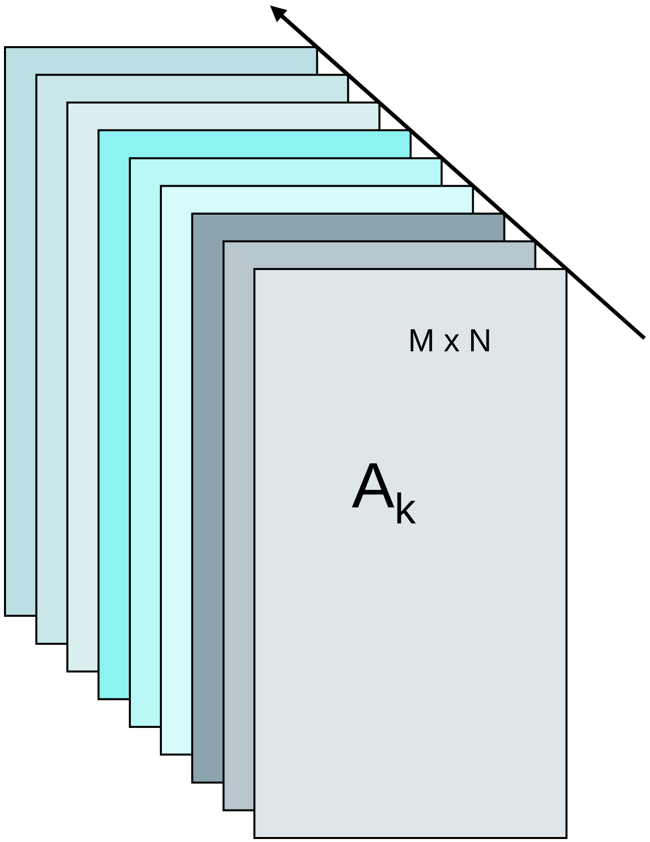
Successive Rank-Revealing Cholesky Factorizations on GPUs

Ty Fridrich
Nikos P. Pitsianis
Xiaobai Sun
CS Dept., Duke Univ.
HPEC-2006, 09/19-21/2006
MIT-LL

Problem Description: Successive Cholesky Factorizations

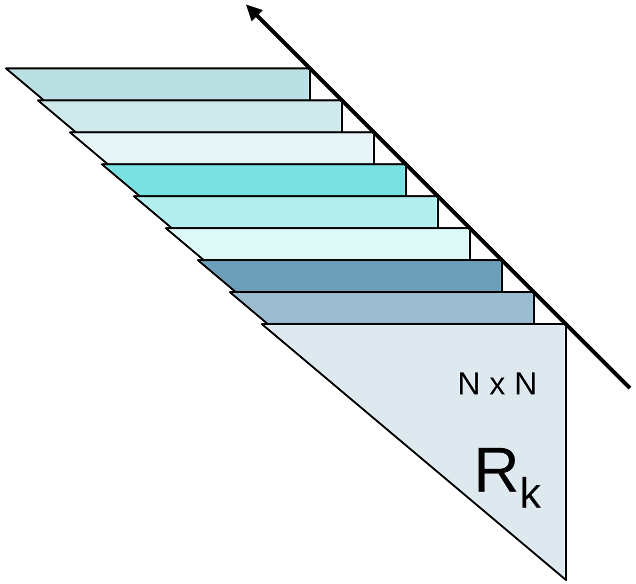
- A sequence of data matrices at input

$$A_k \in R^{m \times n}, \quad k = 1, 2, \cdots$$



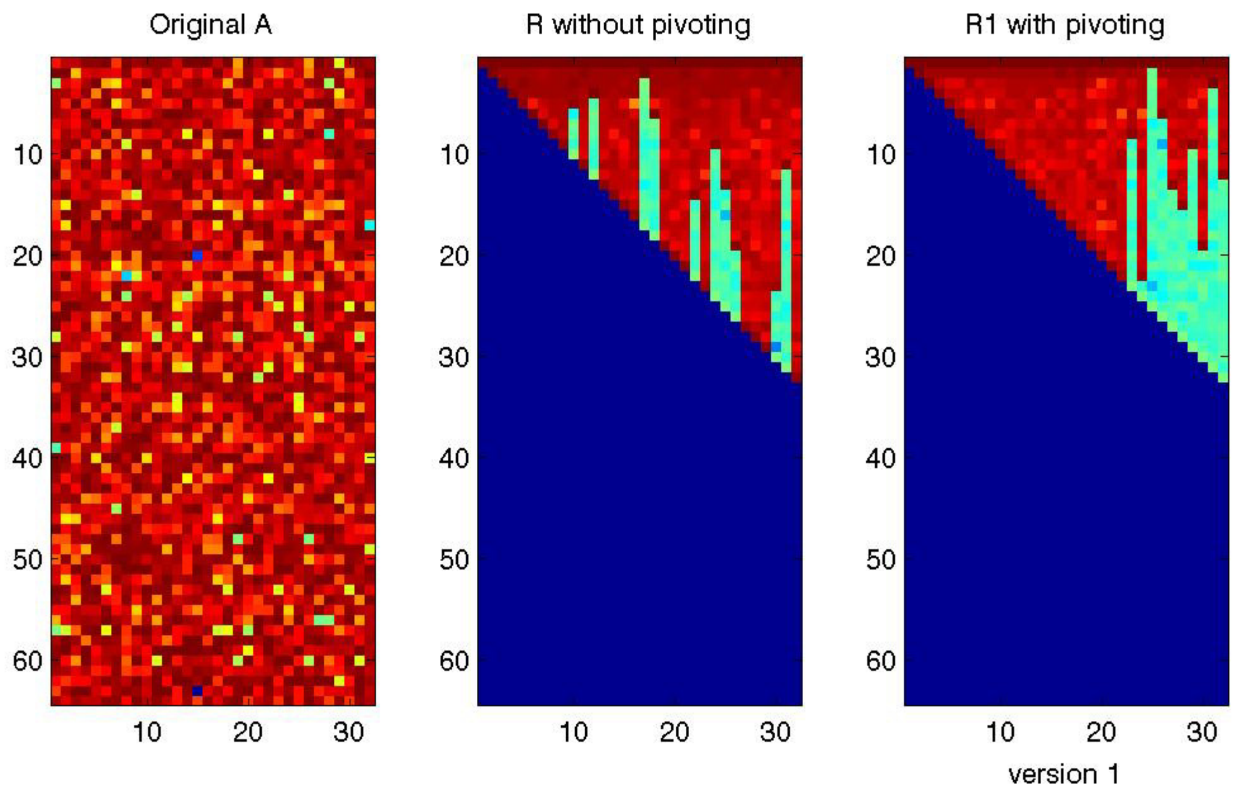
- A sequence of Cholesky factors at output

$$R_k^T R_k = A_k^T A_k$$



- To increase the rate of generating the Cholesky factors in space-time adaptive processing (**STAP**) systems

