Symmetric Block-Cyclic Distribution: Fewer Communications Leads to Faster Dense Cholesky Factorization

Olivier Beaumont, Philippe Duchon, Lionel Eyraud-Dubois, Julien Langou, Mathieu Vérité

LaBRI, Inria Center at the University of Bordeaux
University of Colorado, Denver
Table of Contents

1 Introduction

2 Symmetric Distribution

3 Cholesky factorization

4 2.5D Cholesky implementation

5 Conclusions
Context

Data placement for Linear Algebra operations

- Linear Algebra is everywhere in Scientific Computing
  - Solving Partial Differential Equations becomes $Ax = b$ after discretization
- Very computationally intensive: distributed execution necessary
- Tightly coupled: importance of minimizing communications
- Objective: reduce the total volume of communications

In this talk

- Focus on symmetric operations: SYRK ($C += A \cdot A^T$), Cholesky
- Propose a Symmetric Block Cyclic distribution, improves over the standard 2DBC
- Propose a 2.5D variant of a task-based implementation of Cholesky factorization
- Provide an experimental validation with significantly improved performance
**Matrix Multiplication, 2DBC, communication volume**

**GEneral MMatrix Multiplication:** $C += A \cdot B$

on $P$ nodes

for $i = 1 \ldots M - 1$ do
  for $j = 1 \ldots M - 1$ do
    for $k = 1 \ldots N - 1$ do
      $C_{i,j} += A_{i,k} \cdot B_{k,j}$
Matrix Multiplication, 2DBC, communication volume

\[
A \cdot B = C
\]

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

\[
C_{i,j} := A_{i,k} \cdot B_{k,j} \quad \text{(tiled, owner-computes)}
\]

Distributed execution with a runtime system

- Automatically builds the dependency graph from sequential code
- Data is distributed on the nodes according to the distribution
- Communications are managed seamlessly by the runtime system
Matrix Multiplication, 2DBC, communication volume

\[
C_{i,j} += A_{i,k} \cdot B_{k,j} \quad \text{(tiled, owner-computes)}
\]

2D Block Cyclic 2 × 4, \( P = 8 \) nodes

Distributed execution with a runtime system

- Automatically builds the dependency graph from sequential code
- Data is distributed on the nodes according to the distribution
- Communications are managed seamlessly by the runtime system
Matrix Multiplication, 2DBC, communication volume

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

\[
\begin{align*}
\text{for } i = 1 \ldots M - 1 & \text{ do} \\
\text{for } j = 1 \ldots M - 1 & \text{ do} \\
\text{for } k = 1 \ldots N - 1 & \text{ do} \\
C_{i,j} & += A_{i,k} \cdot B_{k,j} \quad \text{(tiled, owner-computes)}
\end{align*}
\]

Distributed execution with a runtime system

- Automatically builds the dependency graph from sequential code
- Data is distributed on the nodes according to the distribution
- Communications are managed seamlessly by the runtime system
Matrix Multiplication, 2DBC, communication volume

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

$p$

$q$

for $i = 1 \ldots M - 1$ do

for $j = 1 \ldots M - 1$ do

for $k = 1 \ldots N - 1$ do

$C_{i,j} += A_{i,k} \cdot B_{k,j}$ (tiled, owner-computes)

Communication volume: number of values communicated

Each tile of $A$ is used by $q$ nodes, each tile of $B$ by $p$ nodes.

$$V = MN(q - 1) + MN(p - 1) = MN(p + q - 2)$$
Arithmetic Intensity: \( \rho = \frac{\text{number of computations}}{\text{communication volume}} \)

- Total computations: \( 2M^2N \) (\( N \) products and \( N \) additions per element of \( C \))

\[
\rho = \frac{2M^2N}{MN(p + q - 2)}
\]
Arithmetic Intensity: \( \rho = \frac{\text{number of computations}}{\text{communication volume}} \)

- Total computations: \( 2M^2N \) (\( N \) products and \( N \) additions per element of \( C \))

\[
\rho = \frac{2M^2N}{MN(p + q - 2)} \approx \frac{2M^2N}{2MN\sqrt{P}} = \frac{M}{\sqrt{P}} \quad \text{if} \quad p \approx q \approx \sqrt{P}
\]
## Arithmetic Intensity: \( \rho = \frac{\text{number of computations}}{\text{communication volume}} \)

