Performance of Random Sampling for Computing Low-rank Approximations of a Dense Matrix on GPUs

Théo Mary, Ichitaro Yamazaki, Jakub Kurzak,
Piotr Luszczek, Stanimire Tomov, Jack Dongarra
presenter

Low-Rank Approximation

- For a matrix A, find B and C such that

$$\underset{m*n}{A} \approx \underset{m*k}{B} * \underset{k*n}{C}$$

- If $||A-BC|| \leq \varepsilon$ then k=numerical rank of A
- "Low-rank" means k<<min(m,n)
- Approximation allows
  - Reduced computation
  - Reduced storage

Pivoted QR Decomposition

- Pivoted QR decomposition has a form

  $AP = [Q_1\ Q_2]\ [R_{11}\ R_{12}]$
  $\qquad\qquad\qquad\quad [\qquad R_{22}]$

  with

  - $Q = [Q_1\ Q_2]$ – m by n orthogonal matrix
  - $R = [R_{11}\ R_{12}]$ – n by n upper triangular matrix
    $\qquad [\qquad R_{22}]$
  - P – n by n column pivot matrix

- Truncated Pivoted QR decomposition

  $AP \approx Q_1\ [R_{11}\ R_{12}]$

  m*n $\qquad$ m*k $\qquad$ k*n

LAPACK's QP3

- LAPACK's QP3 computes QR factorization with column pivoting using Level 3 BLAS
- No truncated QR available
  - But only a single line change is necessary in the reference code
- Limitations:
  - Includes Level 2 BLAS (in addition to Level 3 BLAS)
  - Synchronization occurs at every step to pick a pivot
  - Limited parallelism and data locality
  - Excessive communication
  - A costly update is needed when column norms drift numerically

Randomized Algorithm: Overview

- Stage I: generate Q, an orthogonal subspace spanning the range of A:

  $A \approx AQ^TQ$

- Stage II: use Q to compute low-rank approximation of A with standard deterministic methods

Stage I: Generating Orthogonal Subspace - Intuition

- Generate random columns of B:
  for i = 1, 2, ..., k do
  w_i = random(m,1)
  $b_i = w_i A$
  end for
- $B = [b_1\ b_2\ \dots\ b_k]$
- B is **probably** linearly independent
- Orthogonalize:
  Q = orth(B)
- To improve robustness, use k+p columns
  - p is the oversampling parameter – a small constant such as 10

Stage I: Generating Orthogonal Subspace - Sampling

- $\ell = k+p$

$$\underset{\ell*n}{B} = \underset{\ell*m}{\Omega} * \underset{m*n}{A}$$

- Ω can be
  - Gaussian random matrix
    - Can use matrix-matrix multiply GEMM
  - FFT matrix
    - Can use FFT routines
    - Padding might be necessary to get power-of-two speed

Sampling with GEMM or FFT

$$\underset{\ell * n}{B} \approx \underset{\ell * m}{S} * \underset{m * m}{\Pi} * \underset{m * n}{A}$$

m=50 000
n=500
ℓ=10..250

Method	Complexity
GEMM	O(mnℓ)
FFT	O(mn log m)
Pruned FFT	O(mn log ℓ)

Stage I: Noise Reduction Through Power Method

- If the singular values of A decay slowly, the sampled matrix B may contain significant noise due to the following error bound:
  $||A-AQ^TQ|| \leq C(\Omega,p)\sigma_{k+1}$
- To reduce the noise, q iterations of power method are applied:
  $B=\Omega A (A^TA)^q$
- This yields a new bound on noise
  $||A-AQ^TQ|| \leq C(\Omega,p)^{1/(2q+1)}\sigma_{k+1}$
- Due to round-off errors we need to reorthogonalize:
  $B_0 = \Omega A$
  repeat q times:
  $Q_0 = orth(B_0)$; $B_1 = Q_0A^T$
  $Q_1 = orth(B_1)$; $B_1 = Q_1A$

