Mixed-precision orthogonalization schemes:
Performance on multicore CPU with GPUs

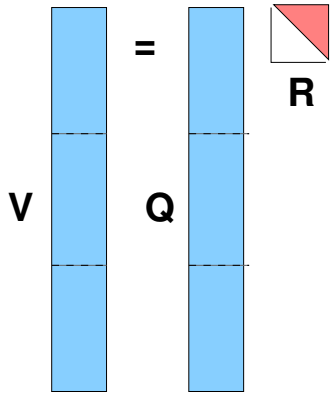
Ichitaro Yamazaki, Stanimire Tomov, Jakub Kurzak, Jack Dongarra
University of Tennessee, Knoxville, USA

Jesse Barlow
Pennsylvania State University, Pennsylvania, USA

Latest Advances in Scalable Algorithms for Large-scale Systems
Austin, TX, USA, 11-16-2015

TSQR: Tall-Skinny QR

orthogonalizes a set of dense columns vectors V (m-by-n, $m \gg n$),



where Q is a set of orthogonal vectors, and R is upper triangular.

- important computational kernels:
  - 1st part of this talk: $n = O(10)$
    "Communication-avoiding" s-step Krylov ($n = s$)
  - 2nd part of this talk: $n = O(100)$
    Random sampling for low-rank matrix approximation ($n = k + \ell$)

TSQR Algorithms for $n = O(10)$

Many ways to compute *TSQR*:

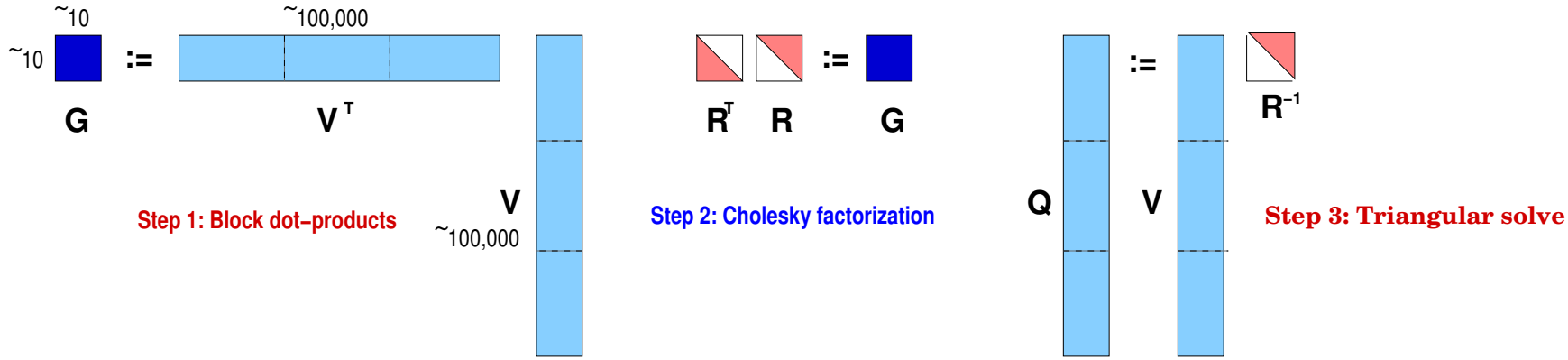
- Householder QR (with $O(s)$ reductions)
  - Householder transform each column based on BLAS-1,2 xGEQR2
- Modified Gram-Schmidt (with $O(s)$ reductions)
  - ortho each column against each column based on BLAS-2,1 xGEMV, xDOT
- Classical Gram-Schmidt (with $O(s)$ reductions)
  - ortho each column against prev columns based on BLAS-2,1 xGEMV, xDOT
- Cholesky QR (or SVQR) (with $O(1)$ reductions)
  - ortho all columns against prev columns based on BLAS-3 xGEMM, xTRSM
- CAQR (with $O(1)$ reductions)
  - ortho all columns against prev columns based on tree-reduction BLAS-1,2 xGEQR2

CholQR factorization for *TSQR* [A. Stathopoulos and K. Wu. 2002]

Step 1 Gram-matrix formation $G := V^T V$ ($\frac{1}{2}ns^2$ ops on GPUs).

Step 2 Cholesky factorization $R^T R := G$ ($\frac{1}{6}s^3$ ops on CPUs).

