communication-avoiding Cholesky-QR2 for rectangular matrices (CA-CQR2)

Edward Hutter and Edgar Solomonik

Laboratory for Parallel Numerical Algorithms
Department of Computer Science
University of Illinois at Urbana-Champaign

IPDPS 2019
Motivation for reducing algorithmic communication costs

Communication and synchronization increasingly dominating algorithm performance on modern architectures

\( \alpha - \beta - \gamma \) cost model

- **\( \alpha \)** - cost to send zero-byte message
- **\( \beta \)** - cost to inject byte of data into network
- **\( \gamma \)** - cost to perform flop with register-resident data

Architectural trend: \( \alpha \gg \beta \gg \gamma \)

**Figure:** Horizontal (internode network) communication along critical path

Communication-avoiding algorithms for **most** dense matrix factorizations present in numerical libraries

**Goal:** A QR factorization algorithm that prioritizes minimizing synchronization and communication cost
3D algorithms utilize available extra memory to reduce communication asymptotically.

We introduce CA-CQR2, a novel practical 3D QR factorization algorithm
- extends CholeskyQR2 algorithm to arbitrary matrices
- requires $O\left(\left(\frac{Pm^2}{n^2}\right)^{\frac{1}{6}}\right)$ less communication than known 2D QR algorithms for $m \times n$ matrices across $P$ processes
- obtains 3x speedups over ScaLAPACK on 1024 nodes
- utilizes first distributed-memory implementation of recursive 3D Cholesky factorization

CA-CQR2’s asymptotic communication reduction incurs tradeoffs
- increased computation ($2 - 4x$ more flops than Householder QR)
- constrained applicability (matrix must be sufficiently well-conditioned)
- requires $O\left(\left(\frac{Pm}{n}\right)^{\frac{1}{3}}\right)$ more memory than known 2D QR algorithms for $m \times n$ matrices across $P$ processes
QR Strong scaling performance on Stampede2

Strong Scaling on Stampede2, 8388608 x 2048 matrix

Figure: Strong scaling for $m \times n$ matrices
Weak Scaling on Stampede2, Up to 8388608 x 8192 matrix

Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count
α − β model captures communication (β) and synchronization (α) costs over P processors.

ScalAPACK’s PGEQRF is communication-optimal assuming minimal memory (2D):

\[ T_{PGEQRF}^{α, β} = \mathcal{O} \left( n \log P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta \right) \]

\[ M_{PGEQRF} = \mathcal{O} \left( \frac{mn}{P} \right) \]

CAQR factors panels using TSQR to reduce synchronization\(^1\) (2D):

\[ T_{CAQR}^{α, β} = \mathcal{O} \left( \sqrt{P} \log^2 P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta \right) \]

\[ M_{CAQR} = \mathcal{O} \left( \frac{mn}{P} \right) \]

CA-CQR2 leverages extra memory to reduce communication (3D):

\[ T_{CA-CQR2}^{α, β} = \mathcal{O} \left( \left( \frac{Pn}{m} \right)^{\frac{2}{3}} \log P \cdot \alpha + \left( \frac{n^2 m}{P} \right)^{\frac{2}{3}} \cdot \beta \right) \]

\[ M_{CA-CQR2} = \mathcal{O} \left( \left( \frac{n^2 m}{P} \right)^{\frac{2}{3}} \right) \]

3D algorithms exist in theory\(^2\)\(^3\)\(^4\), but **CA-CQR2 is the first practical approach**

---

1. J. Demmel et al., "Communication-optimal Parallel and Sequential QR and LU Factorizations", SISC 2012
3. E. Solomonik et al., "A communication-avoiding parallel algorithm for the symmetric eigenvalue problem", SPAA 2017
4. G. Ballard et al., "A 3D Parallel Algorithm for QR Decomposition", SPAA 2018
Instability of Cholesky-QR

QR factorization algorithms used in practice stem from processes of orthogonal triangularization for their superior numerical stability

\[ Q_n Q_{n-1} \ldots Q_1 A = R \]

The Cholesky-QR algorithm is a simple algorithm that follows a numerically unstable process of triangular orthogonalization

\[ AR_1^{-1} R_2^{-1} \ldots R_n^{-1} = Q \]

\[ [Q, R] \leftarrow \textbf{Cholesky-QR} (A) \]

