Paralelni algoritmi Jacobijevog tipa za singularnu i generaliziranu singularnu dekompoziciju

Vedran Novaković

STFC – Daresbury Laboratory, UK
vedran.novakovic@stfc.ac.uk

Javna obrana teme doktorske disertacije
Seminar za numeričku matematiku i znanstveno računanje
Sveučilište u Zagrebu, PMF–Matematički odsjek

14. siječnja 2016.

Sadržaj

SVD and how to compute it

Given a **dense**, full-column-rank matrix G, $n \times k$, $n \geq k$, compute the singular value decomposition (**SVD**) of G,

$$G = U\Sigma V^T, \quad U, V \text{ orthogonal}, \quad \Sigma \text{ diagonal}, \sigma_i > 0.$$

What if n, k are sufficiently small (up to a thousand or two)?

- Call Lapack's `xGESVD` (bidiagonalization + implicit QR), or
- `xGEJSV` by Drmač (Jacobi SVD, for high relative accuracy).

And if they reach tens of thousands???

- Go **parallel**, due to both time and space constraints.
- What's the adequate hardware and software?

SVD and the hardware choices

Clusters of multi-core CPUs

- The established technology, excellent if you can afford to buy it, and power it. ScaLapack has `PxGESVD`. Jacobi SVD works, usable with multi-level blocking (machines → cores & caches).

Multi-core CPU, a single graphics processor (GPU)

- A modern, cheaper solution for not too large problems.
- A **hybrid** `xGESVD` algorithm of Magma: bidiagonalization on a GPU, QR on a CPU. Jacobi SVD works, entirely on a GPU, faster than Magma with 1-core MKL, slower than 4-core MKL.

Hybrid multi-CPU, multi-GPU clusters

- A "trendy" solution (see top500.org) for the large systems.
- Jacobi SVD works and scales well, CPU-assisted for now.

What about other accelerators and systems to come?

- Jacobi SVD stays inherently parallel, whatever a definition or implementation of "parallel processing" might have been so far.
- Hierarchical blocking + optimization tricks for the new hardware: Jacobi will probably adapt to the future hybrid topologies and massively parallel accelerator paradigms.
- A Jacobi-type shared-memory generalized SVD algorithm (Hari–Zimmermann) works on a multi-core, multi-CPU machine and on an Intel Xeon Phi coprocessor – the same code (Fortran + OpenMP), with only re-compiling needed.
- A class of hybrid (multi-CPU, multi-Phi) algorithms possible. Optimal division of work between CPUs and Phis, as well as MPI communication pattern, still a matter of research.

The one-sided Jacobi SVD

The (one-sided) Jacobi algorithm computes the SVD of G,

$$G = U\Sigma V^T, \quad U^T U = I, \quad V^T V = I.$$

In each step of the pointwise algorithm:

- form 2×2 pivot submatrix $\widehat{A}$ of $A := G^T G$

$$\widehat{A} = \begin{bmatrix} a_{kk} & a_{k\ell} \\ a_{k\ell} & a_{\ell\ell} \end{bmatrix} = \begin{bmatrix} g_k^T g_k & g_k^T g_\ell \\ g_\ell^T g_k & g_\ell^T g_\ell \end{bmatrix} = \begin{bmatrix} g_k^T \\ g_\ell^T \end{bmatrix} \begin{bmatrix} g_k & g_\ell \end{bmatrix},$$

  from two pivot columns (e.g. g_k and g_ℓ) of G,
- diagonalize A with a single Jacobi rotation $\widehat{R}_{k\ell}$,

$$\widehat{R} := \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}, \quad \widehat{R}_{k\ell}^T \widehat{A} \widehat{R}_{k\ell} = \operatorname{diag}(\hat{\lambda}_k, \hat{\lambda}_\ell).$$

The one-sided Jacobi SVD (continued)

Implicit orthogonalization: Transformations $\widehat{R}$ are then applied from the right on G, orthogonalizing the pivot columns

$$\widehat{G} = G\widehat{R}.$$

The Jacobi process iterates over pivot pairs (usually in sweeps over all pairs) until the convergence criterion is met.

- How to choose pivots? According to a suitable pivot strategy.
- What criterion to take for the convergence? E.g. the pivot columns are relatively orthogonal up to the machine precision ε

$$\frac{a_{k\ell}}{\sqrt{a_{kk}a_{\ell\ell}}} < \varepsilon\sqrt{n}.$$

- May stop earlier by controlling when the rotations are "sufficiently close" to identity.

Blocking of the Jacobi algorithm

Correspondence between the non-blocked and the blocked Jacobi:

- a pair (g_k, g_l) of columns $\leftrightarrow$ a pair (G_k, G_l) of block-columns;
- 2×2 pivot matrix $\widehat{A}_2 \leftrightarrow m \times m$ pivot matrix $\widehat{A}_m$ $(m \geq 2)$.

Factorize the pivot block-columns, or pivot matrix $\widehat{A}_m$, either:

- implicitly: $G_m := [G_k \ G_l]$, $G_m = Q_m R_m$ (more accurate), or
- explicitly: compute $\widehat{A}_m$ by multiplication and R_m is obtained by the (diagonally pivoted) Cholesky factorization (faster).

With R_m there are (at least) two choices:

- block-oriented: apply one sweep of the "ordinary" one-sided Jacobi algorithm to reduce the pivot's off-diagonal norm, or
- full-block: fully diagonalize the pivot matrix.

The accumulated transformations are applied to the block columns.

Hierarchical (multi-level) blocking

Subdivide recursively the larger blocks into smaller ones:

- the pivot block columns of a level ℓ, $\left[G_k^{(\ell)}\ G_l^{(\ell)}\right]$, are "shortended" by either of the factorizations to $R^{(\ell)}$;
- $R^{(\ell)}$ is further subdivided into block columns at the level $\ell - 1$, as if $\left[U^{(\ell)}, \Sigma^{(\ell)}, V^{(\ell)}\right] = \text{Jacobi}\left(R^{(\ell)}\right)$ is recursively called;
- $V^{(\ell)}$ might be accumulated (product of rotations $\rightarrow$ stable), or found as the solution of a triangular linear system (fast):

$$R^{(\ell)} V^{(\ell)} = U^{(\ell)} \Sigma^{(\ell)};$$

- the pivot columns are updated (orthogonalized):

$$\left[\widehat{G}_k^{(\ell)}\ \widehat{G}_l^{(\ell)}\right] = \left[G_k^{(\ell)}\ G_l^{(\ell)}\right] V^{(\ell)};$$

- Jacobi $\left(R^{(0)}\right)$ is the pointwise algorithm.

Jacobi parallelization $= X + Y + Z$

Three important ingredients of a parallel Jacobi SVD

- X: a parallel pivot strategy makes the Jacobi SVD – parallel;
- Y: the hardware and its constraints;
- Z: a blocking hierarchy that reflects Y's memory hierarchy.

The main idea

- Process as many blocks as possible in parallel.
- Find just the right level and coarseness of blocking to optimally exploit the hardware (e.g., saturate the caches).

Jacobi strategies

Definition

A cyclic Jacobi strategy $\mathcal{O}$ is a periodic sequence of pivot pairs,

$$\mathcal{O} = ((p_k, q_k) \mid k \in \mathbb{N}), \quad 1 \leq p_k < q_k \leq n,$$

where a cycle contains $\tau = n(n-1)/2$ successive pivots.

Well-known (sequential) examples: row- ($\mathcal{R}$) and column-cyclic ($\mathcal{C}$).

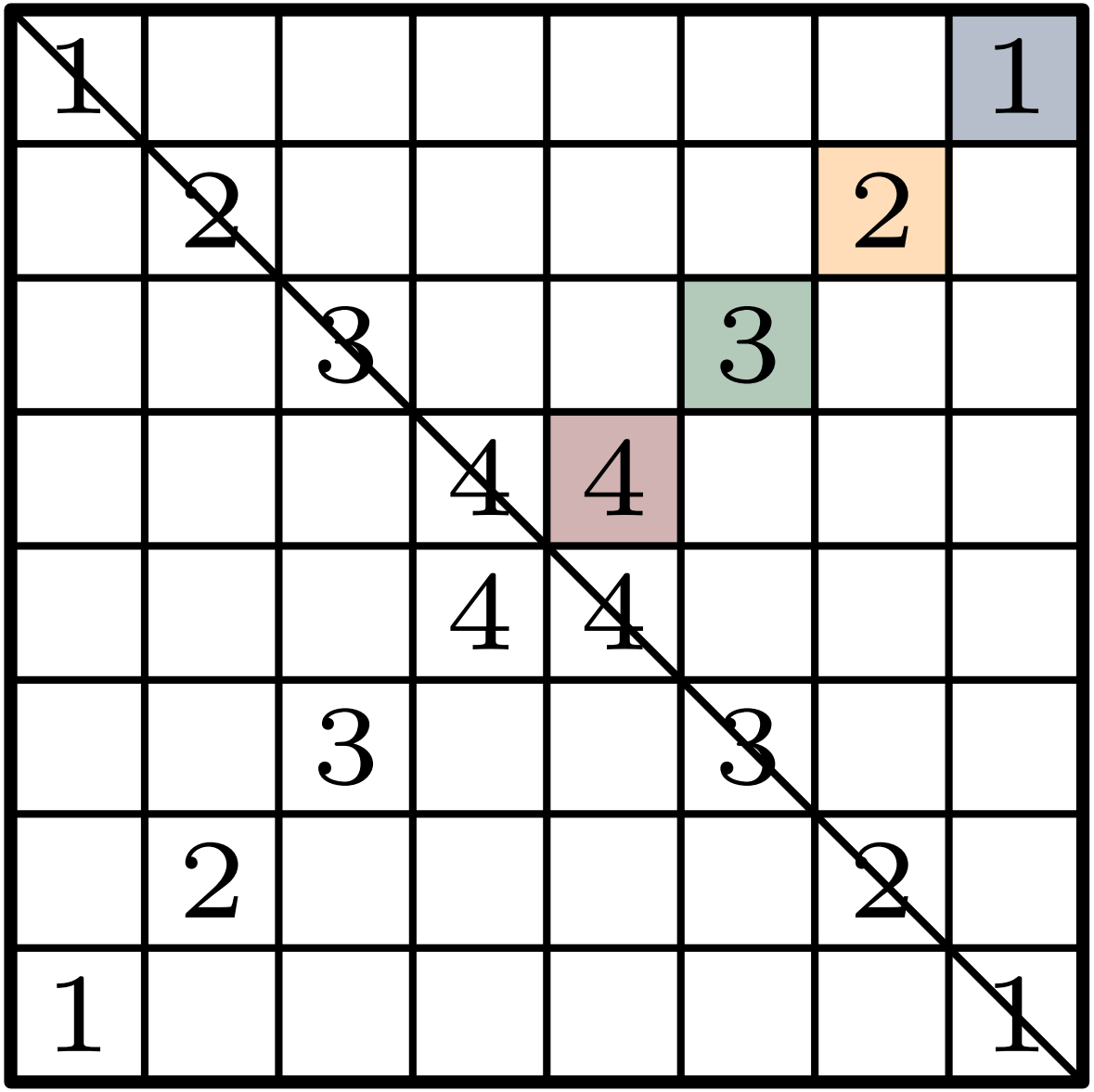
Definition

A parallel Jacobi strategy (p-strategy) simultaneously processes $\leq t$ independent pivots in $\geq s$ parallel steps (p-steps). If, for even n,

$$t = n/2, \qquad s = n - 1,$$

the strategy is perfectly parallel (i.e., minimal number of p-steps, with maximal amount of parallel tasks possible).

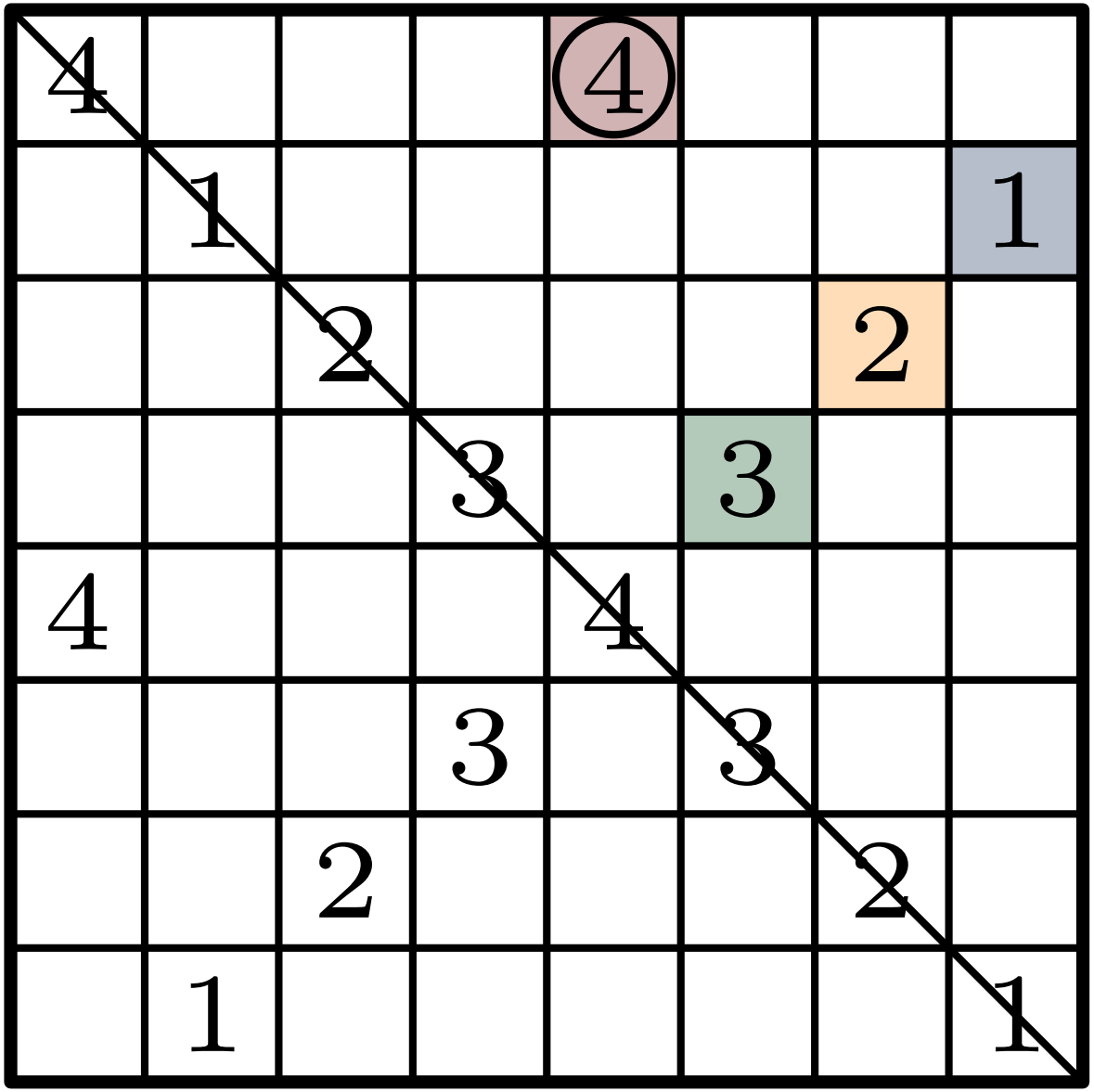
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

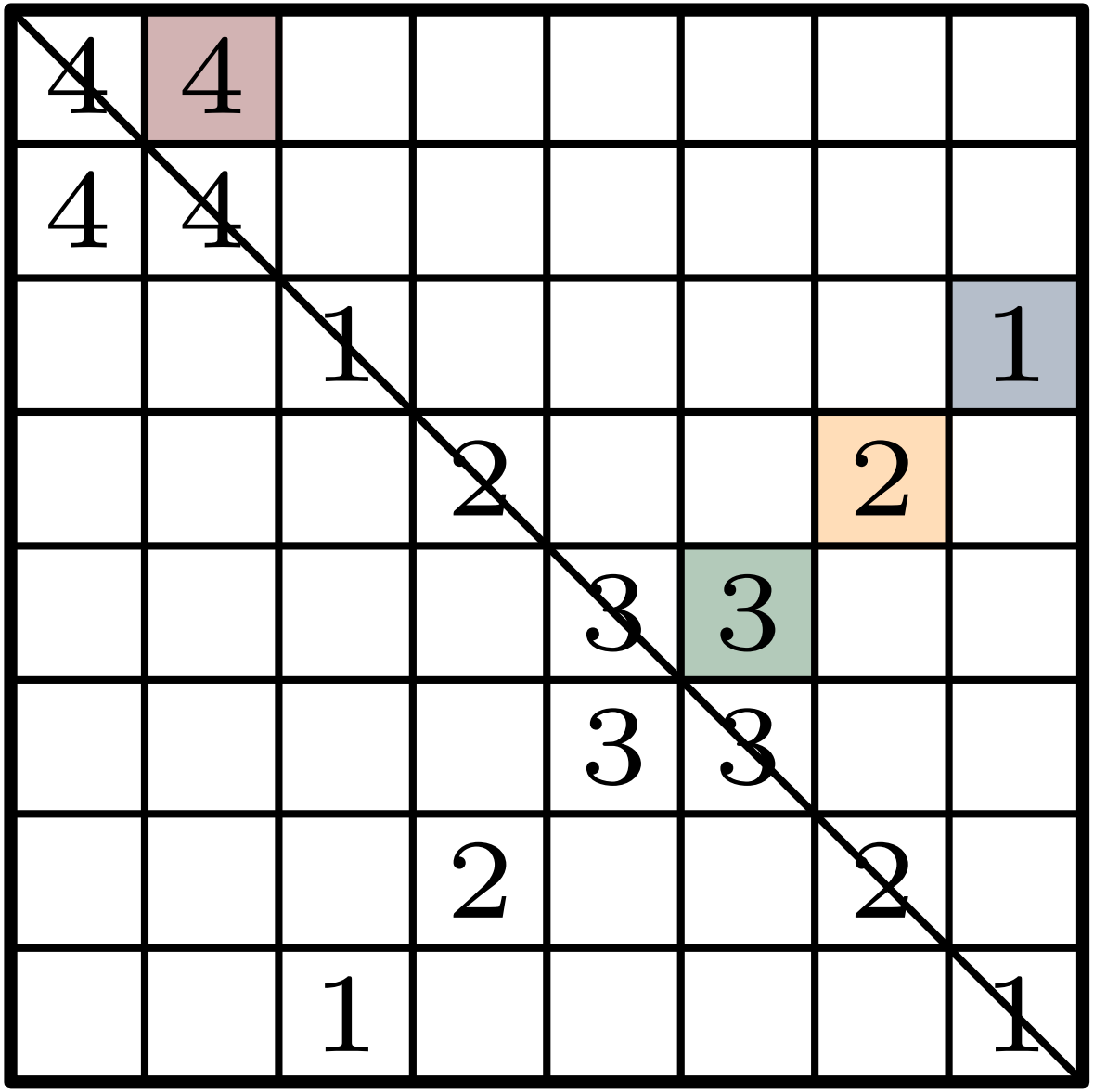
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

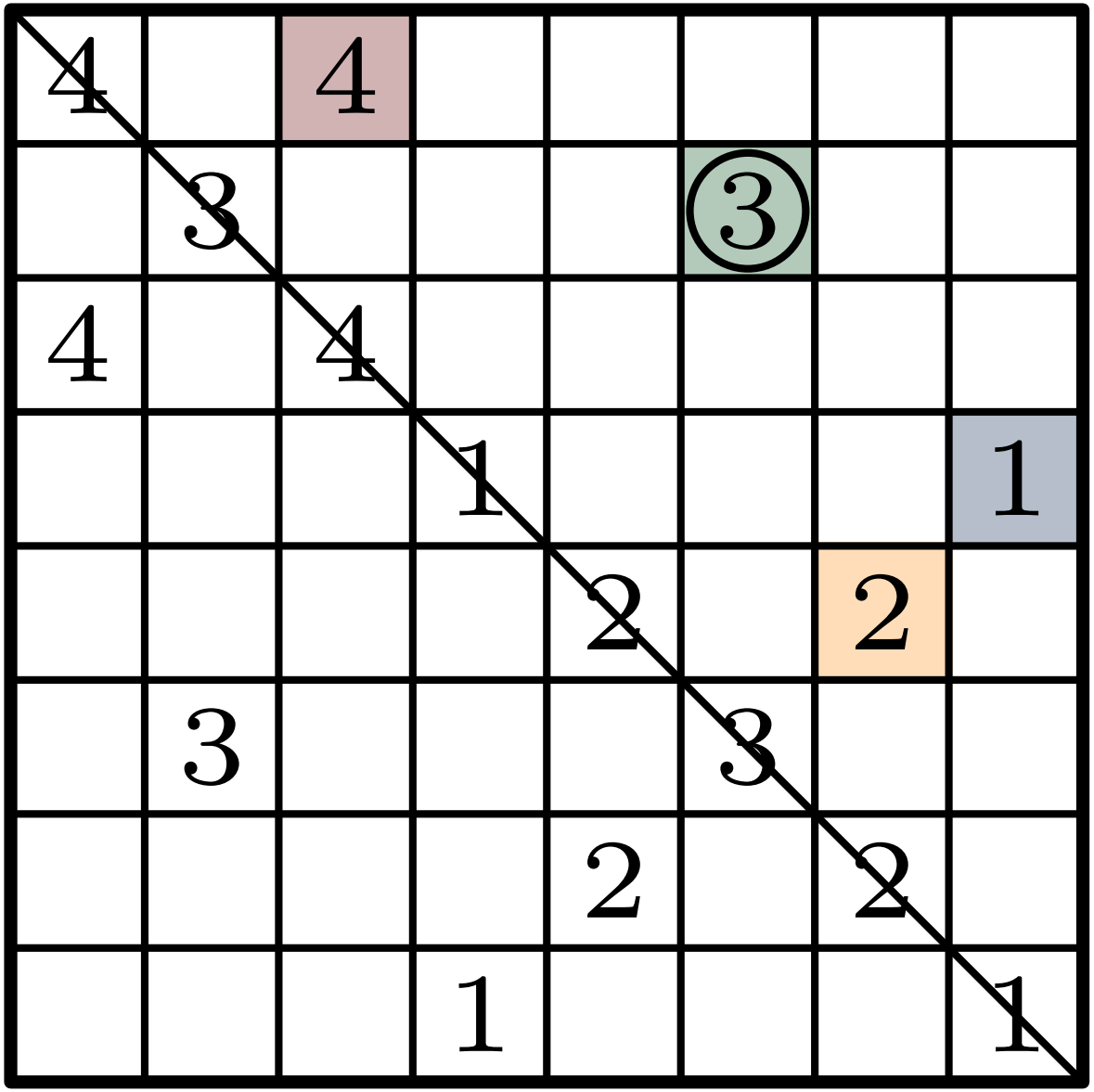
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

