From Categories to HPC Code — a Path for Correctness?

SHONAN # 249 - Fast, furious, and yet correct tensor processing

May 17-22, 2026

J.N. Oliveira

Univ. of Minho & INESC TEC

Self presentation

Jose N. Oliveira

I am affiliated to the University of Minho and the INESC TEC R&D Institute (Portugal).

I have always been interested in **formal methods** and **program calculation**.

Also interested in **parallelism** since my PhD in the Manchester Dataflow group (1980s).

Which led me to functional programming...

Jose N. Oliveira

... and then to **relational** algebra (functions not enough!)...

... and then to **linear algebra** (LA), via a categorial approach aiming at promoting it to a kind of ´**lingua franca**' for computing.

I am currently interested in applying this approach to the design of **correct-by-construction** HPC code.

Bug in AXPY of OpenBLAS (2011)

Commit `f405b5b`

xianyi committed on Mar 18, 2011

```
Fixed the bug about Loongson3A gsLQC1 & gsSQC1 instructions in
daxpy kernel. Now daxpy is correct.
```

develop · v0.3.32 ••• v0.1alpha1

1 parent 2b8643e commit f405b5b 

Bug in AXPY of OpenBLAS (2011)

Bug fix, assembly code.

kernel/mips64/daxpy_loongson3a_simd.S +143 -92

```
@@ -228,20 +228,20 @@ USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
228                                  228
229     .L11:                        229     .L11:
230         //X & Y algin            230         //X & Y algin
231 -       gsLQC1(X_BASE,A2,A1,0*SIZE)   231 +     gsLQC1(X_BASE,A2,A1,0)
232 -       gsLQC1(X_BASE,A4,A3,2*SIZE)   232 +     gsLQC1(X_BASE,A4,A3,1)
233 -       gsLQC1(X_BASE,A6,A5,4*SIZE)   233 +     gsLQC1(X_BASE,A6,A5,2)
234 -       gsLQC1(X_BASE,A8,A7,6*SIZE)   234 +     gsLQC1(X_BASE,A8,A7,3)
235 -                                     235 +
236 -       gsLQC1(X_BASE,A10,A9,8*SIZE)  236 +     gsLQC1(X_BASE,A10,A9,4)
237 -       gsLQC1(X_BASE,A12,A11,10*SIZE) 237 +    gsLQC1(X_BASE,A12,A11,5)
238 -       gsLQC1(X_BASE,A14,A13,12*SIZE) 238 +    gsLQC1(X_BASE,A14,A13,6)
239 -       gsLQC1(X_BASE,A16,A15,14*SIZE) 239 +    gsLQC1(X_BASE,A16,A15,7)
240 -                                     240 +
241 -       gsLQC1(Y_BASE,B2,B1,0*SIZE)   241 +     gsLQC1(Y_BASE,B2,B1,0)
242 -       gsLQC1(Y_BASE,B4,B3,2*SIZE)   242 +     gsLQC1(Y_BASE,B4,B3,1)
243 -       gsLQC1(Y_BASE,B6,B5,4*SIZE)   243 +     gsLQC1(Y_BASE,B6,B5,2)
244 -       gsLQC1(Y_BASE,B8,B7,6*SIZE)   244 +     gsLQC1(Y_BASE,B8,B7,3)
245                                  245
```

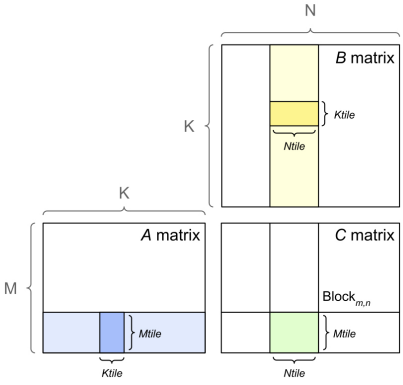
'Fast, furious, and yet correct tensor processing'

"(...) One sees the urgent need in **correctness by construction**.

Correctness has many aspects:

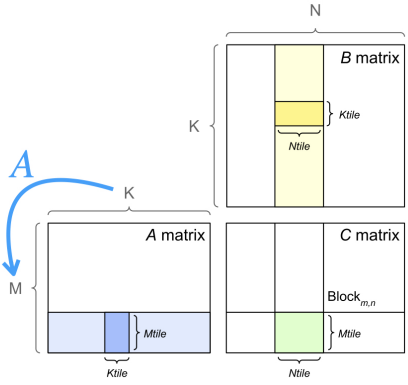
- *at the very least, the generated tensor processing code should be* **well-formed** *and* **well-typed**: *its preparation and (up)loading to the processor should assuredly be* **error-free**;
- *shape correctness: all array accesses must be surely within bounds;* **dimensions** *of tensors to add, reduce, etc. must match;*
- *(...)"*

Matrix multiplication



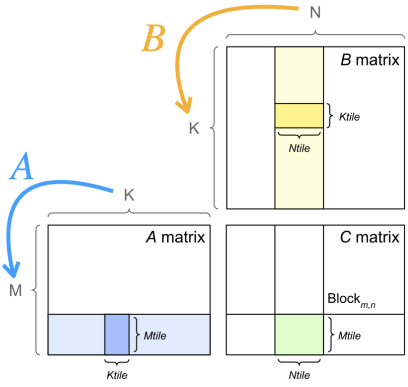
Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication



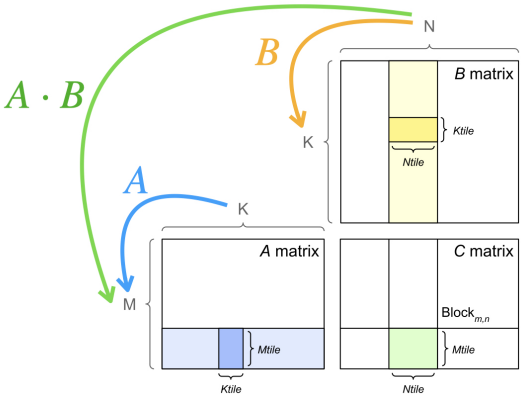
Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication



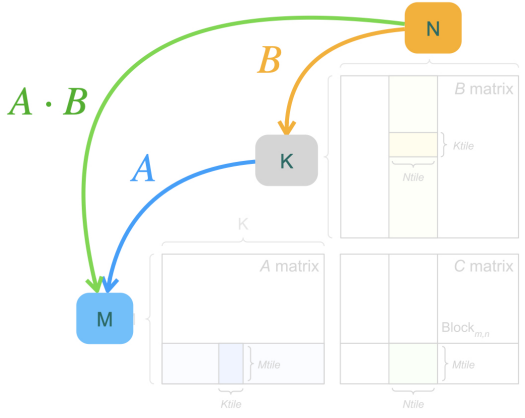
Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication



Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

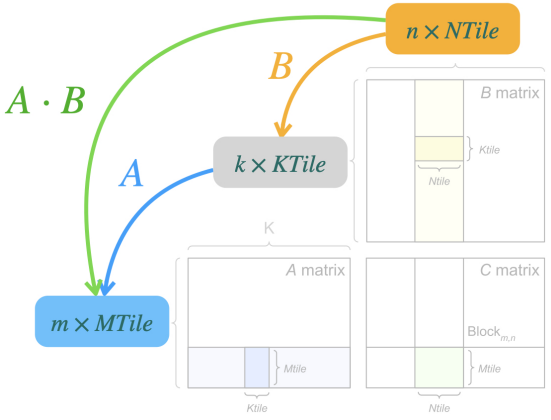
Matrix multiplication



'Matrices as *arrows*'

Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication



'Tensors as arrows' etc

Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Categorial approach to Linear Algebra (2013)

Science of Computer Programming
Volume 78, Issue 11, 1 November 2013, Pages 2160-2191

Typing linear algebra: A biproduct-oriented approach

Hugo Daniel Macedo, José Nuno Oliveira

Show more

Categorial approach to Linear Algebra (2020)

RESEARCH-ARTICLE | FREE ACCESS |

Type your matrices for great good: a Haskell library of typed matrices and applications (functional pearl)

Authors: Armando Santos, José N. Oliveira Authors Info & Claims

Haskell 2020: Proceedings of the 13th ACM SIGPLAN International Symposium on Haskell • Pages 54 - 66
https://doi.org/10.1145/3406088.3409019

Published: 09 August 2020 Publication History

Check for updates

Related Artifact: Linear Algebra of Programming Library • July 2020 • software • https://doi.org/10.1145/3410239

Feedback

Summary of the approach

Objects are matrix dimensions

- Static (dimensional) **type-checking**

The **arrows** ("morphisms") are matrices

- Diagrams!

Blocked linear algebra captured by **biproducts**.

Polymorphism and **genericity**.

Tensor transformation by generic **vectorization** operators (= matrix *'currying'*)

Index-free reasoning

Safe index-free $\leftrightarrow$ index-wise conversion

Interest in **LA** as a 'lingua franca' for computing.

Research team

Joint work with master's students

- Ana Sá Oliveira
- Edgar Araújo
- Gabriel Paiva

who are looking at OpenBLAS routines from a **reverse engineering** perspective, mapping them to high-level **typed LA** specifications.

Example shown in this talk — derivation of DAPXY from APXY by correctness preserving transformations.

