KU LEUVEN

Communication efficient sequences of rotations

BLIS retreat 2024

Thijs Steel, Julien Langou
KU Leuven

September 2024

0 Outline

1 Outline

1. Introduction
2. Optimizing rotation sequences

1 Why Rotation sequences?

Algorithms the use rotation sequences

- implicit QR (symmetric)
- implicit QR (svd)
- implicit QR (nonsymmetric)
- QZ
- Jacobi SVD
- Hessenberg-triangular reduction
- ...

1 Rotation sequence

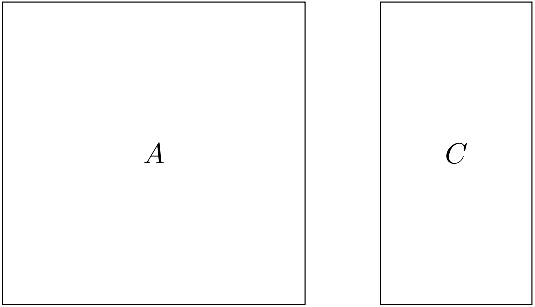
Rotation sequence

Given an $m \times n$ matrix A and two $n-1 \times k$ matrices C and S containing the cosines and sines of rotations. Apply each rotation (i, j) to columns i and $i+1$ of A, respecting the order: $(i, j) \to (i+1, j)$ and $(i, j) \to (i-1, j+1)$.

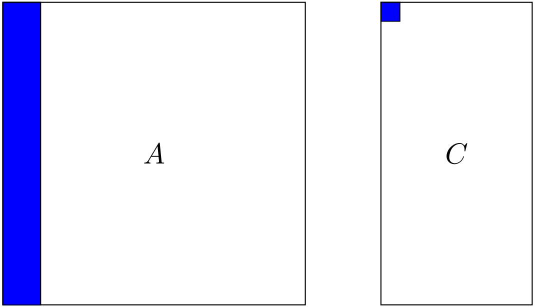
Pseudocode

```
for p = 0, ..., k − 1:
    for j = 0, ..., n − 1:
        for i = 0, ..., m − 1:
            A(i, j : j + 1) = A(i, j : j + 1) ∗ G(i, j)
```

