I/O-Optimal Algorithms for Symmetric Linear Algebra Kernels

Olivier Beaumont
Lionel Eyraud-Dubois
Julien Langou
Mathieu Vérité

Solharis & HPC Scalable Ecosystem – Feb. 2022

Introduction

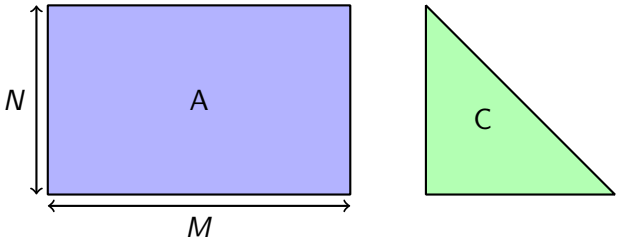
Goal: Minimize communications for Cholesky & SYRK

- **Cholesky factorization:** From A SPD, compute L triangular such that $A = L \cdot L^T$
- **SYRK:** From A N-by-M, compute C SPD such that $C = A \cdot A^T$
- Consider **sequential**, **out-of-core** algorithms
  - Unbounded slow memory, fast memory of size S
  - Find an **ordering** of computations to minimize the data loaded in fast memory
  - Simpler than **parallel** model, but results can be transferred (esp. lower bounds)

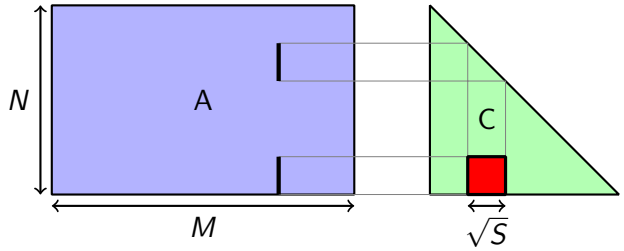
Operational Intensity

- **OI:** ratio $\rho = \frac{\text{number of operations}}{\text{volume of communications}}$
- OI for GEMM and LU factorization: $\rho = \sqrt{S}$
- OI for SYRK and Cholesky: $\sqrt{S} \leqslant \rho \leqslant 2\sqrt{S}$ [Béreux 2014] [Olivry et al 2020]
- **Our results:** $\rho = \sqrt{2}\sqrt{S}$ (matching lower bounds and algorithms for both kernels)

It actually starts with SYRK $C = A \cdot A^T$



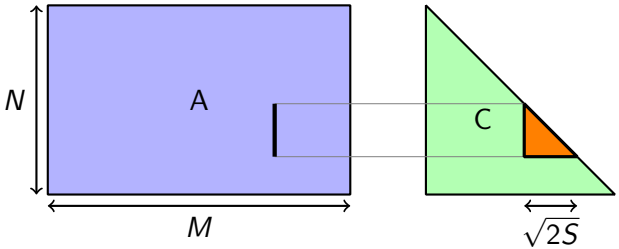
It actually starts with SYRK $C = A \cdot A^T$



$2\sqrt{S}$ data, $2S$ operations:
$\rho = \sqrt{S}$

- Béreux's **OOC_SYRK**: one-tile (square), narrow-block algorithm
  - Keep a large tile of C in memory, load vectors v of A one after the other, $C \mathrel{+}= v \cdot v^T$
  - Actually very close to a GEMM algorithm $C = A \cdot B^T$

It actually starts with SYRK $C = A \cdot A^T$



$2\sqrt{S}$ data, $2S$ operations:
$\rho = \sqrt{S}$

$\sqrt{2S}$ data, $2S$ operations:
$\rho = \sqrt{2S}$

- Béreux's **OOC_SYRK**: one-tile (square), narrow-block algorithm
  - Keep a large tile of C in memory, load vectors v of A one after the other, $C \mathrel{+}= v \cdot v^T$
  - Actually very close to a GEMM algorithm $C = A \cdot B^T$
- But: parts of the computation have higher operational intensity

Proving upper bounds on operational intensity

General methodology by [Kwasniewski et al 2021]

- For X data accessed, largest subset of tasks is: $H_{\max}(X)$
- Provides upper bound on $\rho \leqslant \frac{|H_{\max}(X)|}{X-S}$, valid for any X

