Performance of Random Sampling for Low-rank Approximation on GPUs

Théo Mary,

October 1, 2014

Context

- Application with high computational and storage costs.
  - e.g., solve $Ax = b$
- We can use matrix compression techniques to save computations and storage.
  - Because of redundant information;
  - Because we don't need ε_{mach} accuracy;

Context

- Application with high computational and storage costs.
  - e.g., solve $Ax = b$
- We can use matrix compression techniques to save computations and storage.
  - Because of redundant information;
  - Because we don't need ε_{mach} accuracy;

Mathematical Formulation

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

- $\|E\| = \|A - BC\| \leq \sigma_{k+1} \leq \varepsilon$
- The ε threshold is to be set by the user, according to their needs.

Pivoted QR Decomposition

- Pivoted QR decomposition

$$AP = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$$

  with
  - $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$ an $m \times n$ orthogonal matrix;
  - $R = \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$ an $n \times n$ upper triangular matrix;
  - P a $n \times n$ pivot matrix.
- Truncated Pivoted QR Decomposition

$$\underset{m \times n}{AP} \approx \underset{m \times k}{Q_1} \underset{k \times n}{\begin{pmatrix} R_{11} & R_{12} \end{pmatrix}}$$

LAPACK's QP3

- Computes QR with column pivoting using BLAS-3.
- To compute truncated version $\rightarrow$ one line to change in the code.
- Limitations:
  - Not only BLAS-3: also BLAS-2;
  - Synchronization at every step to pick pivot;
  - Limited parallelism and data locality;
  - Communications.

Randomized Algorithm: Overview

- **Stage A**: generate Q, orthogonal subspace spanning the range of A, i.e.:

$$A \approx AQ^T Q$$

- **Stage B**: use Q to compute low-rank approximations of A (QR, SVD, ...) with standard deterministic methods.

for $i = 1, 2, \ldots, k$ **do**
    Draw random vector ω_i.
    Form the product $b_i = \omega_i A$.
end for

- $B = \{b_1, \ldots, b_k\}$ is **probably** linearly independent
  $\Rightarrow$ $Q = \text{orth}(B)$.

Stage A: Intuition

for $i = 1, 2, \ldots, k + p$ **do**
    Draw random vector ω_i.
    Form the product $b_i = \omega_i A$.
end for

- $B = \{b_1, \ldots, b_{k+p}\}$ is **probably** linearly independent
  $\Rightarrow$ $Q = \text{orth}(B)$.
- p is the oversampling.

Stage A: Sampling

- $\ell = k + p$

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

Stage A: Sampling

- $\ell = k + p$

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

- How do we generate Ω?
  - Gaussian
  - FFT

Stage A: Power Method

- $\|A - AQ^T Q\| \leq C(\Omega, p)\sigma_{k+1}$
  - If $\{\sigma_i\}_{i=1,n}$ decay slowly, $\|A - AQ^T Q\|$ can be big.
- $B = \Omega\, A\, (A^T A)^q$
- $\|A - AQ^T Q\| \leq C(\Omega, p)^{1/(2q+1)}\sigma_{k+1}$

Stage A: Power Method

- $\|A - AQ^T Q\| \leq C(\Omega, p)\sigma_{k+1}$
  - If $\{\sigma_i\}_{i=1,n}$ decay slowly, $\|A - AQ^T Q\|$ can be big.
- $B = \Omega\, A\, (A^T A)^q$
- $\|A - AQ^T Q\| \leq C(\Omega, p)^{1/(2q+1)}\sigma_{k+1}$
- Round-off errors $\Rightarrow$ need to reorthogonalize.

$$
\begin{array}{lcl}
B_0 & = & \Omega\, A \\
\text{repeat } q \text{ times:} & & \\
\acute{Q}_0 & = & \text{orth}(B_0) \quad ; \quad \acute{B}_1 = \acute{Q}_0\, A^T \\
\acute{Q}_1 & = & \text{orth}(\acute{B}_1) \quad ; \quad B_1 = \acute{Q}_1\, A
\end{array}
$$

