# A Simplified Layout Algebra for GPU Kernels

Silares
contact@silares.com

November 11, 2026

**Abstract**

The layout algebra behind NVIDIA's CuTe library describes how tensor tiles map onto threads and memory. A layout is a shape with a stride for each of its entries, so the address of a coordinate is the sum of its indices times their strides. As formalized by Shah and by Cecka, its central operations, such as composition, complement, division, product, are guarded by at least six interacting admissibility conditions. We show that this complexity is an artifact of a representational choice, not of the domain.

We present a layout algebra in which every layout names the space its inputs and outputs live in: logical indices, physical indices, or thread-value indices. A layout into logical space must be a bijection from its $N$ coordinates onto $[0, N)$. A layout into physical space must be alias-free: different coordinates land on different addresses. The exception is broadcast, where many threads read one value on purpose, and the layout has to mark where that happens. Nothing maps out of a physical index.

Composing $f$ with $g$ requires only that $f$'s range be exactly $g$'s flattened set of coordinates. To emit the index arithmetic, naively, we can always compute $f$, split that by division and remainder into $g$'s coordinates, and feed it into $g$. Often the composite can be simplified to a layout as well. We show that if it can be written this way the form is unique, and we emit that simplified form.

A warp's or a thread's slice is a *restriction*. A restricted layout is no longer composable, which forces the user to settle the whole view of a tile before taking slices.

We validate the algebra by showing that CuTe's operations, such as composition, complement, and division, can be reproduced in it. Furthermore, we provide an implementation in OCaml, and use it to rederive the atoms and usage patterns of CuTe.

The implementation is roughly 1,000 lines of OCaml.

## 1 Introduction

For GPU kernels, it is important to have a scheme for describing how to move tiles of tensors between global memory, shared memory and registers. The de facto vocabulary for such schemes is the CuTe layout algebra underlying NVIDIA's CUTLASS library [1, 2]. The most fundamental concept is the layout, which we shall reuse heavily.

**Definition 1** (Layout)**.** A *shape* is a positive integer $n$, or a tuple $(S_1, \ldots, S_k)$ of shapes; its *size* is $\mathrm{size} n = n$ and $\mathrm{size}(S_1, \ldots, S_k) = \prod_i \mathrm{size} S_i$. A *coordinate* of $S$ is an integer $i \in [0, n)$, respectively a tuple $(c_1, \ldots, c_k)$ of coordinates of the $S_i$; write $\mathrm{Coord}(S)$ for the set of them. A *layout* $S{:}D$ is a shape $S$ together with a *stride* $D$ of the same structure: an integer where $S$ is a positive integer, and a tuple $(D_1, \ldots, D_k)$ where $S$ is $(S_1, \ldots, S_k)$. Its index function

$\mathrm{Coord}(S) \to \mathbb{N}$ is defined by recursion on the shape,

$$(n{:}d)(i) = i\,d, \qquad \big((S_1,\dots,S_k){:}(D_1,\dots,D_k)\big)(c_1,\dots,c_k) = \sum_{j=1}^{k} (S_j{:}D_j)(c_j).$$

Flattening is a bijection $\langle\cdot\rangle_S : \mathrm{Coord}(S) \to [0, \mathrm{size}S)$, taken row-major, so that the last component varies fastest, with inverse $\mathrm{unflatten}_S$. A coordinate may therefore be given as a single integer, and we say that integer *is* the coordinate. The *cosize* of the layout is the least $N$ with its image inside $[0, N)$, which is one plus the largest value it takes. The layout is *dense* when its image is an initial segment $[0, N)$ of the integers, and a *dense bijection* when it is moreover injective, in which case $N = \mathrm{size}S$ and every value below $N$ is taken exactly once. A *mode* is one entry of the shape together with the matching entry of the stride. Two layouts *concatenate*: for $A = S{:}D$ and $B = S'{:}D'$, the layout $(A, B)$ is $(S, S'){:}(D, D')$, and it sends $(c, c')$ to $A(c) + B(c')$.

Layouts are fundamentally the correct concept here, because physical memory is linearly addressed while the logical space is multidimensional and dense as we imagine it. They are a natural way to abstract and describe those mappings.

The algebra has four operations. Three of them, composition, complement and logical division, are formalized by Shah [3]; the fourth, logical product, we take from the CUTLASS implementation. The *complement* of a layout $B = (N_0,\dots,N_\alpha){:}(d_0,\dots,d_\alpha)$, sorted by stride, with respect to an integer $M$, is

$$B^*_M \;=\; \Big(d_0,\ \tfrac{d_1}{N_0 d_0},\ \dots,\ \tfrac{M}{N_\alpha d_\alpha}\Big) : \Big(1,\ N_0 d_0,\ \dots,\ N_\alpha d_\alpha\Big),$$

chosen so that the concatenation $(B, B^*_M)$ has size $M$ and its layout function is a bijection of $[0, M)$ onto itself. *Logical division* is $A \,/\, B = A \circ (B, B^*_{\mathrm{size}A})$, which reindexes $A$ so that one mode runs inside the tile $B$ selects and the other runs across copies of it. *Logical product* is $A \otimes B = (A, A^*_M \circ B)$, where $M = \mathrm{size}A \cdot \mathrm{cosize}B$, the smallest $M$ for which the composition is defined. Its first mode is one copy of $A$, and its second walks $B$ through the addresses $A$ does not occupy, so the result is $\mathrm{size}B$ copies of $A$ placed in the pattern $B$ names.

These operations are critical for constructing the layouts one usually wants. There is, however, a lot of complexity underneath them. Layouts are in general non-injective and non-surjective onto $[0, M)$, and CuTe always wants to write the composite as a layout again. This causes at least six condition checks. Two are in the complement, whose shape entries are quotients and so must divide: $N_{i-1}d_{i-1} \mid d_i$ for every $i$, and $N_\alpha d_\alpha \mid M$. The other four belong to composition. Write the flattened shape of $A$ as $M_0 \cdots M_\alpha$. It is *left divisible* by $r$ when $r = M_0 \cdots M_{i-1} \cdot c$ for some $i$, where $c$ is smaller than $M_i$ and divides it, and *weakly left divisible* when the same holds except that $c$ need not divide $M_i$. For each mode $(N){:}(r)$ of $B$ this is three checks, since the prefix must divide $r$, the leftover $c$ must divide $M_i$, and what remains after $r$ must be weakly left divisible by $N$. One more is required across modes, that the intervals on which they act be pairwise disjoint [3]. None of these conditions is visible in the CuTe API, yet all of them are visible failures.

We try to resolve those complexities by restricting what a layout can be.

**Explicit index spaces.** The input and the output of a layout each represent one of three things: a logical index into a tile, a thread-value index, or a physical offset in memory. Therefore a layout should declare which of the three its input and its output are, and we allow only the

maps worth having between them. Logical and thread-value indices map into one another in either direction. Logical indices map into physical offsets. Nothing maps out of a physical offset. By design, a thread reaches memory by composing its map into logical space with the storage map.

**Requirements that follow from the space.** We allow those maps and not others because the three spaces do not carry the same obligations. Logical and thread-value indices have to account for their contents losslessly, so a map into either must be a dense bijection. Composition can only happen when one layout's image and the other's domain agree completely, which is natural because a physical offset is forbidden as a domain. A physical offset carries far fewer requirements, since gaps in memory are common and sharing one address between threads, to read the same element, is often what is wanted, so a map into physical need only be alias-free, meaning that distinct coordinates reach distinct addresses except where the layout marks a broadcast.

