2D Static Resource Allocation for Compressed Linear Algebra and Communication Constraints

Olivier Beaumont
Lionel Eyraud-Dubois
Mathieu Vérité

INRIA Bordeaux Sud Ouest - Université de Bordeaux

SOLHARIS & HPC Scalable Ecosystems
December 7th - 8th, 2020

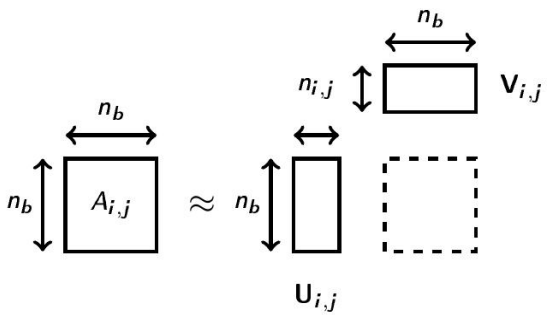
1.1 - Context

Data

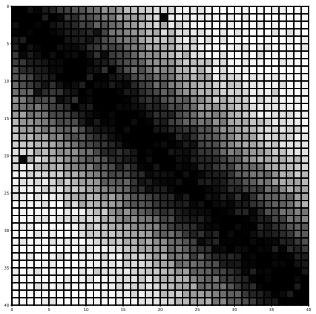
- large linear systems: $\mathbf{A} * x = \mathbf{b}$
- tiled data storage
- *block low rank* compression
  $\Rightarrow$ heterogeneous density

Computation

- distributed memory:
  data chunk (tile) $\leftrightarrow$ resource
- owner compute tasks allocation
- *run-time* tasks scheduler



(a) BLR approximation



(b) Rank dispersion (dark = dense)

Objective: minimize global *makespan*

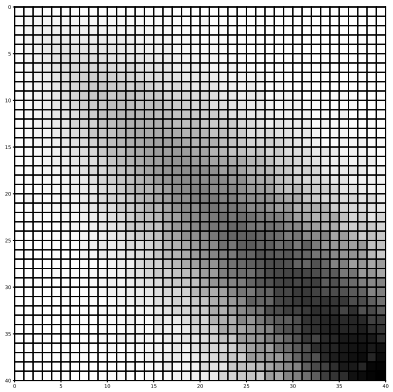
- ensure load balancing
- reduce communications

$\Rightarrow$ **Trade-off**:
load balancing VS limited transferts

1.2 - Formal Problem

Input

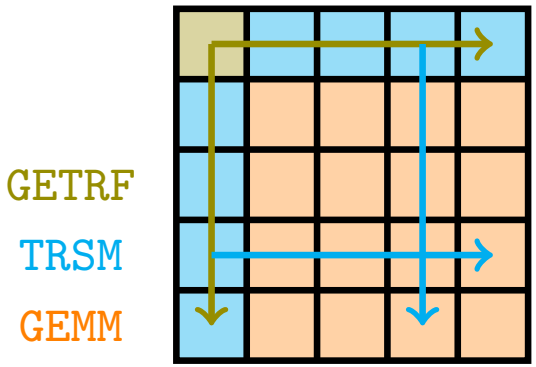
- P homogeneous computation resources/processors
- linear algebra kernel: LU, matrix multiplication
  $\Rightarrow$ set of tasks on tile (i, j)
- $N \times N$ tiled matrix of workloads W:
  $W_{i,j} = \text{density of tile } (i,j) * \text{cost of all tasks on tile } (i,j)$
- assumption: density remains constant along execution
- $\text{cost of all tasks on tile} \Leftrightarrow$ kernel execution time by a single processor on a full rank tile



Cumulated workloads (LU)

Constraint

- communications rely on *row* or *column broadcasts* (one to many).
- volume of communication $\propto \text{density} * \text{nb receivers}$
- to limit data transfers $\Rightarrow$ reduce number of different processors on each row/column



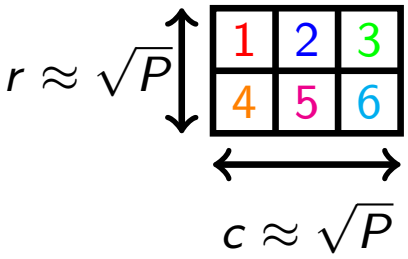
Communication scheme (LU)

