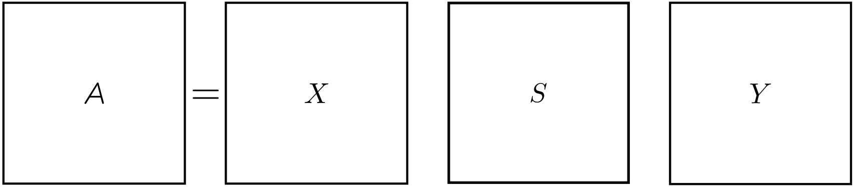
Performance of Random Sampling for Computing Low-rank Approximations of a Dense matrix on GPUs

Theo Mary [1] Ichitaro Yamazaki [2] Jakub Kurzak [2]
Piotr Luszczek [2] Stanimire Tomov [2] Jack Dongarra [2,3,4]

[1]Université de Toulouse, UPS-IRIT, France.
[2]University of Tennessee, Knoxville, Tennessee, U.S.A. Department of Computer Science
[3]Oak Ridge National Laboratory [4]University of Manchester

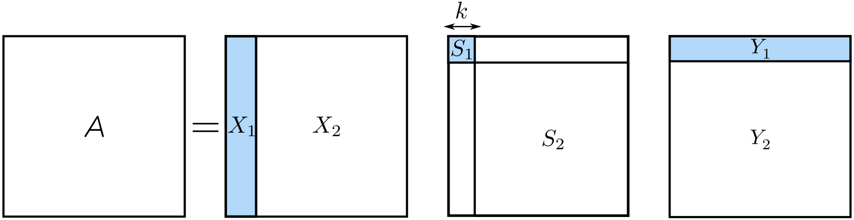
SIAM CSE 15, March 14-18

Take a dense matrix A of size $m \times n$ and compute its SVD $A = XSY$:



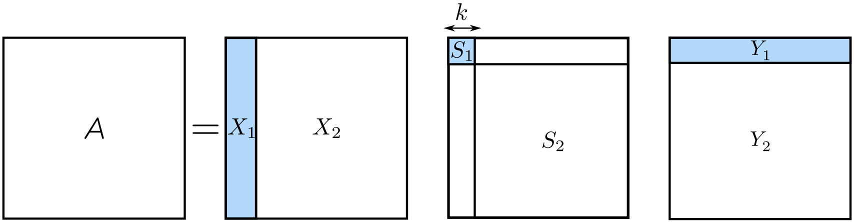
Low-rank matrices

Take a dense matrix A of size $m \times n$ and compute its SVD $A = XSY$:



$$A = X_1 S_1 Y_1 + X_2 S_2 Y_2 \quad \text{with} \quad S_1(k,k) = \sigma_k > \varepsilon,\ S_2(1,1) = \sigma_{k+1} \leq \varepsilon$$

Take a dense matrix A of size $m \times n$ and compute its SVD $A = XSY$:

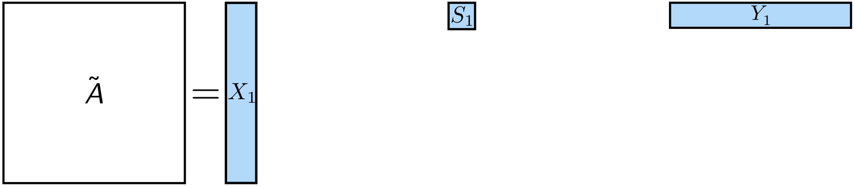


$A = X_1 S_1 Y_1 + X_2 S_2 Y_2 \quad \text{with} \quad S_1(k,k) = \sigma_k > \varepsilon,\ S_2(1,1) = \sigma_{k+1} \leq \varepsilon$

If $\tilde{A} = X_1 S_1 Y_1 \quad \text{then} \quad \|A - \tilde{A}\|_2 = \|X_2 S_2 Y_2\|_2 = \sigma_{k+1} \leq \varepsilon$

Low-rank matrices

Take a dense matrix A of size $m \times n$ and compute its SVD $A = XSY$:



$A = X_1 S_1 Y_1 + X_2 S_2 Y_2$ with $S_1(k,k) = \sigma_k > \varepsilon,\ S_2(1,1) = \sigma_{k+1} \leq \varepsilon$