Randomized PQR

- Truncated PQR step:
  - Compute pivot P using sampled matrix B:

$$BP \approx \widehat{Q}_k \begin{pmatrix} \widehat{R}_{1:k} & \widehat{R}_{k+1:n} \end{pmatrix} = BP_{1:k} \begin{pmatrix} I_k & \widehat{R}_{1:k}^{-1} \widehat{R}_{k+1:n} \end{pmatrix}$$

  - Use pivot P on full matrix A:

$$\Rightarrow AP \approx AP_{1:k} \begin{pmatrix} I_k & \widehat{R}_{1:k}^{-1} \widehat{R}_{k+1:n} \end{pmatrix}$$

- QR step:

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

- Final approximation:

$$\underset{m \times n}{AP} \approx \underset{m \times k}{\underset{\mathbf{Q}}{Q}} \quad \underset{k \times n}{\underset{\mathbf{R}}{\overline{R} \begin{pmatrix} I_k & \widehat{R}_{1:k}^{-1} \widehat{R}_{k+1:n} \end{pmatrix}}}$$

Communications

- Two memory levels hierarchy: fast/slow (M = size of fast memory).

	#flops	#words
Sampling	$\mathcal{O}(mn\ell)$	$\mathcal{O}(mn\ell/M^{1/2})$
Iter. (mult.)	$\mathcal{O}(mn\ell q)$	$\mathcal{O}(mn\ell q/M^{1/2})$
Iter. (orth.)	$\mathcal{O}((m+n)\ell^2)$	$\mathcal{O}((m+n)\ell^2/M^{1/2})$
TruncPQR	$\mathcal{O}(n\ell^2)$	$\mathcal{O}(n\ell^2)$
QR	$\mathcal{O}(m\ell^2)$	$\mathcal{O}(m\ell^2/M^{1/2})$
Total	$\mathcal{O}(mn\ell(1+q))$	$\mathcal{O}(mn\ell(1+q)/M^{1/2})$
TruncQP3	$\mathcal{O}(mnk)$	$\mathcal{O}(mnk)$

- Under assumption that p, q are constant, Randomized PQR converges towards communications lower bound.

Orthogonalization

- Needed for power method and for QR($AP_{1:k}$) step.

	Stability		Performance
Householder QR	☺☺	ε	☹
Cholesky QR	☺	$\kappa(A)^2\varepsilon$	☺
CA QR	☺☺	ε	☺?
CGS	☺	$\kappa(A)^2\varepsilon$	☺
MGS	☺	$\kappa(A)\varepsilon$	☹

- Cholesky QR Algorithm:
  1. Form $S = X^T X$.
  2. Compute Cholesky factorization $R = \text{chol}(S)$.
  3. Solve $Q = XR^{-1}$.

Experimental Setups

- Hardware: Bunsen (16 threads running on 16 cores Genuine Intel(R) CPU @ 2.60GHz + 3 GPUs Tesla K40c)
- Software: MAGMA.
- Test matrices:
  - power: $A = X\Sigma Y$, with $\sigma_i = -i^{\alpha}$, ($\alpha = 3$).
  - exponent: $A = X\Sigma Y$, with $\sigma_i = 10^{-i\gamma}$ ($\gamma = 0.1$).
  - 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	9.9e+03
σ_{k+1}	8e-06	1.3e-05	5e+02
$\kappa(A)$	1.3e+05	7.9e+04	2e+01

