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

SC22 Dallas, TX | hpc accelerates.

Inria LaBRI université de BORDEAUX CU Denver CU IN THE CITY

Table of Contents

Context

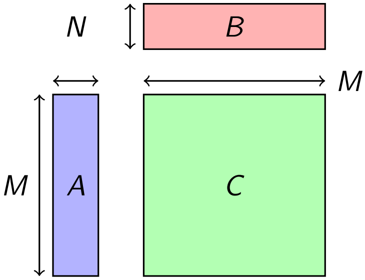
Data placement for Linear Algebra operations

- 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 ($C \mathrel{+}= A \cdot A^T$), 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

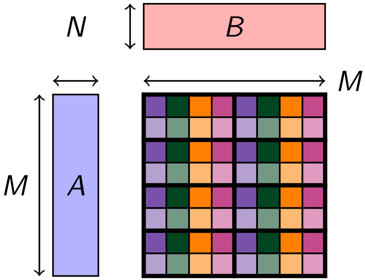
Matrix Multiplication, 2DBC, communication volume



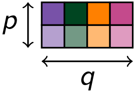
GEneral **M**atrix **M**ultiplication: $C \mathrel{+}= A \cdot B$
on P nodes

for $i = 1 \ldots M-1$ **do**
  for $j = 1 \ldots M-1$ **do**
    for $k = 1 \ldots N-1$ **do**
      $\mathbf{C}_{i,j} \mathrel{+}= \mathbf{A}_{i,k} \cdot \mathbf{B}_{k,j}$

Matrix Multiplication, 2DBC, communication volume



2D Block Cyclic 2×4, $P = 8$ nodes

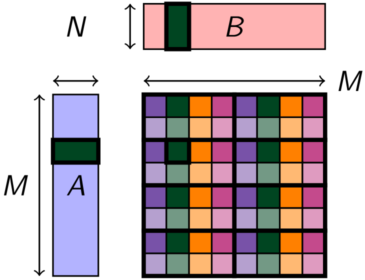


for $i = 1 \ldots M-1$ **do**
  for $j = 1 \ldots M-1$ **do**
    for $k = 1 \ldots N-1$ **do**
      $\mathbf{C}_{i,j} \mathrel{+}= \mathbf{A}_{i,k} \cdot \mathbf{B}_{k,j}$ (tiled, owner-computes)

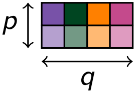
Distributed execution with a runtime system

- 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



2D Block Cyclic 2×4, $P = 8$ nodes

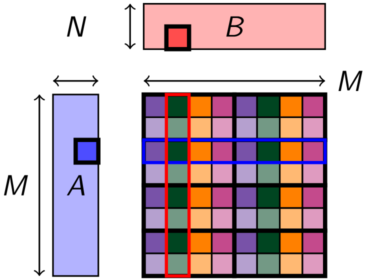


for $i = 1 \ldots M-1$ **do**
  for $j = 1 \ldots M-1$ **do**
    for $k = 1 \ldots N-1$ **do**
      $\mathbf{C}_{i,j} \mathrel{+}= \mathbf{A}_{i,k} \cdot \mathbf{B}_{k,j}$ (tiled, owner-computes)

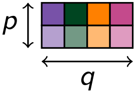
Distributed execution with a runtime system

- 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



2D Block Cyclic 2×4, $P = 8$ nodes

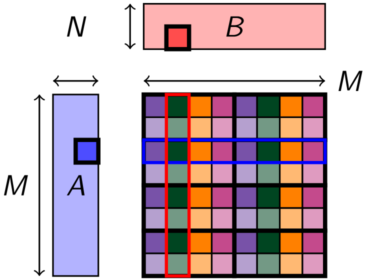


for $i = 1 \dots M-1$ **do**
  for $j = 1 \dots M-1$ **do**
    for $k = 1 \dots N-1$ **do**
      $\mathbf{C}_{i,j} \mathrel{+}= \mathbf{A}_{i,k} \cdot \mathbf{B}_{k,j}$ (tiled, owner-computes)

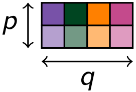
Distributed execution with a runtime system