If $\tilde{A} = X_1 S_1 Y_1$ then $\|A - \tilde{A}\|_2 = \|X_2 S_2 Y_2\|_2 = \sigma_{k+1} \leq \varepsilon$

If the singular values of A decay very fast (e.g. exponentially) then $k \ll \min(m,n)$ even for very small ε (e.g. 10^{-14})
$\Rightarrow$ memory and CPU consumption can be reduced considerably with a controlled loss of accuracy ($\leq \varepsilon$) if $\tilde{A}$ is used instead of A

QRCP Decomposition

- QR decomposition with Column Pivoting

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

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

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

LAPACK's QP3

- QP3 computes a QR with column pivoting factorization using BLAS-3 kernels.
- We modified the code to get the truncated version.
- Limitations:
  - Not only BLAS-3: also BLAS-2;
  - Synchronization at every step to pick pivot;
  - Limited parallelism and data locality;
  - Communications.
  - Column norms may diverge $\rightarrow$ need to recompute and update.

Random Sampling: 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.

Stage A: Sampling

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

- $\ell = k + p$, where p is a small parameter called oversampling.

Stage A: Sampling

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

- $\ell = k + p$, where p is a small parameter called oversampling.
- How do we generate Ω?
  - Gaussian
  - FFT

Stage A: Sampling

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

- $\ell = k + p$, where p is a small parameter called oversampling.
- How do we generate Ω?
  - Gaussian
  - FFT
- We get Q by orthogonalizing B. However, if $\{\sigma_i\}_{i=1,n}$ decay slowly, $\|A - AQ^T Q\|$ can be big.

Stage A: Sampling

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

- $\ell = k + p$, where p is a small parameter called oversampling.
- How do we generate Ω?
  - Gaussian
  - FFT
- We get Q by orthogonalizing B. However, if $\{\sigma_i\}_{i=1,n}$ decay slowly, $\|A - AQ^TQ\|$ can be big.
- We can overcome this issue with a few power iterations:

$$B = \Omega\, A\, (A^T A)^q$$

Stage A: Sampling

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

- $\ell = k + p$, where p is a small parameter called oversampling.
- How do we generate Ω?
  - Gaussian
  - FFT
- We get Q by orthogonalizing B. However, if $\{\sigma_i\}_{i=1,n}$ decay slowly, $\|A - AQ^T Q\|$ can be big.
- We can overcome this issue with a few power iterations:

$$B = \Omega A (A^T A)^q$$

- To avoid round-off errors, we need to reorthogonalize B between each application of A and A^T.

Communications

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

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

Figure : Computation and communication costs on one GPU.

- Random sampling has a flop overhead (oversampling + power iterations), but...
- ...much better communications efficiency

Experimental Setups

- Compiled with `gcc` 4.4.7 and `nvcc` (CUDA 6.0.1), with `-O3` flag, linked to threaded MKL (version 10.3).
- Machine: two eight-core Genuine Intel(R) 2.60GHz CPUs and three NVIDIA Tesla K40c GPUs.

	Matrix Name		
	POWER	EXPONENT	HAPMAP
σ_i	$(i+1)^{-3}$	$10^{-i/10}$	--
σ_0	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
m	500,000	500,000	503,783
n	500	500	506
k	50	50	50
p	10	10	10
ℓ	60	60	60

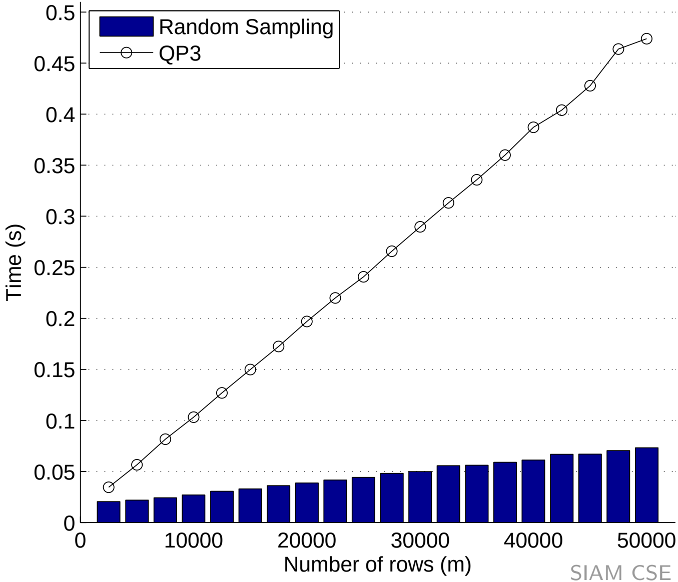
Table : Test matrices.

