A GPU implementation of the filtered Lanczos procedure

Vassilis Kalantzis

Computer Science and Engineering Department
University of Minnesota - Twin Cities, USA

UMN Graduate Colloquium
09-16-2016

Acknowledgments

- Joint work with Jared Aurentz and Yousef Saad.
- Special thanks to Minnesota Supercomputing Institute for allowing access to its supercomputers.
- Work supported by NSF and DOE (DE-SC0008877).

UNIVERSITY OF MINNESOTA

Supercomputing Institute

Introduction

The symmetric eigenvalue problem

$$Ax^{(i)} = \lambda_i x^{(i)}, \quad i = 1, \ldots, n,$$

where $A \in \mathbb{R}^{n \times n}$, $x^{(i)} \in \mathbb{R}^n$ and $\lambda_i \in \mathbb{R}$. A pair $(\lambda_i, x^{(i)})$ is an *eigenpair* of A.

Focus

1. Compute, up to a tolerance `tol`, all eigenpairs $(\lambda_i, x^{(i)})$ located inside the interval $[\alpha, \beta]$ where $\alpha,\ \beta \in \mathbb{R}$ and $\lambda_1 \leq \alpha,\ \beta \leq \lambda_n$.

More focus...

1. A is sparse and n can be in the order of $O(10^4 - 10^6)$
2. $[\alpha, \beta]$ might contain hundreds/thousands of eigenpairs...
3. ...which might also be (heavily) clustered

Electronic structure calculations

- Density Functional Theory $\rightarrow$ Kohn-Sham equations

$$\left[-\frac{\nabla^2}{2} + V_{ion}(r) + V_H(\rho(r), r) + V_{XC}(\rho(r), r)\right] \Psi_i(r) = E_i \Psi_i(r)$$

$$\rho(r) = 2\sum_{i=1}^{n_{occ}} |\Psi_i(r)|^2 \text{ (charge density)}$$

$$\nabla^2 V_H(r) = -4\pi\rho(r).$$

- Many electron $\rightarrow$ charge density
- Both V_{XC} and V_H depend on ρ
- Potentials and charge densities must be self-consistent $\rightarrow$ nonlinear eigenvalue problem

The Lanczos algorithm

Pseudocode

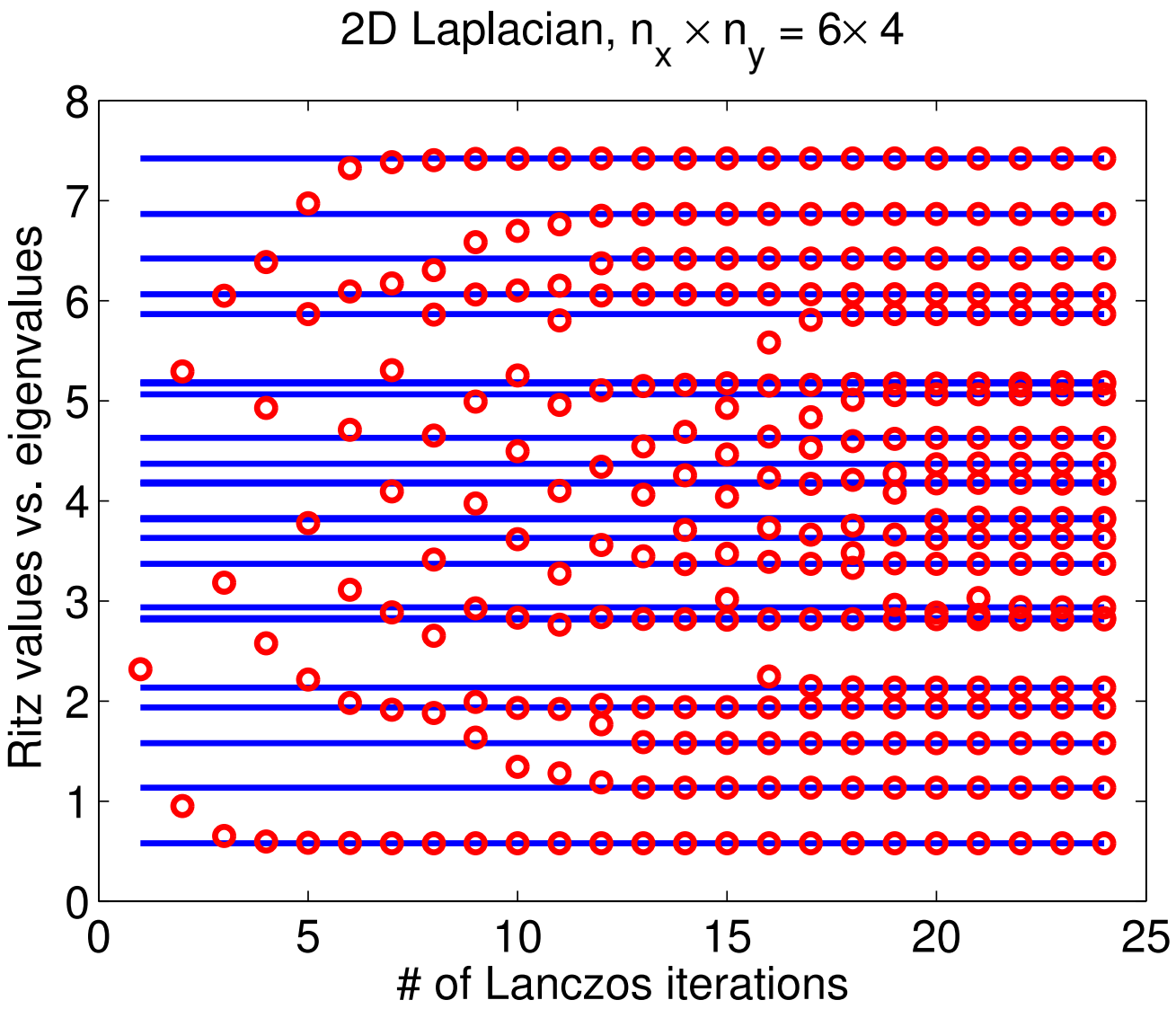
0. **Input** :A, x$\in \mathbb{R}^n$, $\beta_1 = \|x\|$, $q_0 = 0$, $q_1 = x/\beta$, $k \in \mathbb{Z}$
1. *For* j=1,. . . ,k-1:
2. $\quad \alpha_j = q_j^\top A q_j$
3. $\quad q_j = (A - \alpha_j) q_j - \beta_{j-1} q_{j-1}$
4. $\quad \beta_{j+1} = \|q_j\|, \; q_{j+1} = q_j/\beta_{j+1}$
5. *End*

