The new challenges to Krylov subspace methods

Yousef Saad

Department of Computer Science and Engineering

University of Minnesota

SIAM Applied Linear Algebra

Valencia, June 18–22, 2012

Introduction

- Krylov subspace methods offer a good alternative to direct solution methods - especially for 3D problems
- Compromise between performance and robustness
- Current challenges:
  - Highly indefinite systems [Helmholtz, Maxwell, ...]
  - Highly ill-conditioned systems
  - Problems with extremely irregular structure
  - Recent: impact of new architectures [many core, GPUs]

Introduction (cont.)

Two distinct issues:

- Performance degradation due to ‘irregular sparsity’
- Performance degradation due to problem size / GPU memory limitation

Observation:
➤ The wave of GPUs present many of the features of the wave of vector supercomputing and SIMD supercomputing of the 1980’s and 1990’s.

- Need to rethink notion of ‘optimality’

Past: counted only flops – Krylov subspace can be optimal (or near-optimal) for op. counts ➤ questioned already in 1990s

➤ Does anyone remember:

- FPS 164
- Connection Machine
- MasPar
- ICL DAP [1970's]?

➤ Difficulties were quite similar . . . *Go to the past and back !*

➤ Go back to the 1980s and 1990s to search for effective techniques.. ?

GPU Computing

- GPUs popular as : inexpensive attached processers
- Can buy $\sim$ one Teraflop peak power for around \$1,000 +
- Initial use: real-time high-definition 3D graphics
- Highly parallel (SIMD), many-core, high computational power, high memory bandwidth
- Recent announce: NVIDIA K10 - Kepler based, 3K cores, 4.6 TFLOPS peak
- Inexpensive GFLOPS

Tesla C1060

** Joint work with Ruipeng Li

CUDA (Compute Unified Device Architecture)

- SIMD-type parallelism
- Programmable in C/C++ with CUDA extensions/ tools
- Wrapper available for Python, FORTRAN, Java and MATLAB
- CuBLAS, CuFFT
- Some major changes in coding habits (e.g. no OS on GPU side)

The CUDA environment: The big picture

➤ A host (CPU) and an attached device (GPU)

Typical program:

1. Generate data on CPU
2. Allocate memory on GPU

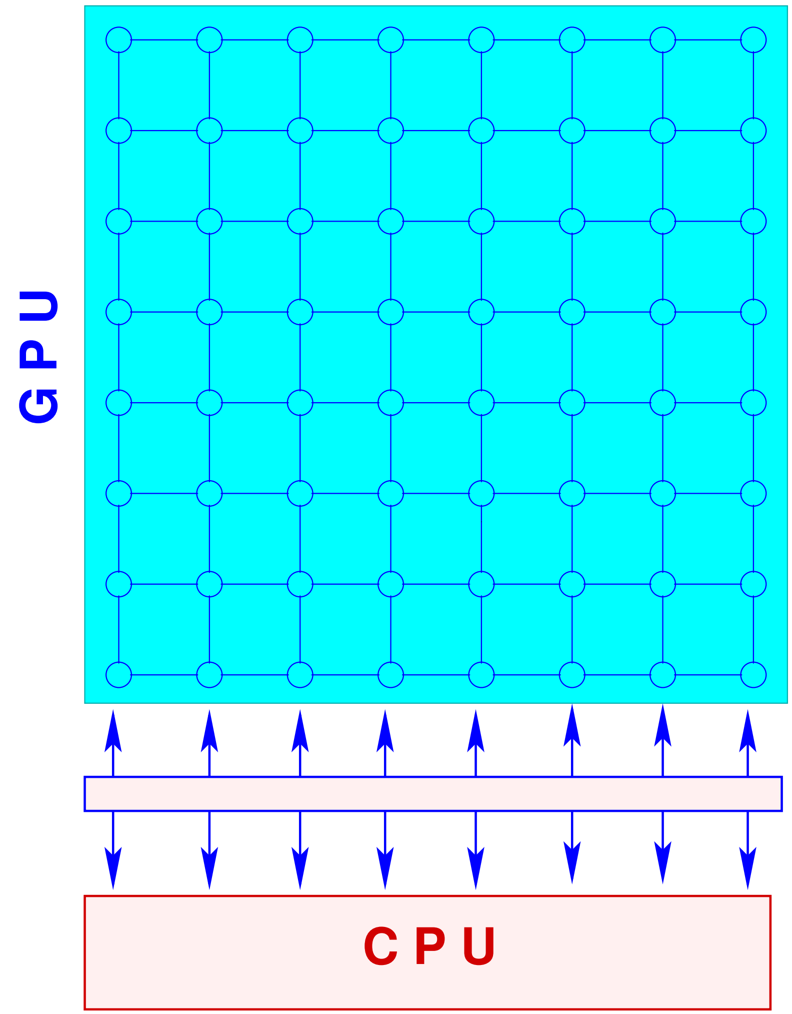
   `cudaMalloc(...)`
3. Send data Host → GPU

