Data Localisation for Distributed Applications

Compressed Cholesky Case Study

Olivier Beaumont
Lionel Eyraud-Dubois
Mathieu Vérité

INRIA Bordeaux - HiePACS Team

Solharis Project Kickoff Meeting
February 5th, 2020

Inria

1. Problem

1.1 - Data Localisation

Distributed application: static data allocation to computation nodes

Objectives

- completion time
- result reliability
- execution replicability

Issues

- load balancing
- communication management

Strategies

- optimise static allocation
- on the run re-allocation

Case study: Cholesky decomposition

1.2 - Problem Description

Formal problem:

- $\mathcal{P}$: set of P computation **resources** / **processes**
- tasks **dependencies**: application DAG
- for each task: known constant proportionality **input "size"** $\leftrightarrow$ **task execution time**

Objective

minimise *makespan*

Methodology

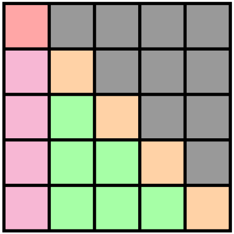
- solve approximate problem: *load balancing*
- evaluate *makespan* in execution

1.3 - Cholesky Decomposition

- Physical problems modeling: $\mathbf{A} * x = \mathbf{b}$
  $\mathbf{A} \in \mathbb{R}^{n \times n}$ typically **symetric positive definite**
- Steps for resolution: $\mathbf{A} = \mathbf{L} * \mathbf{L}^t \rightarrow \mathbf{L} * y = \mathbf{B} \rightarrow \mathbf{L}^t * x = y$

Right-looking algorithm:
$\mathcal{O}(N^3)$ complexity

- **Input** : $\mathbf{A}$
- **Initialisation** : $\mathbf{L} = \mathbf{A}_{\mathrm{tri.inf.}}$
- **For** $k = 1 \rightarrow N$: $\mathtt{POTRF}(k)$
  - **For** $i = k + 1 \rightarrow N$: $\mathtt{TRSM}(i, k)$
  - **For** $j = k + 1 \rightarrow N$: $\mathtt{SYRK}(j, k)$
    - **For** $i = j + 1 \rightarrow N$
      $\mathtt{GEMM}(i, j, k)$
- **Output** : $\mathbf{L}$

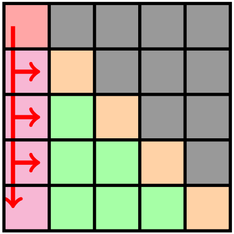


1.3 - Cholesky Decomposition

- Physical problems modeling: $\mathbf{A} * x = \mathbf{b}$
  $\mathbf{A} \in \mathbb{R}^{n \times n}$ typically **symetric positive definite**
- Steps for resolution: $\mathbf{A} = \mathbf{L} * \mathbf{L}^t \rightarrow \mathbf{L} * y = \mathbf{B} \rightarrow \mathbf{L}^t * x = y$

Right-looking algorithm:
$\mathcal{O}(N^3)$ complexity

- **Input** : $\mathbf{A}$
- **Initialisation** : $\mathbf{L} = \mathbf{A}_{\mathrm{tri.inf.}}$
- **For** $k = 1 \rightarrow N$: POTRF(k)
  - **For** $i = k + 1 \rightarrow N$: TRSM(i, k)
  - **For** $j = k + 1 \rightarrow N$: SYRK(j, k)
    - **For** $i = j + 1 \rightarrow N$
      GEMM(i, j, k)
- **Output** : $\mathbf{L}$



Column broadcast

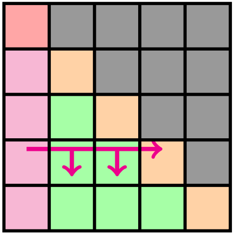
$(k = 1)$

1.3 - Cholesky Decomposition

- Physical problems modeling: $\mathbf{A} * x = \mathbf{b}$
  $\mathbf{A} \in \mathbb{R}^{n \times n}$ typically **symetric positive definite**
- Steps for resolution: $\mathbf{A} = \mathbf{L} * \mathbf{L}^t \rightarrow \mathbf{L} * y = \mathbf{B} \rightarrow \mathbf{L}^t * x = y$

Right-looking algorithm: $\mathcal{O}(N^3)$ complexity

- **Input** : $\mathbf{A}$
- **Initialisation** : $\mathbf{L} = \mathbf{A}_{\mathrm{tri.inf.}}$
- **For** $k = 1 \rightarrow N$: $\mathtt{POTRF}(k)$
  - **For** $i = k+1 \rightarrow N$: $\mathtt{TRSM}(i, k)$
  - **For** $j = k+1 \rightarrow N$: $\mathtt{SYRK}(j, k)$
    - **For** $i = j+1 \rightarrow N$
      $\mathtt{GEMM}(i, j, k)$
- **Output** : $\mathbf{L}$



Row broadcast

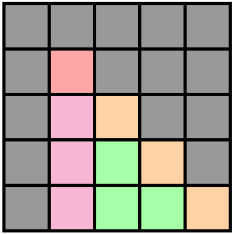
$(k = 1)$

1.3 - Cholesky Decomposition

- Physical problems modeling: $\mathbf{A} * x = \mathbf{b}$
  $\mathbf{A} \in \mathbb{R}^{n \times n}$ typically **symetric positive definite**
- Steps for resolution: $\mathbf{A} = \mathbf{L} * \mathbf{L}^t \rightarrow \mathbf{L} * y = \mathbf{B} \rightarrow \mathbf{L}^t * x = y$

