High-performance computation of pseudospectra

Jack Poulson

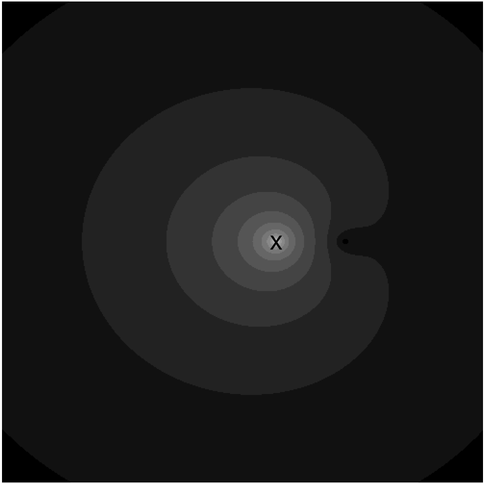
Department of Computational Science and Engineering
Georgia Institute of Technology

Math+X Seminar
Stanford University
January 15, 2014

Analyzing the norm of the resolvent

Eigenvalues only provide singularities of resolvent, $(A - \zeta I)^{-1}$

Not always enlightening about the behavior of the resolvent norm...



Recreation of first published plot of pseudospectra [Demmel-1987]

Analyzing the norm of the resolvent

Many authors [Varah-1967,Landau-1975,Godunov et al.-1982,Trefethen-1990,Hinrichsen/Pritchard-1992] proposed variants of the ϵ-*pseudospectrum*:

$$\begin{aligned} \Lambda_\epsilon(A) &= \{\zeta \in \mathbb{C} : \|(A - \zeta I)^{-1}\|_2 > \epsilon^{-1}\} \\ &= \{\zeta \in \mathbb{C} : \sigma_{\text{min}}(A - \zeta I) < \epsilon\} \\ &= \{\zeta \in \mathbb{C} : \zeta \in \Lambda(A + E), \|E\|_2 < \epsilon\} \end{aligned}$$

- Trivial for normal ($AA^H = A^H A$) matrices: $\sigma_{\text{min}}(A - \zeta I) = \text{dist}(\zeta, \Lambda(A))$
- Please see "Computation of pseudospectra" [Trefethen-1999] or "Spectra and pseudospectra" [Trefethen/Embree-2005] for an in-depth introduction (this talk's title is an homage to the former)

Analyzing the norm of the resolvent

Many authors [Varah-1967,Landau-1975,Godunov et al.-1982,Trefethen-1990,Hinrichsen/Pritchard-1992] proposed variants of the ϵ-*pseudospectrum*:

$$\begin{aligned}\Lambda_\epsilon(A) &= \{\zeta \in \mathbb{C} : \|(A-\zeta I)^{-1}\|_2 > \epsilon^{-1}\} \\ &= \{\zeta \in \mathbb{C} : \sigma_{\min}(A-\zeta I) < \epsilon\} \\ &= \{\zeta \in \mathbb{C} : \zeta \in \Lambda(A+E), \|E\|_2 < \epsilon\}\end{aligned}$$

- Trivial for normal ($AA^H = A^H A$) matrices: $\sigma_{\min}(A-\zeta I) = \text{dist}(\zeta, \Lambda(A))$
- Please see “Computation of pseudospectra” [Trefethen-1999] or “Spectra and pseudospectra” [Trefethen/Embree-2005] for an in-depth introduction (this talk’s title is an homage to the former)

A naïve algorithm

Algorithm 1: Naïve pseudospectrum calculation

foreach (x, y) *in grid* **do**
 $\rho_{x,y} := \min(\mathrm{svd}(A - (x + yi)I))$

$O(N^3)$ **per grid-point.** But very reliable and okay for small matrices (used within `pscont` [Higham-1995])

Much better methods exist, even for arbitrary matrices. Most techniques use some form of inverse-iteration Lanczos at each grid-point.