'Matrices as Arrows'

Categories of matrices — dimensions as types

Objects are matrix dimensions (natural numbers) regarded as **types**. Such types denote initial segments:

$$n = \{0\,..\,n-1\} \tag{1}$$

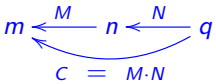
Thus

$$\begin{aligned} 0 &= \{\,\} \\ 1 &= \{0\} \\ 2 &= \{0,1\} \quad \text{and so on.} \end{aligned}$$

Morphisms $n \xleftarrow{M} m$ are **matrices** typed by such numbers (dimensions). **Identities** are (square) identity matrices, $n \xleftarrow{id} n$.

Categories of matrices — matrices as morphisms

Composition

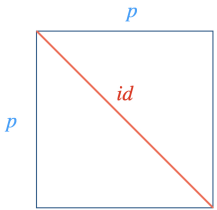


(2)

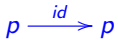
is matrix-matrix multiplication:

$$c\ (M \cdot N)\ a \quad = \quad \langle \Sigma\ b\ ::\ c\ M\ b \times b\ N\ a \rangle \tag{3}$$

NB: *Infix notation for matrix cells, e.g.* $b\ M\ a$, *instead of the more usual* $M_{b\,a}$.



The **identity**



is a **polymorphic** matrix and so $id \cdot M = M = M \cdot id$ holds.

Transposition etc

Converses exist as matrix transpositions, $a\ M^\circ\ b = b\ M\ a$.

The following laws hold: idempotence

$$(M^\circ)^\circ = M$$

and contravariance:

$$(M \cdot N)^\circ = N^\circ \cdot M^\circ$$

Vectors: column (resp. row) **vectors** are matrices of generic type $m \xleftarrow{v} 1$ (resp. $1 \xleftarrow{v} m$).

Index notation $v\,[i]$ will be used to abbreviate vector cells, eg. $(i\ v\ 0)$ written $v\,[i]$.

Categories of matrices — monoidal structures

Matrix **addition**:

$$c\ (M + N)\ a \quad = \quad c\ M\ a + c\ N\ a \tag{4}$$

Hadamard product:

$$c\ (M \times N)\ a \quad = \quad c\ M\ a \times c\ N\ a \tag{5}$$

Scalar product:

$$c\ (\alpha\ M)\ a \quad = \quad \alpha\ (c\ M\ a) \tag{6}$$

(Some underlying algebraic field assumed, typically $\mathbb{R}$ or $\mathbb{C}$.)

LA as 'lingua franca'

Unifying notations

Relations are Boolean matrices, meaning that their cells live in $\{0, 1\}$.

Boolean operations (over $\{0, 1\}$) can be defined **arithmetically**:

$$a \wedge b = ab$$
$$\overline{a} = 1 - a$$
$$a \vee b = a + b - ab$$

Moreover:

$$a \Rightarrow b = a \leqslant b$$

Thus 0 (resp. 1) is the unit of disjunction (resp. conjunction).

Unifying notations

Boolean $\{0,1\}$-matrices can be regarded as binary **predicates**.

For instance, the matrix $n \xleftarrow{(\leqslant)} m$ captures the comparison of indices within dimensions m and n.

> **NB**: *infix notation $j\ M\ i$ comes in handy here — writing e.g. $(\leqslant)_{j\ i}$ instead of $i \leqslant j$ would be rather unnatural.*

Units: $n \xleftarrow{0} m$ (resp. $n \xleftarrow{\top} m$) is the unit of $M + N$ (resp. $M \times N$), where $n \xleftarrow{0} m$ (resp. $n \xleftarrow{\top} m$) is the smallest (resp. largest) $\{0,1\}$-matrix of its type.

Row vector $1 \xleftarrow{\top} m$ is usually denoted by $!$ and is termed "*bang*".

Unifying notations

Functions (denoted by lowercase letters) are special cases of relations, i.e. $\{0,1\}$-matrices:

$$y \; [\![f]\!] \; x = 1 \quad \Leftrightarrow \quad y = f \; x \tag{7}$$

$[\![f]\!]$ denotes the matrix that represents the graph of f. Most often we abbreviate $[\![f]\!]$ to f, as e.g. in the folowing rules:

$$y(f \cdot N)x \quad = \quad \langle \Sigma \; z \; : \; y = f \; z : \; z \; N \; x \rangle \tag{8}$$

$$y(g^{\circ} \cdot M \cdot f)x \quad = \quad (g \; y) \; M \; (f \; x) \tag{9}$$

Law (9) will be very useful wrt. index manipulation.

It will be referred to as the "**nice** rule". 

Unifying notations

This **unified notation** for matrices, relations and functions includes quantifiers:

'Eindhoven notation' for all **quantifiers** *($\Theta := \Sigma$, $\forall$, $\exists$, etc):*

$$\langle \Theta\ vars\ :\ \phi :\ \psi \rangle$$

Predicate ϕ determines the **range** of the quantification; ψ is a numeric **term**.

One-point rule:

$$\langle \Sigma\ k\ :\ k = e :\ \gamma \rangle\ =\ \gamma\ [k := e] \qquad (10)$$

In the **trading rule**

$$\langle \Sigma\ x\ :\ \psi \wedge \phi :\ \gamma \rangle\ =\ \langle \Sigma\ x\ :\ \phi :\ \psi \times \gamma \rangle \qquad (11)$$

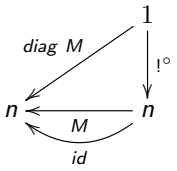
ψ is both regarded as a range predicate or a $\{0, 1\}$ term.

Example — diagonals

Let $n \xleftarrow{M} n$ be a square matrix and define its **diagonal** vector by

$$n \xleftarrow{diag\ M} 1 \ = (M \times id) \cdot !^{\circ} \quad (12)$$

Types:



That $diag\ M\ [j] = j\ M\ j$ indeed holds is calculated aside:

$$
\begin{array}{ll}
 & diag\ M\ [j] \\
= & \quad \{ \text{ definition (12) ; vector notation } \} \\
 & j\ ((M \times id) \cdot !^{\circ})\ 0 \\
= & \quad \{ \text{ Hadamard (5) ; converse ; identity; "bang" } \} \\
 & \langle \Sigma\ k\ ::\ ((j\ M\ k) \times (j = k)) \times (0\ !\ k) \rangle \\
= & \quad \{ \text{ trading rule (11) } \} \\
 & \langle \Sigma\ k\ :\ k = j :\ ((j\ M\ k)) \times (0\ !\ k) \rangle \\
= & \quad \{ \text{ one-point rule (10) ; } ! : n \to 1 \text{ is constant } \} \\
 & (j\ M\ j) \times 1 \\
= & \quad \{ \text{ trivial } \} \\
 & j\ M\ j
\end{array}
$$

Exercise — typing APXY (Kiselyov, 2024)

"(...) The precise specification – or, the reference implementation – is the following short and straightforward C code:

```
void AXPY(const int N, const FLOAT da,
          const FLOAT x[], const int inc_x, FLOAT y[], const int inc_y){
for(int i=0; i<N; i++)
  y[i*inc_y] += da * x[i*inc_x];
}
```

What are the **types** involved in this piece of code?

- Clearly, $i \in n$ and we have two vectors, $1 \xrightarrow{x} m$ and $1 \xrightarrow{y} k$, for some m and k.
- What is the relationship between n, m and k?

Exercise — typing APXY

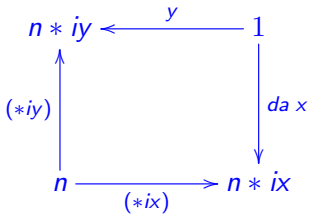
(Let us use abbreviations $iy, ix := inc\_y, inc\_x$ to save ink.)

From `y[i*inc_y]` and $i \in n$, we get $k = n * iy$.

Similarly, from `x[i*inc_x]` we get $k = n * ix$.

Scaled $da\ x$ is of the same type as x.

Let us put everything we already know in a **type diagram**:



Exercise — typing APXY

Next, we rewrite

```
y[i*inc_y] += da * x[i*inc_x];
```

into the less imperative

```
y'[i*inc_y] = y[i*inc_y] + da * x[i*inc_x];
```
(13)

where $1 \xrightarrow{y'} n * iy$ is the updated y.

Let us "reverse specify" (13).