Successive *Rank-Revealing* Cholesky Factorizations



$$R_k^\top R_k = A(:,\pi)_k^\top A(:,\pi)_k, \quad R_k = \begin{pmatrix} R_{k,11} & R_{k,12} \\ 0 & R_{k,22} \end{pmatrix}, \quad k = 1, 2, \cdots$$

Two Basic Approaches for Cholesky Factors

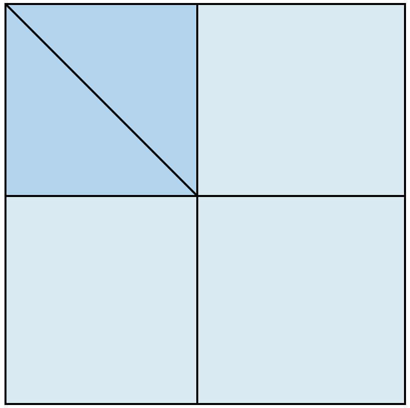
Without Orthogonal Transforms

1. Matrix-matrix multiplication

$$M_k := A_k^\top A_k$$

2. Cholesky factorization

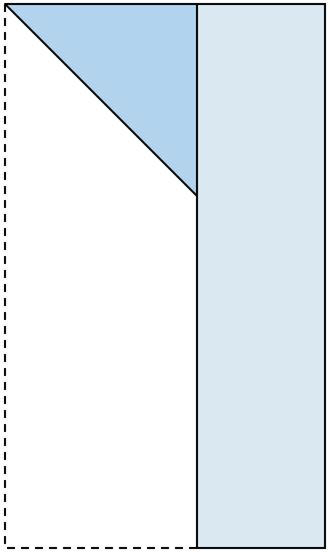
$$M_k = R_k^\top R_k$$



Via Orthogonal Transforms

1. QR factorization (Q-less)

$$A_k = Q_k \cdot R_k$$

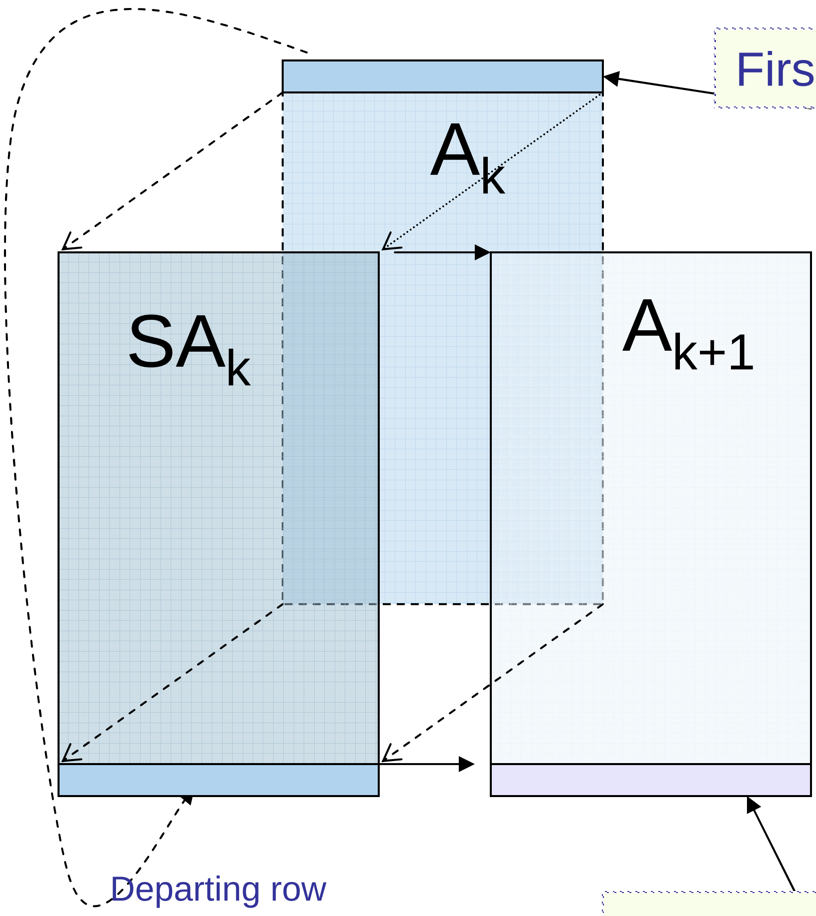


Column-wise Reduction to Upper Triangular Form

Exploiting Relationship in Consecutive Data Matrices

$$A_{k+1} = SA_k + e_m(l_{k+1}^T - f_k^T)$$

S: circulant up-shift



$$f_k^T = e_1^T A_k$$

$$l_{k+1}^T = e_m^T A_{k+1}$$

$$e_i \in R^m, \; e_i(j) = \delta(i,j)$$

- data redundancy ratio m-1 to 1
- A_{k+1} is a rank-1 update of rotated A_k
- circulant shift is an orthogonal transform