Inverse iteration with the shifted Schur factor

$$\begin{aligned}\Lambda_\epsilon(A) &= \{\zeta \in \mathbb{C} : \sigma_{\min}(A - \zeta I) < \epsilon\} \\ &= \{\zeta \in \mathbb{C} : \sigma_{\min}(Q^H(A - \zeta I)Q) < \epsilon\} \\ &= \{\zeta \in \mathbb{C} : \sigma_{\min}(T - \zeta I) < \epsilon\}\end{aligned}$$

- [Lui-97] proposed inverse-iteration Lanczos w/ shifted Schur factor (and path following, which is debatable)
- Only $O(N^2)$ work per iteration, and ideally small number of iterations per shift
- Most common general-purpose algorithm, e.g., used by `EigTool` [Wright et al.-2001]
- Downsides: level 2 BLAS [Dongarra et al.-1988], and single triangular solves parallelize poorly
- "Embarrassingly parallel" over set of shifts, but large enough matrices must be distributed...

Inverse iteration with the shifted Schur factor

Algorithm 2: Schur decomp. + inverse iteration (grid algorithm)

$T = \text{schur}(A)$
foreach (α, β) *in grid* **do**
  $\rho_{\alpha,\beta} := \text{LanczosInverseIteration}(T - (\alpha + \beta i)I)$

- [Lui-97] proposed inverse-iteration Lanczos w/ shifted Schur factor (and path following, which is debatable)
- Only $O(N^2)$ work per iteration, and ideally small number of iterations per shift
- Most common general-purpose algorithm, e.g., used by `EigTool` [Wright et al.-2001]
- Downsides: level 2 BLAS [Dongarra et al.-1988], and single triangular solves parallelize poorly
- "Embarrassingly parallel" over set of shifts, but large enough matrices must be distributed...

Inverse iteration with the shifted Schur factor

Algorithm 2: Schur decomp. + inverse iteration (grid algorithm)

$T = \text{schur}(A)$
foreach (α, β) *in grid* **do**
  $\rho_{\alpha,\beta} := \text{LanczosInverseIteration}(T - (\alpha + \beta i)I)$

- [Lui-97] proposed inverse-iteration Lanczos w/ shifted Schur factor (and path following, which is debatable)
- Only $O(N^2)$ work per iteration, and ideally small number of iterations per shift
- Most common general-purpose algorithm, e.g., used by `EigTool` [Wright et al.-2001]
- Downsides: level 2 BLAS [Dongarra et al.-1988], and single triangular solves parallelize poorly
- "Embarrassingly parallel" over set of shifts, but large enough matrices must be distributed...

Inverse iteration with the shifted Schur factor

Algorithm 2: Schur decomp. + inverse iteration (grid algorithm)

$T = \text{schur}(A)$
foreach (α, β) *in grid* **do**
  $\rho_{\alpha,\beta} := \text{LanczosInverseIteration}(T - (\alpha + \beta i)I)$

- [Lui-97] proposed inverse-iteration Lanczos w/ shifted Schur factor (and path following, which is debatable)
- Only $O(N^2)$ work per iteration, and ideally small number of iterations per shift
- Most common general-purpose algorithm, e.g., used by `EigTool` [Wright et al.-2001]
- Downsides: level 2 BLAS [Dongarra et al.-1988], and single triangular solves parallelize poorly
- “Embarrassingly parallel” over set of shifts, but large enough matrices must be distributed...

Inverse iteration with the shifted Schur factor

Algorithm 2: Schur decomp. + inverse iteration (grid algorithm)

$T = \text{schur}(A)$
foreach (α, β) *in grid* **do**
  $\rho_{\alpha,\beta} := \text{LanczosInverseIteration}(T - (\alpha + \beta i)I)$

- [Lui-97] proposed inverse-iteration Lanczos w/ shifted Schur factor (and path following, which is debatable)
- Only $O(N^2)$ work per iteration, and ideally small number of iterations per shift
- Most common general-purpose algorithm, e.g., used by `EigTool` [Wright et al.-2001]
- Downsides: level 2 BLAS [Dongarra et al.-1988], and single triangular solves parallelize poorly
- "Embarrassingly parallel" over set of shifts, but large enough matrices must be distributed...

