communication-optimal QR factorizations:
performance and scalability on varying architectures

Edward Hutter and Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

Blue Waters Symposium 2019
Motivation for reducing algorithmic communication costs

Communication and synchronization increasingly dominating algorithm performance on modern architectures
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
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: \( \alpha \gg \beta \gg \gamma \)
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: \( \alpha \gg \beta \gg \gamma \)

Communication-avoiding algorithms for most dense matrix factorizations present in numerical libraries
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: $\alpha \gg \beta \gg \gamma$

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

Goal: A QR factorization algorithm that prioritizes minimizing synchronization and communication cost
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: \( \alpha \gg \beta \gg \gamma \)

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

Goal: A QR factorization algorithm that prioritizes minimizing synchronization and communication cost

Our team uses BlueWaters to assess the scalability of new algorithms for numerical tensor algebra at massively large scale
## Architecture trends: machine balance decreasing

<table>
<thead>
<tr>
<th>machine</th>
<th>launch year</th>
<th>peak node perf (Gflops/s)</th>
<th>peak injection bandwidth (Gwords/sec)</th>
<th>machine balance (words/flop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCI Red</td>
<td>1997</td>
<td>0.666</td>
<td>0.4</td>
<td>1/1.665</td>
</tr>
<tr>
<td>ANL BG/P</td>
<td>2007</td>
<td>13.6</td>
<td>1</td>
<td>1/13.6</td>
</tr>
<tr>
<td>ONL Jaguar</td>
<td>2009</td>
<td>124.8</td>
<td>2.2</td>
<td>1/56</td>
</tr>
<tr>
<td>ANL BG/Q</td>
<td>2012</td>
<td>205</td>
<td>2</td>
<td>1/102.5</td>
</tr>
<tr>
<td><strong>NCSA BlueWaters (XE)</strong></td>
<td>2012</td>
<td><strong>313.6</strong></td>
<td>9.6</td>
<td>1/32</td>
</tr>
<tr>
<td><strong>NCSA BlueWaters (XK)</strong></td>
<td>2012</td>
<td><strong>1320</strong></td>
<td>9.6</td>
<td>1/137.5</td>
</tr>
<tr>
<td>ORNL Titan</td>
<td>2013</td>
<td>1320</td>
<td>8</td>
<td>1/165</td>
</tr>
<tr>
<td>ANL Theta</td>
<td>2017</td>
<td>3000+</td>
<td>10.2</td>
<td>1/294</td>
</tr>
<tr>
<td><strong>TACC Stampede2</strong></td>
<td>2017</td>
<td><strong>3000+</strong></td>
<td>12.5</td>
<td>1/240</td>
</tr>
<tr>
<td>LLNL Sierra</td>
<td>2018</td>
<td>28000</td>
<td>12.5</td>
<td>1/2240</td>
</tr>
<tr>
<td>ORNL Summit</td>
<td>2018</td>
<td>44000</td>
<td>12.5</td>
<td>1/3520</td>
</tr>
</tbody>
</table>

Higher arithmetic intensity → higher performance on new architectures

BlueWaters not a favorable machine for communication-avoiding algorithms

Edward Hutter and Edgar Solomonik
Architecture trends: machine balance decreasing

Higher arithmetic intensity → higher performance on new architectures
Architecture trends: machine balance decreasing

<table>
<thead>
<tr>
<th>machine</th>
<th>launch year</th>
<th>peak node perf (Gflops/s)</th>
<th>peak injection bandwidth (Gwords/sec)</th>
<th>machine balance (words/flop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCI Red</td>
<td>1997</td>
<td>0.666</td>
<td>0.4</td>
<td>1/1.665</td>
</tr>
<tr>
<td>ANL BG/P</td>
<td>2007</td>
<td>13.6</td>
<td>1</td>
<td>1/13.6</td>
</tr>
<tr>
<td>ONL Jaguar</td>
<td>2009</td>
<td>124.8</td>
<td>2.2</td>
<td>1/56</td>
</tr>
<tr>
<td>ANL BG/Q</td>
<td>2012</td>
<td>205</td>
<td>2</td>
<td>1/102.5</td>
</tr>
<tr>
<td>NCSA BlueWaters (XE)</td>
<td>2012</td>
<td>313.6</td>
<td>9.6</td>
<td>1/32</td>
</tr>
<tr>
<td>NCSA BlueWaters (XK)</td>
<td>2012</td>
<td>1320</td>
<td>9.6</td>
<td>1/137.5</td>
</tr>
<tr>
<td>ORNL Titan</td>
<td>2013</td>
<td>1320</td>
<td>8</td>
<td>1/165</td>
</tr>
<tr>
<td>ANL Theta</td>
<td>2017</td>
<td>3000+</td>
<td>10.2</td>
<td>1/294</td>
</tr>
<tr>
<td>TACC Stampede2</td>
<td>2017</td>
<td>3000+</td>
<td>12.5</td>
<td>1/240</td>
</tr>
<tr>
<td>LLNL Sierra</td>
<td>2018</td>
<td>28000</td>
<td>12.5</td>
<td>1/2240</td>
</tr>
<tr>
<td>ORNL Summit</td>
<td>2018</td>
<td>44000</td>
<td>12.5</td>
<td>1/3520</td>
</tr>
</tbody>
</table>

Higher arithmetic intensity → higher performance on new architectures

BlueWaters **not** a favorable machine for communication-avoiding algorithms
Communication-avoiding Cholesky-QR2 (CA-CQR2)

3D algorithms utilize available extra memory to reduce communication asymptotically.
3D algorithms utilize available extra memory to reduce communication asymptotically.

