Efficient Algorithms for Evaluating Matrix Polynomials

Sivan Toledo and Niv Hoffman
Blavatnik School of Computer Science
Tel-Aviv University

Oded Schwartz
Hebrew University

ACA 2017

Efficiency in Evaluation of $\mathbf{q(A) = c_0 I + c_1 A + c_2 A^2 + \cdots c_d A^d}$

A is an n-by-n real or complex matrix

Given A^ℓ, A^k,

- $c_k A^k$ requires n^2 scalar multiplications, but
- $A^{\ell+k} = A^\ell A^k$ requires n^3 (or $n^{2.81}$ with Strassen)

Minimize the number of matrix-matrix multiplications required to evaluate $q(A)$; these are called in the literature *nonscalar* multiplications

Maybe other ways to reduce arithmetic

Reduce communication (cache misses in this talk)

A Side Note on Functions of Matrices

When A is normal or close to normal, we can compute $q(A)$ or any other function $f(A)$ using an eigendecompotion,

$$\begin{aligned} q(A) &= q(V\Lambda V^*) \\ &= Vq(\Lambda)V^* \\ &= V \begin{pmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_n) \end{pmatrix} V^* . \end{aligned}$$

It is mostly when A is far from normal when these problems become interesting

Building Blocks

$q(A)$ via Explicit Powers

Allocate three matrices, A, A^k, and Q

Initialize $Q = 0$, and $A^k = I$

for $k \leftarrow 0, \ldots, d$, add $Q \leftarrow Q + c_k A^k$ and multiply $A^{k+1} \leftarrow AA^k$

In step d we produce $q(A) = Q$

A total of $d - 1$ matrix-matrix multiplications

$q(A)$ via Horner's Rule

Allocate two matrices, A and Q

Initialize $Q = c_{d-1}I + c_d A$

In step $k \leftarrow d-2, \ldots, 1, 0$, multiply and add $Q \leftarrow c_k I + QA$

In step 0 we produce $q(A) = Q$

Again a total of $d-1$ matrix-matrix multiplications

$q(A)$ via Is Roots

If the roots $\xi_1, \ldots, \xi_d$ are given, not the coefficients c, then

Allocate two matrices, A and Q

Initialize $Q = A - \xi_1 I$

For $k \leftarrow 2, \ldots, d$ add and multiply $Q \leftarrow Q(A - \xi_k I)$

In step d we produce $q(A) = Q$

Again a total of $d - 1$ matrix-matrix multiplications

(Computing the roots from the coefficients is not advised; often ill conditioned)

Clearly, Matrix Multiplication is Important Here

n^3 multiplications ($\approx 2n^3$ arithmetic operations) using native algorithms

$O(n^{\omega})$ using so-called *fast methods*; e.g., $O(n^{2.81})$ for Strassen-Winograd

$O(n^2 k^{\omega-2})$ to multiply an n-by-n matrix by an n-by-k (block mat-vec); no gain for $k = 1$!

$\times 6$ arithmetic speedup for triangular mat mult (plus can apply Strassen to square blocks within the recursion)

Does it Really Take $d - 1$ Matrix Multiplications?

Classical/Naive methods (e.g. Horner) require $d - 1$.

Is this really necessary?

d/2 Mat-Mults: Rabin-Winograd/Patterson-Stockmeyer

No (Rabin-Winograd 1971, Patterson-Stockmeyer 1973).

$$
\begin{aligned}
q(A) &= c_0 I + c_1 A + c_2 A^2 + \cdots + c_{2r-1} A^{2r-1} \\
&= c_0 I + \cdots + c_{r-1} A^{r-1} + A^r(\tilde{c}_0 I + \cdots + \tilde{c}_{r-1} A^{r-1})
\end{aligned}
$$

use recursion and denote # mat-mults by $N(d)$ (repeated squaring for A^r). For powers of two,

$$
\begin{aligned}
N(2r-1) &= 2N(r-1) + 1 \\
\text{total (inc. repeated squaring)} &\approx d/2 + \log(d)
\end{aligned}
$$

