

TagmaMatrix

Coordinate-Addressed Matrices, the Integer Product, and the Wire Form

SSCCS Initiative*

ssccs.org

September, 2026

Abstract

A `CoordPath<2>` is the smallest composition the coordinate system offers: two `Coords`, and a point. `tagma-matrix` reads that pair as the address of an element of a rank-2 integer matrix. The physical order of the backing bytes is a type parameter and never part of that address, so a coordinate names the same element whether the bytes sit row-major or column-major, the product over the elements is byte-identical across the two, and the bytes a matrix writes are the same in either order. That property, relocation invariance, is what lets a matrix cross between devices without changing its identity, and it is held by a test. The crate adds the integer product, `i8` by `i8` into `i32`, the contraction of two such matrices, the rescale from an accumulator back to an element stated as a result rather than as a way of computing one, and a wire form that carries coordinates and values instead of a layout. It takes no allocator: `tagma-core` is taken with its defaults off, and a link into a program with no operating system and no global allocator is part of the crate's verification. This document is a draft.

1 Introduction

The family answers four different questions over the same space. `CoordPath`¹ decides whether a path exists. `TagmaGeo` decides where paths stand relative to one another. `TagmaMap` decides which value a path holds. `TagmaMatrix` decides what the value at a coordinate is when the values form a structure the program computes with.

The observation it starts from is that a matrix element's position is already an address in this system. A row and a column are two `Coords`, and two `Coords` are a `CoordPath<2>`. Nothing about the element's byte offset enters the statement.

Heap tensor libraries make the opposite choice. The offset of an element is derived from a base pointer and a stride, the stride is part of the API, and the layout of the data is therefore something a consumer must know in order to read it. In that model the physical order is part of the element's address.

Two consequences follow once the coordinate is the address. A reader answers the same coordinate with the same value whatever arrangement the bytes are in, because the arrangement is a storage decision. A matrix can cross to another device as its coordinates and their values, leaving the receiver free to lay the bytes out differently. Both are one statement: the identity of an element is its coordinate, and relocation invariance is that statement made checkable.

*; Corresponding author: contact@ssccs.org

¹Tagma Whitepaper: docs.ssccs.org/projects/syntaxma/tagma/

2 The Address Model

An element is named by two Coords, the row first and the column second. The crate carries them as a `CoordPath<2>`.

The addressable range follows from the primitive. Each axis holds at most 11,172 values, so a matrix is at most 11,172 by 11,172, and at one byte per element a full one occupies about 125 MB. That is the reason the crate offers a borrowed form: an owned matrix carries its buffer inline, and a borrowed one reads a buffer that stays where the linker put it.

The physical order is a type, and its only job is to turn a position into an offset.

Order	Offset of element (i, j) in an R by C matrix	Adjacent in memory
RowMajor	$i * C + j$	consecutive columns
ColMajor	$j * R + i$	consecutive rows

The order is not part of the address. The same coordinate resolved under the two orders lands at two different offsets and answers with the same value.