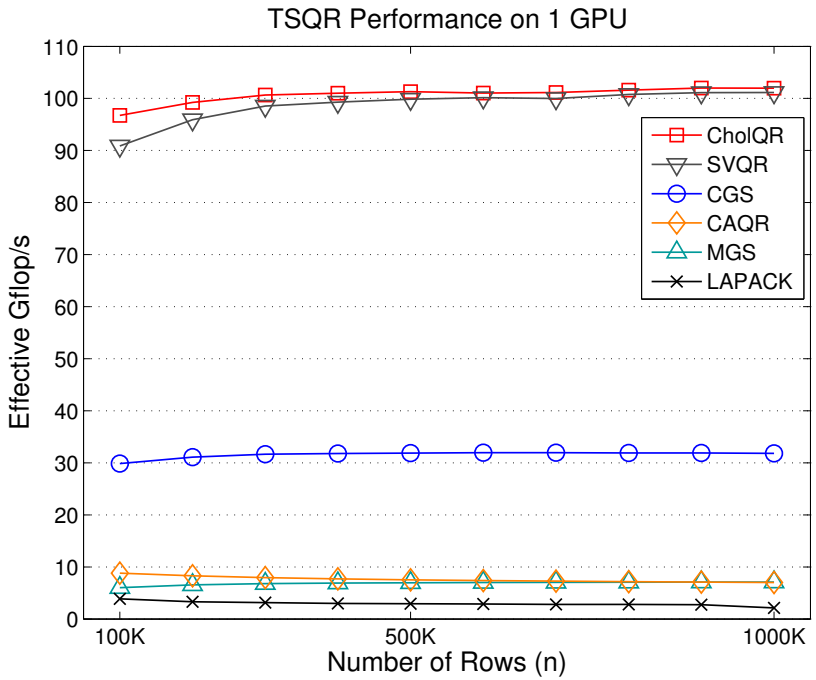
Step 3 Backward-substitution $Q := VR^{-1}$ ($\frac{1}{2}ns^2$ ops on GPUs).

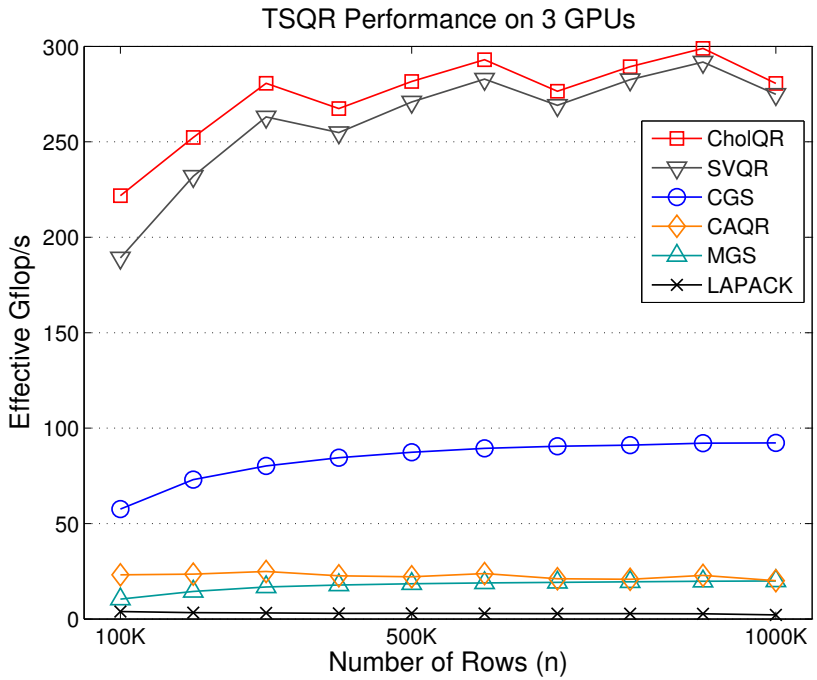


- most of flops using BLAS-3.
- only one global communication (reduction to form G).

$\rightarrow$ great performance on modern computer

TSQR Performance (16-core SandyBridge with three M2090 Fermi, $s = 30$) [IPDPS'14]





- CholQR shows superior performance based on BLAS-3
- great performance on distributed-memory system with one global reduce [SC'14]

TSQR Stability:

- trade-off between performance and stability
  - CholQR obtains great performance with minimum communication.
  - its orthogonality error depends quadratically on condition number of V.