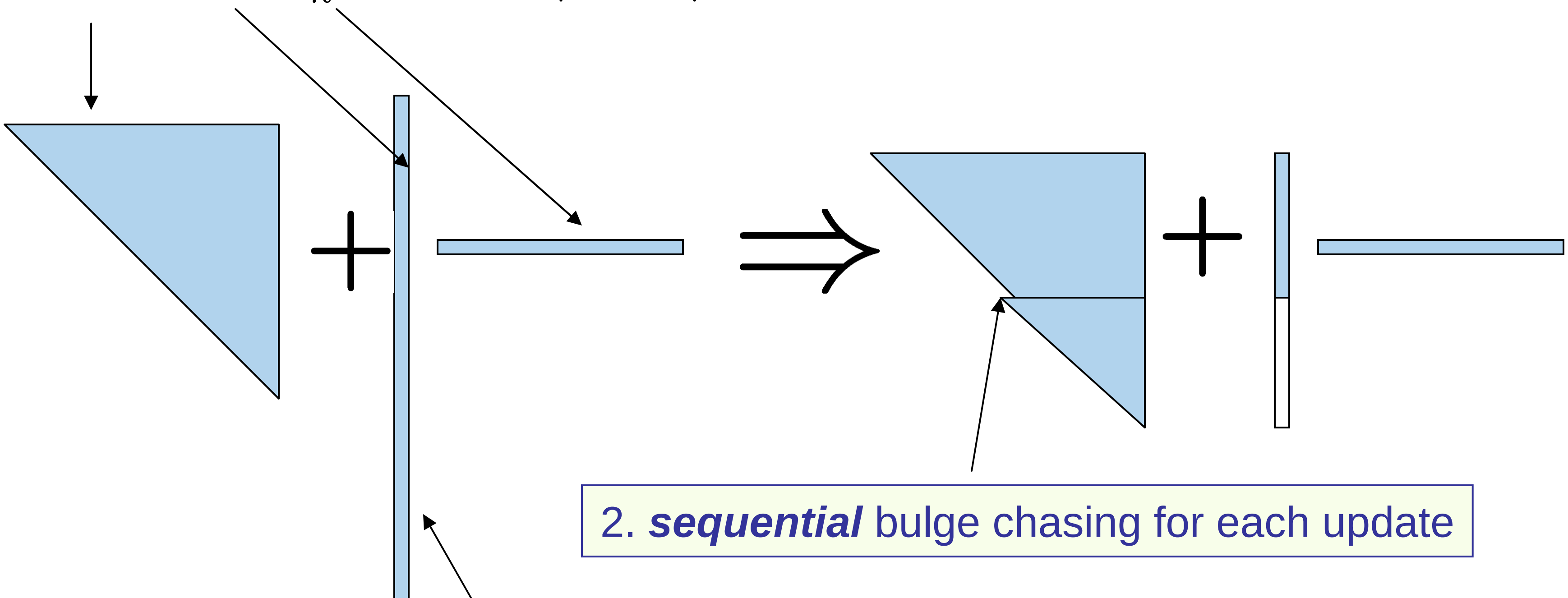
Conventional Rank-1 Update Algorithm

$$A_{k+1} = SA_k + e_m \cdot d_k^T, \qquad d_k^T = l_{k+1}^T - f_k^T,$$

$$= SQ_k(R_k + q_k \cdot d_k^T)$$

SQ$_k$ is *orthogonal*

$$R_k + q_k \cdot d_k^T = U_{k+1}R_{k+1}$$



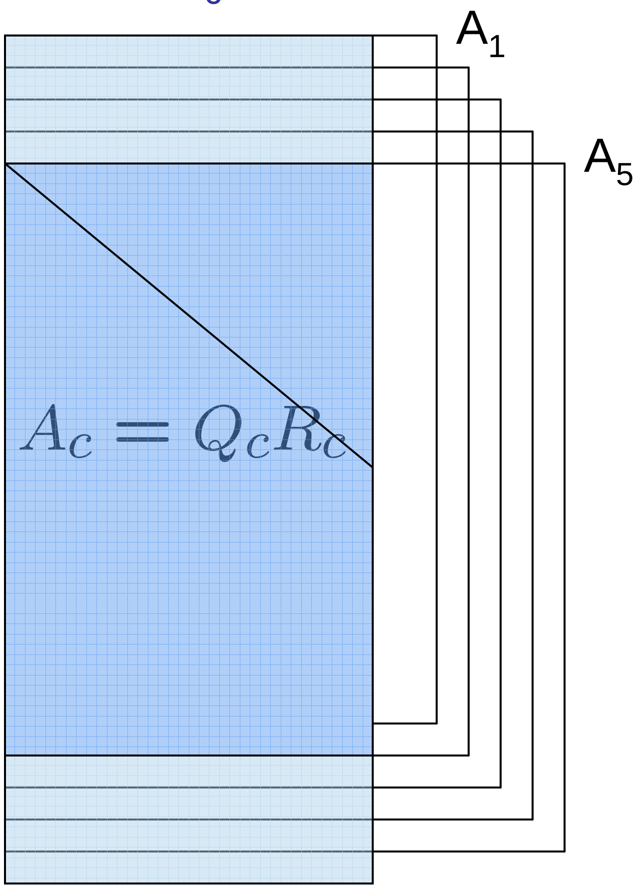
2. ***sequential*** bulge chasing for each update