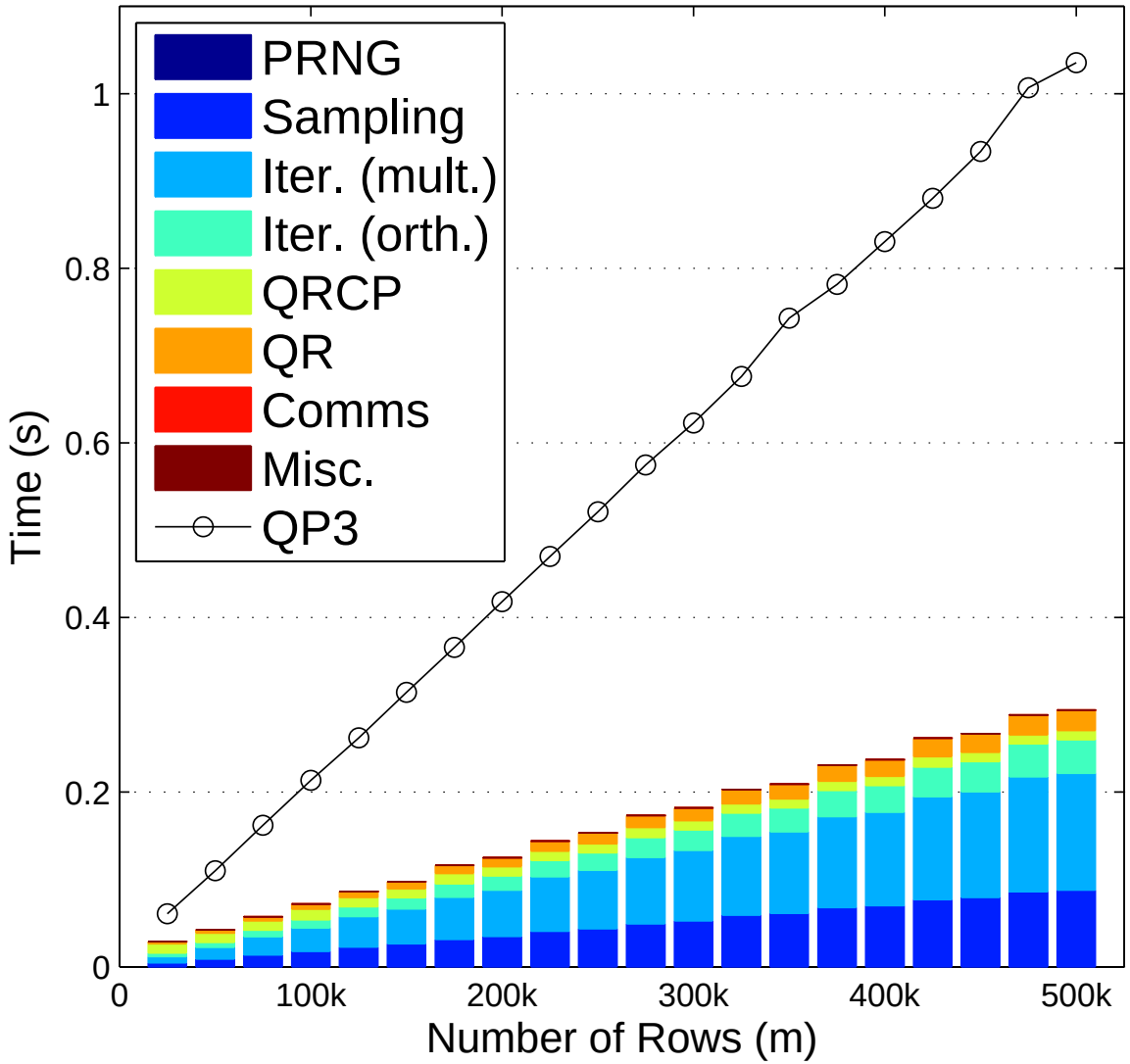
Convergence

- Approximation error $\|AP - QR\|/\|A\|$

	QP3	Rand $q = 0$	Rand $q = 1$	Rand $q = 2$
Power	4.4679e-05	9.0784e-05	4.5948e-05	4.4489e-05
Exponent	2.6892e-05	5.1795e-05	2.6899e-05	2.6898e-05
Hapmap	5.9887e-01	9.8563e-01	8.7363e-01	8.1816e-01