   `cudaMemcpy(...)`
4. Execute GPU 'kernel':

`kernel <<<(...)>>>(..)`

5. Copy data GPU →CPU

   `cudaMemcpy(...)`

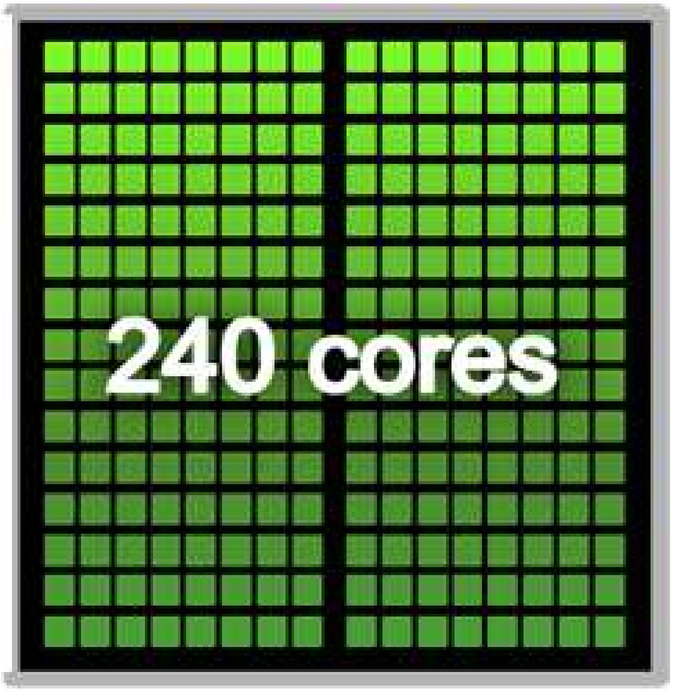


Sparse Matrix Vector Product (Spmv)

- Important operaytion in Krylov subspace methods + in applications (FEM, ...)
- Yields a small fraction of peak performance (indirect and irregular memory accesses)
- High-performance parallel Spmv kernel implemented on GPUs + various optimizations for different formats..

Hardware used

- CPU: Intel Xeon E5504 2.00 GHz
- GPU: NVIDIA Tesla C1060



Tesla C1060:

* 240 cores per GPU
* 4 GB memory
* Peak rate: 930 Gfl [single]
* Clock rate: 1.3 Ghz
* 'Compute Capability': 1.3 [allows double precision]

CSR Format Spmv – CPU vs. GPU

Single prec.

			CPU	GPU
Matrix	N	NNZ	Gflops	Gflops
Boeing/bcsstk36	23,052	1,143,140	0.93	8.1
Boeing/ct20stif	52,329	2,698,463	0.88	8.9
DNVS/ship_003	121,728	8,086,034	0.89	9.1
COP/CASEYK	696,665	4,661,931	0.58	2.9

Double prec.

