# A Tight I/O Bound for $2 \times 2 \times k$ Matrix Multiplication with Four-Word Fast Memory

Anonymous Author(s)

## Abstract

We give a tight communication bound for a deliberately small but nontrivial classical matrix-multiplication family. Consider a single owner computing $C = AB$ with $A \in \mathbb{R}^{2\times k}$ and $B \in \mathbb{R}^{k\times 2}$, a four-word fast memory, one execution of every scalar product, and no recomputation or free copying. Under an explicit word-traffic convention, the optimum is

$$Q^*(2 \times 2 \times k, M = 4) = 6k + 2$$

for every $k \geq 1$. The upper bound is a periodic schedule. For the lower bound we first derive the exact identity $Q = 8k + 4 - H + 2R_C$, where $H$ counts second-use hits of $A/B$ words and $R_C$ counts restarts of a $C$ partial after eviction. We then project every trace onto its high-reuse cycles and analyze a relaxed finite-state automaton. Its 278-state max-plus transition system has a four-step fixed point with potential at most two, which certifies $H - 2R_C \leq 2k + 2$. Together with the schedule this proves the theorem. The result is a machine-checked, capacity-sensitive finite-instance theorem; it does not claim a new asymptotic exponent for general GEMM.

## CCS Concepts

• **Theory of computation → Parallel computing models**; *Design and analysis of algorithms.*

## Keywords

matrix multiplication, I/O complexity, cache complexity, finite-state verification

## 1 Introduction

Communication lower bounds for matrix multiplication are usually presented in an asymptotic regime, where the capacity term has the familiar $\Omega(n^3/\sqrt{M})$ form. Small capacities expose a different question: which reuse opportunities can coexist in one cache, and how much traffic is forced when partial output values are evicted? These effects are easy to miss if input reloads and output movement are summarized by separate counts.

This paper isolates the smallest setting in which the interaction is already nontrivial. We study a single owner multiplying a $2 \times k$ matrix by a $k \times 2$ matrix with a cache of four words. Each product needs one $A$ word, one $B$ word, and one partial $C$ word. The cache can therefore preserve some reuse across products, but it cannot preserve all four $C$ partials together with the operands needed to exploit every reuse edge.

Our main theorem is:

$$Q^*(2 \times 2 \times k, M = 4) = 6k + 2. \tag{1}$$

*Conference'17, Washington, DC, USA*
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The proof has two parts. A periodic product order attains the right-hand side. For the lower bound, let $h_r$ be the number of $A/B$ second-use hits in the cycle associated with inner index $r$, let $H = \sum_r h_r$, and let $R_C$ be the number of times a $C$ partial is reloaded after it was used and evicted while it still had a future use. Exact accounting gives

$$Q = 8k + 4 - H + 2R_C. \tag{2}$$

The remaining question is a compensation inequality:

$$H - 2R_C \leq 2k + 2. \tag{3}$$

We establish (3) with a finite-state certificate. A cycle with $h_r = 3$ is represented by a four-vertex path and contributes one unit of demand; a cycle with $h_r = 4$ is represented by a full four-cycle and contributes two units. The automaton retains the unfinished components, used $C$ labels, and resident $C$ labels. It is a relaxation: deleting low-demand products can only release capacity, while every projected $C$ reload corresponds to a real $C$ restart. Dynamic programming over canonical states reaches a periodic fixed point after 16 layers. The fixed point contains 278 states and its maximum value of demand minus restarts is two.

The contributions are:

- an exact traffic identity separating $A/B$ hits from $C$ restarts;
- a projection lemma reducing every real trace to a relaxed high-demand automaton;
- a finite-state, arbitrary-$k$ certificate for the compensation inequality; and
- a matching schedule attaining the bound for this $2 \times 2 \times k$ family under the stated model.

The scope is intentionally explicit. We do not claim a new exponent for general $n \times n$ GEMM, a multi-owner lower bound, or a universal theorem for all replacement policies. The finite-state certificate is the result, and the checker and its independent reconstruction are part of the proof artifact.

## 2 Model and accounting

Let $A \in \mathbb{R}^{2\times k}$, $B \in \mathbb{R}^{k\times 2}$, and $C \in \mathbb{R}^{2\times 2}$. The scalar products are indexed by $p = (i, r, j)$ with $i, j \in \{0, 1\}$ and $r \in \{0, \ldots, k-1\}$:

$$p: \quad A_{ir}B_{rj} \longrightarrow C_{ij}.$$

Every product is executed exactly once. An execution chooses an arbitrary product order and an arbitrary legal replacement policy for a cache containing four words. At the instant a product is executed, its three operands $A_{ir}$, $B_{rj}$, and the current partial $C_{ij}$ are resident. There is one owner, no recomputation, and no copying of a product result.