**Composition and simplification.** The composite of two layouts is not a layout in general. But we can decide if it can be written as a layout, and when it is a layout we emit one weighted sum of the coordinates. When it is not, we emit the composition literally, at the cost of a longer expression and more work at run time.

## 2 Our construction of layouts

We build on top of Definition 1. First we define the spaces.

**Definition 2** (Spaces)**.** Each side of a layout carries a space from $\{\text{logical}, \text{physical}, \text{thread-value}\}$, written $A \to B$. A layout may be annotated $A \to B$ for $A, B \in \{\text{logical}, \text{thread-value}\}$, or logical $\to$ physical, and no other pair is constructible. Composition takes $A \to$ logical and logical $\to C$ to $A \to C$, so thread-value $\to$ physical exists only as a composite.

**Definition 3** (Replication)**.** Extend Definition 1 by allowing the stride against a positive integer $n$ to be the mark $\perp$ in place of an integer, with

$$(n{:}\perp)(i) = 0 \quad \text{for every } i \in [0, n).$$

A zero stride is rejected rather than read as replication. This is because replication is legal for reads and not for writes, since simultaneous loads of one address are a broadcast while simultaneous stores are a race. It is better not to use 0, so that the two can never be mixed.

**Definition 4** (Swizzled layout)**.** A *swizzle* is a bijection physical $\to$ physical of the form

$$\sigma(x) \;=\; x \oplus \big(((x \gg \mathit{src}) \wedge (2^b - 1)) \ll \mathit{dst}\big),$$

its two fields $[\mathit{src}, \mathit{src}{+}b)$ and $[\mathit{dst}, \mathit{dst}{+}b)$ disjoint. A layout $A \to$ physical followed by a swizzle is a *swizzled layout*, again $A \to$ physical.

A swizzle runs from physical to physical, so it can only ever be the last step, and being a bijection it sends distinct addresses to distinct addresses. It therefore leaves the conditions below untouched, and matters only in the arithmetic that is finally emitted.

**Definition 5** (Composition)**.** Let $f : A \to \text{logical}$ and let $g : \text{logical} \to C$ have domain shape $S_g$. The *composite* $g \circ f : A \to C$ is defined when $f$ is a dense bijection onto $[0, \text{size}S_g)$, and is the map

$$g \circ f \;=\; c \mapsto g\big(\text{unflatten}_{S_g}(f(c))\big),$$

whose domain shape is that of $f$.

A chain is therefore one thread-value $\to$ logical map, then any number of logical $\to$ logical maps, then one logical $\to$ physical map.

**Definition 6** (Declared replicas)**.** For a layout $L$ with domain shape $S$, define a relation $\sim_L$ on $\text{Coord}(S)$:

- if $L$ is a layout, $c \sim_L c'$ when $c$ and $c'$ agree at every mode whose stride is an integer, that is, they differ only at modes with stride $\perp$;
- if $L = g \circ f$ with $g$ of domain shape $S_g$, then $c \sim_L c'$ when $\text{unflatten}_{S_g}(f(c)) \sim_g \text{unflatten}_{S_g}(f(c'))$.

**Definition 7** (Validity)**.** Let $L : A \to B$ be a layout with domain shape $S$. Then $L$ is *valid* when

$$\begin{cases} L \text{ is a dense bijection} & \text{if } B \in \{\text{logical}, \text{thread-value}\}, \\ L(c) = L(c') \implies c \sim_L c' & \text{if } B = \text{physical}. \end{cases}$$

It is *write-valid* when $B =$ physical and in addition

$$L(c) = L(c') \implies c = c'.$$

Only a write-valid layout may be the target of a store.

Whether a layout is a dense bijection can be decided without enumeration.

**Lemma 1** (Dense bijections)**.** *Let $L$ be a layout with no stride $\perp$, its modes carrying sizes $n_1, \ldots, n_k$ and strides $s_1, \ldots, s_k$ with every $n_i \geq 2$, and put $N = \prod_i n_i$. Then $L$ is a dense bijection onto $[0, N)$ if and only if, after sorting the modes by stride, $s_{(1)} = 1$ and $s_{(j+1)} = s_{(j)}\, n_{(j)}$ for every $j$: the sorted strides are the running products of the sorted sizes.*

*Proof.* ($\Leftarrow$) With running-product strides the map is the flattening of Definition 1 for the shape $(n_{(k)}, \ldots, n_{(1)})$, a bijection onto $[0, N)$. Positions of size 1 contribute nothing and are dropped beforehand.

($\Rightarrow$) Sort the modes by stride and write $N_j = \prod_{i \leq j} n_{(i)}$ (so $N_0 = 1$). *Claim:* $s_{(j)} \leq N_{j-1}$ for every $j$. Every image value below $s_{(j)}$ comes from a coordinate supported on modes $1, \ldots, j-1$, since a nonzero index at any later mode contributes at least $s_{(j)}$ and no contribution is negative; those modes produce at most $N_{j-1}$ distinct values, so surjectivity onto $[0, s_{(j)})$ forces $s_{(j)} \leq N_{j-1}$. Consequently

$$\textstyle\sum_j (n_{(j)} - 1)\, s_{(j)} \;\leq\; \sum_j (n_{(j)} - 1)\, N_{j-1} \;=\; N - 1,$$

the last equality by telescoping. But the left side is the maximum of the image, and surjectivity onto $[0, N)$ requires the maximum to be exactly $N - 1$. The two sums therefore agree, and since each term on the left is at most the corresponding term on the right, they agree termwise; as $n_{(j)} - 1 \geq 1$ this gives $s_{(j)} = N_{j-1}$ for every $j$, which is the running-product condition. $\square$

## 3 Operations

We define the following operations. Each takes a layout $L$, and where a second argument is needed it is written out; $S$ ranges over shapes and $A$ over layouts.

canonical($S$) for a flat shape $S = (n_1, \ldots, n_k)$, the layout

$$(n_1, \ldots, n_k) : \Big( \textstyle\prod_{i>1} n_i,\ \prod_{i>2} n_i,\ \ldots,\ n_k,\ 1 \Big).$$

For an entry that is itself a shape, we take the canonical layout of that shape, with every stride multiplied by the product of the sizes to its right.

$$\text{canonical}((3,4)) = (3,4){:}(4,1)$$

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 | 7 |
| 2 | 8 | 9 | 10 | 11 |

split($n$:$d$, $S$) for size$S = n$, the shape $S$ carrying the strides of canonical($S$) each multiplied by $d$, which for a flat $S = (m_1, \ldots, m_r)$ is

$$(m_1, \ldots, m_r) : \Big( d \textstyle\prod_{i>1} m_i,\ d \prod_{i>2} m_i,\ \ldots,\ d\, m_r,\ d \Big)$$

and for a nested $S$ follows its nesting. The argument $n{:}\perp$ gives $S$ with every stride $\perp$.

6:2

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 4 | 6 | 8 | 10 |

$$\text{split}(6{:}2,\ (2,3)) = (2,3){:}(6,2)$$

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 2 | 4 |
| 1 | 6 | 8 | 10 |

divide($L, S$) partitions $L$ into tiles of shape $S$. Let $L = (n_1, \ldots, n_m){:}(d_1, \ldots, d_m)$ and let $S = (k_1, \ldots, k_m)$ with $k_i \mid n_i$. Each mode splits as

$$\text{split}\big(n_i{:}d_i,\ (n_i/k_i,\ k_i)\big) \ = \ (n_i/k_i,\ k_i) : (k_i d_i,\ d_i),$$

and collecting the second component of every split into one group and the first into another gives

$$\text{divide}(L, S) \ = \ \big((k_1, \ldots, k_m),\ (n_1/k_1, \ldots, n_m/k_m)\big) : \big((d_1, \ldots, d_m),\ (k_1 d_1, \ldots, k_m d_m)\big).$$

The first group indexes a mode inside a tile and the second indexes which tile. It is easy to see that precomposing with the flattening that sends each pair $(b_i, t_i)$ back to $c_i = b_i k_i + t_i$ recovers $L$.