We introduce CA-CQR2, a novel practical 3D QR factorization algorithm.
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 $m \times n$ matrices across $P$ processes
Communication-avoiding Cholesky-QR2 (CA-CQR2)

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 \( m \times n \) matrices across \( P \) processes
- requires \( \mathcal{O}\left(\left(\frac{Pm^2}{n^2}\right)^{1/6}\right) \) less communication than known 2D QR algorithms
Communication-avoiding Cholesky-QR2 (CA-CQR2)

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 \( m \times n \) matrices across \( P \) processes
- requires \( \mathcal{O} \left( \left( \frac{P m^2}{n^2} \right)^{1/6} \right) \) less communication than known 2D QR algorithms
- incurs a number of (increasingly profitable) tradeoffs
  - 2 – 4x more flops than Householder QR
  - matrix must be sufficiently well-conditioned
  - requires \( \mathcal{O} \left( \left( \frac{P m}{n} \right)^{1/3} \right) \) more memory than known 2D QR algorithms

All algorithms will be measured along the critical path instead of a volume measure.

Edward Hutter and Edgar Solomonik

4/28
Communication-avoiding Cholesky-QR2 (CA-CQR2)

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 $m \times n$ matrices across $P$ processes
- requires $O\left((Pm^2/n^2)^{1/6}\right)$ less communication than known 2D QR algorithms
- incurs a number of (increasingly profitable) tradeoffs
  - 2 — 4× more flops than Householder QR
  - matrix must be sufficiently well-conditioned
  - requires $O\left((Pm/n)^{1/3}\right)$ more memory than known 2D QR algorithms

All algorithms will be measured along the critical path instead of a volume measure
Communication-avoiding Cholesky-QR2 (CA-CQR2)

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 $m \times n$ matrices across $P$ processes
- requires $O \left( \left( \frac{P m^2}{n^2} \right)^{1/6} \right)$ less communication than known 2D QR algorithms
- incurs a number of (increasingly profitable) tradeoffs
  - $2 - 4x$ more flops than Householder QR
  - matrix must be sufficiently well-conditioned
  - requires $O \left( \left( \frac{P m}{n} \right)^{1/3} \right)$ more memory than known 2D QR algorithms

All algorithms will be measured along the critical path instead of a volume measure

Figure: Horizontal (internode network) communication along critical path

\[
T_{\text{near-neighbor-exchange}}(n, P) = a + n^*\beta
\]
\[
T_{\text{all-reduce}}(n, P) = f(P)\alpha + f(P)n^*\beta
\]
QR Strong scaling performance

Strong Scaling: Stampede2 and BlueWaters, m/n=4096

Figure: Strong scaling for $m \times n$ matrices
Figure: Strong scaling for $m \times n$ matrices
QR Strong scaling performance

Strong Scaling on Stampede2 and BlueWaters, $m/n=64$

**Figure**: Strong scaling for $m \times n$ matrices
Strong Scaling on Stampede2 and BlueWaters, m/n=8

Figure: Strong scaling for $m \times n$ matrices
QR Strong scaling performance

Strong Scaling on Stampede2 and BlueWaters, m/n=1

Figure: Strong scaling for $m \times n$ matrices
Competing costs of parallel QR factorization of $A_{m \times n}$

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

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

$$M_{\text{PGEQRF}} = \mathcal{O}\left(\frac{mn}{P}\right)$$

---

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
5. E. Hutter et al., "Communication-avoiding CholeskyQR2 for rectangular matrices", IPDPS 2019
Competing costs of parallel QR factorization of $A_{m \times n}$

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

$$T_{PGEQRF}^{\alpha, \beta} = O\left(n \log P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta\right)$$

$$M_{PGEQRF} = O\left(\frac{mn}{P}\right)$$

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

$$T_{CAQR}^{\alpha, \beta} = O\left(\sqrt{P} \log^2 P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta\right)$$

$$M_{CAQR} = O\left(\frac{mn}{P}\right)$$

---

\(^1\) 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

\(^5\) E. Hutter et al., "Communication-avoiding CholeskyQR2 for rectangular matrices", IPDPS 2019
Competing costs of parallel QR factorization of $A_{m \times n}$

ScALAPACK's PGEQRF is communication-optimal assuming minimal memory (2D)

$$T^\alpha,\beta_{\text{PGEQRF}} = \mathcal{O}\left(n \log P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta\right)$$
$$M_{\text{PGEQRF}} = \mathcal{O}\left(\frac{mn}{P}\right)$$

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

$$T^\alpha,\beta_{\text{CAQR}} = \mathcal{O}\left(\sqrt{P} \log^2 P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta\right)$$
$$M_{\text{CAQR}} = \mathcal{O}\left(\frac{mn}{P}\right)$$

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

$$T^\alpha,\beta_{\text{CA-CQR2}} = \mathcal{O}\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}} \log P \cdot \alpha + \left(\frac{n^2m}{P}\right)^{\frac{2}{3}} \cdot \beta\right)$$
$$M_{\text{CA-CQR2}} = \mathcal{O}\left(\left(\frac{n^2m}{P}\right)^{\frac{2}{3}}\right)$$

---

\(^1\) 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
\(^5\) E. Hutter et al., "Communication-avoiding CholeskyQR2 for rectangular matrices", IPDPS 2019
Competing costs of parallel QR factorization of $A_{m \times n}$

