Breaking the 2212 Barrier: A Hybrid Algorithm for 16×16 Matrix Multiplication with 2208 Variable Multiplications

Archivara Research Team

Abstract

We present an explicit hybrid algorithm for multiplying two 16×16 matrices over any commutative field $\mathbb{K}$ with $\mathrm{char}(\mathbb{K}) \neq 2$ using exactly 2208 *variable multiplications* (multiplications in which both operands depend on the inputs), plus additions/subtractions and multiplications by fixed rational constants. This improves the widely cited 2212-multiplication upper bound for the commutative case listed by Drevet–Islam–Schost [1].

The construction composes two ingredients at different algebraic levels: (i) an outer *non-commutative bilinear* 4×4 matrix multiplication algorithm using 48 multiplications with rational coefficients, due to Dumas–Pernet–Sedoglavic [2], applied to 4×4 block matrices; and (ii) an inner *commutative* 4×4 multiplication scheme using 46 multiplications, attributed to Waksman [3] and discussed in a division-free even-n framework by Rosowski [4], used to evaluate each of the 48 block products over $\mathbb{K}$. The resulting multiplication count is $48 \cdot 46 = 2208$.

A central point is model clarity: the inner 46-multiplication routine exploits commutativity and is not bilinear; therefore, our improvement concerns *commutative straight-line multiplication count* and does *not* imply a new tensor-rank upper bound for $\langle 16, 16, 16 \rangle$. We provide a compact exact-arithmetic verifier that checks correctness and the stated multiplication counts. A reference implementation is available at [5].

1 Introduction

Matrix multiplication is a fundamental computational primitive. Besides the vast literature on asymptotically fast algorithms, there is a separate and practically motivated line of work on *exact* multiplication counts for fixed small sizes (especially when the base-ring multiplication dominates additions). Drevet–Islam–Schost explicitly motivate this regime for rings such as multiprecision integers, polynomials, and operator algebras [1].

The 16×16 case is a natural milestone:

- It is the smallest "hardware-aligned" power-of-two size often used as a tiling granularity.
- It is large enough that multi-level composition techniques (blockings of blockings) become interesting, but still small enough to admit detailed algebraic constructions and verification.

1.1 The 2212 reference point

In the commutative case, Drevet–Islam–Schost tabulate an upper bound of 2212 multiplications for the product $\langle 16, 16, 16 \rangle$ (Table 4 in [1]), obtained through composition and "peeling" techniques applied to existing small-dimension schemes. This value has been widely used as a benchmark in the small-matrix multiplication literature.

The present note observes that, once a rational-coefficient 48-multiplication *non-commutative* 4×4 building block became available [2], a straightforward hybrid composition yields 2208 variable multiplications for 16×16.

1.2 Contributions and scope

Our contributions are:

1. We formalize a clean hybrid composition that yields 2208 variable multiplications for 16×16 multiplication over commutative fields of characteristic $\neq 2$.
2. We clarify the computation model: the improvement is in commutative straight-line multiplication count and does not translate into a tensor-rank bound for $\langle 16, 16, 16 \rangle$.
3. We provide an exact-arithmetic verifier (instrumented multiplication counter) to validate correctness and operation counts.

We do *not* claim practical speedups for floating point arithmetic; the addition count and data movement may dominate in many settings, as already emphasized in [1].

2 Computation models and terminology

2.1 Non-commutative bilinear algorithms (tensor rank)

Definition 1 (Bilinear algorithm and tensor rank)**.** *Let S be any ring (not necessarily commutative). A bilinear algorithm for multiplying $n \times n$ matrices over S with r multiplications consists of linear forms $U_t(A)$ in the entries of $A \in S^{n \times n}$, linear forms $V_t(B)$ in the entries of $B \in S^{n \times n}$, and output recombination maps, such that AB can be reconstructed from the r products $U_t(A) \cdot V_t(B)$ computed in S. When S is a matrix algebra over a field, the minimal such r is the tensor rank of the associated matrix multiplication tensor.*

The critical property is *stability under blocking*: a bilinear algorithm remains correct if scalars are replaced by blocks, because the proof does not rely on commutativity.

2.2 Commutative straight-line multiplication count

Many "commutative fast schemes" are not bilinear: they exploit commutativity and multiply expressions that may mix entries from both input matrices. This can reduce the number of multiplication gates in a straight-line program, but such schemes generally do not lift to block recursion.