1. need ***accumulation*** of **q$_k$**

3. ***sequential*** from one update to the next

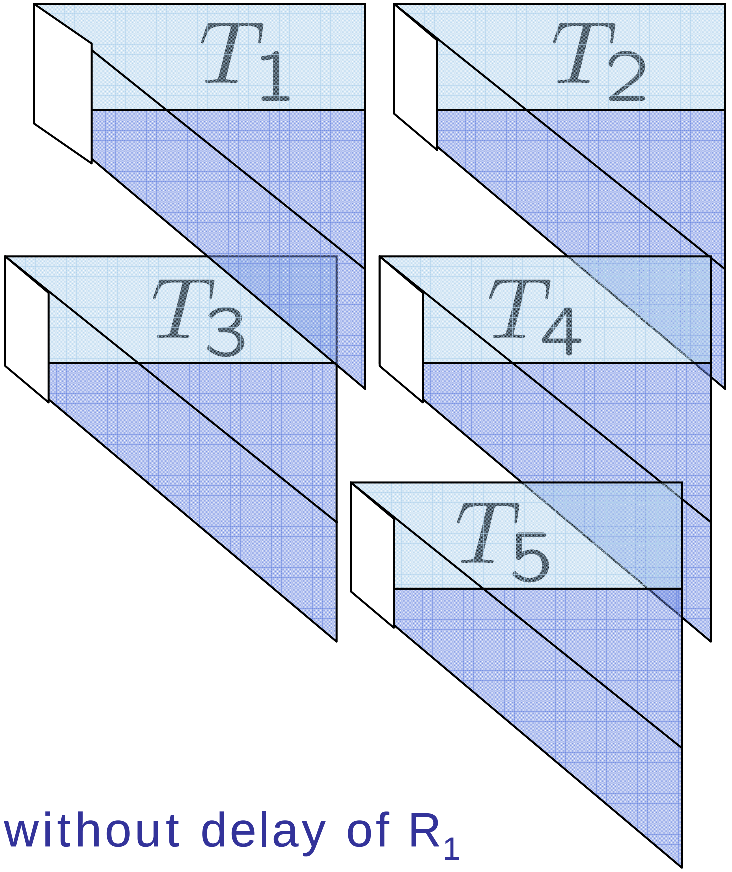
New Algorithm for Successive Factorizations