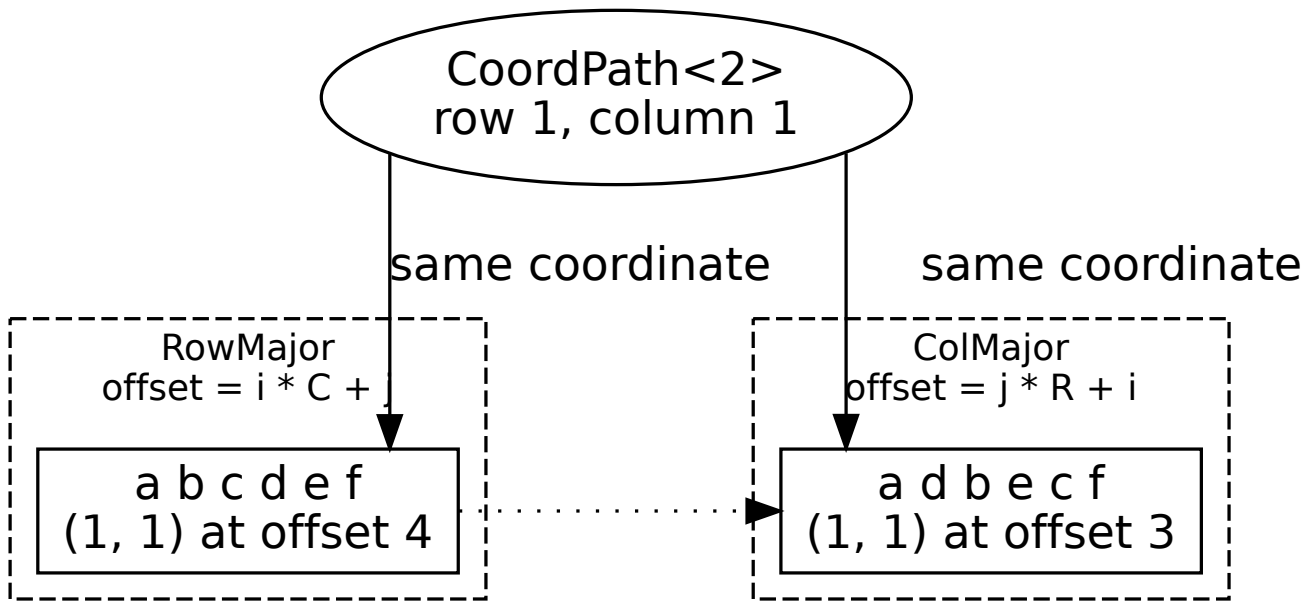


Figure 1: One coordinate, two physical orders, two offsets, one value.

A matrix owns its buffer as `[[i8; C]; R]`, so the buffer's shape is in the type and the offsets above are the only arithmetic between a position and a byte. The type derives `Copy`, as `Coord`, `CoordPath` and `CoordSet` do, which means a copy duplicates the whole buffer. A large matrix is therefore a value to borrow, or to read through the borrowed form.

3 The Bounds Contract

Two surfaces read an element, and they fail differently on purpose.

Surface	Out of range	Reason
<code>get(i, j), set(i, j, v)</code>	panics	indexing a container, which Rust spells as a panic

Surface	Out of range	Reason
<code>path_of(i, j), at(path)</code>	None	an address that names no element of this matrix

The panicking surface exists because the alternative was worse than a panic. The flat offset of a position outside the matrix is the offset of a position inside it. For a two by three row-major matrix the offset of (0, 3) is 3, which is the offset of (1, 0), so an unchecked read returns the neighbouring row's first element, while the same call in column-major order panics on the slice index instead. One mistake, silent in one order and loud in the other, which is a shape of failure a reader cannot reason about. Both readers refuse the position, and the refusal is documented under # Panics.

The dimensions carry a bound of their own. Both are at least one, because a column count is the divisor that turns a flat offset back into a position and a matrix with no row holds no element, and neither exceeds the coordinate space, because an index above it has no coordinate to be named by. Every constructor and every method the trait provides asserts this, so a matrix outside the space fails to build rather than resolving an address to the wrong element.

The assertion relates two const generic parameters, a relation the compiler evaluates at monomorphization. A build reports it, and cargo check on its own does not.

4 Relocation Invariance

Relocation invariance is the property the crate exists to make checkable. Stated as a test rather than as prose, it has four parts: the same logical matrix in row-major and in column-major order answers the same coordinates with the same values, produces a byte-identical product, and writes identical bytes.

The claims are falsifiable, and tests in the crate hold them. If an address depended on where the bytes sit, these runs would differ, and the difference would be the evidence.

Test	What it holds
<code>two_physical_orders_answer_the_same_coordinates</code>	order-independent
<code>two_physical_orders_produce_the_byte_identical_product</code>	the byte-identical product of the same logical elements is the same, element for element
<code>the_stream_does_not_depend_on_the_physical_order</code>	the stream contains the same bytes
<code>a_round_trip_carries_the_matrix_into_the_other_order</code>	the matrix read into the other order holds the same values and produces the same product
<code>a_relocated_value_writes_the_same_bytes_it_writes_back</code>	what it arrived as

Relocation invariance is what the term teleport means here, and it is a statement about identity rather than about transport. A value that moves between devices keeps its identity when the receiver can name it by the same coordinate, and the coordinate is the one thing both sides already share.