Definition 2 (Variable-multiplication count)**.** *Fix a commutative field $\mathbb{K}$ (or commutative ring) and consider straight-line programs using additions/subtractions, multiplication by fixed constants in $\mathbb{K}$, and multiplications. We define the variable-multiplication count as the number of multiplications in which* both *operands depend on the inputs. Multiplication by fixed constants is treated as linear and is not counted.*

Remark 1 (Why tensor rank does not follow)**.** *A commutative* 46*-multiplication scheme for* 4×4 *multiplication does not imply* $\mathrm{rank}(\langle 4, 4, 4 \rangle) \leq 46$*: the scheme can be non-bilinear. Therefore, composing a bilinear* 48*-multiplication block algorithm with a commutative* 46*-multiplication scalar routine yields a* 2208*-multiplication* straight-line *algorithm, but does not yield a tensor-rank bound for* $\langle 16, 16, 16 \rangle$*.*

3 Building blocks

3.1 Outer block algorithm: 48 multiplications for non-commutative 4×4

Dumas–Pernet–Sedoglavic give an explicit 4×4 non-commutative bilinear algorithm using 48 multiplications with rational coefficients; the construction is valid over rings containing an inverse of 2 [2]. Their introduction positions this result as removing a prior "complex coefficients" requirement for 48-multiplication schemes [2].

Lemma 1 (Dumas–Pernet–Sedoglavic [2])**.** *Over any ring S in which 2 is invertible, there exists a non-commutative bilinear algorithm that multiplies two 4×4 matrices over S using 48 multiplications in S.*

3.2 Inner scalar algorithm: 46 multiplications for commutative 4×4

For commutative rings (typically assuming the availability of division by 2), Waksman obtained a 46-multiplication scheme for 4×4 multiplication [3]. Drevet–Islam–Schost list $\langle 4, 4, 4 \rangle = 46$ in their commutative table (Table 4 in [1]). Rosowski discusses division-by-2 issues in commutative schemes and presents division-free algorithms for even sizes [4].

Lemma 2 (Waksman multiplication count (commutative 4×4))**.** *Let $\mathbb{K}$ be a commutative field. There exists a commutative 4×4 multiplication routine using 46 variable multiplications in the sense of Section 2.2. [1, 3]*

Remark 2. *Rosowski explicitly notes that some commutative schemes rely on divisions by 2 and provides division-free constructions for even n [4]. In this paper we assume $\mathrm{char}(\mathbb{K}) \neq 2$ anyway, due to the outer algorithm [2].*

4 Hybrid 16×16 construction

4.1 High-level description

Let $\mathbb{K}$ be a commutative field with $\mathrm{char}(\mathbb{K}) \neq 2$. We write a 16×16 matrix as a 4×4 block matrix with 4×4 blocks over $\mathbb{K}$. The ring of blocks is $S = \mathbb{K}^{4\times 4}$, which is non-commutative, but contains $\frac{1}{2}$ since $\mathbb{K}$ does. Therefore the 48-multiplication non-commutative 4×4 algorithm [2] can be applied to multiply block matrices, producing 48 block products $X_t \cdot Y_t$ with $X_t, Y_t \in \mathbb{K}^{4\times 4}$ and additional linear combinations on blocks.

Each of the 48 block products is then computed over $\mathbb{K}$ using a 46-multiplication commutative 4×4 routine [3, 4]. This yields a total of $48 \cdot 46$ scalar (field) multiplications between input-dependent quantities.

4.2 Main theorem and proof

Theorem 1 (2208 variable multiplications for 16×16)**.** *Let $\mathbb{K}$ be a commutative field with $\mathrm{char}(\mathbb{K}) \neq 2$. Then the product of two 16×16 matrices over $\mathbb{K}$ can be computed using exactly 2208 variable multiplications, plus additions/subtractions and multiplications by fixed rational constants.*

Proof. View $A, B \in \mathbb{K}^{16\times 16}$ as 4×4 matrices over the ring $S = \mathbb{K}^{4\times 4}$. Since $\mathrm{char}(\mathbb{K}) \neq 2$, the element 2 is invertible in S. Apply the Dumas–Pernet–Sedoglavic bilinear 4×4 algorithm over S [2]. This computes the block product using 48 multiplications in S, each being a multiplication of two 4×4 blocks over $\mathbb{K}$.

Compute each block product using a 46-multiplication commutative 4×4 routine over $\mathbb{K}$ [3, 4]. Thus each block product costs 46 variable multiplications in $\mathbb{K}$.

Therefore the total variable-multiplication count is $48 \cdot 46 = 2208$. All other operations are additions and multiplications by fixed constants from $\mathbb{Z}[\frac{1}{2}]$, which are linear under the counting convention of Section 2.2. Correctness follows from correctness of the outer block algorithm over S and correctness of the inner scalar routine over $\mathbb{K}$. □

4.3 Algorithm pseudocode

We state the construction in "two-layer" form. The outer algorithm is a specific straight-line program from [2]; we treat it as a black box that produces 48 block multiplicands and a linear

recombination to output blocks.