$L = (4,6){:}(6,1)$, coloured by tile

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 |
| 1 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 12 | 13 | 14 | 15 | 16 | 17 |
| 3 | 18 | 19 | 20 | 21 | 22 | 23 |

divide$(L, (2,3))$: tile coordinate $(t_1, t_2)$ as row, tile $(b_1, b_2)$ as column

| | (0,0) | (0,1) | (1,0) | (1,1) |
|---|---|---|---|---|
| (0,0) | 0 | 3 | 12 | 15 |
| (0,1) | 1 | 4 | 13 | 16 |
| (0,2) | 2 | 5 | 14 | 17 |
| (1,0) | 6 | 9 | 18 | 21 |
| (1,1) | 7 | 10 | 19 | 22 |
| (1,2) | 8 | 11 | 20 | 23 |

repeat$(L, A)$ copies of $L$ arranged in the layout of $A$, which is required to be a valid logical $\to$ logical layout and so a dense bijection:

$$\text{repeat}(L, A) \;=\; (S_A, S_L) : \big((\text{cosize}L)\, D_A,\; D_L\big),$$

where $L = S_L{:}D_L$, $A = S_A{:}D_A$, and $c\,D$ multiplies every stride of $D$ by $c$. If $L$ is a dense bijection then cosize$L$ = size$L$, and the addresses (size$L$) $A(a) + L(v)$ take every value in $[0, \text{size}A \cdot \text{size}L)$ exactly once: the result is a dense bijection.

When $L$ carries a swizzle $\sigma$, repeat is defined only if cosize$L$ is a power of two and both fields of $\sigma$ lie below it. Then $\sigma\big((\text{cosize}L)\, a + x\big) = (\text{cosize}L)\, a + \sigma(x)$ and every copy is swizzled alike. Otherwise the copy index reaches the bits $\sigma$ reads, and the operation rejects.

interleave$(L, A)$ the same copies interleaved element by element rather than laid out one after another, with $A$ again a valid logical $\to$ logical layout:

$$\text{interleave}(L, A) \;=\; (S_A, S_L) : \big(D_A,\; (\text{size}A)\, D_L\big).$$

The coordinate $(a, v)$ lands at $A(a) + (\text{size}A)\, L(v)$, so each element of $L$ becomes a run of size$A$ consecutive addresses holding that element of every copy, where repeat instead gives each copy a run of its own. If $L$ is a dense bijection the result is again a dense bijection onto $[0, \text{size}A \cdot \text{size}L)$. On a swizzled $L$ it has no defined meaning and rejects.

$L = (2,2){:}(2,1)$

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 2 | 3 |

repeat$(L,\ 3{:}1) = (3, (2,2)){:}(4, (2,1))$

| | (0,0) | (0,1) | (1,0) | (1,1) |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 | 7 |
| 2 | 8 | 9 | 10 | 11 |

interleave$(L,\ 3{:}1) = (3, (2,2)){:}(1, (6,3))$

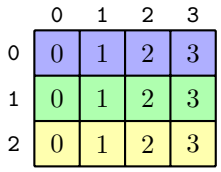
| | (0,0) | (0,1) | (1,0) | (1,1) |
|---|---|---|---|---|
| 0 | 0 | 3 | 6 | 9 |
| 1 | 1 | 4 | 7 | 10 |
| 2 | 2 | 5 | 8 | 11 |

broadcast$(L, S)$ size$S$ copies of $L$ at the same addresses:

$$\text{broadcast}(L, S) \;=\; (S, S_L) : (\bot,\; D_L),$$

writing $\perp$ for a stride of $\perp$ at every entry of $S$. The coordinate $(a, v)$ lands at $L(v)$ whatever $a$ is. The second argument is a shape and not a layout because the copies share their addresses and so have no strides to carry. The result is valid but not write-valid.

broadcast(4:1, (3)) $= (3, 4):(\perp, 1)$: copy as row

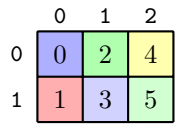



inverse($L$) for $L$ a dense bijection with modes $(n_1, \ldots, n_m):(d_1, \ldots, d_m)$. Give mode $i$ the canonical weight $w_i = \prod_{j>i} n_j$ and list the modes in decreasing order of stride. Then

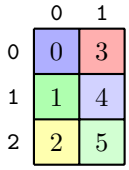
$$\text{inverse}(L) \;=\; (n_{(1)}, \ldots, n_{(m)}) : (w_{(1)}, \ldots, w_{(m)}),$$

again a dense bijection. Read both as maps on $[0, N)$, each unflattening the integer it is given into its own shape, and they are mutually inverse.

$L = (2, 3):(1, 2)$, coloured by element




inverse($L$) $= (3, 2):(1, 3)$, at address $2a + b$ the element $a + 3b$




## 4 Deciding whether a composite is a layout

While a composite can be natively emitted as is, sometimes we can simplify the codegen if that composite is actually itself a layout.

Fix a map $f : \text{Coord}(S) \to \mathbb{Z}$ on a shape $S = (n_1, \ldots, n_m)$. We take $S$ flat, since a nested shape has the coordinates of the flat shape of its leaves, regrouped. Write $c = (c_1, \ldots, c_m)$ for a coordinate and $\mathbf{0}$ for the origin. Every layout sends $\mathbf{0}$ to 0, and composition and swizzle preserve this, so $f(\mathbf{0}) = 0$ for every composite. Maps take values in $\mathbb{Z}$ throughout this section: strides are integers, and Section 5 subtracts a constant.

**Definition 8** (Refinement)**.** A *refinement* of a flat shape $S = (n_1, \ldots, n_m)$ is a shape $S' = (S_1, \ldots, S_m)$ with each $S_i$ flat and size$S_i = n_i$. A coordinate of $S_i$ is identified with its flattening in $[0, n_i)$ (Definition 1), so Coord($S'$) and Coord($S$) are the same set. A map $f : \text{Coord}(S) \to \mathbb{Z}$ *is the layout* $S'{:}D'$ when

$$f(c_1, \ldots, c_m) \;=\; (S'{:}D')\big(\text{unflatten}_{S_1}(c_1),\; \ldots,\; \text{unflatten}_{S_m}(c_m)\big)$$

for every coordinate; for $S' = S$ this says $f = S{:}D$. The question of this section is whether $f$ is a layout $S'{:}D'$ for some refinement $S'$ of $S$ and some stride $D'$.