for non powers of two, split into powers of two so

$$
N(d) \leq d/2 + 2\log(d)
$$

The Algorithm of Paterson & Stockmeyer

Consider an example,

$$
\begin{aligned}
q(A) &= 2I + 3A + 4A^2 \\
&\quad +5A^3 + 6A^4 + 7A^5 \\
&\quad +8A^6 + 9A^7 + 2A^8 \\
&= \left(2I + 3A + 4A^2\right) \\
&\quad + \left(5A^3 + 6A^4 + 7A^5\right) \\
&\quad + \left(8A^6 + 9A^7 + 2A^8\right) \\
&= \left(2I + 3A + 4A^2\right) I \\
&\quad + \left(5I + 6A + 7A^2\right) A^3 \\
&\quad + \left(8I + 9A + 2A^2\right) \left(A^3\right)^2
\end{aligned}
$$

Analysis of the Example

Consider an example,

$$
\begin{aligned}
q(A) \quad = \quad & \left(2I + 3A + 4A^2\right) I \\
& + \left(5I + 6A + 7A^2\right) A^3 \\
& + \left(8I + 9A + 2A^2\right) \left(A^3\right)^2
\end{aligned}
$$

We perform 1 matmult to produce I, A, A^2, 1 to produce A^3, 1 square it, and 2 more to multiply degree-2 polynomials by powers of A^3.

A polynomial in A^3 whose coefficients are quadratics in A.

The General Case of the PS Algorithm

Let $ps = d$ (remainder is easy to handle),

$$\begin{aligned}
q(A) &= c_0 I + c_1 A + c_2 A^2 + \cdots + c_d A^d \\
&= c_0 I + c_1 A + \cdots + c_{p-1} A^{p-1} \\
&\quad + \left(c_p I + c_{p+1} A + \cdots + c_{2p-1} A^{p-1}\right) A^p \\
&\quad + \cdots \\
&\quad + \left(c_{d-p+1} I + c_{d-p+2} A + \cdots + c_d A^{p-1}\right) \left(A^p\right)^{s-1} \\
&= c_0 I + c_1 A + \cdots + c_{p-1} A^{p-1} \\
&\quad + \left(c_p I + c_{p+1} A + \cdots + c_{2p-1} A^{p-1}\right) A^p \\
&\quad + \cdots \\
&\quad + \left(c_{(s-1)p} I + c_{(s-1)p+1} A + \cdots + c_{(s-1)p+p-1} A^{p-1}\right) \left(A^p\right)^{s-1}
\end{aligned}$$

Arithmetic and Memory Complexity of PS

Let $ps = d$ (remainder is easy to handle),

$$
\begin{aligned}
q(A) \quad = \quad & \left(c_0 I + c_1 A + \cdots + c_{p-1} A^{p-1}\right) \\
& + \left(c_p I + c_{p+1} A + \cdots + c_{2p-1} A^{p-1}\right) A^p \\
& + \cdots \\
& + \left(c_{(s-1)p} I + c_{(s-1)p+1} A + \cdots + c_{(s-1)p+p-1} A^{p-1}\right) \left(A^p\right)^{s-1}
\end{aligned}
$$

Form and store $A^2, \ldots A^{p-1}, A^p$ explicitly ($p-1$ MMs, $p+1$ matrices to store)

Set $Q \leftarrow$ highest coefficient polynomial $= \sum_{\ell=0}^{p-1} c_{\ldots} A^\ell$ ($p-1$ scale-add, 0 MMs)

For $k \leftarrow s-1, \ldots, 1, 0$ multiply and add
$Q \leftarrow QA^p + \sum_{\ell=0}^{p-1} c_{\ldots} A^\ell$ (s m-multiplications)

Parameter Optimization for PS

Total number of matrix multiplications is $p - 1 + s$ where $s = \lceil d/p \rceil - 1$

Therefore $ps \approx d$ so $p + s - 1$ is minimized near $p \approx \sqrt{d}$ at about $2\sqrt{d}$ MMs

Can we go even lower?