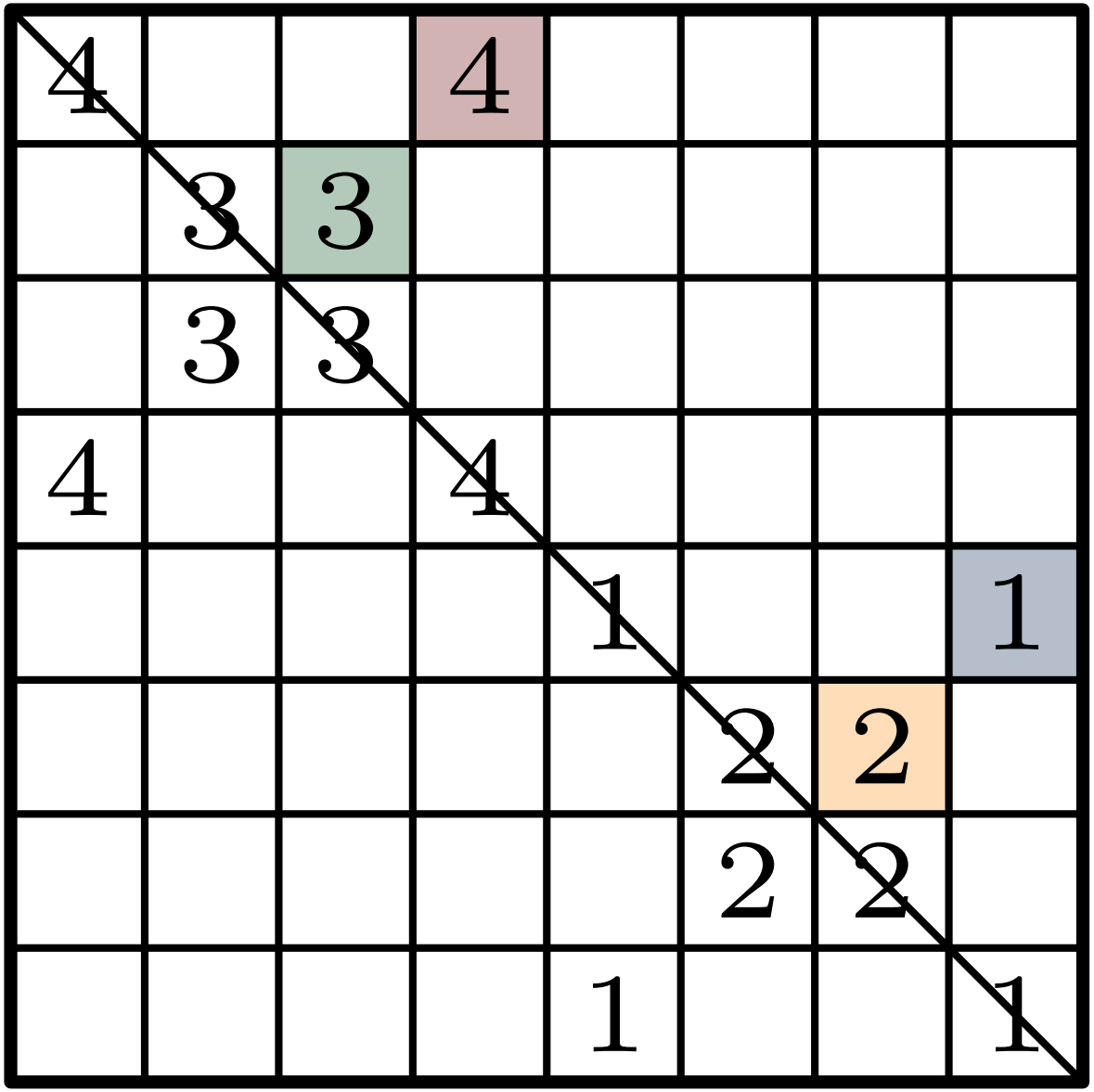
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

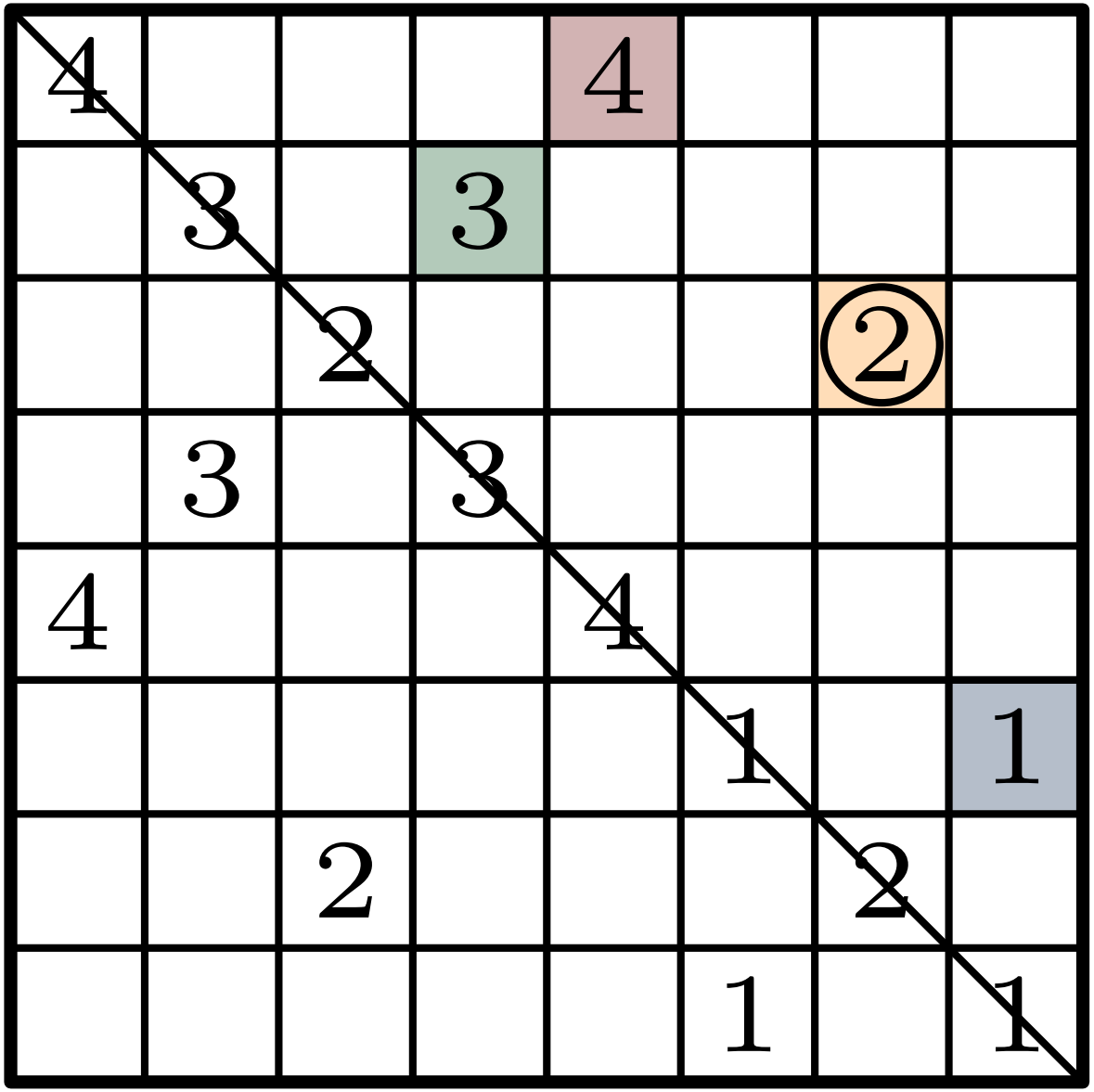
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

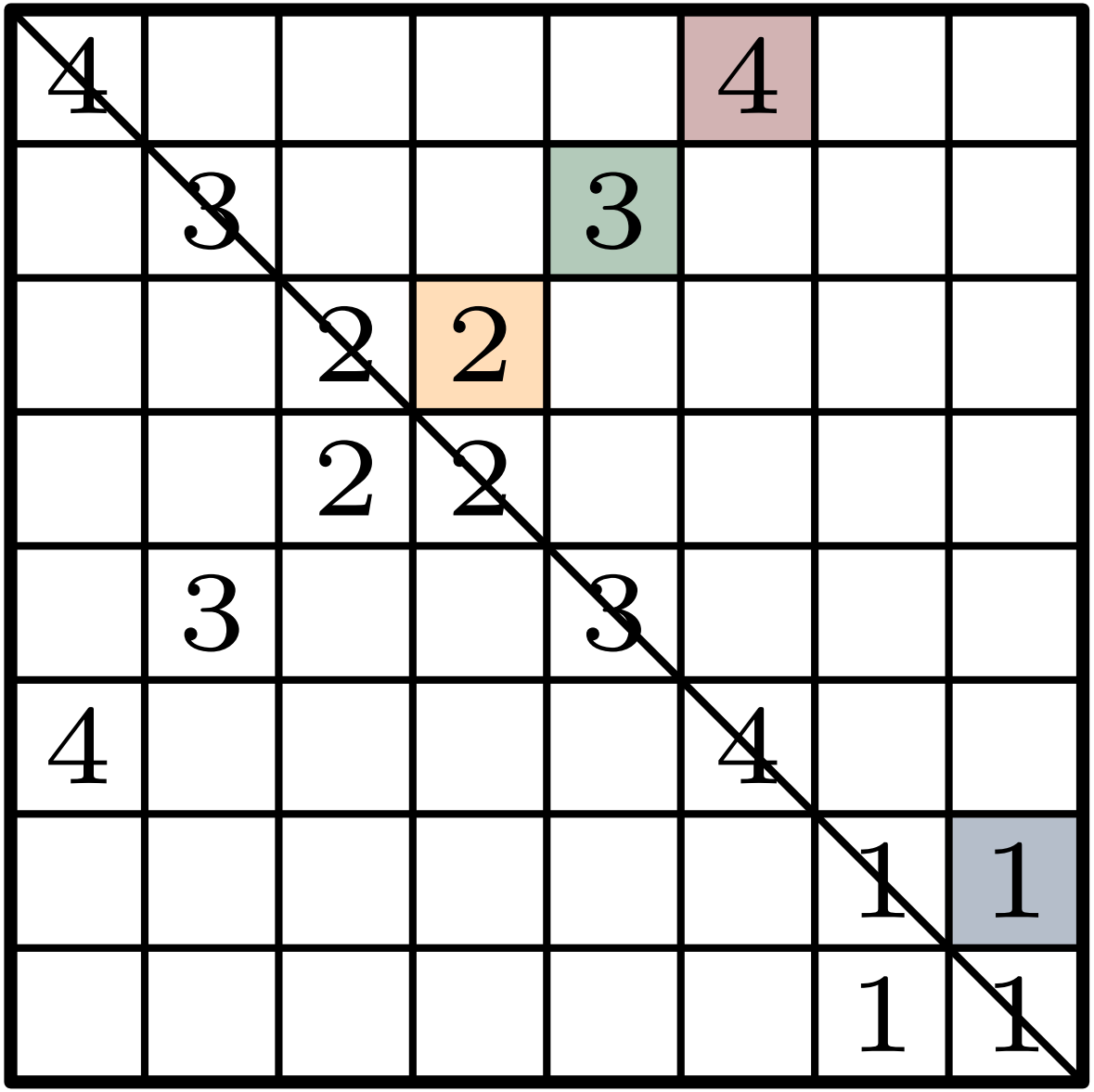
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

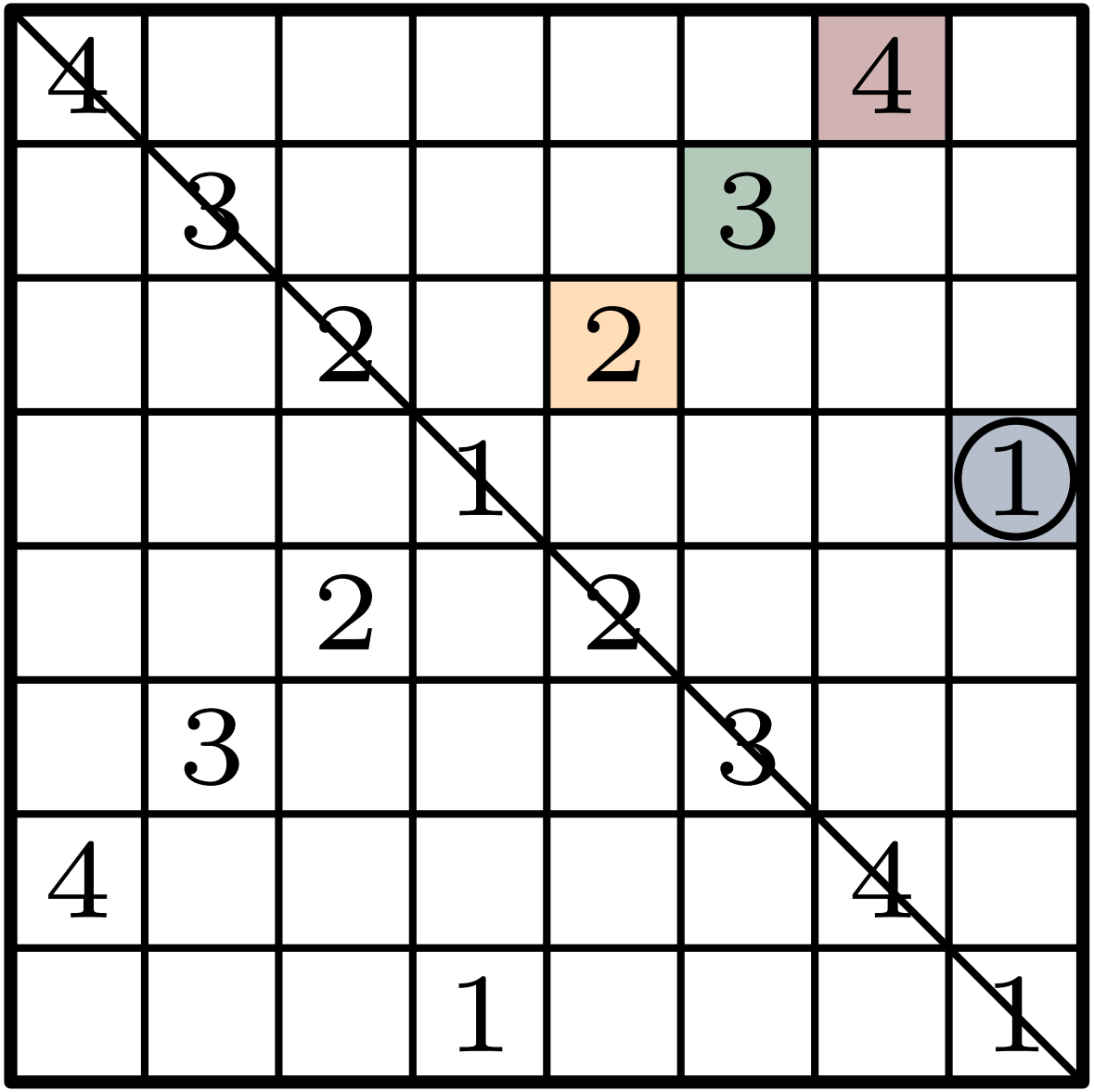
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

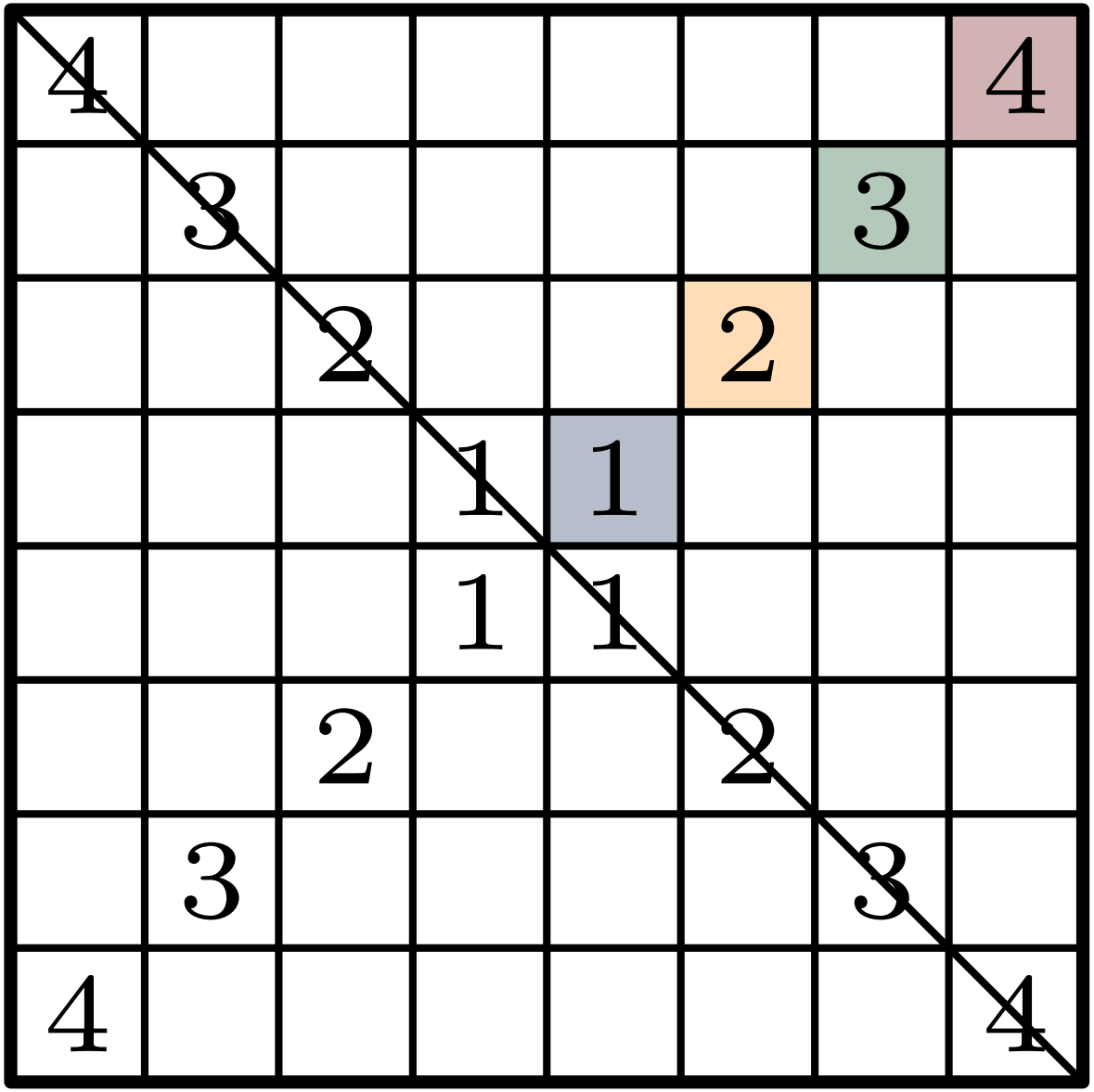
A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

A single sweep of the modified modulus strategy



Suppose that A has 8 block–columns divided among 4 parallel tasks (1..4).

The circled pivots are processed twice, with 8 p-steps instead of min. 7.

Positions of block columns are now "inverted". Another sweep will take them into initial position again.

Cyclic Jacobi p-strategies

Some examples of p-strategies in common usage:

- Modulus (Sameh) – convergent by equivalence to row-cyclic;
- Brent-Luk – convergent for odd n, problematic for even n;
- Mantharam-Eberlein Block Recursive – for the hypercube topologies, i.e., power-of-two n only; convergence unknown.

Can we do better, i.e., find a p-strategy that is:

- **faster** than the "usual" p-strategies;
- with more **accurate** results;
- easy to **generate** or describe;
- and provably **convergent**?

Cyclic Jacobi p-strategies

Some examples of p-strategies in common usage:

- Modulus (Sameh) – convergent by equivalence to row-cyclic;
- Brent-Luk – convergent for odd n, problematic for even n;
- Mantharam-Eberlein Block Recursive – for the hypercube topologies, i.e., power-of-two n only; convergence unknown.

Can we do better, i.e., find a p-strategy that is:

- **faster** than the "usual" p-strategies; YES (probably)
- with more **accurate** results;
- easy to **generate** or describe;
- and provably **convergent**?

Cyclic Jacobi p-strategies

Some examples of p-strategies in common usage:

- Modulus (Sameh) – convergent by equivalence to row-cyclic;
- Brent-Luk – convergent for odd n, problematic for even n;
- Mantharam-Eberlein Block Recursive – for the hypercube topologies, i.e., power-of-two n only; convergence unknown.

Can we do better, i.e., find a p-strategy that is:

- **faster** than the "usual" p-strategies; YES (probably)
- with more **accurate** results; YES (probably)
- easy to **generate** or describe;
- and provably **convergent**?

Cyclic Jacobi p-strategies

Some examples of p-strategies in common usage:

- Modulus (Sameh) – convergent by equivalence to row-cyclic;
- Brent-Luk – convergent for odd n, problematic for even n;
- Mantharam-Eberlein Block Recursive – for the hypercube topologies, i.e., power-of-two n only; convergence unknown.

Can we do better, i.e., find a p-strategy that is:

- **faster** than the "usual" p-strategies; YES (probably)
- with more **accurate** results; YES (probably)
- easy to **generate** or describe; MAYBE (describe for now)
- and provably **convergent**?