**Lemma 2** (Separability)**.** *Let $S = (n_1, \ldots, n_m)$ be a flat shape and $f : \mathrm{Coord}(S) \to \mathbb{Z}$. For $1 \le i \le m$ let $e_i$ be the coordinate with entry $i$ equal to 1 and every other entry 0, and define $g_i : [0, n_i) \to \mathbb{Z}$ by $g_i(v) = f(v\, e_i)$. Then $f$ is a layout $S'{:}D'$ for some refinement $S'$ of $S$ if and only if*

$$f(c) = \sum_{i=1}^{m} g_i(c_i) \quad \textit{for every } c = (c_1, \ldots, c_m) \in \mathrm{Coord}(S), \tag{1}$$

*and each $g_i$ is a layout of some flat shape of size $n_i$.*

*Proof.* ($\Rightarrow$) Let $f = S'{:}(D_1, \ldots, D_m)$ with $S' = (S_1, \ldots, S_m)$. By Definition 1,

$$f(c) = \sum_{i=1}^{m} (S_i{:}D_i)\big(\mathrm{unflatten}_{S_i}(c_i)\big) \quad \text{for every } c.$$

At $c = v\, e_i$ every term but the $i$th is a layout evaluated at coordinate 0, hence 0, so $g_i(v) = (S_i{:}D_i)(\mathrm{unflatten}_{S_i}(v))$: the $i$th term is $g_i(c_i)$, which is (1), and $g_i = S_i{:}D_i$. ($\Leftarrow$) If $g_i = S_i{:}D_i$ for each $i$, the same identity read backwards gives $f = (S_1, \ldots, S_m){:}(D_1, \ldots, D_m)$. □

Checking (1) is one pass over $\mathrm{Coord}(S)$. What remains is one coordinate at a time: a map $g : [0, n) \to \mathbb{Z}$ with $g(0) = 0$, and the question whether there are a flat shape $T$ of size $n$ and a stride $D$ with $g = (T{:}D) \circ \mathrm{unflatten}_T$. To decide it we need this composite as a formula in $v$.

Number the entries of $T$ from the right, $T = (m_k, \ldots, m_1)$, so that indices increase with the strides below. By Definition 1, $\mathrm{unflatten}_T(v)$ is the coordinate $(x_k, \ldots, x_1)$ of $T$ with

$$v = \sum_{j=1}^{k} x_j\, w_j, \qquad w_j = m_1 m_2 \cdots m_{j-1},$$

the $w_j$ being the strides of $\mathrm{canonical}(T)$; so $w_1 = 1$, $w_{j+1} = w_j m_j$, and $w_k m_k = n$. Solving for the components,

$$x_j = \lfloor v / w_j \rfloor \bmod m_j,$$

which we write $d_j(v)$. Therefore

$$\big(T{:}(s_k, \ldots, s_1)\big)\big(\mathrm{unflatten}_T(v)\big) \;=\; \sum_{j=1}^{k} s_j\, d_j(v),$$

and $g$ is the layout $T{:}(s_k, \ldots, s_1)$ exactly when $g(v)$ equals this sum for every $v \in [0, n)$.

Two remarks turn the search for $T$ into a search for the $w_j$. An entry with $m_j = 1$ has $d_j = 0$ and contributes nothing, so we assume $T$ has none; then $1 = w_1 < w_2 < \cdots < w_k < n$, each dividing the next and $w_k$ dividing $n$, which we write $1 = w_1 \mid w_2 \mid \cdots \mid w_k \mid n$ and call the *chain* of $T$. Conversely, every such chain is the chain of exactly one such $T$, given by $m_j = w_{j+1}/w_j$ with $w_{k+1} = n$. Theorem 1 finds $T$ by finding its chain.

Each component $d_j(v) = \lfloor v/w_j \rfloor \bmod m_j$ is the remainder of $\lfloor v/w_j \rfloor$ on division by $m_j$. The lemma rewrites $\sum_j s_j\, d_j(v)$ with floors alone.

**Lemma 3** (Components as differences of floors)**.** *For positive integers $w \mid w'$ and every integer $v \ge 0$,*

$$\lfloor v/w \rfloor \bmod (w'/w) \;=\; \lfloor v/w \rfloor - (w'/w)\lfloor v/w' \rfloor.$$

*Let $T$ be a flat shape of size $n$ with $k$ entries, with chain $1 = w_1 \mid \cdots \mid w_k \mid n$, entry sizes $m_j$, and components $d_j$ as above, and let integers $s_1, \ldots, s_k$ and $a_1, \ldots, a_k$ be related by*

$$a_j = s_j - m_{j-1}\, s_{j-1} \quad (1 \le j \le k), \qquad m_0 s_0 = 0,$$

*a relation that determines either family from the other. Then for a map $g : [0, n) \to \mathbb{Z}$,*

$$g(v) = \sum_{j=1}^{k} s_j\, d_j(v) \ \text{ for every } v \in [0,n) \quad \Longleftrightarrow \quad g(v) = \sum_{j=1}^{k} a_j \lfloor v/w_j \rfloor \ \text{ for every } v \in [0,n),$$

*and when these hold, $s_j = g(w_j)$.*

*Proof.* Write $q = \lfloor v/w \rfloor$ and $r = w'/w$. The remainder of $q$ on division by $r$ is $q - r\lfloor q/r \rfloor$, and $\lfloor q/r \rfloor = \lfloor v/(wr) \rfloor = \lfloor v/w' \rfloor$ for positive integers $w, r$, which is the identity. With $w = w_j$, $w' = w_{j+1}$, and $w'/w = m_j$ it reads

$$d_j(v) = \lfloor v/w_j \rfloor - m_j \lfloor v/w_{j+1} \rfloor,$$

where for $j = k$, $\lfloor v/w_{k+1} \rfloor = \lfloor v/n \rfloor = 0$ on $[0, n)$. Hence

$$\sum_{j=1}^{k} s_j\, d_j(v) = \sum_{j=1}^{k} s_j \lfloor v/w_j \rfloor - \sum_{j=1}^{k-1} s_j m_j \lfloor v/w_{j+1} \rfloor = \sum_{j=1}^{k} \big(s_j - m_{j-1} s_{j-1}\big) \lfloor v/w_j \rfloor = \sum_{j=1}^{k} a_j \lfloor v/w_j \rfloor,$$

which is the equivalence. The relation gives $s_j = a_j + m_{j-1} s_{j-1}$, so either family determines the other. Finally, $d_l(w_j) = \lfloor w_j/w_l \rfloor \bmod m_l$ is 0 for $l > j$ because $w_j < w_l$, is 1 for $l = j$ because $m_j \ge 2$, and is 0 for $l < j$ because $w_j/w_l$ is a multiple of $w_{l+1}/w_l = m_l$; so $g(w_j) = \sum_l s_l\, d_l(w_j) = s_j$. □

With floors alone, first differences read the chain off $g$, because $\lfloor t/w \rfloor - \lfloor (t-1)/w \rfloor$ is 1 if $w \mid t$ and 0 otherwise.

**Theorem 1** (Linear recognition)**.** *Let $n \ge 2$ (for $n = 1$, $g = 0$ is the layout (1):0), let $g : [0, n) \to \mathbb{Z}$ with $g(0) = 0$, and let $h(t) = g(t) - g(t-1)$ for $1 \le t < n$. Run the following scan. Keep a copy $\rho$ of $h$, an integer $w_{\text{last}}$ initially 1, and a set $R$ initially empty. For $t = 1, 2, \ldots, n-1$ in order, if $\rho(t) \ne 0$: if $t \nmid n$ or $w_{\text{last}} \nmid t$, reject; otherwise put $t$ in $R$, set $a_t = \rho(t)$ and $w_{\text{last}} = t$, and replace $\rho(t')$ by $\rho(t') - a_t$ for every multiple $t'$ of $t$ in $[t, n)$. If no $t$ rejects, accept.*

*Then $g$ is a layout of some flat shape of size $n$ if and only if the scan accepts. If it accepts, $\{1\} \cup R$ is the chain of a flat shape $T$ of size $n$; $g$ is the layout $T{:}(s_k, \ldots, s_1)$ with $s_j = g(w_j)$; and every flat shape of size $n$ without entries of size 1 of which $g$ is a layout has a chain containing $\{1\} \cup R$. We call this $T$ the coarsest. The scan takes $O(n)$ arithmetic operations.*