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

$$T^{\alpha, \beta}_{\text{PGEQRF}} = \mathcal{O} \left( n \log P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta \right)$$

$$M_{\text{PGEQRF}} = \mathcal{O} \left( \frac{mn}{P} \right)$$

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

$$T^{\alpha, \beta}_{\text{CAQR}} = \mathcal{O} \left( \sqrt{P} \log^2 P \cdot \alpha + \frac{mn}{\sqrt{P}} \cdot \beta \right)$$

$$M_{\text{CAQR}} = \mathcal{O} \left( \frac{mn}{P} \right)$$

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

$$T^{\alpha, \beta}_{\text{CA-CQR2}} = \mathcal{O} \left( \left( \frac{Pn}{m} \right)^{\frac{2}{3}} \log P \cdot \alpha + \left( \frac{n^2m}{P} \right)^{\frac{2}{3}} \cdot \beta \right)$$

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

3D algorithms exist in theory\(^2\ 3\ 4\), but \textbf{CA-CQR2 is the first practical approach}\(^5\)

---

\(^1\) 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
\(^5\) E. Hutter et al., "Communication-avoiding CholeskyQR2 for rectangular matrices", IPDPS 2019
Instability of Cholesky-QR

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

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

\[ B \leftarrow A^T A \]

Possible failure in Cholesky factorization!

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

may have lost all accuracy!

\[ Q \] may lost orthogonality!

CholeskyQR2 leverages near-perfect conditioning of \( Q \) in a second iteration

---

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

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

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

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

\[
\begin{align*}
[Q, R] &\leftarrow \textbf{Cholesky-QR} \left( A \right) \\
B &\leftarrow A^T A \\
R^T R &\leftarrow B \\
Q &\leftarrow AR^{-1}
\end{align*}
\]

\[ \triangleright B \text{ may be indefinite!} \]

\[ \triangleright \text{Possible failure in Cholesky factorization!} \]

\[ \triangleright R \text{ may have lost all accuracy! } Q \text{ may lost orthogonality!} \]

\[ \text{1Y. Yamamoto et al., } "\text{Roundoff Error Analysis of the CholeskyQR2 algorithm}" , \text{Electron. Trans. Numer. Anal. 2015} \]
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

\[ A R_1^{-1} R_2^{-1} \ldots R_n^{-1} = Q \]

\[
\begin{align*}
[Q, R] & \leftarrow \textbf{Cholesky-QR} (A) \\
B & \leftarrow A^T A \\
R^T R & \leftarrow B \\
Q & \leftarrow AR^{-1}
\end{align*}
\]

▷ B may be indefinite!
▷ Possible failure in Cholesky factorization!
▷ R may have lost all accuracy! Q may lost orthogonality!

CholeskyQR2 leverages near-perfect conditioning of Q in a second iteration\(^1\)

Cholesky-QR2 (CQR2) can achieve superior performance on tall-and-skinny matrices.\footnote{T. Fukaya et al., "CholeskyQR2: A communication-avoiding algorithm", ScalA 2014}
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

\(^1\)T. Fukaya et al., “CholeskyQR2: A communication-avoiding algorithm”, ScalA 2014
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

\[^1\text{T. Fukaya et al., "CholeskyQR2: A communication-avoiding algorithm", ScalA 2014}\]
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

CQR2 attains minimal communication cost (by \(O(\log P)\)), yet simple implementation

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

\(^1\)T. Fukaya et al., “CholeskyQR2: A communication-avoiding algorithm”, ScalA 2014
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\(^2\)

\[\text{Figure: Computation of Gram matrix}
\]

\[\text{Cost: } O\left(\left(\log c + \log \frac{d}{c}\right) \cdot \alpha + \left(\frac{mn}{dc} + \frac{n^2}{c^2}\right) \cdot \beta + \left(\frac{mn^2}{dc^2} + \frac{n^2}{c^2}\right) \cdot \gamma\right)\]

---


CA-CQR2’s communication-optimal parallelization

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

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 \((d/c)\).


CA-CQR2’s communication-optimal parallelization

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

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 \((d/c)\)

\[\text{Cost: } \mathcal{O}((\log c + \log d/c) \cdot \alpha + \left( \frac{mn}{dc} + \frac{n^2}{c^2} \right) \cdot \beta + \left( \frac{mn^2}{dc^2} + \frac{n^2}{c^2} \right) \cdot \gamma\]


CA-CQR2’s communication-optimal parallelization

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

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:** \(\frac{d}{c}\) simultaneous 3D Cholesky on cubes of dimension \(c\)

\[
\text{Cost: } O \left( c^2 \log c^3 \cdot \alpha + \frac{n^2}{c^2} \cdot \beta + \frac{n^3}{c^3} \cdot \gamma \right)
\]


CA-CQR2’s communication-optimal parallelization

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

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 \((d/c)\).

\[\text{Cost: } \mathcal{O} \left( \log c^3 \cdot \alpha + \frac{n^2}{c^2} \cdot \beta + \frac{n^3}{c^3} \cdot \gamma \right)\]


Algorithmic cost analysis: CA-CQR2 vs. competition

CA-CQR2’s cost expression expresses tunable tradeoffs

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

CA-CQR2’s cost expression expresses tunable tradeoffs

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

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
Algorithmic cost analysis: CA-CQR2 vs. competition