5 The Wire Form

A matrix crosses as its coordinates and their values. The encoding opens with a header and then writes one record per element in logical order.

Field	Width	Content
magic	4 bytes	TGMX
version	1 byte	1, the only version this build reads
rows	2 bytes	R, little-endian
columns	2 bytes	C, little-endian

Field	Width	Content
record	5 bytes	row Coord index, column Coord index, value, each little-endian

The length follows from the shape: $9 + 5 * R * C$ bytes. Records are written in logical order rather than in storage order, so the stream is canonical and two matrices with the same logical content write the same bytes whatever order their own storage uses.

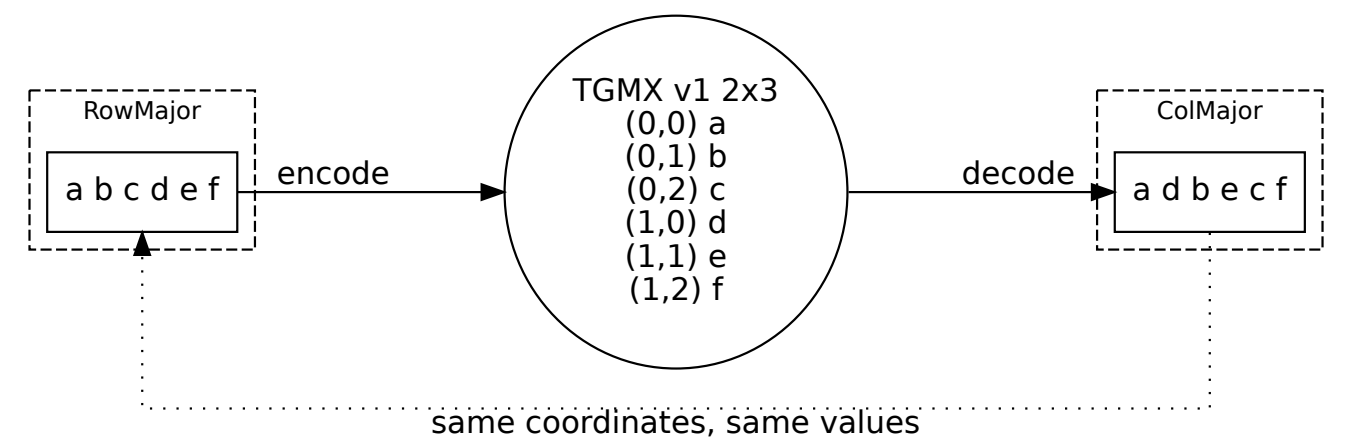


Figure 2: A matrix moves as coordinates and values, and the receiver lays the bytes out as it likes.

The write half is a method on the trait every reader implements, so an owned matrix and a borrowed one write through the same code, and a matrix whose weights live in read-only memory is sent on without being copied into RAM. The read half is an associated function of the owned type, because reading in is the one direction that needs an owner to place the values in.

A reader refuses a stream it cannot account for, and it names the reason rather than reading something plausible.

Refusal	Condition
TooShort	fewer bytes than a header holds
BadMagic	the opening bytes belong to another format
UnsupportedVersion	the version is not one this build reads
ShapeMismatch	the shape in the stream is not the shape of the matrix being filled
Truncated	the stream ends before its records do
TrailingBytes	the stream holds bytes the shape does not account for
CoordinateOutOfRange	a record's coordinate names no element of the matrix
CoordinateMismatch	a record's coordinate names another element than its own position

The last two carry the weight of the format. A receiver places every value at the coordinate its record names, so a stream that names the wrong coordinate, or names one outside the matrix, is rejected instead of filling a plausible matrix with misplaced values.

6 The Integer Product

The product is a matrix-vector multiply over i8 operands accumulating into i32.

$$y_i = \sum_{j=0}^{C-1} a_{i,j} x_j$$

Two properties are worth stating, because they are the whole of the operation's design.