Right-looking algorithm: $\mathcal{O}(N^3)$ complexity

- **Input** : $\mathbf{A}$
- **Initialisation** : $\mathbf{L} = \mathbf{A}_{\text{tri.inf.}}$
- **For** $k = 1 \rightarrow N$: `POTRF(k)`
  - **For** $i = k + 1 \rightarrow N$: `TRSM(i, k)`
  - **For** $j = k + 1 \rightarrow N$: `SYRK(j, k)`
    - **For** $i = j + 1 \rightarrow N$
      `GEMM(i, j, k)`
- **Output** : $\mathbf{L}$



Next iteration

$(k = 2)$

1.4 - Working Assumptions

Communications scheme:

- data transfer on same position
- broadcast on same row / column

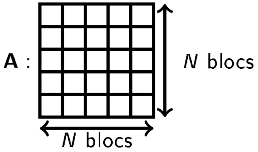
Specific solutions

- unmodified association: **tile** $\leftrightarrow$ **proc.**
- max. number of different proc. on each row/column: m^{row}, m^{col}

Assumption

- **limited communication** $\Rightarrow$ simultaneous with calculation

1.5 - Input Data Handling (1/2): Compression

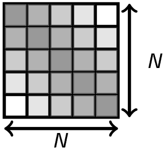


Tiled matrix

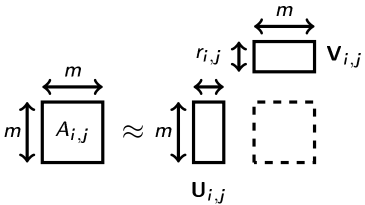
Density distribution

"Pseudo-rank" : $r_{i,j}$

Density: $d(A_{i,j}) = \frac{r_{i,j}}{m}$

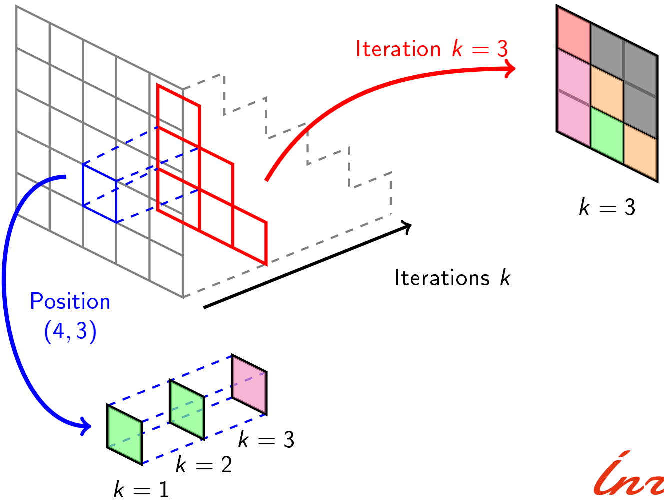


Block Low Rank compression



1.5 - Input Data Handling (2/2): Modification

Allocation: tile ↔ proc.



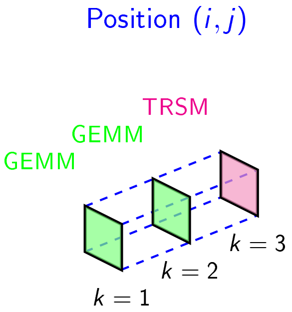
1.5 - Input Data Handling (2/2): Modification

Allocation: tile $\leftrightarrow$ proc.

Weighted sum over all iterations

$$\bar{\mathbf{A}}_{i,j} = d(A_{i,j}) \times \sum_{k=1}^{j} \mathrm{task}[i,j,k]$$