- QR factorization of the common submatrix A_c among p>1 matrices



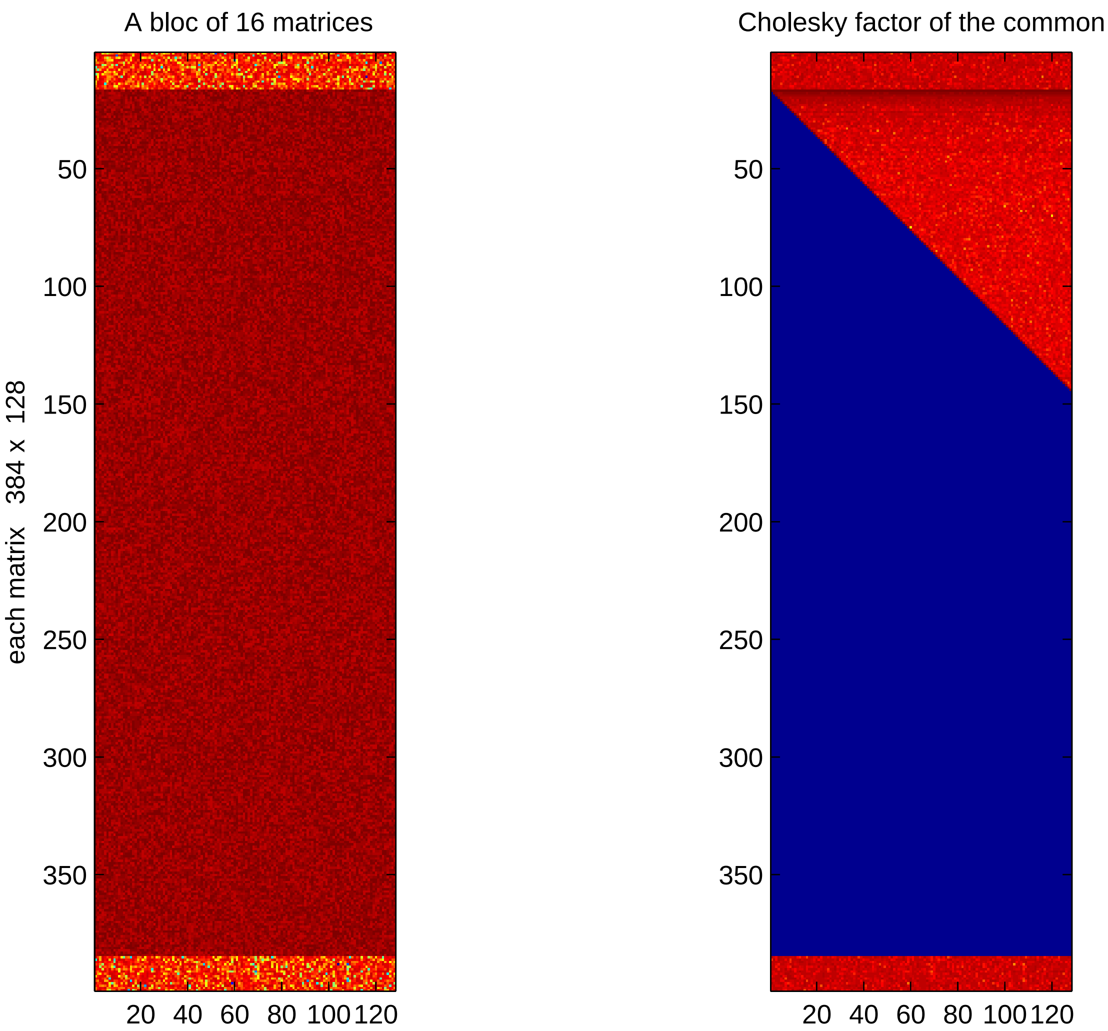
- Without complete data of A_2

- Concurrent adaptation to individual R_k, k=1:p, p>1

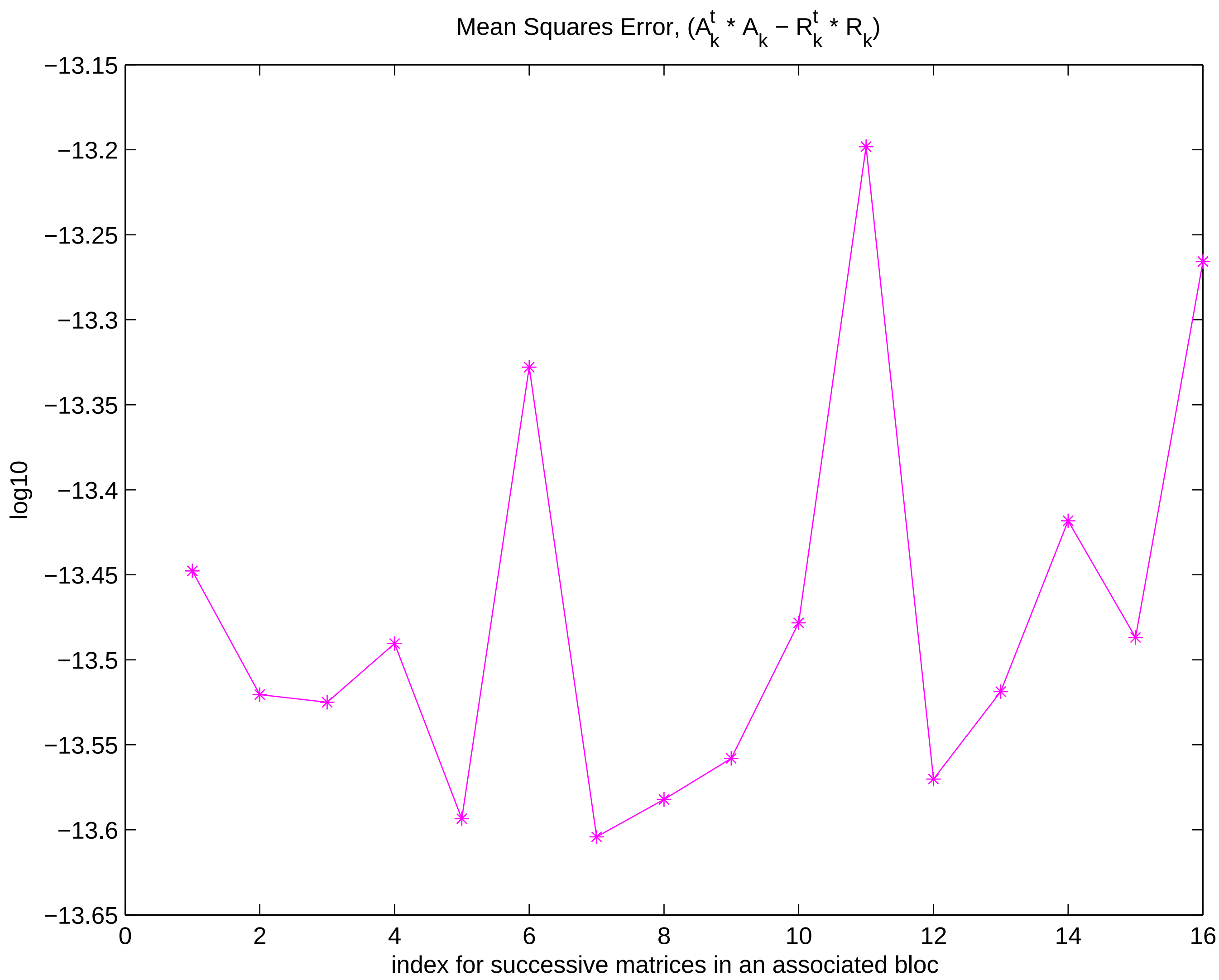


- without delay of R_1
- without bulge chasing in R_k

QR Factorization of the Central Block



Concurrent Completion of Individual Matrices



Matrix Expressions of Successive Factorizations

- Association of every p matrices

$$A_1, A_2, \cdots, A_p, \quad 2 \le p \le \sqrt{m}$$

- Common factorization

$$A_c = A_1(p/2 : m - p/2, 1 : n) = Q_c \cdot Rc$$

- Concurrent adaptation

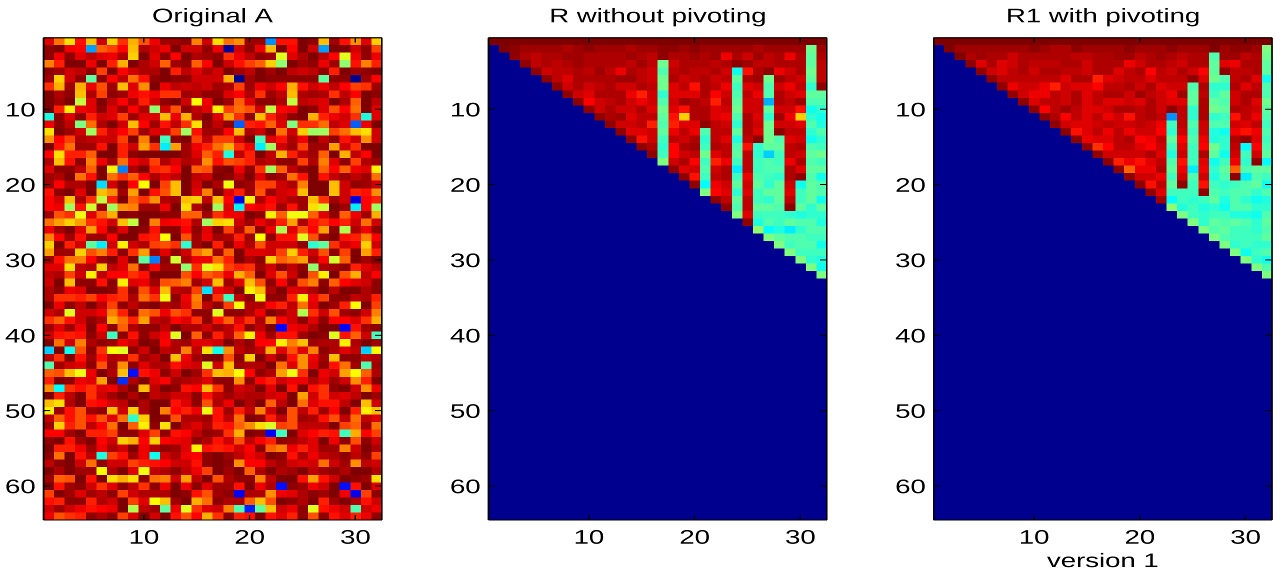
$$\begin{aligned} A_k &= S_k \begin{pmatrix} I_p & 0 \\ 0 & Q_c \end{pmatrix} \begin{pmatrix} A_k(m : -1 : m-k, 1 : n) \\ A_1(k : p/2, 1 : n) \\ R_c \end{pmatrix} \\ &= Q_k R_k \end{aligned}$$