Cyclic Jacobi p-strategies

Some examples of p-strategies in common usage:

- Modulus (Sameh) – convergent by equivalence to row-cyclic;
- Brent-Luk – convergent for odd n, problematic for even n;
- Mantharam-Eberlein Block Recursive – for the hypercube topologies, i.e., power-of-two n only; convergence unknown.

Can we do better, i.e., find a p-strategy that is:

- **faster** than the "usual" p-strategies; YES (probably)
- with more **accurate** results; YES (probably)
- easy to **generate** or describe; MAYBE (describe for now)
- and provably **convergent**? MAYBE (future work)

Cyclic Jacobi p-strategies

Some examples of p-strategies in common usage:

- Modulus (Sameh) – convergent by equivalence to row-cyclic;
- Brent-Luk – convergent for odd n, problematic for even n;
- Mantharam-Eberlein Block Recursive – for the hypercube topologies, i.e., power-of-two n only; convergence unknown.

Can we do better, i.e., find a p-strategy that is:

- **faster** than the "usual" p-strategies; YES (probably)
- with more **accurate** results; YES (probably)
- easy to **generate** or describe; MAYBE (describe for now)
- and provably **convergent**? MAYBE (future work)

Main idea: inherit the "good" properties of the row/column-cyclic strategies by finding a parallel one "closest" to them in some sense.

What it means to tell that $\mathcal{O}_1$ is closer to $\mathcal{O}$ than $\mathcal{O}_2$?

Depends on a definition of "closeness"... let's try with this one!

Take a pivot (p, q), and the same-sized cyclic strategies $\mathcal{O}$ (fixed) and $\mathcal{O}'$. If $(p, q) = (p_k, q_k) \in \mathcal{O}$, then $(p, q) = (p_{\ell(k)}, q_{\ell(k)}) \in \mathcal{O}'$, for some k and $\ell(k)$, and $k \mapsto \ell(k)$ is a permutation. Let

$$I_{\mathcal{O}}(\mathcal{O}') = (\ell(1), \ell(2), \ldots, \ell(k), \ldots, \ell(\tau)) \in \mathrm{Sym}(n).$$

Definition

For any two cyclic Jacobi strategies, $\mathcal{O}_1$ and $\mathcal{O}_2$, we say $\mathcal{O}_1$ is closer to $\mathcal{O}$ than $\mathcal{O}_2$, and denote that by $\mathcal{O}_1 \preceq_{\mathcal{O}} \mathcal{O}_2$, if $I_{\mathcal{O}}(\mathcal{O}_1) \preceq I_{\mathcal{O}}(\mathcal{O}_2)$, where $\preceq$ stands for the lexicographic ordering of permutations.

There exist a unique p-strategy $\mathcal{O}^{\|}$ closest to a given $\mathcal{O}$, since $\preceq$ is a total order. E.g., for $\mathcal{O} = \mathcal{R}_n$ (or $\mathcal{C}_n$), $\mathcal{R}_n^{\|}$ and $\mathcal{C}_n^{\|}$ are well-defined.

So, $\mathcal{O}^{\|}$ exists and is unique... Now, how to find it?

Just rephrase the question in the graph-theoretical parlance!

Let G be a simple, connected graph with τ vertices.

- Each vertex represents a pivot from a given $\mathcal{O}$.
- An edge exist whenever two pivots share a column index (i.e., can not be processed indepenently, in the same p-step).

Any maximal independent set[1] of G may be processed in parallel.

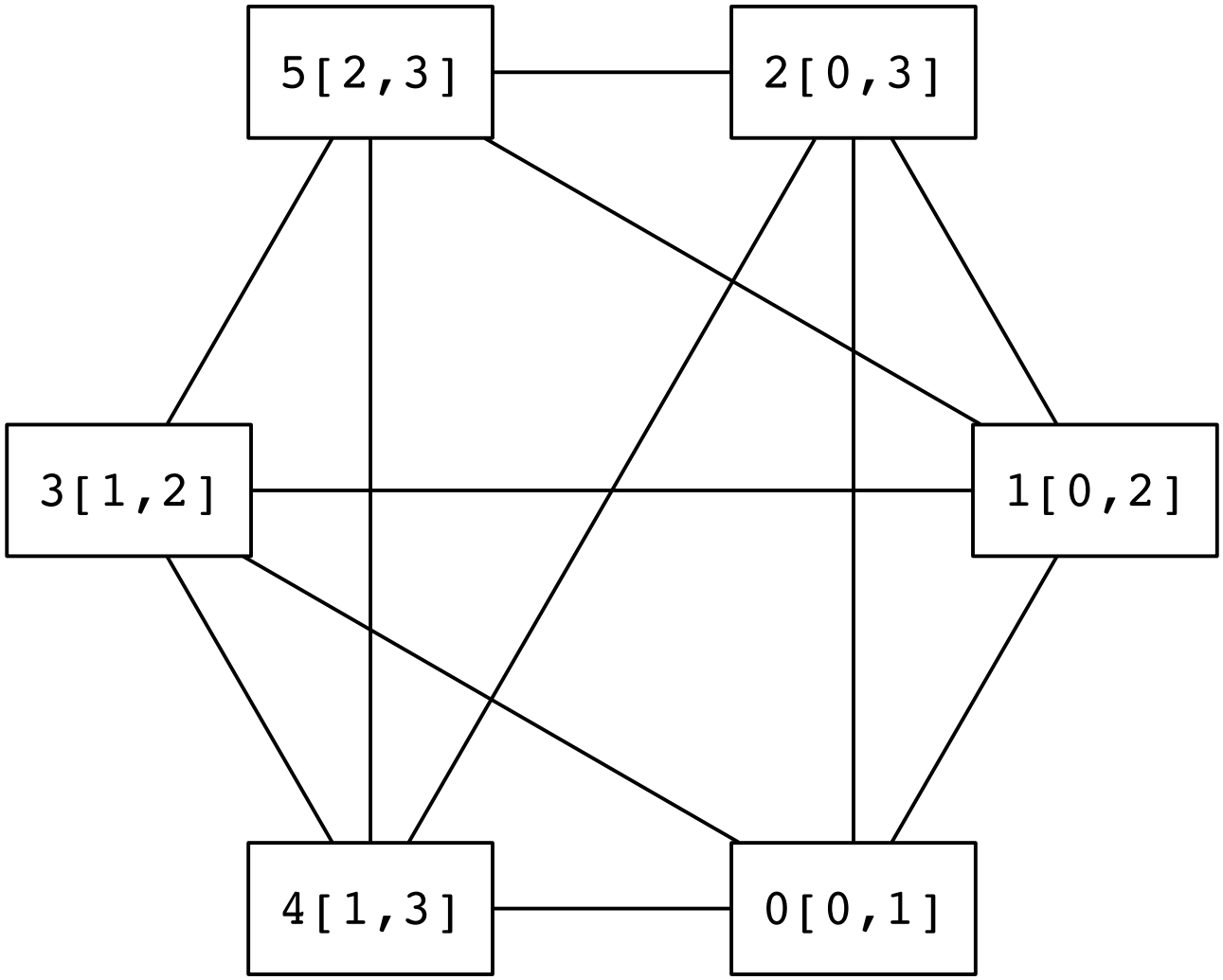
- Each MIS has no more than n vertices/pivots.
- Any MIS S with n vertices is an admissible p-step.
- The previous statements hold for $G \setminus S$, also.

The Jacobi method is invariant to the order of pivots in a p-step, so the vertices in a MIS may be assumed to be sorted ascendingly.

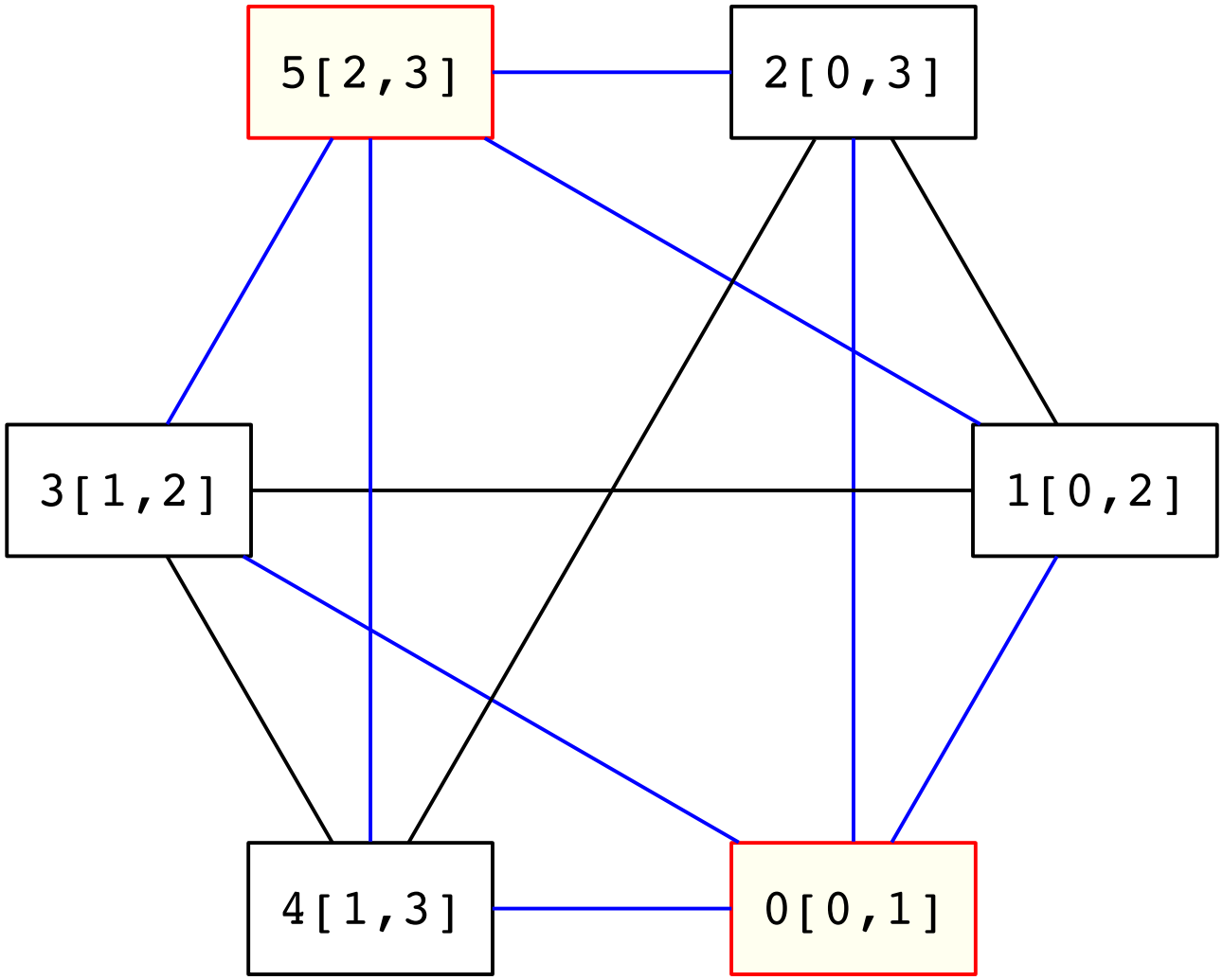
Therefore, p-step candidates may be compared lexicographically!

[1] MIS: a maximal set of vertices, no two of which are adjacent.

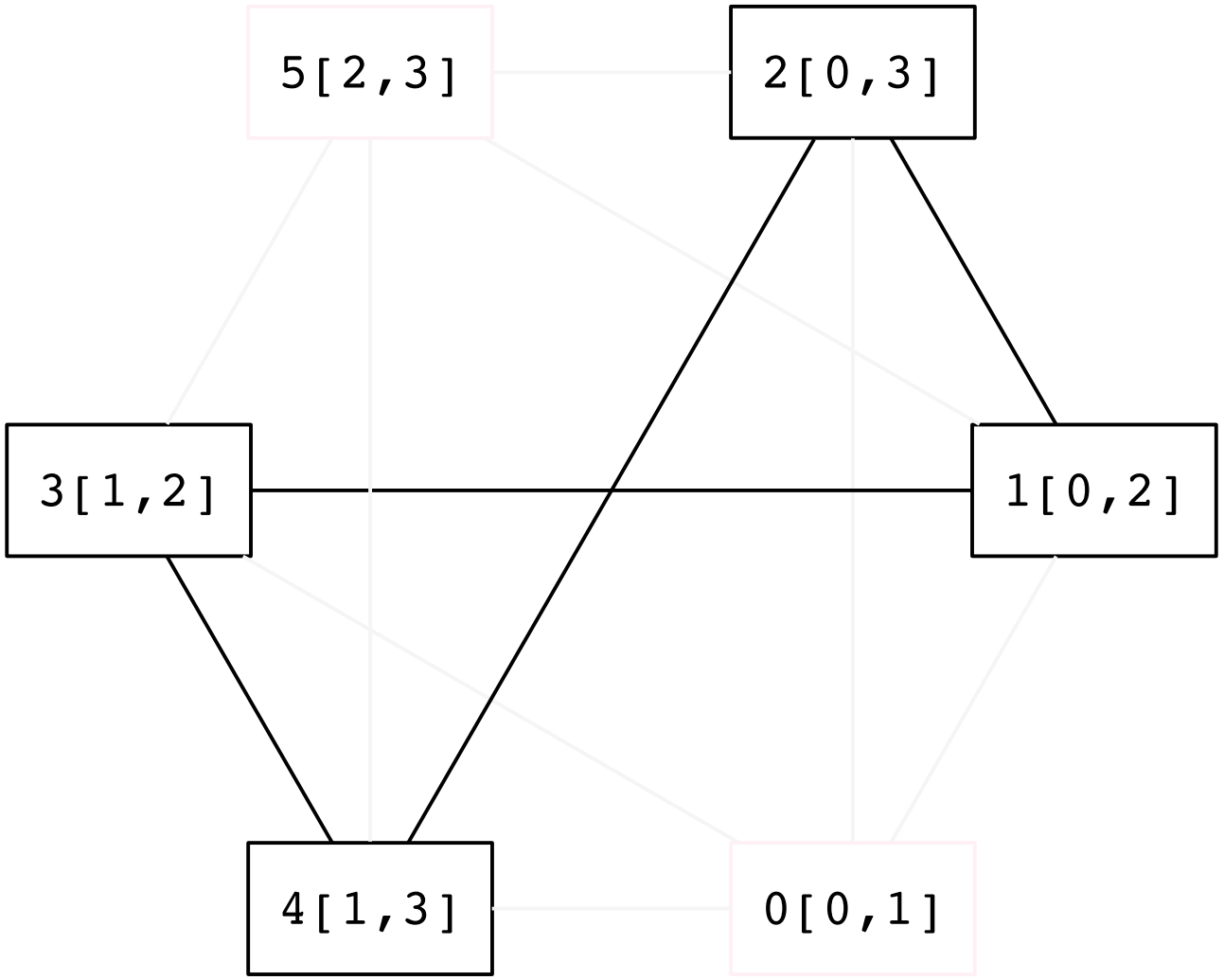
$\mathcal{R}_4^{\|}$ generation, `next_lex` does not retry



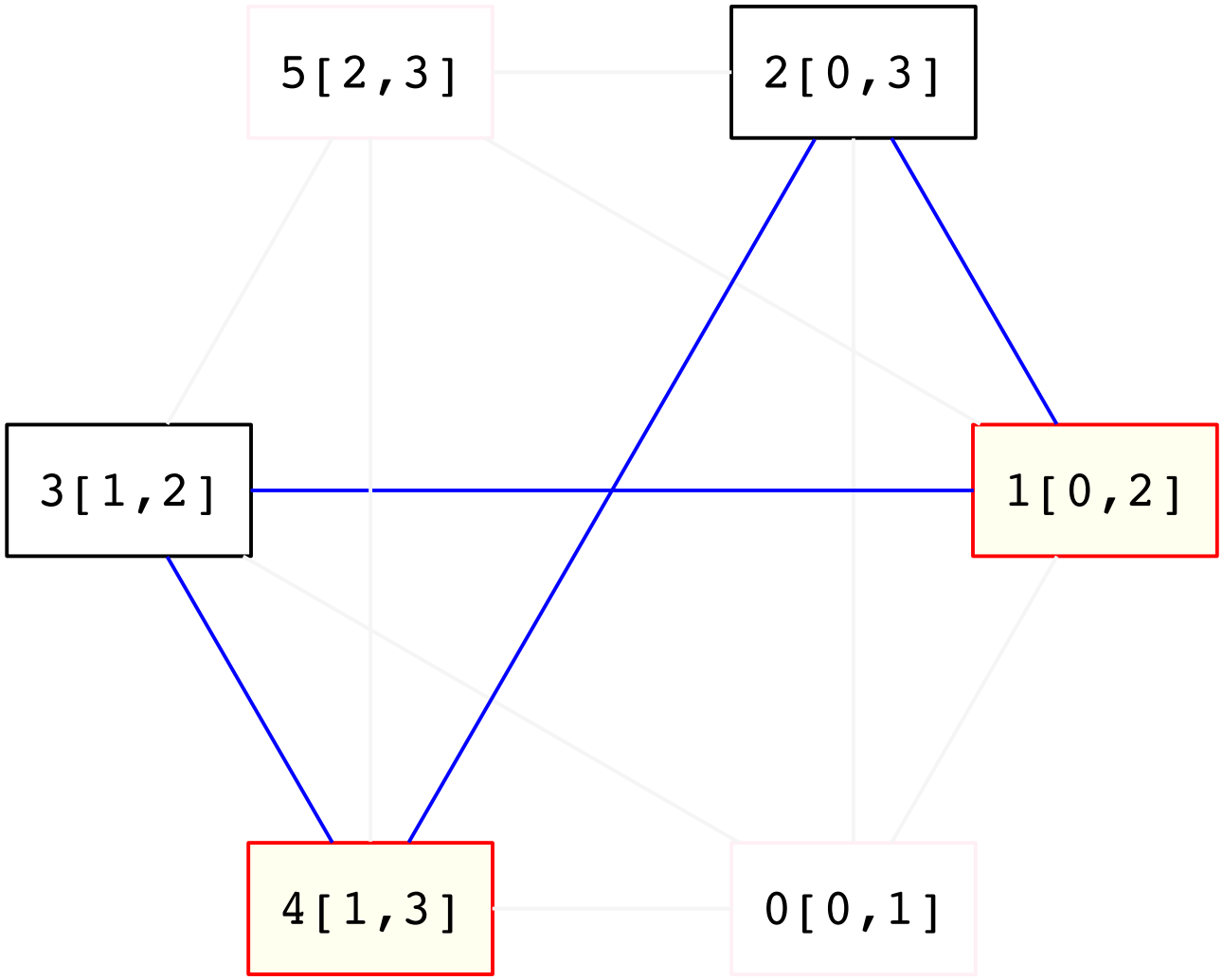
$\mathcal{R}_4^{\|}$ generation, `next_lex` does not retry



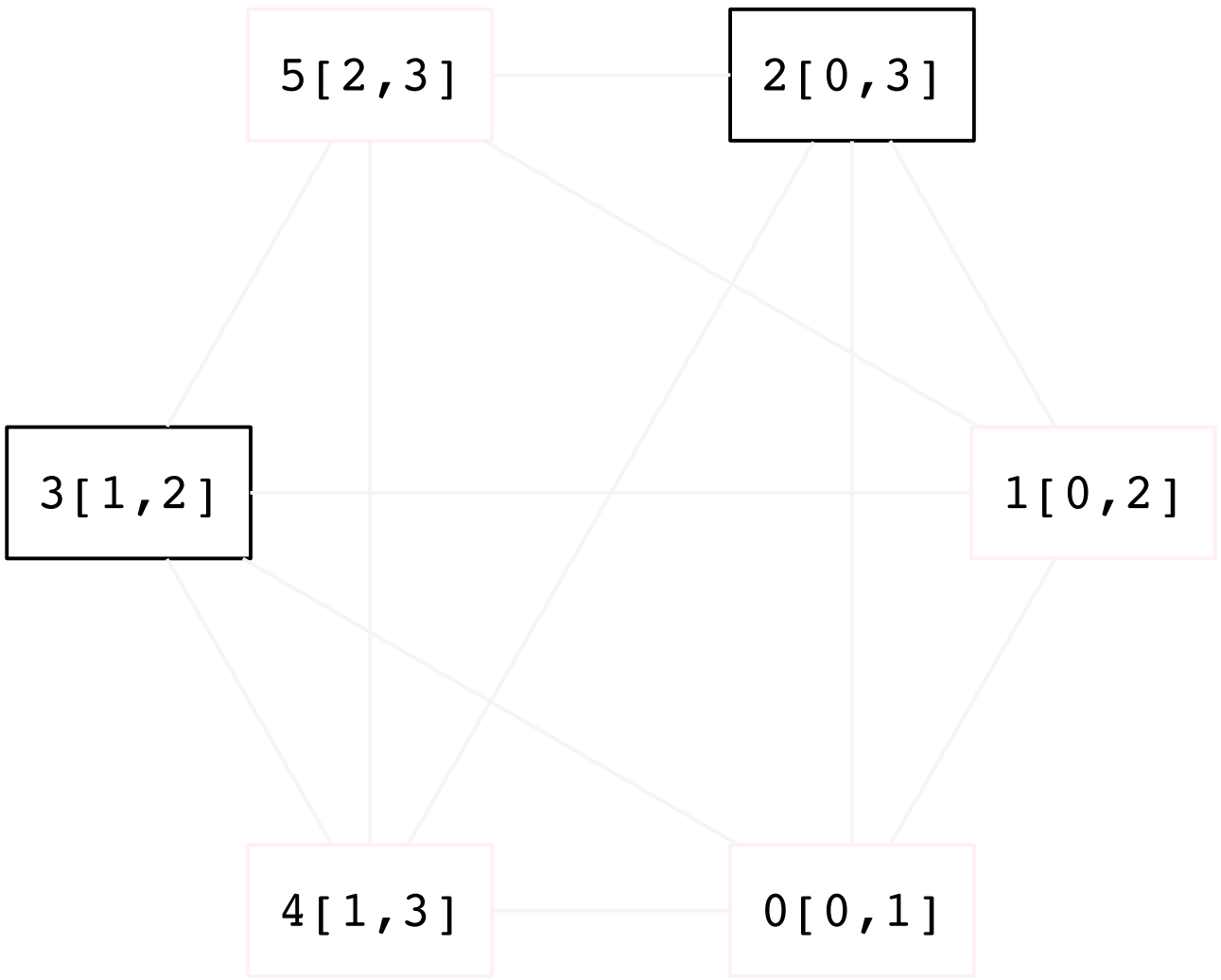
$\mathcal{R}_4^{\|}$ generation, `next_lex` does not retry