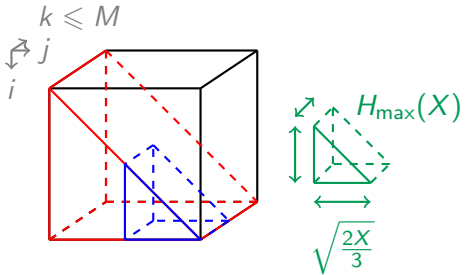
Proving upper bounds on operational intensity

General methodology by [Kwasniewski et al 2021]

- For X data accessed, largest subset of tasks is: $H_{\max}(X)$
- Provides upper bound on $\rho \leqslant \frac{|H_{\max}(X)|}{X-S}$, valid for any X

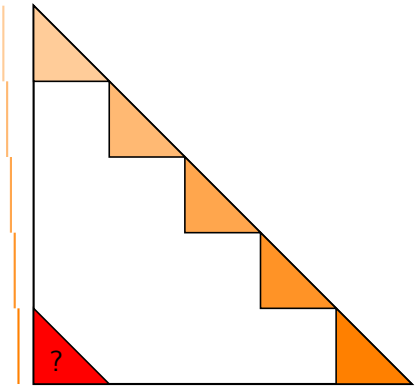
Our result

- For X data accessed, find $H_{\max}(X)$ "by hand"
- Exchange arguments: optimal shape is a prism
- We get $|H_{\max}(X)| \leqslant \frac{1}{2}\left(\frac{2X}{3}\right)^{3/2} \times 2$
- $H_{\max}$ requires $(\frac{1}{2}+1)(\sqrt{\frac{2X}{3}})^2 = X$ data access
- For $X = 3S$, we obtain $\rho \leqslant \frac{|H_{\max}|}{X-S} \leqslant \frac{(2S)^{3/2}}{2S} = \sqrt{2S}$



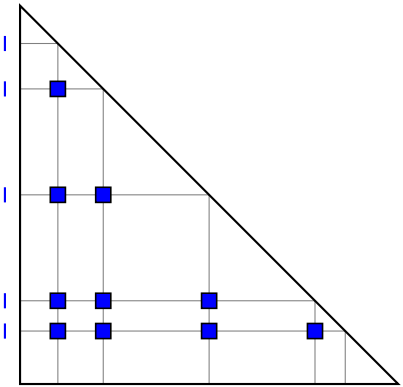
Triangle Blocks

We know $\rho \leqslant \sqrt{2S}$, and it can be achieved on (triangle) diagonal tiles. But what about elements away from the diagonal?



Triangle Blocks

We know $\rho \leqslant \sqrt{2S}$, and it can be achieved on (triangle) diagonal tiles. But what about elements away from the diagonal?



Def: Triangle Block TB(R)

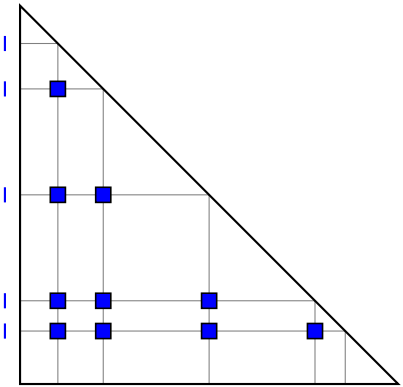
Given a set R of row indices, TB(R) contains all elements that can be updated with R:

$$\mathrm{TB}(R) = \left\{(r, r') \,|\, r, r' \in R \text{ and } r > r'\right\}$$

Question: Can we partition C in triangle blocks of size S?

Triangle Blocks

We know $\rho \leqslant \sqrt{2S}$, and it can be achieved on (triangle) diagonal tiles. But what about elements away from the diagonal?