CA-CQR2’s cost expression expresses tunable tradeoffs

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

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

1D Cholesky-QR2

- messages \( \mathcal{O} (\log P) \)
- words \( \mathcal{O} \left( n^2 \right) \)
- flops \( \mathcal{O} \left( \frac{n^2m}{P} + n^3 \right) \)
- memory \( \mathcal{O} \left( \frac{mn}{P} + n^2 \right) \)
Algorithmic cost analysis: CA-CQR2 vs. competition

CA-CQR2’s cost expression expresses tunable tradeoffs

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

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></th>
<th>1D Cholesky-QR2</th>
<th>2D ScALAPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>messages</td>
<td>(O(\log P))</td>
<td>(O(n \log P))</td>
</tr>
<tr>
<td>words</td>
<td>(O(\sqrt{n}))</td>
<td>(O\left(\frac{mn}{\sqrt{P}}\right))</td>
</tr>
<tr>
<td>flops</td>
<td>(O\left(\frac{n^2m}{P} + n^3\right))</td>
<td>(O\left(\frac{mn^2}{P}\right))</td>
</tr>
<tr>
<td>memory</td>
<td>(O\left(\frac{mn}{P} + n^2\right))</td>
<td>(O\left(\frac{mn}{P}\right))</td>
</tr>
</tbody>
</table>
CA-CQR2’s cost expression expresses tunable tradeoffs

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

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

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

CA-CQR2’s cost expression expresses tunable tradeoffs

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

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

<table>
<thead>
<tr>
<th></th>
<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>
<td>( \mathcal{O}\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}} \log P\right) )</td>
</tr>
<tr>
<td>words</td>
<td>( \mathcal{O}(n^2) )</td>
<td>( \mathcal{O}\left(\frac{m n}{\sqrt{P}}\right) )</td>
<td>( \mathcal{O}\left(\frac{m n}{\sqrt{P}}\right) )</td>
<td>( \mathcal{O}\left(\left(\frac{n^2 m}{P}\right)^{\frac{2}{3}}\right) )</td>
</tr>
<tr>
<td>flops</td>
<td>( \mathcal{O}\left(\frac{n^2 m}{P} + n^3\right) )</td>
<td>( \mathcal{O}\left(\frac{m n^2}{P}\right) )</td>
<td>( \mathcal{O}\left(\frac{m n^2}{P}\right) )</td>
<td>( \mathcal{O}\left(\left(\frac{n^2 m}{P}\right)^{\frac{2}{3}}\right) )</td>
</tr>
<tr>
<td>memory</td>
<td>( \mathcal{O}\left(\frac{m n}{P} + n^2\right) )</td>
<td>( \mathcal{O}\left(\frac{m n}{P}\right) )</td>
<td>( \mathcal{O}\left(\frac{m n}{P}\right) )</td>
<td>( \mathcal{O}\left(\left(\frac{n^2 m}{P}\right)^{\frac{2}{3}}\right) )</td>
</tr>
</tbody>
</table>
Algorithmic cost analysis: CA-CQR2 vs. competition

CA-CQR2’s cost expression expresses tunable tradeoffs

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

Requiring each processor to own a square submatrix \( \frac{m}{d} = \frac{n}{c} \) 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>( O(n \log P) )</td>
<td>( O(\sqrt{P} \log^2 P) )</td>
<td>( O\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}} \log P\right))</td>
</tr>
<tr>
<td>words</td>
<td>( O\left(n^2\right) )</td>
<td>( O\left(\frac{mn}{\sqrt{P}}\right) )</td>
<td>( O\left(\left(n^2\frac{m}{P}\right)^{\frac{2}{3}}\right))</td>
</tr>
<tr>
<td>flops</td>
<td>( O\left(\frac{n^2m}{P} + n^3\right) )</td>
<td>( O\left(\frac{mn^2}{P}\right) )</td>
<td>( O\left(\left(n^2\frac{m}{P}\right)^{\frac{2}{3}}\right))</td>
</tr>
<tr>
<td>memory</td>
<td>( O\left(\frac{mn}{P} + n^2\right) )</td>
<td>( O\left(\frac{mn}{P}\right) )</td>
<td>( O\left(\left(n^2\frac{m}{P}\right)^{\frac{2}{3}}\right))</td>
</tr>
</tbody>
</table>

Minimal communication cost in a QR factorization is reflected by the surface area of the cubic volume of \( O(mn^2/P) \) computation
Implementation and Experiment setup

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.

---

1 Intel Knights Landing (KNL) cluster at TACC
2 Cray XE/XK hybrid machine at NCSA
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.
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 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\(^2\)

---

\(^1\)Intel Knights Landing (KNL) cluster at TACC
\(^2\)Cray XE/XK hybrid machine at NCSA
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
Deeper analysis into Strong Scaling results