No. Paterson & Stockmeyer proved that there are qs for which # of MMs is at least $\sqrt{d}$

($\sqrt{d} - 1/2$ MMs if the coefficients are integers), under some reasonable assumptions

A Comparison

	Work (# MMs)	Memory (# words)
Explicit Powers	$d-1$	$3n^2$
Horner's Rule	$d-1$	$2n^2$
MM PS	$p-1+s$	$(p+1)\,n^2$
MM PS for $ps \approx d$	$\approx 2\sqrt{d}$	$\approx \sqrt{d}n^2$

A work-storage tradeoff

Memory-Efficient Matrix-Vector PS (Van Loan 1979)

$$q(A) = \left(2I + 3A + 4A^2\right) I + \cdots + \left(8I + 9A + 2A^2\right) \left(A^3\right)^2$$

So the jth colomn is

$$\begin{aligned} q(A)_{:,j} &= \left(I\left(2I + 3A + 4A^2\right)\right)_{:,j} \\ &\quad + \left(A^3\left(5I + 6A + 7A^2\right)\right)_{:,j} \\ &\quad + \left(\left(A^3\right)^2\left(8I + 9A + 2A^2\right)\right)_{:,j} \\ &= I\left(2Ie_j + 3Ae_j + 4A^2e_j\right) \\ &\quad + A^3\left(5e_j + 6Ae_j + 7A^2e_j\right) \\ &\quad + \left(A^3\right)^2\left(8e_j + 9Ae_j + 2A^2e_j\right) \end{aligned}$$

Matrix-Vector SPH (Van Loan 1979)

$$
\begin{aligned}
q(A)_{:,j} \quad &= \quad I\left(2Ie_j + 3Ae_j + 4A^2e_j\right) \\
&\quad +A^3\left(5e_j + 6Ae_j + 7A^2e_j\right) \\
&\quad +\left(A^3\right)^2\left(8e_j + 9Ae_j + 2A^2e_j\right)
\end{aligned}
$$

Compute A^p ($\log_2 p$ m-multiplications, can use Strassen)

For $j \leftarrow 1, \ldots, n$

Compute $Ae_j, \ldots, A^{p-1}e_j$ ($p-1$ matvecs)

Set $Q_j \leftarrow \sum_{\ell=0}^{p-1} c_{\ldots} A^{\ell} e_j$ (vec ops)

For $k \leftarrow s-1, \ldots, 1, 0$ multiply and add
$Q_j \leftarrow A^p Q_j + \sum_{\ell=0}^{p-1} c_{\ldots} A^{\ell} e_j$ (s MVs, vec ops)

$n(p-1) + ns$ matvecs $\equiv s + p - 1$ (convensional, no Strassen) MMs

A Comparison (now with Van Loan's schedule)

	Work (# MMs)	Memory (# words)
Explicit Powers	$d - 1$	$3n^2$
Horner's Rule	$d - 1$	$2n^2$
MM PS	$p - 1 + s$	$(p + 1)\, n^2$
MV PS	$\log_2 p + (s + p)_{\text{conv}}$	$3n^2 + pn$

Communication (Cache-Miss) Analysis

Communications Analysis in Two-Level Memories

We assume that memory consists of a large slow memory and a fast memory (cache) of size M, and count the number of accesses to slow memory (cache misses) under an ideal replacement policy.

- Any mat-mult, $M > 2n^2$, only $\Theta(n^2)$ compulsory misses
- Conventional mat-mult, $M < n^2/2$, $\Theta(n^3/\sqrt{M})$ misses (blocking, recursion)
- Fast mat-mult, small cache, $O(n^\omega/M^{\omega/2-1})$ misses (less work, smaller data-reuse ratio)

Cache Misses in MM PS

If $M > (p+1)n^2$, only $\Theta(n^2)$ compulsory misses

If $M < n^2/2$,

$$\Theta(\frac{(p+s)n^3}{\sqrt{M}} + spn^2) = \Theta(\frac{(p+d/p)n^3}{\sqrt{M}} + dn^2)$$

This is minimized for $p \approx \sqrt{d}$ so $\Theta(\sqrt{d}n^3/\sqrt{M} + dn^2)$