Def: Triangle Block TB(R)

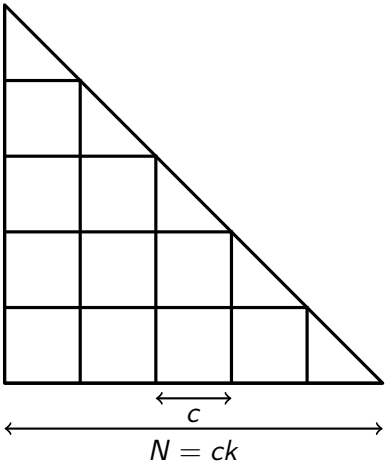
Given a set R of row indices, TB(R) contains all elements that can be updated with R:

$$\mathrm{TB}(R) = \left\{(r, r') \,|\, r, r' \in R \text{ and } r > r'\right\}$$

Question: Can we partition C in triangle blocks of size S?

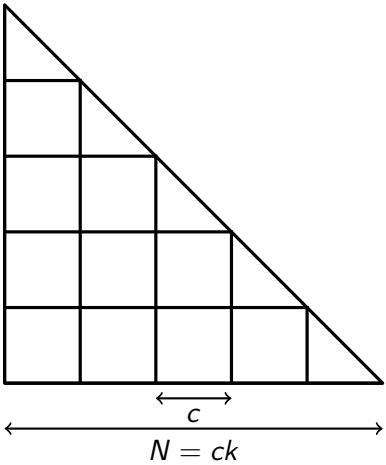
Spoiler: Yes we can!

TBS Algorithm: **T**riangle **B**lock **S**yrk



Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm: **T**riangle **B**lock **S**yrk

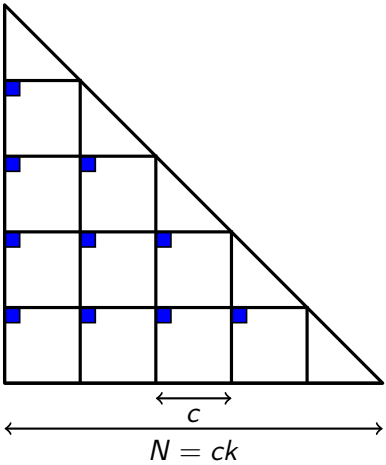


Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

TBS Algorithm: **T**riangle **B**lock **S**yrk

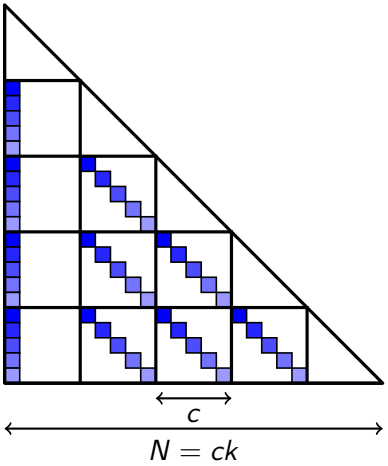


Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

TBS Algorithm: **T**riangle **B**lock **S**yrk

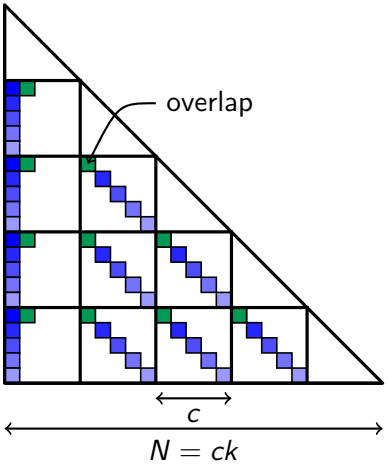


Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

TBS Algorithm: **T**riangle **B**lock **S**yrk



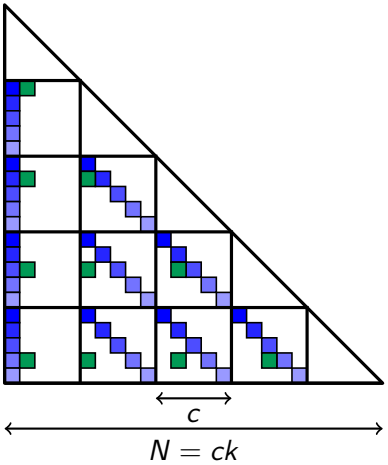
Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