			CPU	GPU
Matrix	N	NNZ	Gflops	Gflops
Boeing/bcsstk36	23,052	1,143,140	0.83	6.3
Boeing/ct20stif	52,329	2,698,463	0.81	7.1
DNVS/ship_003	121,728	8,086,034	0.81	7.2
COP/CASEYK	696,665	4,661,931	0.4	2.0

Sparse Matvecs - 3 different formats

- Matrices:

Matrix -name	N	NNZ
FEM/Cantilever	62,451	4,007,383
Boeing/pwtk	217,918	11,634,424

	Single Precision			*Double Precision*		
Matrix	CSR	JAD	DIA	CSR	JAD	DIA
FEM/Cantilever	9.4	10.8	25.7	7.5	5.0	13.4
Boeing/pwtk	8.9	16.6	29.5	7.2	10.4	14.5

Sparse Forward/Backward Sweeps

➤ Next major ingredient of precond. Krylov subs. methods

➤ ILU preconditioning operations require L/U solves: $x \leftarrow U^{-1}L^{-1}x$

➤ Sequential outer loop.

```
for i=1:n
    for j=ia(i):ia(i+1)
        x(i) = x(i) - a(j)*x(ja(j))
    end
end
```

➤ Parallelism can be achieved with level scheduling:

- Group unknowns into levels
- unknowns $x(i)$ of same level can be computed simultaneously
- $1 \leq nlev \leq n$

ILU: Sparse Forward/Backward Sweeps

- Very poor performance [relative to CPU]

Matrix	N	CPU	GPU-Lev	
		Mflops	#lev	Mflops
Boeing/bcsstk36	23,052	627	4,457	43
FEM/Cantilever	62,451	653	2,397	168
COP/CASEYK	696,665	394	273	142
COP/CASEKU	208,340	373	272	115

Prec: miserable :-)

GPU Sparse Triangular Solve with Level Scheduling

- Performance can be *very* poor when #levs is large: worst case: #levs=n, ≈ 2 Mflops

- Can reduce the number of levels drastically with Min. Degree order.

- Remember Multicolor ordering? Could use this too...

- Other issues involved. Main ones: cost of MD ordering itself for MMD; Number of iterations $\uparrow$ for multicoloring

- In general: best to *avoid ILU-type preconditioners*

ILU Preconditioning : ILU0 Preconditioned GMRES

tol = 1.0e-6; Max Iters=1,000; Matrix format: CSR

Matrix	its.	ITSOL sec.	GPUsol sec.	Speedup
Boeing/msc10848	39	2.13	0.73	2.9
Boeing/ct20stif	Fail	157.5	40.4	
DNVS/ship_003	760	323.0	80.9	4.0
COP/CASEYK	100	61.81	7.72	8.0

Single prec.

ILU0 Preconditioned GMRES Solver Time

ILU(2) Preconditioned GMRES

Matrix	its.	ITSOL sec.	GPUsol sec.	Speedup
Boeing/msc10848	2	0.32	0.21	1.5
Boeing/ct20stif	33	8.72	6.45	1.3
DNVS/ship_003	30	22.4	12.7	1.8
COP/CASEYK	33	33.8	8.88	3.8

Single prec.

ILU(2) Preconditioned GMRES Solver Time

- tol = 1.0e-6; MaxIters=500; Matrix format:CSR
- Speedup drops ! ... Denser L/U, #levels↑ Performance of L/U solve↓

Back to the 1980s: Polynomial Preconditioners

- $M^{-1} = s(A)$, where $s(t)$ is a polynomial of low degree
- Solve: $s(A) \cdot Ax = s(A) \cdot b$
- $s(A)$ need not be formed explicitly
- $s(A) \cdot Av$: Preconditioning Operation: a sequence of matrix-by-vector product to exploit high performance Spmv kernel
- Inner product on space $\mathbb{P}_k$ ($\omega \geq 0$ is a weight on (α, β))

$$\langle p, q \rangle_\omega = \int_\alpha^\beta p(\lambda) q(\lambda) \omega(\lambda)\, d\lambda$$

- Seek polynomial s_{k-1} of degree $\leq k - 1$ which minimizes

$$\|1 - \lambda s(\lambda)\|_\omega$$

Always add diagonal scaling

$$\boxed{A \leftarrow D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}}}$$

- D is the diagonal of A. Scale $A's$ rows and columns symmetrically by $A \leftarrow D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}}$
- $a_{ii} = 1$
- Some improvements (in general) at virtually no cost

Recall one of the main arguments against polynomial preconditioning: It is sub-optimal [consider the SPD case only].