- $\Theta(\sqrt{d}n^3/\sqrt{M})$ for $M \le n^2/d$ (tiny cache, MMs dominate)
- $\Theta(dn^2)$ for $n^2/d \le M < n^2/2$ (small cache, matrix scale-adds dominate)

Mind the gap

Cache Misses in MM PS: Filling the Gap

for $3n^2 \leq M \leq (p+1)n^2$ MMs are comm-cheap but MSAs are not

Number of cache misses $\Theta(spn^2) = \Theta(dn^2)$

Cache misses minimized by setting $p \approx M/n^2$, to get out of this regime

But arithmetic is not; yuck

Cache Misses in the MV PS

If $M > 3n^2 + pn$, only $\Theta(n^2)$ compulsory misses

If $pn \leq M < n^2/2$, $\Theta(\log p \cdot n^3/\sqrt{M})$ to generate A^p, plus n iterations in which we pay $\Theta(pn^2)$ to construct $Ae_j, \ldots, A^{p-1}e_j$ and $\Theta(sn^2)$ for s MVs with A^p

In total,

$$\begin{aligned}\Theta\left(\frac{\log p \cdot n^3}{\sqrt{M}} + n(p+s)n^2\right) &= \Theta\left(\frac{\log p \cdot n^3}{\sqrt{M}} + \left(p + \frac{d}{p}\right)n^3\right) \\ &= \Theta\left(\left(p + \frac{d}{p}\right)n^3\right)\end{aligned}$$

Picking a (Theoretical) Winner

If $M > (p+1)n^2$, use MM-PS (Strassen okay)

If $M > 3n^2 + pn$, use MV-PS (optimal arithmetic sans Strassen)

Otherwise (tiny cache), use MM-PS

But we can do better with new variants

New Algorithmic Ideas

1: Block-Column PS

Compute A^p ($\log_2 p$ m-multiplications)

Pick a block size b

For n/b blocks $E_j \leftarrow I_{:,j:j+b}$

Compute $AE_j, \ldots, A^{p-1}E_j$ ($p-1$ b-mat-vecs)

Set $Q_j \leftarrow \text{(highest coefficient polynomial)}\, E_j$ (b-vec scale-add)

For $k \leftarrow s-1, \ldots, 1, 0$ multiply and add $Q_j \leftarrow A^p Q_j + \text{(next lower coefficient polynomial)}\, E_j$

$\Theta\left(\frac{\log p \cdot n^3}{\sqrt{M}} + \frac{n}{b}(p+s)n^2\right)$ cache misses for $pnb \leq M \leq n^2/2$

2: Use Fast Matrix Multiplication in MM-PS, BMV-PS

Great opportunity for fast mat-mult

Benefit in block-MV PS drops with block size (both arithmetic and data-reuse ratio)

3: Tranform to (Real) Schur Form

If $A = VTV^*$, then $q(A) = Vq(T)V^*$, so the expensive part ($> O(n^3)$) is performed on triangular matrices, $\times 6$ arithmetic benefit

$\sqrt{d}$ needs to be high enough to offset the cost of the Schur decomposition

Unfortunately, level-3 BLAS do not have triangular-triangular matrix multiplication, but can implement fairly easily using recursion

4: Schur Form, Reorder, Parlett+Davies-Higham

If $A = VTV^*$, then $q(A) = Vq(T)V^*$

Apply block Schur-Parlett substitution to $q(T)$ (evaluate q on diagonal blocks, solve Sylvester equations for off-diagonal blocks)

Higham-Davies: ensure Sylvester equations are well conditioned by partitioning $\Lambda(A)$ into well-separated clusters & reordering the Schur form (Bai-Demmel-Kressner)

Benefit: $\Theta(\sqrt{d}n^3)$ super-cubic algorithm applied to diagonal blocks of T, not all of it

Cost: $O(n^3)$ Schur decomposition and reordering

[Not implemented, but shows how novel discoveries can improve old algorithms]

5: Remainder Evaluation (Really an Open Problem)