Exercise — typing APXY

$$y'\ [i * iy] = y\ [i * iy] + da * x\ [i * ix];$$

$\Leftrightarrow$ { unfold vector notation to infix notation }

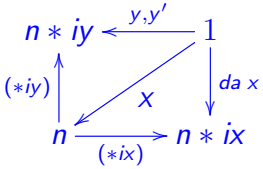
$$(i * iy)\ y'\ 0 = (i * iy)\ y\ 0 + da\ ((i * ix)\ x\ 0)$$

$\Leftrightarrow$ { scalar product (6) ; "nice rule" (9) three times }

$$i\ ((*iy)^\circ \cdot y')\ 0 = i\ ((*iy)^\circ \cdot y)\ 0 + i\ ((*ix)^\circ \cdot (da\ x))\ 0$$

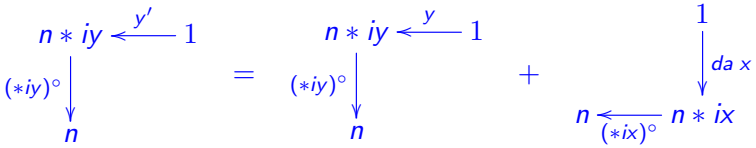
$\Leftrightarrow$ { 0 is the sole inhabitant of type 1; go index-free }

$$(*iy)^\circ \cdot y' = \underbrace{(*iy)^\circ \cdot y + (*ix)^\circ \cdot (da\ x)}_{x}$$



Exercise — typing APXY

In summary:



$$y'\ [i * iy] = y\ [i * iy] + da * x\ [i * ix];$$

Biproducts

Biproducts in general

(In an Abelian category) a **biproduct** diagram for the objects m, n is a diagram of shape

$$m \underset{i_1}{\overset{\pi_1}{\leftrightarrows}} r \underset{i_2}{\overset{\pi_2}{\leftrightarrows}} n$$

whose arrows π_1, π_2, ι_1, ι_2 satisfy the identities which follow:

$$\pi_1 \cdot \iota_1 = id_m \tag{14}$$

$$\pi_2 \cdot \iota_2 = id_n \tag{15}$$

$$\iota_1 \cdot \pi_1 + \iota_2 \cdot \pi_2 = id_r \tag{16}$$

Two **orthogonality** properties follow:

$$\pi_1 \cdot \iota_2 = 0 \quad , \quad \pi_2 \cdot \iota_1 = 0 \tag{17}$$

"Standard" LA biproduct

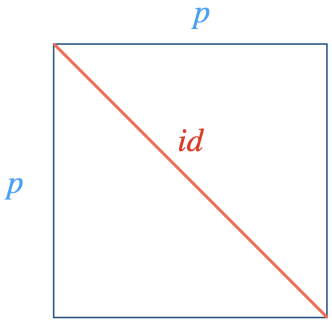
There are **many** solutions to the biproduct equations (14-16) in a category of matrices.

Each biproduct captures a particular way of doing "**blockwise** *operations*".

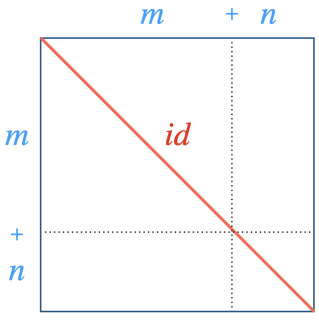
Let us start from the "standard" biproduct, immediate to obtain from identity matrices.

We do that in the following slides.

'Standard' biproduct — identity

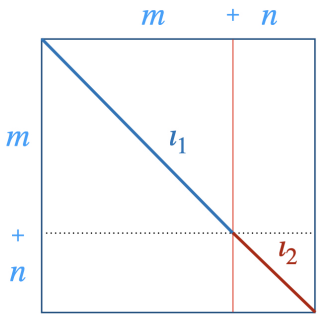


'Standard' biproduct — identity



'Standard' biproduct — injections

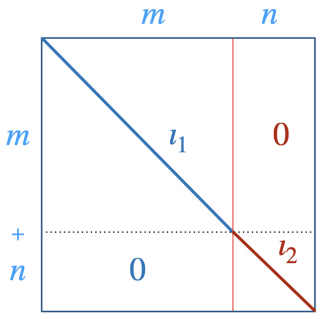
$\iota_1 : m \to m + n$



$\iota_2 : m + n \leftarrow n$

'Standard' biproduct — injections

$$\iota_1 : m \to m + n$$
$$\iota_1 = \left[\frac{id_m}{0}\right]$$

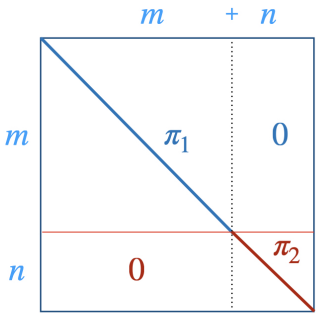


$$\iota_2 : m + n \leftarrow n$$
$$\iota_2 = \left[\frac{0}{id_n}\right]$$

'Standard' biproduct — projections

$\pi_1 : m \leftarrow m + n$

$\pi_1 = \iota_1^\circ = [\, id_m \mid 0 \,]$



$\iota_2 : m + n \rightarrow n$

$\pi_2 = \iota_2^\circ = [\, 0 \mid id_n \,]$

Standard biproduct

It is not hard work to show that

$$m \underset{\iota_1=\left[\begin{array}{c} id_m \\ \hline 0 \end{array}\right]}{\overset{\pi_1=\left[\begin{array}{c|c} id_m & 0 \end{array}\right]}{\leftrightarrows}} m+n \underset{\iota_2=\left[\begin{array}{c} 0 \\ \hline id_n \end{array}\right]}{\overset{\pi_2=\left[\begin{array}{c|c} 0 & id_n \end{array}\right]}{\leftrightarrows}} n$$

indeed form a **biproduct**.

Despite its simplicity, this biproduct is significant: not only

> *it explains the* **"block operations"** *of LA*

but also does so by exposing many **properties** of such operators.

Standard biproduct

It is not hard work to show that

$$m \underset{\iota_1 = \left[\frac{id_m}{0}\right]}{\overset{\pi_1 = \left[\, id_m \mid 0 \,\right]}{\leftrightarrows}} m+n \underset{\iota_2 = \left[\frac{0}{id_n}\right]}{\overset{\pi_2 = \left[\, 0 \mid id_n \,\right]}{\leftrightarrows}} n$$

indeed form a **biproduct**.

Despite its simplicity, this biproduct is significant: not only

---

it explains the **"block operations"** *of LA*

---

but also does so by exposing many **properties** of such operators.

Standard biproduct — pointwise

Injections pointwise:

$$\begin{cases} \iota_1 : m + n \leftarrow m \\ \iota_1\ i = i \end{cases} \tag{18}$$

$$\begin{cases} \iota_2 : m + n \leftarrow n \\ \iota_2\ j = m + j \end{cases} \tag{19}$$

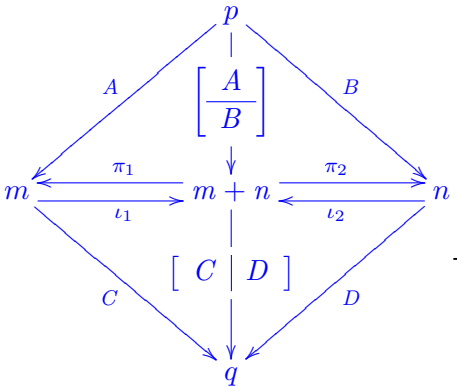
Projections pointwise:[1]

$$\begin{cases} \pi_1 : m \leftarrow m + n \\ i\ \pi_1\ k = (k = \iota_1\ i) \end{cases} \tag{20}$$

$$\begin{cases} \pi_2 : n \leftarrow m + n \\ j\ \pi_2\ k = (k = \iota_2\ j) \end{cases} \tag{21}$$

[1] Recall that projections are converses of injections, e.g. $\pi_1 = \iota_1^\circ$.

Blocked LA — definitions



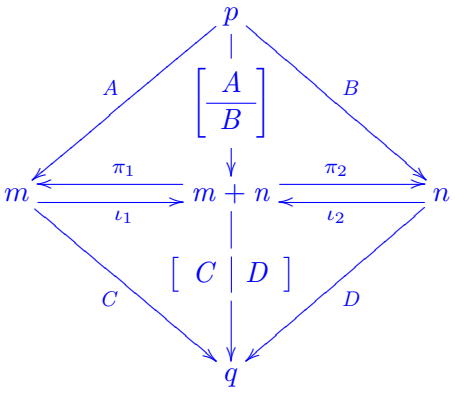
$$\left[\frac{A}{B}\right] = i_1 \cdot A + i_2 \cdot B \quad (22)$$

$$[\ C \mid D\ ] = C \cdot \pi_1 + D \cdot \pi_2 \quad (23)$$

Then (**divide-and-conquer**):

$$[\ C \mid D\ ] \cdot \left[\frac{A}{B}\right] = C \cdot A + D \cdot B \quad (24)$$

Blocked LA — universal properties



$$X = \left[\frac{A}{B}\right] \Leftrightarrow \begin{cases} \pi_1 \cdot X = A \\ \pi_2 \cdot X = B \end{cases} \quad (25)$$

$$X = [\ C \mid D\ ] \Leftrightarrow \begin{cases} X \cdot i_1 = C \\ X \cdot i_2 = D \end{cases} \quad (26)$$

Blocked LA — derived properties

- Fusion:

$$C \cdot \left[\begin{array}{c|c} A & B \end{array}\right] = \left[\begin{array}{c|c} C \cdot A & C \cdot B \end{array}\right] \tag{27}$$

$$\left[\begin{array}{c} \frac{A}{B} \end{array}\right] \cdot C = \left[\begin{array}{c} \frac{A \cdot C}{B \cdot C} \end{array}\right] \tag{28}$$

- Structural equality:

$$\left[\begin{array}{c|c} A & B \end{array}\right] = \left[\begin{array}{c|c} C & D \end{array}\right] \Leftrightarrow A = C \wedge B = D \tag{29}$$

$$\left[\begin{array}{c} \frac{A}{B} \end{array}\right] = \left[\begin{array}{c} \frac{C}{D} \end{array}\right] \Leftrightarrow A = C \wedge B = D \tag{30}$$