Individual permutation

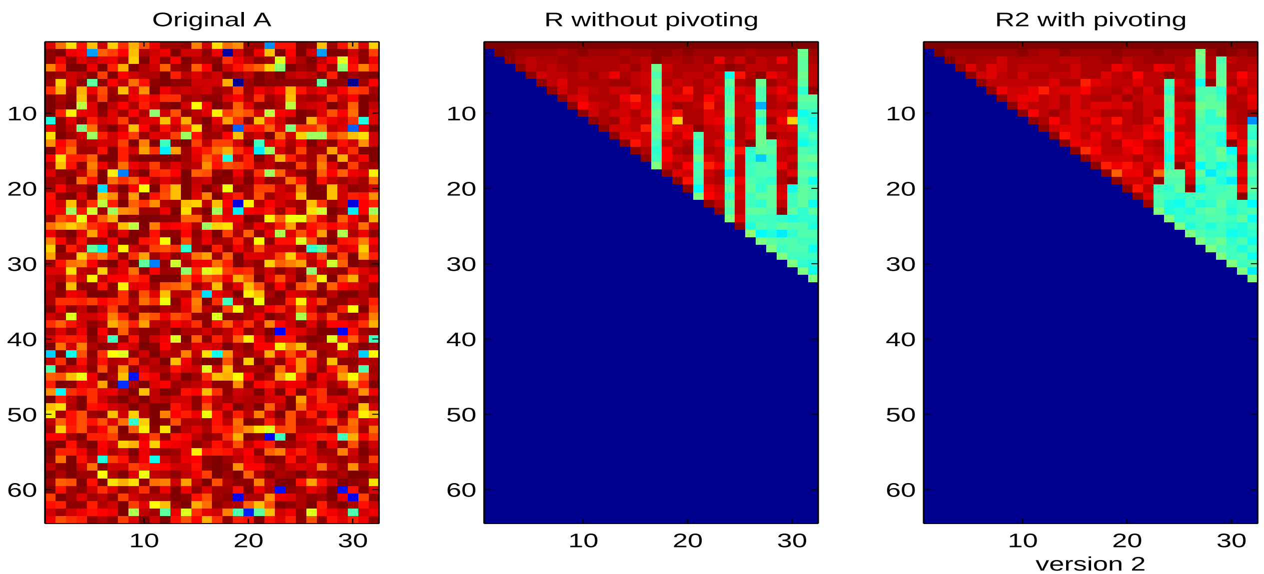
Common Orthogonal Factor

Trapezoidal form with individual top

Pivoting Strategies in the Common Block



Norm-2 calculation at every step



Initial norm-2 calculation followed by norm down-dating

Pivoting Strategies in Individual Adaptation

- Algebraically, if the r columns are linearly independent in Ac, they remain so in each and every Ak,   k = 1, 2, ..., p

- Numerically, the gap between the rank and null spaces of Rc can be carried over to Rk if

$$\left| \max_j \|T_{k,1}(:,j)\| - \min_j \|T_{k,2}(:,j)\| \right| < \text{gap}(R_c)$$

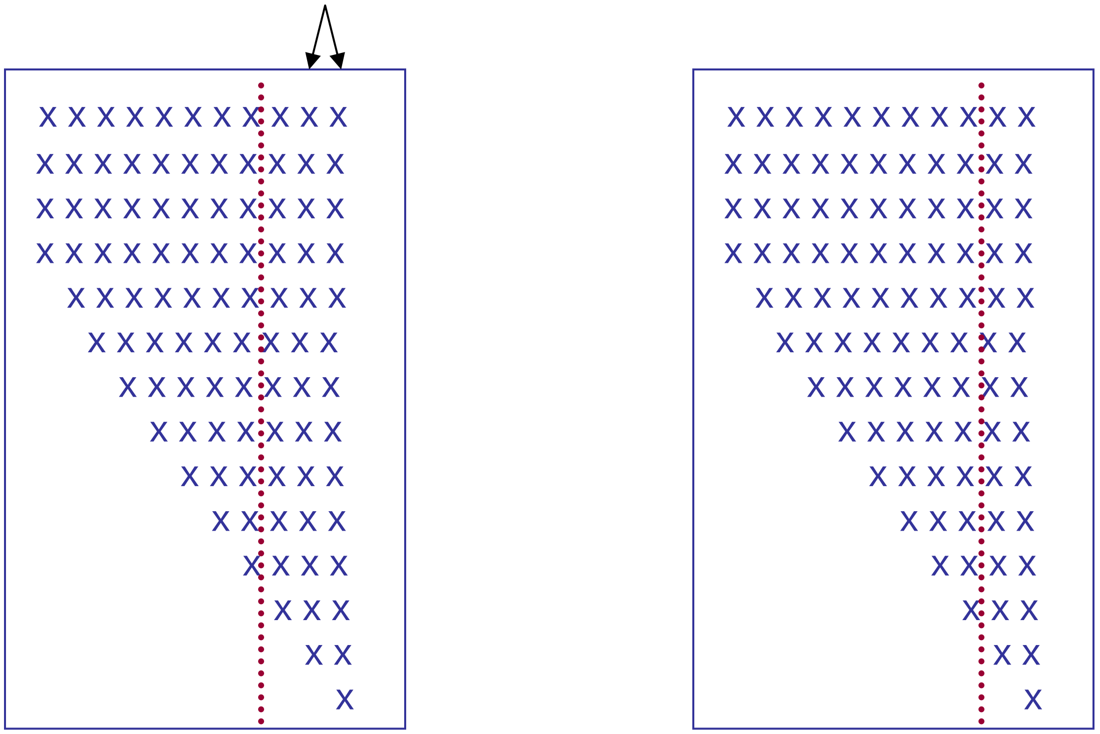
If the condition is satisfied uniformly by all Rk, then no further pivoting in the stage of individual adaptation

Pivoting Strategies in Adaptation (cont'd)

- The uniform condition is most likely met because
  - The associated matrices are highly correlated due to their temporal locality indicated by **p**
  - The individual difference is in at most **p** rows, $p \leq \sqrt{m}$
- The dynamic change beyond the temporal locality can be captured by the next association block