Inverse iteration with the shifted Schur factor

Algorithm 2: Schur decomp. + inverse iteration (grid algorithm)

$T = \text{schur}(A)$
foreach (α, β) *in grid* **do**
  $\rho_{\alpha,\beta} := \text{LanczosInverseIteration}(T - (\alpha + \beta i)I)$

- [Lui-97] proposed inverse-iteration Lanczos w/ shifted Schur factor (and path following, which is debatable)
- Only $O(N^2)$ work per iteration, and ideally small number of iterations per shift
- Most common general-purpose algorithm, e.g., used by `EigTool` [Wright et al.-2001]
- Downsides: level 2 BLAS [Dongarra et al.-1988], and single triangular solves parallelize poorly
- "Embarrassingly parallel" over set of shifts, but large enough matrices must be distributed...

A high-performance kernel

Each shift's Lanczos procedure repeatedly applies $(T - \zeta I)^{-1}$ and then $(T - \zeta I)^{-H}$.

Key insight: Blocked "TRSM" algorithms can be modified to allow for efficiently simultaneously solving with many different shifts

$$\{y_j := T^{-1} x_j\}_j \mapsto \{y_j := (T - \zeta_j I)^{-1} x_j\}_j$$

Thesis: All high-performance general-purpose implementations (sequential, shared-memory, distributed, accelerated, etc.) should be built around this proposed kernel.

A high-performance kernel

Each shift's Lanczos procedure repeatedly applies $(T - \zeta I)^{-1}$ and then $(T - \zeta I)^{-H}$.

Key insight: Blocked "TRSM" algorithms can be modified to allow for efficiently simultaneously solving with many different shifts

$$\{y_j := T^{-1} x_j\}_j \;\mapsto\; \{y_j := (T - \zeta_j I)^{-1} x_j\}_j$$

Thesis: All high-performance general-purpose implementations (sequential, shared-memory, distributed, accelerated, etc.) should be built around this proposed kernel.

A high-performance kernel

Each shift's Lanczos procedure repeatedly applies $(T - \zeta I)^{-1}$ and then $(T - \zeta I)^{-H}$.

Key insight: Blocked "TRSM" algorithms can be modified to allow for efficiently simultaneously solving with many different shifts

$$\{y_j := T^{-1} x_j\}_j \quad \mapsto \quad \{y_j := (T - \zeta_j I)^{-1} x_j\}_j$$

Thesis: All high-performance general-purpose implementations (sequential, shared-memory, distributed, accelerated, etc.) should be built around this proposed kernel.

A high-performance kernel

Each shift's Lanczos procedure repeatedly applies $(T - \zeta I)^{-1}$ and then $(T - \zeta I)^{-H}$.

Key insight: Blocked "TRSM" algorithms can be modified to allow for efficiently simultaneously solving with many different shifts

$$\{y_j := T^{-1} x_j\}_j \;\mapsto\; \{y_j := (T - \zeta_j I)^{-1} x_j\}_j$$

Thesis: All high-performance general-purpose implementations (sequential, shared-memory, distributed, accelerated, etc.) should be built around this proposed kernel.

Review of blocked algorithm for TRSM

Expose $O(1) \times O(1)$ bottom-right triangular submatrix of U:

$$\begin{pmatrix} Y_T \\ Y_B \end{pmatrix} = \begin{pmatrix} U_{TL} & U_{TR} \\ 0 & U_{BR} \end{pmatrix} \begin{pmatrix} X_T \\ X_B \end{pmatrix} = \begin{pmatrix} U_{TL}X_T + U_{TR}X_B \\ U_{BR}X_B \end{pmatrix}$$

Algorithm 3: Upper TRSM

$X_B := U_{BR}^{-1} Y_B$
$Y_T := Y_T - U_{TR}X_B$
$X_T :=$ Recurse(U_{TL}, Y_T)
return X

Key point: Asymptotically, all of the work is within the $U_{TR}X_B$ multiplications.