- 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



2D Block Cyclic 2×4, $P = 8$ nodes



for $i = 1 \ldots M-1$ **do**
  for $j = 1 \ldots M-1$ **do**
    for $k = 1 \ldots N-1$ **do**
      $\mathbf{C}_{i,j} \mathrel{+}= \mathbf{A}_{i,k} \cdot \mathbf{B}_{k,j}$ (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.

$$V = MN(q-1) + MN(p-1) = MN(p+q-2)$$

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)

$$\rho = \frac{2M^2N}{MN(p+q-2)}$$

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)

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

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)

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

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

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

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)

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

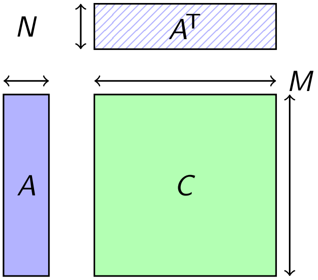
- $S =$ number of elements of C per node $= \frac{M^2}{P}$

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

- This is **optimal**

Table of Contents

Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^\mathsf{T}$

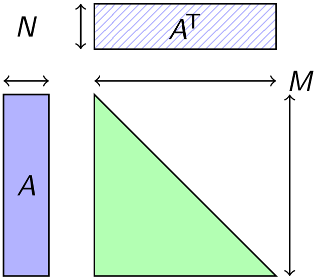


SYRK: $C \mathrel{+}= A \cdot A^\mathsf{T}$ (**SY**mmetric **R**ank-**K** update)

dominant part of Cholesky factorization

(solve $A = L \cdot L^\mathsf{T}$ for symmetric positive definite matrix A)

Symmetric multiplication is SYRK: $C += A \cdot A^{\mathsf{T}}$

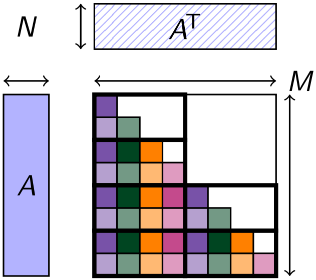


SYRK: $C += A \cdot A^{\mathsf{T}}$ (**SY**mmetric **R**ank-**K** update)

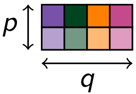
dominant part of Cholesky factorization

(solve $A = L \cdot L^{\mathsf{T}}$ for symmetric positive definite matrix A)

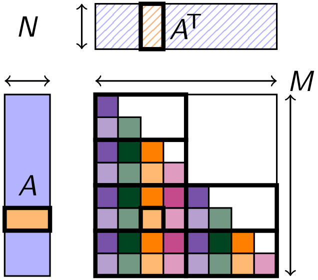
Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^\mathsf{T}$



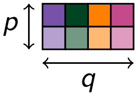
2D Block Cyclic 2×4, $P = 8$ nodes



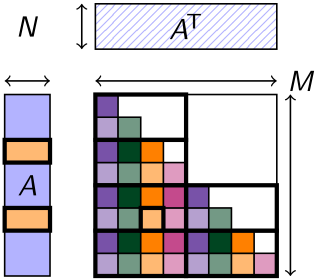
Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^{\mathsf{T}}$



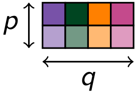
2D Block Cyclic 2×4, $P = 8$ nodes



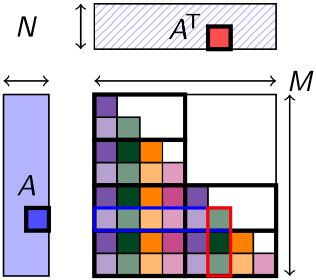
Symmetric multiplication is SYRK: $C += A \cdot A^{\mathsf{T}}$



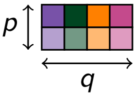
2D Block Cyclic 2×4, $P = 8$ nodes



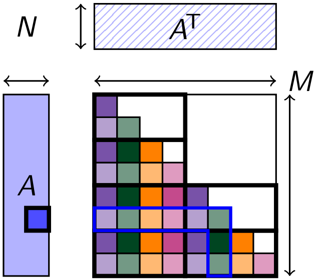
Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^{\mathsf{T}}$