$\Rightarrow$ aggregated matrix $\bar{\mathbf{A}}$



2. Resolution Methods

2.1 - Problem Simplification

Assumptions

set of **independent tasks**

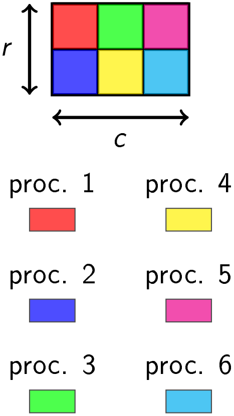
Objective

$$\min\{makespan\} \Leftrightarrow \min\{\max_{p \in \mathcal{P}}\{load_p\}\}$$

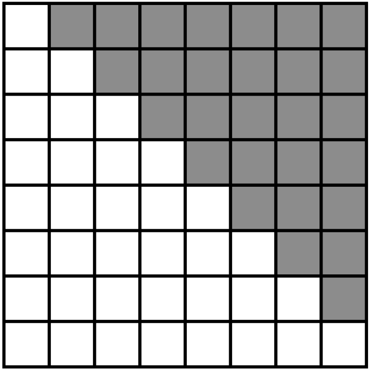
$\rightarrow$ *load balancing* problem

2.2 - Block Cyclic Method

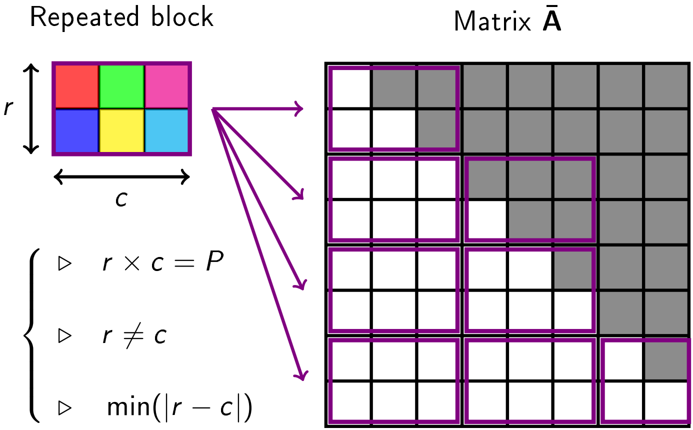
Repeated block



Matrix $\bar{\mathbf{A}}$



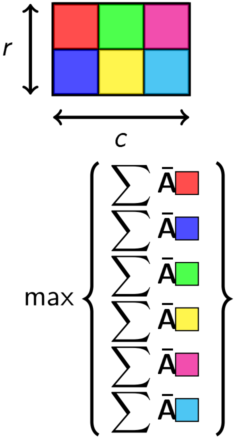
2.2 - Block Cyclic Method



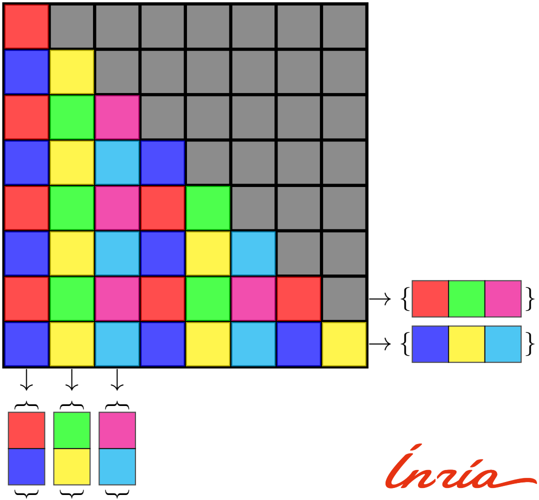
- $r \times c = P$
- $r \neq c$
- $\min(|r - c|)$

2.2 - Block Cyclic Method

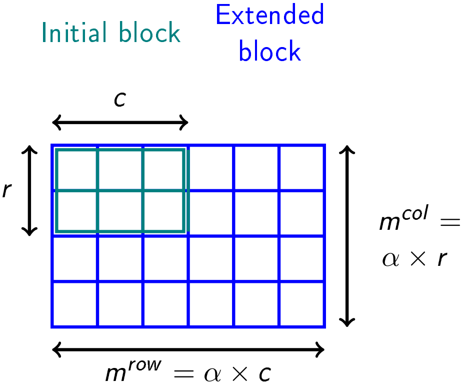
Repeated block



Matrix $\bar{\mathbf{A}}$



2.3 - Extended Block Cyclic (1/3)



$\alpha^2 \times$ more positions

Communication constraints

$\Leftrightarrow$ **parameter**: $\alpha \in [1; +\infty[$

2.3 - Extended Block Cyclic (2/3)

W:

18^{th}	5^{th}	12^{th}	4^{th}	20^{th}	11^{th}
17^{th}	24^{th}	22^{nd}	19^{th}	1^{st}	7^{th}
10^{th}	6^{th}	23^{rd}	15^{th}	2^{nd}	16^{th}
9^{th}	3^{rd}	8^{th}	13^{th}	21^{st}	14^{th}

m^{col}

m^{row}

Greedy procedure

- tiles: $\searrow$ load

$$W_{i,j} = \sum_{\substack{u = m^{col} \times i \in [\![1;N]\!] \\ v = m^{row} \times j \in [\![1;N]\!]}} \bar{A}_{u,v}$$

2.3 - Extended Block Cyclic (2/3)

W:

18$^{\text{th}}$	5$^{\text{th}}$	12$^{\text{th}}$	4$^{\text{th}}$	20$^{\text{th}}$	11$^{\text{th}}$
17$^{\text{th}}$	24$^{\text{th}}$	22$^{\text{nd}}$	19$^{\text{th}}$	1$^{\text{st}}$	7$^{\text{th}}$
10$^{\text{th}}$	6$^{\text{th}}$	23$^{\text{rd}}$	15$^{\text{th}}$	2$^{\text{nd}}$	16$^{\text{th}}$
9$^{\text{th}}$	3$^{\text{rd}}$	8$^{\text{th}}$	13$^{\text{th}}$	21$^{\text{st}}$	14$^{\text{th}}$

m^{col}

m^{row}

Greedy procedure

- tiles: $\searrow$ load
- alloc. least loaded

$$W_{i,j} = \sum_{\substack{u=m^{col}\times i\in[\![1;N]\!] \\ v=m^{row}\times j\in[\![1;N]\!]}} \bar{A}_{u,v}$$

2.3 - Extended Block Cyclic (2/3)

W:

18^{th}	5^{th}	12^{th}	4^{th}	20^{th}	11^{th}
17^{th}	24^{th}	22^{nd}	19^{th}	1^{st}	7^{th}
10^{th}	6^{th}	23^{rd}	15^{th}	2^{nd}	16^{th}
9^{th}	3^{rd}	8^{th}	13^{th}	21^{st}	14^{th}

m^{col}

m^{row}

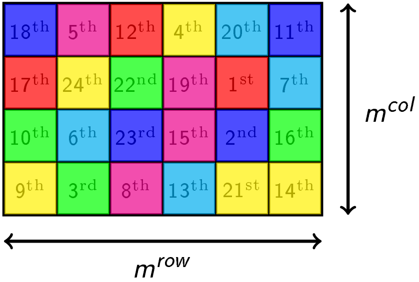
Greedy procedure

- tiles: ↘ load
- alloc. least loaded

$$W_{i,j} = \sum_{\substack{u=m^{col}\times i\in[\![1;N]\!] \\ v=m^{row}\times j\in[\![1;N]\!]}} \bar{A}_{u,v}$$