2.1 - Homogeneous Case

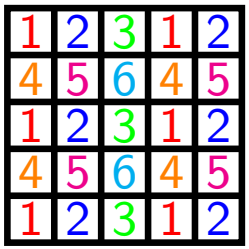
If no compression (i.e. homogeneous ranks) $\Rightarrow$ identical workloads

State-of-the-art method: 2D block cyclic

1. create $r \times c$ block pattern such that: $rc = P$; $r \neq c$
2. each position associated to a processor in a round-robin fashion
3. pattern repeated over the whole matrix to produce the mapping **M**



(a) 2D block pattern ($P = 6$)



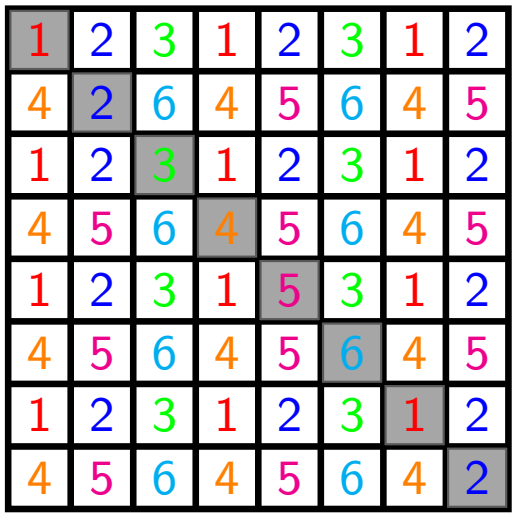
(b) Mapping: repeated *block cyclic* pattern ($N = 5$)

- optimal overall load balancing
- minimum number of processors involved in broadcast ($\approx \sqrt{P}$)
- smooth workload for processors along execution

2.2 - Related Work: Block Cyclic Hybrid

Several application fields:

linear systems featuring specific characteristics ⇒ variations of *2D block cyclic*



1-block cyclic hybrid
($N = 8$, $P = 6$)

Block Cyclic Hybrid method

Handle specifically diagonal tiles:

- vertical round-robin allocation of x-wide diagonal band (*x-block cyclic hybrid*)
- regular *2D block cyclic* allocation elsewhere

3.1 - Strategies and Evaluation

Feasible solution: tile to processor *mapping*: $\mathbf{M}_{i,j} = p \in [\![1; P]\!]$

Constraint:

- **no more than** $\alpha\sqrt{P}$ different processors on any row/column
- $\alpha \geqslant 1$: tightness of constraint $\Rightarrow$ number of data transfers (broadcast receivers)
  ($\alpha = 1 \Leftrightarrow$ *2D block cyclic* method)

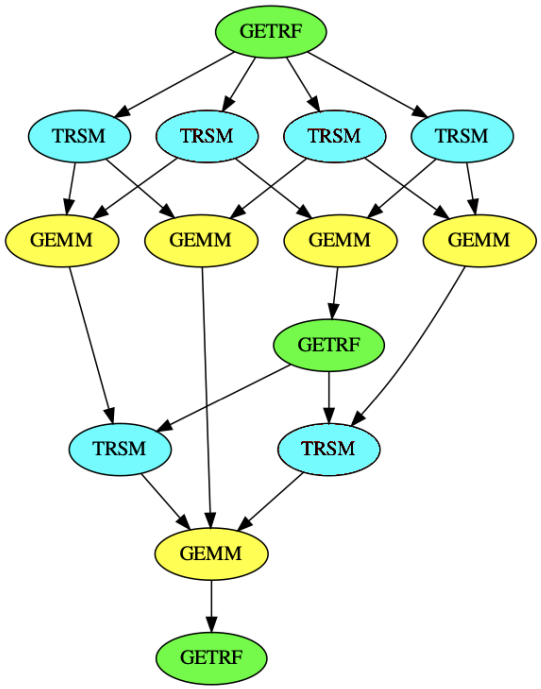
Objective function: maximum load

$$\max_{p \in \{1,\ldots,P\}} \left(\sum_{\substack{(i,j) \in [\![1;N]\!]^2 \\ \mathbf{M}_{i,j}=p}} W_{i,j} \right)$$