	$\|I - Q^T Q\|$	# flops, GPU kernel	# GPU-CPU comm.
MGS	$O(\epsilon \cdot \kappa(V))$	$2ns^2$, BLAS-1 xDOT	$O(s^2)$
CGS	$O(\epsilon \cdot \kappa(V)^{s-1})$	$2ns^2$, BLAS-2 xGEMV	$O(s)$
CholQR	$O(\epsilon \cdot \kappa(V)^2)$	$2ns^2$, BLAS-3 xGEMM	$O(1)$
SVQR	$O(\epsilon \cdot \kappa(V)^2)$	$2ns^2$, BLAS-3 xGEMM	$O(1)$
CAQR	$O(\epsilon)$	$4ns^2$, BLAS-1,2 xGEQR2	$O(1)$

- it often requires reorthogonalization
- it could fail if $\kappa(V) > \epsilon^{-1/2}$,
  e.g., if $\kappa(V) > 10^8$ for working double precision.

Mixed Precision CholQR

- Remove "square" in error bound by selectively using "doubled" precision:

  Step 1 Gram-matrix formation $G := V^T V$ (V in double)
  doubled-precision on GPUs.

  Step 2 Cholesky factorization $R^T R := G$
  doubled-precision on CPUs.

  Step 3 Backward-substitution $Q := VR^{-1}$
  working-precision on GPUs.

$\rightarrow$ orthogonality error depends linearly on $\kappa(V)$ [SISC'15]
$\|I - Q^T Q\| \leq O(\epsilon\kappa(V) + (\epsilon\kappa(V))^2)$ and $\|Q\| \leq 1 + O(\epsilon\kappa(V)$

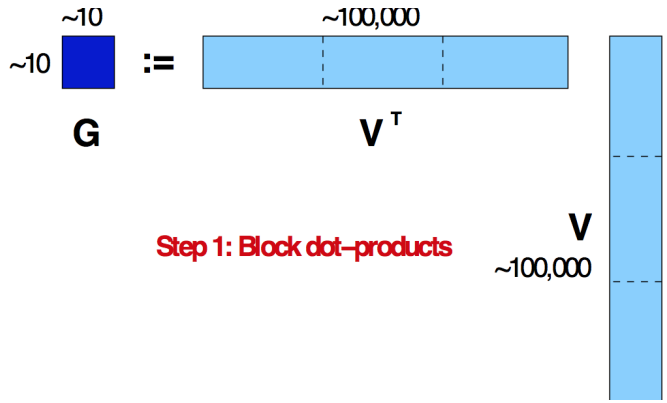
$\rightarrow$ may require software-emulated arithmetics for doubled-precision
e.g., for working 64-bit double,
- we used double-double to emumerate quadruple precision
  [Y. Hida, X. Li, and D. Bailey, '00]
- computation increases by 8.5×
- but with small communication overhead (only volume doubles to form G)
  - CholQR is communication-bounded, $\frac{1+n}{2}$ flops per read
  - mixed-precidion CholQR reads V in double
    and accumulates intermediate results in double-double

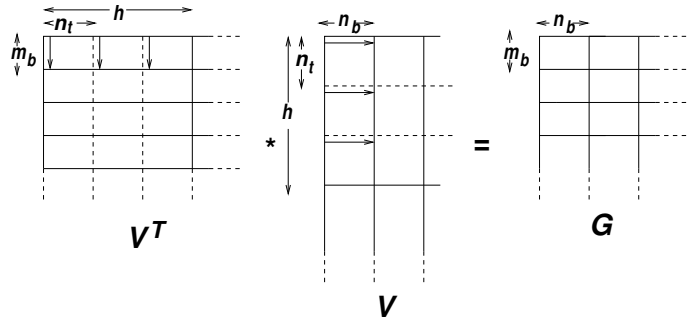
Batched GPU kernels for block inner-products

"batched" xGEMM/xSYRK kernel

1. thread block to compute partial block product
2. local reduction to compute local Gram matrix
3. global all reduce to form final Gram matrix