L-S Polynomial Preconditioning

Tol = 1.0e-6; Max Iters = 1,000; SPD Matrices; Degree = 8; *:MD reordering applied

Matrix	ITSOL-ILU(3)		GPU-ILU(3)		L-S Polyn(8)	
	its.	sec.	its.	sec.	its.	sec.
bcsstk36	Fail	93.7	**351***	**10.6***	586	3.0
ct20stif	27	9.3	**21***	**2.2***	91	0.83
ship_003	27	27.9	27	21.1	142	3.3
msc23052	181	19.4	181	6.0	586	2.9
bcsstk17	46	1.8	46	2.8	303	0.91

single prec.

ILU(3) & L-S Polynomial Preconditioning

L-S Polynomial Preconditioning

Tol=1.0e-6; MaxIts=1,000; *:MD reordering applied

Matrix	ITSOL-ILU(3)		GPU-ILU(3)		L-S Polyn		
	iter.	sec.	iter.	sec.	iter.	sec.	Deg
bcsstk36	Fail		**351***	**10.6***	31	1.34	100
ct20stif	27	9.4	**21***	**2.22***	16	0.70	50
ship_003	27	25.8	27	21.1	10	2.90	100
msc23052	181	18.5	181	6.0	37	1.28	80
bcsstk17	46	1.8	46	2.8	22	0.55	120

single prec.

ILU(3) & L-S Polynomial Preconditioning

Must account for preconditioner construction time

- High level fill-in ILU preconditioner can be very expensive to build
- L-S Polynomial preconditioner set-up time ≈ very low
- Example: ILU(3) and L-S Poly with 20-step Lanczos procedure (for estimating interval bounds).

Matrix	N	ILU(3) sec.	LS-Poly sec.
Boeing/ct20stif	23,052	15.63	0.26

Preconditioner Construction Time

GPUsol Library:

GPUsol.a:

www.cs.umn.edu/~saad/software

- Matrix Formats:
  - CSR, JAD, DIA
- Accelerator: FGMRES
- Preconditioners:
  - ILUT, ILUK (+ level sched.)
  - L-S Polynomial
  - Block ILU
- Utilities:
  - RCM/MMD reordering
  - GPU Lanczos Algorithm

➤ Developed by: Ruipeng Li

Back to the future: An alternative (work in progress)

➤ What would be a good alternative?

Answer:

- A preconditioner requiring few ‘irregular’ computations
- Trade volume of computations for speed
- If possible something that is robust for indefinite case

➤ Good candidate:

- Multilevel Recursive Low-Rank (MRLR) approximate inverse preconditioners

Related work:

- Work on HSS matrices [e.g., JIANLIN XIA, SHIVKUMAR CHANDRASEKARAN, MING GU, AND XIAOYE S. LI, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.]
- Work on H-matrices [Hackbusch, ...]
- Work on 'balanced incomplete factorizations' (R. Bru et al.)
- Work on "sweeping preconditioners" by Engquist and Ying.
- Work on computing the diagonal of a matrix inverse [Jok Tang and YS (2010) ..]

Low-rank Multilevel Approximations

➤ Starting point: symmetric matrix derived from a 5-point discretization of a 2-D Pb on $n_x \times n_y$ grid

$$A = \left(\begin{array}{cccc|ccc} A_1 & D_2 & & & & & \\ D_2 & A_2 & D_3 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & D_\alpha & A_\alpha & D_{\alpha+1} & & \\ \hline & & & D_{\alpha+1} & A_{\alpha+1} & \ddots & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & D_{n_y} & A_{n_y} \end{array}\right)$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \equiv \begin{pmatrix} A_{11} & \\ & A_{22} \end{pmatrix} + \begin{pmatrix} & A_{12} \\ A_{21} & \end{pmatrix}$$