- \( B \leftarrow A^T A \) \quad \text{\( B \) may be indefinite!}
- \( R^T R \leftarrow B \) \quad \text{Possible failure in Cholesky factorization!}
- \( Q \leftarrow AR^{-1} \) \quad \text{\( R \) may have lost all accuracy! \( Q \) may lost orthogonality!}
The Cholesky-QR2 algorithm can achieve stability through iterative refinement:\(^1\)

\[
[Q, R] \leftarrow \text{Cholesky-QR2} (A)
\]

\[
\begin{align*}
Z, R_1 & \leftarrow \text{CQR}(A) \\
Q, R_2 & \leftarrow \text{CQR}(Z) \\
R & \leftarrow R_2 R_1
\end{align*}
\]

- leverages near-perfect conditioning of \(Z\) in a second iteration:\(^1\)
- \(A = ZR_1 = QR_2 R_1\), from \(A^T A = R_1^T Z^T Z R_1 = R_1^T R_2^T Q^T Q R_2 R_1\), where \(R_2\) corrects initial \(R_1\)
- numerical breakdown still possible if first iteration loses positive definiteness in \(A^T A\) via \(\kappa(A) \leq 1/\sqrt{\varepsilon}\)

Shifted Cholesky-QR\(^2\) can attain a stable factorization for any matrix \(\kappa(A) \leq 1/\varepsilon\)

- the eigenvalues of \(A^T A\) are shifted to prevent loss of positive definiteness
- three Cholesky-QR iterations required, essentially 3 – 6x more flops than Householder approaches

\(^2\) T. Fukaya et al., "Shifted CholeskyQR for computing the QR factorization of ill-conditioned matrices", Arxiv 2018
Scalability of Cholesky-QR2

Cholesky-QR2 (CQR2) can achieve superior performance on tall-and-skinny matrices\(^1\):

- Householder QR - \(2mn^2 - \frac{2n^3}{3}\) flops, Cholesky-QR2 - \(4mn^2 + \frac{5n^3}{3}\) flops

\[
T_{\text{Cholesky-QR2}}(m, n, P) = \mathcal{O}\left(\log P \cdot \alpha + n^2 \cdot \beta + \left(\frac{n^2 m}{P} + n^3\right) \cdot \gamma\right)
\]

CA-CQR2 parallelizes Cholesky-QR2 over a 3D processor grid, efficiently factoring any rectangular matrix

\(^1\)T. Fukaya et al., “CholeskyQR2: A communication-avoiding algorithm”, ScalA 2014
CA-CQR2’s communication-optimal parallelization

CA-CQR2 leverages known 3D algorithms for matrix multiplication\(^1\) and Cholesky factorization\(^2\)

A recursion tree for recursive Cholesky factorization and triangular inverse yields a tradeoff in communication and synchronization\(^2\)

A tunable 3D processor grid of dimensions \(c \times d \times c\) determines the replication factor \((c)\), the communication reduction \((\sqrt{c})\), and the number of simultaneous instances of 3D algorithms \((d/c)\)

---


Figure: Start with a tunable $c \times d \times c$ processor grid
Cost: $2 \log_2 c \cdot \alpha + \frac{2mn}{dc} \cdot \beta$
CA-CQR2 – Computation of Gram matrix

**Figure:** Reduce contiguous groups of size \( c \)

Cost: 
\[
2 \log_2 c \cdot \alpha + \frac{2n^2}{c^2} \cdot \beta + \frac{n^2}{c^2} \cdot \gamma
\]
Cost: $2 \log_2 \frac{d}{c} \cdot \alpha + \frac{2n^2}{c^2} \cdot \beta + \frac{n^2}{c^2} \cdot \gamma$
Figure: Broadcast missing pieces of $B$ along depth

Cost: $2 \log_2 c \cdot \alpha + \frac{2n^2}{c^2} \cdot \beta$
CA-CQR2 – Computation of CholeskyInverse

**Figure:** $\frac{d}{c}$ simultaneous 3D CholeskyInverse on cubes of dimension $c$

Cost: $O\left( c^2 \log c^3 \cdot \alpha + \frac{n^2}{c^2} \cdot \beta + \frac{n^3}{c^3} \cdot \gamma \right)$
Figure: \( \frac{d}{c} \) simultaneous 3D matrix multiplication or TRSM on cubes of dimension \( c \)

\[
Q = AR^{-1}
\]

Cost: \( \mathcal{O}(\log_2 c^3 \cdot \alpha + \left( \frac{mn}{dc} + \frac{n^2+nc}{c^2} \right) \cdot \beta + \frac{n^2m}{c^2d} \cdot \gamma) \)
Algorithmic cost analysis for CA-CQR2 and its competition