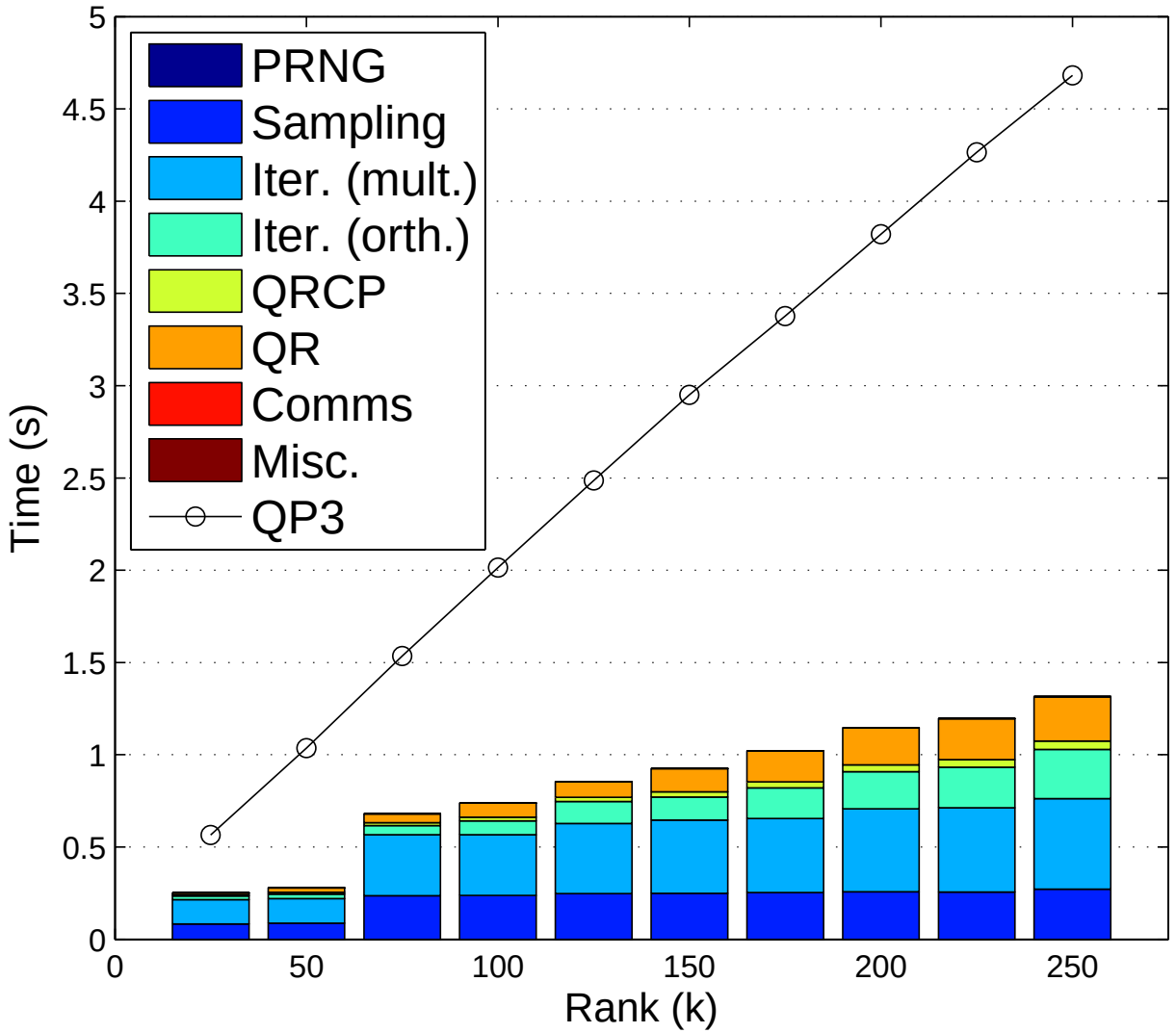
- Usefulness of oversampling: roughly an order of magnitude.

