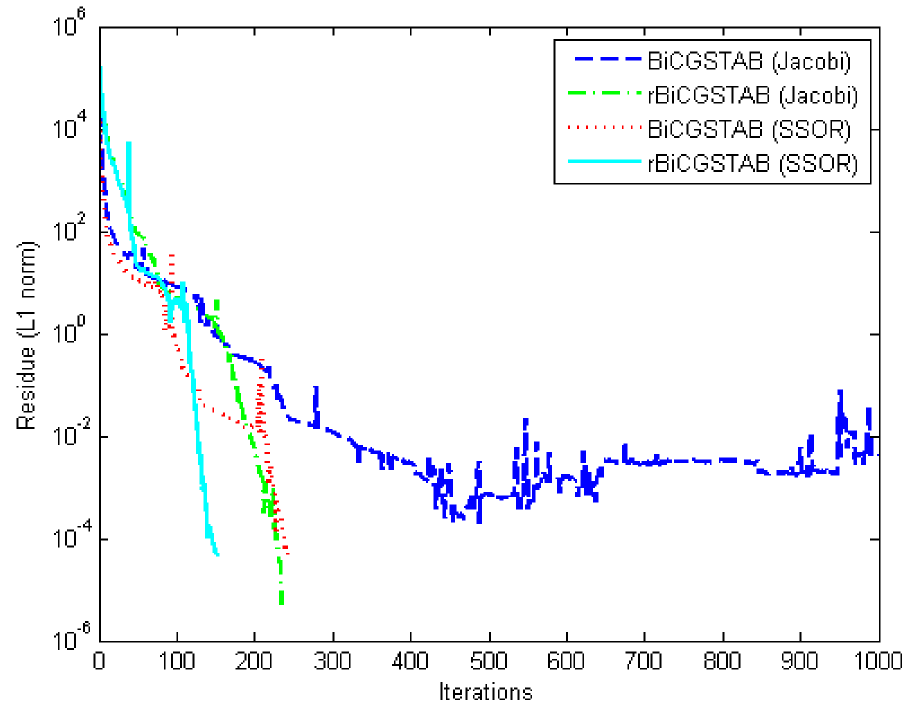
Time to solution

- 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

Solver	Iterations	Time to solution for 10th time steps (s)
BiCGStab	1000	417.6
rBiCGStab	**228**	**208.5**
GMRES(25)	73	631.2
rGCROT(10)	41	213.3

Effect of Preconditioner

- Preconditioners
  - Point Jacobi smoothening
  - Symmetric Successive Over-Relaxation
- Convergence
  - Faster with SSOR
- Why use Point Jacobi?



On GPU	BiCGSTAB (Jacobi)	BiCGSTAB (SSOR)
Time (s)	0.16	0.26

Effect of number of time steps with rGCROT

Time steps with rGCROT before switching to rBiCGStab

Time steps with rGCROT	Solution time for 10th time step (s)
1	unstable
2	232.2
3	**231.7**
4	232.2
5	unstable
6	240.3
7	235.9
8	250.7
9	259.4

Updating Preconditioners – Key Idea

Sequence of systems $A_1x_1 = b_1$, $A_2x_2 = b_2$, … (with small changes)
Very good preconditioner, P_1, for A_1: fast convergence for A_1P_1

Want cheap updates for P_1 such that $A_1P_1 = A_2P_2 = A_3P_3 = \cdots$
Fast solution for all systems, low cost for updating preconditioners

Flexible: map A_k to A_1: $\min \left\| A_kM_k - A_1 \right\| \to A_kM_kP_1 \approx A_1P_1$

So, preconditioner $P_k = M_kP_1$ 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)

droptol	Total (s)	ILUTP	GMRES	# its
1.e-6	4083	1011	5.81	12, 12, 12, 11
1.e-4	733.4	163.4	19.62	176, 211, 179, 230

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

droptol	Total (s)	ILUTP	SAI upd	GMRES	# its
1.e-6	1301	1039	77.4	5.11	12, 12, 11, 13
1.e-4	471.7	165	74.5	18.9	176, 181, 178, 170

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