Properties

- Approximate eigenpairs of A have the form $(\theta, [q_1, \ldots, q_k]y)$, with $T_k y = \theta y$.
- Finite precision arithmetic $\rightarrow$ reorthogonalization

$$T_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \end{pmatrix}$$

Typical convergence of Lanczos



Computation of interior eigenpairs

Shift-and-Invert

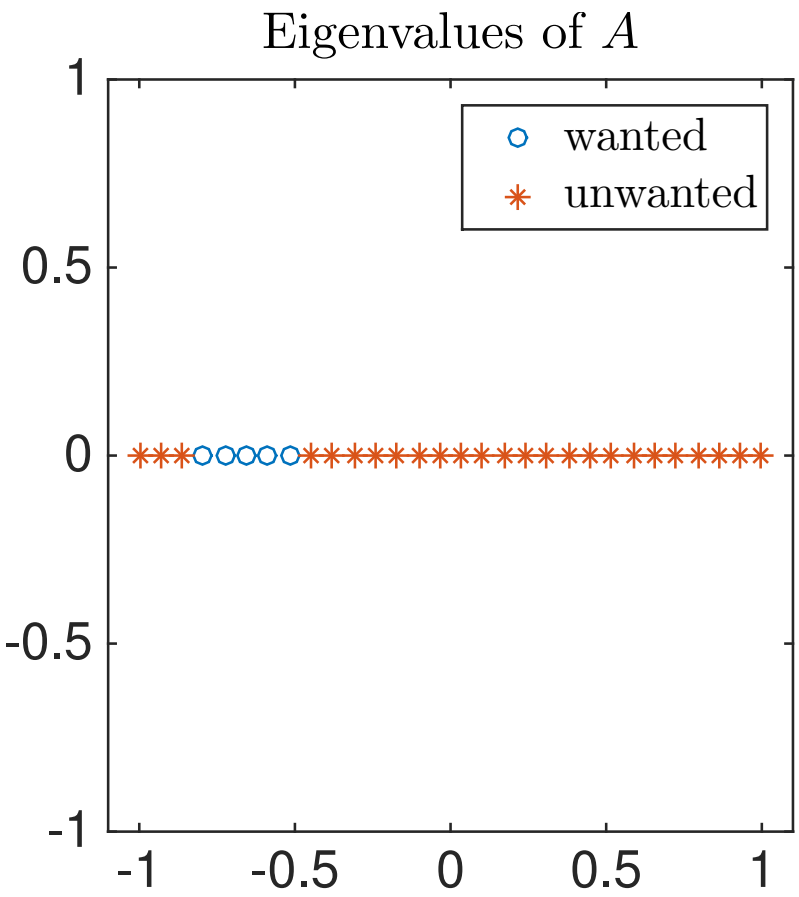
- Lanczos (typically) converges fast to exterior eigenpairs
- Interior eigenpairs? $\rightarrow$ Apply Lanczos on $(A - \sigma I)^{-1}$ (Shift-and-Invert)
- Rapid convergence of eigenpairs close to σ but...Factorization...
- Alternatives: Iterative solvers, out-of-core implementations $\rightarrow$ typically not practical