CA-CQR2’s cost expression expresses tunable tradeoffs

\[ T_{CA-CQR2}^{\alpha-\beta} (m, n, c, d) = \mathcal{O}\left( c^2 \log(d/c) \cdot \alpha + \left( \frac{mn}{dc} + \frac{n^2}{c^2} \right) \cdot \beta + \left( \frac{mn^2}{c^2d} + \frac{n^3}{c^3} \right) \cdot \gamma \right) \]

Requiring each processor to own a square submatrix \( \left( \frac{m}{d} = \frac{n}{c} \right) \) and enforcing \( P = c^2d \), CA-CQR2 finds an optimal processor grid that supports minimal communication

<table>
<thead>
<tr>
<th>1D Cholesky-QR2</th>
<th>2D ScaLAPACK</th>
<th>2D CAQR</th>
<th>3D CA-CQR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>messages</td>
<td>( \mathcal{O}(\log P) )</td>
<td>( \mathcal{O}(n \log P) )</td>
<td>( \mathcal{O}\left( \sqrt{P} \log^2 P \right) )</td>
</tr>
<tr>
<td>words</td>
<td>( \mathcal{O}(n^2) )</td>
<td>( \mathcal{O}\left( \frac{mn}{\sqrt{P}} \right) )</td>
<td>( \mathcal{O}\left( \frac{mn}{\sqrt{P}} \right) )</td>
</tr>
<tr>
<td>flops</td>
<td>( \mathcal{O}\left( \frac{n^2m}{P} + n^3 \right) )</td>
<td>( \mathcal{O}\left( \frac{mn^2}{P} \right) )</td>
<td>( \mathcal{O}\left( \frac{mn^2}{P} \right) )</td>
</tr>
<tr>
<td>memory</td>
<td>( \mathcal{O}\left( \frac{mn}{P} + n^2 \right) )</td>
<td>( \mathcal{O}\left( \frac{mn}{P} \right) )</td>
<td>( \mathcal{O}\left( \frac{mn}{P} \right) )</td>
</tr>
</tbody>
</table>

Minimal communication cost in a QR factorization is reflected by the surface area of the cubic volume of \( \mathcal{O}(mn^2/P) \) computation
We factor $m \times n$ matrices with $m \gg n$ to highlight the effect CA-CQR2’s communication reduction and algorithmic tradeoffs have on performance.

Scaling studies highlight the interplay between CA-CQR2’s increased arithmetic intensity and an architecture’s machine balance:

- ratio of peak-flops to network bandwidth is 8x higher on Stampede2\(^1\) than BlueWaters\(^2\)

We show only the most-performant variants at each node count of CA-CQR2 and ScaLAPACK’s PGEQRF:

- ScaLAPACK tuned over 2D processor grid dimensions and block sizes
- CA-CQR2 tuned over processor grid dimensions $d$ and $c$
- each tested/tuned over a number of resource configurations
- both algorithms use Householder’s flop cost in determining performance

---

\(^1\)Intel Knights Landing (KNL) cluster at TACC
\(^2\)Cray XE/XK hybrid machine at NCSA
Figure: Weak scaling for \( m \times n \) matrices so \( mn^2 \) scales linearly with node count
QR Strong scaling on Stampede2 and Blue Waters

Figure: Strong scaling for $m \times n$ matrices
CA-CQR2’s performance improvements over ScaLAPACK on Stampede2 range from 1.1 - 3.3x at 1024 nodes

**CA-CQR2 leverages current and future architectural trends**

- machines with highest ratio of peak node performance to peak injection bandwidth will benefit most
- asymptotic communication reductuction increasingly evident as we scale, despite overheads in synchronization and computation

These results motivate increasingly wide overdetermined systems, a **critical use case for solving linear least squares and eigenvalue problems**

Our study shows that **communication-optimal parallel QR factorizations can achieve superior performance and scaling up to thousands of nodes**\(^1\) \(^2\)

---

\(^1\) Our preprint detailing CA-CQR2 can be found at https://arxiv.org/abs/1710.08471

\(^2\) Our C++ implementation can be found at https://github.com/huttered40/CA-CQR2
I’d like to acknowledge the Department of Energy and Krell Institute for supporting this research via awarding me a DOE Computational Science Graduate Fellowship\(^1\)

We’d also like to acknowledge a number of computing centers for providing benchmarking resources

- Texas Advanced Computing Center (TACC) via Stampede2\(^2\)
- National Center for Supercomputing Applications (NCSA) via Blue Waters\(^3\)
- Argonne Leadership Computing Facility (Cetus, Mira, Theta) for preliminary benchmarking

---

1 Grant number DE-SC0019323
2 Allocation TG-CCR180006
3 Awards OCI-0725070 and ACI-1238993
CA-CQR2 building block #1 – 3D Matrix Multiplication