Two triangle blocks **which share the same row twice** will **overlap**

TBS Algorithm: **T**riangle **B**lock **S**yrk



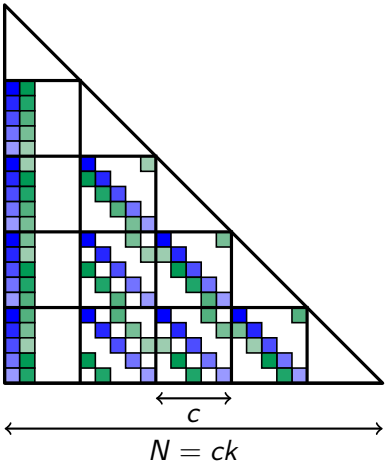
Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

Two triangle blocks **which share the same row twice** will **overlap**

TBS Algorithm: **T**riangle **B**lock **S**yrk



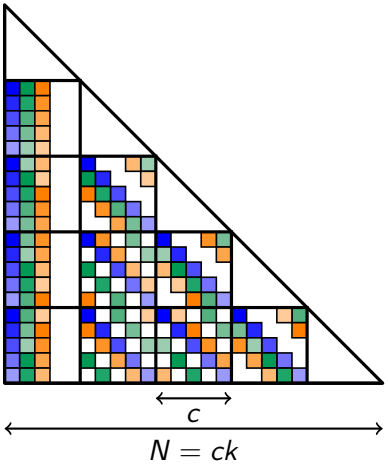
Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

Two triangle blocks **which share the same row twice** will **overlap**

TBS Algorithm: **T**riangle **B**lock **S**yrk



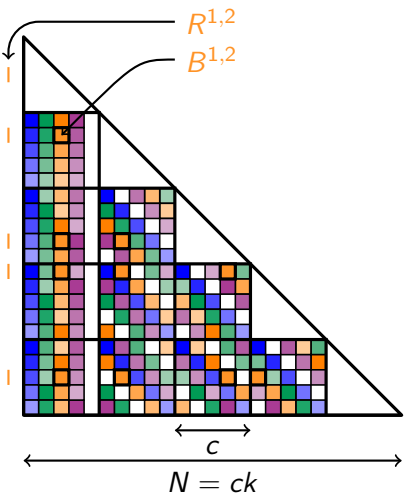
Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

Two triangle blocks **which share the same row twice** will **overlap**

TBS Algorithm: **T**riangle **B**lock **S**yrk



Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

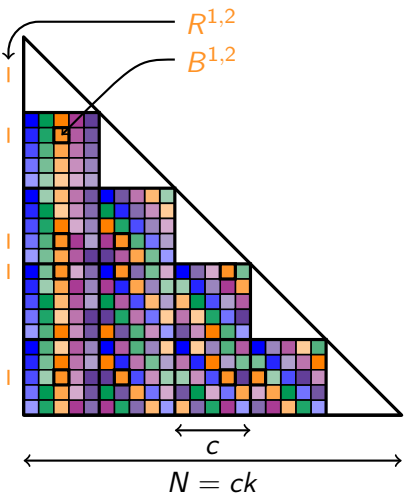
Two triangle blocks **which share the same row twice** will **overlap**

Cyclic definition of blocks

$$B^{i,j} = \mathrm{TB}(R^{i,j})$$

$$\text{with } R^{i,j} = \{uc + (i + j(u-1) \mod c) \mid u \leqslant k\}$$

TBS Algorithm: **T**riangle **B**lock **S**yrk



Assume $S = \frac{k(k+1)}{2}$. We want blocks of side k
Assume $N = ck$. We divide C in zones of size $c \times c$

TBS Algorithm

- c^2 blocks
- each block has one element in each zone

Two triangle blocks **which share the same row twice** will **overlap**

Cyclic definition of blocks

$$B^{i,j} = \mathrm{TB}(R^{i,j})$$

$$\text{with } R^{i,j} = \{uc + (i + j(u-1) \mod c) \mid u \leqslant k\}$$

TBS Algorithm: **T**riangle **B**lock **S**yrk

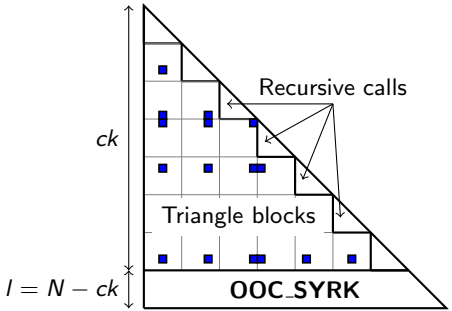
Lemma: cyclic blocks are valid

If $c \geqslant k-1$ is coprime with all integers in $[|2, k-2|]$, then the triangle blocks $B^{i,j}$ are disjoint.