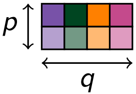
2D Block Cyclic 2×4, $P = 8$ nodes



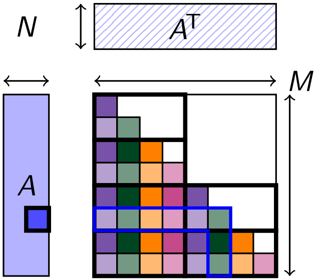
Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^{\mathsf{T}}$



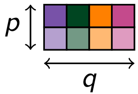
2D Block Cyclic 2×4, $P = 8$ nodes



Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^{\mathsf{T}}$



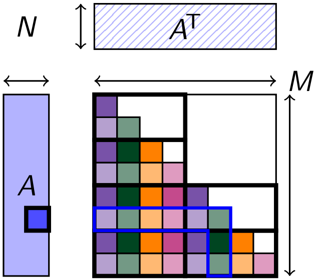
2D Block Cyclic 2×4, $P = 8$ nodes



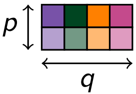
Communication volume

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

Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^\mathsf{T}$



2D Block Cyclic 2×4, $P = 8$ nodes

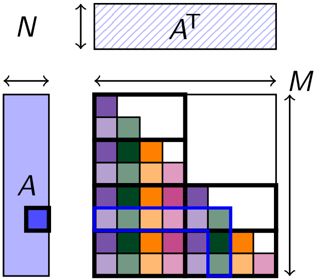


Communication volume

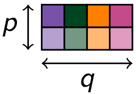
Each tile of A is used by $p + q - 1$ nodes: $V = MN(p + q - 2)$

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

Symmetric multiplication is SYRK: $C \mathrel{+}= A \cdot A^{\mathsf{T}}$



2D Block Cyclic 2×4, $P = 8$ nodes



Communication volume

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

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

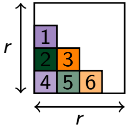
Upper bound (tight): $\sqrt{2S}$ [SPAA'2022]

Goal: same nodes on rows and columns

SBC: Symmetric Block Cyclic – basic version

Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$



$$P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}$$

SBC: Symmetric Block Cyclic – basic version

Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

	1	2	4
1		3	5
2	3		6
4	5	6	

r × r

$$P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}$$

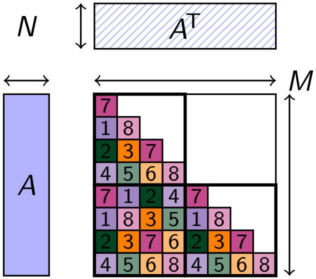
SBC: Symmetric Block Cyclic – basic version

Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

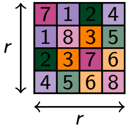
$$
r \updownarrow \underbrace{\begin{array}{|c|c|c|c|}
\hline
7 & 1 & 2 & 4 \\ \hline
1 & 8 & 3 & 5 \\ \hline
2 & 3 & 7 & 6 \\ \hline
4 & 5 & 6 & 8 \\ \hline
\end{array}}_{r}
\qquad P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}
$$

SBC: Symmetric Block Cyclic – basic version

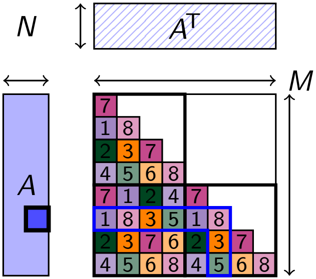


Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$

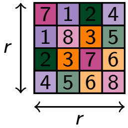


$$P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}$$



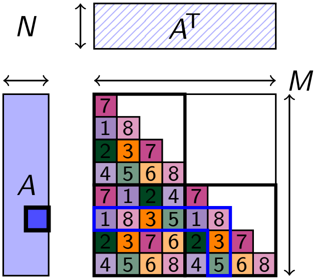
Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$



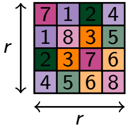
$$P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}$$

SBC: Symmetric Block Cyclic – basic version



Goal: same nodes on rows and columns

Symmetric Block Cyclic $P = 8$



$$P = \frac{r^2}{2} \quad \Leftrightarrow \quad r = \sqrt{2P}$$

Communication volume

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

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

Limitations of basic version

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

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.