Figure: 3D algorithm for square matrix multiplication \(^1\) \(^2\) \(^3\)

\[ C = AB \]

Broadcast across rows  \hspace{1cm} \text{Broadcast along columns} \hspace{1cm} \text{AllReduce along depth}

\[ T_{3D-MM}(n, P) = \mathcal{O} \left( \log P \cdot \alpha + \frac{n^2}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^3}{P} \cdot \gamma \right) \]

\(^1\)Bersten 1989, "Communication-efficient matrix multiplication on hypercubes"

\(^2\)Aggarwal, Chandra, Snir 1990, "Communication complexity of PRAMs"

\(^3\)Agarwal et al. 1995, "A three-dimensional approach to parallel matrix multiplication"
We can embed the recursive definitions of Cholesky factorization and triangular inverse to find matrices $R, R^{-1}$

Tuning the recursion tree yields a tradeoff in horizontal bandwidth and synchronization\(^1\)

\[
\begin{bmatrix}
    L_{11} & L_{11}^{-1} \\
    L_{21} & L_{21}^{-1}
\end{bmatrix} \leftarrow \text{CholeskyInverse}(A)
\]

\[
\begin{align*}
L_{21} & \leftarrow A_{21} L_{11}^{-T} \\
\begin{bmatrix}
    L_{11} & L_{11}^{-1} \\
    L_{22} & L_{22}^{-1}
\end{bmatrix} & \leftarrow \text{CholeskyInverse}(A_{22} - L_{21}^T L_{21}) \\
L_{21}^{-1} & \leftarrow -L_{22}^{-1} L_{21} L_{11}^{-1}
\end{align*}
\]

\[
T_{\text{CholeskyInverse3D}}(n, P) = O\left( P^{\frac{2}{3}} \log P \cdot \alpha + \frac{n^2}{P^{\frac{2}{3}}} \cdot \beta + \frac{n^3}{P} \cdot \gamma \right)
\]

\[
T_{\text{ScalAPACK}}(n, P) = O\left( \sqrt{P} \log P \cdot \alpha + \frac{n^2}{\sqrt{P}} \cdot \beta + \frac{n^3}{P} \cdot \gamma \right)
\]

\(^1\)A. Tiskin 2007, "Communication-efficient generic pairwise elimination"
Figure: Start with a tunable $c \times d \times c$ processor grid
Figure: Compute Gram matrix $A^T A$
Figure: Compute 3D CholeskyInverse on processor grid cubes of dimension $c$
Figure: Compute 3D matrix multiplication or 3D TRSM on processor grid cubes of dimension $c$.

$$Q = AR^{-1}$$
The advantage of using a tunable grid lies in the ability to frame the shape of the grid around the shape of rectangular $m \times n$ matrix $A$. Optimal communication can be attained by ensuring that the grid perfectly fits the dimensions of $A$, or that the dimensions of the grid are proportional to the dimensions of the matrix. We derive the cost for the optimal ratio $\frac{m}{d} = \frac{n}{c}$ below. Using equation $P = c^2 d$ and 

$$\frac{m}{d} = \frac{n}{c},$$

solve for $d, c$ in terms of $m, n, P$. Solving the system of equations yields $c = \left( \frac{Pn}{m} \right)^{\frac{1}{3}}, d = \left( \frac{Pm^2}{n^2} \right)^{\frac{1}{3}}$. We can plug these values into the cost of Cholesky-QR2_Tunable to find the optimal cost.

The cost of Cholesky-QR2_Tunable is

$$T^{\alpha - \beta}_{\text{Cholesky-QR2-Tunable}} \left( m, n, \left( \frac{Pn}{m} \right)^{\frac{1}{3}}, \left( \frac{Pm^2}{n^2} \right)^{\frac{1}{3}} \right) = O \left( \left( \frac{Pn}{m} \right)^{\frac{2}{3}} \log P \cdot \alpha \right)$$

$$+ \frac{\left( \frac{Pn}{m} \right)^{\frac{1}{3}} mn + n^2 \left( \frac{Pm^2}{n^2} \right)^{\frac{1}{3}}}{\left( \frac{Pm^2}{n^2} \right)^{\frac{1}{3}} \left( \frac{Pn}{m} \right)^{\frac{2}{3}}} \beta + \frac{n^3 \left( \frac{Pm^2}{n^2} \right)^{\frac{1}{3}} + n^2 m \left( \frac{Pn}{m} \right)^{\frac{1}{3}}}{\left( \frac{Pn}{m} \right) \left( \frac{Pm^2}{n^2} \right)^{\frac{1}{3}}} \gamma$$

$$= O \left( \left( \frac{Pn}{m} \right)^{\frac{2}{3}} \log P \cdot \alpha + \left( \frac{n^2 m}{P} \right)^{\frac{2}{3}} \cdot \beta + \frac{n^2 m}{P} \cdot \gamma \right)$$