Randomized Pivoted QR

- Truncated pivoted QR step:

$$BP \approx \bar{Q}\,(\bar{R}_{1:k}\ \bar{R}_{k+1:n})$$
$$= \bar{Q}_k\bar{R}_{1:k}(I_k\ \bar{R}_{1:k}^{-1}\bar{R}_{k+1:n})$$
$$= BP_{1:k}(I_k\ \bar{R}_{1:k-1}\bar{R}_{k+1:n})$$
$$\Rightarrow AP \approx AP_{1:k}(I_k\ \bar{R}_{1:k}^{-1}\bar{R}_{k+1:n})$$

- QR step:

$$AP_{1:k} = Q\tilde{R}$$

- Final approximation:

$$AP \approx Q\ \tilde{R}(I_k\ \bar{R}_{1:k}^{-1}\bar{R}_{k+1:n})$$

m*n	m*k	k*n
	Q	**R**

Pseudocode of the Implementation

1) Input: m*n matrix A
2) $B_0 = \Omega A$
3) for 1, 2, …, q do
4) $Q_o = orth(B_0)$
5) $B_1 = Q_0 A^T$
6) $Q_1 = orth(B_1)$
7) $B_1 = Q_1 A$
8) end for
9) $\bar{Q}, \bar{R}, P = TruncatedPQR(B_q)$
10) $Q, \tilde{R} = QR(AP_{1:k})$
11) $R = \tilde{R}(I_k\ \bar{R}_{1:k}^{-1}\bar{R}_{k+1:n})$
12) Output: Q, R, P such that AP≈QR

Communication Cost

- Assuming two-level memory hierarch: fast (size=M) and slow

Algorithm	#flops	#words
Sampling (Gaussian)	$O(mn\ell)$	$O(mn\ell/M^{1/2})$
Iter. (mult.)	$O(mn\ell q)$	$O(mn\ell q/M^{1/2})$
Iter. (orth.)	$O((m+n)\ell^2)$	$O((m+n)\ell^2/M^{1/2})$
QRCP	$O(n\ell^2)$	$O(n\ell^2)$
QR	$O(m\ell^2)$	$O(n\ell^2/M^{1/2})$
Total	$O(mn\ell(1+q))$	$O(mn\ell(1+q)/M^{1/2})$
QP3	$O(mnk)$	$O(mnk)$
CAQP3	$O(mn(m+n))$	$O(mn^2/M^{1/2})$

- If p and q are constant then randomized PQR converges towards communication lower bound

Orthogonalization and Numerical Stability

Algorithm	Stability	Cost
Householder QR	ε	high
Cholesky QR	$\kappa(A)^2\varepsilon$	low
CA QR	ε	low
Classical Gram-Schmidt	$\kappa(A)^2\varepsilon$	moderate
Modified Gram-Schmidt	$\kappa(A)\varepsilon$	high

We need orthogonalization for:

- Power method
- Factorization of sampled matrix: QR(B)

Cholesky QR Orthogonalization

- Cholesky QR algorithm:
  1) Form $S=X^TX$
  2) Compute Cholesky factorization R=chol(S)
  3) Solve $Q=XR^{-1}$
- Possible orthogonalization schemes
  - Repeat Cholesky QR multiple times
  - Try Cholesky and if it fails use Householder QR
  - For power method and tall-and-skinny matrices perform Cholesky on the bigger matrix and Householder on the smaller one
  - For power method orthogonalize only at some iterations
  - Use mixed-precision Cholesky QR
    - Our method of choice