Blocked algorithm for shifted TRSM

There is an obvious trivial modification for $Y := (U - \zeta I)^{-1} X$:

$$\begin{pmatrix} Y_T \\ Y_B \end{pmatrix} = \begin{pmatrix} U_{TL} - \zeta I & U_{TR} \\ 0 & U_{BR} - \zeta I \end{pmatrix} \begin{pmatrix} X_T \\ X_B \end{pmatrix} = \begin{pmatrix} (U_{TL} - \zeta I) X_T + U_{TR} X_B \\ (U_{BR} - \zeta I) X_B \end{pmatrix}$$

Algorithm 4: Shifted upper TRSM

$X_B := (U_{BR} - \zeta I)^{-1} Y_B$
$Y_T := Y_T - U_{TR} X_B$
$X_T := \text{Recurse}(U_{TL}, \zeta, Y_T)$
return X

Notice that ζ is only used within the small diagonal block triangular solves.

Blocked algorithm for multi-shift TRSM

Algorithm 5: Multi-shift TRSM

$X_B := \text{MultiShiftTrsm}(U_{BR}, \{\zeta_j\}_j, Y_B)$
$Y_T := Y_T - U_{TR}X_B$
$X_T := \text{Recurse}(U_{TL}, \{\zeta_j\}_j, Y_T)$
return X

Again, vast majority of work lies in $U_{TR}X_B$ matrix-matrix multiplies.

In parallel: each process can redundantly perform the small multi-shift TRSM and then participate in a parallel matrix-matrix multiply for $U_{TR}X_B$.

Implementation in Elemental

- Recently implemented various distributed interleaved inverse-iteration algorithms using Elemental [P. et al.-2013]:
  - Power method (w/ optional deflation)
  - Lanczos w/o reorthog. (w/ optional deflation)
  - Lanczos w/ reorthog.+restarting (w/ optional deflation)
- Analyzed $20,000 \times 20,000$ upper-triangular matrices using 2048 cores of TACC's Stampede (500×500 grid in 16 pieces)
- **Execution time sometimes as low as two minutes.** Already achieving 25% of peak w/o tuning for modest-sized parallel problems
- Preliminary implementation of Spectral Divide and Conquer [Bai et al.-1997,Demmel et al.-2007] (still working out stability issues, already works in some large-scale cases)

Available under New BSD License from `libelemental.org`
Active Elemental Collaborators: Bientinesi, Grant, Hammond, Marker, Petschow, Schatz, and van de Geijn

Implementation in Elemental

- Recently implemented various distributed interleaved inverse-iteration algorithms using Elemental [P. et al.-2013]:
  - Power method (w/ optional deflation)
  - Lanczos w/o reorthog. (w/ optional deflation)
  - Lanczos w/ reorthog.+restarting (w/ optional deflation)
- Analyzed $20,000 \times 20,000$ upper-triangular matrices using 2048 cores of TACC's Stampede (500×500 grid in 16 pieces)
- **Execution time sometimes as low as two minutes.** Already achieving 25% of peak w/o tuning for modest-sized parallel problems
- Preliminary implementation of Spectral Divide and Conquer [Bai et al.-1997,Demmel et al.-2007] (still working out stability issues, already works in some large-scale cases)

Available under New BSD License from `libelemental.org`
Active Elemental Collaborators: Bientinesi, Grant, Hammond, Marker, Petschow, Schatz, and van de Geijn

Implementation in Elemental

- Recently implemented various distributed interleaved inverse-iteration algorithms using Elemental [P. et al.-2013]:
  - Power method (w/ optional deflation)
  - Lanczos w/o reorthog. (w/ optional deflation)
  - Lanczos w/ reorthog.+restarting (w/ optional deflation)
- Analyzed $20,000 \times 20,000$ upper-triangular matrices using 2048 cores of TACC's Stampede (500×500 grid in 16 pieces)
- **Execution time sometimes as low as two minutes**. Already achieving 25% of peak w/o tuning for modest-sized parallel problems
- Preliminary implementation of Spectral Divide and Conquer [Bai et al.-1997,Demmel et al.-2007] (still working out stability issues, already works in some large-scale cases)