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

Generic pattern

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.

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

Generic pattern

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

Pattern 1

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.

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

Generic pattern

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

Pattern 1

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.

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

Generic pattern

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

Pattern 1

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.

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

Generic pattern

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

Pattern 1

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

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.

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

Generic pattern

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

Pattern 1

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

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.

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

Generic pattern

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

Pattern 1

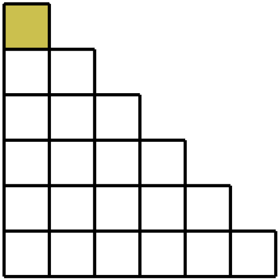
2	1	2	4	7
1	5	3	5	8
2	3	9	6	9
4	5	6	4	10
7	8	9	10	8

Pattern 2

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

Table of Contents

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

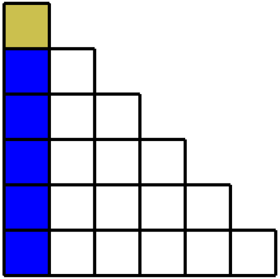


for $i = 1 \ldots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \text{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \text{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \text{SYRK}(\mathbf{A}_{k,k},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

Right-looking Cholesky is mainly iterated SYRK

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

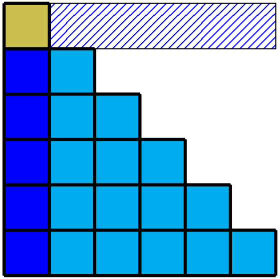
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix A



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

Right-looking Cholesky is mainly iterated SYRK

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

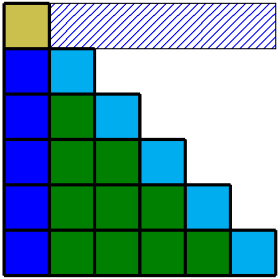


for $i = 1 \ldots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \text{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \text{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \text{SYRK}(\mathbf{A}_{k,k},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

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

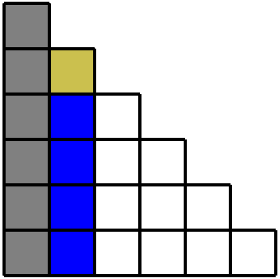


for $i = 1 \dots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \text{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \dots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \text{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \dots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \text{SYRK}(\mathbf{A}_{k,k}, \mathbf{A}_{k,i})$
    for $j = k+1 \dots N-1$ **do**
      $\mathbf{A}_{j,k} \leftarrow \text{GEMM}(\mathbf{A}_{j,k}, \mathbf{A}_{j,i}, \mathbf{A}_{k,i})$

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 $A = L \cdot L^{\mathrm{T}}$ for SPD matrix A

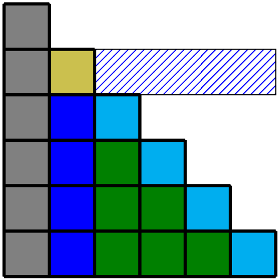


for $i = 1 \ldots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \mathrm{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \mathrm{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \mathrm{SYRK}(\mathbf{A}_{k,k}, \mathbf{A}_{k,i})$
    for $j = k+1 \ldots N-1$ **do**
      $\mathbf{A}_{j,k} \leftarrow \mathrm{GEMM}(\mathbf{A}_{j,k}, \mathbf{A}_{j,i}, \mathbf{A}_{k,i})$

Right-looking Cholesky is mainly iterated SYRK

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

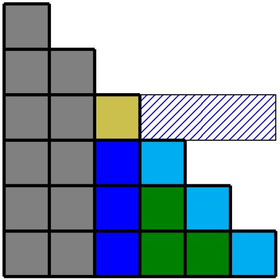
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix A



for $i = 1 \ldots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \mathrm{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \mathrm{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \mathrm{SYRK}(\mathbf{A}_{k,k}, \mathbf{A}_{k,i})$
    for $j = k+1 \ldots N-1$ **do**
      $\mathbf{A}_{j,k} \leftarrow \mathrm{GEMM}(\mathbf{A}_{j,k}, \mathbf{A}_{j,i}, \mathbf{A}_{k,i})$