- Total computations: \( 2M^2N \) (\( N \) products and \( N \) additions per element of \( C \))

\[
\rho = \frac{2M^2N}{MN(p + q - 2)} \approx \frac{2M^2N}{2MN\sqrt{P}} = \frac{M}{\sqrt{P}} \quad \text{if} \quad p \approx q \approx \sqrt{P}
\]

- \( S = \) number of elements of \( C \) per node = \( \frac{M^2}{P} \)

\[
\rho = \sqrt{S}
\]
2DBC, Arithmetic Intensity

Arithmetic Intensity: $\rho = \frac{\text{number of computations}}{\text{communication volume}}$

- Total computations: $2M^2N$ ($N$ products and $N$ additions per element of $C$)
  
  $$\rho = \frac{2M^2N}{MN(p + q - 2)} \approx \frac{2M^2N}{2MN\sqrt{P}} = \frac{M}{\sqrt{P}} \quad \text{if } p \approx q \approx \sqrt{P}$$

- $S = \text{number of elements of } C \text{ per node} = \frac{M^2}{P}$

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

- This is optimal
Table of Contents

1 Introduction

2 Symmetric Distribution

3 Cholesky factorization

4 2.5D Cholesky implementation

5 Conclusions
Symmetric multiplication is SYRK: \( C += A \cdot A^T \)

SYRK: \( C += A \cdot A^T \) (SYmmetric Rank-K update)

dominant part of Cholesky factorization

(solve \( A = L \cdot L^T \) for symmetric positive definite matrix \( A \))
Symmetric multiplication is SYRK: \( C += A \cdot A^T \)

SYRK: \( C += A \cdot A^T \) (SYmmetric Rank-K update)

...dominant part of Cholesky factorization

(solve \( A = L \cdot L^T \) for symmetric positive definite matrix \( A \))
Symmetric multiplication is SYRK: $C += A \cdot A^T$

2D Block Cyclic $2 \times 4$, $P = 8$ nodes
Symmetric multiplication is SYRK: $C := A \cdot A^T$

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

Arithmetic Intensity: $\rho = \frac{M^2 N^2}{MN \sqrt{P}} = \frac{M^2}{\sqrt{P}}$, and since $S = \frac{M^2}{2P}$ now:

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

Upper bound (tight): $\sqrt{2S} [SPAA'2022]$
Symmetric multiplication is SYRK: \( C \leftarrow A \cdot A^T \)

2D Block Cyclic \( 2 \times 4 \), \( P = 8 \) nodes

N
\[ A^T \]
M

Arithmetic Intensity:
\[ \rho = \frac{M^2 N^2}{\sqrt{P}} \]

Now:
\[ \rho = \sqrt{S} \sqrt{\frac{1}{2}} \]

Upper bound (tight):
\[ \sqrt{2} S \]
Symmetric multiplication is SYRK: \( C \leftarrow A \cdot A^T \)

2D Block Cyclic \( 2 \times 4 \), \( P = 8 \) nodes

Communication volume
Each tile of \( A \) is used by \( p + q - 1 \) nodes:
\[
V = MN(p + q - 2)
\]

Arithmetic Intensity:
\[
\rho = \frac{M^2N^2}{\sqrt{P}} \quad \text{and since} \quad S = \frac{M^2}{2P}
\]

\[\rho = \sqrt{2S} \sqrt{2} \]

Upper bound (tight):
\[\sqrt{2S} \[\text{SPAA’2022}\]

Beaumont et. al

Symmetric Block Cyclic Distribution
Symmetric multiplication is SYRK: $C += A \cdot A^T$

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

Communication volume

Each tile of $A$ is used by $p + q - 1$ nodes:

$V = MN(p + q - 2)$

Arithmetic Intensity:

$\rho = \frac{M^2N}{\sqrt{P}}$, and since $S = \frac{M^2}{2P}$ now:

$\rho = \sqrt{\frac{S}{2}} \sqrt{\frac{2}{S}}$
Symmetric multiplication is SYRK: $C += A \cdot A^T$