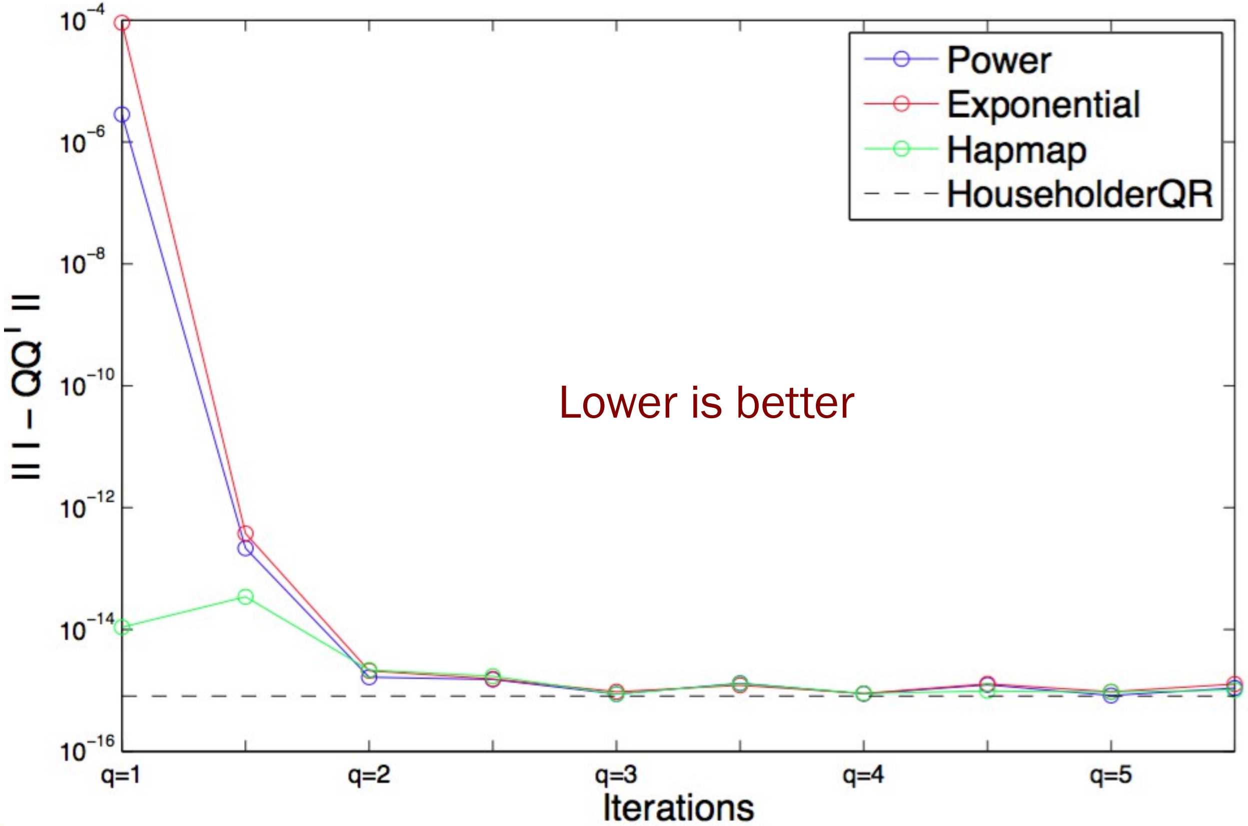
Experimental Setup

- Hardware:
  - CPU: Intel Sandy Bridge, 16 cores
  - GPU: NVIDIA Tesla K40c
- Matrices:
  - Power spectrum: $\sigma_i = -i^{\alpha}$ ($\alpha=3$)
  - Exponent spectrum: $\sigma_i = 10^{-i\gamma}$ ($\gamma=0.1$)
  - HapMap

	Power	Exponent	HapMap
m	500 000	500 000	503 783
n	500	500	506
k	50	50	50
p	10	10	10
ℓ	60	60	60
σ_1	1	1	9900
σ_{k+1}	$8*10^{-6}$	$1.3*10^{-5}$	500
$\kappa(A)$	$1.3*10^{5}$	$7.9*10^{4}$	20

Orthogonalization: Numerical Results

- Test orthogonality at each iteration: $||I_\ell - Q_0 Q_0^T||$ and $||I_\ell - Q_1 Q_1^T||$
- κ(B)≈κ(A)