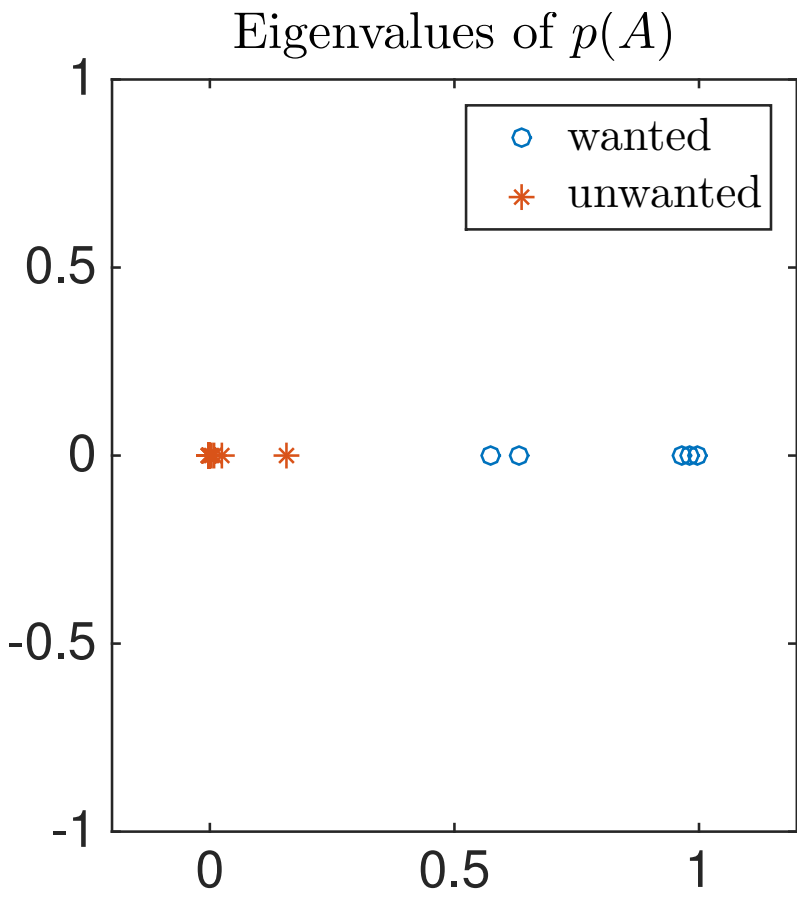
Alternative: access A only in a MV form

- Bring interior eigenvalues to the exterior part, e.g., $(A - \sigma I)^2$, $I - \gamma(A - \sigma I)^2$...
- Lanczos will work but...
- ...eigenvalues are now poorly separated
- The right key (polynomial) but the wrong keyhole (badly chosen)

Polynomial filtering

- **Idea:** Apply Lanczos to $p(A)$ instead of A
- $p(A)$ is a matrix polynomial
- **Goal:** Amplify/damp wanted/unwanted portion of the spectrum





Polynomial filtering (cont.)

Key properties of $p(A)$

1. Map eigenvalues of A in $[\alpha, \beta]$ to the dominant ones of $p(A)$
2. Damp all eigenvalues of A outside $[\alpha, \beta]$ to zero eigenvalues of $p(A)$

(Almost) Optimal $p(A)$ is known

$$\phi(z) = \begin{cases} 1, & z \in [\alpha, \beta], \\ 0, & \text{otherwise.} \end{cases}$$

The eigenvalues of $\phi(A)$ are 0 or 1.

- $\phi()$ is approximated by $p()$ of degree m
- More tricky than it looks: $p()$ should not be overfitting $\phi()$

- Each MV product with $p(A)$ requires m MV products with A
- Heavily dependent on an efficient MV routine with A
- Not a (very) good alternative for matrices with irregular spectra (but this is another story...)

Chebyshev polynomials

Chebyshev polynomials of the first kind

$$T_{i+1}(z) = 2zT_i(z) - T_{i-1}(z), \ i \geq 1.$$

starting with $T_0(z) = 1, \ T_1(z) = z$.

Chebyshev series approximation

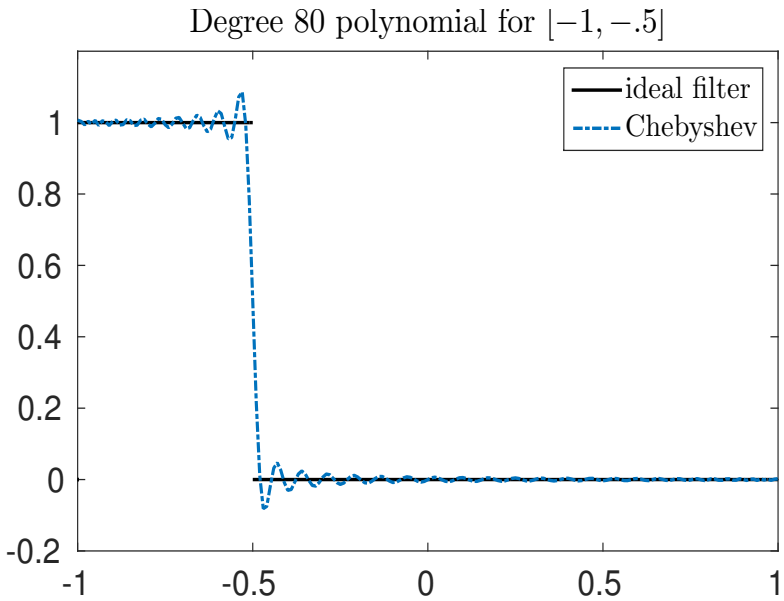
$$\phi(z) = \sum_{i=0}^{\infty} b_i T_i(z),$$