TBS Algorithm: **T**riangle **B**lock **S**yrk

Lemma: cyclic blocks are valid

If $c \geqslant k-1$ is coprime with all integers in $[|2, k-2|]$, then the triangle blocks $B^{i,j}$ are disjoint.

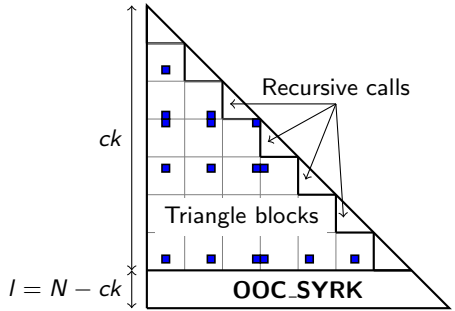


$c \leftarrow$ largest valid value $\leqslant \frac{N}{k}$
if $c < k-1$ **then** *c is too small*
  OOC_SYRK (A, C);
else
  Compute each triangle block iteratively
  Use **OOC_SYRK** for the last $l = N - ck$ rows
  Use **TBS** recursively on the diagonal

TBS Algorithm: **T**riangle **B**lock **S**yrk

Lemma: cyclic blocks are valid

If $c \geqslant k-1$ is coprime with all integers in $[|2, k-2|]$, then the triangle blocks $B^{i,j}$ are disjoint.



$c \leftarrow$ largest valid value $\leqslant \frac{N}{k}$
if $c < k-1$ **then** *c is too small*
  OOC_SYRK (A, C);
else
  Compute each triangle block iteratively
  Use **OOC_SYRK** for the last $I = N - ck$ rows
  Use **TBS** recursively on the diagonal

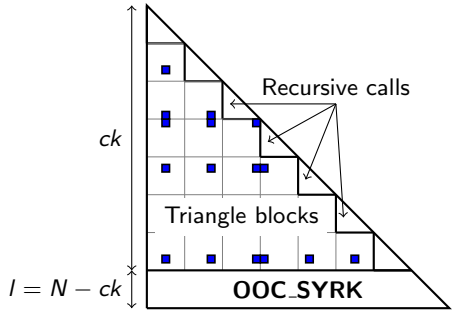
Lemma: this constraint is OK

With this definition of c, $I \doteq N - ck = \mathcal{O}(1)$.

TBS Algorithm: Triangle Block Syrk

Lemma: cyclic blocks are valid

If $c \geqslant k-1$ is coprime with all integers in $[|2, k-2|]$, then the triangle blocks $B^{i,j}$ are disjoint.



$c \leftarrow$ largest valid value $\leqslant \frac{N}{k}$
if $c < k-1$ **then** *c is too small*
  OOC_SYRK (A, C);
else
  Compute each triangle block iteratively
  Use **OOC_SYRK** for the last $l = N - ck$ rows
  Use **TBS** recursively on the diagonal

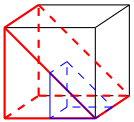
Lemma: this constraint is OK

With this definition of c, $l \doteq N - ck = \mathcal{O}(1)$.

Mission accomplished

$\sqrt{2S}$ OI except on $\mathcal{O}(N \log N)$ elements

Let us now look at Cholesky

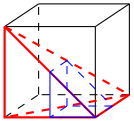


Upper bound on OI for Cholesky

Set of updates in Cholesky $\subseteq$ operations in SYRK

$\Longrightarrow$ Operational Intensity $\leqslant \sqrt{2S}$

Let us now look at Cholesky

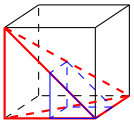


Upper bound on OI for Cholesky

Set of updates in Cholesky $\subseteq$ operations in SYRK

$\Longrightarrow$ Operational Intensity $\leqslant \sqrt{2S}$

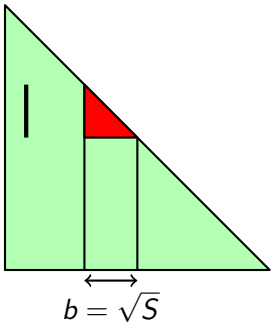
Let us now look at Cholesky



Upper bound on OI for Cholesky

Set of updates in Cholesky $\subseteq$ operations in SYRK

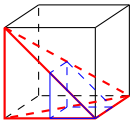
$\Longrightarrow$ Operational Intensity $\leqslant \sqrt{2S}$