	QP3	Random Sampling		
		$q = 0$	$q = 1$	$q = 2$
POWER	4.47e-05	9.08e-05	4.59e-05	4.45e-05
EXPONENT	2.69e-05	5.18e-05	2.69e-05	2.69e-05
HAPMAP	5.99e-01	9.86e-01	8.74e-01	8.18e-01

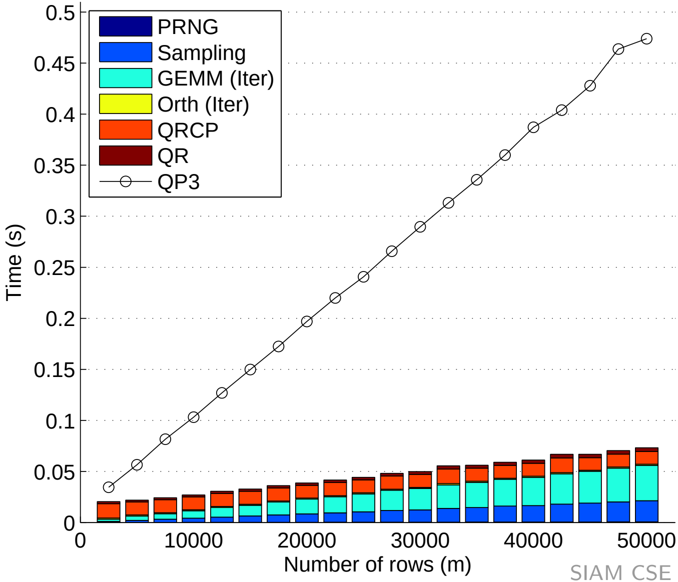
Figure : Approximation error norm $\|AP - QR\| / \|A\|$.

The same order of accuracy is already reached with $q = 0$.

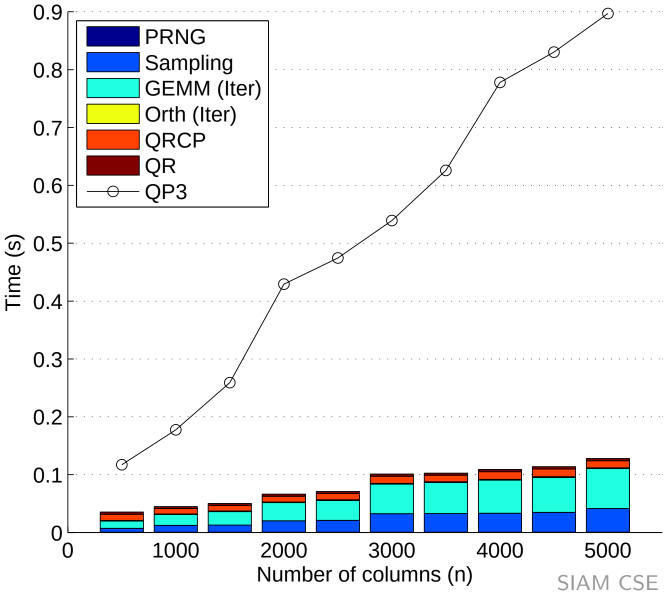
Time with different numbers of rows (m)