Convergence

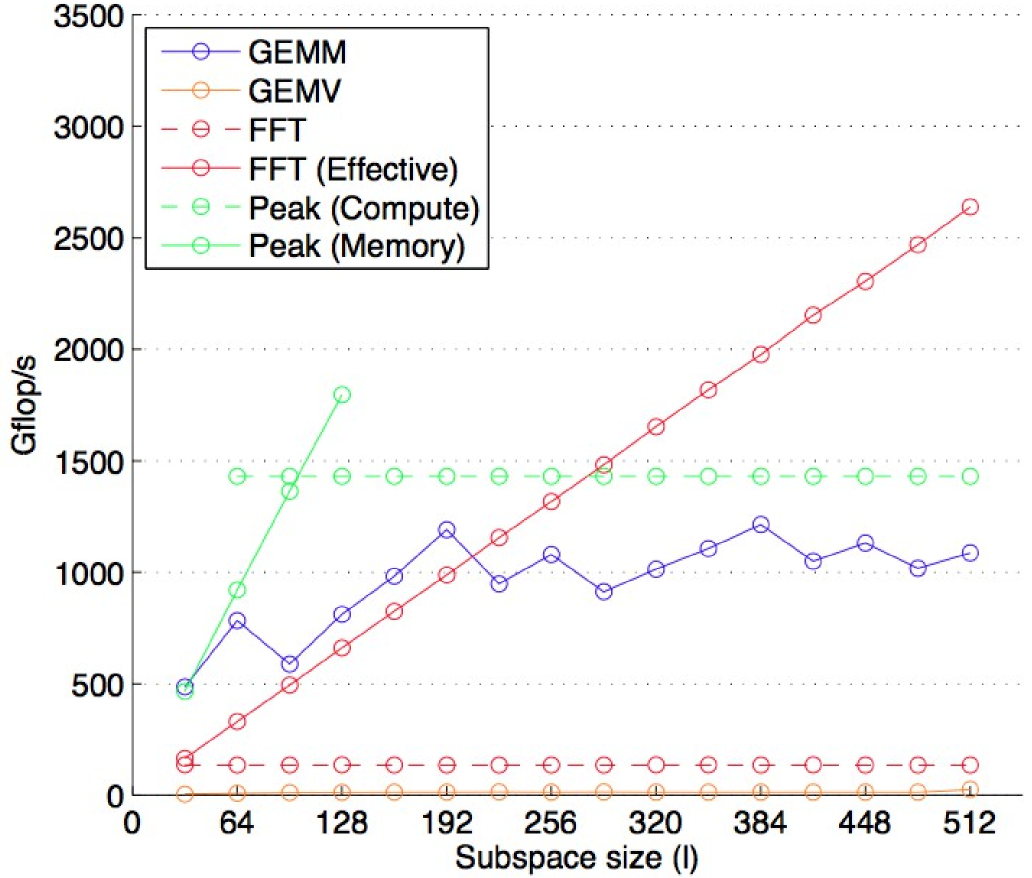
- Approximation error: ||AP – QR|| / ||A||

	QP3	Rand q=0	Rand q=1	Rand q=2
Power	$4.47*10^{-5}$	$9.08*10^{-5}$	$4.59*10^{-5}$	$4.45*10^{-5}$
Exponent	$2.69*10^{-5}$	$5.15*10^{-5}$	$2.69*10^{-5}$	$2.69*10^{-5}$
HapMap	$5.99*10^{-5}$	$9.86*10^{-1}$	$8.74*10^{-1}$	$8.18*10^{-1}$

- Oversampling helps a lot
  - No oversampling (p=0) gives an order of magnitude larger error than with oversampling (p=10)