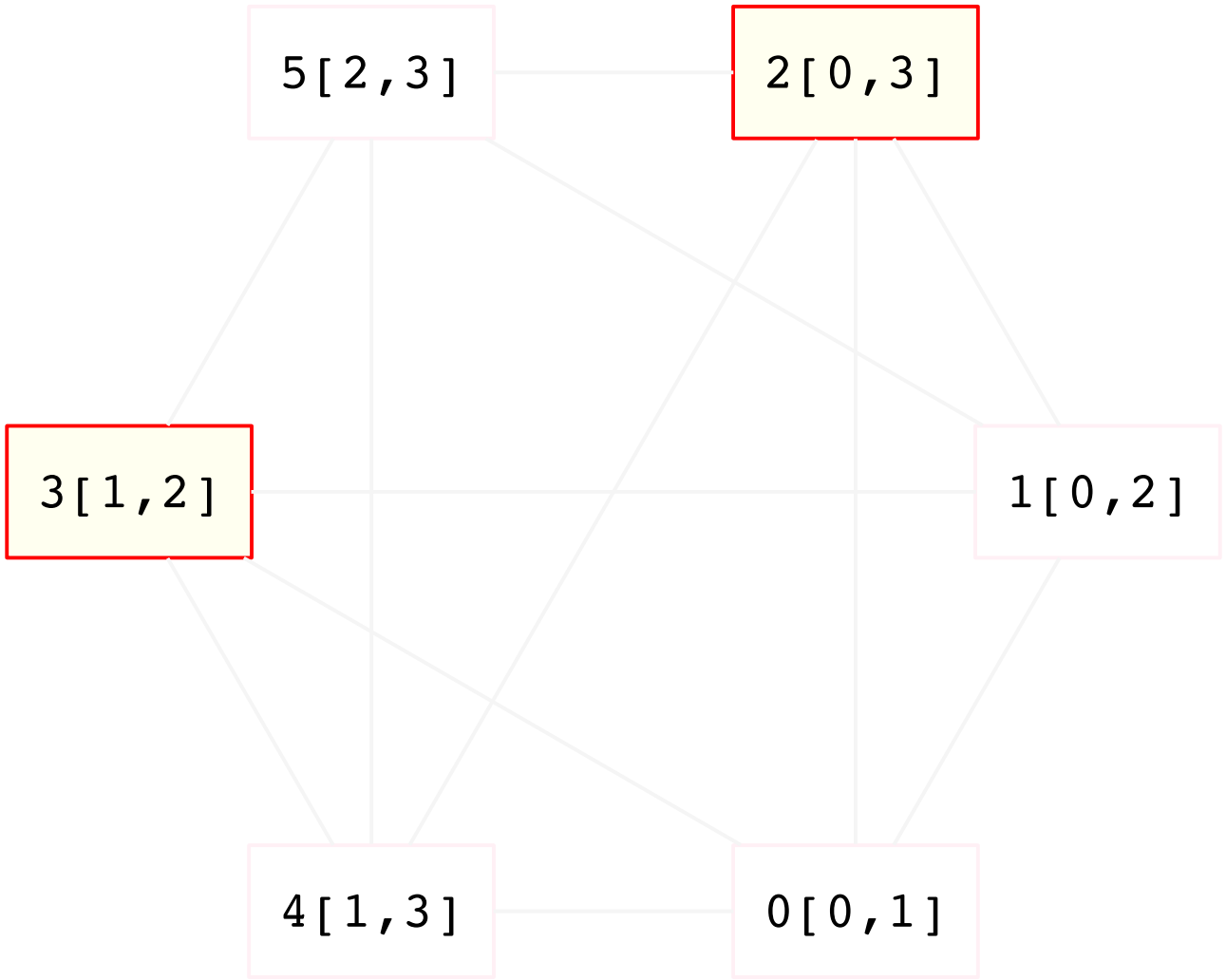
$\mathcal{R}_4^{\|}$ generation, `next_lex` does not retry



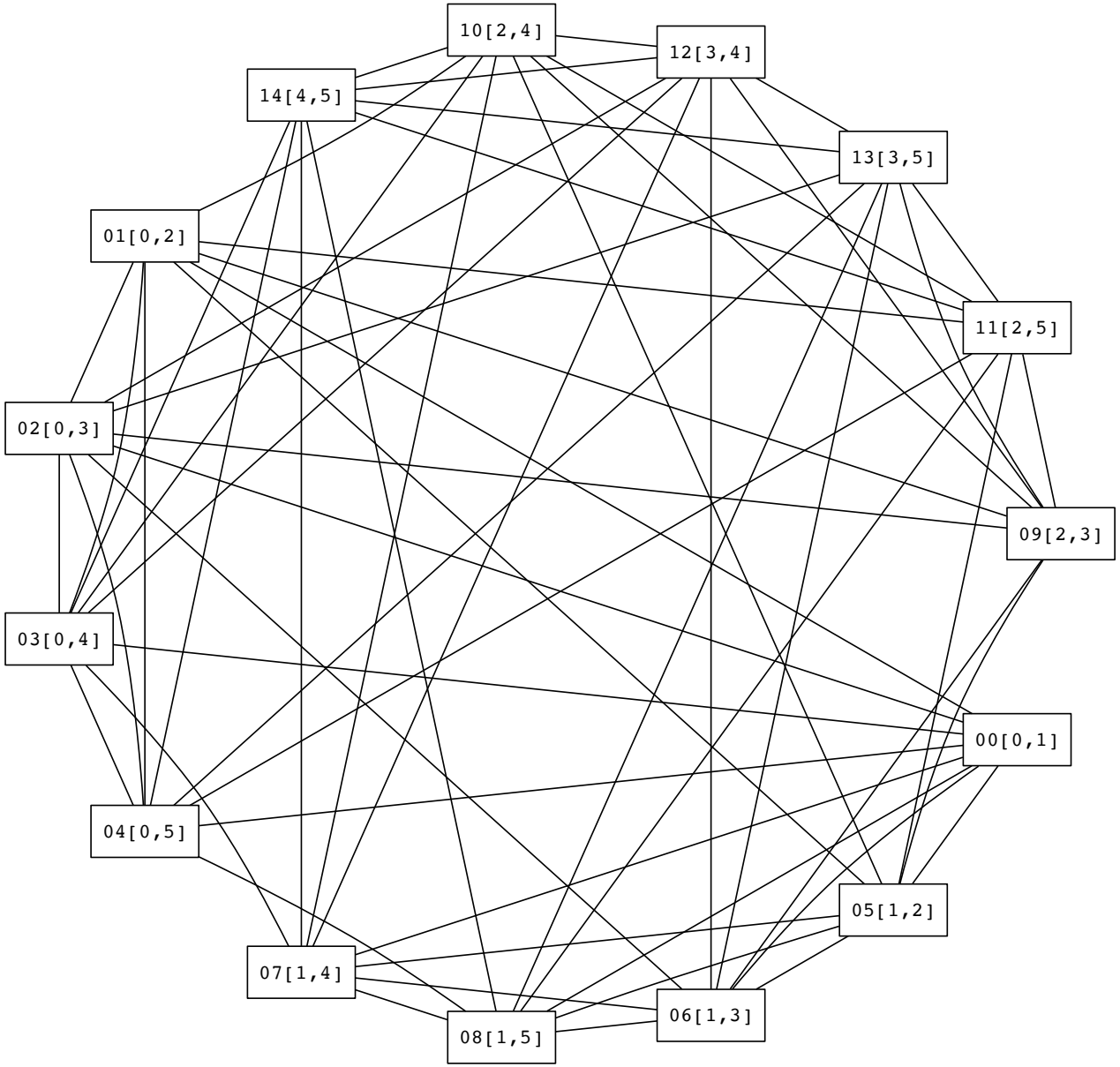
$\mathcal{R}_4^{\|}$ generation, `next_lex` does not retry



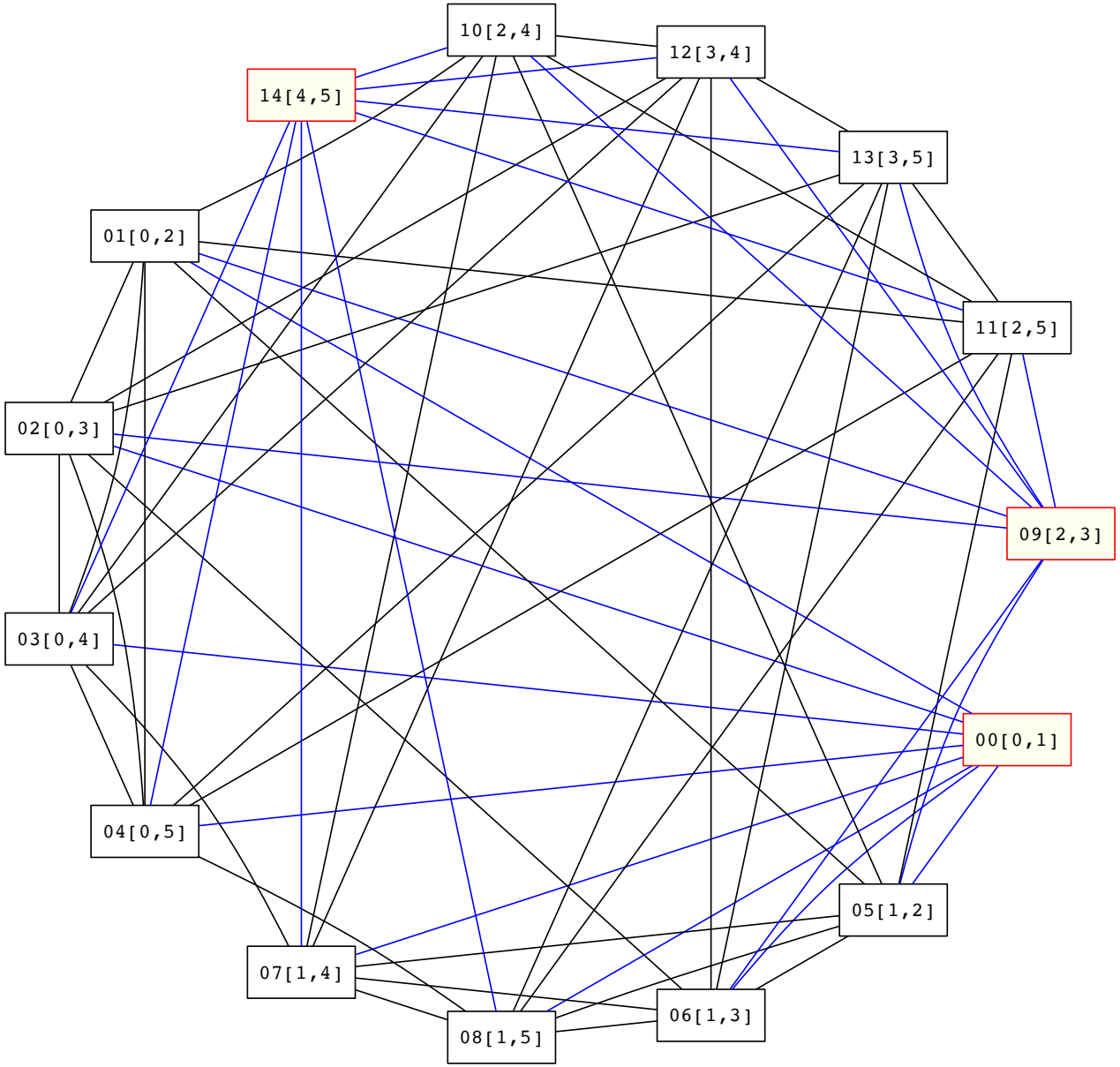
$\mathcal{R}_4^{\|}$ generation, `next_lex` does not retry



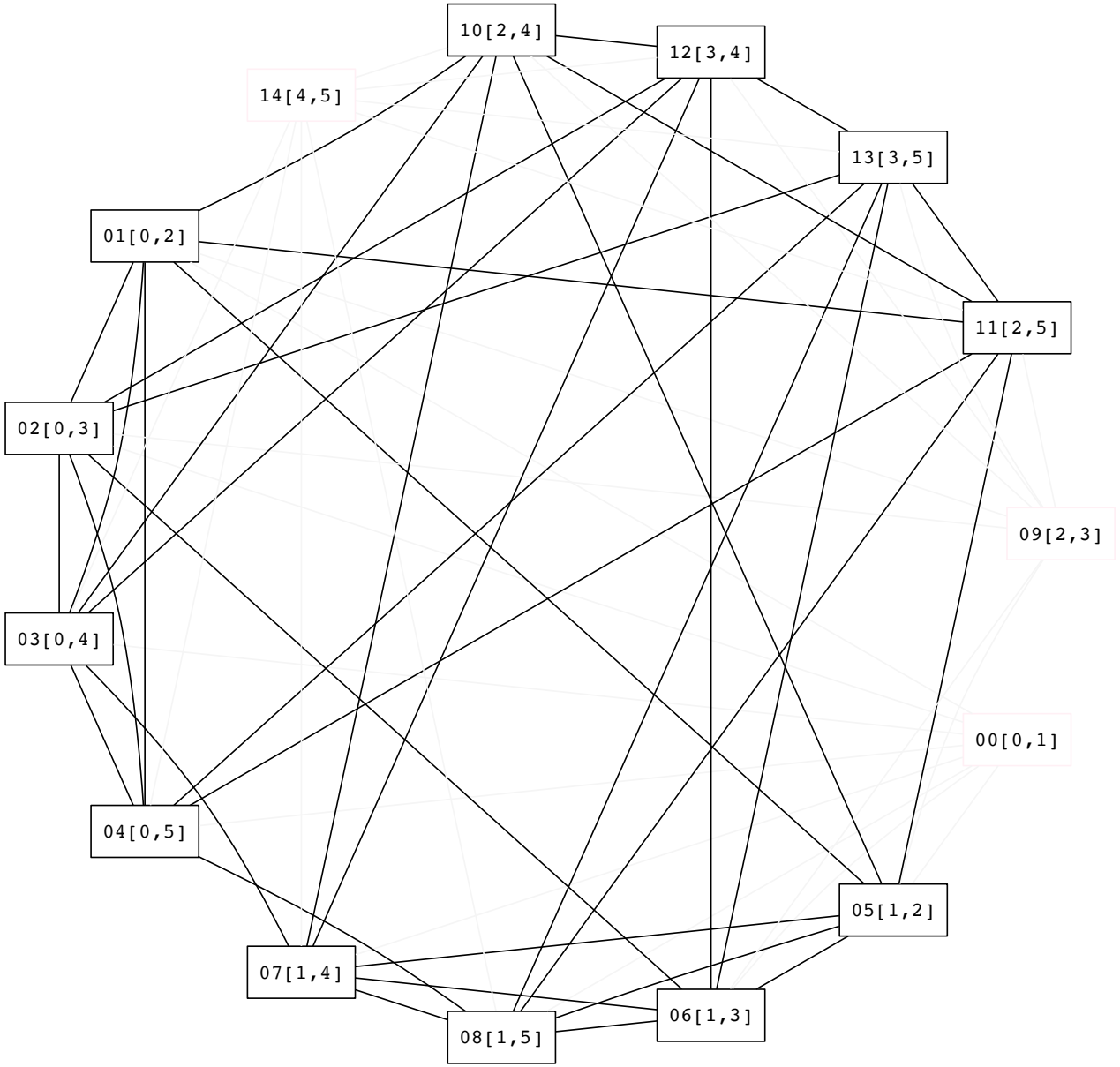
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



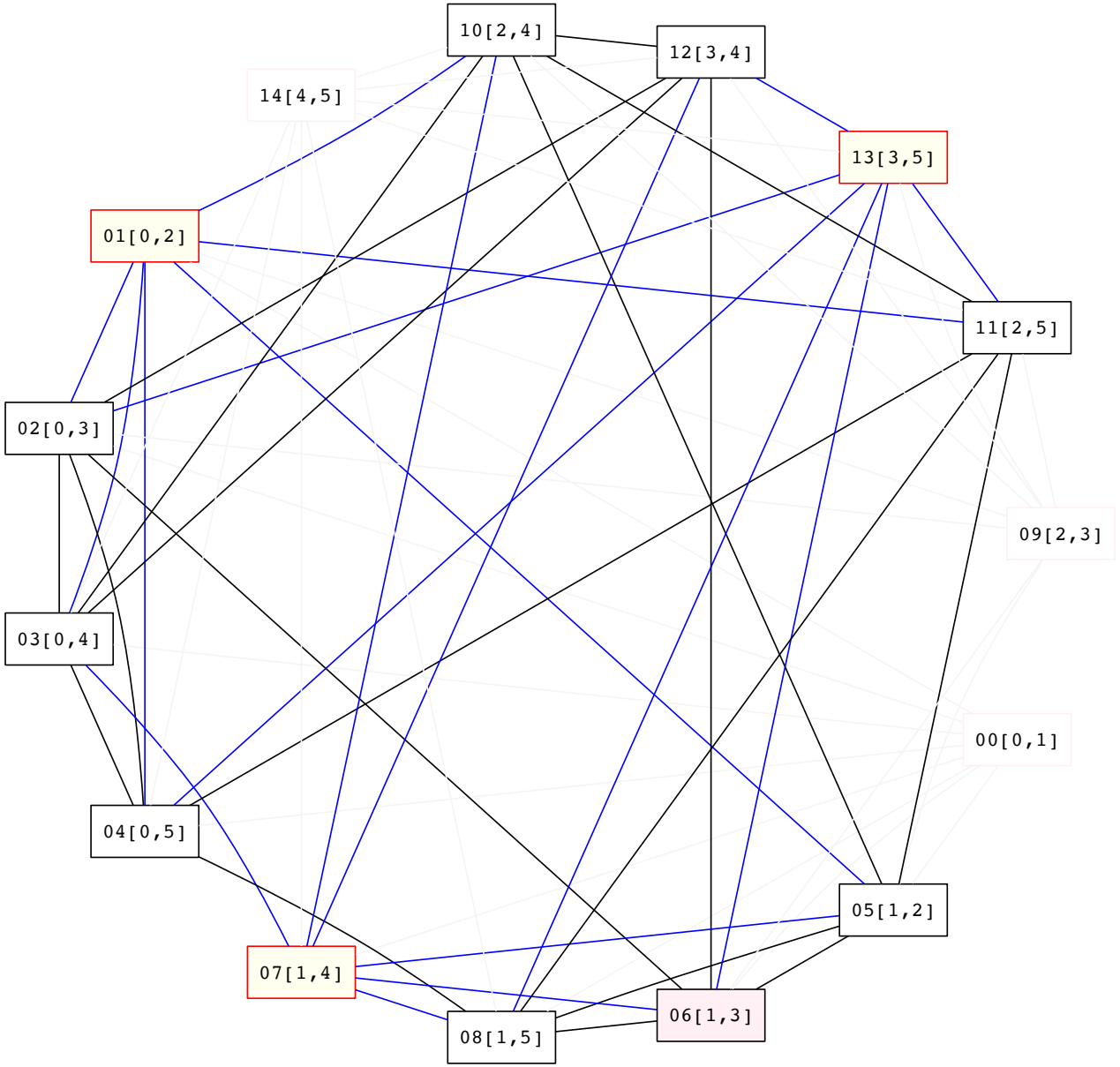
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



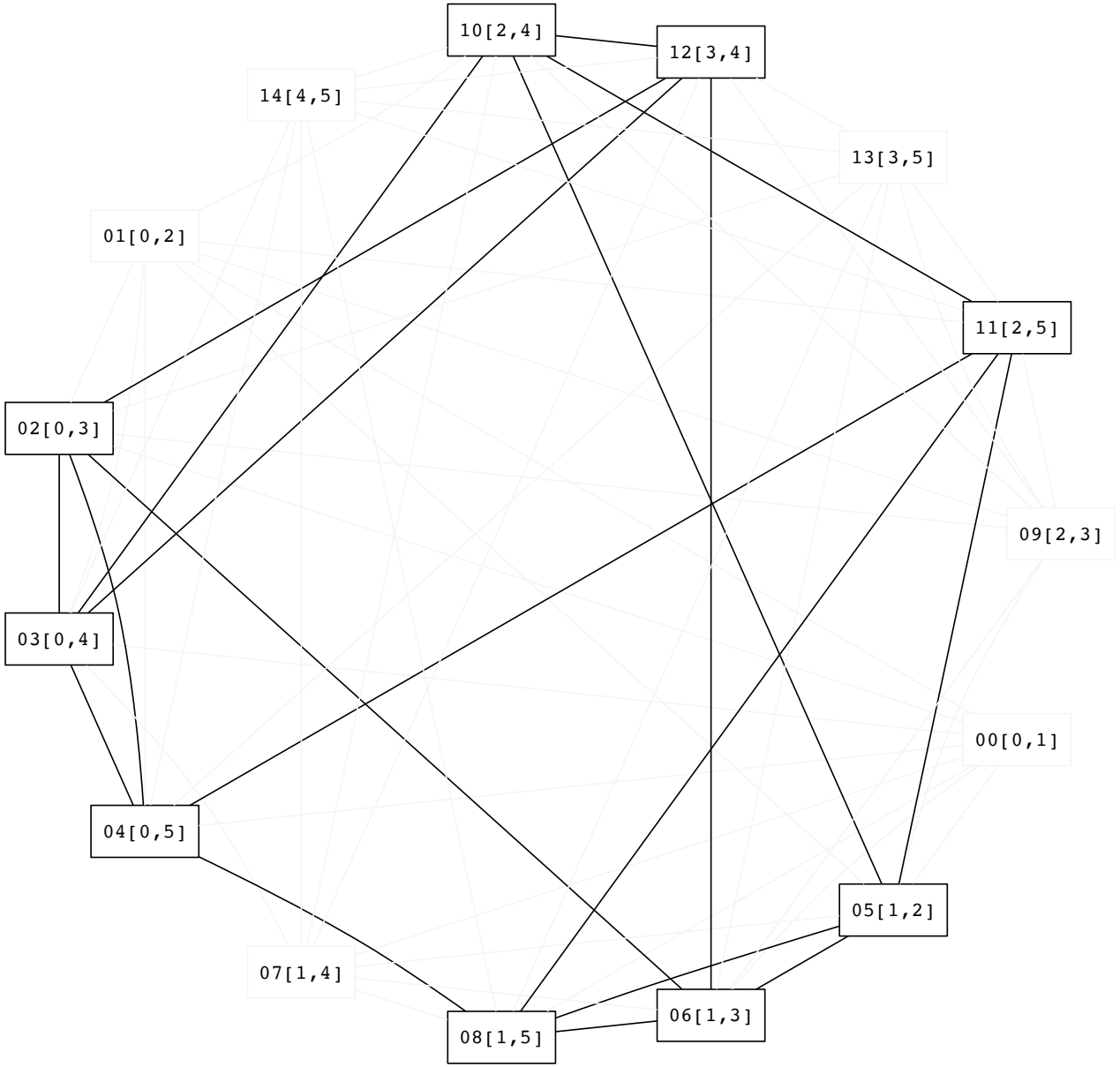
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



