IEEE SiPS 2026

Low-Latency Coded Tensor–Matrix Multiplication

for Distributed Signal Processing Systems

Ahmad Tanha
IPSA, France

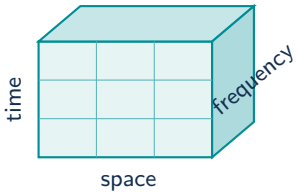
Ali Khalesi
IPSA & LINCS Lab, Paris, France







Signal processing data already have multiple axes



Radar / wireless
antenna × time × frequency

Hyperspectral imaging
row × column × wavelength

Multi-sensor processing
structured measurements across modes

Large tensor operations need distributed execution under latency constraints.

Practical examples of decoding bottlenecks

Research implementations on commercial cloud platforms

Microsoft Azure: matrix multiplication [2]

With 24 workers, conventional MDS decoding takes most of the job time for a 4800×100 by 100×4800 product.

Microsoft Azure: iterative linear regression [2]

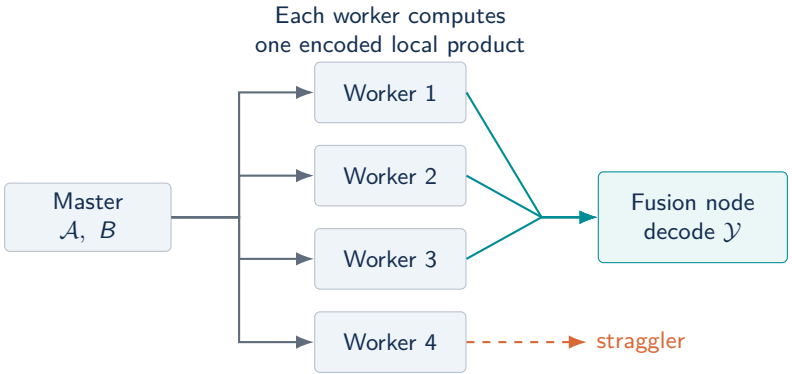
Repeated decoding makes a fine-grained coded scheme slower overall. Lower decoding cost can outweigh the benefit of faster workers.

AWS Lambda: large matrix products for analytics [3]

Global decoding must read the full output. Cloud I/O and the master's memory become recovery bottlenecks.

Fast parallel products still need efficient reconstruction.

System model: a master, workers, and a fusion node



Each worker receives encoded blocks and returns one local product.

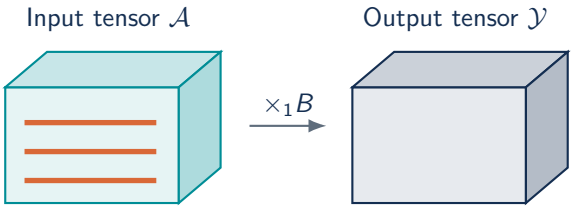
Practical workloads apply the same matrix along one tensor axis

Common operation: $\mathcal{Y} = \mathcal{A} \times_1 B$. $\mathcal{A}$: input data; B: shared transformation; $\mathcal{Y}$: transformed data.

Practical operation	**Input tensor $\mathcal{A}$ and matrix B**	**Result $\mathcal{Y}$**
Hyperspectral spectral projection	$\mathcal{A}$: wavelength $\times$ height $\times$ width. B: spectral projection weights.	A feature vector at every pixel; spatial locations stay unchanged.
Nonlinear-network training [6]	A: activation/error tensor. B: weights or transpose.	Forward/backward tensor–matrix products within nonlinear training.
Fourier transform along one axis [7]	$\mathcal{A}$: time $\times$ sensors $\times$ recordings. B: Fourier transform matrix.	Frequency coefficients for every sensor and recording.

Why distribute? Large arrays may exceed one machine's memory or processing capacity.