If $d > n$, let χ be the Characteristic polynomial of A and let

$$q(A) = \chi(A)\delta(A) + \rho(A) = \rho(A) \ .$$

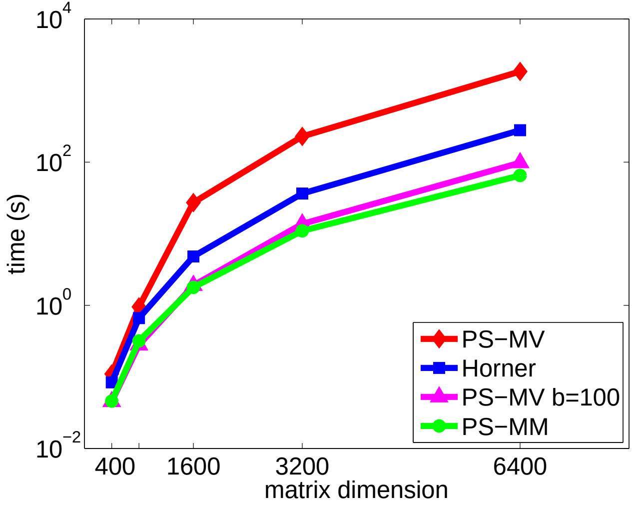
Clearly, evaluating $\rho(A)$ is cheaper than applying $q(A)$

But can we determine the coefficients of ρ in a stable way?
Open problem (AFAIK)

I'll mention another interesting and related open problem at the end of the talk

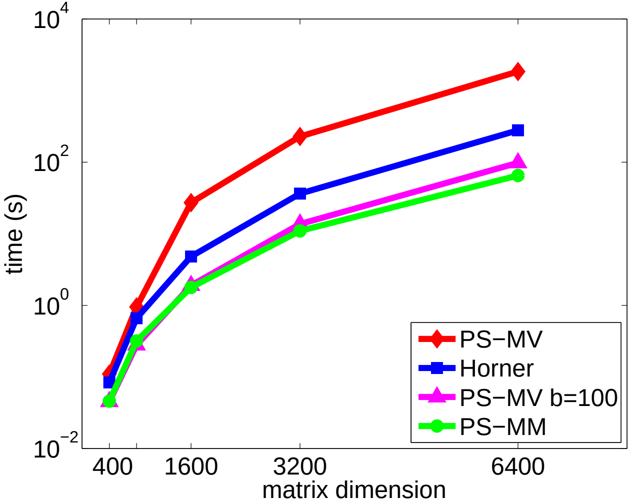
Experimental Results

d = 100; MM-PS and Block-MM-PS Better than Horner

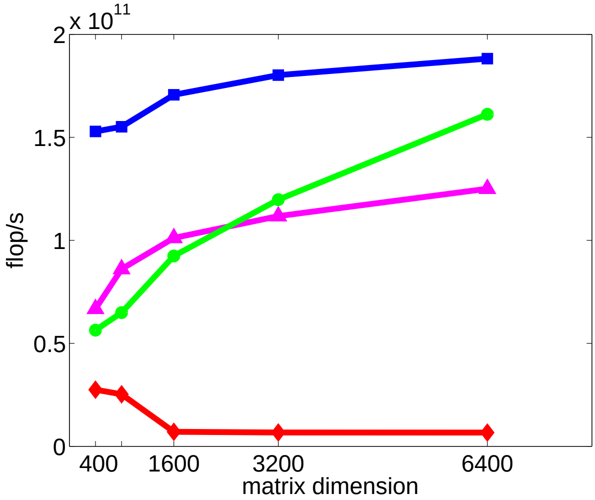


Quad-core
i7
C+MKL

MV PS is Terrible! Cache Misses!