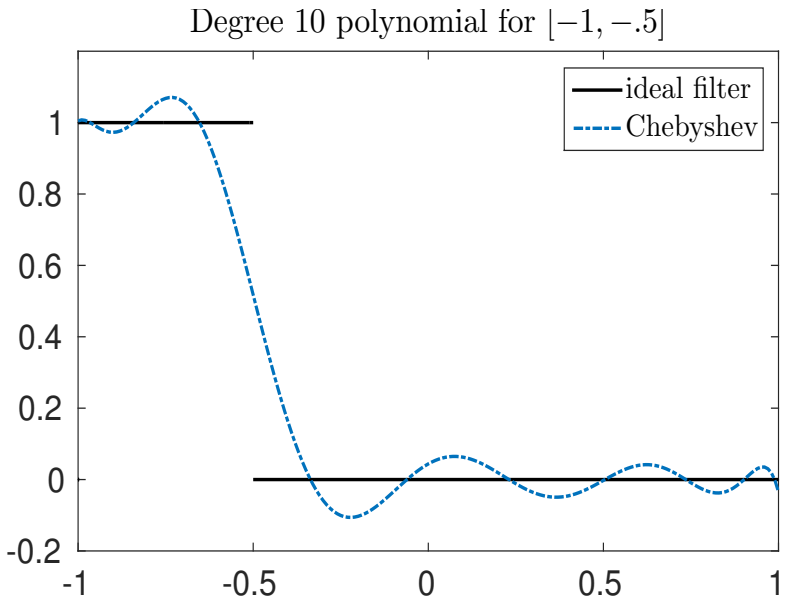
where (for given α and β),

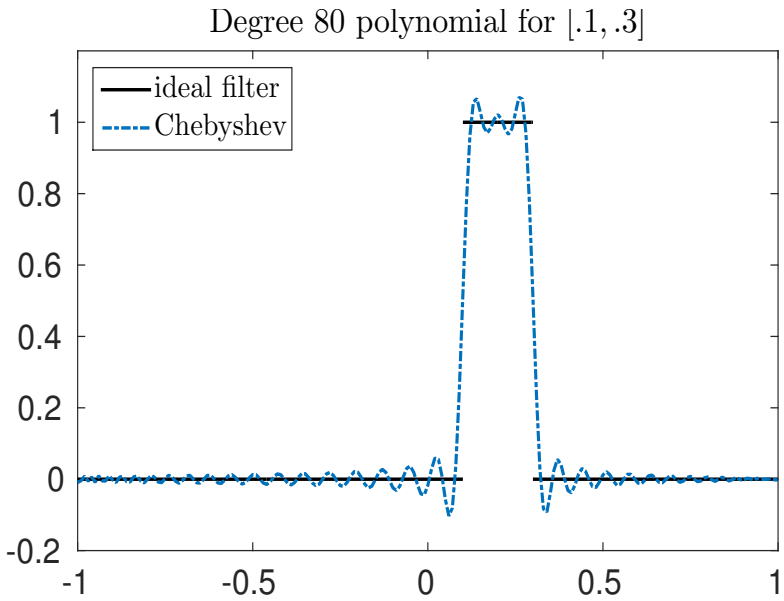
$$b_i = \begin{cases} (\arccos(\alpha) - \arccos(\beta))/\pi, & i = 0, \\ 2\left(\sin\left(i \arccos(\alpha)\right) - \sin\left(i \arccos(\beta)\right)\right)/i\pi, & i > 0. \end{cases}$$

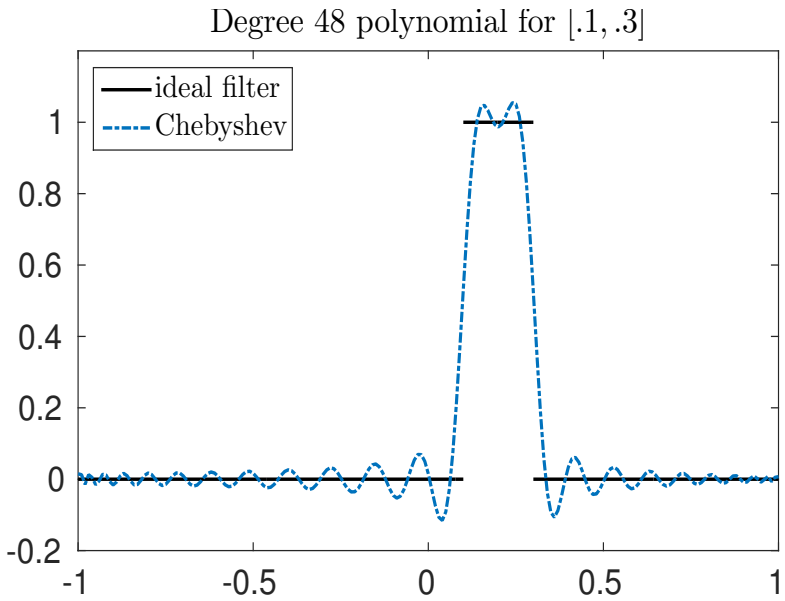
In practice we fix m and truncate: $\mathrm{p}_m(z) = \sum_{i=0}^{m} b_i T_i(z)$.