A matrix acts on every mode-1 fiber



$$\mathcal{Y} = \mathcal{A} \times_1 B$$

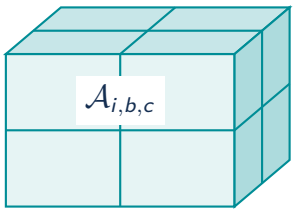
$B \in \mathbb{R}^{P \times I_1}$

The same matrix transforms each fiber along the first mode.

$$\mathcal{Y}(p, i_2, i_3) = \sum_{i_1=1}^{I_1} B(p, i_1)\mathcal{A}(i_1, i_2, i_3)$$

Partition the contraction axis and the free axes

Contraction index i
free-mode indices b, c



$s \times u_2 \times u_3$ subtensors

$$B = [B_0 \; B_1 \; \cdots \; B_{s-1}]$$

$$\mathcal{A}_{i,b,c} \in \mathbb{R}^{\frac{I_1}{s} \times \frac{I_2}{u_2} \times \frac{I_3}{u_3}}$$

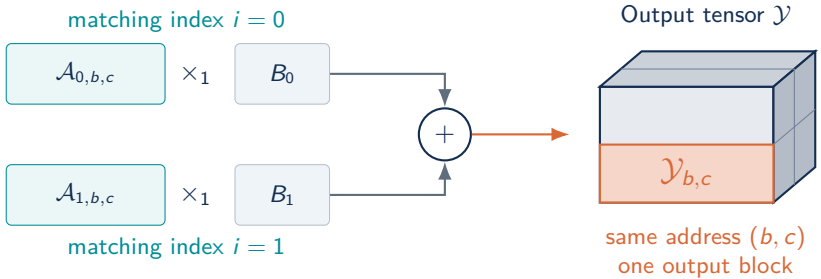
$$\mathcal{Y}_{b,c} = \sum_{i=0}^{s-1} \mathcal{A}_{i,b,c} \times_1 B_i$$

$$M = I_2 I_3, \qquad t = u_2 u_3$$

All partition factors divide their dimensions.

Matching input blocks produce one output block

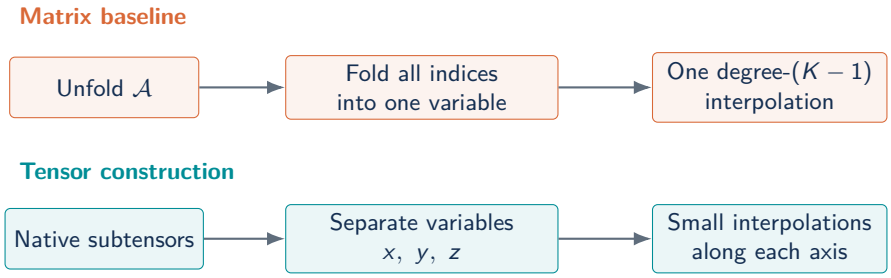
Fix the free-mode address (b, c). Example: $s = 2$ contraction blocks.



$$\mathcal{Y}_{b,c} = \sum_{i=0}^{s-1} \mathcal{A}_{i,b,c} \times_1 B_i \qquad \mathcal{Y}_{b,c} \in \mathbb{R}^{P \times \frac{I_2}{u_2} \times \frac{I_3}{u_3}}$$

b: mode-2 block index c: mode-3 block index $t = u_2 u_3,\ M = I_2 I_3$

Tensor-PolyDot keeps the free modes separate



The local multiplication stays the same size. The decoder changes.

Separate variables encode separate block indices

$$\widehat{B}(x) = \sum_{i=0}^{s-1} B_i x^i$$

$$\widehat{\mathcal{A}}(x,y,z) = \sum_{j=0}^{s-1} \sum_{b=0}^{u_2-1} \sum_{c=0}^{u_3-1} \mathcal{A}_{j,b,c}\, x^{s-1-j} y^b z^c$$

$$H(x,y,z) = \widehat{\mathcal{A}}(x,y,z) \times_1 \widehat{B}(x)$$

x: contraction alignment

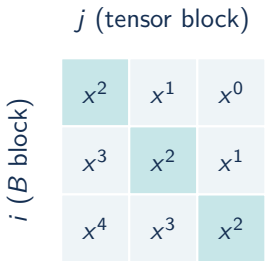
y: mode-2 address

z: mode-3 address

Worker (α, β, γ) evaluates and multiplies locally.

Matching contraction indices align at x^{s-1}

$$H = \sum_{i,j,b,c} (\mathcal{A}_{j,b,c} \times_1 B_i)\, x^{s-1+i-j} y^b z^c$$



Example: $s = 3$

$$i = j \implies s - 1 + i - j = s - 1$$

$$\boxed{[x^{s-1}]H = \sum_{b,c} \mathcal{Y}_{b,c}\, y^b z^c}$$

Diagonal terms give the required block sums.