*Proof.* By Lemma 3, $g$ is a layout of the shape with chain $W$ if and only if there are integers $a_w$, $w \in W$, with $g(v) = \sum_{w \in W} a_w \lfloor v/w \rfloor$ for every $v \in [0, n)$. Since $\lfloor t/w \rfloor - \lfloor (t-1)/w \rfloor = [\, w \mid t \,]$ and $g(0) = 0 = \sum_w a_w \lfloor 0/w \rfloor$, this is equivalent to

$$h(t) = \sum_{w \in W} a_w\, [\, w \mid t \,] \quad \text{for } 1 \le t < n. \tag{2}$$

Suppose (2) holds for some chain $W$ and coefficients $a_w$. We show by induction on $t$ that when the scan reaches $t$, $R = \{\, w \in W : w < t,\ a_w \ne 0 \,\}$ with the recorded $a_w$ equal to those of

(2), and $\rho(t) = a_t$ if $t \in W$ and $\rho(t) = 0$ otherwise. At $t = 1$, $R$ is empty and $\rho(1) = h(1) = a_1$ because $1 \in W$ is the only divisor of 1. Given the claim at $t$: $\rho(t) = h(t) - \sum_{w \in R,\, w|t} a_w = \sum_{w \in W,\, w|t} a_w - \sum_{w \in W,\, w<t,\, w|t} a_w = [\, t \in W \,]\, a_t$. If $\rho(t) \neq 0$ then $t \in W$ and $a_t \neq 0$; $t \mid n$ because every element of $W$ divides $n$; and $w_{\text{last}}$, which is the largest element of $R$ or else 1, lies in $W$ below $t$ and so divides $t$ because $W$ is a chain. The scan records $t$ with $a_t$, and the claim holds at $t+1$; if $\rho(t) = 0$ nothing is recorded and the claim holds at $t+1$ as well. So the scan accepts, and $R \subseteq W$; with $1 \in W$, $\{1\} \cup R \subseteq W$.

Conversely suppose the scan accepts. At each $t$ either $\rho(t)$ was 0 or $a_t = \rho(t)$ was subtracted from it, and later steps change $\rho$ only at or beyond their own $t$; so at the end $\rho \equiv 0$, that is, $h(t) = \sum_{w \in R,\, w|t} a_w$ for $1 \leq t < n$. Every element of $R$ divides $n$, lies below $n$, and is a multiple of the element recorded before it or of 1, so $\{1\} \cup R$ is a chain; let $T$ be its shape, and put $a_1 = 0$ if $1 \notin R$. Then (2) holds for $W = \{1\} \cup R$, so $g$ is a layout of $T$ by Lemma 3, with strides $s_j = g(w_j)$.

Each element of $R$ is at least twice the one before, so the subtractions touch fewer than $n + n/2 + n/4 + \cdots < 2n$ entries of $\rho$; the rest of the scan is one pass over $t$. □

The methodology is similar to Appendix A.2.1 of Axe [7].

Codegen therefore emits one of two expressions. If the map is a layout $S'{:}D'$ for a refinement $S' = (S_1, \ldots, S_m)$, it emits $\sum_{i,j} s_{ij}\, d_{ij}(c_i)$, with $d_{ij}$ the components of $\text{unflatten}_{S_i}(c_i)$, for the coarsest $S'$; no such expression has fewer terms. Otherwise it emits the composite as Definition 5 writes it: the value of the right operand, unflattened into the left operand's domain shape, evaluated by the left operand. Section 5 extends this to restricted maps, which do not vanish at the origin.

# 5 Restriction

A warp or a thread uses one slice of an address map: the map with its own warp and thread coordinates fixed.

**Definition 9** (Restriction)**.** Let $f : \text{Coord}(S) \to \mathbb{N}$ with $S = (n_1, \ldots, n_m)$ flat, let $F \subseteq \{1, \ldots, m\}$, and let $p_i \in [0, n_i)$ for each $i \in F$. The *restriction* of $f$ to $p$ is the map $f|_p$ on the coordinates of the shape $(n_i)_{i \notin F}$ given by

$$f|_p(c) = f(c'), \qquad c'_i = \begin{cases} p_i & i \in F, \\ c_i & i \notin F. \end{cases}$$

$f|_p$ is in general not a layout: $f|_p(\mathbf{0})$ is the contribution of the fixed entries, which need not be 0, while every layout vanishes at $\mathbf{0}$.

$f|_p$ is in general not dense, since its image omits the other slices, so it is not a valid map into logical or thread-value (Definition 7) and cannot be the right operand of a composition (Definition 5). Restriction is therefore allowed only for maps into physical, and is applied after composition, where the warp and thread coordinates are entries of the composite's domain. Restricting a restriction is restricting by the union of the fixed entries.

The address expression of a slice is Section 4 applied to $f|_p - f|_p(\mathbf{0})$, which vanishes at the origin, plus the constant $f|_p(\mathbf{0})$; when that map is not a layout, the composite's expression is emitted with the fixed coordinates replaced by their values. The difference can take negative values, which is why Section 4 is stated over $\mathbb{Z}$: for $(2,2){:}(2,1)$ under the swizzle $\{1;1;0\}$,

$f(a,b) = (2a+b) \oplus a$, fixing $a = 1$ gives $f|_p(b) = 3 - b$ and $f|_p - f|_p(\mathbf{0}) = -b$. The emitted arithmetic then carries a negative coefficient. For $f = (4,8,4){:}(32,1,8)$, $F = \{1\}$ and $p_1 = 2$, the slice is $f|_p(r,v) = 64 + r + 8v$.

## 6 CuTe in this algebra

CuTe lists the modes of a layout with the first varying fastest; we reverse the list, recursively, and the two then agree as functions on the same integers. CuTe's composition$(A, B)$ applies $B$ first and is $A \circ B$ in our algebra, with $B$ the right operand of Definition 5. We take the four operations in the order of Section 1, state what each is in our algebra, and name the documented examples [1] on which the two are checked to agree.

**Composition.** In CuTe, composition$(A, B)$ is the layout $L$ with $L(c) = A(B(c))$ for every $c \in \mathrm{Coord}(S_B)$. In our algebra, $A \circ B$ is the map $c \mapsto A(\mathrm{unflatten}_{S_A}(B(c)))$, defined whenever $B$ is a dense bijection onto $[0, \mathrm{size}A)$, and Theorem 1 decides whether it is a layout. For such a $B$, when CuTe's composition exists the two are the same function, so Theorem 1 accepts and returns the coarsest shape that realizes it. Checked on the three documented examples,

$$\begin{aligned}
\mathrm{composition}\big((6,2){:}(8,2),\ (4,3){:}(3,1)\big) &= \big((2,2),3\big){:}\big((24,2),8\big),\\
\mathrm{composition}\big(20{:}2,\ (5,4){:}(4,1)\big) &= (5,4){:}(8,2),\\
\mathrm{composition}\big((10,2){:}(16,4),\ (5,4){:}(1,5)\big) &= \big(5,(2,2)\big){:}\big(16,(80,4)\big),
\end{aligned}$$

as functions at every coordinate; in all three $B$ is a dense bijection onto $[0, \mathrm{size}A)$.