Right-looking Cholesky is mainly iterated SYRK

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

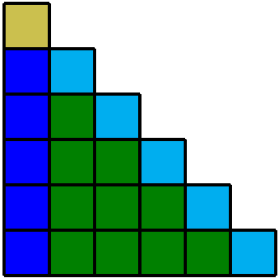


for $i = 1 \ldots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \text{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \text{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \ldots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \text{SYRK}(\mathbf{A}_{k,k},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

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 $A = L \cdot L^{\mathrm{T}}$ for SPD matrix A

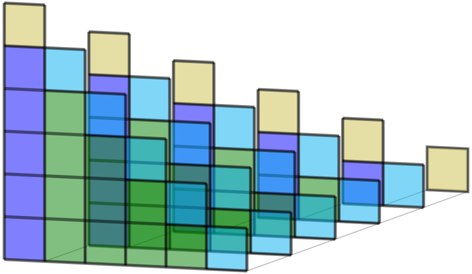


for $i = 1 \dots N-1$ **do**
  $\mathbf{A}_{i,i} \leftarrow \mathrm{POTRF}(\mathbf{A}_{i,i})$
  for $j = i+1 \dots N-1$ **do**
    $\mathbf{A}_{j,i} \leftarrow \mathrm{TRSM}(\mathbf{A}_{j,i}, \mathbf{A}_{i,i})$
  for $k = i+1 \dots N-1$ **do**
    $\mathbf{A}_{k,k} \leftarrow \mathrm{SYRK}(\mathbf{A}_{k,k}, \mathbf{A}_{k,i})$
    for $j = k+1 \dots N-1$ **do**
      $\mathbf{A}_{j,k} \leftarrow \mathrm{GEMM}(\mathbf{A}_{j,k}, \mathbf{A}_{j,i}, \mathbf{A}_{k,i})$

Right-looking Cholesky is mainly iterated SYRK

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

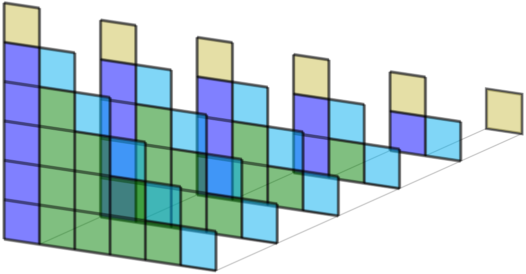
Task-based Cholesky – solve $A = L \cdot L^{\mathsf{T}}$ for SPD matrix A



Right-looking Cholesky is mainly iterated SYRK

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

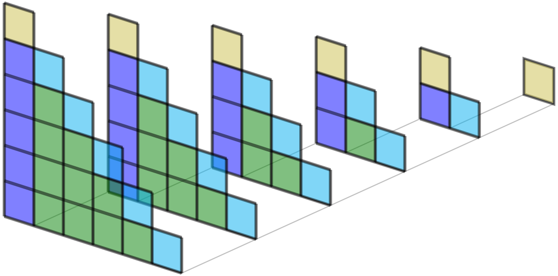
Task-based Cholesky – solve $A = L \cdot L^T$ for SPD matrix A



Right-looking Cholesky is mainly iterated SYRK

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

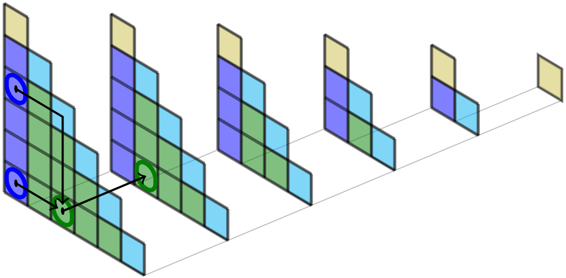
Task-based Cholesky – solve $A = L \cdot L^{\mathsf{T}}$ for SPD matrix A