$\Leftrightarrow$ *makespan* assuming *no task dependency*

Evaluation: simulated makespan

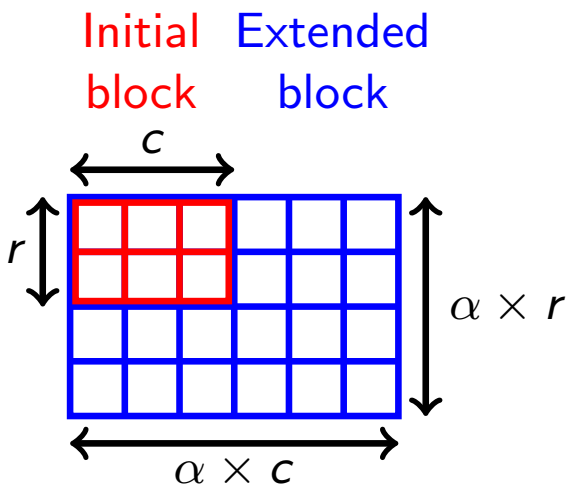
- tasks *dependencies* only (DAG)
- and *data transfer time*



DAG for 3×3 LU

3.2 - Block Cyclic Extended (1/2)

Natural adaptation of *2D block cyclic*



Extended pattern
($P = 6$, $\alpha = 2$)

Steps:

1. *2D block cyclic* pattern of dimension $r \times c$
2. consider extended $\alpha r \times \alpha c$ pattern of $\alpha^2 P$ virtual processors
3. $\tilde{\mathbf{M}}$ is the *2D block cyclic* mapping using the extended pattern
   $\Rightarrow$ virtual processors workload
4. associate virtual processors to the real ones: pattern mapping **G**
5. derive the actual mapping **M**

- pattern directly matches constraints
- allows less regular mapping $\Rightarrow$ some room for load balancing

3.2 - Block Cyclic Extended (2/2)

- double sampling steps (3)+(4): reduce loads variability
- virtual to real processors association $\Leftrightarrow$ unconstrained *bin packing* problem
  $\Rightarrow$ approximate resolution: *Best Fit Decreasing* heuristic

Algorithm 1: *Best Fit Decreasing*

input : P, W

Load for each processor: $L_p = 0,\ \forall p \in \{1, ..., P\}$
for $(i, j) \in [\![1; \alpha r]\!] \times [\![1; \alpha c]\!]$ considered in decreasing order of $W_{i,j}$ **do**
  find least loaded processor: $p_{\text{least}} = \underset{p \in [\![1;P]\!]}{\text{argmin}}(L_p)$
  allocate to processor p_{least} to position (i, j): $\mathbf{G}_{i,j} = p_{\text{least}}$
  update the load of processor p_{least}: $L_{p_{\text{least}}} \leftarrow L_{p_{\text{least}}} + W_{i,j}$

output: $\mathbf{G}$, L

3.3 - Random Subsets (1/2)

Rationale

- allow more general mappings (not pattern-based)
- ensure solution feasibility

Principle

- randomly generate subsets of processors for rows $(\mathcal{R}_1, ..., \mathcal{R}_Q)$; columns $(\mathcal{C}_1, ..., \mathcal{C}_Q)$
- each having $\alpha\sqrt{P}$ processors
- A pair $(\mathcal{R}_q, \mathcal{C}_s)_{(q,s)\in[\![1;Q]\!]^2}$ is **compatible** if $\mathrm{Card}(\mathcal{R}_q \bigcap \mathcal{C}_s) \geqslant 1$