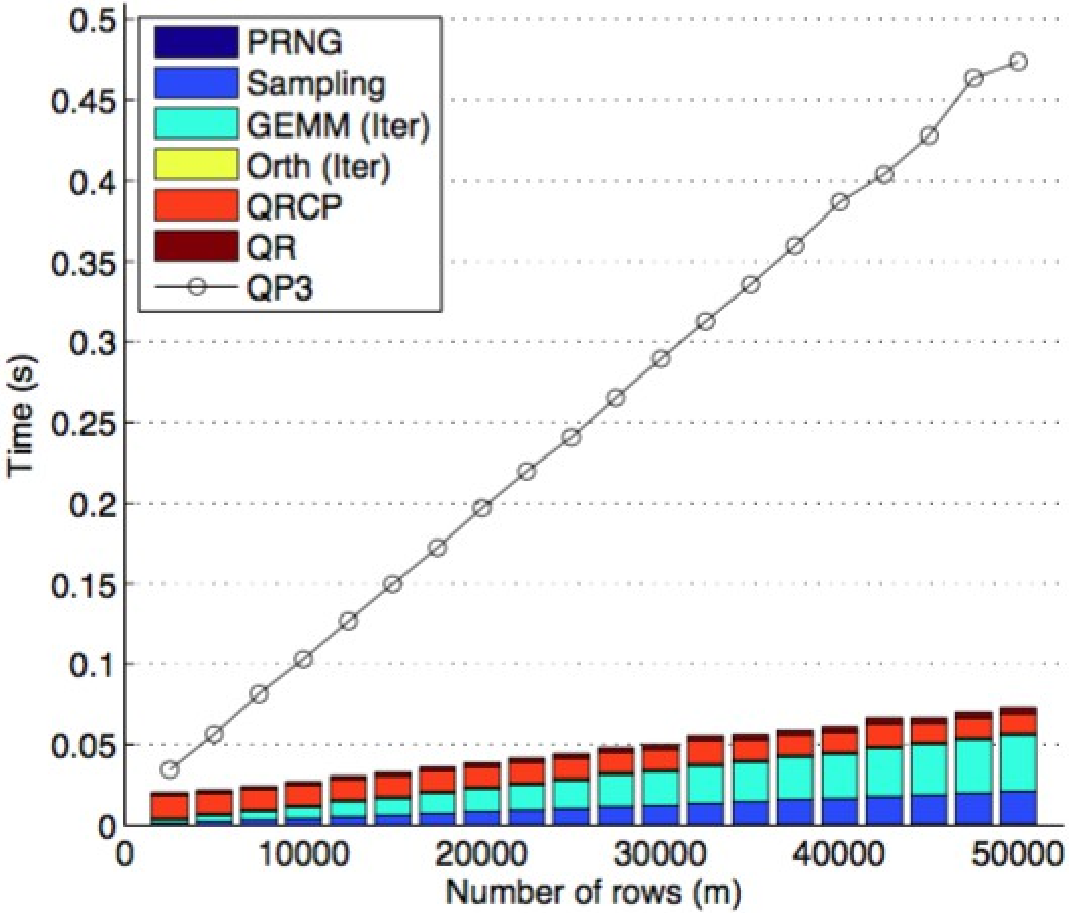
Sampling Performance

Higher is better



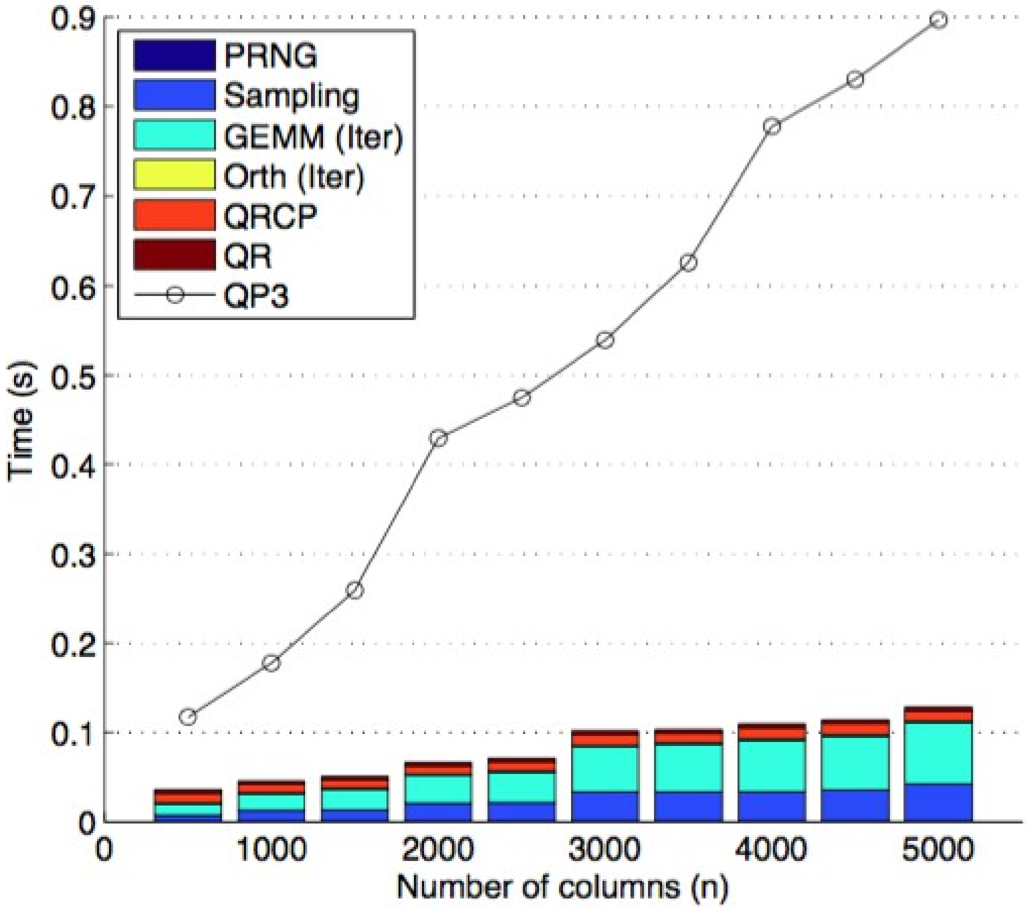
Random QR Approx. vs QP3 – Rows Variable