Efficient Pivoting Strategies (cont'd)

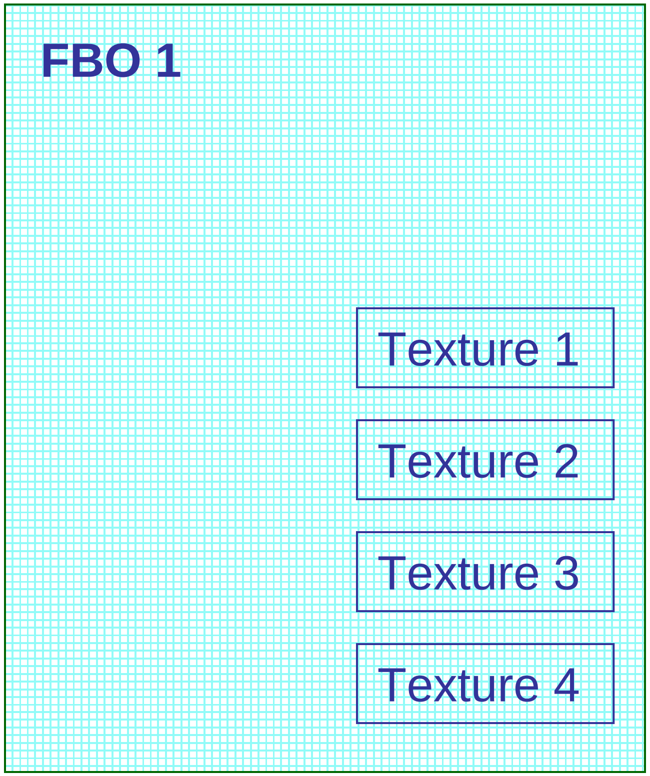
- When the dynamic change is significant within a block, use efficient backward pair-wise swapping to shift the gap individually and concurrently

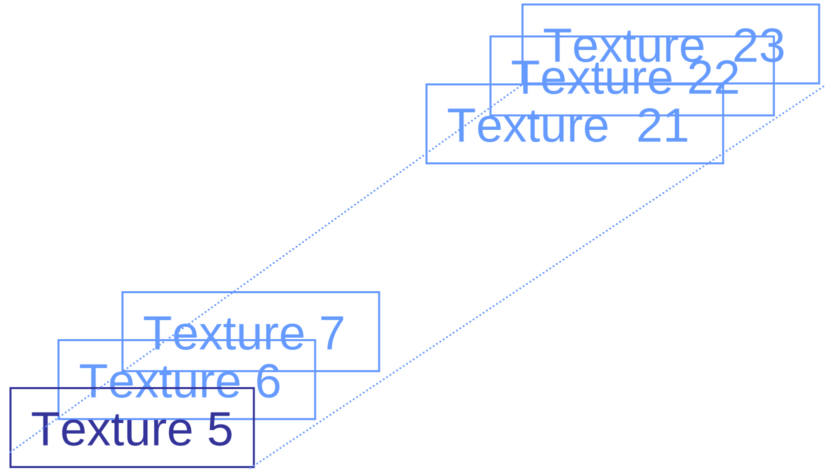


Successive QR and RR-QR on GPUs

- Matrix layout
- Reduction of redundancy in both computation and memory
- Factorization of the common block
- Concurrent completion of individual factorizations
- Performance

GPU Implementation : direct matrix-texture mapping

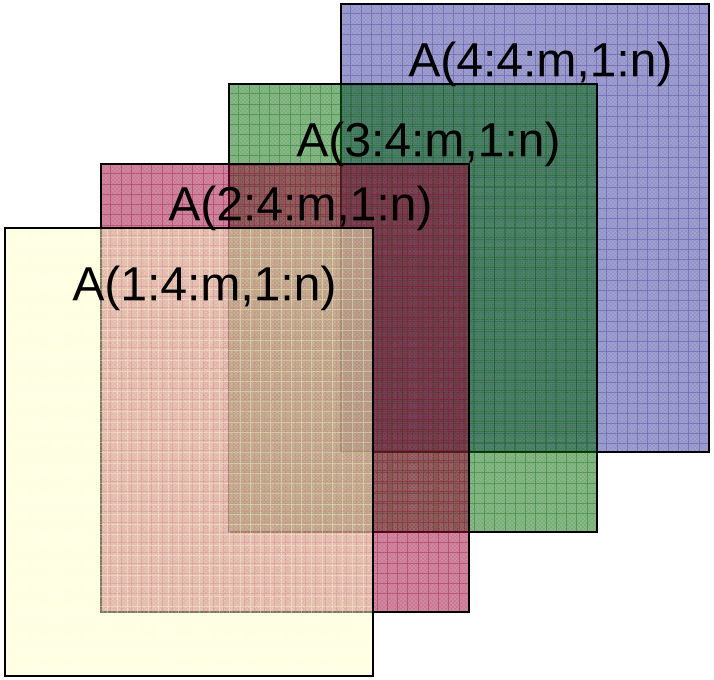




- a texture for each matrix or sub-matrix
  - the source for p successive matrices
  - intermediate matrices
  - the output : p rank-revealing Rk
- bypass the rendering to a screen
- read-only or write only operations, and not simultaneously

GPU Implementation : common block factorization

Matrix Layout for common factorization



Strategy for concurrent adaptation

1. Single copy of the shared factor R_c
   - minimize spatial redundancy
2. Subject to memory constraint :
   - q Cholesky factors at a time, $q <= p$
3. Stack un-common data blocks
   - one from each data matrix A_k, $k=1:q$
   - enhance data locality
4. Single instruction multiple data : parallelization
   - Read a single row of R_c at each step from top
   - Complete a corresponding row for every R_k, $k=1:q$

GPU Implementation : pivoting

Norm-2 Calculation

- at every reduction step
  - **$m\,n^2 - n^3 / 2$** extra flops
  - additional pipeline operations
- at the **initial step** only and followed by '**down-dating**' in subsequent steps
  - reduce extra flops to **$2\,mn + n^2$**
  - with numerical threshold for severe cancellations