- $A_{11} \in \mathbb{R}^{m \times m}$, $A_{22} \in \mathbb{R}^{(n-m)\times(n-m)}$
Assume $0 < m < n$, and m is a multiple of n_x

- In the simplest case second matrix is:

$$\begin{pmatrix} & A_{12} \\ A_{21} & \end{pmatrix} = \begin{bmatrix} & -\mathbf{I} \\ -\mathbf{I} & \end{bmatrix}$$

Write this as:

$$\begin{bmatrix} & -\mathbf{I} \\ -\mathbf{I} & \end{bmatrix} = \begin{bmatrix} +\mathbf{I} & \\ & +\mathbf{I} \end{bmatrix} - \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{I} \end{bmatrix}}_{\mathbf{E}\,\mathbf{E}^T}$$

$$\mathbf{E}^T = \begin{bmatrix} \mathbf{I} & \mathbf{I} \end{bmatrix}$$

- Above splitting can be rewritten as

$$A = \begin{pmatrix} A_{11} + E_1E_1^T & \\ & A_{22} + E_2E_2^T \end{pmatrix} - \begin{pmatrix} E_1E_1^T & E_1E_2^T \\ E_2E_1^T & E_2E_2^T \end{pmatrix}$$. i.e.,

$$A = B - EE^T,$$

$$B := \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad E := \begin{pmatrix} E_1 \\ E_2 \end{pmatrix} \in \mathbb{R}^{n\times n_x},$$

Note: $B_1 := A_{11} + E_1E_1^T, \quad B_2 := A_{22} + E_2E_2^T.$

- Shermann-Morrison formula:

$$A^{-1} \equiv B^{-1} + B^{-1}EX^{-1}E^TB^{-1}$$
$$X = I - E^TB^{-1}E$$

- First thought : approximate X and exploit recursivity

$$B^{-1}[v + E\tilde{X}^{-1}E^T B^{-1}v].$$

- However wont work : cost explodes with # levels

- Alternative: low-rank approx. for $B^{-1}E$

$$\boxed{B^{-1}E \approx U_k V_k^T,} \quad \begin{array}{l} U_k \in \mathbb{R}^{n \times k}, \\ V_k \in \mathbb{R}^{n_x \times k}, \end{array}$$

- Replace $B^{-1}E$ by $U_k V_k^T$ in $X = I - (E^T B^{-1})E$:

$X \approx G_k = I - V_k U_k^T E, \quad (\in \mathbb{R}^{n_x \times n_x})$ Leads to ...

- Preconditioner:

$$M^{-1} = B^{-1} + U_k[V_k^T G_k^{-1} V_k]U_k^T$$

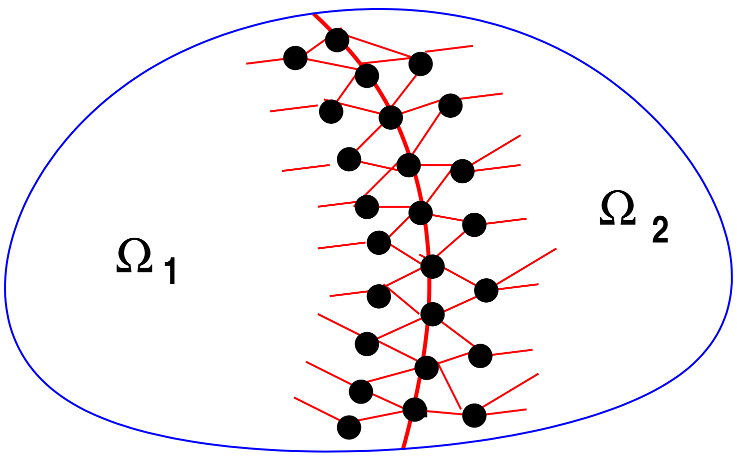
Use recursivity

$$M^{-1} = B^{-1} + U_k H_k U_k^T, \quad \text{with} \quad H_k = V_k^T G_k^{-1} V_k.$$

- We can show: $\boxed{H_k = (I - U_k^T E V_k)^{-1}}$
  ... and $H_k^T = H_k$

Question: How to generalize this?