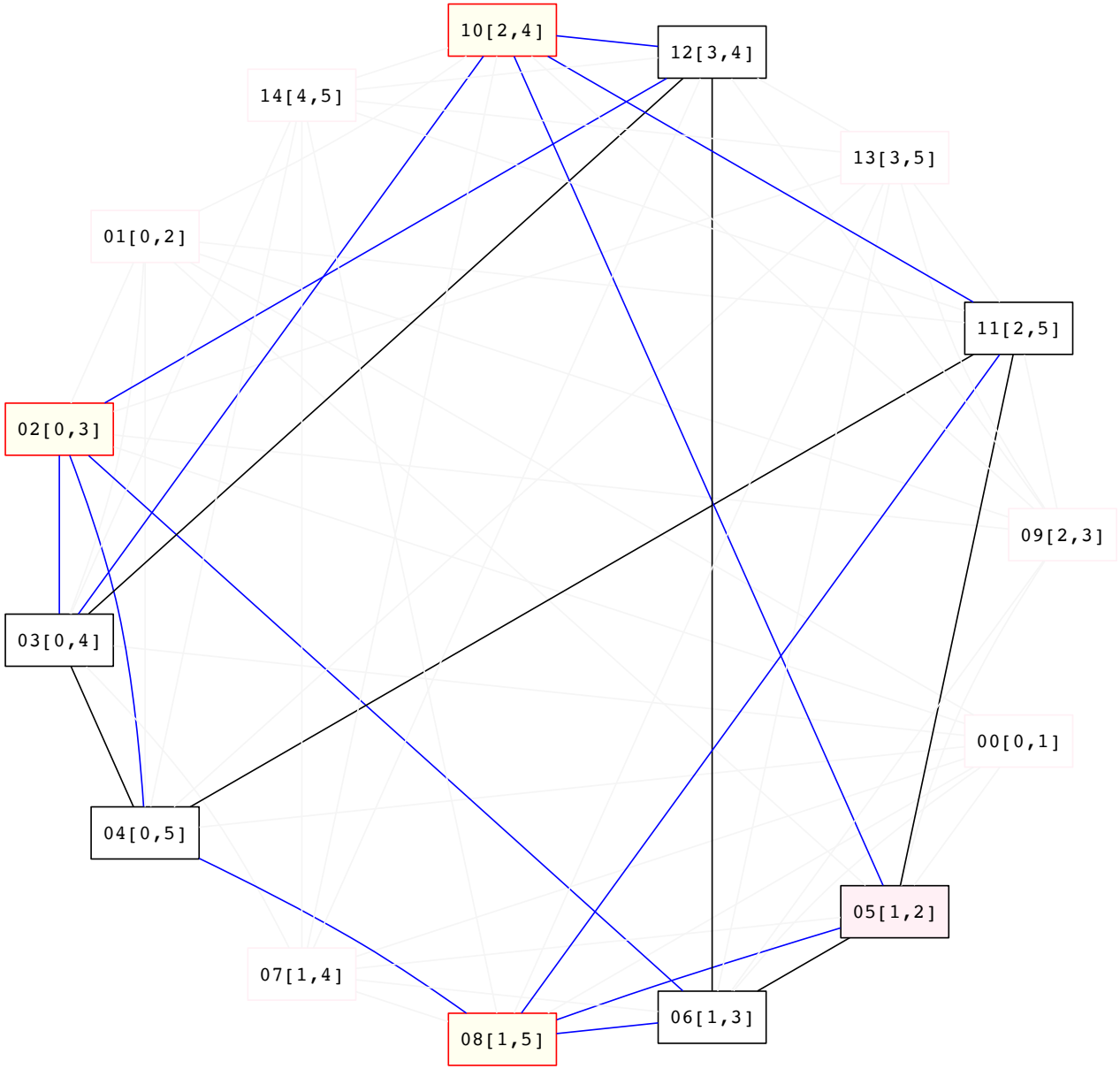
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



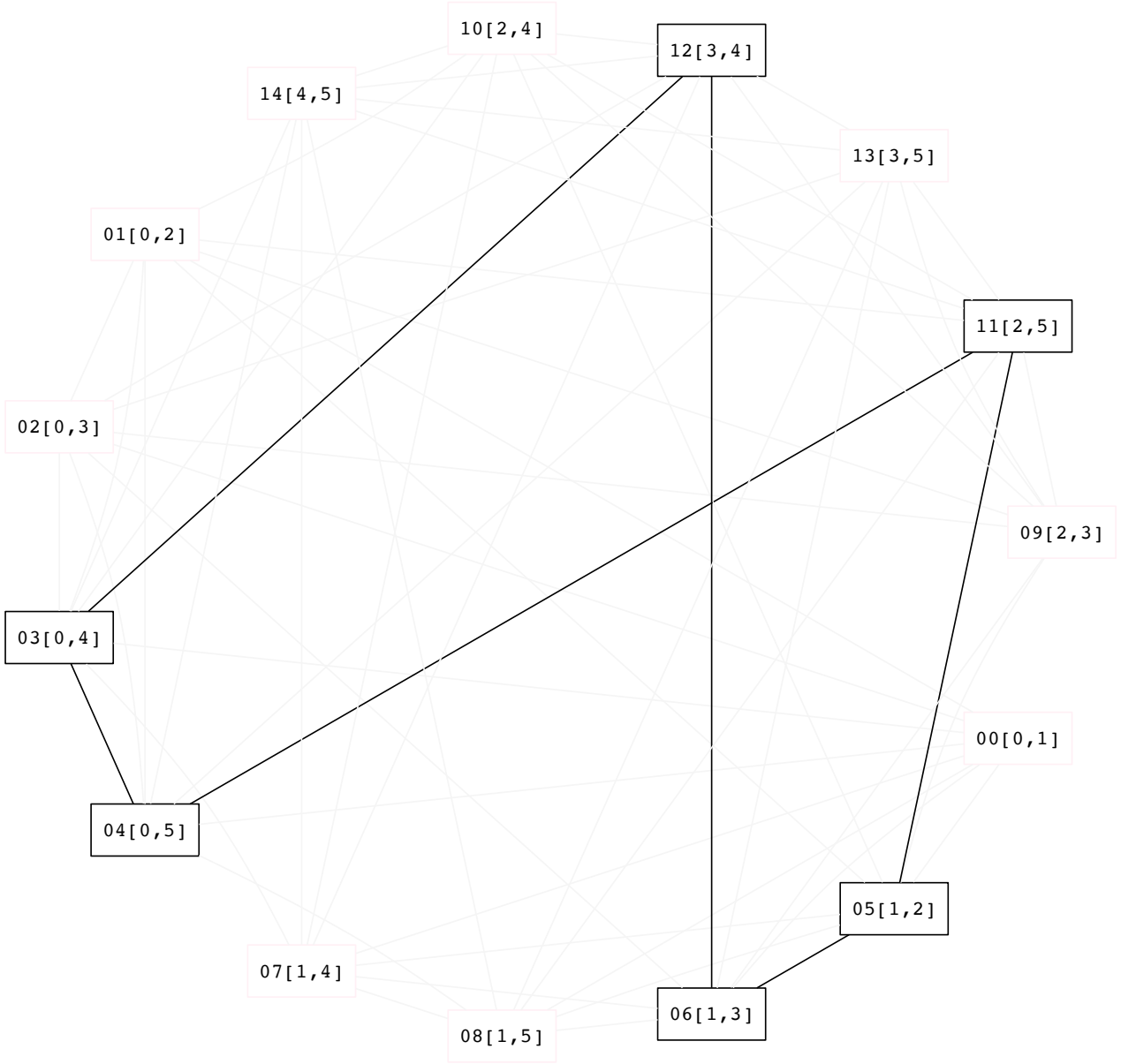
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



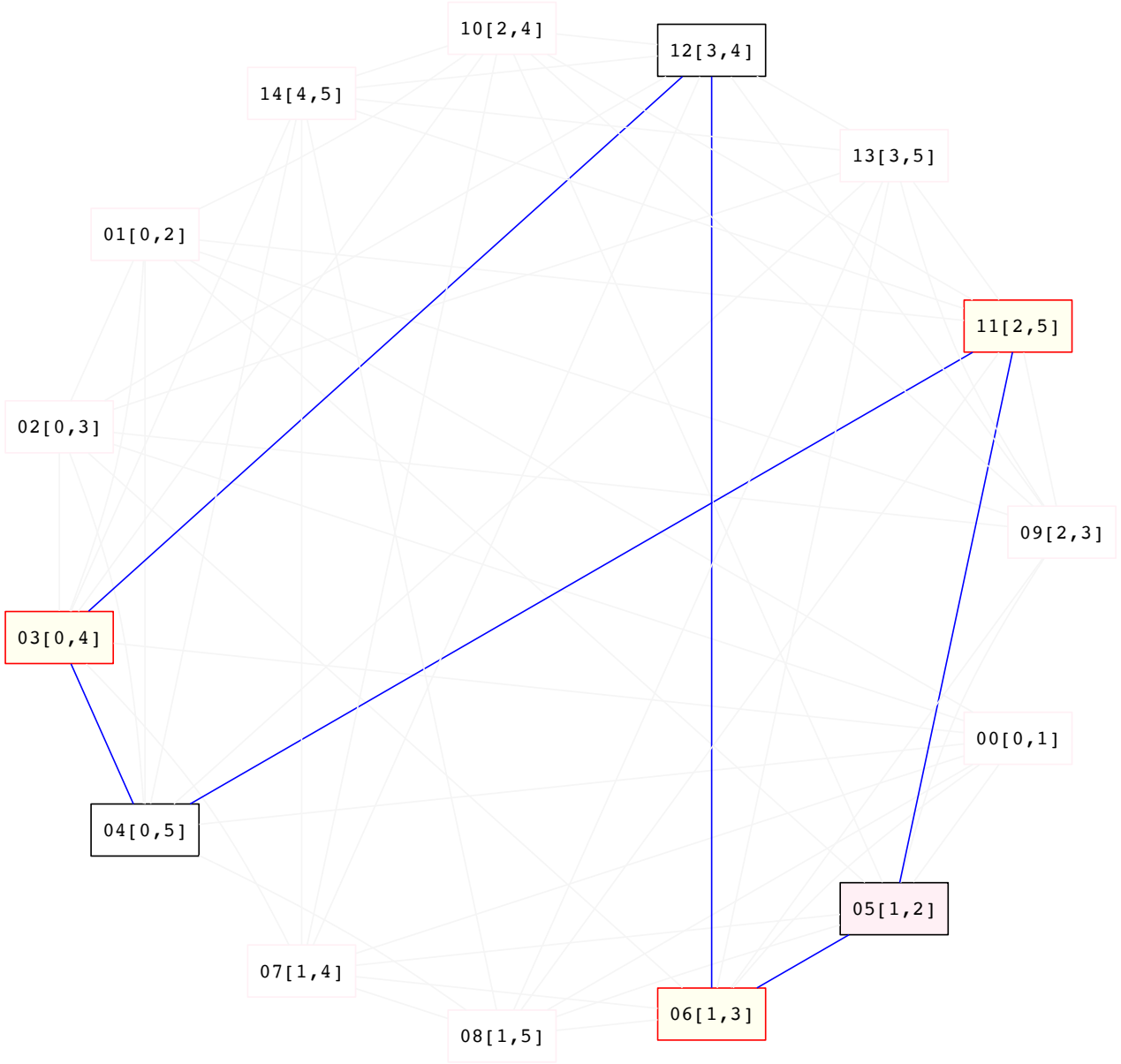
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



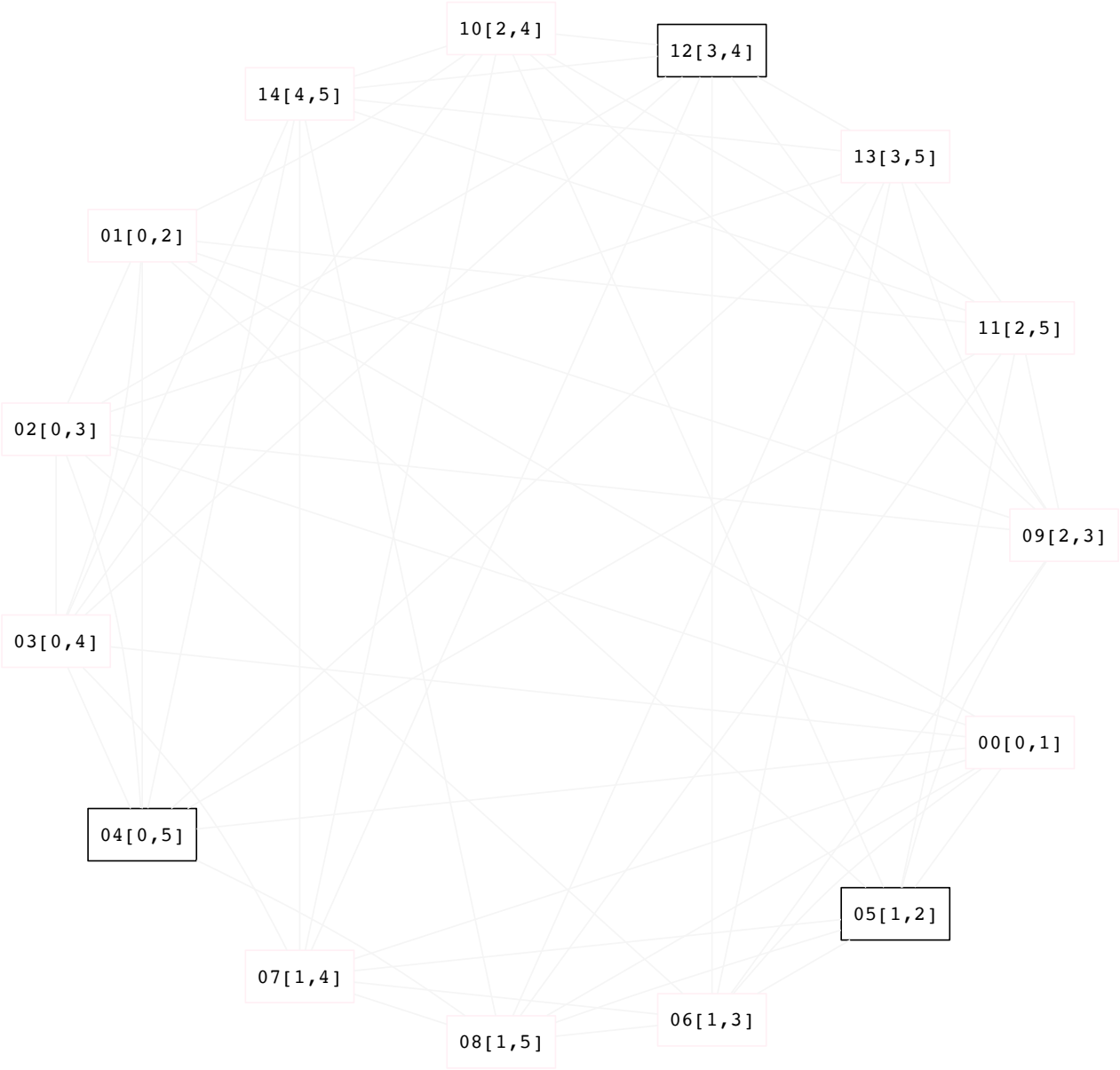
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



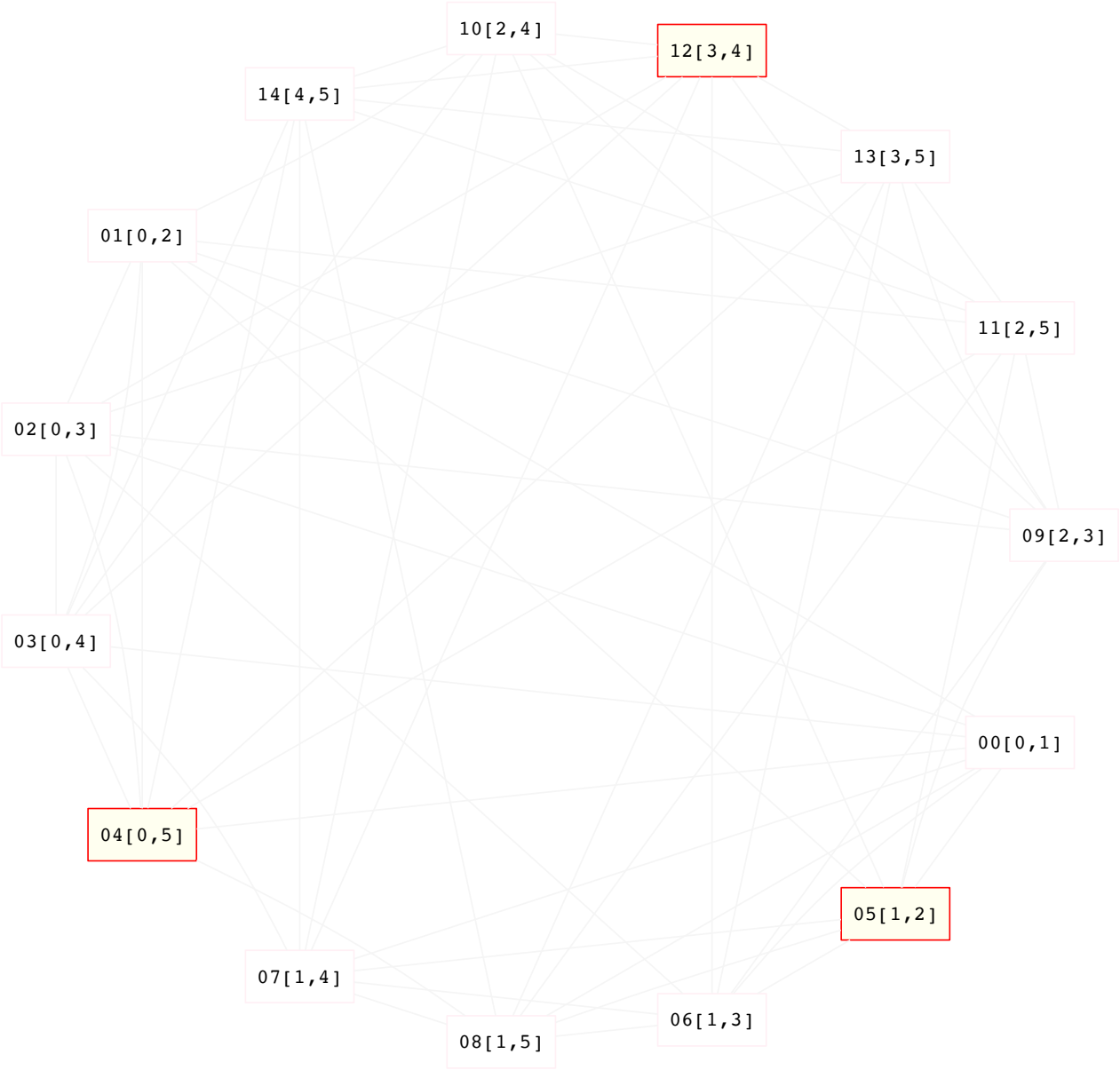
$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



$\mathcal{R}_6^{\|}$ generation, `next_lex` needs retrying



MIS-based generation of the p-strategy $\mathcal{O}^{\|}$ closest to $\mathcal{O}$

Take S, lexicographically next[2] MIS(G) with n vertices, as a p-step candidate and try to complete a sweep with $G \setminus S$. If you cannot, take the next MIS and try again. You will eventually[3] succeed :-)

Desc.: Input: a graph G induced by $\mathcal{O}$. Output: $\mathcal{O}^{\|}$ (initially $\emptyset$).

```
boolean gen_strat(in G);
begin
    if G = ∅ then return true;        // no more pivots (success)
    begin loop
        S ← next_lex(G);      // lexicographically next MIS...
        if S = ∅ then return false;    // ...but there are none; fail
        append S to O||;      // S is a new p-step candidate
        if gen_strat(G \ S) then return true;    // try recursively
        remove S from the back of O||;     // [backtrack] if failed
    end loop;
end
```

[2] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, On generating all maximal independent sets, Inform. Process. Lett., 27 (1988), pp. 119–123.