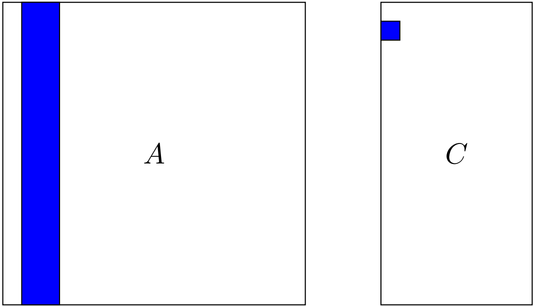
1 Rotation sequence



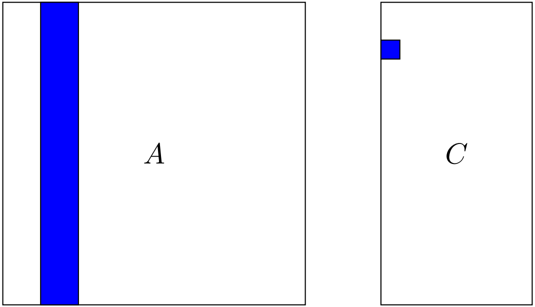
1 Rotation sequence



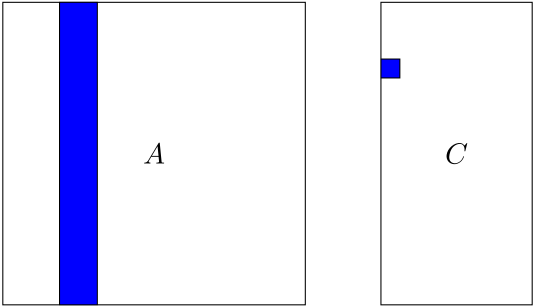
1 Rotation sequence



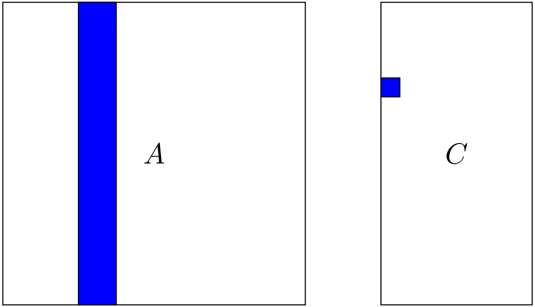
1 Rotation sequence



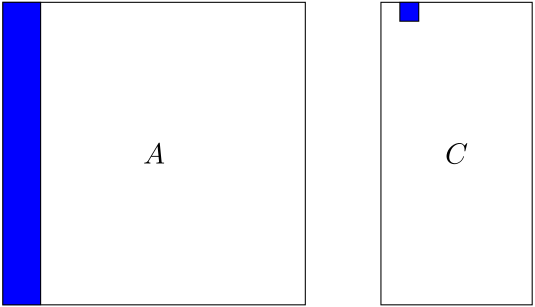
1 Rotation sequence



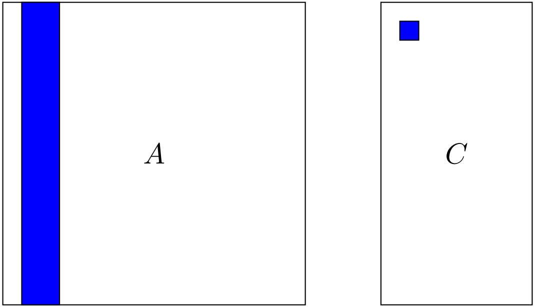
1 Rotation sequence



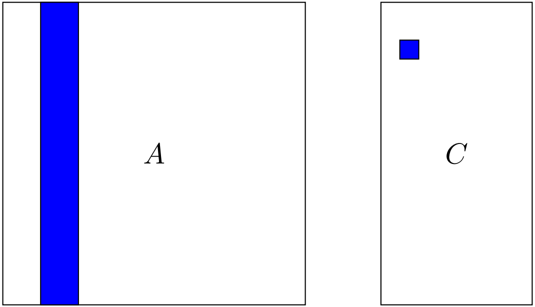
1 Rotation sequence



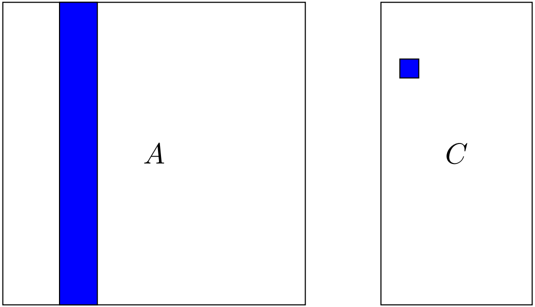
1 Rotation sequence



1 Rotation sequence



1 Rotation sequence



1 Rotation sequence

Rotation sequence variants

- Apply rotations in reverse order
- Apply rotations to rows instead of columns
- Account for trapezoidal structure in C and S
- Apply (small) reflections instead of rotations

2 Outline

2 Accumulating rotations Braman et al. (2002)

Algorithm

1. Accumulate $k \times k$ rotations into $2k \times 2k$ orthogonal matrix.
2. Apply $2k \times 2k$ orthogonal matrix to A using optimized BLAS.

Cost

1. Normal rotations: $6mk^2$ flops
2. Accumulate + GEMM + TRMM $\approx 3k^3 + 6mk^2$ flops
3. If $k << m$, most flops are in GEMM and TRMM.

2 Fusing rotationsKågström et al. (2008)

- Apply multiple rotations in one loop.
- Reuse values in register $\rightarrow$ less memory operations.

For $i = 1, \ldots, m$:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = G_1 * \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

For $i = 1, \ldots, m$:

$$\begin{bmatrix} y_i \\ z_i \end{bmatrix} = G_2 * \begin{bmatrix} y_i \\ z_i \end{bmatrix}$$

For $i = 1, \ldots, m$:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = G_1 * \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\begin{bmatrix} y_i \\ z_i \end{bmatrix} = G_2 * \begin{bmatrix} y_i \\ z_i \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} \mathbf{g_{1,1}} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & \mathbf{g_{1,2}} & g_{1,3} \\ \mathbf{g_{2,1}} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & \mathbf{g_{1,3}} \\ g_{2,1} & \mathbf{g_{2,2}} & g_{2,3} \\ \mathbf{g_{3,1}} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & \mathbf{g_{2,3}} \\ g_{3,1} & \mathbf{g_{3,2}} & g_{3,3} \\ \mathbf{g_{4,1}} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & \mathbf{g_{3,3}} \\ g_{4,1} & \mathbf{g_{4,2}} & g_{4,3} \\ \mathbf{g_{5,1}} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & \mathbf{g_{4,3}} \\ g_{5,1} & \mathbf{g_{5,2}} & g_{5,3} \\ \mathbf{g_{6,1}} & g_{6,2} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & \mathbf{g_{5,3}} \\ g_{6,1} & \mathbf{g_{6,2}} & g_{6,3} \end{bmatrix}$$

2 Wavefront pattern Van Zee et al. (2014)

Cache efficiency

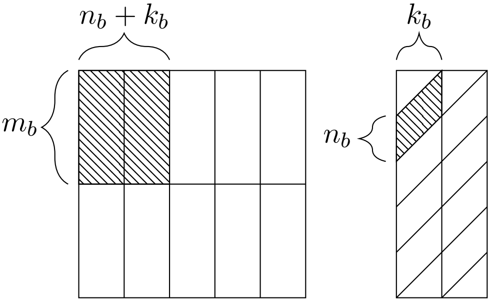
- Normal pattern: access n columns of A before reusing.
- Wavefront pattern: access k columns of A before reusing.
- Higher likelihood of cache hits.