Right-looking Cholesky is mainly iterated SYRK

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

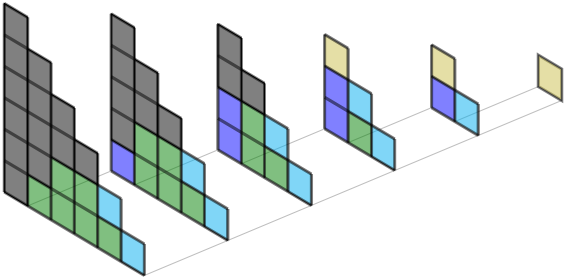
Task-based Cholesky – solve $A = L \cdot L^{\mathsf{T}}$ for SPD matrix A



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 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)

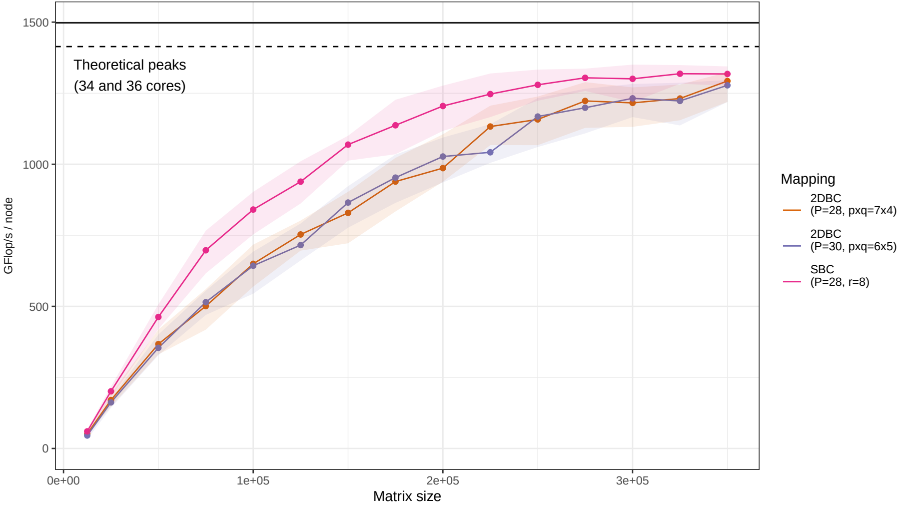
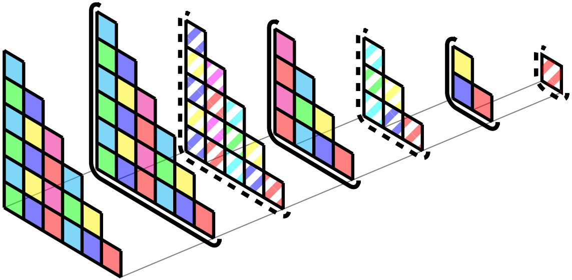


Table of Contents

Main ideas

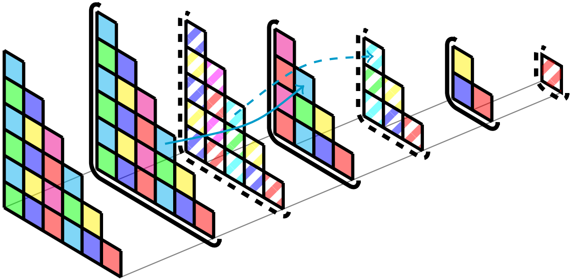
- **Replicate** the matrix on c slices of nodes
- Perform iteration k on slice $k \mod c$: updates of a tile accumulate on c nodes
- Reduce operation at the end to merge all updates
- **Task-based** implentation: high lookahead avoids idle time



2.5D Cholesky implementation

Main ideas

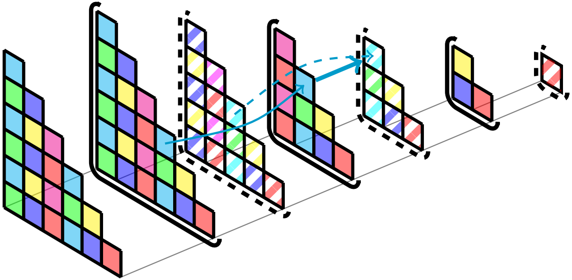
- **Replicate** the matrix on c slices of nodes
- Perform iteration k on slice $k \mod c$: updates of a tile accumulate on c nodes
- Reduce operation at the end to merge all updates
- **Task-based** implentation: high lookahead avoids idle time