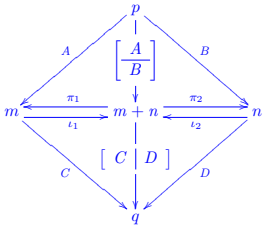
- exchange law:

$$\left[\begin{array}{c} \frac{\left[\begin{array}{c|c} A & B \end{array}\right]}{\left[\begin{array}{c|c} C & D \end{array}\right]} \end{array}\right] = \left[\begin{array}{c|c} \left[\begin{array}{c} \frac{A}{C} \end{array}\right] & \left[\begin{array}{c} \frac{B}{D} \end{array}\right] \end{array}\right] = \left[\begin{array}{c} \frac{\begin{array}{c|c} A & B \end{array}}{\begin{array}{c|c} C & D \end{array}} \end{array}\right] \tag{31}$$

Blocked LA — MMM modes

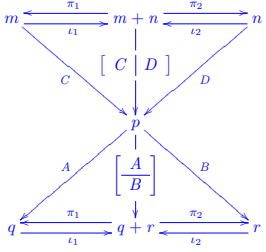
"Inner"-mode:

$$\begin{bmatrix} C \mid D \end{bmatrix} \cdot \left[\frac{A}{B}\right] = C \cdot A + D \cdot B$$

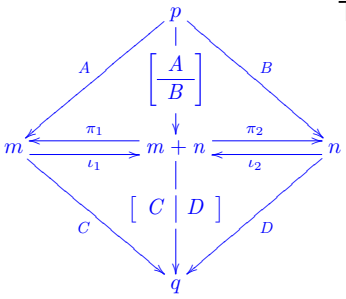


"Outer"-mode:

$$\left[\frac{A}{B}\right] \cdot \begin{bmatrix} C \mid D \end{bmatrix} = \left[\begin{array}{c|c} A \cdot C & A \cdot D \\ \hline B \cdot C & B \cdot D \end{array}\right]$$



Block parallelism



Thanks to the universal properties:

$$\{M := \left[\frac{A}{B}\right]\} \quad = \quad \{\pi_1 \cdot M := A\} \parallel \{\pi_2 \cdot M := B\}$$

$$\{M := [\ C \mid D\ ]\} \quad = \quad \{M \cdot \iota_1 := C\} \parallel \{M \cdot \iota_2 := D\}$$

Parallelism ensured by the **orthogonality** (ie. "separation") properties (17) of the biproduct.

Biproduct-oriented Haskell library

```
data Matrix e c r where
  One  :: e -> Matrix e () ()
  Join :: Matrix e a r -> Matrix e b r
       -> Matrix e (Either a b) r
  Fork :: Matrix e c a -> Matrix e c b
       -> Matrix e c (Either a b)
```

Haskell '20, August 27, 2020

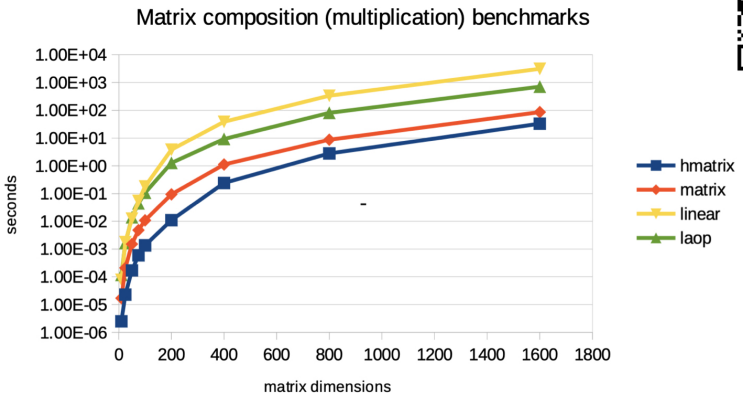
divide & conquer

fork-fusion

join-fusion

```
comp :: Num e => Matrix e cr rows
     -> Matrix e cols cr -> Matrix e cols rows
comp (One a) (One b)       = One (a * b)
comp (Join a b) (Fork c d) = comp a c + comp b d
comp (Fork a b) c          =
    Fork (comp a c) (comp b c)
comp c (Join a b)          =
    Join (comp c a) (comp c b)
```

Biproduct-oriented Haskell library



Padding

Any matrix

$$m \xrightarrow{M} n$$

can be extended to a **larger** type

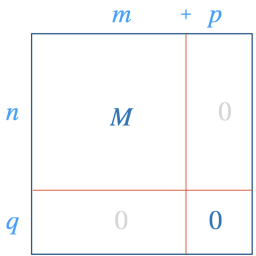
$$m' \xrightarrow{M'} n'$$

(where $m' > m$ and $n' > n$) by defining

$$M' = M \oplus 0$$

which uses the **direct sum** (derived) operator:

$$M \oplus N = \left[\; \iota_1 \cdot M \;\middle|\; \iota_2 \cdot N \; \right] \tag{32}$$



Building biproducts out of biproducts

The **permutation biproduct**:

Given biproduct $m \underset{\iota_1}{\overset{\pi_1}{\leftrightarrows}} r \underset{\iota_2}{\overset{\pi_2}{\leftrightarrows}} n$ and index-permutations $\alpha : n \to n$ and $\beta : m \to m$, then

$$m \underset{\iota_1 \cdot \beta}{\overset{\beta^\circ \cdot \pi_1}{\leftrightarrows}} r \underset{\iota_2 \cdot \alpha}{\overset{\alpha^\circ \cdot \pi_2}{\leftrightarrows}} n$$

is a biproduct.

Building biproducts out of biproducts

α-**biproduct** — generalization

$$\pi'_1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad , \quad \pi'_2 = \begin{bmatrix} -\alpha & 1 \end{bmatrix}$$

$$i'_1 = \begin{bmatrix} 1 \\ \alpha \end{bmatrix} \quad , \quad i'_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

of the standard biproduct, parametric on α.

Running the definitions, the block operators (22,23) of this biproduct are:

$$\left[\begin{array}{c|c} A & B \end{array}\right]_\alpha = \left[\begin{array}{c|c} A - \alpha\, B & B \end{array}\right] \tag{33}$$

$$\left[\frac{C}{D}\right]_\alpha = \left[\frac{C}{\alpha\, C + D}\right] \tag{34}$$

α biproduct

Just as an exercise, let us check *divide-and-conquer* (35) for this biproduct:

$$\left[\begin{array}{c|c} A & B \end{array}\right] \cdot \left[\frac{C}{D}\right] \quad = \quad A \cdot C + B \cdot D$$

We have

$$\left[\begin{array}{c|c} A & B \end{array}\right]_\alpha \cdot \left[\frac{C}{D}\right]_\alpha \quad = \quad (A - \alpha\, B) \cdot C + B \cdot (\alpha\, C + D) = A \cdot C + 0 + B \cdot D$$

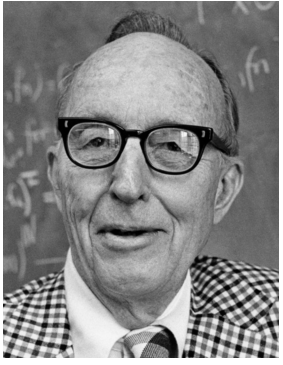
It does multiply the matrices in the right way, as expected.

(As any other biproduct does!)

Prophetic statement (1978)

The famous category theorist Sounders Mac Lane laconically wrote (1978):

> *"In other words, the [biproduct] equations contain the familiar calculus of matrices."*



S. Mac Lane (1909-2005)

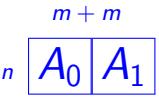
Vectorization

Product types

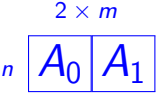
A matrix of type

$$m + m \to n$$

is by construction made of two blocks of type $m \to n$ each, put aside of each other.



Since $m + m = 2 \times m$, one might write the type $2 \times m \to n$ instead, where $2 = \{0, 1\}$ indicates each block's index.

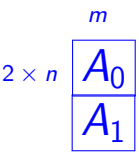


Product types

Clearly, type $m + m \to n$ — i.e.

$$2 \times m \to n$$

— is isomorphic to $m \to 2 \times n$: it corresponds to stacking such blocks **vertically** instead of glueing them **horizontally**.



Such is the essence of **vectorization**, *a feature of matrix categories that is very relevant for our purposes.*

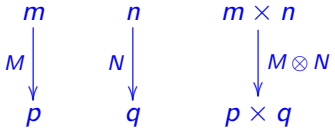
A question, to begin with: as a **type**, what does $m \times n$ mean?

Product types

Product types:

$$n \times m = \{(b, a) \mid b \in n, a \in m\} \tag{35}$$

Kronecker product:

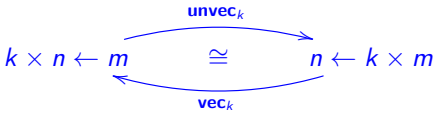


$$(y, z)\ (M \otimes N)\ (b, a) \;=\; (y\ M\ b) \times (z\ N\ a) \tag{36}$$

NB: (36) generalizes relational product.