The accumulator cannot overflow at any addressable width. The largest magnitude a term can reach is $128 * 127$, since the bound on a product of two i8 values is the extreme magnitudes, 128 and 127, and the longest sum has 11,172 terms. The bound is therefore 181,612,032 against an i32 maximum of 2,147,483,647, with room to spare, so no saturating arithmetic and no widening to 64 bits is needed.

The product reads its operands through the coordinate surface, so an owned matrix and a borrowed one serve the same call. It computes into a caller-provided slice, so the operation allocates nothing: the accumulator is a register and the output is the caller's memory.

The contraction takes the same form over two matrices.

$$c_{i,j} = \sum_{k=0}^{K-1} a_{i,k} b_{k,j}$$

The overflow bound is the shared one, because the largest term is still $128 * 127$ and a contraction is at most 11,172 terms long, so the same i32 accumulator carries it and no saturating arithmetic is introduced. Both operands are read through the coordinate surface, so the result does not depend on the order either one's bytes sit in. A contraction that is to be carried through a rescale can take it as each accumulator completes, which leaves the product without a matrix of intermediate values of its own.

7 The Rescale

A product accumulates in i32 and an element is an i8. The step between them is one multiply by a fixed point value followed by a right shift, and it is the step in a quantized model where the arithmetic is easiest to get subtly wrong: the product does not fit in thirty-two bits, and a rounding that is right for a magnitude is wrong for a signed value.

The contract is stated as one result rather than as a way of computing one, so any implementation that reproduces the result is conformant whatever it does inside.

Step	Operation
multiply	the accumulator times a signed 32-bit multiplier
round	add $1 \ll (\text{shift} - 1)$, so a half moves toward positive infinity
shift	move right by <code>shift</code> , arithmetic, which floors
saturate	clamp into the i8 range

The intermediate is sixty-four bits wide. A thirty-two bit target reaches it through a high multiply and a low multiply rather than through a wider register, which is why the bound on the shift is a bound on the intermediate: the product of two thirty-two bit values occupies at most sixty-two bits, so a rounding constant derived from a shift above sixty-two would push the sum past sixty-four bits. The crate accepts a shift up to sixty-two and refuses one above it. A shift of zero adds no rounding constant and takes the product as it stands.

The rounding is asymmetric, and the asymmetry is the part worth stating plainly. Half of a negative odd value sits exactly between two integers, and it moves toward positive infinity, so half of -1 is 0. That is the convention the TFLite line and its gemmlowp ancestor state. Their other spelling, a doubling high multiply composed with a power-of-two divide, rounds the same product another way, and a model carrying that spelling is converted before it is run here.

A result beyond the element range lands on its end. Saturation belongs to the step rather than to a policy the caller chooses, which keeps a stream of elements inside the type the wire form carries.

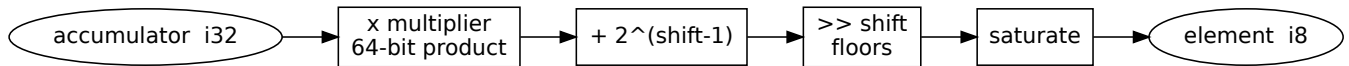


Figure 3: One accumulator, one result, and the four steps any implementation has to reproduce.

Stating the rescale as a result is what makes the channels comparable. The reference implementation reaches it through a sixty-four bit product and an arithmetic shift. A hand-written kernel on a target without a wide multiply reaches it through `mul` and `mulh`, with a carry into the high half. A hardware unit reaches it through a multiplier sized for the product. Three paths, one result, and the comparison between them is exact because the result was fixed before any of them was written.

8 The No-Allocator Property

The crate is `#![no_std]`, and it takes `tagma-core` with its default features off so that the primitive's `alloc` feature stays off with it.

A claim of that kind is worth making into an artifact, because the link is the evidence. The check links the crate into a program with no operating system and no global allocator, for `riscv32imac-unknown-none-elf`, and a successful link is the proof: a use of the allocator anywhere underneath fails the link with `no global memory allocator found but one is required`. The check reaches the whole surface, including the borrowed form and the wire form, so an addition that allocates is reached by the link.

The check lives outside the Rust workspace on purpose. It has to be built on its own, because a workspace-wide build also selects `TagmaGeo` and `TagmaMap`, which unify `tagma-core`'s `alloc` feature on, and the link would then fail for a reason that has nothing to do with this crate².