- Adopt a Domain Decomposition viewpoint
- Implemented & tested for general matrices
- See paper for details



- Note: implementation on GPUs still far away

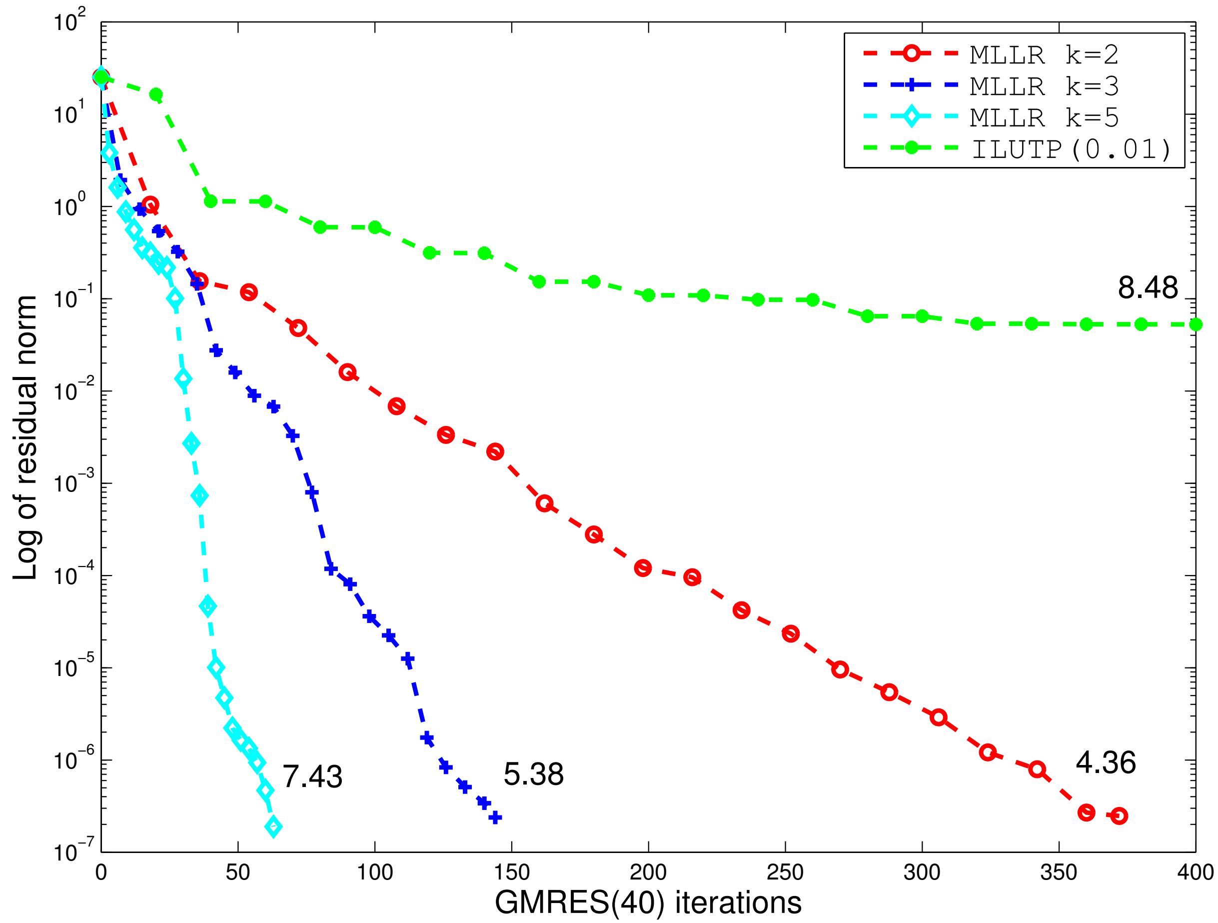
An example - Helmoltz-like equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \rho u = -6 - \rho\left(2x^2 + y^2\right) \text{ in } \Omega,$$

\+ Boundary conditions so solution is known

- ρ = constant selected to make problem more or less difficult
- Finite differences on a 66×66 mesh (matrix size 4,096).
- $\rho = 845$ selected so original Laplacean is shifted by 0.2
- Observation: MRLR starts converging for $k = 2$.