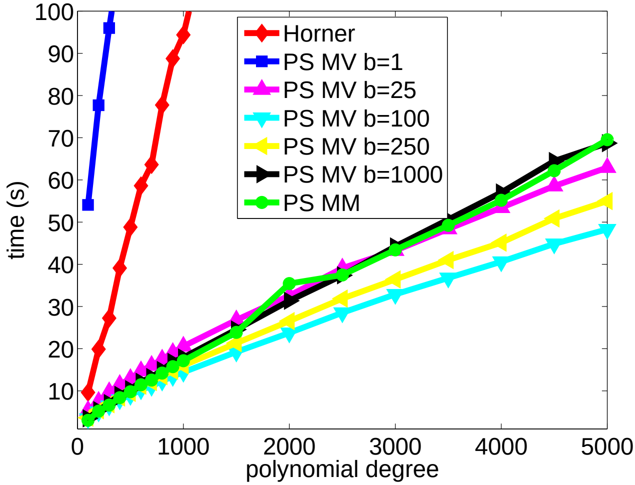


Flop/s Rates



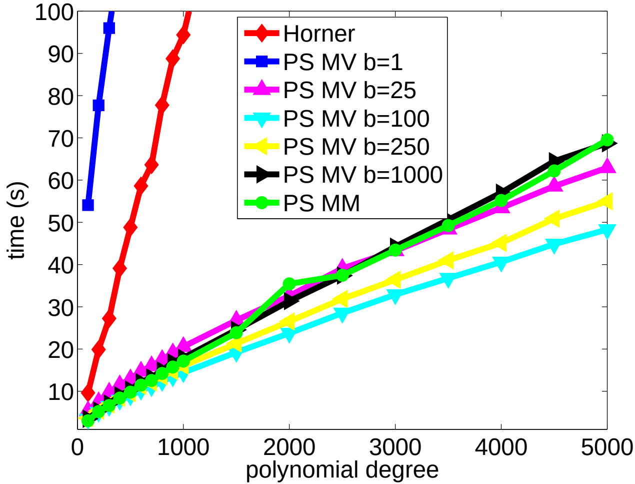
Block-Size Sensitivity (Not High but Not Monotone)

$n = 2000$



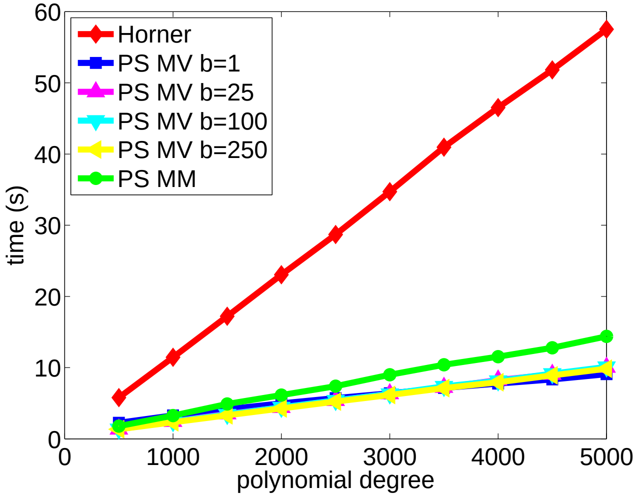
Also, For High Degrees Horner is Terrible