Lower is better



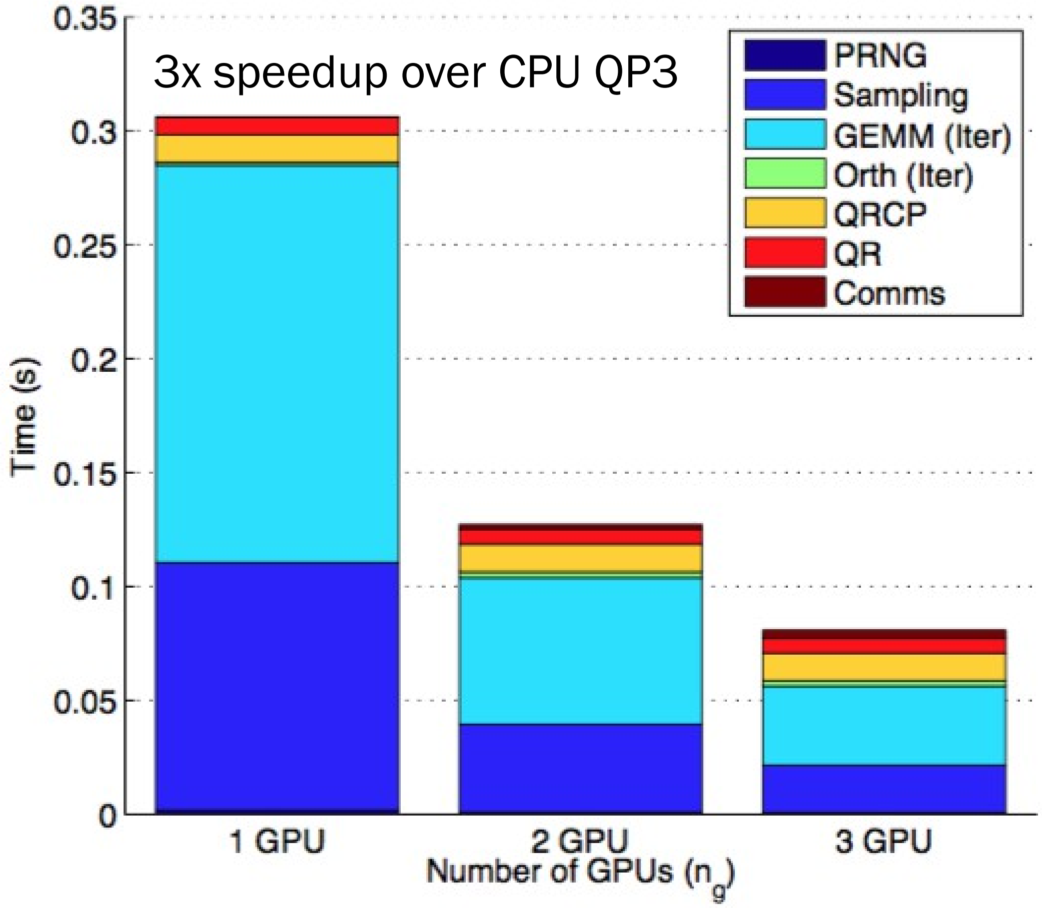
Random QR Approx. vs QP3 – Columns Variable