2.3 - Extended Block Cyclic (2/3)

W:



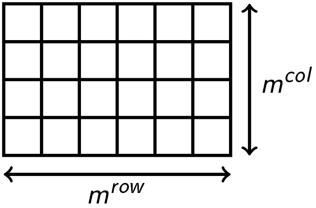
Greedy procedure

- tiles: $\searrow$ load
- alloc. least loaded

$$W_{i,j} = \sum_{\substack{u=m^{col}\times i\in[\![1;N]\!]\\ v=m^{row}\times j\in[\![1;N]\!]}} \bar{A}_{u,v}$$

2.3 - Extended Block Cyclic (3/3)

W:



$$W_{i,j} = \sum_{\substack{u=m^{col}\times i\in[\![1;N]\!]\\ v=m^{row}\times j\in[\![1;N]\!]}} \bar{A}_{u,v}$$

Decision variables:

$$\left[\begin{array}{l} \mathbf{x}_{i,j}^{(p)} = \begin{cases} 1 & \text{if proc. } p \text{ on } (i,j) \\ 0 & \text{otherwise} \end{cases} \\ \mathbf{w} = \text{maximum working time among all processes} \end{array}\right.$$

2.3 - Extended Block Cyclic (3/3)

Integer linear program:

$$\min\{\mathbf{w}\}$$

$$\begin{cases} (1) \quad \forall p \in [\![1;P]\!] & \displaystyle\sum_{\substack{i \in [\![1;m^{col}]\!] \\ j \in [\![1;m^{row}]\!]}} \mathbf{x}_{i,j}^{(p)} \times W_{i,j} \leqslant \mathbf{w} \quad \text{max. time} \\ (2) \quad \forall (i,j) \in [\![1;m^{col}]\!] \times [\![1;m^{row}]\!] & \displaystyle\sum_{p \in [\![1;P]\!]} \mathbf{x}_{i,j}^{(p)} = 1 \quad \text{allocation} \end{cases}$$

Inria

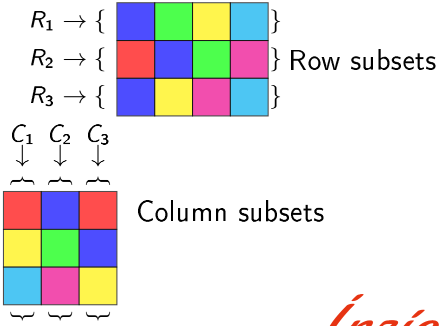
2.4 - Random Subsets Methods (1/3)

Idea: randomly generate subsets of proc. for rows $(R_1, ..., R_Q)$ / columns $(C_1, ..., C_Q)$

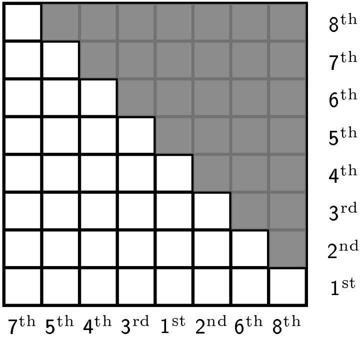
- limited number of proc.: m^{row}; m^{col}
- each (R_i, C_j) pair **compatible** : $\mathrm{Card}(I_{i,j} = R_i \bigcap C_j) \geqslant K$

Advantages

- **independent** of N
- managing sampling difficulty: Q, K
- degree of freedom for tile allocation: $\mathrm{Card}(I)$



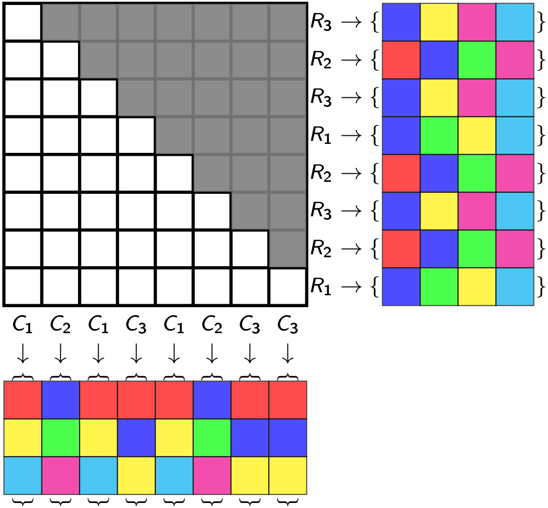
2.4 - Random Subsets Methods (2/3): Two Steps



Step 1

- rows / columns:
  ↘ sum load

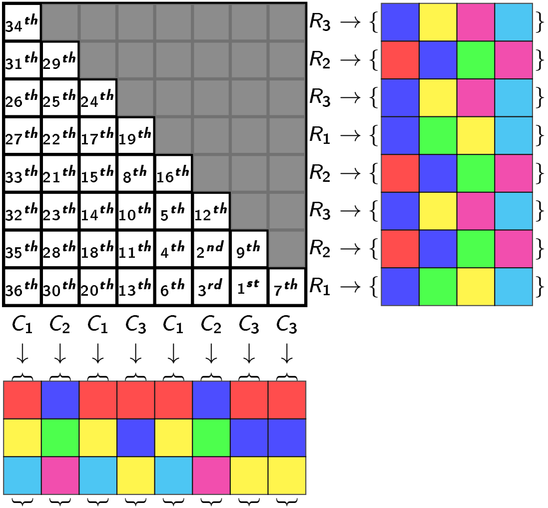
2.4 - Random Subsets Methods (2/3): Two Steps