<table>
<thead>
<tr>
<th>Grid shape</th>
<th>Metric</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal</td>
<td># of messages</td>
<td>$O \left( \left( \frac{Pn}{m} \right)^{\frac{2}{3}} \log P \right)$</td>
</tr>
<tr>
<td></td>
<td># of words</td>
<td>$O \left( \left( \frac{n^2 m}{P} \right)^{\frac{2}{3}} \right)$</td>
</tr>
<tr>
<td></td>
<td># of flops</td>
<td>$O \left( \left( \frac{n^2 m}{P} \right) \right)$</td>
</tr>
<tr>
<td></td>
<td>Memory footprint</td>
<td>$O \left( \left( \frac{n^2 m}{P} \right)^{\frac{2}{3}} \right)$</td>
</tr>
</tbody>
</table>
QR Weak Scaling on BlueWaters, $65536a \times 2048b$ matrix

(a,b), Nodes=$16a^2b^2$

Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count
QR Weak Scaling on BlueWaters, 262144*a x 1024*b matrix

Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count
Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count.
QR Weak Scaling on Stampede2, 262144*a x 8192*b matrix

Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count
Weak scaling on Stampede2 and Blue Waters

Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count
QR Weak Scaling on Stampede2, 1048576*a x 2048*b matrix

Figure: Weak scaling for $m \times n$ matrices so $mn^2$ scales linearly with node count
Weak scaling on Stampede2 and Blue Waters

QR Weak Scaling on Stampede2, $2097152 \times a \times 1024 \times b$ matrix

Figure: Weak scaling for $m \times n$ matrices so $m n^2$ scales linearly with node count
Figure: Strong scaling for QR factorization
Figure: Strong scaling for QR factorization
QR Strong Scaling on Stampede2, 524288 x 8192 matrix

Figure: Strong scaling for QR factorization
Strong scaling on Stampede2 and Blue Waters

QR Strong Scaling on Stampede2, 2048576 x 4096 matrix

Figure: Strong scaling for QR factorization

Edward Hutter and Edgar Solomonik
Strong scaling on Stampede2 and Blue Waters

Figure: Strong scaling for QR factorization
QR Strong Scaling on Stampede2, 33554432 x 1024 matrix

Figure: Strong scaling for QR factorization
Weak scaling performance on Blue Waters with 16 MPI processes/node

Figure: Weak scaling for matrices with dimensions given in legend
Weak scaling performance on Blue Waters with 16 MPI processes/node

Figure: Weak scaling for matrices with dimensions given in legend
Weak scaling performance on Blue Waters with 16 MPI processes/node

Figure: Weak scaling for matrices with dimensions given in legend
Weak scaling performance on Stampede2 with 64 MPI processes/node

Figure: Weak scaling for matrices with dimensions given in legend
Weak scaling performance on Stampede2 with 64 MPI processes/node

Weak Scaling, $524288 \times a \times 2048 \times b$

Gigaflops/s/Node

$\begin{pmatrix} 2,1 & 1,2 & 2,2 & 4,2 & 8,2 & 4,4 & 8,4 \end{pmatrix}$

$\begin{pmatrix} a, b \end{pmatrix}$, Nodes = $8 \times a \times b \times b$

ScaLAPACK-$(512ab,32,64,1)$

ScaLAPACK-$(512ab,64,64,1)$

CA-CQR2-$(64a/b,1,64,1)$

CA-CQR2-$(128a/b,0,16,4)$

Figure: Weak scaling for matrices with dimensions given in legend
Weak scaling performance on Stampede2 with 64 MPI processes/node

Figure: Weak scaling for matrices with dimensions given in legend
Figure: Weak scaling for matrices with dimensions given in legend
Strong scaling performance on Blue Waters with 16 MPI processes/node

Figure: Strong scaling for matrices with dimensions given in legend
Strong scaling performance on Blue Waters with 16 MPI processes/node

Figure: Strong scaling for matrices with dimensions given in legend
Strong scaling performance on Stampede2 with 64 MPI processes/node

Figure: Strong scaling for matrices with dimensions given in legend
Strong scaling performance on Stampede2 with 64 MPI processes/node

Figure: Strong scaling for matrices with dimensions given in legend
Figure: Strong scaling for matrices with dimensions given in legend
Strong scaling performance on Stampede2 with 64 MPI processes/node

Figure: Strong scaling for matrices with dimensions given in legend