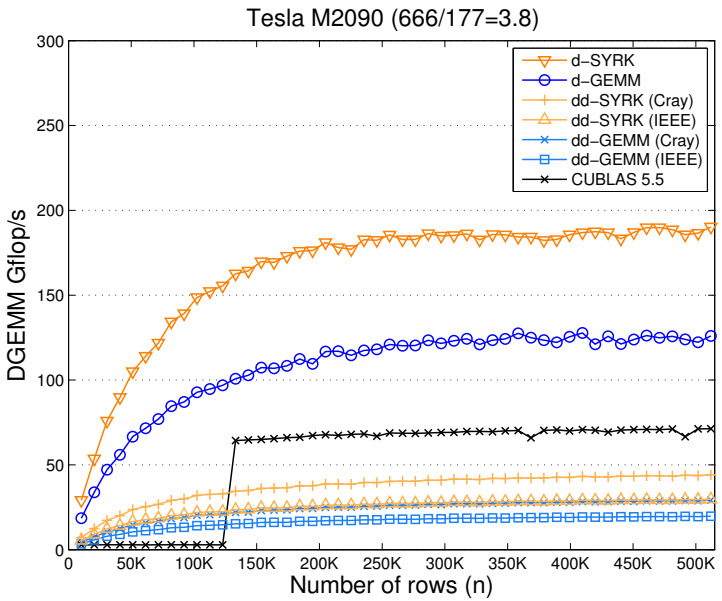
brute-force tune for dimension and precision on GPU
(by Tim Dong)

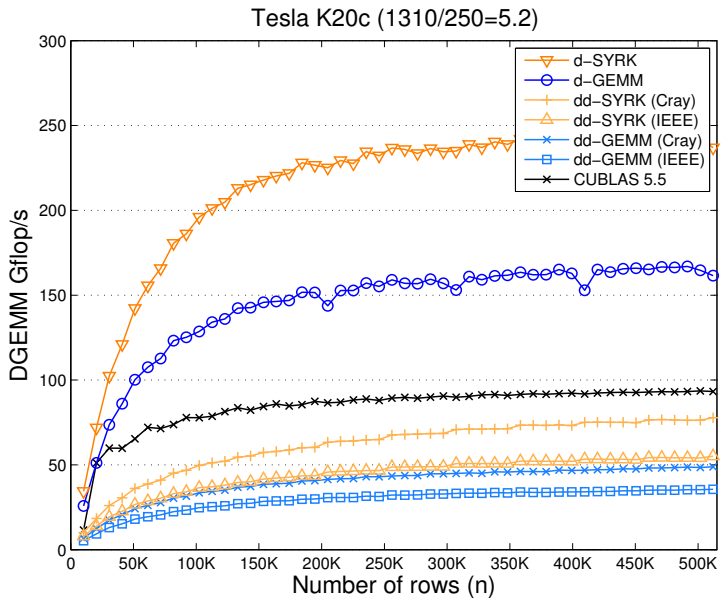




Block inner-products in double-double vs. double precision

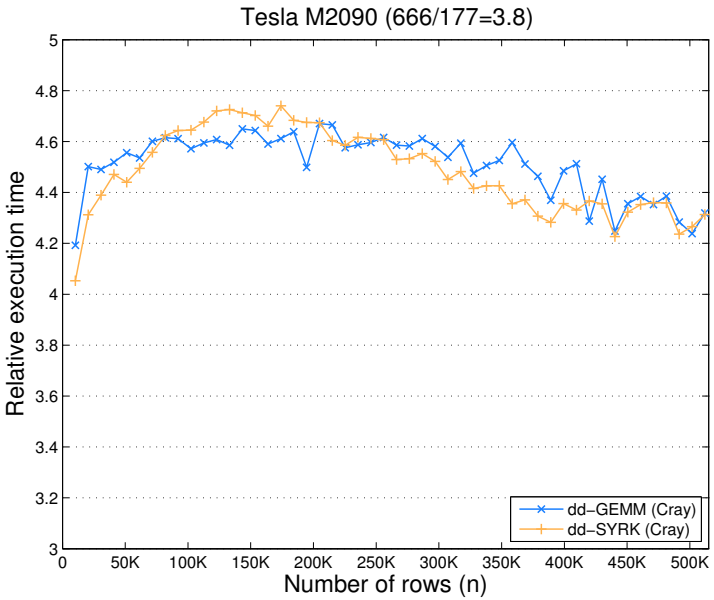
- optimized batched xGEMM kernel for block inner-product, $n = O(10^5), s = O(10)$.
  - 1.7× speedups over CUBLAS 5.5 for d-precision.
    30% of the peak based on memory bandwidth
  - 16× more ops for dd-precision (Cray).
    - input matrix in d-precision, compute intermediate results in dd-precision