**Complement.** In CuTe, $B^*_M$ is the layout of Section 1, defined under two divisibility conditions. In our algebra there is no complement operator. For $B = (N_0, \ldots, N_\alpha){:}(d_0, \ldots, d_\alpha)$ the two conditions make every factor of $M = d_0 \cdot N_0 \cdot \frac{d_1}{N_0 d_0} \cdot N_1 \cdots \frac{M}{N_\alpha d_\alpha}$ an integer. Let $R$ be the shape of these factors in our order, $R = \big(\frac{M}{N_\alpha d_\alpha},\ N_\alpha,\ \frac{d_\alpha}{N_{\alpha-1} d_{\alpha-1}},\ \ldots,\ N_0,\ d_0\big)$, and let $S$ be $R$ with every entry other than the $N_i$ replaced by 1. Then

$$\mathrm{divide}\big(\mathrm{canonical}(R),\ S\big) \ =\ \big(B,\ B^*_M\big)$$

after dropping entries of size 1: the tile is $B$ and the rest is its complement. Whether a given pair $(B, B^*)$ is a bijection onto $[0, M)$ is decided by Lemma 1. Checked on the six documented complements: divide(canonical(24), (4)) has tile 4:1 and rest 6:4, which are $(4{:}1)^*_{24}$ and $(6{:}4)^*_{24}$; divide(canonical((12, 2)), (4, 1)) has tile 4:2 and rest $(3,2){:}(8,1)$, which is $(4{:}2)^*_{24} = (2,3){:}(1,8)$; and all six pairs $(B, B^*_{24})$ are bijections onto $[0, 24)$.

**Division.** In CuTe, $A/B = A \circ (B, B^*_{\mathrm{size}A})$. In our algebra, for a tile given as a shape $S$ with $k_i \mid n_i$, this is divide$(A, S)$ of Section 3, and

$$\mathrm{divide}(A, S) \ =\ A \circ \mathrm{divide}\big(\mathrm{canonical}(S_A),\ S\big),$$

whose right operand is a regrouping of the identity on $[0, \mathrm{size}A)$ and so a dense bijection onto it. For a tile given as a layout $B$ it is the composite $A \circ (B, B^*_{\mathrm{size}A})$ itself: $(B, B^*_{\mathrm{size}A})$ is a bijection

onto $[0, \text{size}A)$, so Definition 5 applies with no further condition. Checked, as functions, on the documented strided tiler,

$$(4,2,3){:}(2,1,8) \,/\, 4{:}2 = \big((2,2),(2,3)\big){:}\big((4,1),(2,8)\big).$$

CuTe's `zipped_divide` groups the result as ((tile), (rest)), which is the grouping divide produces; `logical_divide` groups it per mode, a regrouping of the same layout.

**Product.** In CuTe, $A \otimes B = (A, A^*_M \circ B)$ with $M = \text{size}A \cdot \text{cosize}B$. In our algebra,

$$\text{repeat}(A,B) = (S_B, S_A){:}\big((\text{cosize}A)\, D_B,\ D_A\big).$$

When $A$ is a dense bijection, $A^*_M = (\text{cosize}B){:}(\text{size}A)$, so

$$A \otimes B = \big(A,\ S_B{:}(\text{size}A)\, D_B\big) = \text{repeat}(A,B).$$

When $\text{cosize}A > \text{size}A$ the two differ. Take $A = (2,2){:}(4,1)$ and $B = 6{:}1$. Printed as CuTe prints layouts, one cell per coordinate holding its address and coloured by the copy it belongs to, $A$ is

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 4 | 5 |

and the two products are, with the element $(i,j)$ of $A$ as the row and the copy $k$ as the column,

In CuTe: $A \otimes B$

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| (0,0) | 0 | 2 | 8 | 10 | 16 | 18 |
| (0,1) | 1 | 3 | 9 | 11 | 17 | 19 |
| (1,0) | 4 | 6 | 12 | 14 | 20 | 22 |
| (1,1) | 5 | 7 | 13 | 15 | 21 | 23 |

In our algebra: $\text{repeat}(A,B)$

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| (0,0) | 0 | 6 | 12 | 18 | 24 | 30 |
| (0,1) | 1 | 7 | 13 | 19 | 25 | 31 |
| (1,0) | 4 | 10 | 16 | 22 | 28 | 34 |
| (1,1) | 5 | 11 | 17 | 23 | 29 | 35 |

In CuTe the copy offsets are $A^*_{24} = (2,3){:}(2,8)$, that is $0, 2, 8, 10, 16, 18$, and

$$A \otimes B = \big((2,2),(2,3)\big){:}\big((4,1),(2,8)\big)$$

is a bijection onto $[0, 24)$. In our algebra the copy offsets are the multiples of $\text{cosize}A = 6$, and $\text{repeat}(A,B) = (6,(2,2)){:}(6,(4,1))$ is injective and not dense. CuTe's placement is available in our algebra: $\text{divide}(\text{canonical}(R), S)$ is the pair $(A, A^*_M)$ of the complement paragraph, and composing it on the right with $\text{repeat}(B, \text{canonical}(S_A))$, which sends $(a,b)$ to $(a, B(b))$, gives $A \otimes B$ for every dense bijection $B$; for $B = k{:}1$ that composition is the identity, and for this $A$

$$A \otimes B = \text{divide}\big(\text{canonical}((6,4)),\ (2,2)\big).$$

All of this is checked. With the operands exchanged and $B$ a dense bijection, $B \otimes A = (B,\ S_A{:}(\text{size}B)\, D_A)$, which is $\text{interleave}(A,B)$ up to the order of the two coordinate groups. CuTe's `blocked_product` and `raked_product` are both $A \otimes B$ with the two coordinate groups zipped per mode, block index first or copy index first: the same map on coordinates, regrouped. For $A = 4{:}1$ and $B = 2{:}1$, $A \otimes B = \text{repeat}(A,B) = (2,4){:}(4,1)$, `raked_product` is $(4,2){:}(1,4)$, the same map with its two coordinates exchanged, and $B \otimes A = \text{interleave}(A,B) = (2,4){:}(1,2)$.

**Conditions.** CuTe's six conditions exist because its composition and complement must return a layout, so the operands are checked in advance for the result to be expressible as one. In our algebra neither operation has to: a composite is a map, and whether it is a layout is decided afterwards, for code generation only; and a complement is never formed on its own, it is the rest of a divide. So none of the six is checked. What is checked instead is that the right operand of a composition is a dense bijection onto the left operand's coordinates, by the stride criterion of Lemma 1; that a map into logical or thread-value space is a dense bijection and a map into physical space is alias-free; and the precondition of each operation, such as $k_i \mid n_i$ for divide.

**Example.** Two hardware instructions and a kernel's use of them. The thread-value maps are transcribed from the PTX ISA fragment tables [8]. For `mma.m16n8k16`, lane $4g + t_4$ holds four accumulator values $(v_m, v_n)$, at row $g + 8v_m$ and column $2t_4 + v_n$ of the $16 \times 8$ tile; with the tile row-major,

$$F_C = \big((8,4),(2,2)\big){:}\big((8,2),(64,1)\big) : \text{thread-value} \to \text{logical}.$$

For `ldmatrix`, lane $4g + q$ holds elements $(g, 2q)$ and $(g, 2q+1)$ of an $8 \times 8$ matrix,

$$F_{\text{ld}} = \big((8,4),2\big){:}\big((8,2),1\big) : \text{thread-value} \to \text{logical}.$$

Printed as CuTe prints thread-value layouts, one cell per element of the tile, labelled with the lane and value that hold it and coloured by lane:

$F_C$: the $16 \times 8$ accumulator

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 |
| 1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 |
| 2 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 |
| 3 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 |
| 4 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 |
| 5 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 |
| 6 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 |
| 7 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 |
| 8 | T0 V2 | T0 V3 | T1 V2 | T1 V3 | T2 V2 | T2 V3 | T3 V2 | T3 V3 |
| 9 | T4 V2 | T4 V3 | T5 V2 | T5 V3 | T6 V2 | T6 V3 | T7 V2 | T7 V3 |
| 10 | T8 V2 | T8 V3 | T9 V2 | T9 V3 | T10 V2 | T10 V3 | T11 V2 | T11 V3 |
| 11 | T12 V2 | T12 V3 | T13 V2 | T13 V3 | T14 V2 | T14 V3 | T15 V2 | T15 V3 |
| 12 | T16 V2 | T16 V3 | T17 V2 | T17 V3 | T18 V2 | T18 V3 | T19 V2 | T19 V3 |
| 13 | T20 V2 | T20 V3 | T21 V2 | T21 V3 | T22 V2 | T22 V3 | T23 V2 | T23 V3 |
| 14 | T24 V2 | T24 V3 | T25 V2 | T25 V3 | T26 V2 | T26 V3 | T27 V2 | T27 V3 |
| 15 | T28 V2 | T28 V3 | T29 V2 | T29 V3 | T30 V2 | T30 V3 | T31 V2 | T31 V3 |

$F_{\text{ld}}$: one $8 \times 8$ matrix

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 |
| 1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 |
| 2 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 |
| 3 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 |
| 4 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 |
| 5 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 |
| 6 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 |
| 7 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 |

A kernel uses them through the operations. Two warps, $W = 2{:}1$, computing a $16 \times 16$ tile side by side have the thread-value map

$$\text{divide}\big(\text{canonical}((16,16)),\ (16,8)\big) \circ \text{interleave}(F_C, W),$$

with domain $(\text{warp}, (\text{lane}, \text{value}))$, and one warp loading the four $8 \times 8$ matrices of an $8 \times 32$ tile with `ldmatrix.x4` has

$$\text{divide}\big(\text{canonical}((8,32)),\ (8,8)\big) \circ \text{interleave}(F_{\text{ld}}, 4{:}1),$$

with domain $(\text{matrix}, (\text{lane}, \text{value}))$. Coloured by warp and by matrix respectively:

two warps on $16 \times 16$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 |
| 1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 |
| 2 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 |
| 3 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 |
| 4 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 |
| 5 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 |
| 6 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 |
| 7 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 |
| 8 | T0 V2 | T0 V3 | T1 V2 | T1 V3 | T2 V2 | T2 V3 | T3 V2 | T3 V3 | T0 V2 | T0 V3 | T1 V2 | T1 V3 | T2 V2 | T2 V3 | T3 V2 | T3 V3 |
| 9 | T4 V2 | T4 V3 | T5 V2 | T5 V3 | T6 V2 | T6 V3 | T7 V2 | T7 V3 | T4 V2 | T4 V3 | T5 V2 | T5 V3 | T6 V2 | T6 V3 | T7 V2 | T7 V3 |
| 10 | T8 V2 | T8 V3 | T9 V2 | T9 V3 | T10 V2 | T10 V3 | T11 V2 | T11 V3 | T8 V2 | T8 V3 | T9 V2 | T9 V3 | T10 V2 | T10 V3 | T11 V2 | T11 V3 |
| 11 | T12 V2 | T12 V3 | T13 V2 | T13 V3 | T14 V2 | T14 V3 | T15 V2 | T15 V3 | T12 V2 | T12 V3 | T13 V2 | T13 V3 | T14 V2 | T14 V3 | T15 V2 | T15 V3 |
| 12 | T16 V2 | T16 V3 | T17 V2 | T17 V3 | T18 V2 | T18 V3 | T19 V2 | T19 V3 | T16 V2 | T16 V3 | T17 V2 | T17 V3 | T18 V2 | T18 V3 | T19 V2 | T19 V3 |
| 13 | T20 V2 | T20 V3 | T21 V2 | T21 V3 | T22 V2 | T22 V3 | T23 V2 | T23 V3 | T20 V2 | T20 V3 | T21 V2 | T21 V3 | T22 V2 | T22 V3 | T23 V2 | T23 V3 |
| 14 | T24 V2 | T24 V3 | T25 V2 | T25 V3 | T26 V2 | T26 V3 | T27 V2 | T27 V3 | T24 V2 | T24 V3 | T25 V2 | T25 V3 | T26 V2 | T26 V3 | T27 V2 | T27 V3 |
| 15 | T28 V2 | T28 V3 | T29 V2 | T29 V3 | T30 V2 | T30 V3 | T31 V2 | T31 V3 | T28 V2 | T28 V3 | T29 V2 | T29 V3 | T30 V2 | T30 V3 | T31 V2 | T31 V3 |

`ldmatrix.x4` on $8 \times 32$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 | T0 V0 | T0 V1 | T1 V0 | T1 V1 | T2 V0 | T2 V1 | T3 V0 | T3 V1 |
| 1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 | T4 V0 | T4 V1 | T5 V0 | T5 V1 | T6 V0 | T6 V1 | T7 V0 | T7 V1 |
| 2 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 | T8 V0 | T8 V1 | T9 V0 | T9 V1 | T10 V0 | T10 V1 | T11 V0 | T11 V1 |
| 3 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 | T12 V0 | T12 V1 | T13 V0 | T13 V1 | T14 V0 | T14 V1 | T15 V0 | T15 V1 |
| 4 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 | T16 V0 | T16 V1 | T17 V0 | T17 V1 | T18 V0 | T18 V1 | T19 V0 | T19 V1 |
| 5 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 | T20 V0 | T20 V1 | T21 V0 | T21 V1 | T22 V0 | T22 V1 | T23 V0 | T23 V1 |
| 6 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 | T24 V0 | T24 V1 | T25 V0 | T25 V1 | T26 V0 | T26 V1 | T27 V0 | T27 V1 |
| 7 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 | T28 V0 | T28 V1 | T29 V0 | T29 V1 | T30 V0 | T30 V1 | T31 V0 | T31 V1 |

Both are dense bijections, as Definition 7 requires of a map into logical. Composed onto storage they become address maps, and Section 4 emits their code. With the $16 \times 16$ tile stored row-major with leading dimension 24, that is with $(16,16){:}(24,1)$ : logical $\to$ physical in place of $\text{canonical}((16,16))$, the two-warp map is decided to be a layout and the emitted address is

```
8*c0 + 24*c1_0_0 + 2*c1_0_1 + 192*c1_1_0 + c1_1_1
```

where `c0` is the warp and `c1_0_0`, `c1_0_1`, `c1_1_0`, `c1_1_1` are $g$, $t_4$, $v_m$, $v_n$: the address $24(g + 8v_m) + 8w + 2t_4 + v_n$. The slice for warp 1 (Section 5) is

```
24*c1_0_0 + 2*c1_0_1 + 192*c1_1_0 + c1_1_1 + 8
```

With the $8 \times 32$ tile stored with leading dimension 40, the `ldmatrix` map emits

```
8*c0 + 40*c1_0_0 + 2*c1_0_1 + c1_1
```

with `c0` the matrix and `c1_0_0`, `c1_0_1`, `c1_1` the lane's $g$ and $q$ and the value $v$. The tests build the same construction for a $32 \times 16$ tile and four warps, and the `ldmatrix` reader of a GEMM mainloop (Section 7).