Table: Strong scaling: CA-CQR2 performance relative to ScaLAPACK

<table>
<thead>
<tr>
<th></th>
<th>m/n</th>
<th>computation</th>
<th>512 PEs</th>
<th>1024 PEs</th>
<th>2048 PEs</th>
<th>4096 PEs</th>
<th>8192 PEs</th>
<th>16384 PEs</th>
<th>32768 PEs</th>
<th>65536 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueWaters</td>
<td>4096</td>
<td>2.00x</td>
<td>1.01x</td>
<td>0.88x</td>
<td>0.70x</td>
<td>0.62x</td>
<td>0.62x</td>
<td>0.73x</td>
<td>1.00x</td>
<td>-</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>512</td>
<td>2.00x</td>
<td>0.51x</td>
<td>0.48x</td>
<td>0.51x</td>
<td>0.56x</td>
<td>0.66x</td>
<td>0.86x</td>
<td>1.36x</td>
<td>-</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>64</td>
<td>2.02x</td>
<td>0.51x</td>
<td>0.53x</td>
<td>0.53x</td>
<td>0.61x</td>
<td>0.73x</td>
<td>0.91x</td>
<td>0.92</td>
<td>-</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>8</td>
<td>2.20x</td>
<td>0.53x</td>
<td>0.54x</td>
<td>0.55x</td>
<td>0.72x</td>
<td>0.75x</td>
<td>0.67x</td>
<td>0.47x</td>
<td>-</td>
</tr>
<tr>
<td>Blue Waters</td>
<td>1</td>
<td>4.25x</td>
<td>0.26x</td>
<td>0.21x</td>
<td>0.18x</td>
<td>0.27x</td>
<td>0.21x</td>
<td>0.13x</td>
<td>0.13x</td>
<td>-</td>
</tr>
<tr>
<td>Stampede2</td>
<td>4096</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.70x</td>
<td>1.02x</td>
<td>1.27x</td>
<td>1.72x</td>
<td>3.13x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>512</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.52x</td>
<td>0.99x</td>
<td>1.47x</td>
<td>2.01x</td>
<td>3.34x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>64</td>
<td>2.02x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.19x</td>
<td>1.59x</td>
<td>1.82x</td>
<td>2.61x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>8</td>
<td>2.20x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.00x</td>
<td>1.21x</td>
<td>1.36x</td>
<td>1.60x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>1</td>
<td>4.25x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.48x</td>
<td>0.55x</td>
<td>0.66x</td>
<td>1.41x</td>
<td>1.02x</td>
</tr>
</tbody>
</table>
## Deeper analysis into Strong Scaling results

<table>
<thead>
<tr>
<th></th>
<th>m/n</th>
<th>computation</th>
<th>512 PEs</th>
<th>1024 PEs</th>
<th>2048 PEs</th>
<th>4096 PEs</th>
<th>8192 PEs</th>
<th>16384 PEs</th>
<th>32768 PEs</th>
<th>65536 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlueWaters</strong></td>
<td>4096</td>
<td>2.00x</td>
<td>1.01x</td>
<td>0.88x</td>
<td>0.70x</td>
<td>0.62x</td>
<td>0.62x</td>
<td>0.73x</td>
<td>1.00x</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>2.00x</td>
<td>0.51x</td>
<td>0.48x</td>
<td>0.51x</td>
<td>0.56x</td>
<td>0.66</td>
<td>0.86x</td>
<td>1.36x</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.02x</td>
<td>0.51x</td>
<td>0.53x</td>
<td>0.53x</td>
<td>0.61x</td>
<td>0.73x</td>
<td>0.91x</td>
<td>0.92</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.20x</td>
<td>0.53x</td>
<td>0.54x</td>
<td>0.55x</td>
<td>0.72x</td>
<td>0.75x</td>
<td>0.67x</td>
<td>0.47x</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4.25x</td>
<td>0.26x</td>
<td>0.21x</td>
<td>0.18x</td>
<td>0.27x</td>
<td>0.21x</td>
<td>0.13x</td>
<td>0.13x</td>
<td>-</td>
</tr>
<tr>
<td><strong>Stampede2</strong></td>
<td>4096</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.70x</td>
<td>1.02x</td>
<td>1.27x</td>
<td>1.72x</td>
<td>3.13x</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.52x</td>
<td>0.99x</td>
<td>1.47x</td>
<td>2.01x</td>
<td>3.34x</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>2.02x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.19x</td>
<td>1.59x</td>
<td>1.82x</td>
<td>2.61x</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.20x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.00x</td>
<td>1.21x</td>
<td>1.36x</td>
<td>1.60x</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4.25x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.48x</td>
<td>0.55x</td>
<td>0.66x</td>
<td>1.41x</td>
<td>1.02x</td>
</tr>
</tbody>
</table>

**Table:** Strong scaling: CA-CQR2 performance relative to ScaLAPACK
**Deeper analysis into Strong Scaling results**

Table: Strong scaling: CA-CQR2 performance relative to ScaLAPACK

<table>
<thead>
<tr>
<th></th>
<th>m/n</th>
<th>computation</th>
<th>512 PEs</th>
<th>1024 PEs</th>
<th>2048 PEs</th>
<th>4096 PEs</th>
<th>8192 PEs</th>
<th>16384 PEs</th>
<th>32768 PEs</th>
<th>65536 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueWaters</td>
<td>4096</td>
<td>2.00x</td>
<td>1.01x</td>
<td>0.88x</td>
<td>0.70x</td>
<td>0.62x</td>
<td>0.62x</td>
<td>0.73x</td>
<td>1.00x</td>
<td>-</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>512</td>
<td>2.00x</td>
<td>0.51x</td>
<td>0.48x</td>
<td>0.51x</td>
<td>0.56x</td>
<td>0.66</td>
<td>0.86x</td>
<td>1.36x</td>
<td>-</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>64</td>
<td>2.02x</td>
<td>0.51x</td>
<td>0.53x</td>
<td>0.61x</td>
<td>0.73x</td>
<td>0.91x</td>
<td>0.92</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>BlueWaters</td>
<td>8</td>
<td>2.20x</td>
<td>0.53x</td>
<td>0.54x</td>
<td>0.55x</td>
<td>0.72x</td>
<td>0.75x</td>
<td>0.67x</td>
<td>0.47x</td>
<td>-</td>
</tr>
<tr>
<td>Blue Waters</td>
<td>1</td>
<td>4.25x</td>
<td>0.26x</td>
<td>0.21x</td>
<td>0.18x</td>
<td>0.27x</td>
<td>0.21x</td>
<td>0.13x</td>
<td>0.13x</td>
<td>-</td>
</tr>
<tr>
<td>Stampede2</td>
<td>4096</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.70x</td>
<td>1.02x</td>
<td>1.27x</td>
<td>1.72x</td>
<td>3.13x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>512</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.52x</td>
<td>0.99x</td>
<td>1.47x</td>
<td>2.01x</td>
<td>3.34x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>64</td>
<td>2.02x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.19x</td>
<td>1.59x</td>
<td>1.82x</td>
<td>2.61x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>8</td>
<td>2.20x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.00x</td>
<td>1.21x</td>
<td>1.36x</td>
<td>1.60x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>1</td>
<td>4.25x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.48x</td>
<td>0.55x</td>
<td>0.66x</td>
<td>1.41x</td>
<td>1.02x</td>
</tr>
</tbody>
</table>
Deeper analysis into Strong Scaling results