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

Communication volume

Each tile of $A$ is used by $p + q - 1$ nodes: $V = MN(p + q - 2)$
Symmetric multiplication is SYRK: $C += A \cdot A^T$

2D Block Cyclic $2 \times 4$, $P = 8$ nodes

Communication volume

Each tile of $A$ is used by $p + q - 1$ nodes: $V = MN(p + q - 2)$

Arithmetic Intensity: $\rho = \frac{M^2N}{2MN\sqrt{P}} = \frac{M}{2\sqrt{P}}$, and since $S = \frac{M^2}{2P}$ now: $\rho = \frac{\sqrt{S}}{\sqrt{2}}$
Symmetric multiplication is SYRK: 
\[ C += A \cdot A^T \]

Communication volume

Each tile of \( A \) is used by \( p + q - 1 \) nodes: \( V = MN(p + q - 2) \)

Arithmetic Intensity: \( \rho = \frac{\frac{M^2N}{2MN\sqrt{P}}}{\sqrt{2N \sqrt{P}}} = \frac{\frac{M}{\sqrt{2P}}}{2\sqrt{P}} \), and since \( S = \frac{M^2}{2P} \) now: \( \rho = \frac{\sqrt{S}}{\sqrt{2}} \)

Upper bound (tight): \( \sqrt{2S} \) [SPAA'2022]
Goal: same nodes on rows and columns
Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

$P = \frac{r^2}{2} \iff r = \sqrt{2P}$
Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

$$P = \frac{r^2}{2} \quad \iff \quad r = \sqrt{2P}$$
Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

$\begin{bmatrix}
7 & 1 & 2 & 4 \\
1 & 8 & 3 & 5 \\
2 & 3 & 7 & 6 \\
4 & 5 & 6 & 8
\end{bmatrix}$

$P = \frac{r^2}{2} \iff r = \sqrt{2P}$
SBC: Symmetric Block Cyclic – basic version

Goal: same nodes on rows and columns

Symmetric Block Cyclic \( P = 8 \)

\[ P = \frac{r^2}{2} \iff r = \sqrt{2P} \]
SBC: Symmetric Block Cyclic – basic version

Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

Communication volume
One tile of $A$ is needed by $r$ nodes:

$$V = MN(r - 1)$$

Arithmetic intensity:

$$\rho = \frac{M^2N}{MN(r - 1)} = M\sqrt{2P}$$

$$P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}$$
SBC: Symmetric Block Cyclic – basic version

Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

Communication volume

One tile of $A$ is needed by $r$ nodes: $V = MN(r - 1)$

Arithmetic intensity: $\rho = \frac{M^2N}{MN(r - 1)} = \frac{M}{\sqrt{2P}} = \sqrt{S}$

Beaumont et. al Symmetric Block Cyclic Distribution
SBC: Extended version

Limitations of basic version

- not valid for odd values of $r$;
- only a small subset of nodes on the diagonal of $A$. 

Alternative way of allocating diagonal tiles of the SBC pattern

Keep the set of $r$ ($r - 1$) nodes, reuse them on the diagonal
Create several patterns, alternate between them on matrix $A$. 

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

Generic pattern

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

Pattern 1

1 3 6 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

Pattern 2

2 5 9 4
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

Create $r - 1$ patterns for odd $r$ 
Create $r$ patterns for even $r$ 

Beaumont et. al Symmetric Block Cyclic Distribution
SBC: Extended version

Limitations of basic version
- not valid for odd values of $r$;
- only a small subset of nodes on the diagonal of $A$.

Alternative way of allocating diagonal tiles of the SBC pattern
- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create several patterns, alternate between them on matrix $A$. 

Generic pattern
**SBC: Extended version**

**Limitations of basic version**
- not valid for **odd** values of $r$;
- only a small subset of nodes on the diagonal of $A$.

**Alternative way of allocating diagonal tiles of the SBC pattern**
- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create **several** patterns, alternate between them on matrix $A$.

<table>
<thead>
<tr>
<th>Generic pattern</th>
<th>Pattern 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
</tr>
</tbody>
</table>

Beaumont et. al
Symmetric Block Cyclic Distribution
SBC: Extended version

Limitations of basic version

- not valid for **odd** values of $r$;
- only a small subset of nodes on the diagonal of $A$.