Step 1

- rows / columns: ↘ sum load
- alloc. least loaded subset

2.4 - Random Subsets Methods (2/3): Two Steps



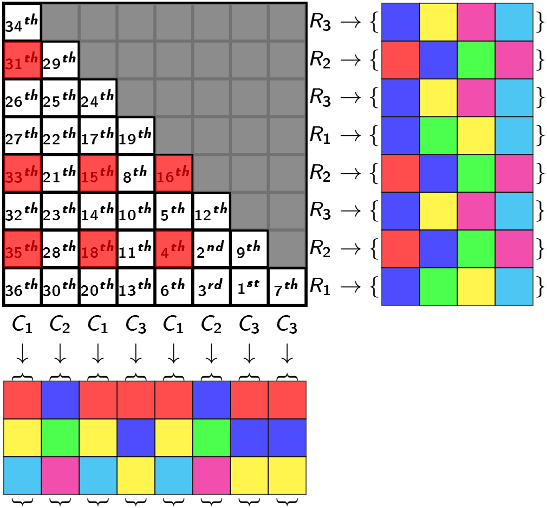
Step 1

- rows / columns: ↘ sum load
- alloc. least loaded subset

Step 2

- tiles: ↘ load

2.4 - Random Subsets Methods (2/3): Two Steps



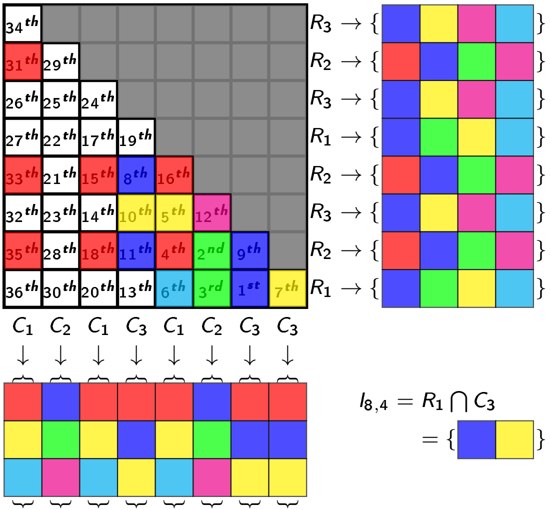
Step 1

- rows / columns: ↘ sum load
- alloc. least loaded subset

Step 2

- tiles: ↘ load
- alloc *no choice* positions

2.4 - Random Subsets Methods (2/3): Two Steps



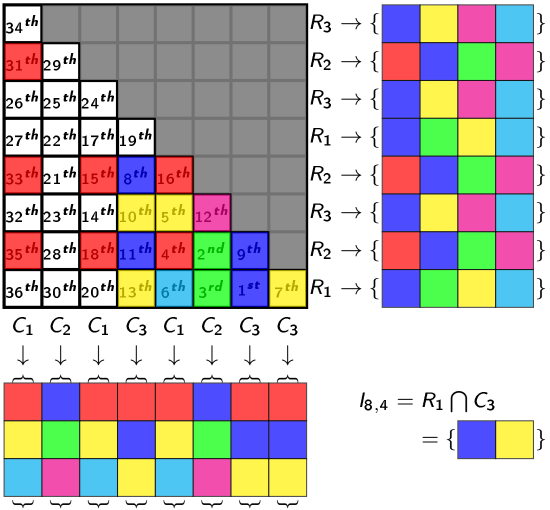
Step 1

- rows / columns: ↘ sum load
- alloc. least loaded subset

Step 2

- tiles: ↘ load
- alloc *no choice* positions
- available proc.: $I_{i,j} = R_i \bigcap C_j$

2.4 - Random Subsets Methods (2/3): Two Steps



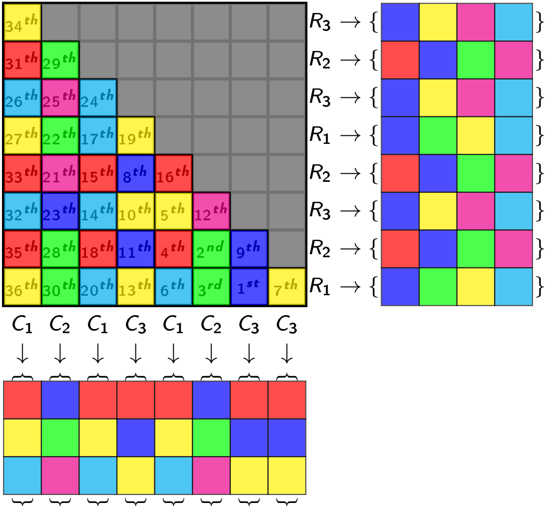
Step 1

- rows / columns: ↘ sum load
- alloc. least loaded subset

Step 2

- tiles: ↘ load
- alloc *no choice* positions
- available proc.: $I_{i,j} = R_i \bigcap C_j$
- alloc. least loaded proc.

2.4 - Random Subsets Methods (2/3): Two Steps



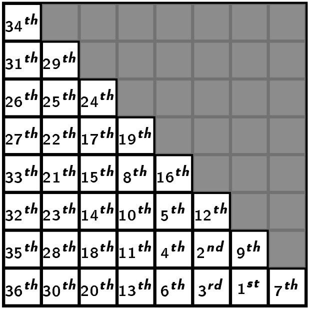
Step 1

- rows / columns: $\searrow$ sum load
- alloc. least loaded subset

Step 2

- tiles: $\searrow$ load
- alloc *no choice* positions
- available proc.: $I_{i,j} = R_i \bigcap C_j$
- alloc. least loaded proc.