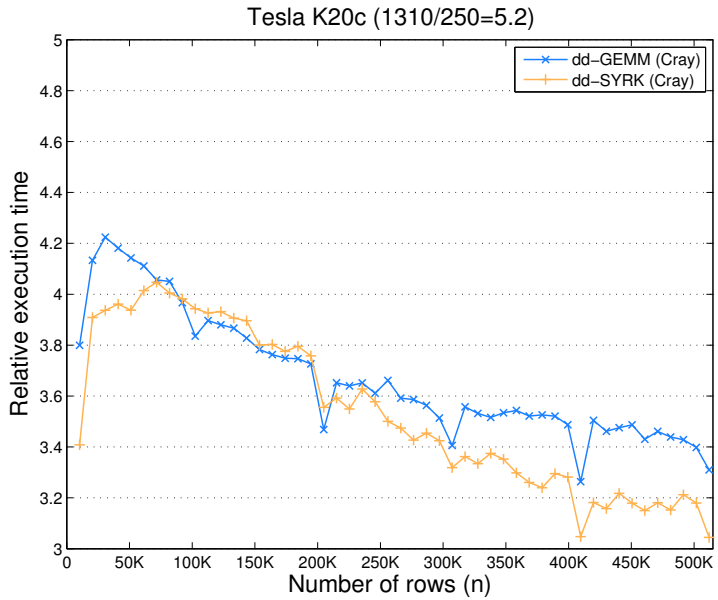




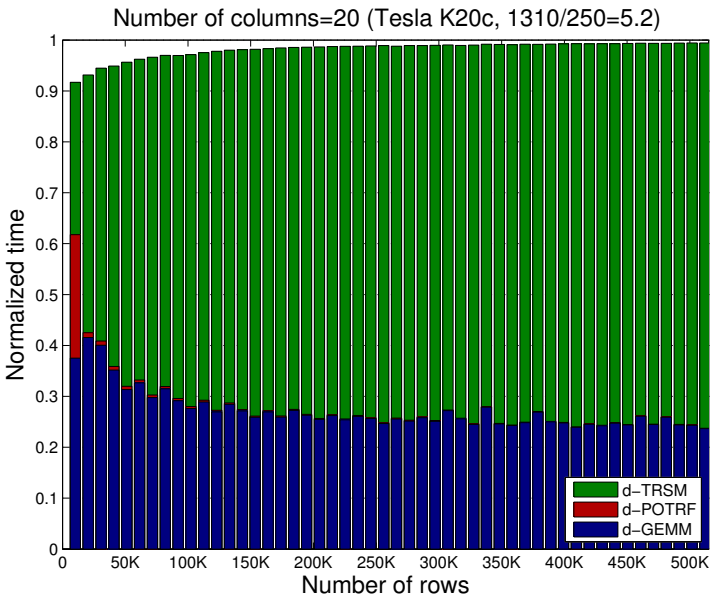
Block inner-products in double vs. double-double precision

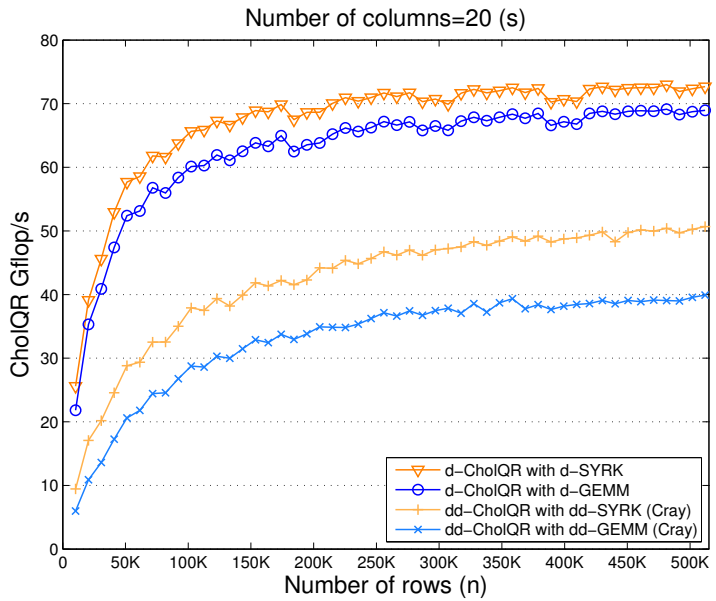
- optimized batched xGEMM kernel for block inner-product, $n = O(10^5), s = O(10)$.
  - 1.7× speedups over CUBLAS 5.5 for d-precision.
  - 16× more ops for dd-precision (Cray).
  - memory-bound operation.
    $\rightarrow$ 4.5× or 3.5× slower on Fermi or Kepler.