Chebyshev approximation (eigvls on A lie in [-1,1])

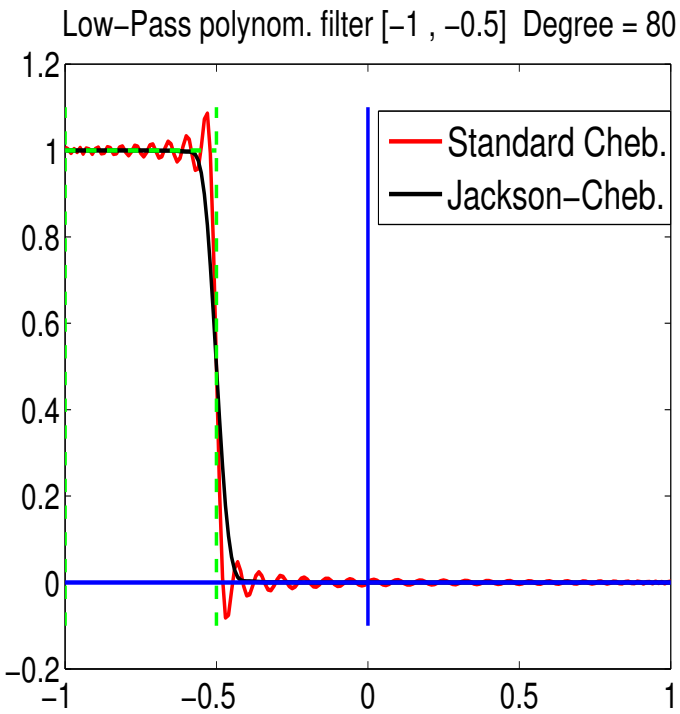


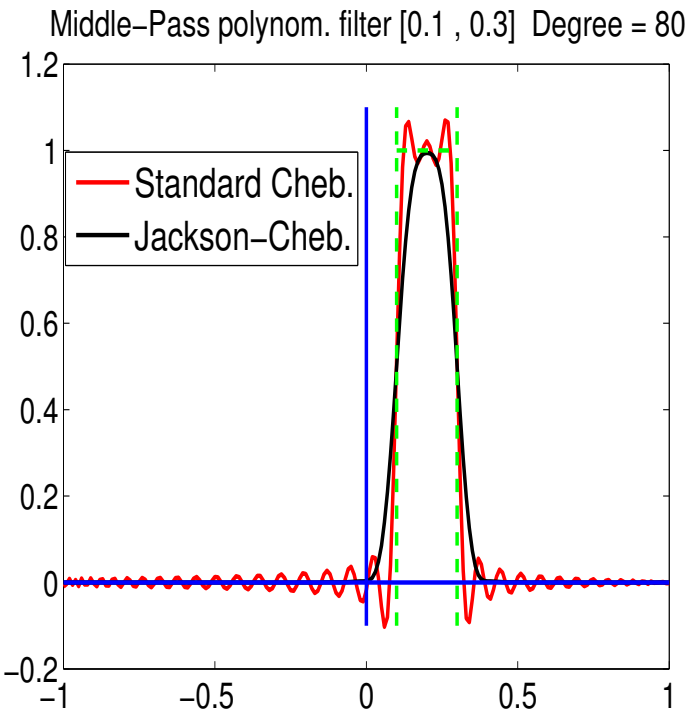






Chebyshev with Jackson damping





Chebyshev filtered MV product

Pseudocode

/* Assume that A is scaled so that its eigenvalues lie in $[-1, 1]$ */

0. **Input** :A, x$\in \mathbb{R}^n$, $b \in \mathbb{R}^{m+1}$, $m \in \mathbb{Z}$
1. y=Ax, f = $b_0 x + b_1 Ax$
2. *For* j=2,. . . ,m:
3. t = 2Ay - x
4. f = f + $b_j t$
5. x = y, y = t
6. *End*

The GPU architecture

- Ideal for number crunching
- Throughput-oriented model of computation
- Internal hierarchy: grids $\rightarrow$ blocks $\rightarrow$ warps

Source: https://people.duke.edu/ ccc14/sta-663/CUDAPython.html

Polynomial filtering and GPGPU computing (cont.)

Why the GPU?

- Easily programmable (CUDA), high performance/$ ratio
- Give an extra tool to GPU practitioners that might not be willing/have enough time to study advanced numerical techniques

Email by a manager in a top semiconductors company

... I recently learned of another approach to extract non-extremal eigenvalues... It is called the Folded Spectrum Method, and allows the user to target any value in the spectrum. The cost is roughly two sparse MV applications per iteration... Have you ever worked on this method?...

The cucheb library (Aurentz, VK, Saad)

The numerical part

- (Block) Lanczos without restart $\rightarrow$ full orthogonalization...
- ...combined with Chebyshev filtering
- Main rationale: "Shift" the cost to the MV product with $p(A)$
- Not ideal to compute more than a few hundreds of eigenpairs (nor that was our intention $\rightarrow$ spectrum slicing)
- Simple criteria to compute an "optimum" degree m

The implementation part

- Implemented having simplicity/compactness as our top goal
- Written in CUDA C/C++
- Linear algebra operations are performed using the cuBLAS and cuSPARSE libraries
- In-house MV routines and dense eigenvalue solvers are provided
- Supports different types of filters and sparse matrix formats

`https://github.com/jaurentz/cucheb`

Code snippet ($[\alpha, \beta] = [.3, .5]$)

```
int main()
// initialize cuchebmatrix object
cuchebmatrix ccm;
string mtxfile("H2O.mtx");
cuchebmatrix_init(mtxfile, ccm);

// declare cucheblanczos variable
cucheblanczos ccl;

// compute eigenvalues in [.5,.6] using block
filtered Lanczos
cuchebmatrix_filteredlanczos(.5, .6, 3, ccm, ccl);

. . .
```

Experimental framework (double precision arithmetic)