9 Scope and Boundaries

In scope: the rank-2 element addressed by a coordinate, the integer product over the elements, the wire form that moves a matrix between devices, and relocation invariance held as a test.

Out of scope, deliberately:

- Choosing a scale. Which multiplier and which shift a model needs belongs to the pipeline that measured them, and a crate that guessed one would be making a decision it cannot see. Applying a scale the caller supplies is a different matter: the rescale is arithmetic over the crate's own accumulators, so it is here, stated as a result.
- Optimized kernels. A kernel belongs to the consumer that owns a target.
- Ranks other than two. A rank-2 element is the smallest thing two `Coords` can address, and the wire form is written for that shape.
- A C++ or Java port. The ports carry the in-memory contracts that storage needs, and this crate owns no persistence, so there is nothing for them to mirror yet.

Performance is measured for two operations at one shape, and those numbers are under `Status`. No wider claim is made.

²No-allocator link check: github.com/ssccsorg/syntaxma/tree/main/sw/rust/verify/linkcheck

10 Comparison with Existing Systems

The comparison here is structural, over the axes this crate changes. No comparison against another system's throughput is offered, because none has been measured. The one measurement that exists is of the address model against a flat loop over the same buffer, and it is under Status.

Aspect	TagmaMatrix	Heap tensor libraries	no_std inference kernels
Allocator	none	heap, in the runtime	none
What an element's address is	a pair of Coords	a base pointer and a stride	index arithmetic over a buffer
Physical order in the API	a type parameter, absent from the address	strides, observable and commonly significant	a convention of the kernel
Accumulator rescale	stated as a result, saturating into the element type	a dequantize op in the graph, with the runtime's own rounding	the kernel's own rounding
Wire form	coordinates and values, defined by the crate	an external format such as ONNX or NNEF	none
Relocation invariance	held by a test	not a property of the type	not a property of the type
Rank	two	any	fixed by the model

The second column stands for frameworks whose tensor type is a view over heap memory with strides. The third stands for the no_std inference runtimes that provide zero-allocation kernels without a coordinate address model, such as embedded-nn and ember-rs. This crate replaces neither; it sits between them and tagma-core. A hosted engine such as ONNX Runtime or tract is the usual choice when a model has to be loaded rather than addressed, and both require std and an allocator.

The axis that separates them is the address. Where the runtime gives an element an address derived from a base pointer and a stride, this crate gives it a coordinate that a second device can recompute without knowing anything about the first device's memory.

11 The Crate Surface

Item	Description
Matrix<R, C, O>	rank-2 i8 elements that own their buffer, with the physical order as a type parameter
MatrixRef<'a, R, C, O>	the same read surface over a borrowed buffer, for weights that stay in read-only memory
Order, RowMajor, ColMajor	the trait that turns a position into an offset, and the two orders
Elements<R, C, O>	the read surface (element, path_of, at) and the write half of the wire form
gemv	the integer product, i8 by i8 into i32
gemv_requantized	the product with the rescale applied to each accumulator
matmul	the contraction of two matrices, into the caller's buffer
matmul_requantized	the contraction with the rescale applied as each accumulator completes
Requant, requantize_into	a multiplier and a shift, with requantize stating the result of one accumulator and requantize_into applying it to a run
Matrix::decode	the read half of the wire form
EncodeError, DecodeError	why a write or a read was refused, one variant per condition

12 Status

This document is a draft. The crate is version 0.1.0, in the `sw/rust/matrix` directory of the `synTagma`³ repository⁴, added under issue 65⁵.

The crate carries 49 tests across seven binaries, and one doc-test, all green.

Binary	Tests	Subject
<code>scalar_ref</code>	5	the product against a scalar reference that reads through coordinates
<code>matmul</code>	6	the contraction against a coordinate reference, across the orders of both operands and of the destination, and against the identity
<code>relocation</code>	7	the invariance claims in the table above, including the requantized product
<code>wire</code>	10	the write path and the reader's refusals
<code>borrowed</code>	6	the borrowed form against the owned one, and its write path
<code>addressing</code>	5	the refusals at the boundary
<code>requant</code>	10	the rescale against a floor-division reference, the half case, the element ends, and the shift bound

Verification runs on the host and for `riscv32imac-unknown-none-elf`, with `cargo clippy` warnings denied in both, and the no-allocator link check is built for the target.