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** implentation: 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: $\boxed{V = \frac{1}{2}\frac{M^3}{\sqrt{S}}} + o(M^3)$ [Kwasniewki et al, SC'21]: $\boxed{V \sim 1 \cdot \frac{M^3}{\sqrt{S}}}$

With large memory

- Select the value of c to minimize the communication volume
- For **SBC**, we obtain $c \sim \sqrt[3]{P/2}$ and $r = 2c$, so that $\boxed{V \sim 3\sqrt[3]{1/2} \cdot S\sqrt[3]{P}}$
- With **2DBC**, $c \sim \sqrt[3]{P}$ and $\boxed{V \sim 3 \cdot S\sqrt[3]{P}}$: factor $\sqrt[3]{2} \simeq 1.26$ on comms and memory

2.5D version: experimental results ($c = 3$)

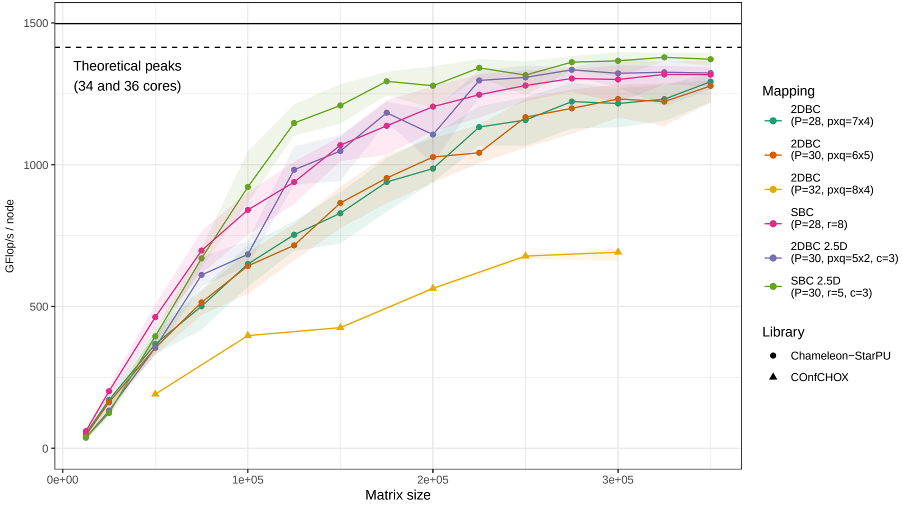


Table of Contents

Conclusions

Contributions

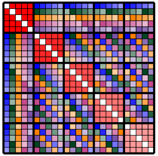
- New **Symmetric Block Cyclic** distribution, adapted for SYRK & Cholesky
- Lowers communication volume **by a factor of** $\sqrt{2}$
- Task-based **2.5D** implementation of Cholesky factorization
- Significantly **improved performance** and **scalability**
- Can be applied to many other symmetric computations

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?

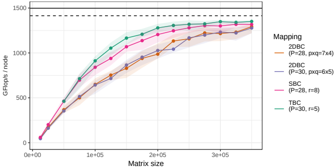
Optimal **TBC** distribution

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



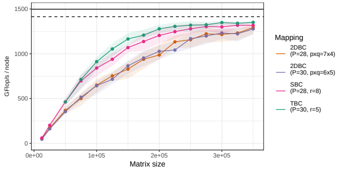
Optimal **TBC** distribution

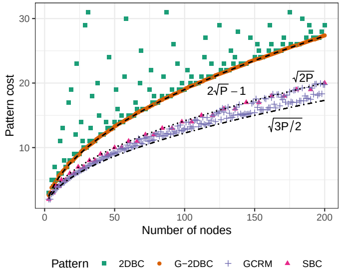
- Based on the **TBS** sequential algorithm from [SPAA'2022]
- Used in the context of the **SYMM** operation
- Also improves the performance of **Cholesky**



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/