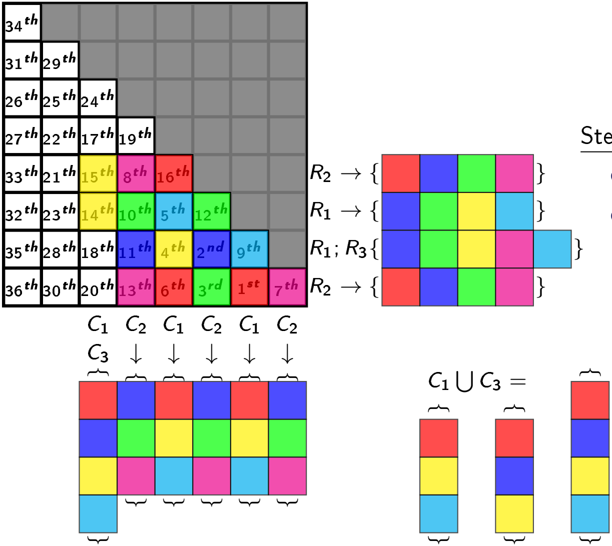
2.4 - Random Subsets Methods (3/3): Direct



Steps

- tiles: ↘ load

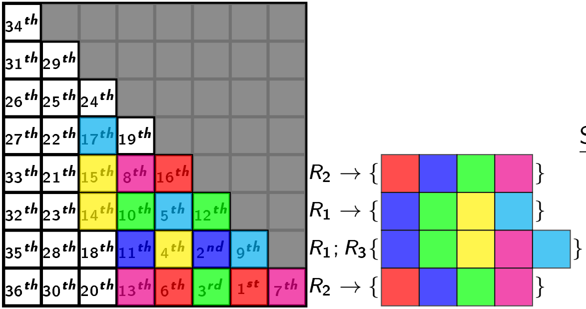
2.4 - Random Subsets Methods (3/3): Direct

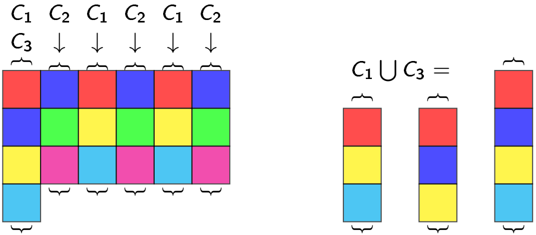


Steps

- tiles: ↘ load
- available proc.

2.4 - Random Subsets Methods (3/3): Direct

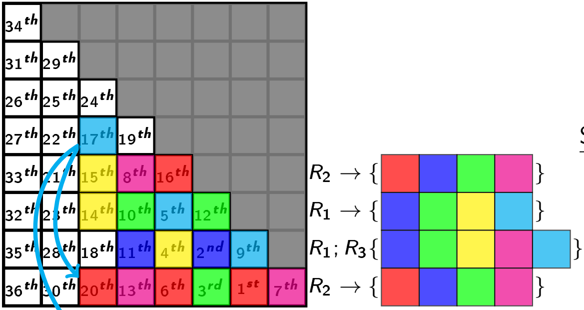


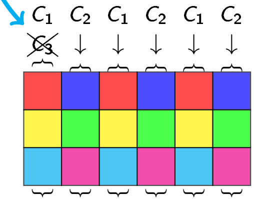


Steps

- tiles: ↘ load
- available proc.
- alloc. least loaded

2.4 - Random Subsets Methods (3/3): Direct

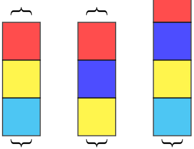




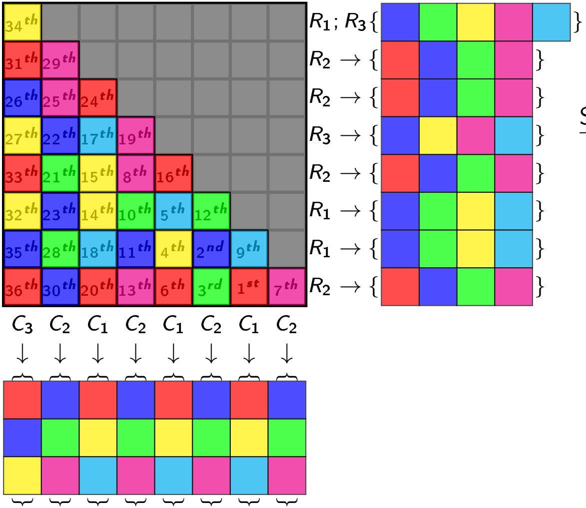
Steps

- tiles: ↘ load
- available proc.
- alloc. least loaded
  ⇒ subsets
  ⇒ *no choice*

$C_1 \bigcup C_3 =$



2.4 - Random Subsets Methods (3/3): Direct

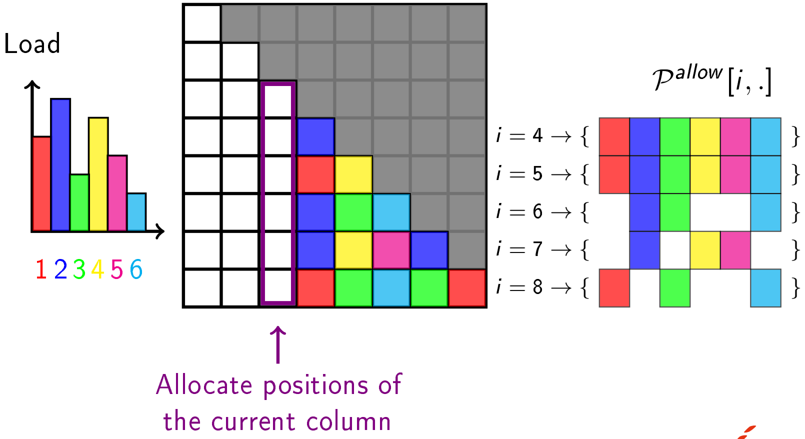


Steps

- tiles: ↘ load
- available proc.
- alloc. least loaded
  ⇒ subsets
  ⇒ *no choice*