Algorithm 1 Hybrid 16×16 matrix multiplication (multiplication count $48 \cdot 46$)

Require: $A, B \in \mathbb{K}^{16\times 16}$, with $\mathrm{char}(\mathbb{K}) \neq 2$
Ensure: $C = AB$

1: Partition A, B into 4×4 blocks $A_{ij}, B_{ij} \in \mathbb{K}^{4\times 4}$.
2: Use the 48-multiplication non-commutative 4×4 bilinear algorithm over $S = \mathbb{K}^{4\times 4}$ [2] to form linear combinations $(X_t)_{t=1}^{48}$ of blocks of A and $(Y_t)_{t=1}^{48}$ of blocks of B, and output recombination coefficients.
3: **for** $t = 1$ to 48 **do**
4: Compute $P_t \leftarrow \mathrm{Comm46}(X_t, Y_t)$ using a 46-multiplication commutative 4×4 routine [3,4].
5: **end for**
6: Linearly recombine the P_t's (as prescribed by [2]) to obtain output blocks C_{ij}.
7: Assemble the 16×16 matrix C from the 4×4 blocks C_{ij}.

5 Relation to published tables and the "2212 barrier"

5.1 What Drevet–Islam–Schost report

Drevet–Islam–Schost provide tables of upper bounds for small dimensions derived from systematic compositions of known schemes. Their commutative case table (Table 4) lists:

- $\langle 4, 4, 4\rangle = 46$ ("Waksman"),
- $\langle 16, 16, 16\rangle = 2212$.

[1]

At the time of [1], the best-known non-commutative bilinear cost for $\langle 4, 4, 4\rangle$ over general rings was 49 (from two levels of Strassen recursion), so the natural block-based "outer $\times$ inner" product could not reach $48 \cdot 46$.

5.2 Why 48 matters

Dumas–Pernet–Sedoglavic explicitly state that the non-commutative 4×4 multiplication count was reduced from 49 to 48 with complex coefficients by a recent automated discovery pipeline, and that their contribution is a rational-coefficient version [2]. Once a rational 48-multiplication outer block algorithm exists, the $48 \cdot 46$ hybrid becomes available.

6 Verification

6.1 Counting model

Our verifier counts only variable multiplications: multiplying two values that depend on the inputs increments a global counter; multiplying by a fixed constant (e.g., $\pm\frac{1}{2}$) is treated as linear and is not counted. This matches a straight-line (arithmetic-circuit) counting convention where we count multiplications between input-dependent expressions, and aligns with the definition in Section 2.2.

6.2 Exact arithmetic and tests

We implement:

- a 46-multiplication commutative 4×4 routine ("Rosowski-style"),

- a 48-multiplication bilinear 4×4 routine with rational coefficients,
- the hybrid 16×16 composition,

and validate equality against the naive product on random integer test cases using exact arithmetic (`fractions.Fraction`). The corresponding code artifact is available at [5].

7 Practical considerations

7.1 Addition count and data movement

For floating-point BLAS-style workloads, reducing the multiplication count by a fraction of a percent rarely translates directly into a speedup: additions, temporaries, and memory traffic may dominate. This is consistent with the discussion in [1], which highlights contexts where base-ring multiplication (not addition) is the bottleneck.

7.2 Constant scalings

The outer 48-multiplication algorithm uses rational coefficients and requires a ring containing $\frac{1}{2}$ [2]. Under our counting model, multiplications by fixed constants are treated as linear. If one wishes to count *all* scalar multiplications (including by constants), one should account for these scalings separately; this paper does not attempt to optimize that metric.

8 Scaling impact and cascaded savings

The improvement from 2212 to 2208 variable multiplications is small in relative terms, but it scales linearly with the number of 16×16 products executed inside a larger computation.

8.1 Per-kernel reduction

Relative to the 2212-multiplication upper bound in [1], our kernel saves 4 variable multiplications:

$$\frac{2212 - 2208}{2212} = \frac{4}{2212} \approx 1.808 \times 10^{-3} = 0.1808\%.$$

In a cost model where variable multiplications dominate time and energy, this corresponds to an idealized speedup factor

$$\frac{2212}{2208} \approx 1.00181.$$

In practice, realized wall-clock speedups can be smaller because additions, memory traffic, and constant scalings may dominate (Section 7).

8.2 Propagation to larger powers of two

Let an overall matrix multiplication method (tiled, recursive, or otherwise) invoke a 16×16 multiplication subroutine T times on scalar entries of $\mathbb{K}$. Replacing a 2212-multiplication subroutine by the present 2208-multiplication one reduces the variable-multiplication count by exactly $4T$.

Two common ways T scales with matrix size are:

Classical blocked (tiled) multiplication. For multiplying $N \times N$ matrices with N a multiple of 16, a purely blocked classical algorithm performs $T = (N/16)^3$ multiplications of 16×16 tiles, hence saves

$$4\left(\frac{N}{16}\right)^3$$

variable multiplications versus a 2212-multiplication 16×16 kernel.