Order of the rotations

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \\ g_{5,1} & g_{5,2} & g_{5,3} \\ g_{6,1} & g_{6,2} & \mathbf{g_{6,3}} \end{bmatrix}$$

2 Blocking

- Split into $m_b \times n_b \times k_b$ blocks.
- Each block fits in cache.
- Overlap between blocks $\rightarrow$ reuse.

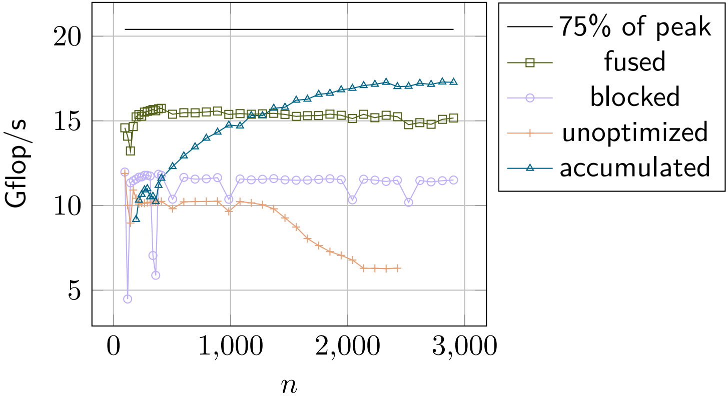


2 Rotations are limited to 75% peak

- Rotation: $4n$ multiplications and $2n$ additions.
- Can't always use FMA instructions.

2 Results

- $k = 180$, varying n, $m = n$
- Xeon Gold E5-2650 V2, 2.6 GHz



2 Reuse accross loop iterations

Normal rotation

for $j = 0, 1, \ldots, n-1$
    Load $c[j]$ and $s[j]$ into registers
    for $i = 0, 1, \ldots, m-1$
        load $A(i, j)$ and $A(i, j+1)$ into registers
        apply rotation to $A(i, j)$ and $A(i, j+1)$
        store $A(i, j)$ and $A(i, j+1)$

$c[j]$ and $s[j]$ are reused.

2 Reuse accross loop iterations

Vectorization

- $$\begin{bmatrix} C(j,p) \\ C(j,p) \\ C(j,p) \\ C(j,p) \end{bmatrix} * \begin{bmatrix} A(i,j) \\ A(i+1,j) \\ A(i+2,j) \\ A(i+3,j) \end{bmatrix} + \begin{bmatrix} S(j,p) \\ S(j,p) \\ S(j,p) \\ S(j,p) \end{bmatrix} * \begin{bmatrix} A(i,j+1) \\ A(i+1,j+1) \\ A(i+2,j+1) \\ A(i+3,j+1) \end{bmatrix}$$
- c and s are broadcast.
- Reusing $A(i,j)$ instead of c and s leads to much more reuse.

2 Reuse accross loop iterations

Shifting kernel

```
for $i = 0, 1, \dots, m-1$
    load $A(i,0)$ into registers
    for $j = 0, 1, \dots, n-1$
        Load $A(i,j+1)$, $c[j]$ and $s[j]$ into registers
        apply rotation to $A(i,j)$ and $A(i,j+1)$
        store $A(i,j)$
    store $A(i,n-1)$
```

$A(i,j)$ is reused.

2 Combined loop reuse with fused rotations

Full kernel

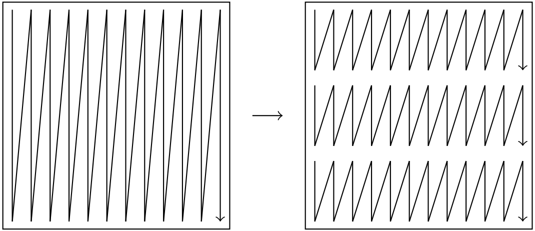
- m_r rows of A.
- fuse waves of k_r rotations.
- shuffle to apply n_b of these waves.
- 16 AVX registers $\rightarrow m_r = 8$ and $k_r = 5$.

Memops

- No reuse: $6mnk$ memops
- 2×2 fusing + reuse c and s: $2mnk$ memops
- 8×5 shuffling kernel: $0.65mnk$ memops (arithmetic intensity of 9.23!!!)