$n = 2000$

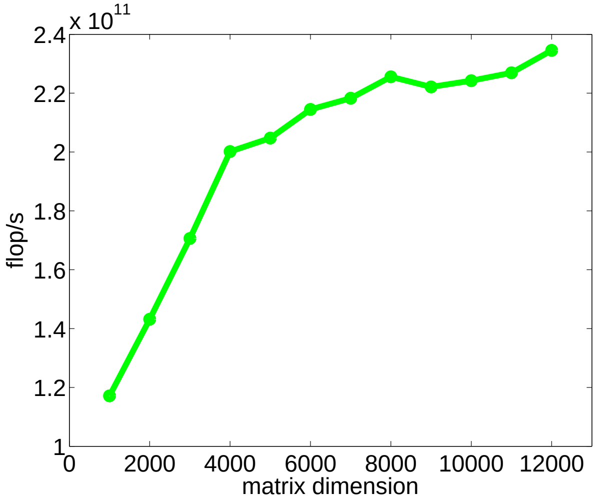


Reduced Sensitivity at Smaller Sizes

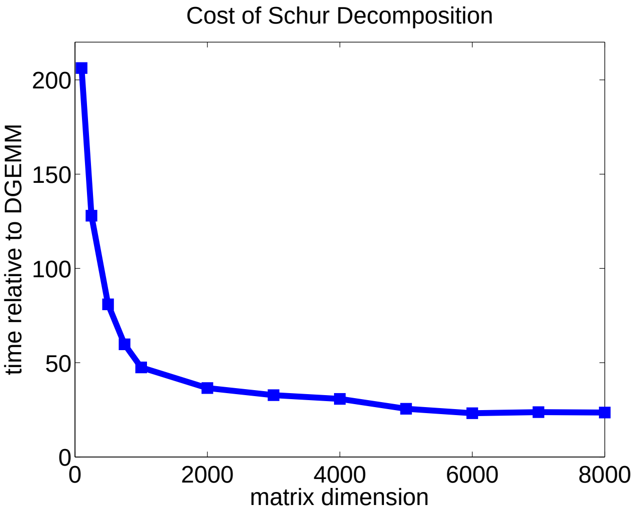
$n = 750$



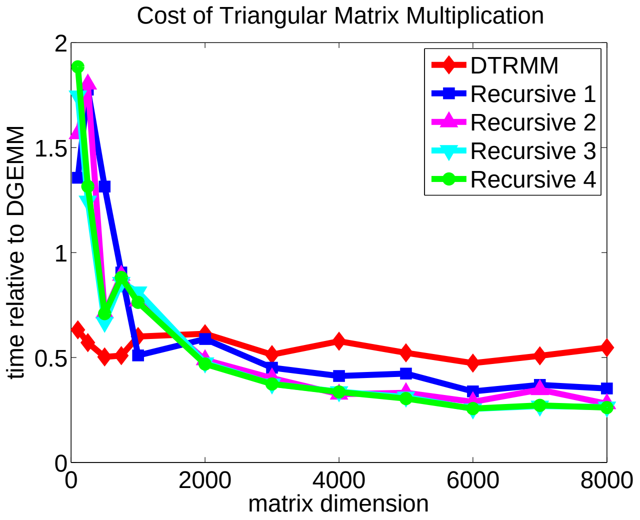
d = 10 Emphasizes Cost of Matrix Additions



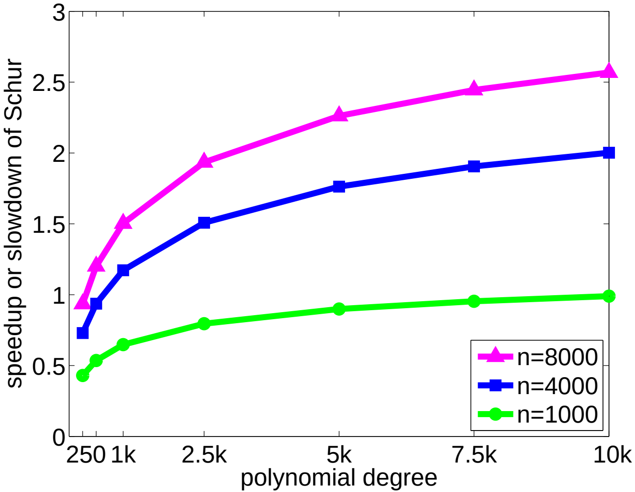
The Cost of the Schur Decomposition ($\approx$ 25 Mat-Mults)



Savings from Triangular Matrix Multiplication ($\approx \times 4$)



Estimating The Cost of Schur + Triangular PS ($> \times 2.5$)



Conclusions and Open Problems

Conclusions

Old algorithms must be revisited (fairly obvious);

MM-PS is memory inefficient, MV-PS super slow due to cache misses

Block MV-PS fixes that

More novel tricks thanks to various innovations, mostly in extreme cases (very high degrees)

Open Problem 1: Applications?

Do these algorithms have interesting applications?

We were attracted to this due to the complex tradeoffs in Patterson-Stockmeyer variants

Seems that rational approximations are used more often in applications

Open Problem 2: Alternate Representations

Numerical analysis of Patterson-Stockmeyer in other (more useful) representations of q

Is there an efficient and numerically-stable version of PS for $d > n$? Think of $\rho(A)$ given in terms of the coefficients of $q(A) = \chi(A)\delta(A) + \rho(A)$

Is there an efficient and numerically-stable version of PS for Newton polynomials, or for any other forms of interpolation/least-squares polynomials?

Looking forward to discussions during the rest of the week!