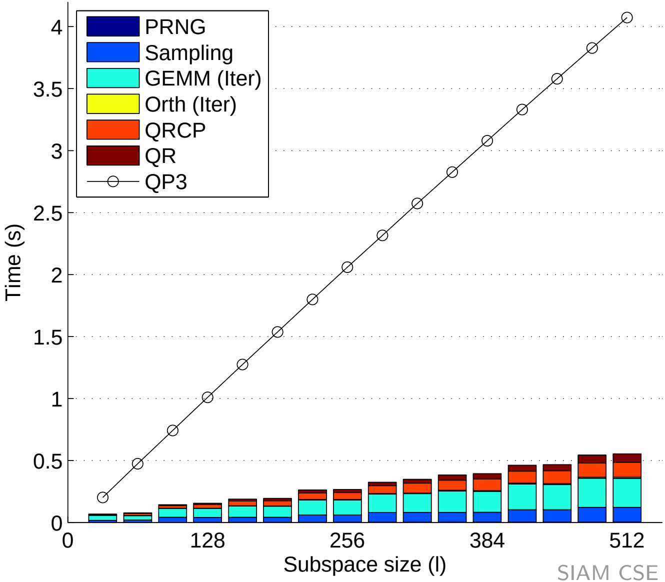
Time with different numbers of rows (m)



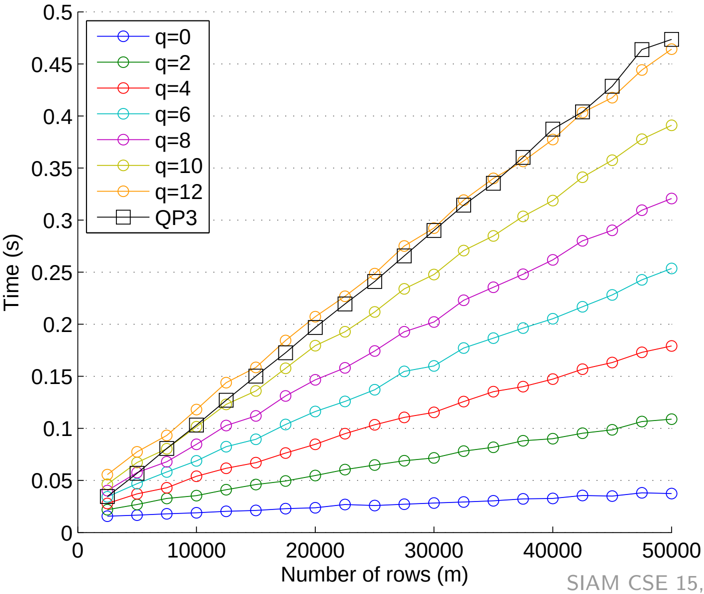
Time with different numbers of columns (n)



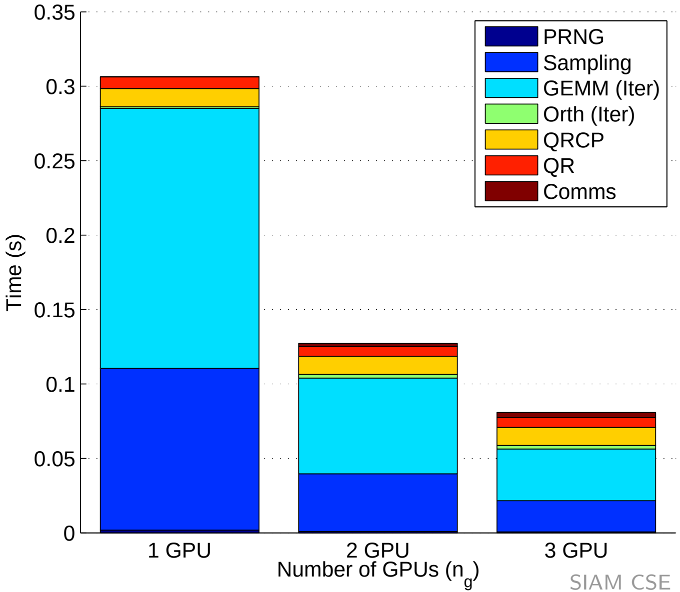
Time with different subspace sizes (ℓ)



Time with different numbers of iterations (q)



Time on 1, 2 and 3 GPUs



Conclusion

Summary: Random Sampling vs. QP3

- Comparable accuracy on the test matrices used.
- Small flop overhead but substantial communication improvement $\Rightarrow$ performance speedups above 13.
- Scaling on multiple GPUs.

Reference

- MS 266 & 291: Randomized Algorithms in Numerical Linear Algebra
  - Performance of Computing Low-Rank Approximation on Hybrid CPU/GPU Architectures

?

Thanks!

Questions?