Available under New BSD License from `libelemental.org`
Active Elemental Collaborators: Bientinesi, Grant, Hammond, Marker, Petschow, Schatz, and van de Geijn

Implementation in Elemental

- Recently implemented various distributed interleaved inverse-iteration algorithms using Elemental [P. et al.-2013]:
  - Power method (w/ optional deflation)
  - Lanczos w/o reorthog. (w/ optional deflation)
  - Lanczos w/ reorthog.+restarting (w/ optional deflation)
- Analyzed $20,000 \times 20,000$ upper-triangular matrices using 2048 cores of TACC's Stampede (500×500 grid in 16 pieces)
- **Execution time sometimes as low as two minutes**. Already achieving 25% of peak w/o tuning for modest-sized parallel problems
- Preliminary implementation of Spectral Divide and Conquer [Bai et al.-1997,Demmel et al.-2007] (still working out stability issues, already works in some large-scale cases)

Available under New BSD License from `libelemental.org`
Active Elemental Collaborators: Bientinesi, Grant, Hammond, Marker, Petschow, Schatz, and van de Geijn

Implementation in Elemental

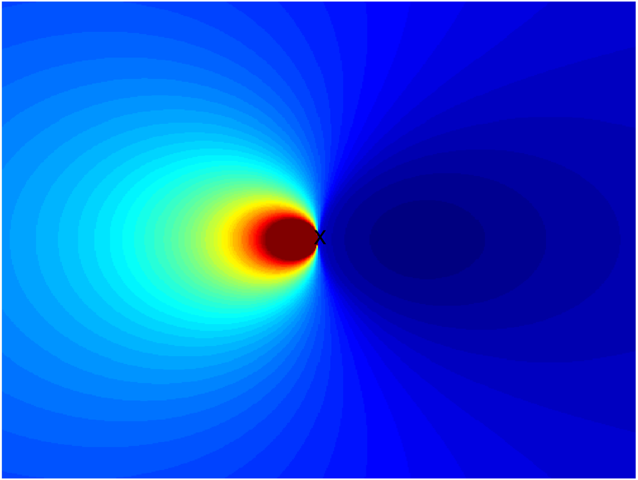
- Recently implemented various distributed interleaved inverse-iteration algorithms using Elemental [P. et al.-2013]:
  - Power method (w/ optional deflation)
  - Lanczos w/o reorthog. (w/ optional deflation)
  - Lanczos w/ reorthog.+restarting (w/ optional deflation)
- Analyzed $20,000 \times 20,000$ upper-triangular matrices using 2048 cores of TACC's Stampede (500×500 grid in 16 pieces)
- **Execution time sometimes as low as two minutes**. Already achieving 25% of peak w/o tuning for modest-sized parallel problems
- Preliminary implementation of Spectral Divide and Conquer [Bai et al.-1997,Demmel et al.-2007] (still working out stability issues, already works in some large-scale cases)

Available under New BSD License from `libelemental.org`

Active Elemental Collaborators: Bientinesi, Grant, Hammond, Marker, Petschow, Schatz, and van de Geijn

Demmel matrix

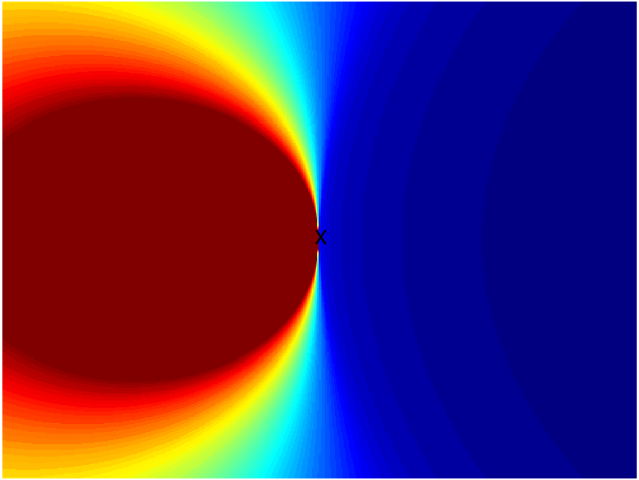
Upper-triangular Toeplitz matrix with main diagonal equal to -1, and with steady decrease to -10^4 in the top-right entry.