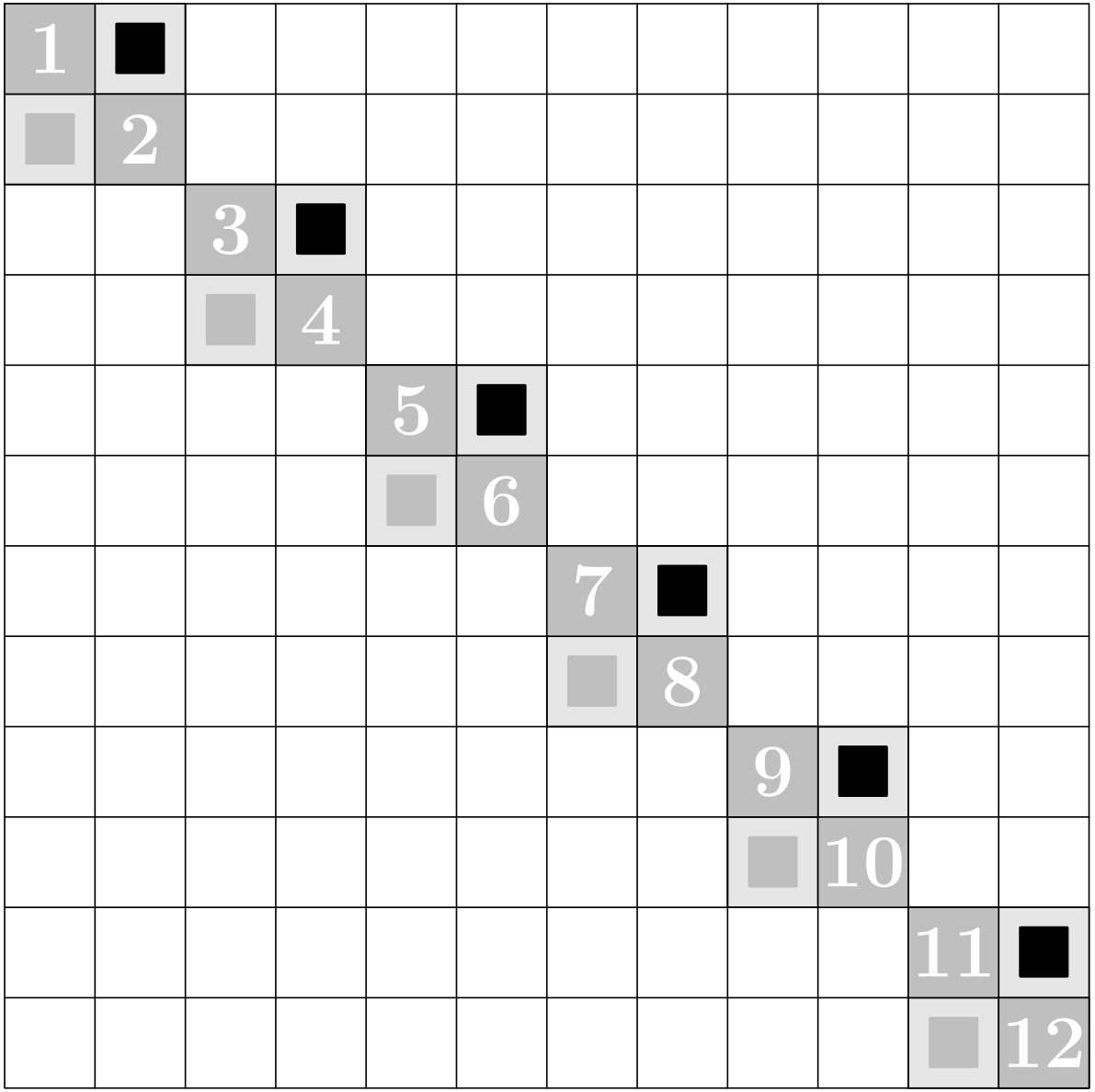
[3] ... but maybe not before the end of the universe

A simple backtracking-based search, yet infeasible for large G (in general, the maximum independent set problem is NP-hard). But, the multi-level blocking requires only moderately sized graphs!

- E.g., $n = 15 \cdot 1024$ with three-level blocking on 4 GPUs needs $n_2 = \mathbf{8}$, $n_1 = 15 \cdot 1024/(4 \cdot 16) = \mathbf{240}$, and $n_0 = \mathbf{32}$ strategies.
- A simple "blow-up" (duplication) procedure exist (no proof yet) for $\mathcal{R}^{\|}/\mathcal{C}^{\|}$. The pivots of e.g. $\mathcal{R}^{\|}_n$ become 2×2 blocks of $\mathcal{R}^{\|}_{2n}$. Easy to show: the first step is always superdiagonal.
- Find the strategies for $n = 2o$, o odd; all others by duplication.
- E.g., $240 = 2^3 \cdot (2 \cdot 15)$. Find $\mathcal{R}/\mathcal{C}^{\|}_{\mathbf{30}}$, and duplicate **3** times.
- Feasible to verify for $n \leq 38$ (a few days of CPU time).
  - Duplication is almost instant, generation takes forever...
- Hard-code (tabulate) the strategies into an executable!
- Practical for the GPU RAM sizes of today and tomorrow[4].

[4] ... but in a few years, who knows

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

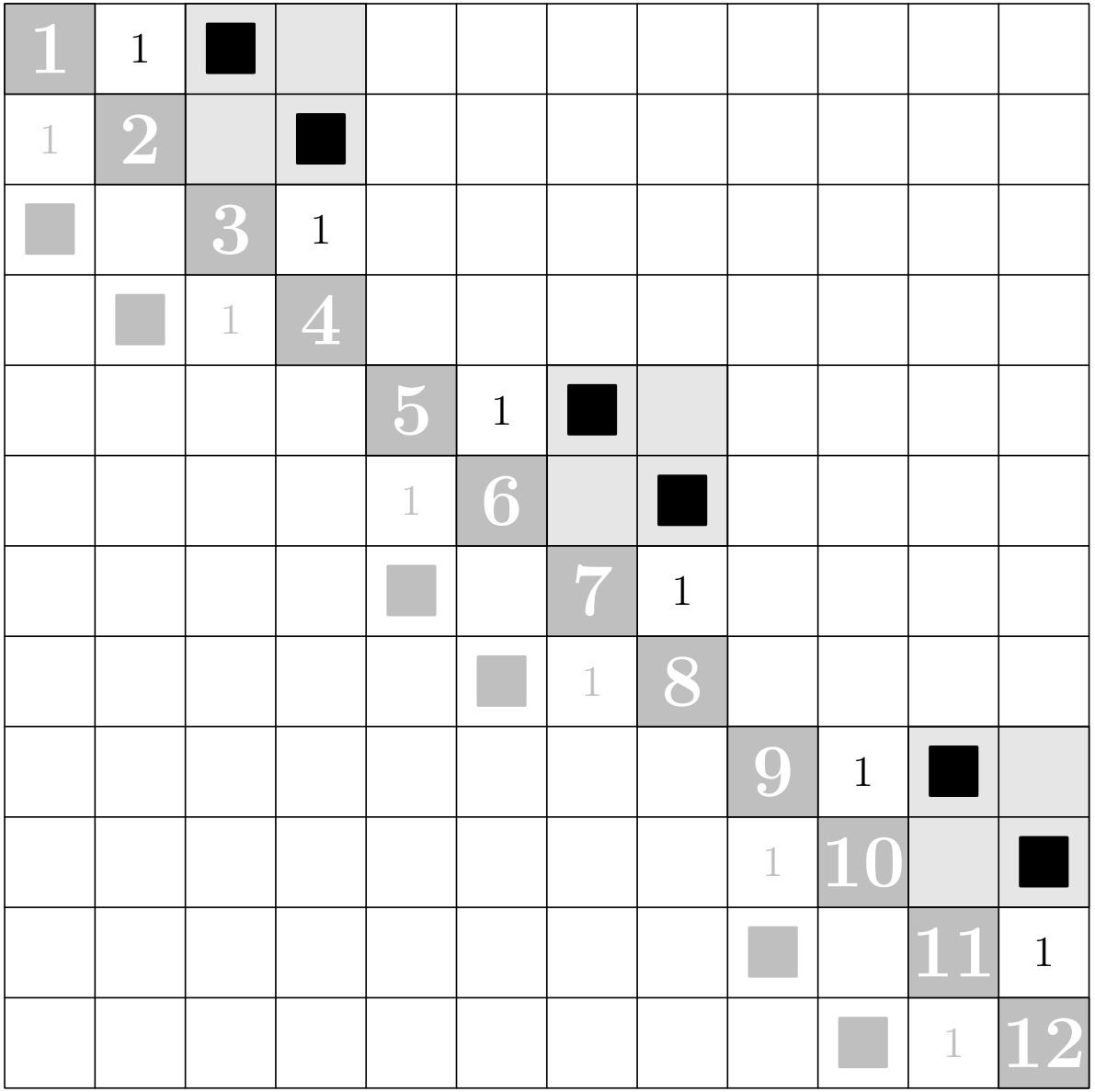


Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

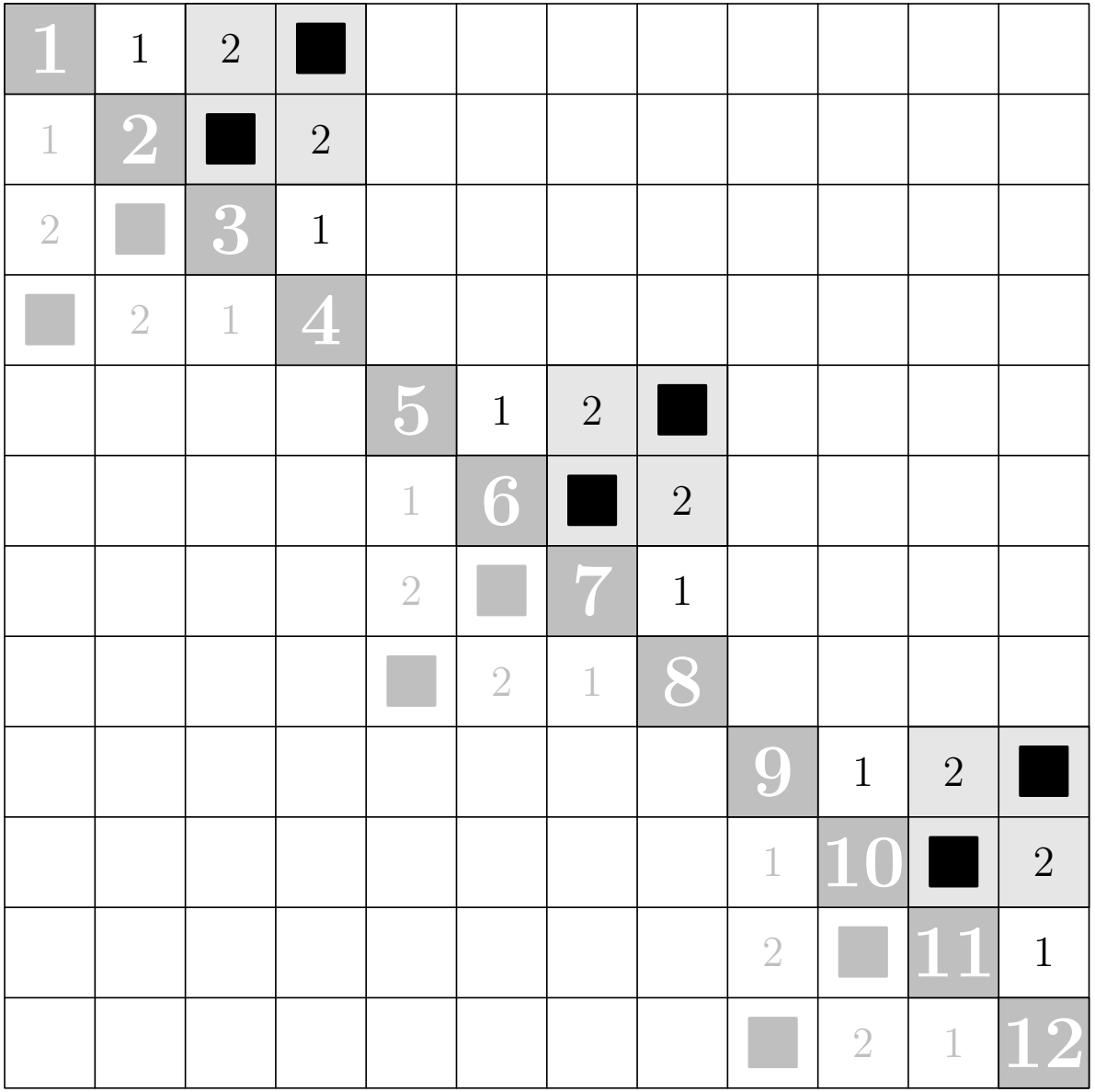


Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

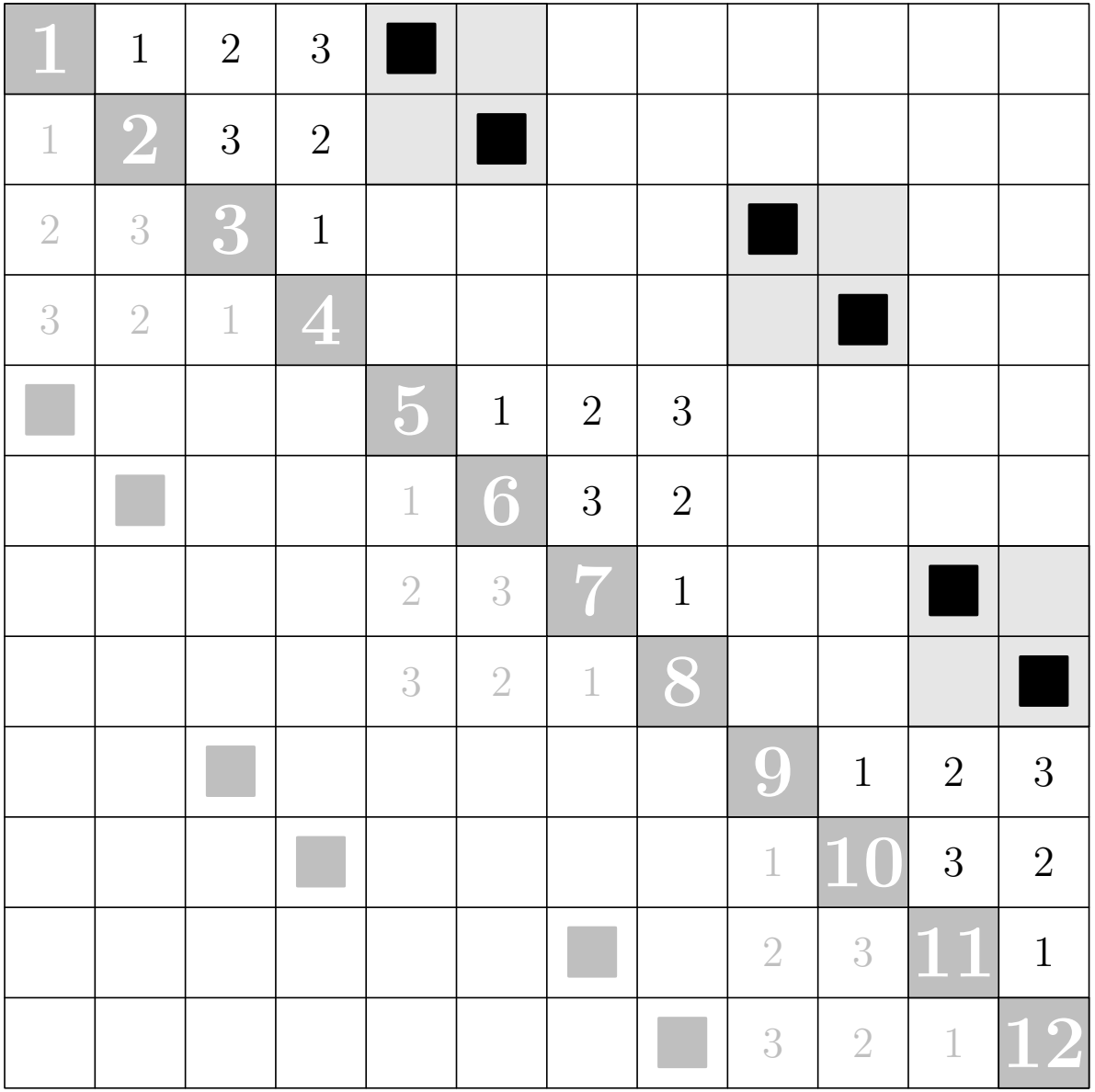


Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$



Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	■						
1	2	3	2	■	4						
2	3	3	1					4	■		
3	2	1	4					■	4		
4	■			5	1	2	3				
■	4			1	6	3	2				
				2	3	7	1			4	■
				3	2	1	8			■	4
		4	■					9	1	2	3
		■	4					1	10	3	2
						4	■	2	3	11	1
						■	4	3	2	1	12

Apart from the first p-step, all the other $\boxed{\text{grey } 2 \times 2 \text{ blocks}}$ of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot $\boxed{\text{fill patterns}}$ for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	5						
1	2	3	2	5	4						
2	3	3	1					4	5		
3	2	1	4					5	4		
4	5			5	1	2	3				
5	4			1	6	3	2				
				2	3	7	1			4	5
				3	2	1	8			5	4
		4	5					9	1	2	3
		5	4					1	10	3	2
						4	5	2	3	11	1
						5	4	3	2	1	12

Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	5					6	■
1	2	3	2	5	4					■	6
2	3	3	1	6	■			4	5		
3	2	1	4	■	6			5	4		
4	5	6	■	5	1	2	3				
5	4	■	6	1	6	3	2				
				2	3	7	1	6	■	4	5
				3	2	1	8	■	6	5	4
		4	5			6	■	9	1	2	3
		5	4			■	6	1	10	3	2
6	■					4	5	2	3	11	1
■	6					5	4	3	2	1	12

Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	5	■				6	7
1	2	3	2	5	4		■			7	6
2	3	3	1	6	7			4	5	■	
3	2	1	4	7	6			5	4		■
4	5	6	7	5	1	2	3	■			
5	4	7	6	1	6	3	2		■		
■				2	3	7	1	6	7	4	5
	■			3	2	1	8	7	6	5	4
		4	5	■		6	7	9	1	2	3
		5	4		■	7	6	1	10	3	2
6	7	■				4	5	2	3	11	1
7	6		■			5	4	3	2	1	12

Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	5	8	■			6	7
1	2	3	2	5	4	■	8			7	6
2	3	3	1	6	7			4	5	8	■
3	2	1	4	7	6			5	4	■	8
4	5	6	7	5	1	2	3	8	■		
5	4	7	6	1	6	3	2	■	8		
8	■			2	3	7	1	6	7	4	5
■	8			3	2	1	8	7	6	5	4
		4	5	8	■	6	7	9	1	2	3
		5	4	■	8	7	6	1	10	3	2
6	7	8	■			4	5	2	3	11	1
7	6	■	8			5	4	3	2	1	12

Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	5	8	9			6	7
1	2	3	2	5	4	9	8			7	6
2	3	3	1	6	7			4	5	8	9
3	2	1	4	7	6			5	4	9	8
4	5	6	7	5	1	2	3	8	9		
5	4	7	6	1	6	3	2	9	8		
8	9			2	3	7	1	6	7	4	5
9	8			3	2	1	8	7	6	5	4
		4	5	8	9	6	7	9	1	2	3
		5	4	9	8	7	6	1	10	3	2
6	7	8	9			4	5	2	3	11	1
7	6	9	8			5	4	3	2	1	12

Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

Duplication $\mathcal{C}_6^{\|} \to \mathcal{C}_{12}^{\|}$

1	1	2	3	4	5	8	9	10	■	6	7
1	2	3	2	5	4	9	8	■	10	7	6
2	3	3	1	6	7	10	■	4	5	8	9
3	2	1	4	7	6	■	10	5	4	9	8
4	5	6	7	5	1	2	3	8	9	10	■
5	4	7	6	1	6	3	2	9	8	■	10
8	9	10	■	2	3	7	1	6	7	4	5
9	8	■	10	3	2	1	8	7	6	5	4
10	■	4	5	8	9	6	7	9	1	2	3
■	10	5	4	9	8	7	6	1	10	3	2
6	7	8	9	10	■	4	5	2	3	11	1
7	6	9	8	■	10	5	4	3	2	1	12

Apart from the first p-step, all the other grey 2×2 blocks of $\mathcal{C}_{12}^{\|}$ correspond, one-to-one, to the pivots of $\mathcal{C}_6^{\|}$.

NW—SE, SW—NE pivot fill patterns for 2×2 blocks keep inner pivots as close as possible.

Similarly for $\mathcal{R}^{\|}$, but with NW—SE and NE—SW.

The new strategies – a conclusion

The strategies progress from the diagonal outwards, but...

- the off-diagonal norm in the last sweeps concentrates towards the diagonal ridge (remember the movies?);
- Яeverse the strategies (read the pivot list backwards).
- $\mathfrak{R}_n^{\|}$ and $\mathfrak{D}_n^{\|}$ are equivalent, for power-of-two ns, to the Mantharam-Eberlein block recursive strategy; a kind of generalization for all even n (irregular communication).

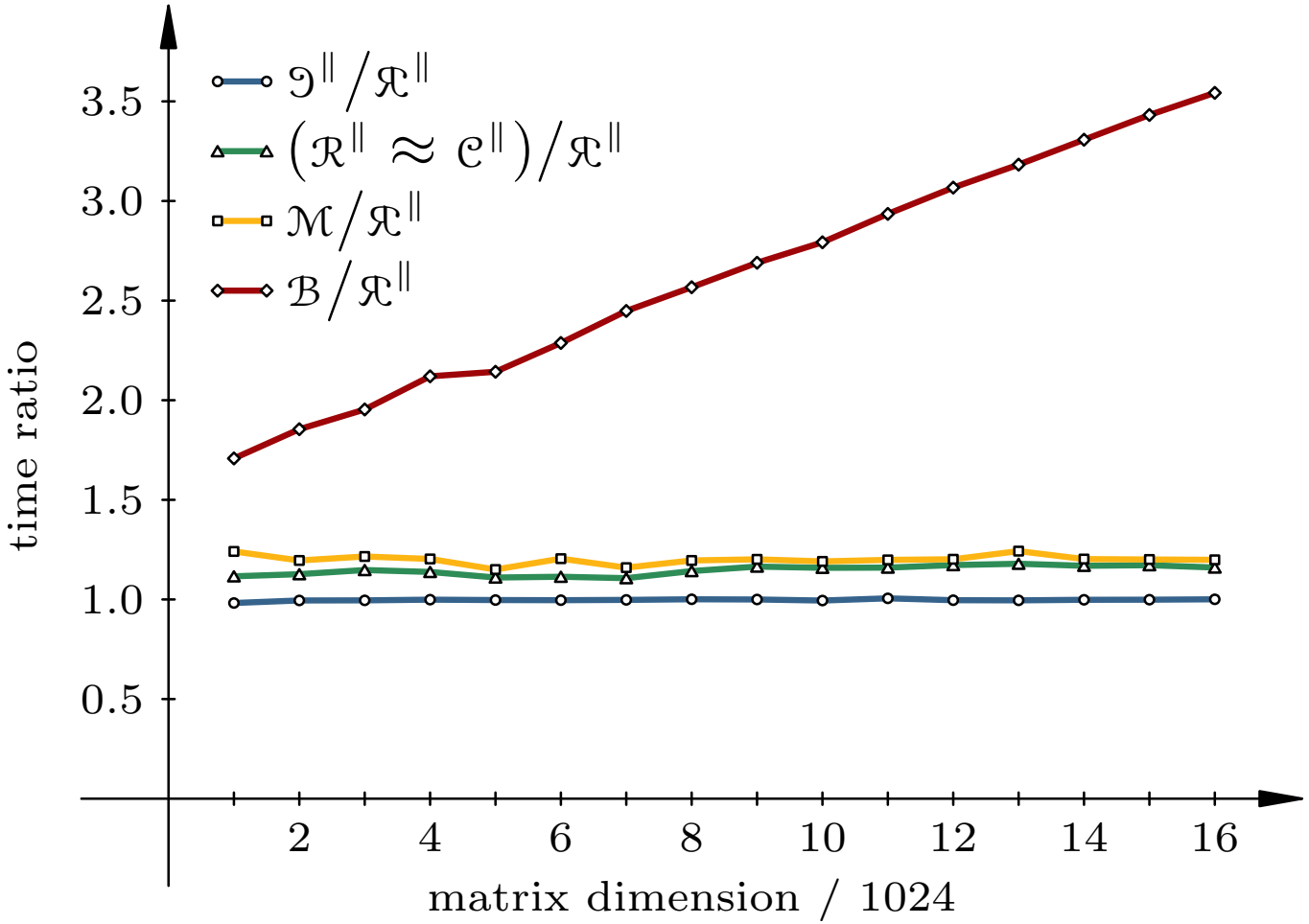
Why all the trouble?