Alternative way of allocating diagonal tiles of the SBC pattern

- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create **several** patterns, alternate between them on matrix $A$. 

![Generic pattern](image1.png)

![Pattern 1](image2.png)
SBC: Extended version

Limitations of basic version
- not valid for odd values of $r$;
- only a small subset of nodes on the diagonal of $A$.

Alternative way of allocating diagonal tiles of the SBC pattern
- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create several patterns, alternate between them on matrix $A$.

Generic pattern

Pattern 1
Limitations of basic version

- not valid for **odd** values of $r$;
- only a small subset of nodes on the diagonal of $A$.

Alternative way of allocating diagonal tiles of the SBC pattern

- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create **several** patterns, alternate between them on matrix $A$. 

---

![Generic pattern](image1)

![Pattern 1](image2)

![Pattern 2](image3)
**SBC: Extended version**

**Limitations of basic version**
- not valid for **odd** values of $r$;
- only a small subset of nodes on the diagonal of $A$.

**Alternative way of allocating diagonal tiles of the SBC pattern**
- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create **several** patterns, alternate between them on matrix $A$.

![Generic pattern](#)

![Pattern 1](#)

![Pattern 2](#)
SBC: Extended version

Limitations of basic version

- not valid for odd values of $r$;
- only a small subset of nodes on the diagonal of $A$.

Alternative way of allocating diagonal tiles of the SBC pattern

- Keep the set of $\frac{r(r-1)}{2}$ nodes, reuse them on the diagonal
- Create several patterns, alternate between them on matrix $A$.

- Create $\frac{r-1}{2}$ patterns for odd $r$
- Create $r-1$ patterns for even $r$
Table of Contents

1 Introduction

2 Symmetric Distribution

3 Cholesky factorization

4 2.5D Cholesky implementation

5 Conclusions
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do  
\[ A_{i,i} \leftarrow \text{POTRF}(A_{i,i}) \]
for $j = i + 1 \ldots N - 1$ do  
\[ A_{j,i} \leftarrow \text{TRSM}(A_{j,i}, A_{i,i}) \]
for $k = i + 1 \ldots N - 1$ do  
\[ A_{k,k} \leftarrow \text{SYRK}(A_{k,k}, A_{k,i}) \]
for $j = k + 1 \ldots N - 1$ do  
\[ A_{j,k} \leftarrow \text{GEMM}(A_{j,k}, A_{j,i}, A_{k,i}) \]

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do
  $A_{i,i} \leftarrow \text{POTRF}(A_{i,i})$
  for $j = i + 1 \ldots N - 1$ do
    $A_{j,i} \leftarrow \text{TRSM}(A_{j,i}, A_{i,i})$
  for $k = i + 1 \ldots N - 1$ do
    $A_{k,k} \leftarrow \text{SYRK}(A_{k,k}, A_{k,i})$
    for $j = k + 1 \ldots N - 1$ do
      $A_{j,k} \leftarrow \text{GEMM}(A_{j,k}, A_{j,i}, A_{k,i})$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do
   $A_{i,i} \leftarrow$ POTRF($A_{i,i}$)
for $j = i + 1 \ldots N - 1$ do
   $A_{j,i} \leftarrow$ TRSM($A_{j,i}$, $A_{i,i}$)
for $k = i + 1 \ldots N - 1$ do
   $A_{k,k} \leftarrow$ SYRK($A_{k,k}$, $A_{k,i}$)
for $j = k + 1 \ldots N - 1$ do
   $A_{j,k} \leftarrow$ GEMM($A_{j,k}$, $A_{j,i}$, $A_{k,i}$)

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do
    $A_{i,i} \leftarrow$ POTRF($A_{i,i}$)
for $j = i + 1 \ldots N - 1$ do
    $A_{j,i} \leftarrow$ TRSM($A_{j,i}, A_{i,i}$)
for $k = i + 1 \ldots N - 1$ do
    $A_{k,k} \leftarrow$ SYRK($A_{k,k}, A_{k,i}$)
for $j = k + 1 \ldots N - 1$ do
    $A_{j,k} \leftarrow$ GEMM($A_{j,k}, A_{j,i}, A_{k,i}$)