**Usage.** The table lists the idioms CUTLASS kernels are written with and what each is in our algebra. $L|_{x=v}$ fixes the entries named $x$ to $v$ (Definition 9). $F$ : thread-value $\to$ logical is a valid map, such as $F_C$ above; $W$ is a dense bijection arranging the warps, as above; and $T$ : logical $\to$ thread-value is a dense bijection from the positions of a tile to the threads.

| **CuTe** | **Our algebra** |
|---|---|
| `make_layout(S, D)` | $S{:}D$, modes reversed |
| `composition(A, B)` | $A \circ B$ |
| `complement(B, M)` | rest of divide(canonical$(R), S)$, above |
| `logical_divide(A, S)`, `zipped_divide` | divide$(A, S)$ |
| `local_tile(A, S, b)` | $\text{divide}(A, S)\big\|_{\text{rest}=b}$ |
| `local_partition(A, T, t)` | $\text{divide}(A, S_T)\big\|_{\text{tile}=\text{inverse}(T)(t)}$ |
| `partition_A/B/C`, `tensor(_, k)` | $(A \circ F)\big\|_{\text{thread}=t}$; $A\big\|_k$ |
| `logical_product(A, B)`, `blocked_product`, `tile_to_shape` | repeat$(A, B)$, $A$ a dense bijection |
| `raked_product(A, B)` | repeat$(A, B)$ with the two coordinate groups exchanged in each mode |
| `logical_product(B, A)`, copies first | interleave$(A, B)$, $B$ a dense bijection, up to the order of the groups |
| stride-0 modes | broadcast$(L, S)$; read-only |
| `Swizzle<B,M,S>` $\circ\ L$ | $\sigma \circ L$ with $b = B$, $dst = M$, $src = M{+}S$ (Definition 4) |
| $k$-stage pipeline buffer | repeat$(L, k{:}1)$ |
| `TiledCopy`: `partition_S/D`, vector width $v$ | divide$(A, (1, v)) \circ$ interleave$(v{:}1,\ 32{:}1)$ over the lanes |
| `TiledMMA`: $F$ over a CTA tile with warp grid $W$ | divide$(A, S_F) \circ$ interleave$(F, W)$ |
| `right_inverse(L)`; `retile_S/D` | inverse$(L)$; inverse$(F_{\text{mma}}) \circ F_{\text{copy}}$ |
| `coalesce` | not needed: the kernel declares the shape it iterates |
| `group_modes`, `select`, `flatten` | regrouping the shape; same function |
| `recast`, `upcast` by $v$ | divide$(A, (1, \ldots, 1, v))$; the change of element size is outside the algebra |
| TMA box $S$; multicast | divide$(A, S)$; broadcast across CTAs |

Every row is one of composition, divide, restrict, repeat, interleave, broadcast, a swizzle, inverse, or a regrouping of a shape.

# 7 Evaluation

The implementation[1] is 983 lines of OCaml; the tests are 2,305.

[1] `https://github.com/silares-ai/Warpc-public-release`

**The decision procedure is correct.** The criterion of Lemma 1 agrees with enumeration on 400 random layouts. The scan of Theorem 1 is compared with an oracle that tries every chain of divisors of $n$ and checks the strides it forces at every point. Over every function $g : [0, n) \to [0, k)$ with $g(0) = 0$, for $n \in \{2, 3, 4, 6, 8, 9, 12\}$ and $k = 5$ if $n \leq 4$, 3 otherwise, that is 186,293 functions, the two accept the same 58, with the same chain.

**It matters in practice.** Of the 178 layouts the tests build from CUTLASS's atoms and CuTe's examples, 147 are affine in their coordinates, 9 are layouts of a refinement with one coordinate split, and 22 are not layouts and are emitted as composites. On 600 random composites of a dense bijection with a padded storage map, 524 are layouts of some refinement, and their emitted forms contain 686 divisions and moduli where the composites contain 1,026. With a random swizzle on the storage map, none of 300 is a layout.

**The algebra is enough for a kernel.** The SM90 GMMA $64 \times 128$ accumulator, built from the SM80 `mma.m16n8k16` $C$ fragment as

$$\mathrm{divide}\big(\mathrm{canonical}((64, 128)),\ (16, 8)\big) \circ \mathrm{interleave}\big(F_C,\ \mathrm{canonical}((4, 16))\big),$$

equals NVIDIA's printed layout at all 8192 coordinates. A GEMM mainloop built from the operations alone, an $8 \times 32$ tile of 16-bit elements copied in 16-byte vectors by one warp into two shared-memory stages of the SW64 atom and read back by `ldmatrix`, with the writer and reader of Section 6 and the stages by repeat, is consistent: for every element, the address the copy writes is the address the `mma` operand is read from.

## 8 Related work

**CuTe** [1, 2] is the system under study; its operations and their conditions are stated in Section 1. **Shah** [3] formalizes the algebra and gives the conditions under which the composite of two layouts is again a layout. In our algebra that is decided after the fact by Theorem 1, and a composite that is not a layout is still a legal map. **Carlisle et al.** [4] model a class of CuTe layouts as morphisms of two categories, define composition, logical product and logical division on the morphisms, and characterize the layouts so obtained. **Linear layouts** [5], in Triton, represent a layout as a matrix over $\mathbb{F}_2$ acting on the bits of the hardware index, which gives generic layout-to-layout conversion; a swizzle of Definition 4 is such a matrix. **Bhaskaracharya et al.** [6] represent both CuTe layouts and linear layouts as integer set relations in ISL and implement composition, inversion and complement with its operations; the question of Section 4 can be posed there, and Theorem 1 answers it without a solver. **Axe** [7] maps logical coordinates to a multi-axis physical space through named axes and unifies tiling, sharding, replication and offsets from device meshes to threads; its replication is explicit, as broadcast is in our algebra, and its Appendix A.2.1 uses the floor expansion and first differences of Section 4 to prove a normalized representation unique. **Halide** [9], **TVM** [10] and **MLIR** [11] carry general integer expression simplifiers; Section 4 decides the layout question directly instead.

## 9 The DSL this serves

The algebra is built for a GPU DSL in which each warp is a SIMD program and warps communicate through pipes, in the sense of Unix pipes, lowered to mbarriers at compile time. The

layouts of this paper describe how the lanes of a warp access a tile.

## References


[1] NVIDIA. *CuTe: Layouts and Layout Algebra*. CUTLASS documentation, `docs.nvidia.com/cutlass`, accessed 2026.

[2] C. Cecka. *CuTe Layout Representation and Algebra*. NVIDIA Research; arXiv:2603.02298, 2026.

[3] J. Shah. *A Note on the Algebra of CuTe Layouts*. Colfax Research technical note, 2024.

[4] J. Carlisle, J. Shah, R. Stern, P. VanKoughnett. *Categorical Foundations for CuTe Layouts*. arXiv:2601.05972, 2026.

[5] K. Zhou, M. Lezcano, A. Goucher, A. Rakhmati, J. Niu, J. Lebar, P. Szczerbuk, P. Bell, P. Tillet, T. Raoux, Z. Moudallal. *Linear Layouts: Robust Code Generation of Efficient Tensor Computation Using* $\mathbb{F}_2$. ASPLOS 2026; arXiv:2505.23819.

[6] S. G. Bhaskaracharya, A. Acharya, B. Hagedorn, V. Grover. *Modeling Layout Abstractions Using Integer Set Relations*. arXiv:2511.10374, 2025.

[7] B. Hou, H. Jin, G. Wang, J. Chen, Y. Cai, L. Yang, Z. Ye, Y. Ding, R. Lai, T. Chen. *Axe: A Simple Unified Layout Abstraction for Machine Learning Compilers*. arXiv:2601.19092, 2026.

[8] NVIDIA. *Parallel Thread Execution ISA*. Matrix fragment layout tables, accessed 2026.

[9] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, S. Amarasinghe. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013.

[10] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018.

[11] C. Lattner et al. *MLIR: Scaling Compiler Infrastructure for Domain Specific Computation*. CGO 2021.