Other x coefficients contain cross terms.

Example: from numerical evaluations to four output blocks

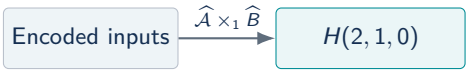
For $s = u_2 = u_3 = 2$: $H(x, y, z) = C_0(y, z) + xF(y, z) + x^2 C_2(y, z), \quad F = \mathcal{Y}_{0,0} + y\mathcal{Y}_{1,0} + z\mathcal{Y}_{0,1} + yz\mathcal{Y}_{1,1}.$

1. Assign numerical points to workers

Choose $x \in \{0, 1, 2\}$, $y, z \in \{0, 1\}$. Every triple identifies one of $3 \cdot 2 \cdot 2 = 12$ workers.

At $(x, y, z) = (2, 1, 0)$, the master sends

$$\widehat{B} = B_0 + 2B_1,$$
$$\widehat{\mathcal{A}} = 2(\mathcal{A}_{0,0,0} + \mathcal{A}_{0,1,0}) + \mathcal{A}_{1,0,0} + \mathcal{A}_{1,1,0}.$$



A point is a choice of weights, not a tensor index.

2. Remove the unwanted x coefficients

For each fixed (y, z), combine three returns:

$$F(y, z) = \frac{-3H(0, y, z) + 4H(1, y, z) - H(2, y, z)}{2}$$

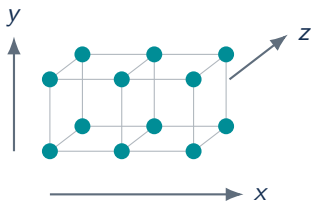
3. Separate the four output addresses

Write $F_{uv} = F(u, v)$ for $u, v \in \{0, 1\}$.

$$\mathcal{Y}_{0,0} = F_{00}, \qquad \mathcal{Y}_{1,0} = F_{10} - F_{00},$$
$$\mathcal{Y}_{0,1} = F_{01} - F_{00},$$
$$\mathcal{Y}_{1,1} = F_{11} - F_{10} - F_{01} + F_{00}.$$

These additions apply entry by entry to whole blocks.

A Cartesian evaluation grid enables axis-wise decoding



Toy example: $x = 0, 1, 2$; $y, z = 0, 1$
$3 \times 2 \times 2 = 12$ evaluation points

$$\deg_x H = 2s - 2, \quad \deg_y H = u_2 - 1$$

$$\deg_z H = u_3 - 1, \quad q \equiv 2s - 1$$

$$\boxed{K = q\, u_2 u_3 = q\, t}$$

Distinct nodes on each axis give an invertible tensor-product interpolation system.

The response set must support the structured decoder.

The decoder solves small systems along each axis

Matrix baseline [4,5]: one system of length $K = q\,u_2 u_3$



For $s = 8$, $u_2 = u_3 = 4$: 240 versus axis lengths 15, 4, 4.

Independent fibers within each stage can run in parallel.

What do we count? Work needed to reconstruct the output

Use the same three-mode tensor as in the example.

P: output length on the multiplied axis
s: partitions on the contraction axis
$q = 2s - 1$: number of x evaluations
$M = I_2 I_3$: entries across the two free axes
u_2, u_3: partitions on the two free axes
$t = u_2 u_3$: number of output blocks

Thus PM is the output size and $K = qt$ is the required grid-response count.

Matrix route [4,5], cost model [1]

$$C_{\mathrm{mat}} \sim PMq^2t$$

One long interpolation problem per returned-block entry.

Tensor route, cost model [1]

$$C_{\mathrm{ten}} \sim PM(q^2 + u_2^2 + u_3^2)$$

Separate interpolation along the three coding axes.

Modeled work reduction: $S = C_{\mathrm{mat}}/C_{\mathrm{ten}} = q^2t/(q^2 + u_2^2 + u_3^2)$.

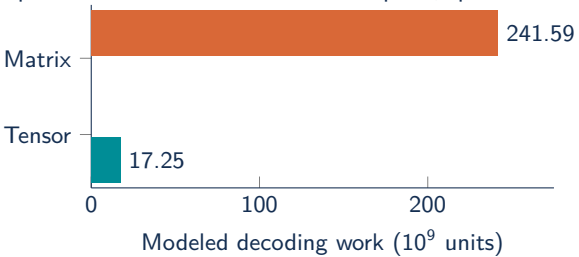
C: decoding work; $\sim$: scaling model, not an exact instruction count.