Window: $[-3500, 3500] \times [-3500, 3500]$

Demmel matrix

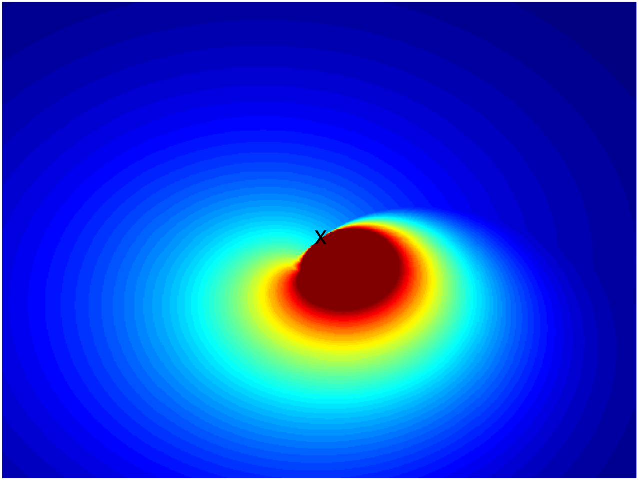
Upper-triangular Toeplitz matrix with main diagonal equal to -1, and with steady decrease to -10^4 in the top-right entry.



Window: $[-500, 500] \times [-500, 500]$

Upper-triangular Fox-Li

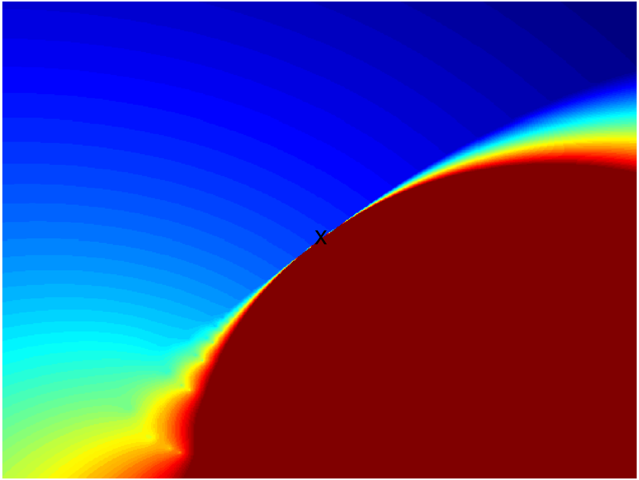
$$(\mathcal{A}u)(x) = \sqrt{iF/\pi} \int_{-1}^{x} e^{iF(x-y)^2} u(y)dy$$



Window: $[-0.35, 0.35] \times [-0.35, 0.35]$

Upper-triangular Fox-Li

$$(\mathcal{A}u)(x) = \sqrt{iF/\pi}\int_{-1}^{x} e^{iF(x-y)^2}u(y)dy$$



Window: $[-0.05, 0.05] \times [-0.05, 0.05]$

Related work on parallel pseudospectra

A wide variety of existing literature:

- Parallel path following [Mezher-2001,Bekas et al.-2001]
- "Embarrassingly parallel" Lanczos: `Fvpspack` [Braconnier-1996]
- Parallel sparse-direct shift-and-invert [Fraysse et al.-1996]

Differences from current work:

- New multi-shift TRSM drastically reduces data movement
- Other approaches not pushing for scalable Schur decomposition (will build on [Granat et al.-2010,Bai et al.-1997,Demmel et al.-2007])
- Goal is massively parallel blackbox equivalent to `EigTool`

Future work for computing pseudospectra

Overall algorithm:

- Modify `EigTool` to support high-performance kernel
- Multi-shift TRSM via BLAS-Like Interface Software (BLIS) [van Zee et al.-2013]
- Accelerator support for multi-shift TRSM
- Intelligent choice of independent subteams after Schur
- Full-scale runs on preconditioned 2D operators

Spectral Divide and Conquer:

- Intelligent Mobius transformations [Ballard et al.-2011]
- Tuned matrix-sign function implementation (scaling, Newton-Schulz, robust tolerance, accelerator support) [Higham-2008]
- Means of falling back to AED [Bai et al.-1997]

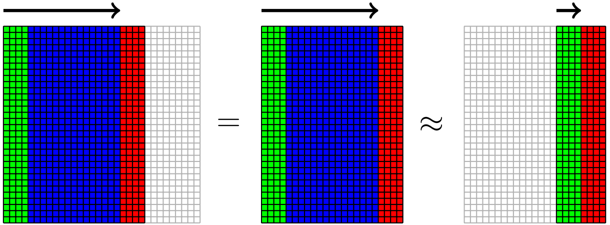
Aggressive Early Deflation (for Hessenberg QR algorithm):

- Wrap ScaLAPACK [Choi et al.-1992] parallelization [Granat et al.-2010] of AED [Braman et al.-2002])

Sparse-direct solvers and sweeping preconditioners

Collaborators: P. Tsuji and L. Ying

`github.com/poulson/Clique` and `github.com/poulson/PSP`



- 2D distributions for sparse-direct TRSM
- Extension of selective inversion [Raghavan-1998] for supernodal Bunch-Kaufman
- Originally built to support a *sweeping preconditioner* [Engquist/Ying-2011]

Butterfly algorithm and directional FMM

Collaborators: A. Benson, L. Demanet, and L. Ying
`github.com/poulson/dist-butterfly` and
`github.com/arbenson/ddfmm`

$$(\mathcal{K}g)(x) = \int_Y K(x,y)g(y)dy$$

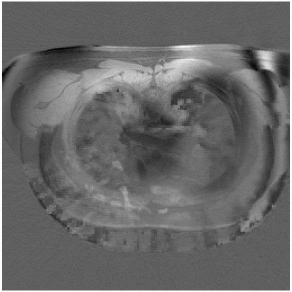
Future work:

- Near-optimal parallel algorithm for quasi-uniform butterfly
- "Cell-level" parallel sparse directional FMM
- Combine ideas in future: parallel sparse butterfly and more scalable directional FMM

Low-rank + sparse MRI

Collaborators: E. Candès and R. Otazo

`github.com/poulson/rt-lps-mri`

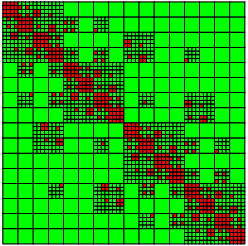


Main ideas:

- Fourier-domain work independent for each (coil,time) pair
- Image-domain work involves Tall-Skinny matrices
- Carefully alternate between appropriate 1D distributions
- Reduced reconstruction time from hours to roughly one minute

Structured matrix factorization

Collaborators: A. Benson, K. Ho, Y. Li, and L. Ying
`bitbucket.org/poulson/dmhm`



- Working scalable parallel $\mathcal{H}$-matrix composition/inversion
- $\mathcal{H}$-matrix inversion more work than fact., but more scalable
- Heirarchical Interpolative Factorizations a promising alternative

Closing

Project websites (software + references):

- Elemental: `libelemental.org`
- Clique: `github.com/poulson/Clique`
- Parallel Sweeping Preconditioner: `github.com/poulson/PSP`
- DistButterfly: `github.com/poulson/dist-butterfly`
- DDFMM: `github.com/arbenson/ddfmm`
- Low-rank + sparse MRI: `github.com/poulson/rt-lps-mri`
- $\mathcal{H}$-matrices: `bitbucket.org/poulson/dmhm`

Thanks:
National Science Foundation (SSI-SI2, ACI-1148125)
Texas Advanced Computing Center
XSEDE
Argonne National Laboratory:

- Jed Brown (MCS)
- Jeff Hammond (ALCF)

Questions?