### Table: Strong scaling: CA-CQR2 performance relative to ScaLAPACK

<table>
<thead>
<tr>
<th>m/n</th>
<th>computation</th>
<th>512 PEs</th>
<th>1024 PEs</th>
<th>2048 PEs</th>
<th>4096 PEs</th>
<th>8192 PEs</th>
<th>16384 PEs</th>
<th>32768 PEs</th>
<th>65536 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueWaters</td>
<td>4096</td>
<td>2.00x</td>
<td>1.01x</td>
<td>0.88x</td>
<td>0.70x</td>
<td>0.62x</td>
<td>0.62x</td>
<td>0.73x</td>
<td>1.00x</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>512</td>
<td>2.00x</td>
<td>0.51x</td>
<td>0.48x</td>
<td>0.51x</td>
<td>0.56x</td>
<td>0.66x</td>
<td>0.86x</td>
<td>1.36x</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>64</td>
<td>2.02x</td>
<td>0.51x</td>
<td>0.53x</td>
<td>0.53x</td>
<td>0.61x</td>
<td>0.73x</td>
<td>0.91x</td>
<td>0.92</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>8</td>
<td>2.20x</td>
<td>0.53x</td>
<td>0.54x</td>
<td>0.55x</td>
<td>0.72x</td>
<td>0.75x</td>
<td>0.67x</td>
<td>0.47x</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>1</td>
<td>4.25x</td>
<td>0.26x</td>
<td>0.21x</td>
<td>0.18x</td>
<td>0.27x</td>
<td>0.21x</td>
<td>0.13x</td>
<td>0.13x</td>
</tr>
</tbody>
</table>

| Stampede2 | 4096 | 2.00x | - | - | - | 0.70x | 1.02x | 1.27x | 1.72x | 3.13x |
| Stampede2 | 512 | 2.00x | - | - | - | 0.52x | 0.99x | 1.47x | 2.01x | 3.34x |
| Stampede2 | 64 | 2.02x | - | - | - | 0.77x | 1.19x | 1.59x | 1.82x | 2.61x |
| Stampede2 | 8 | 2.20x | - | - | - | 0.77x | 1.00x | 1.21x | 1.36x | 1.60x |
| Stampede2 | 1 | 4.25x | - | - | - | 0.48x | 0.55x | 0.66x | 1.41x | 1.02x |
Deeper analysis into Strong Scaling results

**Table: Strong scaling: CA-CQR2 performance relative to ScaLAPACK**

<table>
<thead>
<tr>
<th>m/n</th>
<th>computation</th>
<th>512 PEs</th>
<th>1024 PEs</th>
<th>2048 PEs</th>
<th>4096 PEs</th>
<th>8192 PEs</th>
<th>16384 PEs</th>
<th>32768 PEs</th>
<th>65536 PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueWaters</td>
<td>4096</td>
<td>2.00x</td>
<td>1.01x</td>
<td>0.88x</td>
<td>0.70x</td>
<td>0.62x</td>
<td>0.62x</td>
<td>0.73x</td>
<td>1.00x</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>512</td>
<td>2.00x</td>
<td>0.51x</td>
<td>0.48x</td>
<td>0.51x</td>
<td>0.56x</td>
<td>0.66</td>
<td>0.86x</td>
<td>1.36x</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>64</td>
<td>2.02x</td>
<td>0.51x</td>
<td>0.53x</td>
<td>0.53x</td>
<td>0.61x</td>
<td>0.73x</td>
<td>0.91x</td>
<td>0.92</td>
</tr>
<tr>
<td>BlueWaters</td>
<td>8</td>
<td>2.20x</td>
<td>0.53x</td>
<td>0.54x</td>
<td>0.55x</td>
<td>0.72x</td>
<td>0.75x</td>
<td>0.67x</td>
<td>0.47x</td>
</tr>
<tr>
<td>Blue Waters</td>
<td>1</td>
<td>4.25x</td>
<td>0.26x</td>
<td>0.21x</td>
<td>0.18x</td>
<td>0.27x</td>
<td>0.21x</td>
<td>0.13x</td>
<td>0.13x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>4096</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.70x</td>
<td>1.02x</td>
<td>1.27x</td>
<td>1.72x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>512</td>
<td>2.00x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.52x</td>
<td>0.99x</td>
<td>1.47x</td>
<td>2.01x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>64</td>
<td>2.02x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.19x</td>
<td>1.59x</td>
<td>1.82x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>8</td>
<td>2.20x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.77x</td>
<td>1.00x</td>
<td>1.21x</td>
<td>1.36x</td>
</tr>
<tr>
<td>Stampede2</td>
<td>1</td>
<td>4.25x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.48x</td>
<td>0.55x</td>
<td>0.66x</td>
<td>1.41x</td>
</tr>
</tbody>
</table>
QR Strong scaling critical path analysis