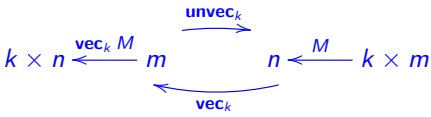
(Generic) Vectorization

Isomorphism ('adjunction'):



(37)

Pointwise meaning:



(38)

$$(c, b)\ (\mathbf{vec}_k\ M)\ a \quad = \quad b\ M\ (c, a)$$

We will refer to k as the **thinning factor** of the vectorization.

Vectorization in practice

$\mathbf{vec}_{Color}$

$Model \longleftarrow Color \times Year$						
	$Blue$		$Green$		Red	
	1990	1991	1990	1991	1990	1991
$Chevy$	87	0	0	0	5	0
$Ford$	99	7	64	0	0	8

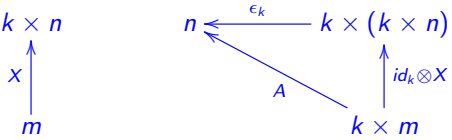
$=$

$Color \times Model \longleftarrow Year$			
		1990	1991
$Blue$	$Chevy$	87	0
	$Ford$	99	7
$Green$	$Chevy$	0	0
	$Ford$	64	0
Red	$Chevy$	5	0
	$Ford$	0	8

Ref: *The data cube as a typed linear algebra operator*, DBPL'17

Vectorization — universal property

Matrix "currying",



(39)

$$X = \mathbf{vec}_k\, A \qquad \Leftrightarrow \qquad A = \epsilon_k \cdot (id_k \otimes X)$$

where ϵ_k is the relation:

$$b'\ \epsilon_k\ (c', (c, b)) \ = \ (c' = c) \times (b' = b) \tag{40}$$

(Put in another way, $\mathbf{vec}_k\, \epsilon_k = id$ — see slide 88 in the Annex.)

Vectorization — properties for free by adjunction (37)

reflection:

$$\mathbf{vec}_k\, \epsilon = id \tag{41}$$

that is,

$$\epsilon = \mathbf{unvec}_k\, id \tag{42}$$

cancellation:

$$\epsilon \cdot (id \otimes \mathbf{vec}_k\, A) = A \tag{43}$$

Vectorization — properties for free by adjunction (37)

fusion:

$$(\mathbf{vec}_k\, B) \cdot C = \mathbf{vec}_k\, (B \cdot (id \otimes C)) \tag{44}$$

absorption:

$$(id \otimes B) \cdot (\mathbf{vec}_k\, C) = \mathbf{vec}_k\, (B \cdot C) \tag{45}$$

naturality:

$$A \cdot \epsilon = \epsilon \cdot (id \otimes (id \otimes A)) \tag{46}$$

functor:

$$id \otimes A = \mathbf{vec}_k\, (A \cdot \epsilon) \tag{47}$$

Vectorization — more properties

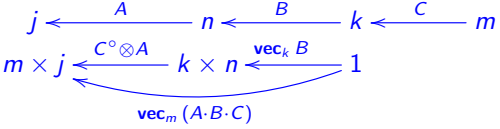
Finally, the equality

$$\mathbf{vec}_m\,(C \cdot B) \quad = \quad (B^\circ \otimes id_n) \cdot \mathbf{vec}_k\, C \tag{48}$$

leads to the well-known **Roth's relationship**,

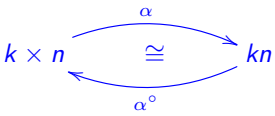
$$\mathbf{vec}_m\,(A \cdot B \cdot C) \quad = \quad (C^\circ \otimes A) \cdot \mathbf{vec}_k\, B \tag{49}$$

whose principal type is:



Strict interpretation of product types

"Go strict" isomorphism:



$$\begin{cases} \alpha : k \times n \to k\ n \\ \alpha\ (c, b) = cn + b \end{cases} \tag{50}$$

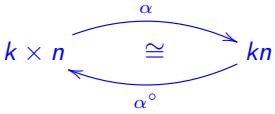
Range checking:

$$\alpha\ (0, 0) = 0$$
$$\alpha\ (k-1, n-1) = (k-1)\ n + n - 1 = kn - 1.$$

Terminology: c in (50) is the **stride** and b is the **offset**.

Strict interpretation of product types

Converse (non-strict) interpretation



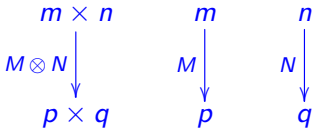
$$\begin{cases} \alpha^\circ : kn \to k \times n \\ \alpha^\circ\ (cn + b) = (c, b) \end{cases} \tag{51}$$

that is,

$$\alpha^\circ\ x = (x \div n, x\ mod\ n)$$

$$\begin{array}{c|c} x & n \\ \hline b & c \end{array}$$

Strict interpretation of the Kronecker product



$$(y, z)\,(M \otimes N)\,(b, a) \;=\; (y\ M\ b) \times (z\ N\ a)$$

$$(yq + z)\,(M \otimes N)\,(bn + a) \;=\; (y\ M\ b) \times (z\ N\ a)$$

NB: note the "type-driven" strides and offsets.

Vectorization (pointwise summary)

$$
\begin{array}{rcll}
k \times n \xleftarrow{\mathbf{vec}_k\ M} m & \cong & n \xleftarrow{M} k \times m & \\
(a, c)\ (\mathbf{vec}_k\ M)\ b & = & c\ M\ (a, b) & \\
(a\ n + c)\ (\mathbf{vec}_k\ M)\ b & = & c\ M\ (a\ m + b) & \text{(strict interpretation)}
\end{array}
$$

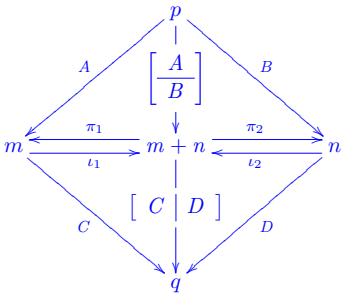
In case $m = 1 = \{0\}$, b is bound to be 0 — column vector. Then:

$$
\begin{array}{rcll}
k \times n \xleftarrow{\mathbf{vec}_k\ M} 1 & \cong & n \xleftarrow{M} k & \\
(a, c)\ (\mathbf{vec}_k\ M)\ 0 & = & c\ M\ (a, 0) & \\
(a\ n + c)\ (\mathbf{vec}_k\ M)\ 0 & = & c\ M\ a & \text{(strict interpretation)}
\end{array}
\tag{52}
$$

From diagrams to code

Block (vectorized) parallelism

Recall

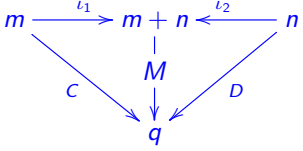


$$\{M := \left[\frac{A}{B}\right]\} \quad = \quad \{\pi_1 \cdot M := A\} \parallel \{\pi_2 \cdot M := B\}$$

$$\{M := [\ C \mid D\ ]\} \quad = \quad \{M \cdot \iota_1 := C\} \parallel \{M \cdot \iota_2 := D\}$$

What is the outcome of vectorizing this?

From diagram to code



$$\{ M \cdot \iota_1 := C \}$$

$$= \quad \{ \text{ vectorizing } \}$$

$$\{ \overline{M \cdot \iota_1}^{\,m} := \overline{C}^{\,m} \}$$

$$= \quad \{ \text{ Roth (48) for functions } \}$$

$$\{ (\iota_1^\circ \otimes id) \cdot \overline{M}^{\,n+m} := \overline{C}^{\,m} \}$$

$$= \quad \{ \text{ go pointwise } \}$$

$$\textbf{for } (i \in m, j \in q) \textbf{ do } \{ (i,j)\ ((\iota_1^\circ \otimes id) \cdot \overline{M}^{\,m+n}\ 0) := (i,j)\ \overline{C}^{\,m}\ 0 \}$$

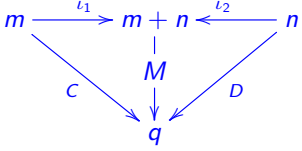
$$= \quad \{ \text{ nice rule ; go strict } \}$$

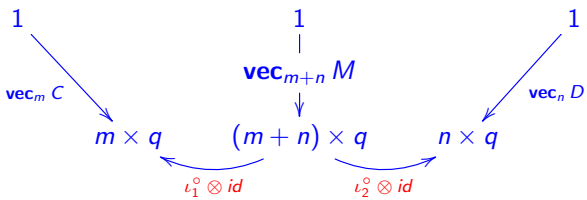
$$\textbf{for } (i \in m, j \in q) \textbf{ do } \{ (i\ q + j)\ \overline{M}^{\,m+n}\ 0 := (i\ q + j)\ \overline{C}^{\,m}\ 0 \}$$

$$= \quad \{ \text{ vector notation ; drop subscripts } \}$$

$$\textbf{for } (i \in m, j \in q) \textbf{ do } \{ \overline{M}\ [i\ q + j] := \overline{C}\ [i\ q + j] \}$$

From diagram to code





$\{ M \cdot \iota_1 := C \}$

$=$ { vectorizing }

$\{ \overline{M \cdot \iota_1}^{\,m} := \overline{C}^{\,m} \}$

$=$ { Roth (48) for functions }

$\{ (\iota_1^\circ \otimes id) \cdot \overline{M}^{\,n+m} := \overline{C}^{\,m} \}$