A larger S means less reconstruction work for the same output.

A concrete configuration gives $14.0\times$ less decoding work

Input: $512 \times 512 \times 512$ tensor; 256×512 matrix. **Output:** $256 \times 512 \times 512$ tensor.

Split the contraction axis into $s = 8$ parts; split each free axis into $u_2 = u_3 = 4$ parts.



$q = 2s - 1 = 15$ x evaluations.

$t = 4 \cdot 4 = 16$ output blocks.

$K = 15 \cdot 16 = 240$ grid responses.

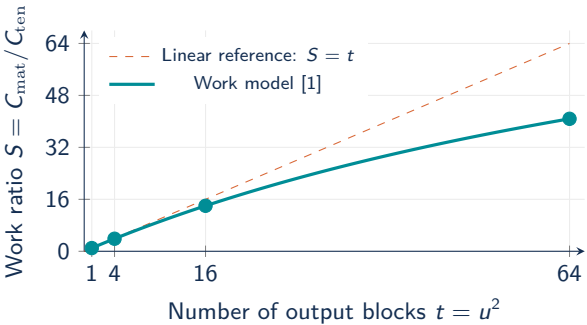
Work ratio: $S = \dfrac{241.59}{17.25} \approx 14.0$

92.9% less modeled decoding work.

Values come from the formulas in [1]; they are not measured execution times.

More output blocks increase the modeled gain

Keep $s = 8$ contraction partitions, so $q = 2s - 1 = 15$. Use u equal partitions on each free axis.



Reading the curve

$u_2 = u_3 = u$, so $u_2^2 + u_3^2 = 2t$.

$$S(t) = \frac{225t}{225 + 2t}$$

$225 = 15^2$ is the fixed x-axis cost factor.

$2t$ is the growing free-axis cost factor.

16 blocks: 14.0× work ratio.
64 blocks: 40.8× work ratio.

Dots use $u = 1, 2, 4, 8$, which divide 512. The curve illustrates the formula between them.

Less decoding work reduces total time by a smaller factor

From work to estimated seconds [1]

R: assumed decoding throughput.
$R = 5 \times 10^9$ work units per second.

$$\text{Decoding time} = C/R$$

C: the work shown on slide 16.

Matrix: 48.32 s



Tensor: 3.45 s



From decoder gain to whole-job gain

f: fraction of original job time spent decoding.
$S_{\text{decode}} = 14$: decoder time ratio, assuming equal throughput.

$$S_{\text{total}} = \frac{1}{(1-f) + f/S_{\text{decode}}}$$

S_{total}: original job time / new job time.

50% originally decoding: 1.87× overall.

80% originally decoding: 3.89× overall.

Other stages retain their original duration.

The benefit depends on how much of the job decoding occupies.

Comparison: the workers do the same-sized local products

Same input sizes and partitions: s contraction parts, $t = u_2 u_3$ output blocks.

What we compare	Matrix route [4,5]	Tensor route [1]
Required responses	$K = (2s-1)t$	Same count; structured set needed
Worker computation	One encoded local product	Same input and output block sizes
Worker memory	Store two inputs and one result	Same modeled block storage
Results transmitted	K blocks, each with PM/t entries	Same total: $(2s-1)PM$ entries
Reconstruction	One polynomial in one variable	Separate interpolation along each axis
Example decoding work	241.59 billion units	17.25 billion units (model [1])

P: output length on axis 1; $M = l_2 l_3$: size across free axes; PM: total output entries.

The modeled saving comes from reconstructing the returned results. Equal response counts do not establish equal tolerance of arbitrary stragglers.

Keeping the tensor axes separate simplifies reconstruction

What we construct

Encoded blocks keep the two output-block addresses in separate polynomial variables.

How we reconstruct

Combine worker results to isolate the contraction sum, then recover each output block along the free axes.

What stays the same

For fixed partitions, local product sizes and the modeled return-data volume match the matrix route.

$14\times$

less modeled decoding work in the example [1]

241.59/17.25 billion work units

Next: test missing-worker patterns, numerical accuracy, and measured whole-job time.