524288 x 2048 matrix: Stampede2 (S2) vs. BlueWaters (BW)

- **Computation**
- **Communication**
- **Overlap**

Graph showing the time taken for different PEs on Stampede2 (S2) and BlueWaters (BW).
QR Strong scaling critical path analysis

131072 x 4096 matrix: Stampede2 (S2) vs. BlueWaters (BW)

Edward Hutter and Edgar Solomonik
QR Strong scaling critical path analysis

32768 x 8192 matrix: Stampede2 (S2) vs. BlueWaters (BW)

Time (s)

Overlap

Communication

Computation

Edward Hutter and Edgar Solomonik
CA-CQR2’s performance improvements over ScaLAPACK on Stampede2 range from 1.1 - 3.3x at 1024 nodes

Our preprint detailing CA-CQR2 can be found at https://arxiv.org/abs/1710.08471
Our C++ implementation can be found at https://github.com/huttered40/CA-CQR2
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

---

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
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

---

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
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

Offloading computation to GPUs on XK nodes is a work in progress

---

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
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 reduction 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**

Offloading computation to GPUs on XK nodes is a work in progress

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
Cyclops Tensor Framework (CTF)

https://github.com/cyclops-community/ctf

- MPI sparse/dense tensors + OpenMP and CUDA acceleration

```cpp
Matrix<int> A(n, n, AS|SP, World(MPI_COMM_WORLD));
Tensor<float> T(order, is_sparse, dims, syms, ring, world);
T.read(...); T.write(...); T.slice(...); T.permute(...);
```

Parallel contraction/summation/transformation of tensors:

```cpp```
Z"abij" += V"ijab"; // C++
W"mnij" += 0.5 * W"mnef"*T"efij"; // C++
M"ij" += Function<>( [](double x){ return 1/x; })( v"j");
```cpp```

```python```
W.i("mnij") << 0.5 * W.i("mnef")*T.i("efij") // Python
```python```

```python```
```python```

```python```
einsum("mnef,efij->mnij",W,T) // numpy-style Python
```python```

Cyclops applications (some using Blue Waters): tensor decomposition, tensor completion, tensor networks (DMRG), quantum chemistry, quantum circuit simulation, graph algorithms, bioinformatics

Edward Hutter and Edgar Solomonik

27/28
Cyclops Tensor Framework (CTF)

https://github.com/cyclops-community/ctf

- MPI sparse/dense tensors + OpenMP and CUDA acceleration

```cpp
Matrix<int> A(n, n, AS|SP, World(MPI_COMM_WORLD));
Tensor<float> T(order, is_sparse, dims, syms, ring, world);
T.read(...); T.write(...); T.slice(...); T.permute(...);
```

- parallel contraction/summation/transformation of tensors

```cpp
Z"abij" += V"ijab";                                  // C++
W"mni" += 0.5*W"mnen"*T"efij";                      // C++
M"ij" += Function<>([](double x){ return 1/x; })(v"j");
W.i"mni" << 0.5*W.i"mnen"*T.i"efij";                 // Python
einsum("mnen,efij->mnij",W,T) // numpy-style Python
```
Cyclops Tensor Framework (CTF)

https://github.com/cyclops-community/ctf

- MPI sparse/dense tensors + OpenMP and CUDA acceleration

```cpp
Matrix<int> A(n, n, AS|SP, World(MPI_COMM_WORLD));
Tensor<float> T(order, is_sparse, dims, syms, ring, world);
T.read(...); T.write(...); T.slice(...); T.permute(...);
```

- parallel contraction/summation/transformation of tensors

```cpp
Z["abij"] += V["ijab"]; // C++
W["mni"] += 0.5 * W["mnef"] * T["efij"]; // C++
M["ij"] += Function<>{(double x){ return 1/x; }}(v["j"]) ;
W.i("mni") <<= 0.5 * W.i("mnef") * T.i("efij") // Python
einsum("mnef,efij->mni",W,T) // numpy-style Python
```

- Cyclops applications (some using Blue Waters): tensor decomposition, tensor completion, tensor networks (DMRG), quantum chemistry, quantum circuit simulation, graph algorithms, bioinformatics
We’d also like to acknowledge NCSA and TACC for providing benchmarking resources
- Texas Advanced Computing Center (TACC) via Stampede2
- National Center for Supercomputing Applications (NCSA) via Blue Waters

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 Grant number DE-SC0019323
2 Allocation TG-CCR180006
3 Awards OCI-0725070 and ACI-1238993
The Cholesky-QR2 algorithm can achieve stability through iterative refinement\(^1\)


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

The Cholesky-QR2 algorithm can achieve stability through iterative refinement\(^1\)

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

\[
Z, R_1 \leftarrow CQR(A)
Q, R_2 \leftarrow CQR(Z)
R \leftarrow R_2 R_1
\]

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

The Cholesky-QR2 algorithm can achieve stability through iterative refinement\(^1\)

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

\[
Z, R_1 \leftarrow \text{CQR}(A)
Q, R_2 \leftarrow \text{CQR}(Z)
R \leftarrow R_2 R_1
\]