The cost of addressing is measured rather than assumed. `sw/rust/benches/bench_matrix.rs` runs the product at the size of a layer and beside a flat loop over the same buffer, which is what an engine that addresses elements by offset writes.

Benchmark	Shape	Host time
<code>matrix/gemv_addressed</code>	64 x 256	7.32 μ s
<code>matrix/gemv_flat</code>	64 x 256	1.82 μ s
<code>matrix/gemv_requantized</code>	64 x 256	7.36 μ s
<code>matrix/wire_encode</code>	64 x 256	16.26 μ s
<code>matrix/matmul_64x64x64</code>	64 x 64 x 64	38.18 μ s
<code>matrix/matmul_requantized_64x64x64</code>	64 x 64 x 64	37.73 μ s
<code>matrix/requantize_256</code>	256	194.8 ns

³`synTagma` repository: [Github](#) (Pre-release, Apache 2.0)

⁴`tagma-matrix` reference implementation: github.com/ssccsorg/syntagma/tree/main/sw/rust/matrix

⁵Issue 65, Add `tagma-matrix`: github.com/ssccsorg/syntagma/issues/65

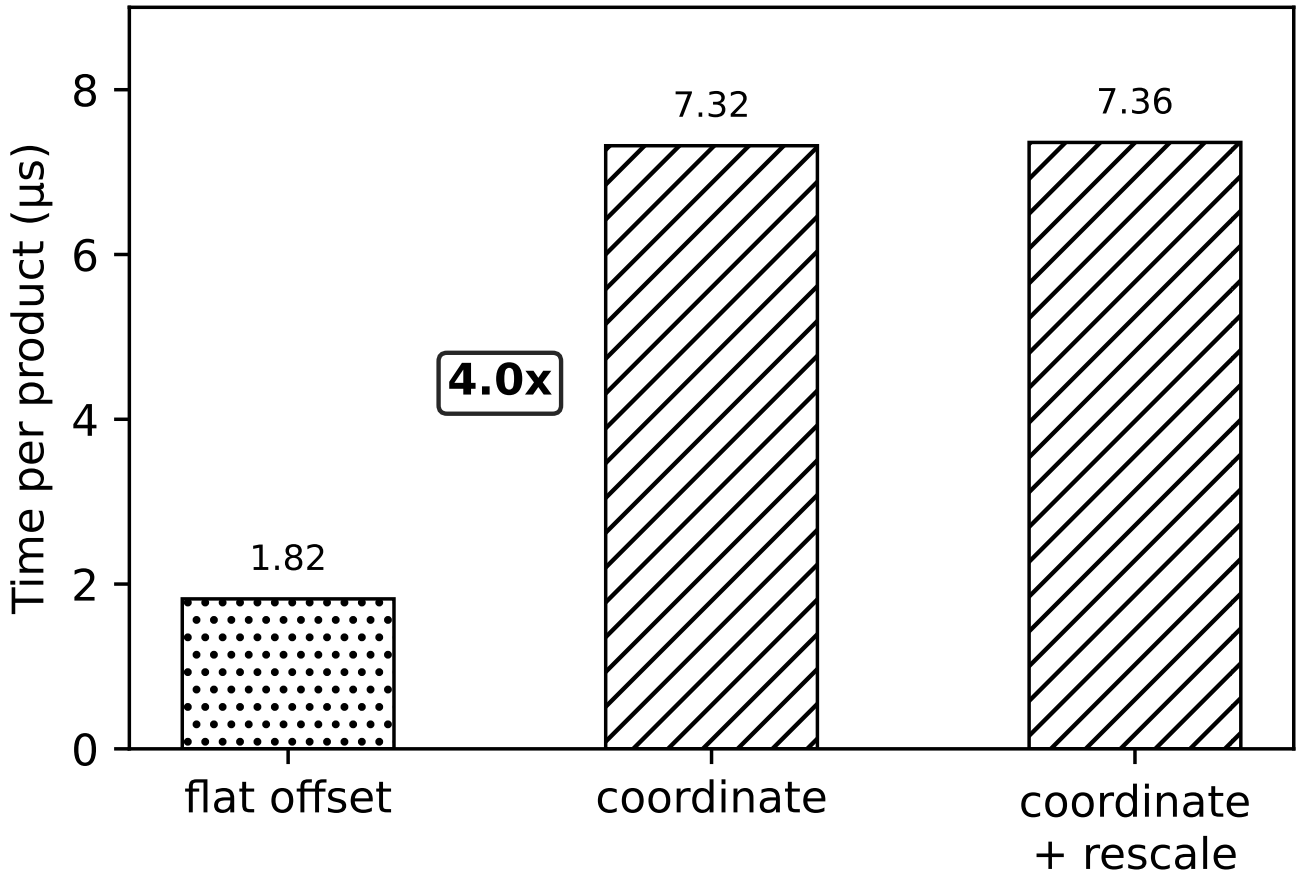


Figure 4: The same product over the same buffer at 64 by 256, addressed by flat offset and by coordinate. The address model costs a factor of 4.0 on a scalar loop, and the rescale costs nothing measurable beside it, since it is one multiply and one shift against 256 of them. Host: Apple M1 Max, rustc 1.95.0, 30 samples, release profile. Source: `sw/rust/benches/bench_matrix.rs`.

The product through coordinates costs about a factor of four over the same product through a flat loop, which is the price of an element's address being its coordinate on a scalar implementation. The rescale adds nothing measurable to the product, since it is one multiply and one shift against 256 of them. The wire form writes five bytes per element and takes about a nanosecond per element to do it. These are host numbers from a scalar build, and no kernel is claimed for any target.

References

Appendices

12.1 The Wire Format, Byte by Byte

The encoding of a two by three matrix with the values

$$a = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$$

is 39 bytes: nine for the header, then one five-byte record per element in logical order.

Offset	Bytes	Meaning
0	54 47 4D 58	magic, TGMX