Strassen-style recursion with a 16×16 base case. For $N = 16 \cdot 2^k$, applying Strassen recursion k times yields $T = 7^k$ leaf products of size 16×16, hence saves

$$4 \cdot 7^k$$

variable multiplications in the leaves. (Additional additions occur at each recursion level; we do not attempt to optimize those.)

Table 1: Cascaded savings in variable multiplications relative to a 2212-multiplication 16×16 kernel. For $N = 16 \cdot 2^k$, the table shows the number of leaf 16×16 products and total saved variable multiplications for (i) k levels of Strassen recursion and (ii) classical 16×16 tiling.

k	N	#leaves (Strassen)	saved mults	#tiles (blocked)	saved mults
0	16	1	4	1	4
1	32	7	28	8	32
2	64	49	196	64	256
3	128	343	1372	512	2048
4	256	2401	9604	4096	16384
5	512	16807	67228	32768	131072
6	1024	117649	470596	262144	1.05×10^6
7	2048	823543	3.29×10^6	2.1×10^6	8.39×10^6
8	4096	5.76×10^6	2.31×10^7	1.68×10^7	6.71×10^7

As a concrete example, for $N = 4096$ (i.e., $k = 8$), the savings are $4 \cdot 7^8 = 23{,}059{,}204$ variable multiplications for an 8-level Strassen recursion down to 16×16, and $4 \cdot (4096/16)^3 = 67{,}108{,}864$ variable multiplications for a purely

Remark 3 (No direct block-level recursion)**.** *The* 2208*-multiplication method is a commutative straight-line program and is* not *a non-commutative bilinear algorithm. Consequently, it cannot be used directly as a block-level multiplication routine over non-commutative rings (e.g., where scalars are themselves matrices). The propagation discussed above concerns using the* 16×16 *algorithm as a* leaf *routine on scalar matrices over* $\mathbb{K}$ *inside larger tiled or recursive schemes.*

blocked classical algorithm.

8.3 Interpreting "small" percentage improvements

Even a 0.1808% reduction in variable multiplications can be meaningful in regimes where: (i) multiplication is substantially more expensive than addition (e.g., multiprecision integers, polynomial rings, exact arithmetic); and (ii) the workload executes enormous numbers of small products. In such regimes, the absolute savings can be large, while the relative savings remain fixed at 4/2212 on the multiplication-dominated portion of the computation.

9 Conclusion

We record a simple hybrid composition that yields a 16×16 matrix multiplication algorithm using 2208 variable multiplications over any commutative field of characteristic $\neq 2$. The construction relies on:

- a rational-coefficient 48-multiplication non-commutative 4×4 bilinear algorithm [2], used at the block level, and
- a 46-multiplication commutative 4×4 scheme [3, 4], used only at the scalar level.

We stress that the result concerns commutative straight-line multiplication count, not tensor rank for $\langle 16, 16, 16 \rangle$. An exact-arithmetic verifier accompanies the construction.

A Key multiplication counts (summary table)

Table 2: Selected commutative multiplication counts appearing in the cited literature

Problem	Source / description	Variable mults
$\langle 4,4,4\rangle$ (commutative)	Waksman scheme listed by Drevet–Islam–Schost (Table 4) [1]; see also [3, 4]	46
$\langle 4,4,4\rangle$ (non-commutative)	Dumas–Pernet–Sedoglavic [2]	48
$\langle 16,16,16\rangle$ (commutative)	Drevet–Islam–Schost (Table 4) [1]	2212
$\langle 16,16,16\rangle$ (commutative)	This hybrid construction ($48 \cdot 46$)	2208

References

[1] C.-É. Drevet, M. N. Islam, and É. Schost, "Optimization techniques for small matrix multiplication," *Theoretical Computer Science*, 412(22):2219–2236, 2011. DOI: 10.1016/j.tcs.2010.12.012.

[2] J.-G. Dumas, C. Pernet, and A. Sedoglavic, "A non-commutative algorithm for multiplying 4×4 matrices using 48 non-complex multiplications," *arXiv:2506.13242*, 2025.

[3] A. Waksman, "On Winograd's algorithm for inner products," *IEEE Transactions on Computers*, C-19(4):360–361, 1970. DOI: 10.1109/T-C.1970.222926.

[4] A. Rosowski, "Fast Commutative Matrix Algorithm," *arXiv:1904.07683*, 2019.

[5] spicylemonade, "faster_16x16: Reference implementation and verifier for a 2208-multiplication 16×16 matrix multiplication algorithm," GitHub repository (main branch), accessed 2026-02-02. `https://github.com/spicylemonade/faster_16x16/tree/main`.

[6] A. Fawzi et al., "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, 610:47–53, 2022. DOI: 10.1038/s41586-022-05172-4.