Time with increasing number of rows



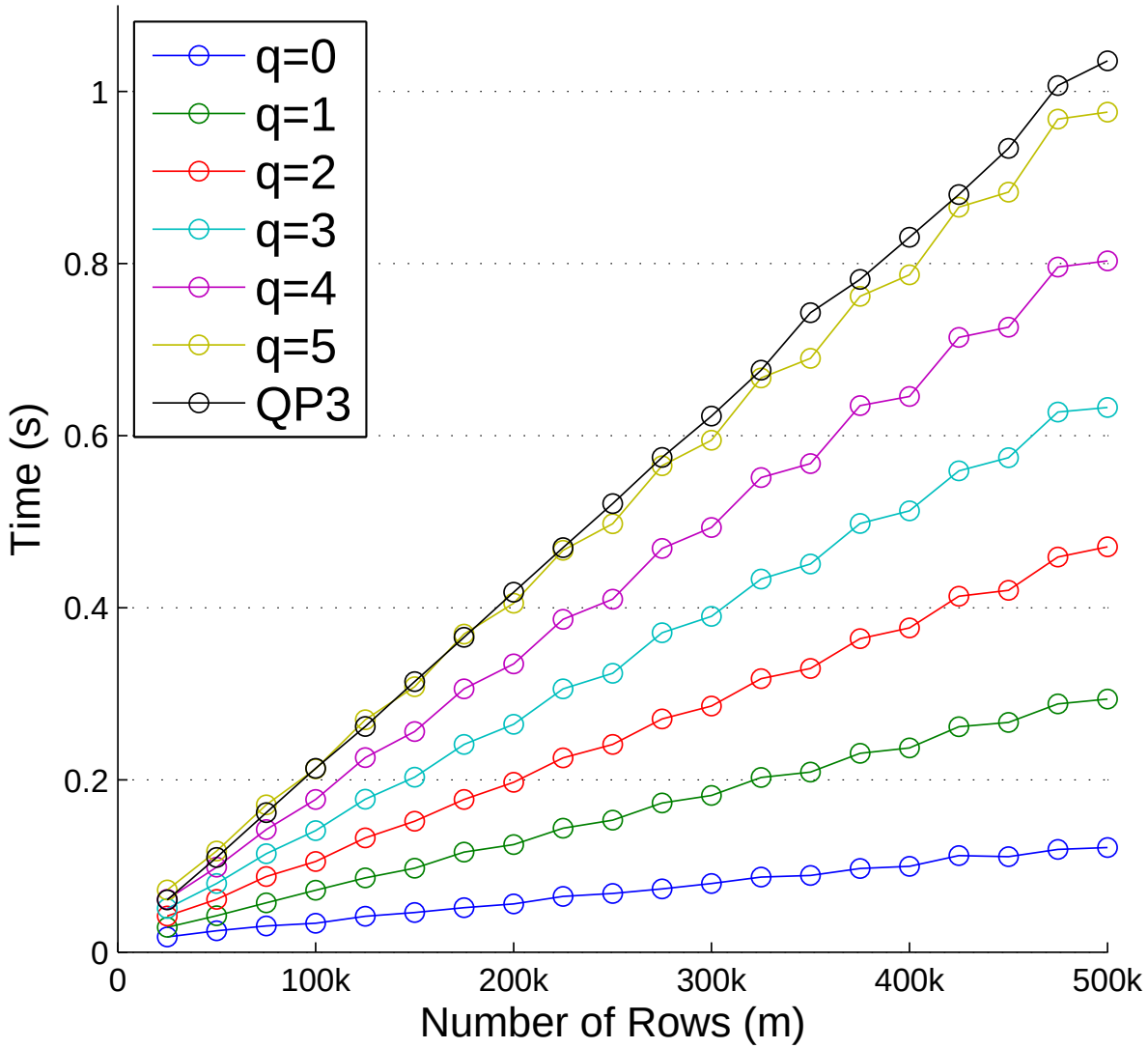
$(m = 25,000 \rightarrow 500,000;\ n = 500;\ \ell = k + p = 50 + 10;\ q = 1)$