Comparison with ILUTP for 2D Helmholtz example



Standard ILUTP vs. MRLR-E; # levels = 7 for MRLR

k	nlev=7		nlev=6		nlev=5		nlev=4		nlev=3	
2	318	3.56	372	4.36	261	4.77	183	4.80	47	5.53
3	192	4.78	144	5.38	144	5.59	102	5.41	38	5.94
4	181	6.03	132	6.41	74	6.41	45	6.02	35	6.35
5	75	7.20	63	7.43	39	7.22	33	6.63	31	6.76
6	45	8.52	41	8.46	35	8.04	29	7.24	28	7.16

MRLR-E: GMRES(40) iteration counts and fill ratios

Helmoltz-like equation - a 3D case

- Similar set-up to 2D case. Solution known.
- $26 \times 26 \times 26$ grid $\rightarrow$ size $n = 24^3 = 13,824$
- $\rho = 312.5 \rightarrow$ shift $== 0.5 \rightarrow$ very indefinite problem

GMRES(40)-MRLR iteration counts and fill ratios

	nlev=6		nlev=5		nlev=4	
k	# its	fill	# its	fill	# its	fill
2	377	5.49	177	6.66	114	8.46
4	293	6.97	138	7.84	88	9.35
6	187	8.46	101	9.03	73	10.23
8	116	9.95	78	10.22	51	11.12

- ILUTP fails even for quite small values of droptol (fill-fact $>$ 11.60)

In summary:

- $\approx$ 10-x speed-up for sparse matvecs with GPUs relative to (Intel Xeon E5504) CPU
- Modest gains on overall preconditoned Krylov solver on GPU (up to ≈ 7-x speedup) with ILU
- General rule: Avoid ILU - especially with high fill level
- 'Sub-optimal' polynomial preconditioner does well
- Usual 'optimal' approaches must be revisited.
- Promising approach: RMLR approximate inverse

Conclusion

➤ Dont know what future will bring, but ...

➤ ... if you need to implement irregular sparse computations on GPUs ...

Conclusion

- ➤ Dont know what future will bring, but ...
- ➤ ... if you need to implement irregular sparse computations on GPUs ...

- ➤ ... your future is likely to include lots of hard work ...
- ➤ ... and disappointment

Conclusion

➤ Dont know what future will bring, but ...

➤ ... if you need to implement irregular sparse computations on GPUs ...

➤ ... your future is likely to include lots of hard work ...

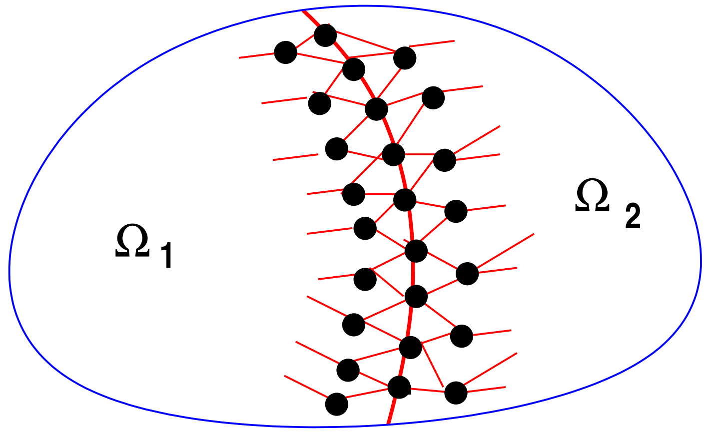
➤ ... and disappointment

➤ Either the hardware will evolve to yield good performance for sparse computations or we will need to be *very* creative ...