Questions?

ali.khalesi@ipsa.fr

Backup: how three coefficient addresses become one index

x, y, z: coding variables. a, b, c: their exponents, identifying a coefficient.

$q = 2s - 1$: number of x coefficients; u_2, u_3: numbers along y, z.

Replace y by x^q and z by x^{qu_2} to obtain one polynomial:

$$x^a y^b z^c \longmapsto x^{a+qb+qu_2c}, \qquad 0 \le a < q, \quad 0 \le b < u_2, \quad 0 \le c < u_3.$$

The new exponent is a unique address for every coefficient.

$$\text{Largest exponent} = (q-1) + q(u_2 - 1) + qu_2(u_3 - 1) = K - 1.$$

Here $K = qu_2u_3$ is the number of coefficients, and also the interpolation sample count.

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

Example: $q = 3$, $u_2 = u_3 = 2$ gives 12 coefficients, indexed 0 to 11.

Backup: the count of responses is not the whole recovery guarantee

K: number of worker results used. A straggler is a worker whose result is still missing.

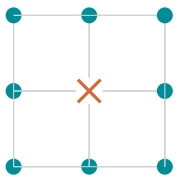
One-variable interpolation

A polynomial of degree $K - 1$ is determined by any K distinct evaluation points in exact arithmetic.

The illustrated axis-wise decoder

Needs a complete grid: q points along x, u_2 along y, u_3 along z.

$q = 2s - 1$; u_2, u_3 are free-axis partition counts.



Schematic grid slice: orange marks a missing result.

Another result at an arbitrary location does not fill this gap. Redundancy or another decoder is needed.

Backup: a work model must specify the decoding algorithm

What the main results establish [1]

The $14\times$ value compares the manuscript's work formulas. It is not a universal bound against every matrix decoder.

Example of a different count: dense recovery of all coefficients

For the three-mode example, the work scales as

$$\underbrace{D}_{\text{entries per returned block}} \quad \underbrace{K}_{\text{grid size}} \quad \underbrace{(q + u_2 + u_3)}_{\text{sum of axis lengths}}.$$

$D = PM/t$, where PM is the output size and $t = u_2 u_3$ its block count.

$K = qt$ and $q = 2s - 1$, where s counts contraction partitions.

Why implementation matters

Extracting only the needed coefficient, precomputing interpolation weights, or using a fast interpolation algorithm changes the operation count.

A fair measured comparison must specify both decoders and the hardware.

References: manuscript and practical motivation

[1] Tensor method and numerical work model
A. Tanha and A. Khalesi, "Low-Latency Coded Tensor–Matrix Multiplication for Distributed Signal Processing Systems," supplied manuscript. Eqs. (12)–(14) underpin the numerical comparisons in this talk.

[2] Azure matrix multiplication and linear regression
X. Fan, P. Soto, X. Zhong, D. Xi, Y. Wang, and J. Li, "Leveraging Stragglers in Coded Computing with Heterogeneous Servers," IEEE/ACM IWQoS, 2020. Sec. VII-A, Figs. 7–8; Sec. VII-B, Fig. 11.

[3] Serverless recovery I/O and memory
V. Gupta, D. Carrano, Y. Yang, V. Shankar, T. Courtade, and K. Ramchandran, "Serverless Straggler Mitigation using Local Error-Correcting Codes," IEEE ICDCS, 2020. Sec. II-B, Fig. 5. arXiv:2001.07490.

References: coding background and practical operations

[4] Generalized PolyDot
S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A Unified Coded Deep Neural Network Training Strategy based on Generalized PolyDot codes," IEEE ISIT, 2018, pp. 1585–1589. doi:10.1109/ISIT.2018.8437852

[5] MatDot / PolyDot and recovery tradeoffs
S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the Optimal Recovery Threshold of Coded Matrix Multiplication," IEEE Transactions on Information Theory, vol. 66, no. 1, pp. 278–301, 2020. doi:10.1109/TIT.2019.2929328

[6] Neural-network linear layers
PyTorch documentation, "Linear." docs.pytorch.org: torch.nn.Linear.

[7] Large distributed Fourier transforms
FFTW documentation, "Distributed-memory FFTW with MPI." fftw.org: MPI transforms.

How to read the comparison: [4,5] give the coding background. The specific dense-decoder baseline, tensor cost expression, and 14× estimate shown here come from [1].