Hardware/Software

- Host: Intel Xeon ES-2667 v2 3.30GHz with 256GB of CPU RAM
- GPU: NVIDIA K40 with 12GB of GPU RAM and 2880 compute cores
- Software: CUDA C/C++ (nvcc)
- External libraries: cuSPARSE, cuBLAS

Lanczos setup

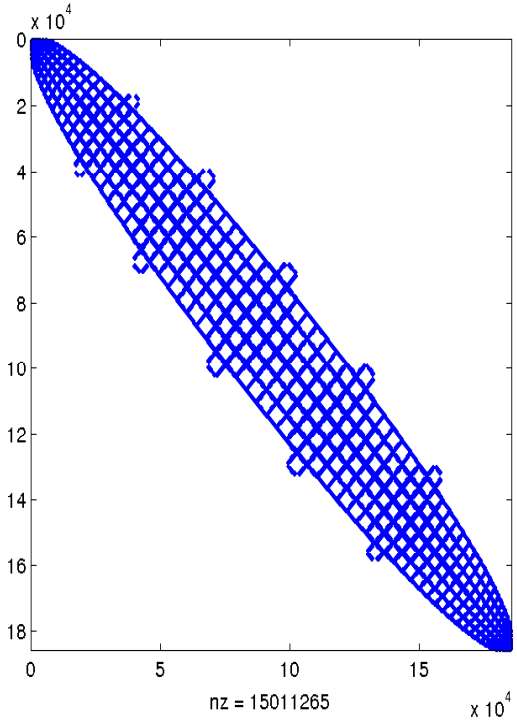
- We used the block variant of Lanczos (blocksize $\equiv 3$)
- Comparisons against Lanczos on CPU and GPU

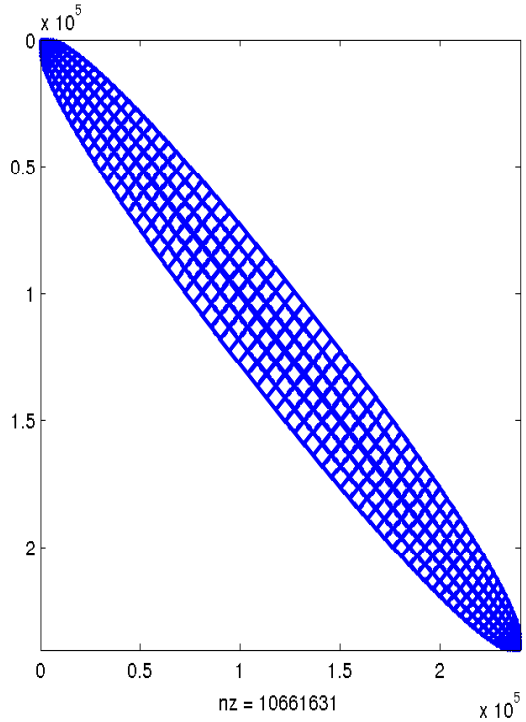
Test matrices: University of Florida Sparse Matrix Collection

- Taken from disciplines such as quantum mechanics, network analysis, and engineering

Table: A list of the matrices used to evaluate our GPU implementation, where n is the dimension of the matrix, nnz is the number of nonzero entries and $[\lambda_{\min}, \lambda_{\max}]$ is the spectral interval.

Matrix	n	nnz	nnz/n	Spectral interval
Ge87H76	112,985	7,892,195	69.9	[−1.21e+0, 3.28e+1]
Ge99H100	112,985	8,451,395	74.8	[−1.23e+0, 3.27e+1]
Si41Ge41H72	185,639	15,011,265	80.9	[−1.21e+0, 4.98e+1]
Si87H76	240,369	10,661,631	44.4	[−1.20e+0, 4.31e+1]
Ga41As41H72	268,096	18,488,476	69.0	[−1.25e+0, 1.30e+3]





Left: Sparsity pattern of $Si_{41}Ge_{41}H_{72}$. Right: Sparsity pattern of $Si_{87}H_{76}$.

Table: Computing the eigenpairs inside an interval using FLP with various filter polynomial degrees.

Matrix	interval	eigs	m	iters	MV	time	residual
			50	210	31,500	44	$4.3e{-}14$
Ge87H76|	$[-0.645, -0.0053]$	212	100	180	54,000	62	$6.4e{-}13$
			50	210	31,500	45	$3.7e{-}13$
Ge99H100|	$[-0.650, -0.0096]$	250	100	180	54,000	65	$4.0e{-}12$
			50	210	31,500	77	$3.2e{-}13$
Si41Ge41H72|	$[-0.640, -0.0028]$	218	100	180	54,000	112	$2.7e{-}11$
			50	150	22,500	55	$1.3e{-}14$
Si87H76|	$[-0.660, -0.3300]$	107	100	90	27,000	56	$3.3e{-}15$
			300	180	162,000	386	$3.2e{-}15$
Ga41As41H72|	$[-0.640, 0.0000]$	201	400	180	216,000	506	$8.1e{-}15$

Table: Percentage of total compute time required by various components of the algorithm.