Béreux's **OOC_CHOL**

- **left-looking**, **narrow-block**
  - Update a tile once **all** its contributions are computed
  - Allows to load each tile **once**
- Dominant: updates, $\sqrt{S}$ OI

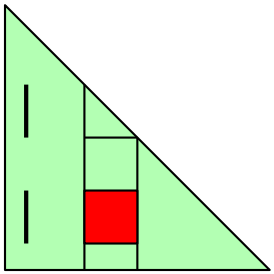
Let us now look at Cholesky



Upper bound on OI for Cholesky

Set of updates in Cholesky $\subseteq$ operations in SYRK

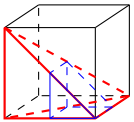
$\Longrightarrow$ Operational Intensity $\leqslant \sqrt{2S}$



Béreux's **OOC_CHOL**

- **left-looking**, **narrow-block**
  - Update a tile once **all** its contributions are computed
  - Allows to load each tile **once**
- Dominant: updates, $\sqrt{S}$ OI

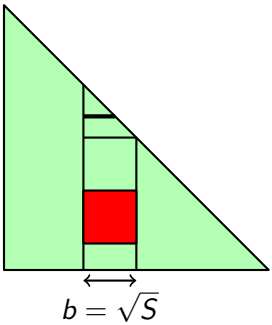
Let us now look at Cholesky



Upper bound on OI for Cholesky

Set of updates in Cholesky $\subseteq$ operations in SYRK

$\Longrightarrow$ Operational Intensity $\leqslant \sqrt{2S}$



Béreux's **OOC_CHOL**

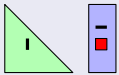
- **left-looking**, **narrow-block**
  - Update a tile once **all** its contributions are computed
  - Allows to load each tile **once**
- Dominant: updates, $\sqrt{S}$ OI

LBC Algorithm: **L**arge **B**lock **C**holesky

OOC_CHOL

$$Q = \frac{N^3}{3\sqrt{S}} + \mathcal{O}(NM)$$

OOC_TRSM



$$Q = \frac{N^2 M}{\sqrt{S}} + \mathcal{O}(NM)$$

TBS

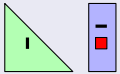
$$Q = \frac{N^2 M}{\sqrt{2S}} + \frac{N^2}{2} + \mathcal{O}(NM \log N)$$

LBC Algorithm: **L**arge **B**lock **C**holesky

OOC_CHOL

$$Q = \frac{N^3}{3\sqrt{S}} + \mathcal{O}(NM)$$

OOC_TRSM

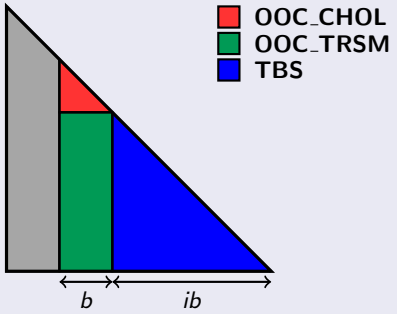


$$Q = \frac{N^2 M}{\sqrt{S}} + \mathcal{O}(NM)$$

TBS

$$Q = \frac{N^2 M}{\sqrt{2S}} + \frac{N^2}{2} + \mathcal{O}(NM \log N)$$

LBC

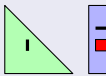


LBC Algorithm: **L**arge **B**lock **C**holesky

OOC_CHOL

$$Q = \frac{N^3}{3\sqrt{S}} + \mathcal{O}(NM)$$

OOC_TRSM

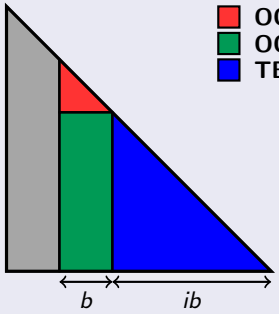


$$Q = \frac{N^2 M}{\sqrt{S}} + \mathcal{O}(NM)$$

TBS

$$Q = \frac{N^2 M}{\sqrt{2S}} + \frac{N^2}{2} + \mathcal{O}(NM \log N)$$

LBC



Communication volume:

$$\sum_{i \leqslant \frac{N}{b}} \frac{b^3}{3\sqrt{S}} + \frac{b^2(ib)}{\sqrt{S}} + \frac{(ib)^2 b}{\sqrt{2S}} + \frac{(ib)^2}{2} + \mathcal{O}(N^2 \log N)$$

$$= \frac{N}{b}\frac{b^3}{3\sqrt{S}} + \frac{b^3(\frac{N}{b})^2}{2\sqrt{S}} + \frac{b^3(\frac{N}{b})^3}{3\sqrt{2S}} + \frac{b^2(\frac{N}{b})^3}{6} + \mathcal{O}(N^2 \log N)$$

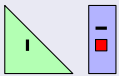
$$= \frac{b^2 N}{3\sqrt{S}} + \frac{bN^2}{2\sqrt{S}} + \frac{N^3}{3\sqrt{2S}} + \frac{\frac{N^3}{b}}{6} + \mathcal{O}(N^2 \log N)$$

LBC Algorithm: **L**arge **B**lock **C**holesky

OOC_CHOL

$$Q = \frac{N^3}{3\sqrt{S}} + \mathcal{O}(NM)$$

OOC_TRSM

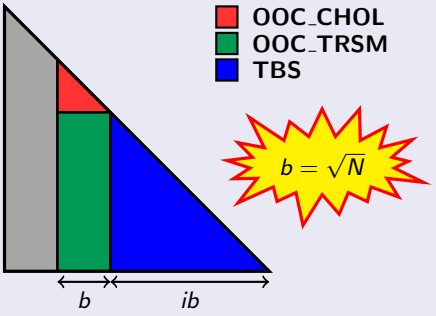


$$Q = \frac{N^2 M}{\sqrt{S}} + \mathcal{O}(NM)$$

TBS

$$Q = \frac{N^2 M}{\sqrt{2S}} + \frac{N^2}{2} + \mathcal{O}(NM \log N)$$

LBC



Communication volume:

$$\sum_{i \leqslant \frac{N}{b}} \frac{b^3}{3\sqrt{S}} + \frac{b^2(ib)}{\sqrt{S}} + \frac{(ib)^2 b}{\sqrt{2S}} + \frac{(ib)^2}{2} + \mathcal{O}(N^2 \log N)$$

$$= \frac{N}{b}\frac{b^3}{3\sqrt{S}} + \frac{b^3(\frac{N}{b})^2}{2\sqrt{S}} + \frac{b^3(\frac{N}{b})^3}{3\sqrt{2S}} + \frac{b^2(\frac{N}{b})^3}{6} + \mathcal{O}(N^2 \log N)$$

$$= \frac{b^2 N}{3\sqrt{S}} + \frac{bN^2}{2\sqrt{S}} + \frac{N^3}{3\sqrt{2S}} + \frac{\frac{N^3}{b}}{6} + \mathcal{O}(N^2 \log N)$$

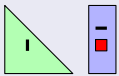
$$= \frac{N^3}{3\sqrt{2S}} + \mathcal{O}(N^{5/2})$$

LBC Algorithm: Large Block Cholesky

OOC_CHOL

$$Q = \frac{N^3}{3\sqrt{S}} + \mathcal{O}(NM)$$

OOC_TRSM

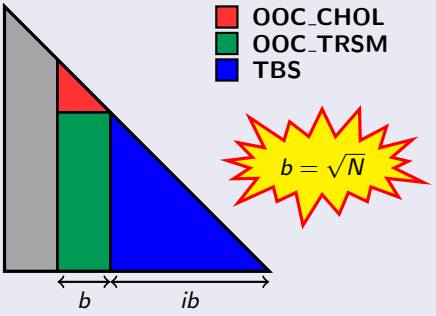


$$Q = \frac{N^2 M}{\sqrt{S}} + \mathcal{O}(NM)$$

TBS

$$Q = \frac{N^2 M}{\sqrt{2S}} + \frac{N^2}{2} + \mathcal{O}(NM \log N)$$

LBC



Communication volume:

$$\sum_{i \leqslant \frac{N}{b}} \frac{b^3}{3\sqrt{S}} + \frac{b^2(ib)}{\sqrt{S}} + \frac{(ib)^2 b}{\sqrt{2S}} + \frac{(ib)^2}{2} + \mathcal{O}(N^2 \log N)$$

$$= \frac{N}{b}\frac{b^3}{3\sqrt{S}} + \frac{b^3(\frac{N}{b})^2}{2\sqrt{S}} + \frac{b^3(\frac{N}{b})^3}{3\sqrt{2S}} + \frac{b^2(\frac{N}{b})^3}{6} + \mathcal{O}(N^2 \log N)$$

$$= \frac{b^2 N}{3\sqrt{S}} + \frac{bN^2}{2\sqrt{S}} + \frac{N^3}{3\sqrt{2S}} + \frac{\frac{N^3}{b}}{6} + \mathcal{O}(N^2 \log N)$$

$$= \frac{N^3}{3\sqrt{2S}} + \mathcal{O}(N^{5/2})$$

$$\rho = \sqrt{2S}$$

Conclusions

Summary: Communication Complexity

		Lower bound	Best Algorithm
SYRK	Before	$\frac{1}{2}\frac{N^2M}{\sqrt{S}}$	$1\frac{N^2M}{\sqrt{S}} + \frac{N^2}{2}$
	Ours	$\frac{1}{\sqrt{2}}\frac{N^2M}{\sqrt{S}}$	$\frac{1}{\sqrt{2}}\frac{N^2M}{\sqrt{S}} + \frac{N^2}{2} + \mathcal{O}(NM\log N)$
Cholesky	Before	$\frac{1}{6}\frac{N^3}{\sqrt{S}}$	$\frac{1}{3}\frac{N^3}{\sqrt{S}} + \mathcal{O}(N^2)$
	Ours	$\frac{1}{3\sqrt{2}}\frac{N^3}{\sqrt{S}}$	$\frac{1}{3\sqrt{2}}\frac{N^3}{\sqrt{S}} + \mathcal{O}(N^{5/2})$

Future Works

- Parallel Algorithms!
- Improve lower order terms
- Generalize to other kernels with data reuse

Ongoing (in Chameleon)

Symmetric Block Cyclic
$\Rightarrow$ Improves over Block Cyclic

2.5D implementations

Symmetric Block Cyclic Performance on bora

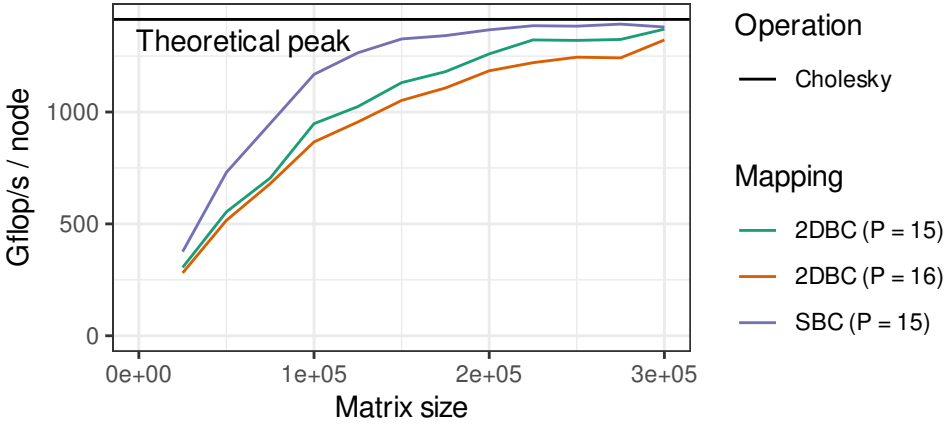


Figure: Gflop rate per processor VS matrix size ($P = 15$)