Q U E S T I O N S ??

Generalization: Domain Decomposition framework

Domain partitioned into 2 domains with an edge separator



➤ Matrix can be permuted to:

$$PAP^T = \left(\begin{array}{cc|cc} \hat{B}_1 & \hat{F}_1 & & \\ \hat{F}_1^T & C_1 & & -X \\ \hline & & \hat{B}_2 & \hat{F}_2 \\ & -X^T & \hat{F}_2^T & C_2 \end{array}\right)$$

➤ Interface nodes in each domain are listed last.

➤ Each matrix $\hat{B}_i$ is of size $n_i \times n_i$ (interior var.) and the matrix C_i is of size $m_i \times m_i$ (interface var.)

$$\text{Let:} \qquad E_\alpha = \begin{pmatrix} 0 \\ \alpha I \\ 0 \\ \frac{X^T}{\alpha} \end{pmatrix} \qquad \text{then we have:}$$

$$PAP^T = \begin{pmatrix} B_1 & \\ & B_2 \end{pmatrix} - EE^T \quad \text{with} \quad B_i = \begin{pmatrix} \hat{B}_i & \hat{F}_1 \\ \hat{F}_i^T & C_i + D_i \end{pmatrix}$$

$$\text{and} \quad \begin{cases} D_1 & = \alpha^2 I \\ D_2 & = \frac{1}{\alpha^2} X^T X \end{cases} .$$

➤ α used for balancing

- Better way to achieve balancing: $X = LU$
- $L \in \mathbb{R}^{m_1 \times l}$ and $U \in \mathbb{R}^{l \times m_2}$, in which $l = \min(m_1, m_2)$.
- Note: X not square.

$$\text{Then take:} \qquad E_{LU} = \begin{pmatrix} 0 \\ L \\ 0 \\ U^T \end{pmatrix},$$

- $D_1 = LL^T$ and $D_2 = U^T U$. Now E is of size $n \times l$.

General matrices

- 17 matrices from the Univ. Florida sparse matrix collection + one from a shell problem.
- 7 matrices are SPD
- Size varies from $\boldsymbol{n = 1,224}$ (HB/bcsstm27) to $\boldsymbol{n = 9,000}$ (AG-Monien/3elt1 dual)
- nnz varies from $\boldsymbol{nnz = 5,300}$ (HB/bcspwr06) to $\boldsymbol{nnz = 355,460}$ (Boeing/bcsstk38).

MATRICES (SPD)	RMLR				ICT/ILUTP	
	nlev	k	fill-ratio	#its	fill-ratio	#its
FIDAP/ex10	3	4	0.7	220	1.4	F
FIDAP/ex10hs	3	4	0.7	151	1.2	F
HB/bcsstk24	3	50	2.6	149	4.2	348
HB/bcsstk28	3	60	2.5	127	2.5	204
Cylshell/s3rmt3m1	3	50	2.6	213	2.8	F
Cylshell/s3rmt3m3	4	50	2.9	127	3.2	249
Boeing/bcsstk38	3	40	2.6	112	2.6	F

RMLR vs. ICT/ILUTP

MATRICES (Non SPD)	RMLR				ICT/ILUTP	
	nlev	k	fill-ratio	#its	fill-ratio	#its
HB/bcsstm27	4	50	1.8	26	2.3	73
HB/bcspwr06	4	5	3.1	6	5.2	F
HB/bcspwr07	5	5	3.2	6	4.8	F
HB/bcspwr08	4	5	2.1	17	5.8	F
HB/blckhole	5	50	12.8	32	21.8	F
HB/jagmesh3	4	5	5.9	30	9.7	111
Boeing/nasa1824	4	60	3.6	116	4.9	150
AG-Monien/3elt_dual	6	5	9.3	12	13.9	F
AG-Monien/airfoil1_dual	6	5	9.5	5	12.7	F
AG-Monien/ukerbe1_dual	4	5	9.1	25	10.5	F
SHELL/COQUE8E3	3	70	5.0	83	5.06	F

RMLR vs. ICT/ILUTP