Time with increasing rank



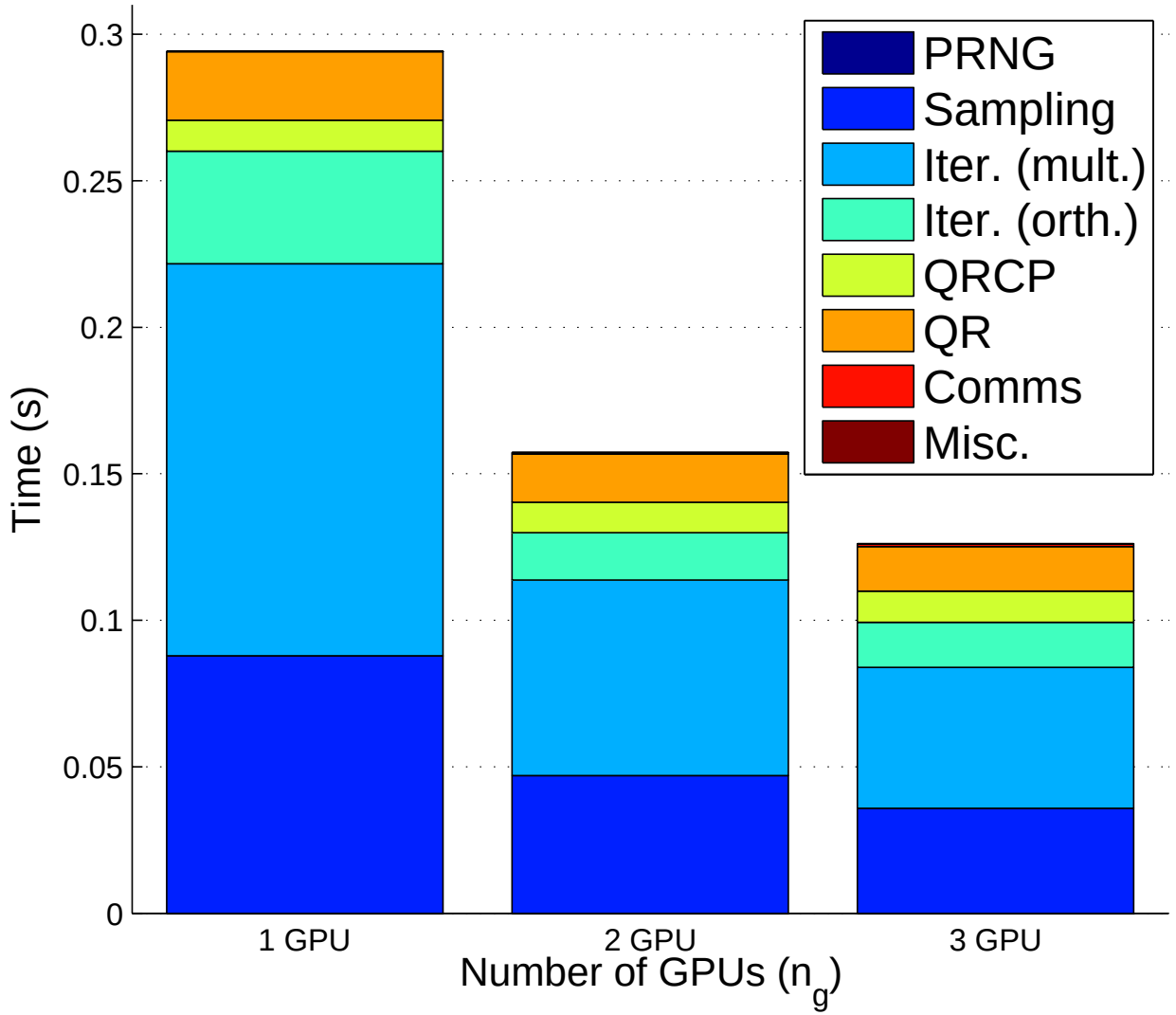
$(m = 500,000;\ n = 500;\ \ell = k + p;\ k = 25 \to 250;\ p = 10;\ q = 1)$

Time with increasing number of iterations



$(m = 25,000 \rightarrow 500,000;\ n = 500;\ \ell = k + p = 50 + 10;\ q = 0 \rightarrow 5)$

Time on 1,2,3 GPUs



$(m = 500,000;\ n = 500;\ \ell = k + p = 50 + 10;\ q = 1)$

Conclusion

Summary

- Substitute QP3 with Randomized, which is dominated by matrix-matrix multiplications $\Rightarrow$ very nice properties: data locality, higher parallelism, no synchronizations, minimized communications.
- Comparable accuracy on the test matrices used.
- Performance speedups above 8 and scaling on multiple GPUs.
- For the future:
  - Use other error measurements (applications e.g. clustering).
  - Orthogonalization studies.

PhD

- Using low-rank approximations to improve multifrontal solvers.

Thank you!
Questions?