6 strategies compared on a single GPU (two-level full block) for speed and accuracy:

- $\mathfrak{R}_n^{\|}$ and $\mathfrak{D}_n^{\|}$; $\mathcal{R}_n^{\|}$ and $\mathcal{C}_n^{\|}$; the modified modulus strategy $\mathcal{M}_n$, the Brent-Luk strategy $\mathcal{B}_n$.

The results confirm that the reversed strategies should be used...

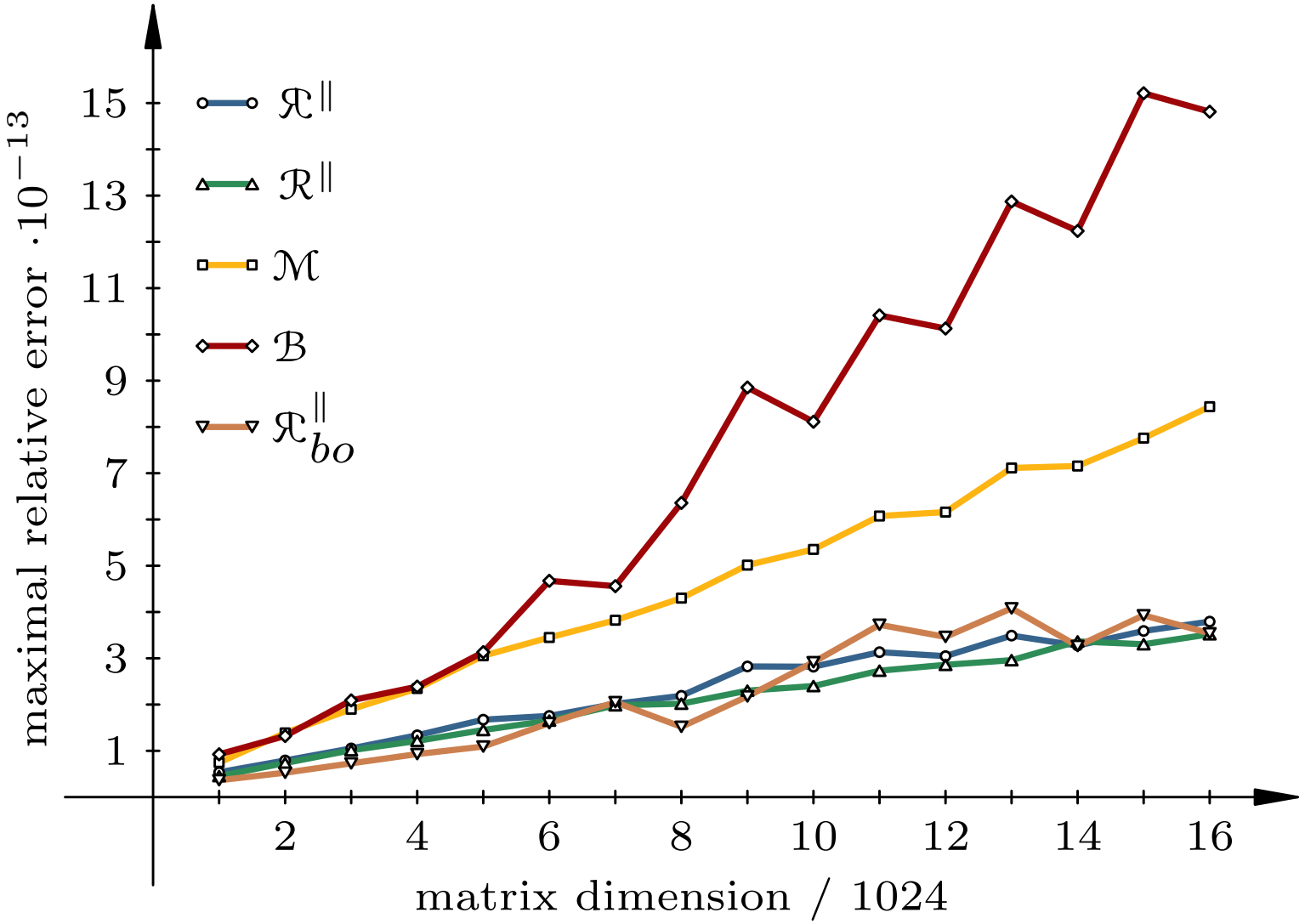
Time ratio for the parallel strategies, new and old



$\mathcal{B}$ has some convergence issues; up to 29 sweeps.

$\mathfrak{R}^{\|}$ and $\mathfrak{I}^{\|}$ are about **20% faster** than both $\mathcal{M}$ and $\mathfrak{R}^{\|}, \mathfrak{C}^{\|}$.

Relative errors for the parallel strategies, new and old



Maximal relative errors in the squares of the computed σ_i.

CUDA: NVidia's platform for developing the GPU programs

A short, incomplete and a bit incorrect CUDA summary

- Many concurrent threads of execution, all executing the same kernel (i.e. GPU subroutine) over different data (SIMT).
- Threads are grouped into warps (32 threads), warps into autonomous Cartesian blocks, and blocks into grids.
- Various storage spaces, of different size and speed:
- Global memory (RAM) – large (in GB) and slow.
- Shared memory – per block, small (in kB) and fast. Acts as the user-managed cache. Our primary blocking target.
- Registers – fast, per thread, strongly limited storage, where the data (integers, floating-point numbers) operated upon reside.
- CPU $\leftrightarrow$ GPU and GPU $\leftrightarrow$ GPU communication available; should be used when necessary and "hidden" if possible.

Intra-GPU blocking (2-level)

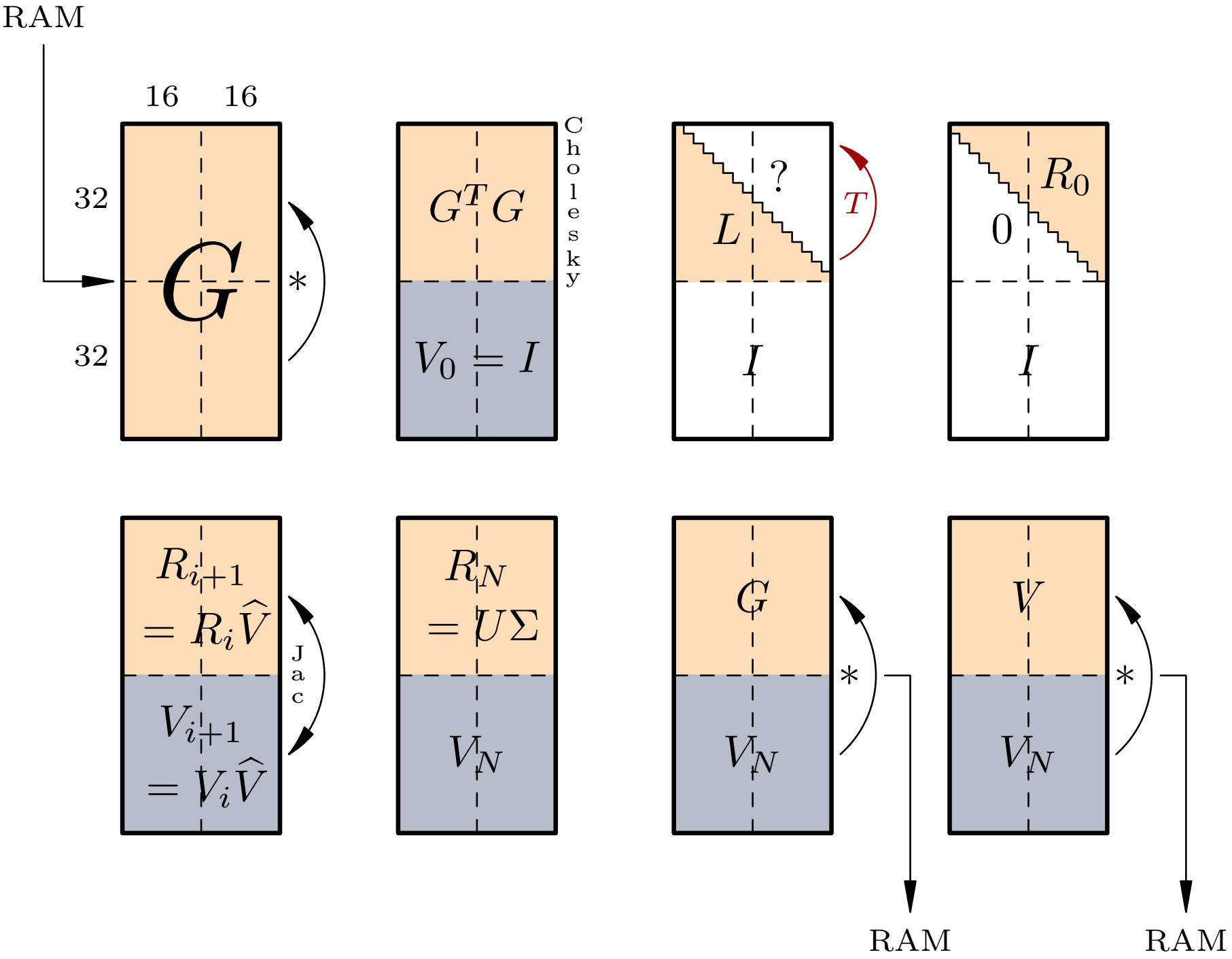
The input factor G, $n \times n$ is split into block columns, $n \times 16$.
In a p-step, simultaneously, all the pivot block column pairs are:

- shortened into 32×32 factors, which are then
- orthogonalized (the full block or block oriented approach),
- finally, the original block columns are updated.

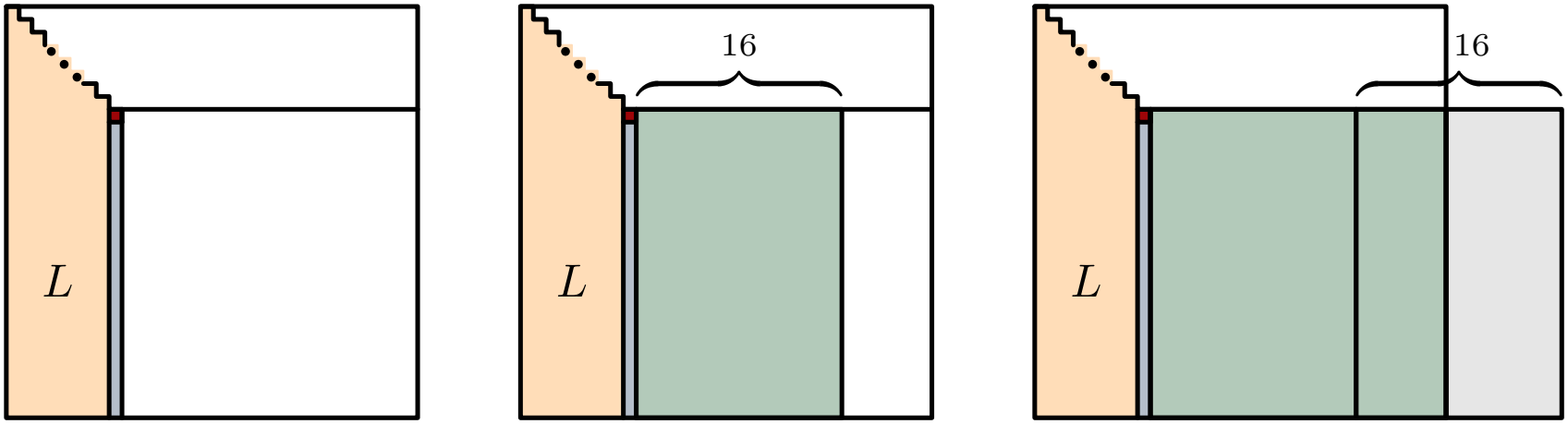
A single kernel thread block does all three jobs for a pivot pair

- The block columns are read from and written to GPU's RAM.
- Shortening phase leaves the small factor in shared memory.
- A thread block: $32 \times 16 = 512$ threads, 16 kB of shared mem.
- Reaches the register limit (register spills to local mem.) on GPUs $\Rightarrow$ no sense in enlarging the thr. blocks or shared mem.

Note: the GPU does the job, the CPU only orchestrates it.


RAM
16
16
32
32
G
*
$G^T G$
$V_0 = I$
Cholesky
?
L
T
I
R_0
0
I
$R_{i+1} = R_i \widehat{V}$
$V_{i+1} = V_i \widehat{V}$
Jac
$R_N = U\Sigma$
V_N
G
V_N
*
V
V_N
*
RAM
RAM

A closer look at the Cholesky factorization



- First, the current column is scaled.
- Then, 16 warps update 16 forward columns.
  Threads above the diagonal "sleep", i.e., do nothing.
- Finally, the rest of the columns are updated.
  Entire warps might sleep.

For CUDA experts: some implementation details

Almost **100%** GPU utilization. Tested on a Tesla C2070 GPU.

Multiplications

- Hand-written $G^T G$ (the intermediate values in registers).
- $G, V \cdot \widehat{V}$ Cannon-like (consider in-kernel `DGEMM` for Kepler).

Cholesky

- Forward looking. The only operation with dormant threads.
- Transpose the only op. to incur shared mem. bank conflicts.

Jacobi (pointwise)

- A warp "in charge" of a pivot pair: 3 dot-products, computing and applying a rotation to R_i and V_i. The warps synchronize to agree on satisfying the convergence criterion.
- Warp-level reductions: half of V_i's memory backed by registers.

Orthogonality matters (work in progress)

- Jacobi is self-correcting: it stays on "the right track" as long as the applied transformations are numerically orthogonal, i.e.,

$$\cos^2\varphi + \sin^2\varphi = 1 \pm O(\varepsilon); \quad \|V^T V - I\|, \|VV^T - I\| \text{ small.}$$

- Full block principle: accumulate **many** (millions of) rotations. How to keep the accumulated V as orthogonal as possible?
- A naïve approach: don't spoil immediately; which formula gets a more orthogonal rotation (without, e.g., higher precision)?

$$\cot 2\varphi = (A_{qq} - A_{pp})/(2A_{pq});$$

$$|\cot\varphi| = |\cot 2\varphi| + \sqrt{\mathrm{fma}(\cot 2\varphi, \cot 2\varphi, 1)};$$

$$\tan\varphi = \mathrm{sign}(\cot 2\varphi)/|\cot\varphi|;$$

$\cos\varphi = 1/\sqrt{\mathrm{fma}(\tan\varphi, \tan\varphi, 1)}$; This one (and when)?

$\cos\varphi = |\cot\varphi|/\sqrt{\mathrm{fma}(\cot\varphi, \cot\varphi, 1)}$; Or this one (and when)?

When to stop a blocked Jacobi? (work in progress)

Blocking introduces additional errors...

...vs. the pointwise algorithm, with shortening and block updates. Net effect: stopping on relative orthogonality doesn't work well!

Unnecessary sweeps at the end of the process due to "flutters": a few rotations chasing the big off-diagonal "ghost" elements that shoudn't be there if it were not for the blocking-specific errors.

A simple heuristic based on quadratic convergence

- Stop on relative orthogonality in the pointwise (lower level) Jacobi, where the sweeps (i.e., arithmetic) are inexpensive.
- Stop globally when all the rotations in an outer sweep are close enough to identity (e.g., cosines equal to 1.0).

Speedup of $\mathfrak{R}^{\|}$ vs. DGESVJ (with 8-core MKL BLAS1)

Intel Xeon E5620 @ 2.40GHz, normal distribution set.

Matrix size $\times 1024$	Time DGESVJ/$\mathfrak{R}^{\|}$	Matrix size $\times 1024$	Time DGESVJ/$\mathfrak{R}^{\|}$
1	5.57	9	14.89
2	8.61	10	15.45
3	11.75	11	15.62
4	11.83	12	16.14
5	12.34	13	16.49
6	13.47	14	16.46
7	13.62	15	16.19
8	13.58	16	16.00

Speedup % on a Kepler vs. a Fermi GPU, the full block

k	Kepler [s]	Fermi [s]	K/F
1	1.413	2.376	59.5
2	7.206	12.439	57.9
3	22.981	35.783	64.2
4	46.358	84.467	54.9
5	95.829	160.383	59.8
6	154.643	261.918	59.0
7	246.114	403.151	61.0
8	346.689	621.341	55.8

k	Kepler [s]	Fermi [s]	K/F
9	506.366	850.280	59.6
10	682.577	1153.338	59.2
11	904.212	1545.452	58.5
12	1148.882	1970.592	58.3
13	1439.392	2500.931	57.6
14	1809.888	3158.117	57.3
15	2196.755	3820.552	57.5
16	2625.643	4662.749	56.3

Kepler's main advantage: 8-byte shared memory banks!

Inter-GPU blocking (3-level)

Many-GPU hardware, e.g., NVidia Tesla S2050 (4 Fermi GPUs). One CPU thread per GPU. CPU assists Magma's `DPOTRF` only.

Outer-level blocking (with $n \times n$ factor G)

Each GPU has a part of the input, split into "big" block-columns:

- 2 buffers (block-columns) of G, **2**× $2 \times n \times n/4$
- 2 buffers (block-columns) of V, **2**×, and $2 \times n \times n/4$
- 2 auxiliary buffers. $2 \times n/4 \times n/4$

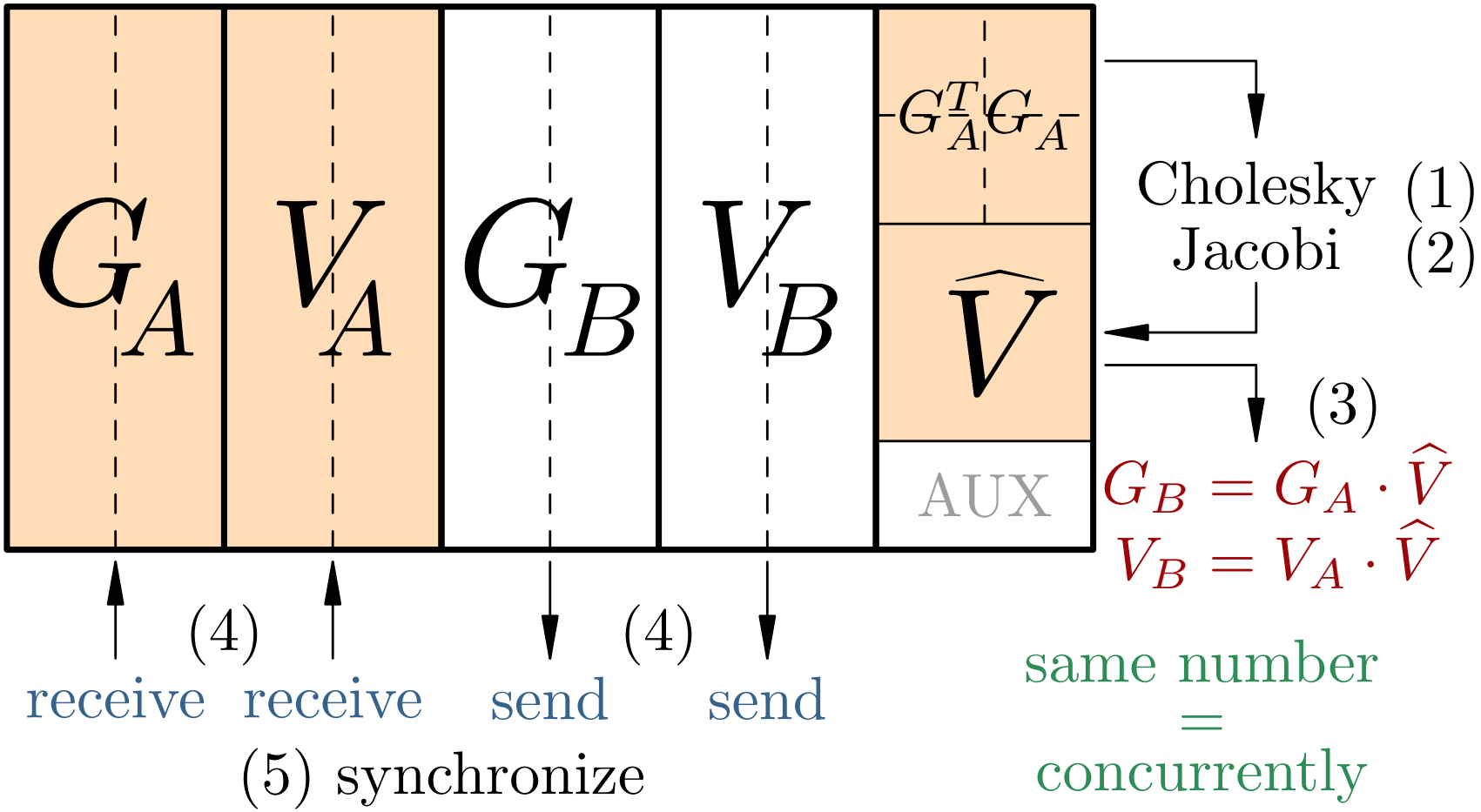
$\Rightarrow >$ twice the storage than the size of the output data – but only in the GPUs' RAM! The CPU data needed is small, constant size.