Mixed Precision CholQR Performance



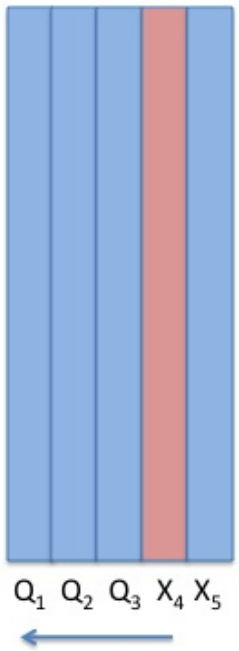


- only about 30% of d-CholQR in d-GEMM.
- dd-CholQR 8.5× ops, but 1.7× slower than d-CholQR
  - dd-CholQR may be competitive with 2×d-CholQR
    d- or dd-CholQR could fail if $\kappa(V) > \epsilon^{-1/2}$ or $> \epsilon^{-1}$
- CA-Krylov performance can be improved [VECPAR'14]
  - reduced orthogonalization time, larger step size, or faster convergence

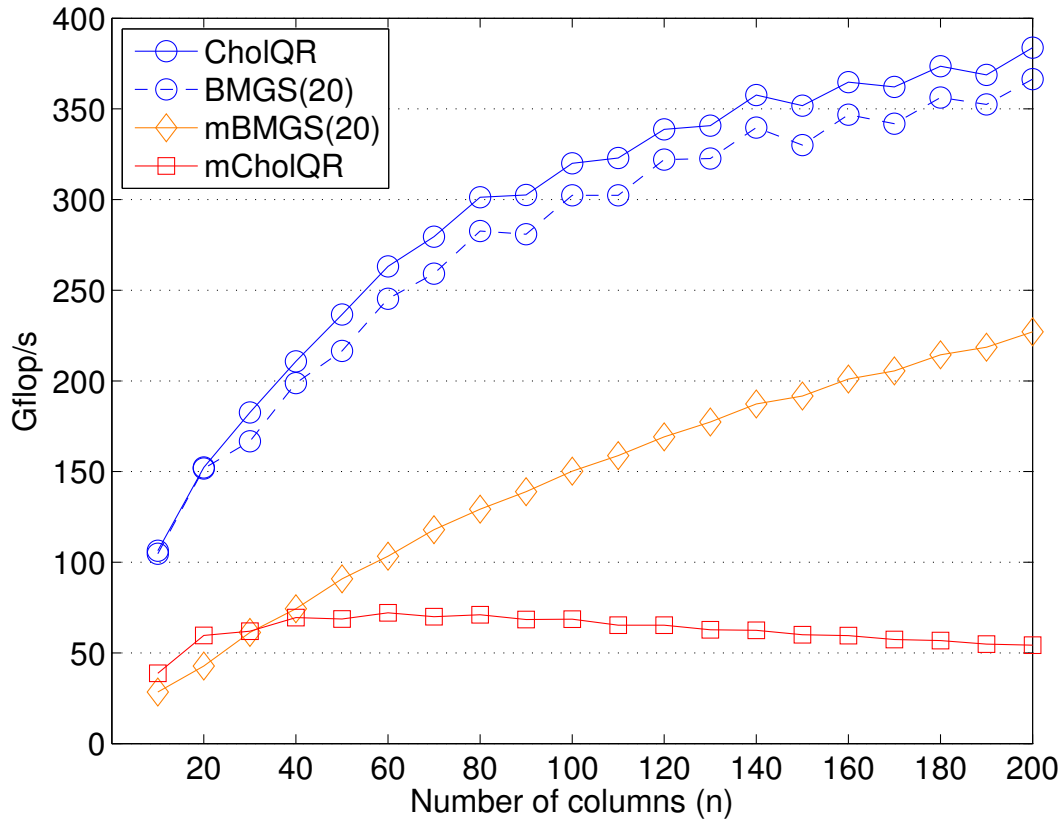
Extension to orthogonalize many columns [ScalA'15]