$=$ { go pointwise }

for $(i \in m, j \in q)$ do $\{ (i,j)\ ((\iota_1^\circ \otimes id) \cdot \overline{M}^{\,m+n}\ 0) := (i,j)\ \overline{C}^{\,m}\ 0 \}$

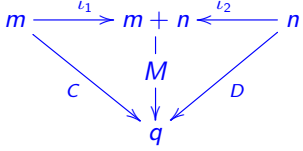
$=$ { nice rule ; go strict }

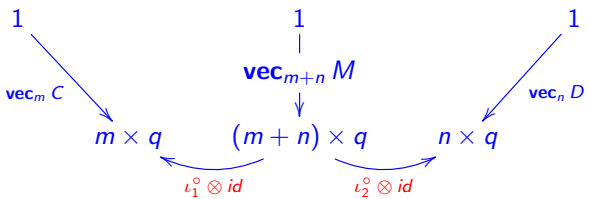
for $(i \in m, j \in q)$ do $\{ (i\ q + j)\ \overline{M}^{\,m+n}\ 0 := (i\ q + j)\ \overline{C}^{\,m}\ 0 \}$

$=$ { vector notation ; drop subscripts }

for $(i \in m, j \in q)$ do $\{ \overline{M}\ [i\ q + j] := \overline{C}\ [i\ q + j] \}$

From diagram to code

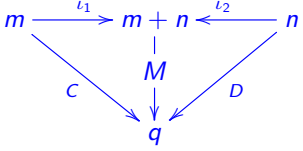


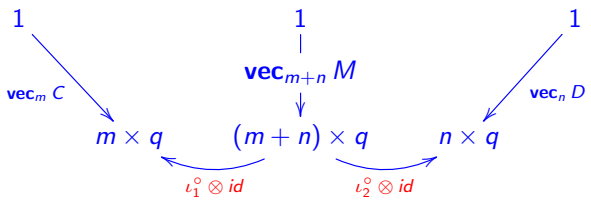


$$
\begin{aligned}
& \{\, M \cdot \iota_1 := C \,\} \\
= \quad & \{ \text{ vectorizing } \} \\
& \{\, \overline{M \cdot \iota_1}^{\,m} := \overline{C}^{\,m} \,\} \\
= \quad & \{ \text{ Roth (48) for functions } \} \\
& \{\, (\iota_1^{\circ} \otimes id) \cdot \overline{M}^{\,n+m} := \overline{C}^{\,m} \,\}
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \{ \text{ go pointwise } \} \\
& \textbf{for } (i \in m, j \in q) \textbf{ do } \{\, (i,j)\ ((\iota_1^{\circ} \otimes id) \cdot \overline{M}^{\,n+m}\ 0) := (i,j)\ \overline{C}^{\,m}\ 0 \,\} \\
= \quad & \{ \text{ nice rule ; go strict } \} \\
& \textbf{for } (i \in m, j \in q) \textbf{ do } \{\, (i\ q + j)\ \overline{M}^{\,n+m}\ 0 := (i\ q + j)\ \overline{C}^{\,m}\ 0 \,\} \\
= \quad & \{ \text{ vector notation ; drop subscripts } \} \\
& \textbf{for } (i \in m, j \in q) \textbf{ do } \{\, \overline{M}\ [i\ q + j] := \overline{C}\ [i\ q + j] \,\}
\end{aligned}
$$

From diagram to code

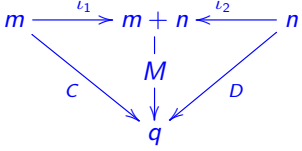


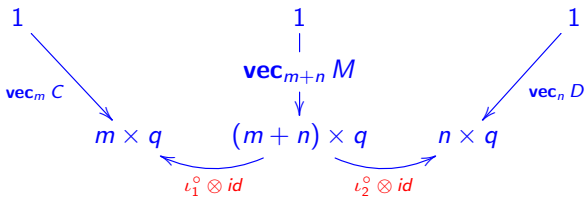


$$
\begin{aligned}
& \{\, M \cdot \iota_1 := C \,\} \\
= \quad & \{ \text{ vectorizing } \} \\
& \{\, \overline{M \cdot \iota_1}^{\,m} := \overline{C}^{\,m} \,\} \\
= \quad & \{ \text{ Roth (48) for functions } \} \\
& \{\, (\iota_1^{\circ} \otimes id) \cdot \overline{M}^{\,n+m} := \overline{C}^{\,m} \,\}
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \{ \text{ go pointwise } \} \\
& \textbf{for } (i \in m, j \in q) \textbf{ do } \{\, (i,j)\ ((\iota_1^{\circ} \otimes id) \cdot \overline{M}^{\,n+m}\ 0) := (i,j)\ \overline{C}^{\,m}\ 0 \,\} \\
= \quad & \{ \text{ nice rule ; go strict } \} \\
& \textbf{for } (i \in m, j \in q) \textbf{ do } \{\, (i\ q + j)\ \overline{M}^{\,n+m}\ 0 := (i\ q + j)\ \overline{C}^{\,m}\ 0 \,\} \\
= \quad & \{ \text{ vector notation ; drop subscripts } \} \\
& \textbf{for } (i \in m, j \in q) \textbf{ do } \{\, \overline{M}\ [i\ q + j] := \overline{C}\ [i\ q + j] \,\}
\end{aligned}
$$

From diagram to code





$\{ M \cdot \iota_2 := D \}$

$=$ { vectorizing }

$\{ \overline{M \cdot \iota_2}^{n} := \overline{D}^{n} \}$

$=$ { Roth (48) for functions }

$\{ (\iota_2^{\circ} \otimes id) \cdot \overline{M}^{n+m} := \overline{D}^{n} \}$

$=$ { go pointwise }

for $(k \in n, j \in q)$ do $\{ ((\iota_2^{\circ} \otimes id) \cdot \overline{M}^{n+m})\ 0) := (k, j)\ \overline{D}^{n}\ 0 \}$

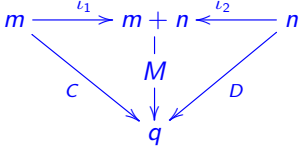
$=$ { nice rule ; $\iota_2\ k = m + k$; go strict }

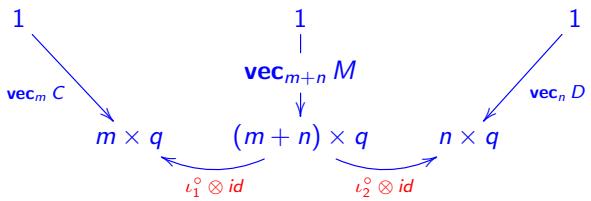
for $(k \in n, j \in q)$ do $\{ ((m + k)\ q + j)\ \overline{M}^{m+n}\ 0 := (k\ q + j)\ \overline{D}^{n}\ 0 \}$

$=$ { vector notation ; drop subscripts }

for $(k \in n, j \in q)$ do $\{ \overline{M}\ [(m + k)\ q + j] := \overline{D}\ [k\ q + j] \}$

From diagram to code





$\{ M \cdot \iota_2 := D \}$

$=$ { vectorizing }

$\{ \overline{M \cdot \iota_2}^{\,n} := \overline{D}^{\,n} \}$

$=$ { Roth (48) for functions }

$\{ (\iota_2^\circ \otimes id) \cdot \overline{M}^{\,n+m} := \overline{D}^{\,n} \}$

$=$ { go pointwise }

for $(k \in n, j \in q)$ **do** $\{ (k,j)\ ((\iota_2^\circ \otimes id) \cdot \overline{M}^{\,n+m}\ 0) := (k,j)\ \overline{D}^{\,n}\ 0 \}$

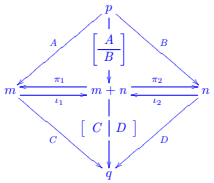
$=$ { nice rule ; $\iota_2\ k = m + k$; go strict }

for $(k \in n, j \in q)$ **do** $\{ ((m+k)\ q + j)\ \overline{M}^{\,m+n}\ 0 := (k\ q + j)\ \overline{D}^{\,n}\ 0 \}$

$=$ { vector notation ; drop subscripts }

for $(k \in n, j \in q)$ **do** $\{ \overline{M}\ [(m+k)\ q + j] := \overline{D}\ [k\ q + j] \}$

Block (vectorized) parallelism

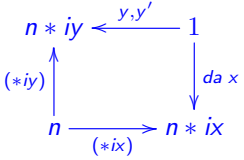


$$\{ M := \left[\begin{array}{c|c} C & D \end{array}\right] \} \quad = \quad \left\{ \begin{array}{l} \textbf{for } (i \in m, j \in q) \textbf{ do } \{ \overline{M}\ [i\ q + j] := \overline{C}\ [i\ q + j] \} \\ \| \\ \textbf{for } (k \in n, j \in q) \textbf{ do } \{ \overline{M}\ [(m + k)\ q + j] := \overline{D}\ [k\ q + j] \} \end{array} \right.$$

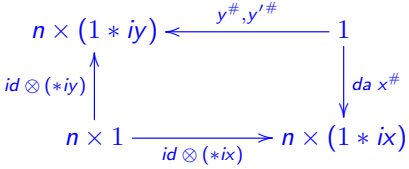
Similarly for $\{ M := \left[\frac{A}{B}\right] \}$ and so on.