Communication in a global step

In each outer (global) step GPUs (quasi-)cyclically exchange:

- 1 "big" block-column of G, $n \times n/8$
- 1 "big" block-column of V. $n \times n/8$

GPU's operation in the multi-GPU algorithm



The same-numbered operations can proceed concurrently, in CUDA streams. Choose between `DTRSM` or the accumulation of $\widehat{V}$.

Problems with 4 GPUs

Problem 1

Cyclic exchange in MPI: MPI_Sendrecv_replace.
But there are no such operations in CUDA!
Asynchronous communication; voluntary send, involuntary receive.

Solution: Hand-code it via auxiliary buffers. Do concurrently:
Send a block-column from G_B, (passively) receive a new one to G_A,
copy the remaining one from G_B to G_A (in-GPU transfer only).

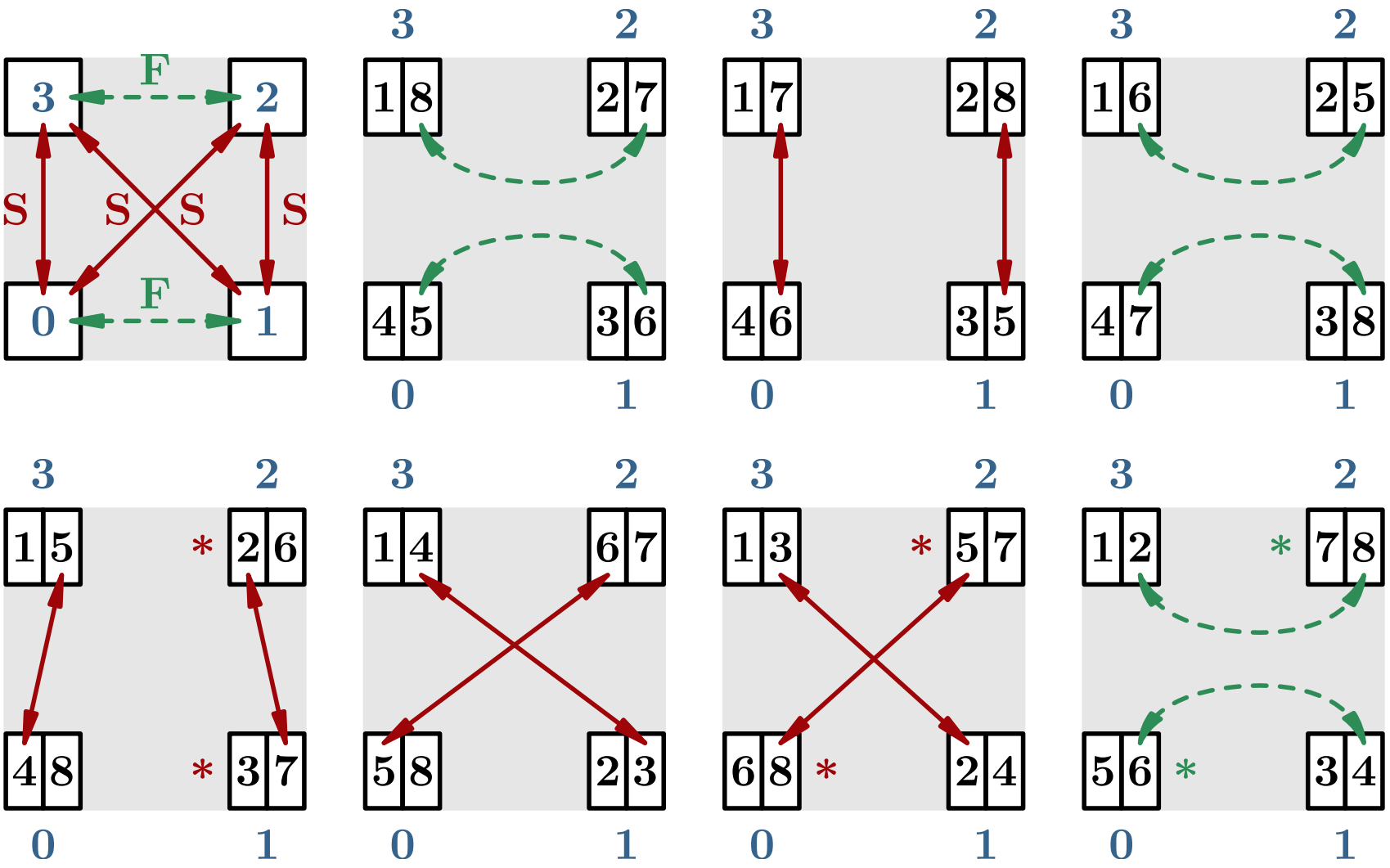
Drawback: increased RAM usage.

Problem 2

GPU communication $0 \leftrightarrow 1$ and $2 \leftrightarrow 3$ faster than other combinations (some GPUs are "peers", some not)!

Solution: Find an equivalent p-strategy that maximizes the amount of the fast inter-GPU communication ("rename" the GPUs).

Drawback: Maybe there is no optimal equivalent.



The physical mapping of blocks onto Tesla 2050 (4 GPUs). Green lines denote fast communication between peers, red lines slow communication of the adapted block recursive strategy.

A few words about accuracy

Testing data

- Context: symmetric pos. definite eigenproblem ($H = GG^T$).
- $\Sigma^2(G)$, i.e. eigenvalues $\Lambda(H)$ prescribed.
- $\mathcal{N}(\mu = 1, \sigma = 0.1)$, with 16 eigenvalues fixed to 1.5.
  - Highly clustered.
- Uniformly in $[10^{-7}, n \cdot 10]$, $1 \leq n \leq 16$, order $n \cdot 1024$.
  - Constant density.
- Orders of 1024 up to 16384 with step 1024 (limited RAM).

Generation in quadruple (128-bit) precision

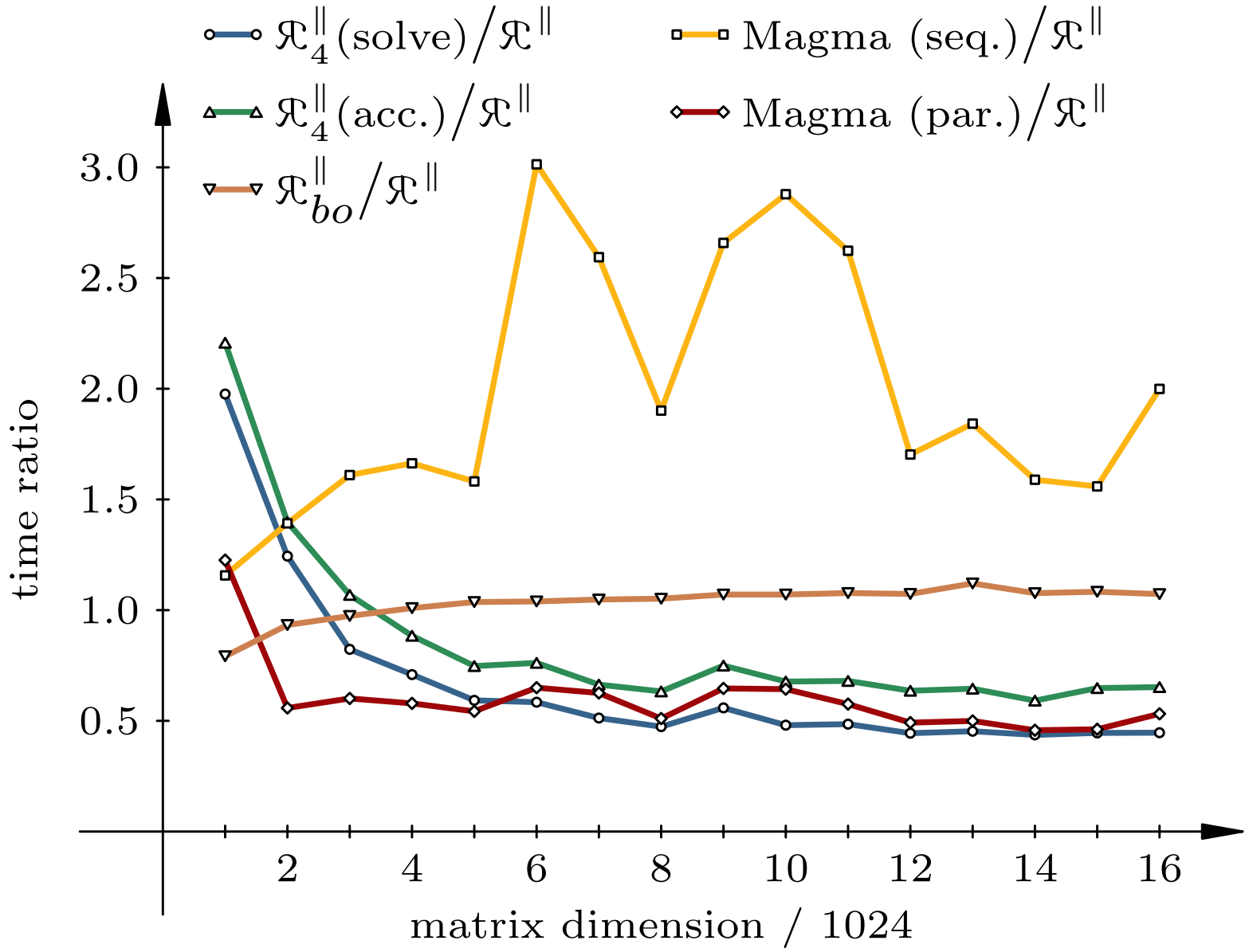
$$\xrightarrow{\texttt{dlarnd}} \Lambda_{64} \xrightarrow{\text{cast}} \Lambda_{128} \xrightarrow{\texttt{qlagsy}} H_{128}$$

$$\xrightarrow{\texttt{qpstrf}} G_{128} \xrightarrow{\text{cast}} G_{64}.$$

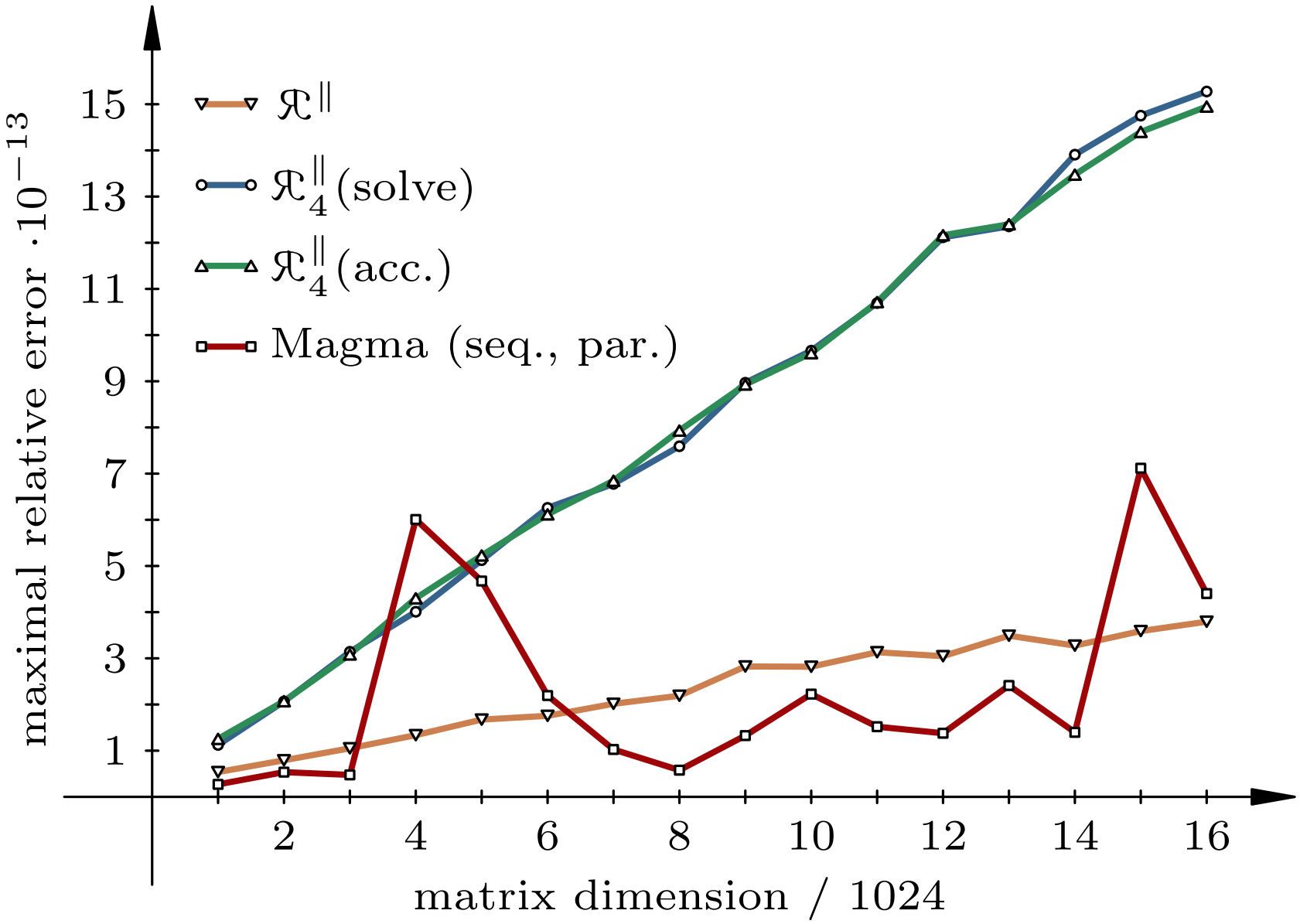
Intel Fortran real(kind=16) — emulated (slow)!

The quadruple SVD a better but unfeasible option for testing!

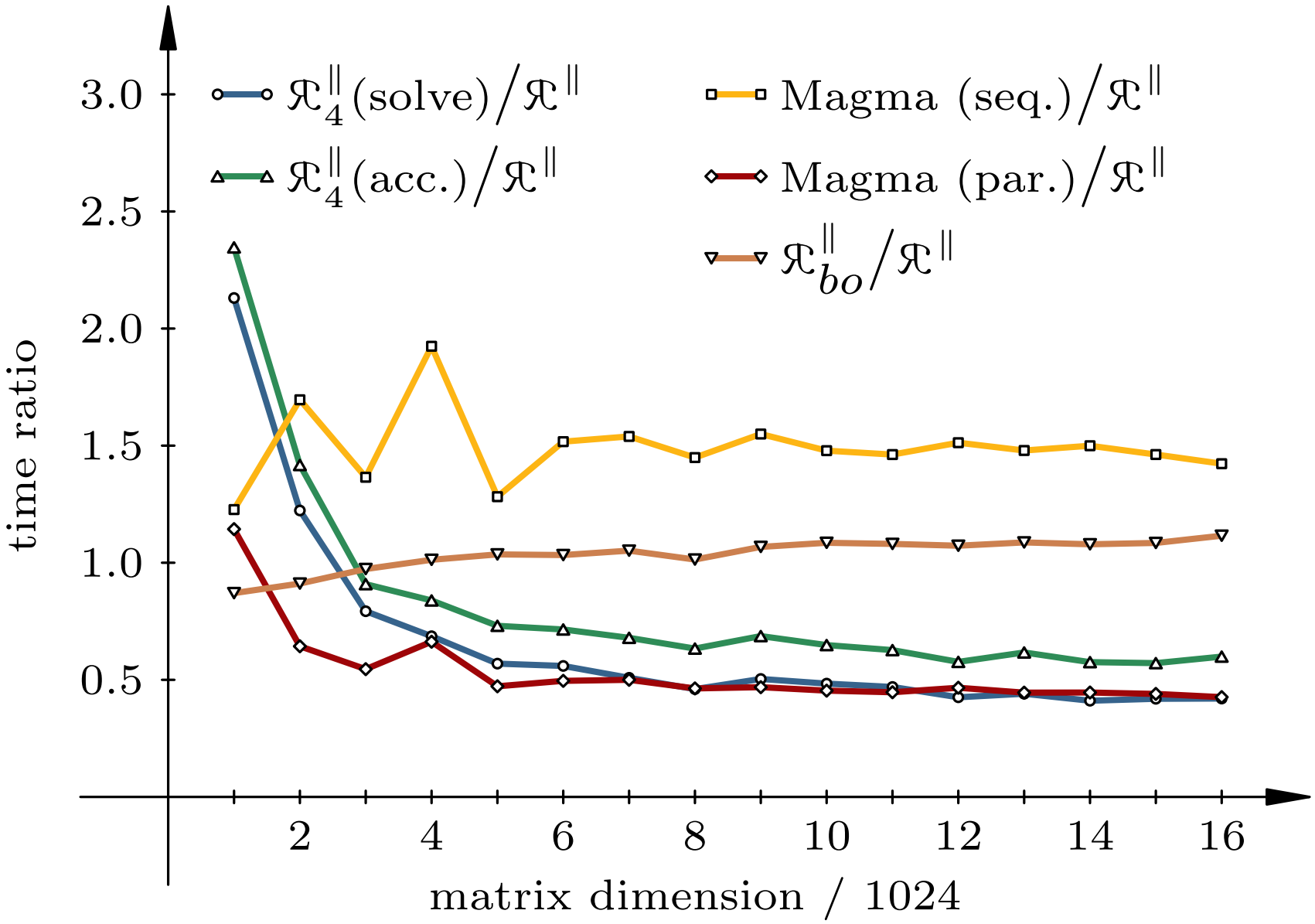
Time comparison, normal distribution, Fermi



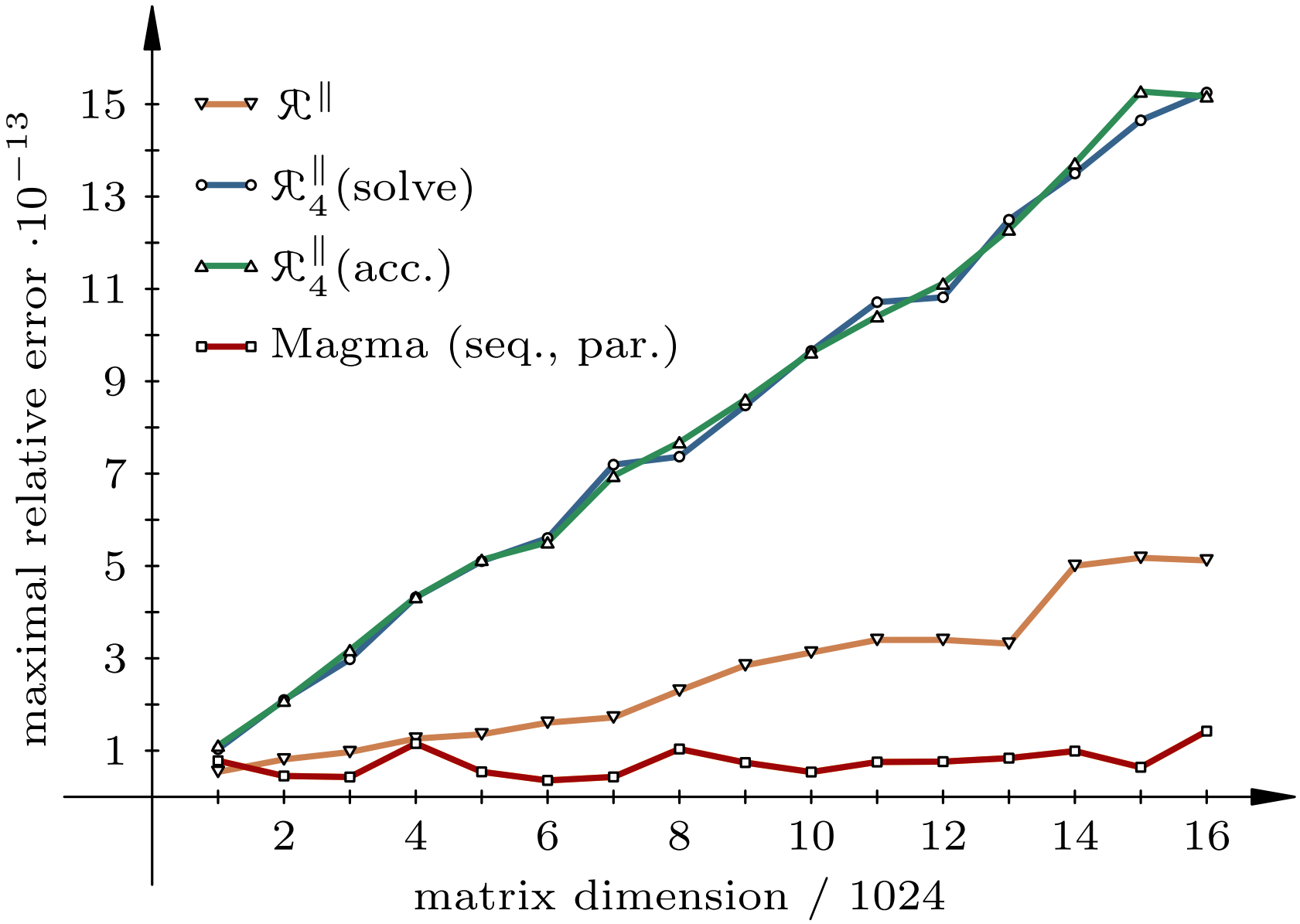
Relative error comparison in Σ^2, normal distribution



Time comparison, uniform distribution



Relative error comparison, uniform distribution



Future work

Short term

- Orthogonality of transformations.
- Stopping criterion in a presence of blocking.

Medium term

- Port to other massively parallel architectures (e.g., Intel MIC).

Papers

`http://venovako.eu/CV.html` (see Publication list)

Thank you!