- leverages near-perfect conditioning of \(Z\) in a second iteration\(^1\)

---


\(^2\) T. Fukaya et al., "Shifted CholeskyQR for computing the QR factorization of ill-conditioned matrices", Arxiv 2018
The Cholesky-QR2 algorithm \textit{can} achieve stability through iterative refinement\textsuperscript{1}

\[
[Q, R] \leftarrow \textbf{Cholesky-QR2} \left( A \right)
\]

\[
Z, R_1 \leftarrow \text{CQR} \left( A \right)
Q, R_2 \leftarrow \text{CQR} \left( Z \right)
R \leftarrow R_2 R_1
\]

- leverages near-perfect conditioning of \( Z \) in a second iteration\textsuperscript{1}
- \( A = ZR_1 = QR_2 R_1 \), from \( A^T A = R_1^T Z^T Z R_1 = R_1^T R_2^T Q^T QR_2 R_1 \), where \( R_2 \) corrects initial \( R_1 \)

\textsuperscript{2}T. Fukaya et al., "Shifted CholeskyQR for computing the QR factorization of ill-conditioned matrices", Arxiv 2018
The Cholesky-QR2 algorithm can achieve stability through iterative refinement.\(^1\)

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

\[
Z, R_1 \leftarrow \text{CQR}(A)
Q, R_2 \leftarrow \text{CQR}(Z)
R \leftarrow R_2R_1
\]

- leverages near-perfect conditioning of \(Z\) in a second iteration.\(^1\)
- \(A = ZR_1 = QR_2R_1, \) from \(A^T A = R_1^T Z^TZR_1 = R_1^T R_2^T Q^T QR_2 R_1,\) where \(R_2\) corrects initial \(R_1\)
- numerical breakdown still possible if first iteration loses positive definiteness in \(A^T A\) via \(\kappa(A) \leq 1/\sqrt{\epsilon}\)


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

The Cholesky-QR2 algorithm can achieve stability through iterative refinement

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

\[
Z, R_1 \leftarrow \text{CQR}(A)
Q, R_2 \leftarrow \text{CQR}(Z)
R \leftarrow R_2 R_1
\]

- leverages near-perfect conditioning of \(Z\) in a second iteration
- \(A = ZR_1 = QR_2 R_1\), 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 \(\kappa(A) \leq 1/\sqrt{\epsilon}\)

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

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

\(^2\)T. Fukaya et al., "Shifted CholeskyQR for computing the QR factorization of ill-conditioned matrices", Arxiv 2018
CA-CQR2 building block #1 – 3D Matrix Multiplication

Figure: 3D algorithm for square matrix multiplication

$$C = AB$$

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

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\(^1\)

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

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

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

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

\(^1\)A. Tiskin 2007, "Communication-efficient generic pairwise elimination"
Figure: Start with a tunable $c \times d \times c$ processor grid
Cost: $2 \log_2 c \cdot \alpha + \frac{2mn}{dc} \cdot \beta$
Cost: $2 \log_2 c \cdot \alpha + \frac{2n^2}{c^2} \cdot \beta + \frac{n^2}{c^2} \cdot \gamma$
CA-CQR2 – Computation of Gram matrix

**Figure:** Allreduce alternating groups of size $\frac{d}{c}$

Cost: $2 \log_2 \frac{d}{c} \cdot \alpha + \frac{2n^2}{c^2} \cdot \beta + \frac{n^2}{c^2} \cdot \gamma$
Cost: $2 \log_2 c \cdot \alpha + \frac{2n^2}{c^2} \cdot \beta$
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: \(d\) simultaneous 3D matrix multiplication or TRSM on cubes of dimension \(c\)

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

Cost: \(O(\log_2 c^3 \cdot \alpha + \left( \frac{mn}{dc} + \frac{n^2 + nc}{c^2} \right) \cdot \beta + \frac{n^2 m}{c^2 d} \cdot \gamma)\)
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

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

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.

$$T^{\alpha - \beta}_{\text{Cholesky-QR2_Tunable}}\left(\frac{m}{n}, \left(\frac{Pn}{m}\right)^{\frac{1}{3}}, \left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}\right) = \mathcal{O}\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}} \log P \cdot \alpha\right) + \frac{\left(\frac{Pn}{m}\right)^{\frac{1}{3}}}{\left(\frac{Pn}{m}\right)^{\frac{2}{3}}} \cdot \beta + \frac{n^3 \left(\frac{Pm^2}{n^2}\right)^{\frac{1}{3}}}{\left(\frac{Pn}{m}\right)^{\frac{2}{3}}} + n^2 m \left(\frac{Pn}{m}\right)^{\frac{1}{3}} \cdot \gamma\right)$$

(1)

<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>$\mathcal{O}\left(\left(\frac{Pn}{m}\right)^{\frac{2}{3}} \log P\right)$</td>
</tr>
<tr>
<td></td>
<td># of words</td>
<td>$\mathcal{O}\left(\left(\frac{n^2 m}{P}\right)^{\frac{2}{3}}\right)$</td>
</tr>
<tr>
<td></td>
<td># of flops</td>
<td>$\mathcal{O}\left(\frac{n^2 m}{P}\right)$</td>
</tr>
<tr>
<td></td>
<td>Memory footprint</td>
<td>$\mathcal{O}\left(\left(\frac{n^2 m}{P}\right)^{\frac{2}{3}}\right)$</td>
</tr>
</tbody>
</table>