Transforming APXY into DAPXY

Recall the type diagram of APXY,



the same as[2]

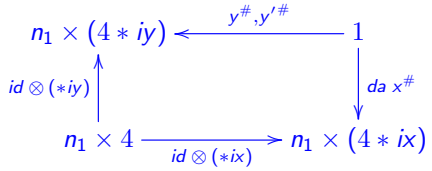


since $1 = \{0\}$.

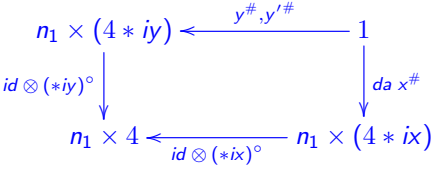
[2]NB: $y^{\#}$ abbreviates $\alpha^{\circ} \cdot y$ etc, where α makes the strict index conversion explicit.

Transforming APXY into DAPXY

Suggested by inspecting the DAPXY code, suppose $n = n_1 \times 4$ (n unrolled to n_1 "4-blocks"):



That is, reversing the stride-arrows, as before:



Back to the properties of vectorization

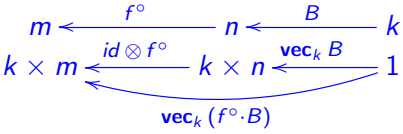
Recall absorption (45),

$$(id \otimes A) \cdot (\mathbf{vec}_k\, B) = \mathbf{vec}_k\, (A \cdot B)$$

and instantiate it for functions as follows

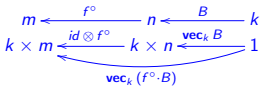
$$(id \otimes f^\circ) \cdot (\mathbf{vec}_k\, B) = \mathbf{vec}_k\, (f^\circ \cdot B) \tag{53}$$

cf:

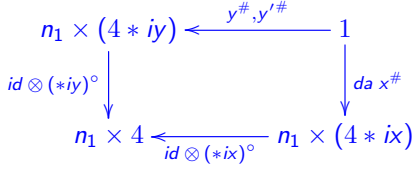


Transforming APXY into DAPXY

Let $\begin{cases} y^{\#} = \mathbf{vec}_{n_1}\ Y \\ y'^{\#} = \mathbf{vec}_{n_1}\ Y' \\ da\ x^{\#} = \mathbf{vec}_{n_1}\ X \end{cases}$



Then, by (53):

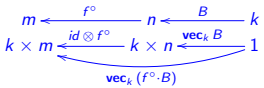


$\mapsto$

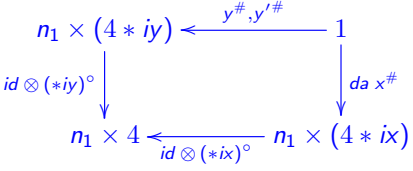
$$(*iy)^{\circ} \cdot Y' = (*iy)^{\circ} \cdot Y + (*ix)^{\circ} \cdot X \quad (54)$$

Transforming APXY into DAPXY

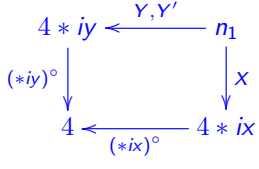
Let $\begin{cases} y^{\#} = \mathbf{vec}_{n_1}\, Y \\ y'^{\#} = \mathbf{vec}_{n_1}\, Y' \\ da\; x^{\#} = \mathbf{vec}_{n_1}\, X \end{cases}$



Then, by (53):



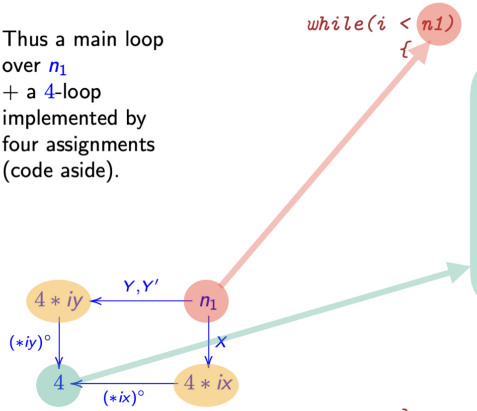
$\mapsto$



$$(*iy)^{\circ} \cdot Y' = (*iy)^{\circ} \cdot Y + (*ix)^{\circ} \cdot X \quad (54)$$

Transforming APXY into DAPXY

Thus a main loop over n_1 + a 4-loop implemented by four assignments (code aside).



(More details in the appendix.)

Going generic

What if $n \neq 4 * n_1$?

In general, we will have $n = k * n_1 + r$.

By universal property (25), any $k * n_1 + r \xleftarrow{A} m$ is uniquely divided into $k * n_1 \xleftarrow{\pi_1 \cdot A} m$ and $r \xleftarrow{\pi_2 \cdot A} m$.

We've done $\pi_1 \cdot A$.

The remainder — the "tail" $\pi_2 \cdot A$ — can be calculated in the same way.

NB: unlike in the DAPXY code, it could actually run in parallel with the first component $\pi_1 \cdot A$.

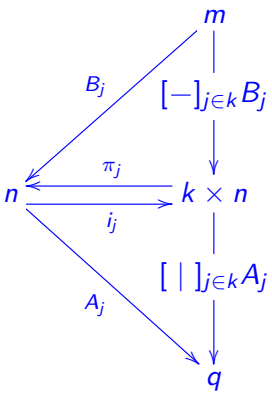
Static type checking 

kernel/mips64/daxpy_loongson3a_simd.S +143 -92

```
@@ -228,20 +228,20 @@ USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
228                                  228
229     .L11:                        229     .L11:
230         //X & Y algin            230         //X & Y algin
231 -       gsLQC1(X_BASE,A2,A1,0*SIZE)   231 +     gsLQC1(X_BASE,A2,A1,0)
232 -       gsLQC1(X_BASE,A4,A3,2*SIZE)   232 +     gsLQC1(X_BASE,A4,A3,1)
233 -       gsLQC1(X_BASE,A6,A5,4*SIZE)   233 +     gsLQC1(X_BASE,A6,A5,2)
234 -       gsLQC1(X_BASE,A8,A7,6*SIZE)   234 +     gsLQC1(X_BASE,A8,A7,3)
235 -                                     235 +
236 -       gsLQC1(X_BASE,A10,A9,8*SIZE)  236 +     gsLQC1(X_BASE,A10,A9,4)
237 -       gsLQC1(X_BASE,A12,A11,10*SIZE) 237 +    gsLQC1(X_BASE,A12,A11,5)
238 -       gsLQC1(X_BASE,A14,A13,12*SIZE) 238 +    gsLQC1(X_BASE,A14,A13,6)
239 -       gsLQC1(X_BASE,A16,A15,14*SIZE) 239 +    gsLQC1(X_BASE,A16,A15,7)
240 -                                     240 +
241 -       gsLQC1(Y_BASE,B2,B1,0*SIZE)   241 +     gsLQC1(Y_BASE,B2,B1,0)
242 -       gsLQC1(Y_BASE,B4,B3,2*SIZE)   242 +     gsLQC1(Y_BASE,B4,B3,1)
243 -       gsLQC1(Y_BASE,B6,B5,4*SIZE)   243 +     gsLQC1(Y_BASE,B6,B5,2)
244 -       gsLQC1(Y_BASE,B8,B7,6*SIZE)   244 +     gsLQC1(Y_BASE,B8,B7,3)
245                                  245
```

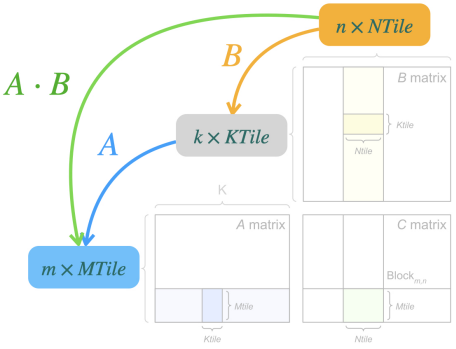
Tiling (products as finitary sums)

Generalizing to **finitary biproducts**:



$$\begin{cases} k \times n = \langle \Sigma\ j\ :\ j \in k :\ n \rangle \\ i_j\ a = (j, a) \\ \pi_j = i_j^{\circ} \end{cases}$$

Tiled MMM



Using the finitary versions of laws (27,28,35) etc, we get:

$$C = A \cdot B$$

$$\Leftrightarrow \quad \{ \text{ for all } j \in m,\ l \in k \text{ and } i \in n \ \}$$

$$\pi_j \cdot C \cdot i_l = (\pi_j \cdot A \cdot i_l) \cdot (\pi_l \cdot B \cdot i_k)$$

(**NB**: $\pi_j \cdot X \cdot \iota_k$ is the (j,k)-th **tile** of X.)

$$MTile \xleftarrow{\pi_j \cdot A \cdot \iota_l} KTile \xleftarrow{\pi_l \cdot B \cdot \iota_i} NTile$$

$$MTile \xleftarrow{\pi_j \cdot C \cdot \iota_i} NTile$$

Questions and Challenges

- Scalability?
- Practical viability of generating (efficient + type-safe + correct-by-construction) **HPC code** from such high-level **specifications**?
- Are there any **biproducts** out there that could unveil new 'smart' **tensor** processing strategies?
- Can **ML** help in this regard?