2 Packing

- blocking and shuffling lead to more reuse, but access is strided.
- Solution: pack matrix into packed format.
- In many algorithms, we can keep the matrix in packed format.

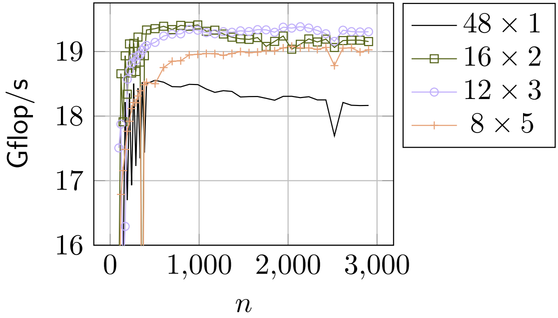


2 Parallelization

- Application of rotations to different rows of A is independent, fully parallelizable.
- Other loops are difficult to parallelize.

2 Results

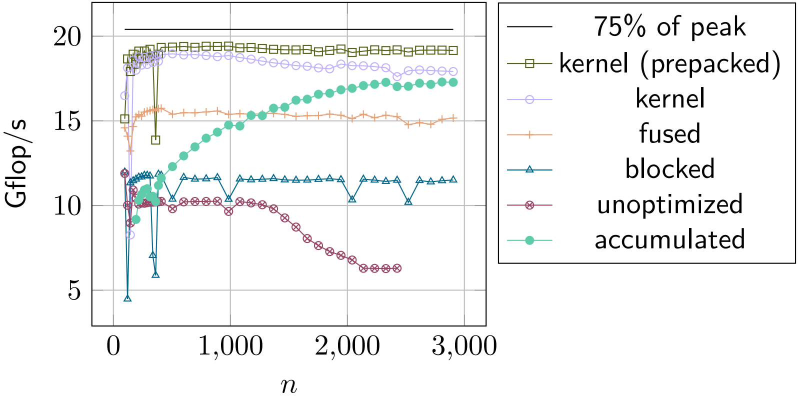
- $k = 180$, varying n, $m = n$
- 2 Xeon Gold E5-2650 V2, 2.6 GHz
- Test different kernels



KU LEUVEN

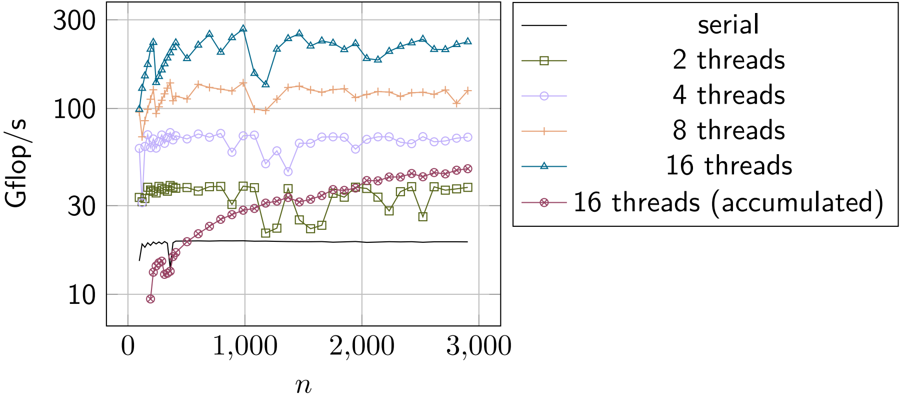
2 Results

- $k = 180$, varying n, $m = n$
- Xeon Gold E5-2650 V2, 2.6 GHz



2 Results

- $k = 180$, varying n, $m = n$
- 2 Xeon Gold E5-2650 V2, 2.6 GHz
- Up to 16 cores



2 Conclusion

Results

- Shifting kernel leads to better register reuse
- Blocking, with tuning of block size for multiple cache levels leads to better cache reuse
- Full algorithm can achieve 75% of peak performance

Future work

- Optimize reflector sequences
- Use rotation sequences in QR, QZ, SVD, ...

3 References I

Braman, K., Byers, R., and Mathias, R. (2002). The multishift qr algorithm. part i: Maintaining well-focused shifts and level 3 performance. *SIAM Journal on Matrix Analysis and Applications*, 23(4):929–947.

Kågström, B., Kressner, D., Quintana-Ortí, E., and Quintana-Orti, G. (2008). Blocked algorithms for the reduction to Hessenberg-triangular form revisited. *BIT Numerical Mathematics*, 48:563–584.

Van Zee, F. G., Van de Geijn, R. A., and Quintana-Orti, G. (2014). Restructuring the tridiagonal and bidiagonal QR algorithms for performance. *ACM Transactions on Mathematical Software (TOMS)*, 40(3):1–34.