2.5 - By-Column Greedy Algorithm

Column allocation using integer linear program:



2.6 - Load Balancing: Some Results

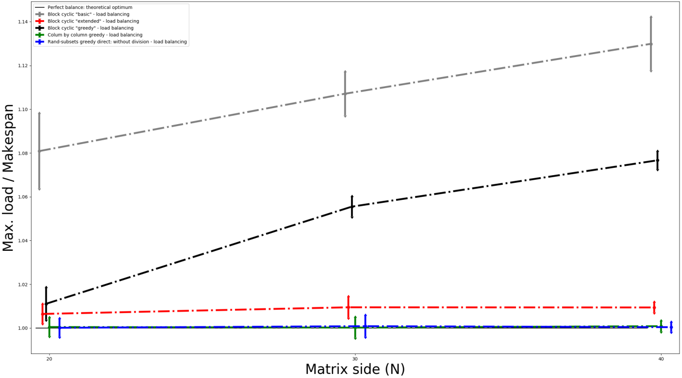


Figure: Maximum load VS input matrix size
($\alpha = 2$; $N = 20$ to 40; $\frac{N^2}{P} \approx$ cste)

Inria

2.6 - Load Balancing: Some Results

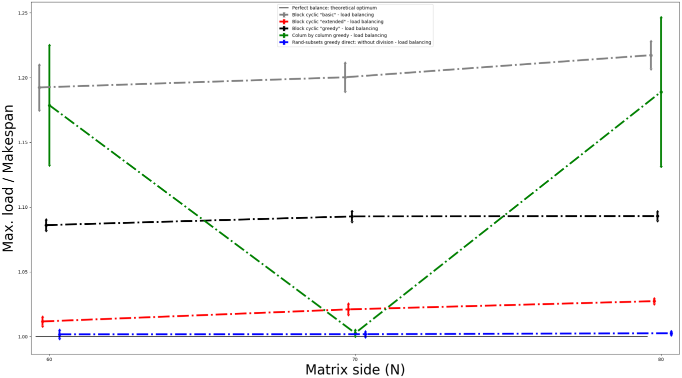


Figure: Maximum load VS input matrix size
($\alpha = 2$; $N = 60$ to 80; $\frac{N^2}{P} \approx$ cste)

3. Evaluation

3.1 - Simulated Execution

Assumption

taking into account tasks dependencies

Simulate execution: task based scheduler

- DAG + tile allocation
- prioritised queues of ready tasks
- execution at proc. scale ⇒ **preemption** allowed
- **no communication**

3.2 - Makespan: Some Results

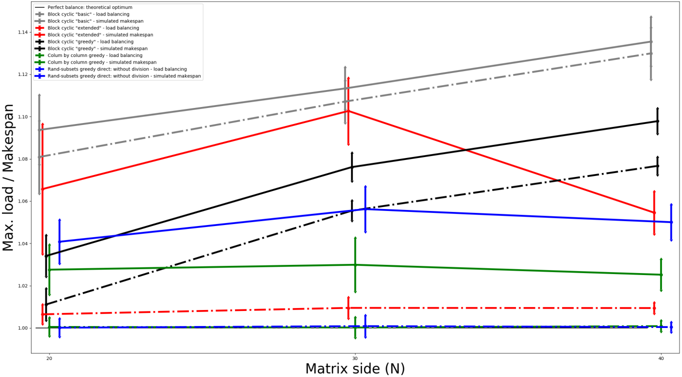


Figure: Makespan VS input matrix size
($\alpha = 2$; $N = 20$ to 40; $\frac{N^2}{P} \approx$ cste)

3.2 - Makespan: Some Results

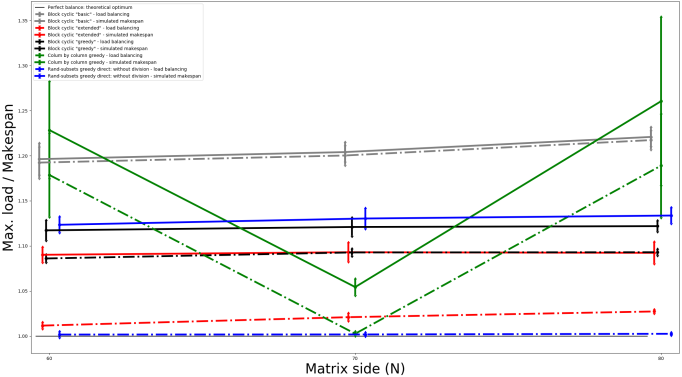
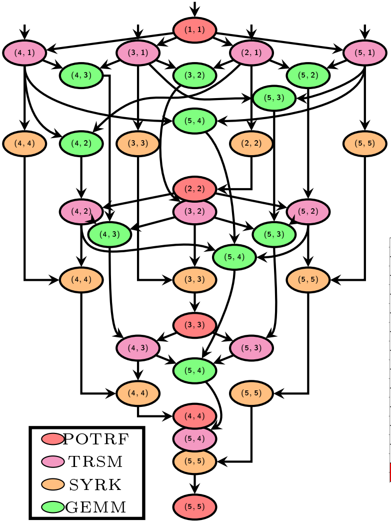