Annex

Follow-up of (54)

Go index-wise via the 'nice' rule (9):

```
for (i in n1, j in 4) do { (j*iy) Y' i =  (j*iy) Y i + (j*ix) (da X) i }
```

Go back to original vectors:

```
for (i in n1, j in 4) do
    { (i,(j*iy)) y' 0 =  (i,(j*iy)) y 0 + (i,(j*ix)) (da X) 0 }
```

Nest loops and introduce vector-index notation:

```
for (i in n1) do {
      for (j in 4) do {
           y' [(4i +j) * iy] =  y [(4i +j) * iy] + x [(4i +j) * ix ]
           }
}
```

Follow-up of (54)

Rename strides to original identifiers:

```
for (i in n1) do {
      for (j in 4) do {
           y' [(4i +j) * inc_y] =  y [(4i +j) * inc_y] + x [(4i +j) * inc_x ]
           }
}
```

Unroll inner loop:

```
for (i in n1) do {
    let ix = 4i*inc_x; iy = 4i*inc_y
    in y' [iy]           =  y [iy] + x [ix] ;
       y' [iy + inc_y]   =  y [iy +inc_y] + x [ix +inc_x] ;
       y' [iy + 2*inc_y] =  y [iy + 2*inc_y] + x [ix + 2* inc_x ] ;
       y' [iy + 3*inc_y] =  y [iy + 3*inc_y] + x [ix + 3* inc_x ]
}
```

Calculation of (40)

Making $X = id$ in (39) we get $id = \mathbf{vec}_k\, A \Leftrightarrow A = \epsilon_k$, and thus $\mathbf{vec}_k\, \epsilon_k = id$. Then:

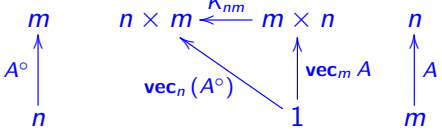
$$
\begin{array}{ll}
 & \mathbf{vec}_k\, \epsilon_k = id \\
\Leftrightarrow & \quad \{ \text{ go pointwise } \} \\
 & y\ (\mathbf{vec}_k\, \epsilon_k)\ x \ = \ (y = x) \\
\Leftrightarrow & \quad \{ \text{ change variables } \} \\
 & (c', b')\ (\mathbf{vec}_k\, \epsilon_k)\ (c, b) \ = \ ((c', b') = (c, b)) \\
\Leftrightarrow & \quad \{ \text{ pointwise definition (38) ; pairwise equality } \} \\
 & b'\ \epsilon_k\ (c', (c, b)) \ = \ (c' = c) \times (b' = b)
\end{array}
$$

Transposition (pointwise)

Commutation matrix K_{nm}:

$$\mathbf{vec}_n\,(A^\circ) = K_{nm} \cdot \mathbf{vec}_m\,A$$

Diagram:



K_{nm} 'is' the *swap* function

$$swap\,(a, b) = (b, a)$$

which is an isomorphism.

Transposition (pointwise)

(Abbreviating $\mathbf{vec}_k\ M$ to $\overline{M}^k$ to save ink):

$$\overline{A^\circ}^n = K_{nm} \cdot \overline{A}^m$$

$$\Leftrightarrow \qquad \{\ K_{nm} \text{ is swap, which is an isomorphism}\ \}$$

$$swap^\circ \cdot \overline{A^\circ}^n = \overline{A}^m$$

$$\Leftrightarrow \qquad \{\ \text{go pointwise; "nice rule" (9)}\ \}$$

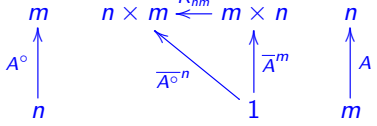
$$(i,j)\ \overline{A^\circ}^n\ 0 = (j,i)\ \overline{A}^m\ 0$$

$$\Leftrightarrow \qquad \{\ \text{go strict: (52)}\ \}$$

$$(i\ m + j)\ \overline{A^\circ}^n\ 0 = (j\ n + i)\ \overline{A}^m\ 0$$

$$\Leftrightarrow \qquad \{\ \text{vectorial form}\ \}$$

$$\overline{A^\circ}^n\ [i\ m + j] = \overline{A}^m\ [j\ n + i] \qquad (55)$$



Transposition (C code)

Thus the code, extracted from algoritmica.org:

```
void matmul(const float *a, const float *_b, float *c, int n) {
    float *b = new float[n * n];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            b[i * n + j] = _b[j * n + i];
....
```

Note the different "thinning factors" m and n in the vectorized matrices of (55). In the code above, $m = n$.

Vectorization of diagonals (pointwise)

How is the **diagonal** of a square matrix $n \xrightarrow{M} n$ (12) obtained from the vectorization

$$n^2 \xleftarrow{\mathbf{vec}_n\, M} 1$$

of M?

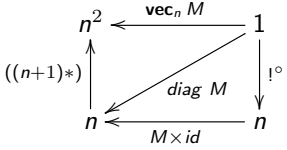
Pointwise calculation aside.

$$
\begin{aligned}
& j\ M\ j \\
= & \quad \{ \text{ vectorize } n \xleftarrow{M} n \text{ to } n^2 \xleftarrow{\mathbf{vec}_n\, M} 1 \ \} \\
& (j, j)\ (\mathbf{vec}_n\, M)\ 0 \\
= & \quad \{ \text{ strict interpretation } \} \\
& (n\, j + j)\ (\mathbf{vec}_n\, M)\ 0 \\
= & \quad \{ \text{ arithmetics } \} \\
& ((n+1)\, j)\ (\mathbf{vec}_n\, M)\ 0 \\
= & \quad \{ \text{ nice rule (9) } \} \\
& j\ (((n+1)*)^\circ \cdot (\mathbf{vec}_n\, M))\ 0
\end{aligned}
$$

Vectorization of diagonals (pointwise)

Summary: the j-component of the diagonal of $n \xrightarrow{M} n$ can be found in cell $(\mathbf{vec}_n\, M)\, [n\, j + j]$ of its vectorization.

Diagram:



$$((n+1)*)^\circ \cdot (\mathbf{vec}_n\, M) = diag\ M$$

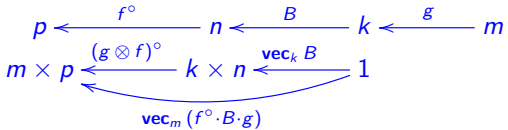
NB: $(n+1)\,(n-1) = n^2 - 1$

Vectorization versus "nice" rule

For functions, the Roth relationship (49) becomes:

$$\mathbf{vec}_m\,(f^\circ \cdot B \cdot g) = (g \otimes f)^\circ \cdot \mathbf{vec}_k\,B \tag{56}$$

cf:



Pointwise:[3]

$$(a, b)\ (\mathbf{vec}_m\,(f^\circ \cdot B \cdot g))\ 0 = (g\ a, f\ b)\ (\mathbf{vec}_k\,B)\ 0 \tag{57}$$

Pointwise strict:

$$\mathbf{vec}_m\,(f^\circ \cdot B \cdot g)\ [a\ p + b] = \mathbf{vec}_k\,B\ [(g\ a)\ n + f\ b]$$

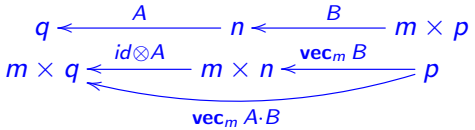
[3] (57) can be easily shown to be the vectorization of the "nice" rule (9).

Roth pointwise

$C = id$, ie. absorption:

$$\mathbf{vec}_m\,(A \cdot B) \;=\; (id \otimes A) \cdot \mathbf{vec}_m\,B \tag{58}$$

Diagram:



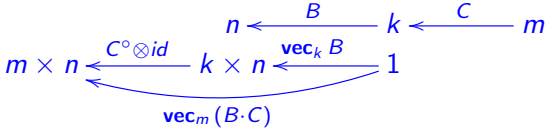
Pointwise:

$$(y, x)\;(\mathbf{vec}_m\,(A \cdot B))\;z = \langle \Sigma\; w \;:\; w \in n :\; (x\;A\;w) * ((y, w)\;\mathbf{vec}_m\,B\;z)\rangle$$

Roth pointwise

$A = id$:

$$\mathbf{vec}_m\,(B \cdot C) \;=\; (C^\circ \otimes id) \cdot \mathbf{vec}_k\,B \tag{59}$$



Pointwise:

$$\mathbf{vec}_m\,(B \cdot C)\,(a,b) = \langle \Sigma\ c\ :\ c \in n :\ (c\ C\ a) * \mathbf{vec}_k\,B\,[(c,b)] \rangle$$

References

O. Kiselyov. Optimizing axpy with twists and turns, 2024. URL `https://okmij.org/ftp/meta-programming/tutorial/daxpy.html`. https://okmij.org/ftp/meta-programming/tutorial/daxpy.html.

H.D. Macedo and J.N. Oliveira. Typing linear algebra: A biproduct-oriented approach. *SCP*, 78(11):2160–2191, 2013.

S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, September 1998.

J. N. Oliveira and H. D. Macedo. The data cube as a typed linear algebra operator. In *Proc. of the 16th Int. Symposium on Database Programming Languages*, DBPL '17, pages 6:1–6:11, New York, NY, USA, 2017. ACM. (DOI).