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations

Beaumont et. al
Symmetric Block Cyclic Distribution
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do
  $A_{i,i} \leftarrow \text{POTRF}(A_{i,i})$
  for $j = i + 1 \ldots N - 1$ do
    $A_{j,i} \leftarrow \text{TRSM}(A_{j,i}, A_{i,i})$
  for $k = i + 1 \ldots N - 1$ do
    $A_{k,k} \leftarrow \text{SYRK}(A_{k,k}, A_{i,i})$
    for $j = k + 1 \ldots N - 1$ do
      $A_{j,k} \leftarrow \text{GEMM}(A_{j,k}, A_{j,i}, A_{k,i})$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do
   $A_{i,i} \leftarrow $ POTRF($A_{i,i}$)
   for $j = i + 1 \ldots N - 1$ do
      $A_{j,i} \leftarrow $ TRSM($A_{j,i}$, $A_{i,i}$)
   for $k = i + 1 \ldots N - 1$ do
      $A_{k,k} \leftarrow $ SYRK($A_{k,k}$, $A_{k,i}$)
      for $j = k + 1 \ldots N - 1$ do
         $A_{j,k} \leftarrow $ GEMM($A_{j,k}$, $A_{j,i}$, $A_{k,i}$)

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

for $i = 1 \ldots N - 1$ do
    $A_{i,i} \leftarrow \text{POTRF}(A_{i,i})$
    for $j = i + 1 \ldots N - 1$ do
        $A_{j,i} \leftarrow \text{TRSM}(A_{j,i}, A_{i,i})$
    for $k = i + 1 \ldots N - 1$ do
        $A_{k,k} \leftarrow \text{SYRK}(A_{k,k}, A_{k,i})$
        for $j = k + 1 \ldots N - 1$ do
            $A_{j,k} \leftarrow \text{GEMM}(A_{j,k}, A_{j,i}, A_{k,i})$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix $A$

Right-looking Cholesky is mainly iterated SYRK

- One iteration: factorize panel, update trailing matrix (SYRK)
- Typical MPI-based implementations synchronize between iterations
- Task-based allows for large lookahead and thus more parallelism
- Automatic handling of communications: easy to change the data allocation
Experimental results

Experimental setting

- **bora** nodes of PlaFRIM, Bordeaux:
  - 42 nodes, Intel Xeon Skylake Gold 6240, 36 cores per node
  - 100Gb/s OmniPath network

- **chameleon** library, based on **starpu** runtime

- Intel MKL 2020, Open MPI version 4.0.3

- One **starpu** process per node, each task executed on one core

- One core reserved for handling MPI comms, one for task submission & scheduling

- Tile size $b = 500$
Experimental results: Cholesky factorization on $P \sim 28$ nodes

**Chameleon** + **StarPU** on **bora** cluster (36 cores per node: 1008 cores)

Theoretical peaks

<table>
<thead>
<tr>
<th>Mapping</th>
<th>(34 and 36 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2DBC ($P=28$, $pxq=7x4$)</td>
<td></td>
</tr>
<tr>
<td>2DBC ($P=30$, $pxq=6x5$)</td>
<td></td>
</tr>
<tr>
<td>SBC ($P=28$, $r=8$)</td>
<td></td>
</tr>
</tbody>
</table>

![Graph showing performance results](image)
2.5D Cholesky implementation

Main ideas

- **Replicate** the matrix on $c$ slices of nodes
- Perform iteration $k$ on slice $k \mod c$: updates of a tile accumulate on $c$ nodes
- Reduce operation at the end to merge all updates
- **Task-based** implementation: high lookahead avoids idle time
Main ideas

- **Replicate** the matrix on $c$ slices of nodes
- Perform iteration $k$ on slice $k \mod c$: updates of a tile accumulate on $c$ nodes
- Reduce operation at the end to merge all updates
- **Task-based** implementation: high lookahead avoids idle time
2.5D Cholesky implementation