- Motivation: random sampling of large sparse matrix, $n = O(100)$
- CholQR performs $\frac{1+n}{2}$ flops on each numerical value read.
- As n increases,
  - it becomes more compute-bound
  - mixed-precision CholQR becomes slower

- Use mixed-precision CholQR in block MGS
  - BMBS and then CholQR
    same bound by using mCholQR+CholQR
    with comp. overhead of $\frac{8.5}{n_t}\times$
  - restarted CA-Krlov
    to orthogonalize s vectors at a time
    to generate total of n basis vectors



Performance of BMGS: $m = 100K$ on one GPU

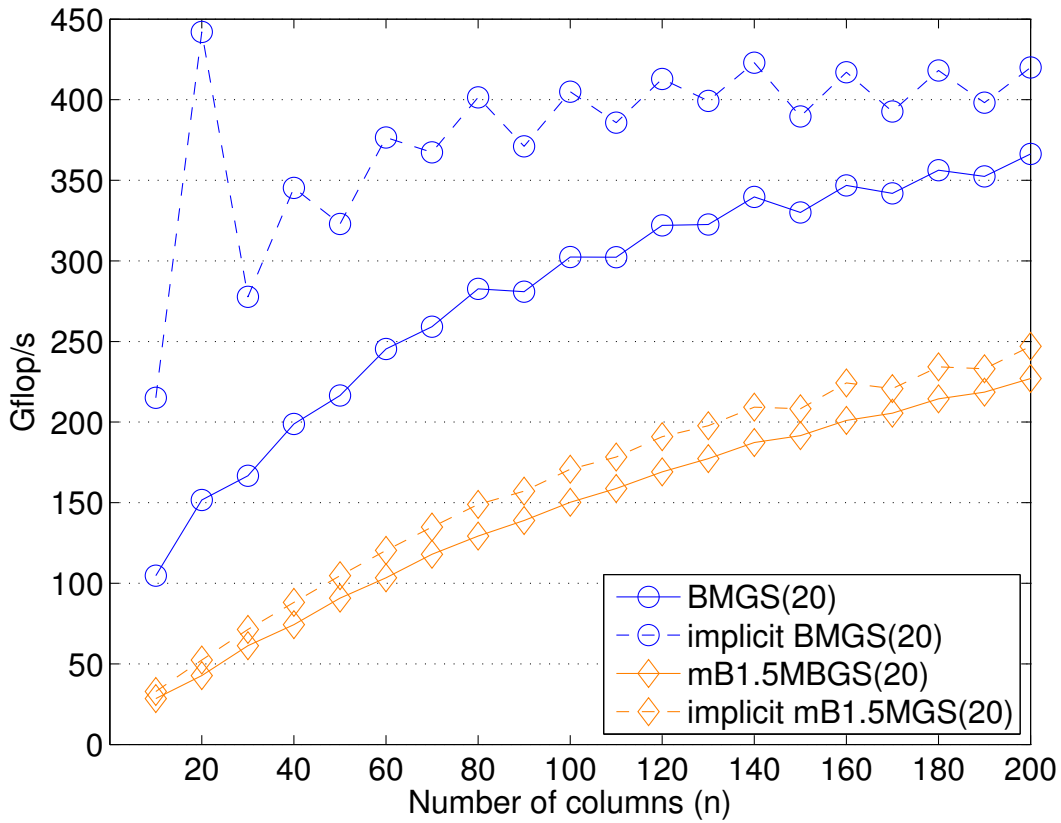


with $n = 200$,

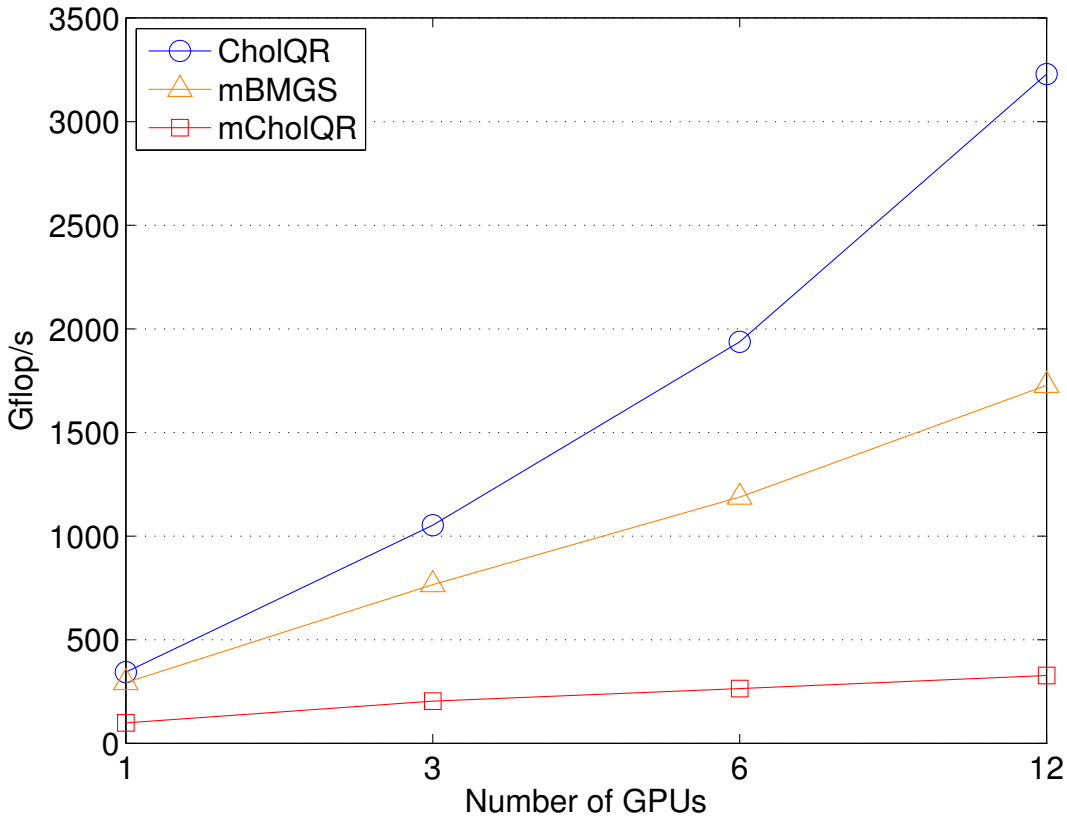
- mCholQR was 7.1× slower than CholQR (with 8.5× ops)
- mB1.5MGS was 1.7× slower than CholQR (with 1.8× ops) and was 4.1× faster than mCholQR

Algorithmic variant with implicit Q_j:

- each Q_j implicitly stored with X_j and $R_{j,j}$, i.e., $Q_j := X_j R_{j,j}^{-1}$,
  BCGS: $X_j := X_j - X_{1:j-1}(G_j^{-1}(X_{1:j-1}^T X_j))$
- stable if X_j is well conditioned
  $\rightarrow$ use implicit form for re-orthogonalization, i.e., $Q_j := Q_j \widehat{R}_{j,j}^{-1}$



Performance of BMGS: $(m, n) = (500K, 200)$ on multiple GPUs



compared to CholQR,

- B1.5MGS communicates $n_t\times$ more
- mCholQR has greater bottleneck with ddPOTRF

Final Remarks

- Mixed-precision CholQR
  - performs 8.5× more computation
  - reduces 2× more words, $O(n^2)$ with $n \ll m$
  - was 1.4× slower when $n = O(10)$
  - was 7.1× slower when $n = O(100)$
  - smaller overhead if supported by hardware (e.g., single)
- BMGS combined with dd-CholQR + d-CholQR for TSQR
  - performs $\frac{8.5}{n_t}\times$ more computation, where n_t is number of block columns
  - was 1.7× slower when $n = O(100)$
  - communicates $n_t\times$ more often

Current Work

- Numerical studies and theoretical bounds
- CAQR [J. Demmel, L. Grigori, M. Hoemmen, J. Langou, 2012] using "batched" QR

Thank you!!