The four $C$ values start at zero. Loading an $A$ or $B$ word costs one word of traffic, and a repeated use costs zero exactly when the word is still resident. A $C$ partial is initially available at zero cost. If it is evicted after a use and later needed again, its reload and the corresponding store/re-entry account for two words; each such

event is one $C$ restart. After the last product using a $C_{ij}$, writing its final value costs one word. This convention is the one implemented by the exact checker and is fixed for the rest of the paper.

For cycle $r$, define $h_r$ to be the number of $A/B$ tokens whose second use is a cache hit while the four products with inner index $r$ are processed. There are exactly $8k$ $A/B$ requests in total. Thus every hit saves one word from the compulsory $8k$ requests, while every $C$ restart adds two words and the four final writes add four words.

## 3 An attaining schedule

The schedule family used by the upper-bound checker is periodic in the inner index. It keeps two $C$ labels active, moves to the next pair when a product has exhausted the current partial, and arranges the four products of each new inner index so that six of the eight $A/B$ requests are hits in the steady pattern. The boundary pattern uses the four cache slots exactly; no $C$ partial is restarted in the construction.

The generator `work/check_2x2k_m4_schedule.py` emits the order and the exact checker reports

$$Q = 6k + 2$$

for $k = 1, \ldots, 10$. The pattern is valid for every $k$: the first and last cycles supply the two boundary units, and each additional cycle contributes exactly six words after the reuse savings are accounted for. Therefore

$$Q^*(2 \times 2 \times k, M = 4) \le 6k + 2. \tag{4}$$

## 4 Exact traffic identity

Let $H = \sum_r h_r$. The $8k$ $A/B$ requests would all be loads in the absence of reuse. Each second-use hit removes one load, so the $A/B$ contribution is $8k - H$. A $C$ value that is used in several separated intervals must be re-entered after every nonfinal eviction. Under the accounting convention of Section 2, each such restart contributes two words. Finally, each of the four outputs is written once. Hence every legal trace satisfies

$$Q = (8k - H) + 2R_C + 4 = 8k + 4 - H + 2R_C. \tag{5}$$

This identity is bookkeeping, not a lower bound. It reduces the theorem to showing that input reuse cannot grow faster than the output restarts can pay for it, except for two boundary units.

## 5 High-demand projection

The four possible $A/B$ hit edges for one inner index form a cycle $C_4$. If $h_r = 3$, the selected edges are $C_4$ with one edge deleted, hence a spanning path. If $h_r = 4$, all four edges are selected. Define

$$\epsilon_r = \max(h_r - 2, 0).$$

Thus a path component has weight one and a full-cycle component has weight two. Cycles with $h_r \le 2$ are discarded. The discarded products and operands are also deleted from the projected trace.

The projection keeps only selected $A/B$ ports and the four shared $C$ labels. An open selected port must remain resident until its second endpoint is processed. A path component retains one temporary endpoint operand; a full cycle has no such endpoint slack. A projected $C$ label may be evicted and reloaded, with one unit of restart cost after its first use.

Lemma 5.1 (Projection soundness). *For every legal real trace, the projected automaton has a run whose demand minus restart value is at least*

$$\sum_r \epsilon_r - R_C.$$

Proof. Keep the real order of all selected product events and delete every other product. Selected $A/B$ hit edges remain resident in the real trace until the corresponding second endpoint; deleting events only releases cache pressure. The projected automaton may choose any legal eviction of a selected $C$ label. Whenever it reloads a label after a previous selected use, the real trace must also have lost and re-entered that partial, so the projected restart count is no larger than $R_C$. The projected cache therefore has at least as much free capacity as the real cache at every selected event. Completing a path or a full-cycle component records exactly its weight. Summing over components gives the claimed inequality. □

## 6 Finite-state compensation certificate

An automaton state is

$$\sigma = (P, U, F),$$

where $P$ is a canonical multiset of unfinished components, $U$ is the set of $C$ labels already used, and $F \subseteq U$ is the set of currently resident $C$ labels. Each component stores its kind, its missing edge (for a path), and its visited-vertex mask. All unfinished selected ports are implicitly resident.

A transition chooses one currently executable selected product. Its capacity test counts open selected ports, resident $C$ labels, newly introduced selected ports, a path endpoint's temporary operand, and one slot if the current $C$ is a reload. If a component is completed, its weight is added. A reload subtracts one. Dynamic programming keeps the largest accumulated value

$$\Phi = D - R$$

for each canonical state.