Advantages

- **independent** of N
- managing sampling difficulty: larger $Q \Rightarrow$ longer sampling process

$$\mathcal{C}_1 = \begin{Bmatrix} 1 \\ 4 \\ 6 \end{Bmatrix} ; \mathcal{C}_2 = \begin{Bmatrix} 2 \\ 3 \\ 5 \end{Bmatrix} ; \mathcal{C}_3 = \begin{Bmatrix} 1 \\ 2 \\ 4 \end{Bmatrix} \qquad \begin{matrix} \mathcal{R}_1 = \{2 & 3 & 4 & 6\} \\ \mathcal{R}_2 = \{1 & 2 & 3 & 5\} \\ \mathcal{R}_3 = \{2 & 4 & 5 & 6\} \end{matrix}$$

Example of random subsets ($P = 6$, $Q = 3$)

3.3 - Random Subsets (2/2)

Procedure

At each step one tile (i, j) is considered:

- one or several subset(s) $\mathcal{R}_{q_1,\ldots,q_m}$ (resp. $\mathcal{C}_{s_1,\ldots,s_n}$) associated to row i (resp. column j)
- candidate processor(s) for allocation at (i, j): $(\bigcup \mathcal{R}_{q_1,\ldots,q_m}) \bigcap (\bigcup \mathcal{C}_{s_1,\ldots,s_n})$
- all $(\mathcal{R}_q, \mathcal{C}_s)$ **compatible** $\Rightarrow$ always at least one candidate processor

Allocation rules: close to *Best Fit Decreasing*

- consider tiles according to decreasing $W_{i,j}$
- tile with only one candidate processor left are considered first
- allocate the least loaded among candidate processors

4.1 - Test Case: Synthetic data

Rank dispersion to mimic realistic case

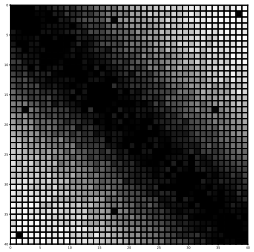
- diagonal tiles: full rank
- decreasing rank with distance to diagonal: δ parameter
- random *abnormal* tiles: full rank off diagonal

Sampling density values: tile (i, j) density
$d_{i,j} = \max(\min(v(i,j) + \gamma, 1), 0)$

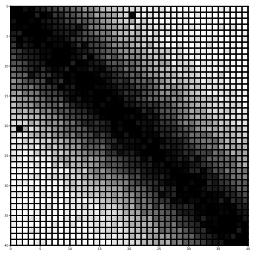
$$\begin{cases} \text{baseline value} & v(i,j) = \exp^{-\frac{\delta}{2} * (\frac{i-j}{N-1})^2} \\ \text{random "noise"} & \gamma \sim \mathcal{N}(0, \frac{1}{20}) \end{cases}$$

Anomalies:

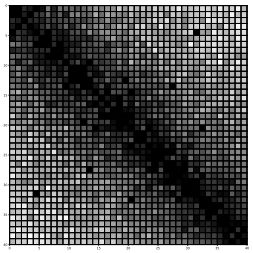
- $\eta \sim \mathcal{N}(\sqrt{N}, \frac{\sqrt{N}}{2})$ full rank tiles
- distributed uniformly at random, off-diagonal