Matrix	m	iters	PREPROC	ORTH	MV
Ge87H76	50	210	5%	15%	69%
	100	180	3%	8%	82%
Ge99H100	50	210	4%	15%	70%
	100	180	3%	8%	83%
Si41Ge41H72	50	210	7%	14%	70%
	100	180	5%	8%	83%
Si87H76	50	150	8%	16%	69%
	100	90	8%	8%	82%
Ga41As41H72	300	180	2%	3%	93%
	400	180	2%	2%	95%

GPU speedup over CPU (cucheb vs. FILTLAN)

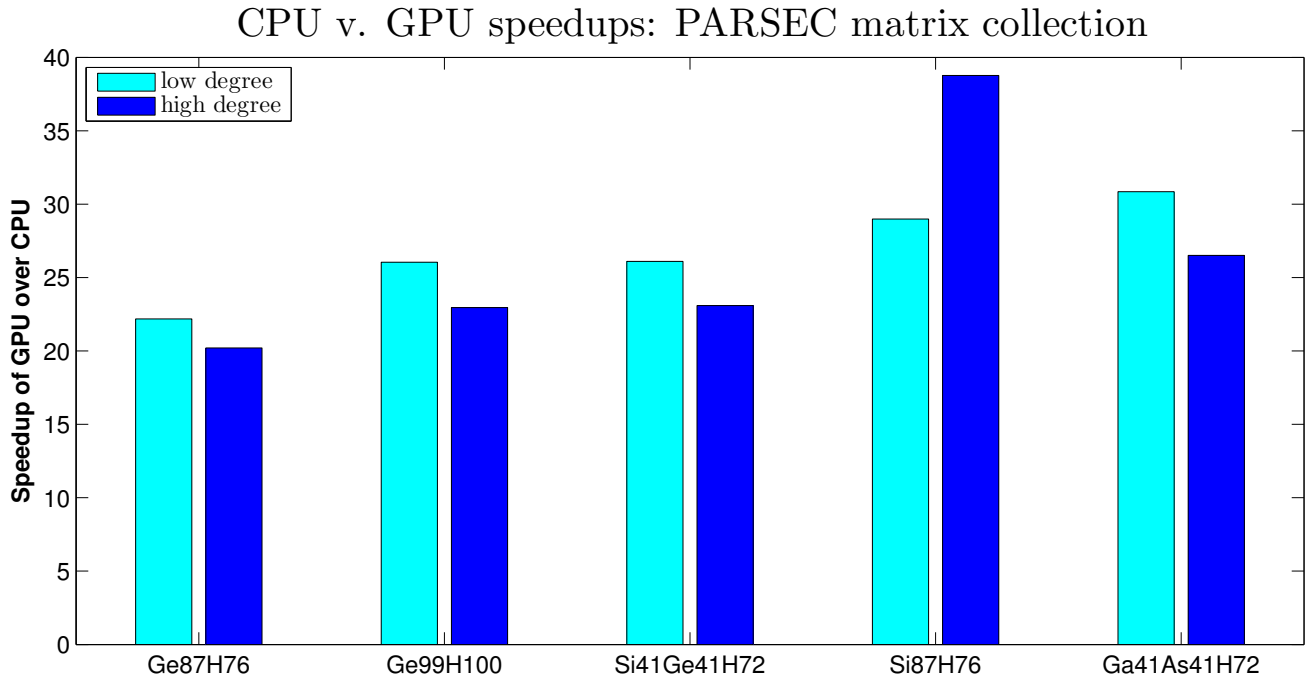


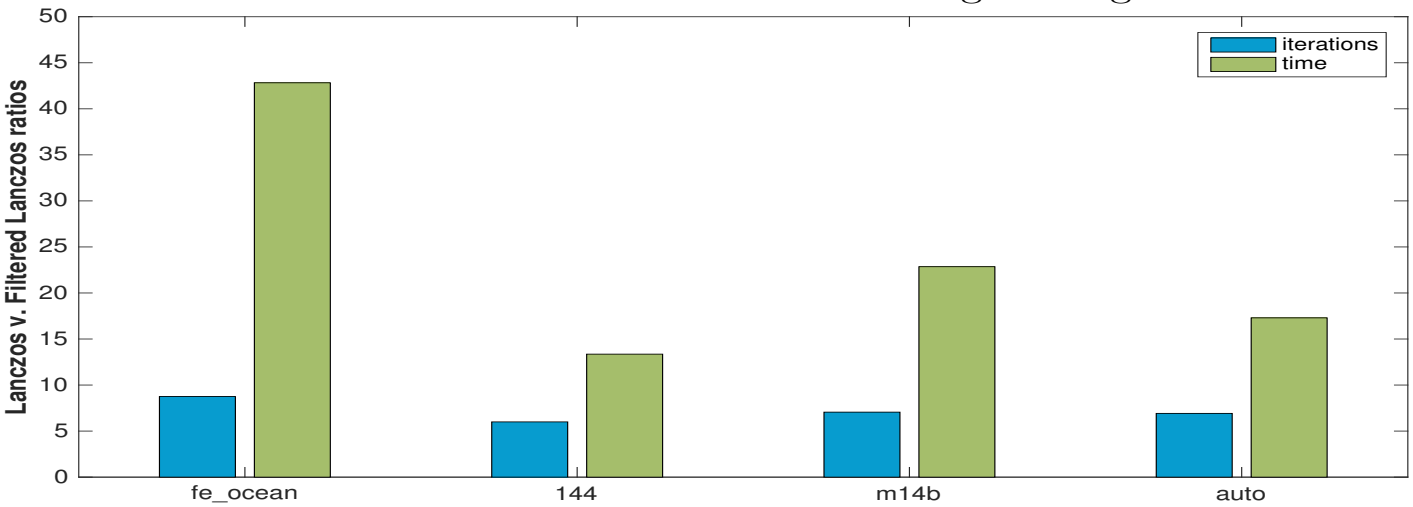
Figure: Low degree: m=50. High degree: m=100.