The transition relation is finite and stationary. The main checker constructs the reachable max-plus layers through step 20 and verifies that the complete state-to-value map at steps 16 and 20 is identical. Thus the transition map has a four-step fixed point. The certificate output is:

```
high-demand automaton certificate: passed
layers=20 max_potential= 2
fixed_point_steps=16..20 period=4 states= 278
state_counts= [20, 212, 272, 278, 216, 304, 288, 278,
               216, 304, 288, 278, 216, 304, 288, 278,
               216, 304, 288, 278]
```

Because the state contains every piece of information relevant to future capacity and restart choices, equality of the layer maps implies that all later layers repeat the same four-step map. The maximum potential is therefore at most two for an arbitrary number of high-demand components. An independent implementation, which does not import the main checker, reconstructs the same 278-state fixed point and the same maximum sequence.

Theorem 6.1 (High-demand compensation). *For every legal trace,*

$$\sum_r \epsilon_r - R_C \le 2.$$

Proof. By projection soundness, the real trace is bounded by the automaton's maximum potential. The finite-state fixed point certificate proves that this maximum is two for every length. Therefore $\sum_r \epsilon_r - R_C \le 2$. □

## 7 Main theorem

Since $h_r \le 2 + \epsilon_r$ for every $r$,

$$H = \sum_r h_r \le 2k + \sum_r \epsilon_r \le 2k + R_C + 2.$$

Consequently,

$$H - 2R_C \le 2k + 2.$$

Substituting this inequality into (5) gives

$$Q \ge 8k + 4 - (2k + 2) = 6k + 2.$$

Combining with the schedule in (4) proves (1).

Theorem 7.1 (Tight four-word I/O bound). *Under the model of Section 2, for every integer $k \ge 1$,*

$$\boxed{Q^*(2 \times 2 \times k, M = 4) = 6k + 2.}$$

## 8 Context, limits, and reproducibility

For comparison, the same single-owner family with $M = 3$ has the exact value $Q^* = 8k + 1$: a run-boundary argument limits the total $A/B$ hits to three more than the number of nonfinal $C$ movements. When the cache is large enough to retain all required operands and partials for the three-cycle construction, the compulsory floor is reached. These neighboring regimes show why a single smooth formula in $M$ should not be expected for finite instances.

The theorem is deliberately narrower than the classical sequential lower bound for $n \times n$ GEMM. It fixes the dimensions, one owner, a four-word cache, once-only products, and a precise treatment of $C$ movement. It does not cover multiple owners, free multicast, recomputation, asymmetric read/write costs, LRU specifically, or nested memory edges. The finite-state certificate proves an arbitrary-$k$ theorem within this model; it is not evidence that the same constant holds for general GEMM.

The artifact contains four independently checkable pieces:

(1) `work/check_2x2k_m4_schedule.py` verifies the attaining schedule;
(2) `work/check_high_demand_automaton.py` constructs the main certificate;
(3) `check_high_demand_fixed_point_independent.py` reconstructs the fixed point independently; and
(4) `work/check_high_demand_projection.py` audits the projection against exact traces for 153 fixed-seed random high-demand cases with $k \le 4$.

The first three are deterministic proof checks. The projection audit is a consistency test for event encoding and restart accounting, not a replacement for Lemma 1.

## 9 Related work

The red-blue pebble game of Hong and Kung initiated the sequential I/O view of matrix multiplication. Subsequent work established communication lower bounds for distributed matrix multiplication and generalized the geometric Loomis–Whitney argument to parallel numerical linear algebra. Tight leading constants for classical GEMM and bounds for Strassen-like algorithms study different asymptotic or recursive regimes. Our theorem complements that line: it fixes a tiny capacity-sensitive instance and resolves its exact finite traffic, including the interaction between operand hits and partial-output restarts.

## References

[1] J.-W. Hong and H. T. Kung. 1981. I/O complexity: The red-blue pebble game. In *Proceedings of STOC '81*, 9–19.
[2] O. Irony, S. Toledo, and A. Tiskin. 2004. Communication lower bounds for distributed-memory matrix multiplication. *JPDC* 64, 9, 1017–1026.
[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. 2011. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.* 32, 3, 866–901.
[4] J. Scott, O. Holtz, and O. Schwartz. 2015. Matrix multiplication I/O-complexity by path routing. In *SPAA '15*, 35–45.
[5] S. Smith, J. Lowery, J. Langou, and R. van de Geijn. 2020. A tight I/O lower bound for matrix multiplication. *ACM TOMS* 46, 2.