Lower is better



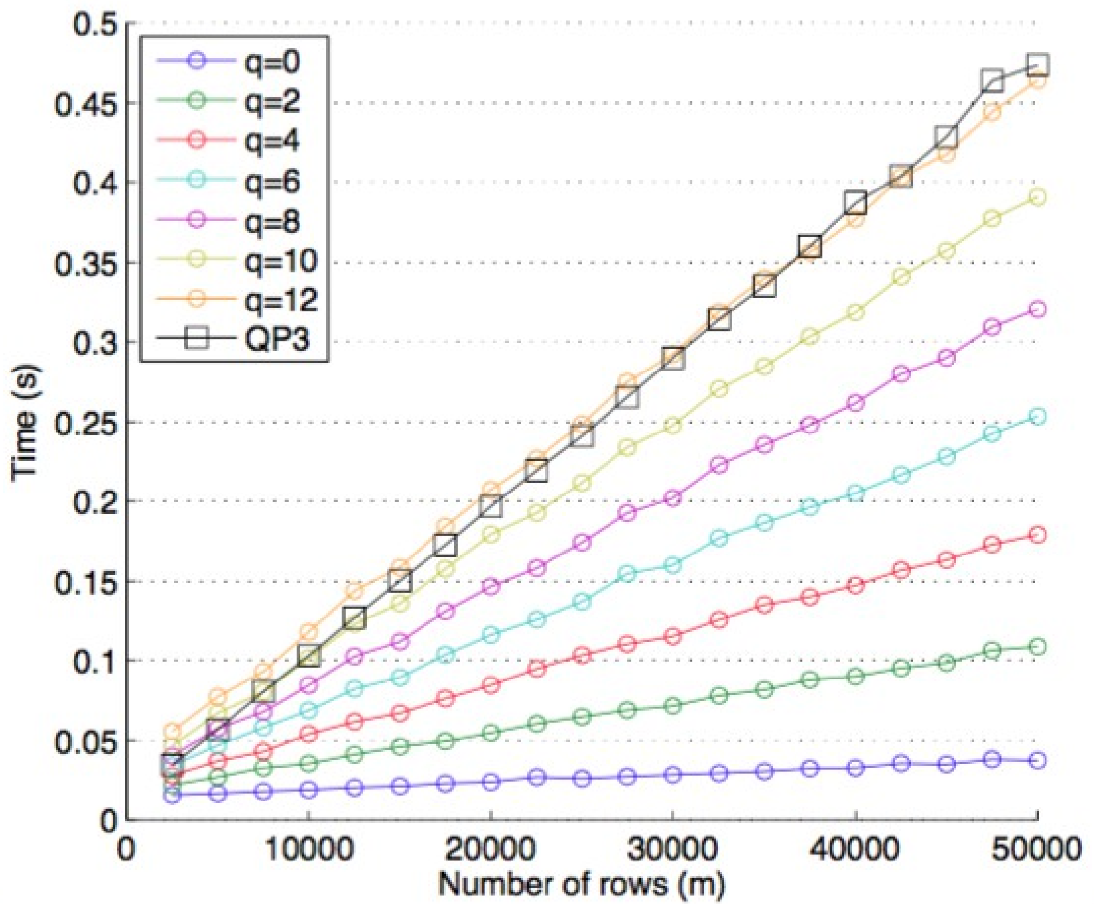
Random QR Approx. across GPUs

Lower is better



Random QR Approx. vs QP3 – Power Iterations Variable

Lower is better



Summary and Conclusions

- Randomization works effectively for pivoted QR and may be considered a replacement for QP3
  - Accuracy on test matrices is indistinguishable
  - Further testing needed
- Randomized algorithms has attractive properties (Exascale-compliant)
  - Data locality
  - Higher parallelism levels
  - Lack of synchronization
  - Minimized communication
- New tests of usefulness needed from applications
  - Clustering, ...
- Possible extension: more comprehensive survey of QR implementations for low-rank approximation