FILTLAN is a C++ sequential package (Lanczos with partial reor/ion, least-squares filters)

Table: Comparison of Lanczos ($m = 1$) and FLP method for matrices arising in engineering. For each matrix we computed all the eigenvalues in the specified fraction of the total spectral interval that included the upper bound.

Matrix	fraction	eigs	m	iters	MV	time	residual
`fe_ocean`	2%	217	47	480	22,560	19	$1.3e{-12}$
			1	4,200	4,200	803	$2.9e{-14}$
`144`	4%	195	33	450	14,850	19	$1.7e{-12}$
			1	2,700	2,700	252	$4.1e{-13}$
`m14b`	3%	235	38	510	19,380	37	$1.3e{-12}$
			1	3,600	3,600	833	$3.2e{-14}$
`auto`	4%	172	33	390	12,870	56	$1.8e{-11}$
			1	2,700	2,700	961	$3.5e{-14}$

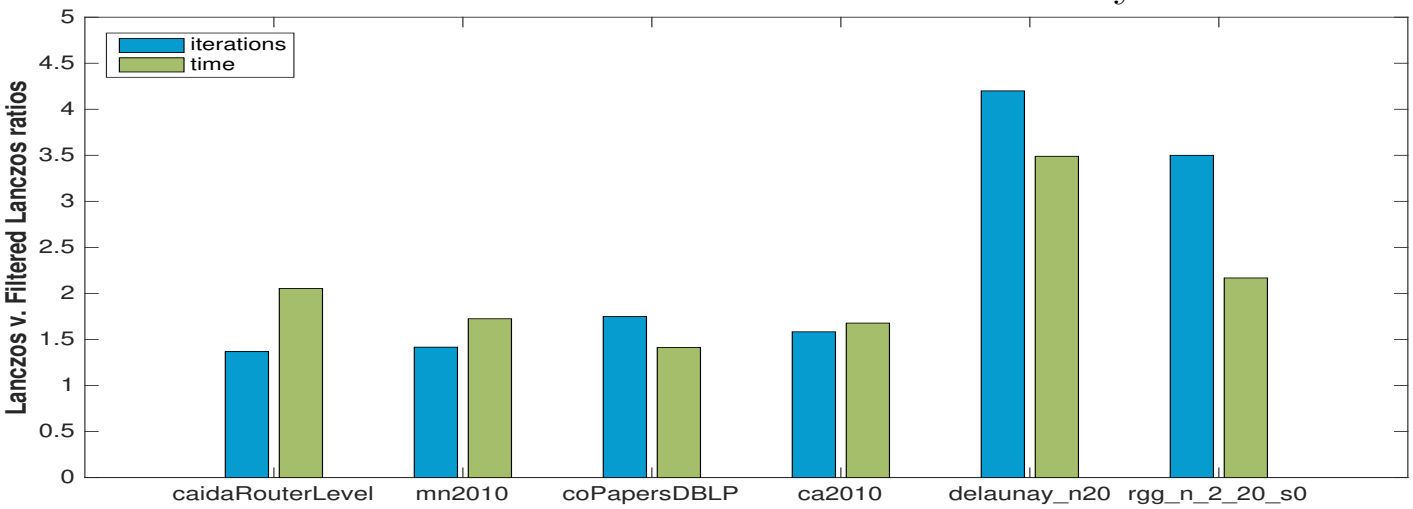
Lanczos v. Filtered Lanczos: engineering



Matrix	m	iters	ORTH	MV
fe_ocean	47	480	29%	25%
	1	4,200	35%	<1%
144	33	450	27%	43%
	1	2,700	47%	1%
m14b	38	510	25%	48%
	1	3,600	56%	<1%
auto	33	390	23%	56%
	1	2,700	73%	1%

Table: Comparison of Lanczos ($m = 1$) and FLP method for matrices arising in network analysis. For each matrix we computed all the eigenvalues in the specified fraction of the total spectral interval that included the upper bound.

Matrix	fraction	eigs	m	iters	MV	time	residual
caidaRouterLevel	42%	137	10	570	5,700	21	1.2e−12
			1	780	780	41	1.8e−14
mn2010	42%	79	10	360	3,600	13	3.0e−13
			1	510	510	21	1.9e−14
coPapersDBLP	50%	134	9	360	3,240	44	2.1e−11
			1	630	630	57	5.8e−14
ca2010	38%	103	10	360	3,600	34	5.2e−12
			1	570	570	52	2.3e−14
delaunay_n20	9%	35	22	150	3,300	27	1.8e−13
			1	630	630	82	9.8e−14
rgg_n_2_20_s0	15%	24	17	120	2,040	26	3.2e−10
			1	420	420	44	4.1e−8



Matrix	m	iters	ORTH	MV
caidaRouterLevel	10	570	47%	15%
	1	780	38%	1%
mn2010	10	360	52%	12%
	1	510	54%	1%
coPapersDBLP	9	360	32%	41%
	1	630	56%	6%
ca2010	10	360	54%	12%
	1	570	68%	1%
delaunay_n20	22	150	33%	19%
	1	630	69%	1%