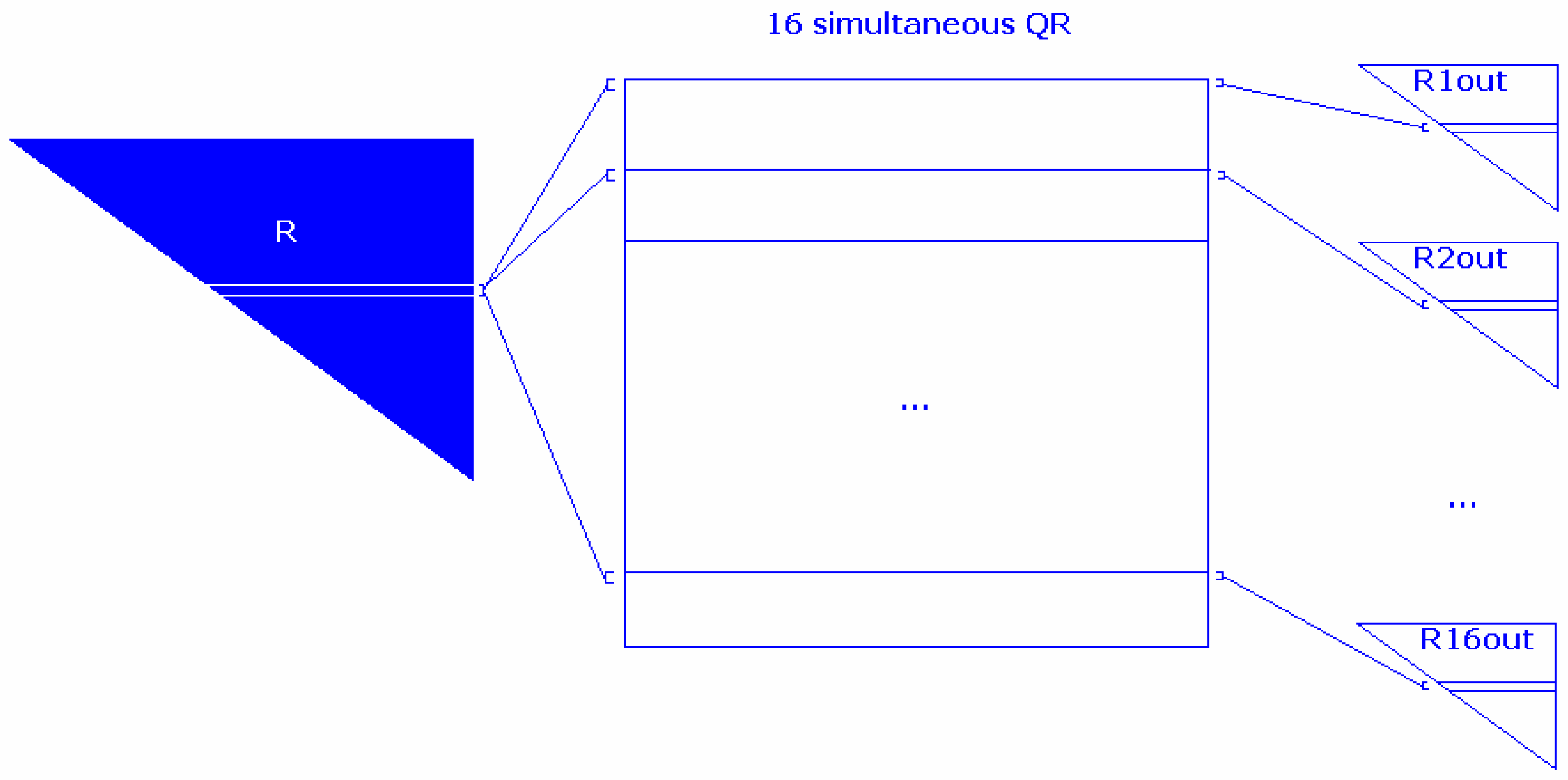
Locating Pivot Columns

- **arg max** operation
- index mapping vector

Column Permutation

- implicit (in place)
  - extra cost in indexing
- explicit
  - extra cost in data movement
- interface with triangular system solver

GPU Implementation : concurrent individual adaptation



The output Cholesky factors R_k, k=1:q, q=16, are produced simultaneously, row by row, by adapting the common R to individual un-common blocks, which are stacked together

Experiments. Development Techniques & Results

GPU architecture specifics

- Hardware
  - NVIDIA GeForce 7900 GT
- Software
  - Cg, OpenGL (GLUT, GLEW)
- Single-precision floating-point arithmetic

From MATLAB to GPUs

- Algorithm prototyping
- Simulation of GPU computing
- Debugging
- Numerical comparison

Parallel Computation

- Householder-based orthogonal transforms
- Separation of common block factorization and individual adaptation
- Rank-revealing in the common block factorization

Application specifics

- Matrix Size
  $n = 128$, $m = 384, 512, 640$
- Bloc size: $p = 24$
- Test data
  - Numerically full rank
  - Numerically 10% rank deficient

Comparison in Latency and Memory Usage

m = 512 n = 128 p = 23	Latency Absolute (ms)	Latency Relative	Memory (m*n)
Plain Q-less QR	31.2	1	p
GAXPY*	30.4	0.99	
RR-QR-V1	60.0	1.92	p
RR-QR-V2	43.2	1.38	p
Successive-RR-QR	5.53	0.18	2.0

* The GAXPY performance is based on the GPUBENCH code with unrolling parameter 6

Conclusion

- The new adaptation algorithm is
  - efficient in terms of flops via exploiting the redundancy
  - highly parallelizable via removing unnecessary dependency
- It can be used for
  - successive Cholesky factorizations
  - successive QR factorizations
  - with or without pivoting
  - with or without accumulating the Q factors
- The concurrency can be exploited in different parallel fashions
- The use of other rank-revealing schemes are under investigation

- The factorizations using Givens rotations are implemented by D. Braunreiter's group at SAIC.

- **Acknowledgements**

This work is in part supported by DARPA-MTO. We thank also Jeremy D. Furtek at SAIC for his helpful comments on performance tuning.