Figure: Makespan VS input matrix size

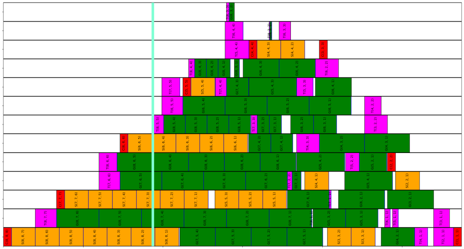
$(\alpha = 2;\ N = 60 \text{ to } 80;\ \frac{N^2}{P} \approx \text{cste})$

3.3 - Random Subsets: Improved Version (1/2)



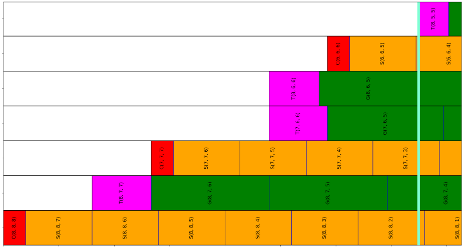
- start from end
- unlimited number of proc.
- As Last As Possible

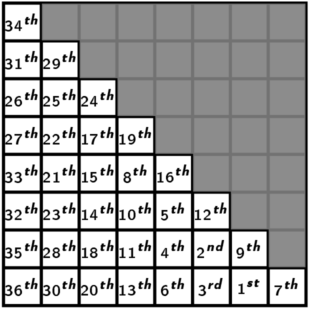
⇒ time threshold:
first use of unavailable proc.



3.3 - Random Subsets: Improved Version (2/2)

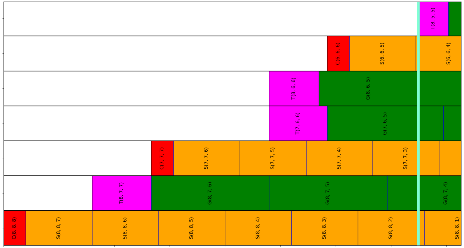
Tasks before / after threshold
→ tiles splitting + reordering

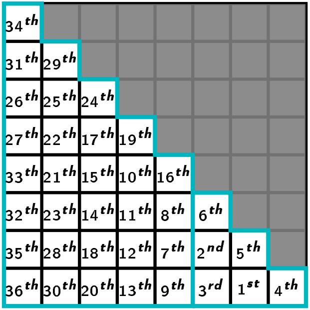




3.3 - Random Subsets: Improved Version (2/2)

Tasks before / after threshold
→ tiles splitting + reordering





splitting + reordering

3.4 - Additional Results

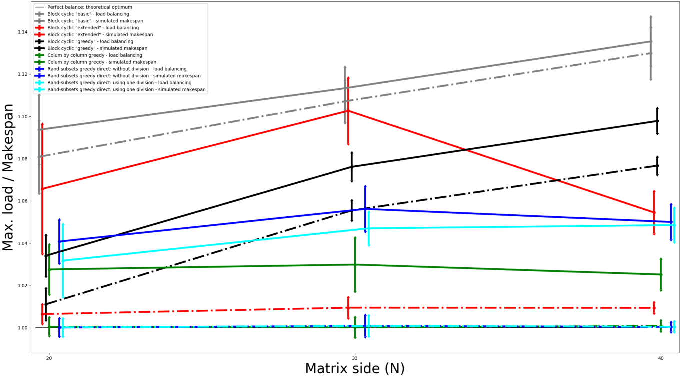


Figure: Makespan VS input matrix size

($\alpha = 2$; $N = 20$ to 40; $\frac{N^2}{P} \approx$ cste)

3.4 - Additional Results

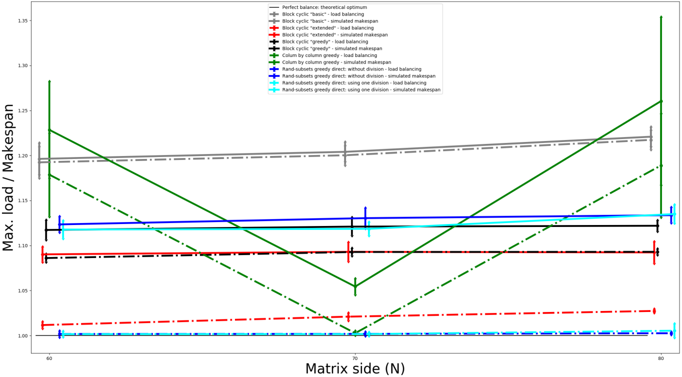


Figure: Makespan VS input matrix size
($\alpha = 2$; $N = 60$ to 80; $\frac{N^2}{P} \approx$ cste)

4. Perspectives

So far

- efficient methods for load balancing
- many options for strategies

So far

- efficient methods for load balancing
- many options for strategies

Future work

- secure results: larger scale, parameters sets, real data
- improve strategies
- explore new ones: hybrid, relaxed constraints
- dig in scheduling aspect
- other applications / use cases

So far

- efficient methods for load balancing
- many options for strategies

Future work

- secure results: larger scale, parameters sets, real data
- improve strategies
- explore new ones: hybrid, relaxed constraints
- dig in scheduling aspect
- other applications / use cases

Tools improvement

- evaluation: introduce communications

Thank you