4	01	version
5	02 00	rows, 2
7	03 00	columns, 3
9	00 00 00 00 01	(0, 0) = 1
14	00 00 01 00 FE	(0, 1) = -2
19	00 00 02 00 03	(0, 2) = 3
24	01 00 00 00 04	(1, 0) = 4
29	01 00 01 00 05	(1, 1) = 5
34	01 00 02 00 FA	(1, 2) = -6

Negative values are two's complement, so -2 is 0xFE and -6 is 0xFA. The crate reads these bytes back into either physical order and writes them again from either order, which is the invariance the format exists to carry.

12.2 The Rescale, Worked

The rescale of this draft's example model is a multiplier of 38,997,123 and a shift of 27, so the divisor is 134,217,728 and the rounding constant is 67,108,864.

Accumulator	Product	Plus rounding	Shifted	Element
0	0	67,108,864	0.50	0
1	38,997,123	106,105,987	0.79	0
4	155,988,492	223,097,356	1.66	1
300	11,699,136,900	11,766,245,764	87.66	87
-300	-11,699,136,900	-11,632,028,036	-86.66	-87
2,000,000	77,994,246,000,000	77,994,313,108,864	581,102.92	127

The last row is beyond the element range and lands on its end. The fourth and fifth rows are one magnitude with two signs and land on two elements of one magnitude, which is the rounding behaving symmetrically where the values are not halves.

The rounding is asymmetric where they are. A multiplier of one with a shift of one puts the difference in one place: an accumulator of -1 has a product of -1, which sits exactly between 0 and -1 after the shift, and it lands on 0. Rounding away from zero would put it on -1, and a test in the crate fixes which of the two this contract means.

12.3 What Is Not in the Buffer

The buffer holds elements and nothing else. There is no shape word, no stride, no scale, and no header in memory: the shape is in the type, the order is in the type, and the scale belongs to the consumer. A matrix in RAM is $R * C$ bytes of `i8`, and the only thing between a coordinate and one of those bytes is the offset the order computes.

That is the difference the crate is built around. A format needs a description of its layout; this one does not, because the coordinate that names an element is the same on both sides of a transfer.