(a) $\delta = 8$



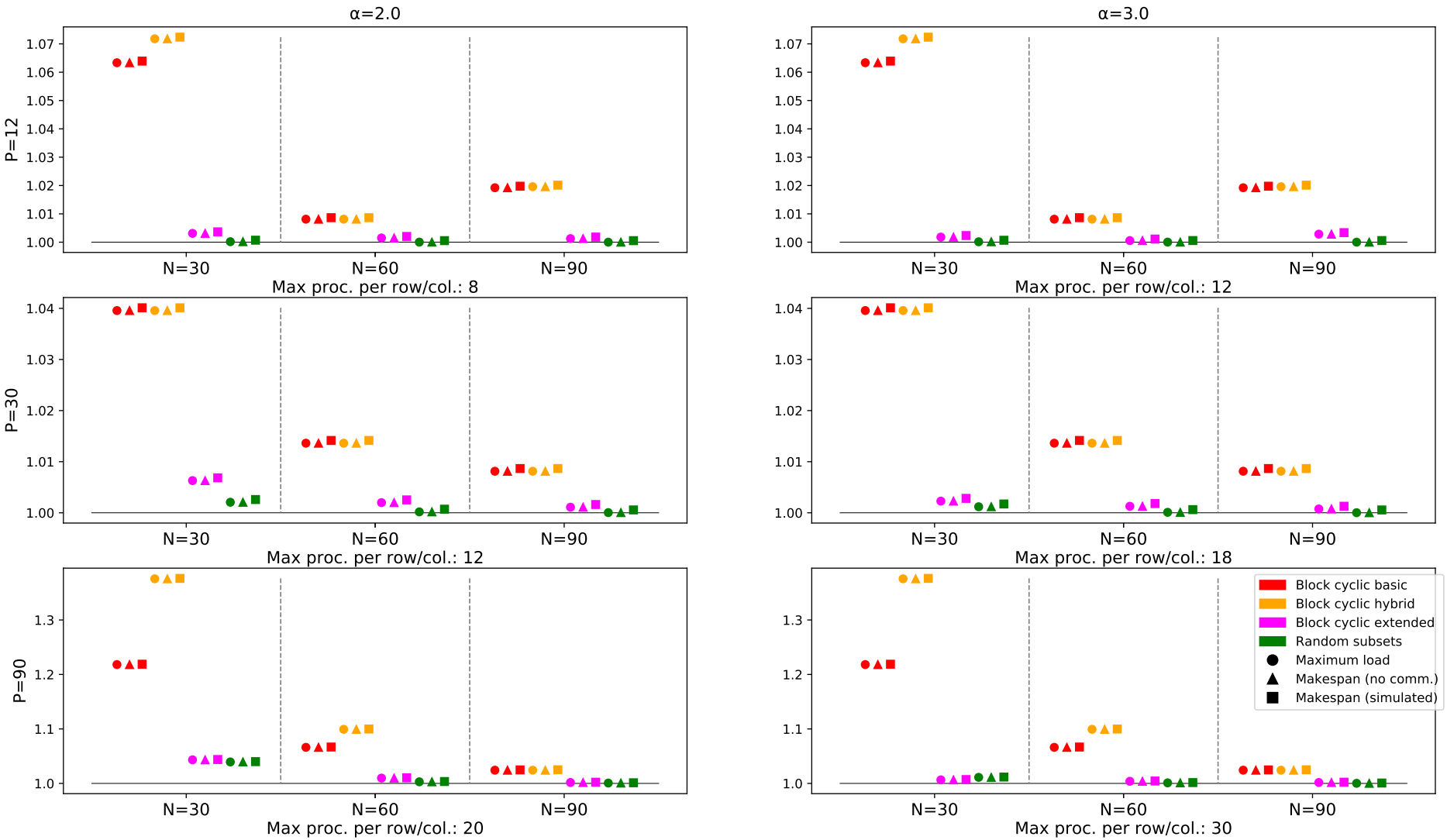
(b) $\delta = 16$



(c) $\delta = 128$

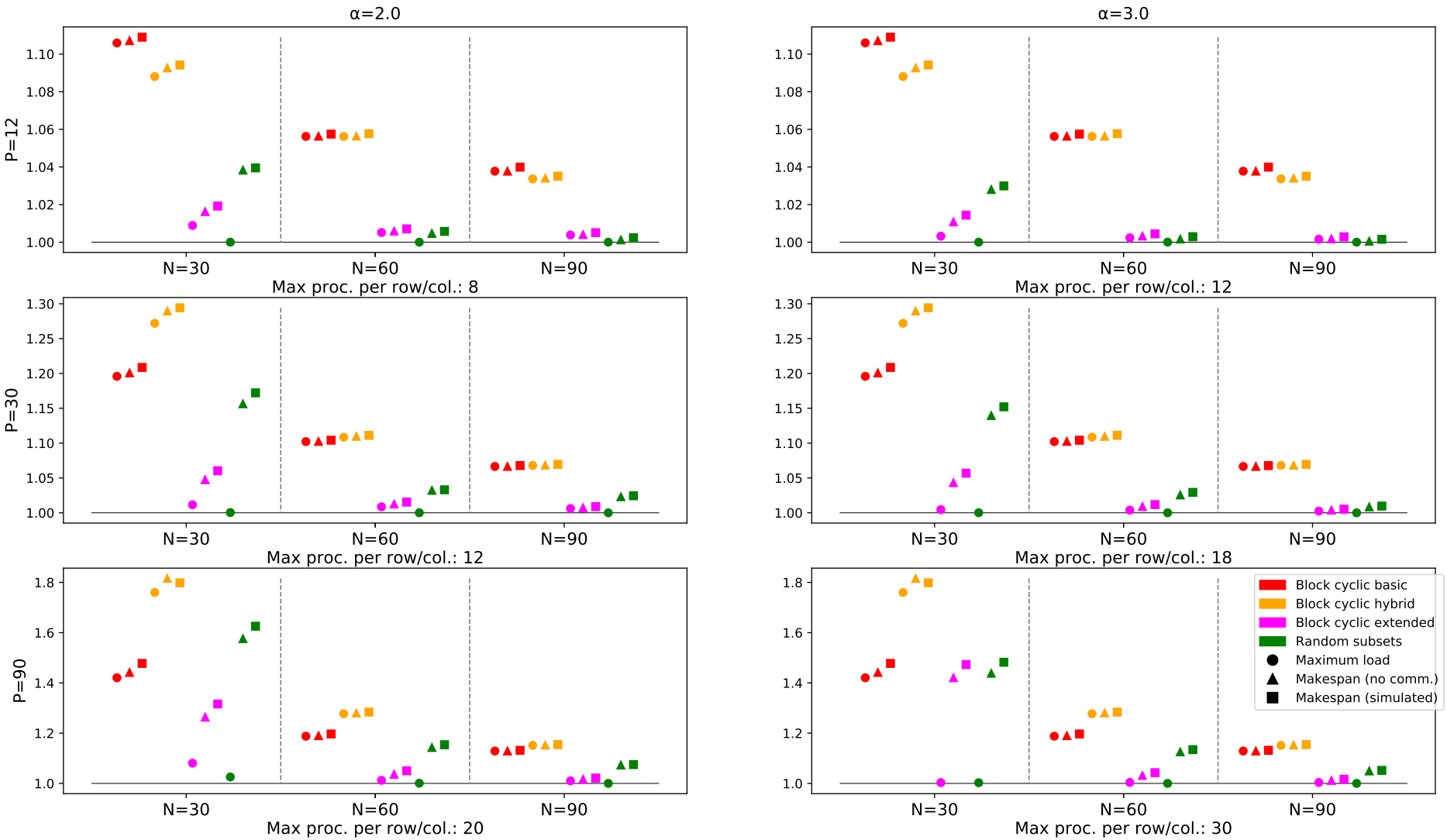
Rank ($N = 40$)

4.2 - Results (1/2)



Maximum load and makespan values, normalized with pefect load balancing;
Matrix multiplication; $\delta = 8$

4.2 - Results (2/2)



Maximum load and makespan values, normalized with pefect load balancing; LU factorization; $\delta = 8$

Conclusion and Perspectives (1/2)

Observations

- load balancing for *random subsets*, to a lesser extend for *block cyclic extended*, close to optimum in any case
- *2D block clyclic* and *hybrid* variation globally worse load balancing
- in the case of LU factorization makespan values (both with and without communications) significantly larger than maximum load for *block cyclic extended* and even larger for *random subsets*
- the gap increases with ratio $\frac{P}{N}$

Possible interpretations

- difference between maximum load and makespan value may come from mapping irregularity $\Rightarrow$ poor load balancing along execution because of task dependencies
  - *2D block cyclic* and *hybrid* variant almost perfectly regular
  - *block cyclic extended* and *random subsets* irregular
    the latter featuring none, inducing the largest gap
- pattern based methods achieve best performance regarding load balancing when using a multiple of P as block size $\Rightarrow$ selected test cases imply relative advantage for those methods over *random subsets*

Conclusion and Perspectives (2/2)

Some perspectives

- possible meta-strategy: select best method according to maximum load criteria
- repeat a pattern designed using *random subsets* to produce a complete mapping; possibility to overcome limited performance along execution

Thank you for your attention