Main ideas

- **Replicate** the matrix on $c$ slices of nodes
- Perform iteration $k$ on slice $k \mod c$: updates of a tile accumulate on $c$ nodes
- Reduce operation at the end to merge all updates
- **Task-based** implementation: high lookahead avoids idle time
### 2.5D Cholesky: communication volume

- Can be used with **any** 2D distribution, reproduced on \( c \) slices. \( \frac{P}{c} \) nodes per slice
- Communication volume:
  - **2DBC**: \( M^2(2\sqrt{\frac{P}{c}} + c - 1) \)
  - **SBC**: \( M^2(\sqrt{\frac{2P}{c}} + c - 1) \)

### With limited memory \( S \)

- Use as many slices as possible: \( c = \frac{2PS}{M^2} \)
- Communication volume: \( V = \frac{1}{2} \frac{M^3}{\sqrt{S}} + o(M^3) \) \[\text{[Kwasniewski et al., SC'21]}\]: \( V \sim 1 \cdot \frac{M^3}{\sqrt{S}} \)

### With large memory

- Select the value of \( c \) to minimize the communication volume
- For **SBC**, we obtain \( c \sim \frac{3\sqrt{P}}{2} \) and \( r = 2c \), so that \( V \sim 3\sqrt[3]{\frac{1}{2}} \cdot S^{\frac{3}{2}} \)
- With **2DBC**, \( c \sim \frac{3}{\sqrt{P}} \) and \( V \sim 3 \cdot S^{\frac{3}{2}} \): factor \( \sqrt{2} \approx 1.26 \) on comms and memory
2.5D version: experimental results ($c = 3$)

**Theoretical peaks**
(34 and 36 cores)

---

**Mapping**
- 2DBC ($P=28$, $pxq=7x4$)
- 2DBC ($P=30$, $pxq=6x5$)
- 2DBC ($P=32$, $pxq=8x4$)
- SBC ($P=28$, $r=8$)
- 2DBC 2.5D ($P=30$, $pxq=5x2$, $c=3$)
- SBC 2.5D ($P=30$, $r=5$, $c=3$)

---

**Library**
- Chameleon-StarPU
- COnfCHOX

---

---

---
Conclusions

Contributions

- New **Symmetric Block Cyclic** distribution, adapted for SYRK & Cholesky
- Lowers communication volume by a factor of \(\sqrt{2}\)
- Task-based **2.5D** implementation of Cholesky factorization
- Significantly improved performance and scalability
- Can be applied to many other symmetric computations

Open questions

- **SBC**: each node appears twice. Would higher counts improve the performance further?
- Efficiency of **2.5D** for Matrix Multiplication:
  - In Cholesky, some reductions start very early \(\Rightarrow\) overlap with computations
  - For GEMM/SYRK, same amount of work on all tiles: how to organize the reductions?
Ongoing work: recent results (under evaluation)

**Optimal TBC distribution**

- Based on the TBS sequential algorithm from [SPAA’2022]
- Used in the context of the SYMM operation
- Also improves the performance of Cholesky

<table>
<thead>
<tr>
<th>Pattern cost</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2DBC</td>
<td></td>
</tr>
<tr>
<td>G−2DBC</td>
<td></td>
</tr>
<tr>
<td>GCRM</td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td></td>
</tr>
</tbody>
</table>

Distributions for any value of $P$ are efficient for specific values of $P$.

Proposed Generalized 2DBC for non-symmetric case

Proposed greedy GCR&M for symmetric case
Ongoing work: recent results

(under evaluation)

Optimal **TBC** distribution

- Based on the **TBS** sequential algorithm from [SPAA’2022]
- Used in the context of the **SYMM** operation
- Also improves the performance of **Cholesky**
Ongoing work: recent results (under evaluation)

Optimal TBC distribution

- Based on the TBS sequential algorithm from [SPAA’2022]
- Used in the context of the SYMM operation
- Also improves the performance of Cholesky

Distributions for any value of $P$

- 2DBC and SBC only efficient for specific values of $P$
- Proposed Generalized 2DBC for non-symmetric case
- Proposed greedy GCR&M for symmetric case
Thank you!

Questions?

https://solverstack.gitlabpages.inria.fr/chameleon/