Cuboid Partitioning for Hierarchical Coded Matrix Multiplication

Shahrzad Kiani, Nuwan Ferdinand, Stark C. Draper

shahrzad.kianidehkordi@mail. utoronto.ca,
{nuwan.Ferdinand, stark.draper}@utoronto.ca

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Abstract

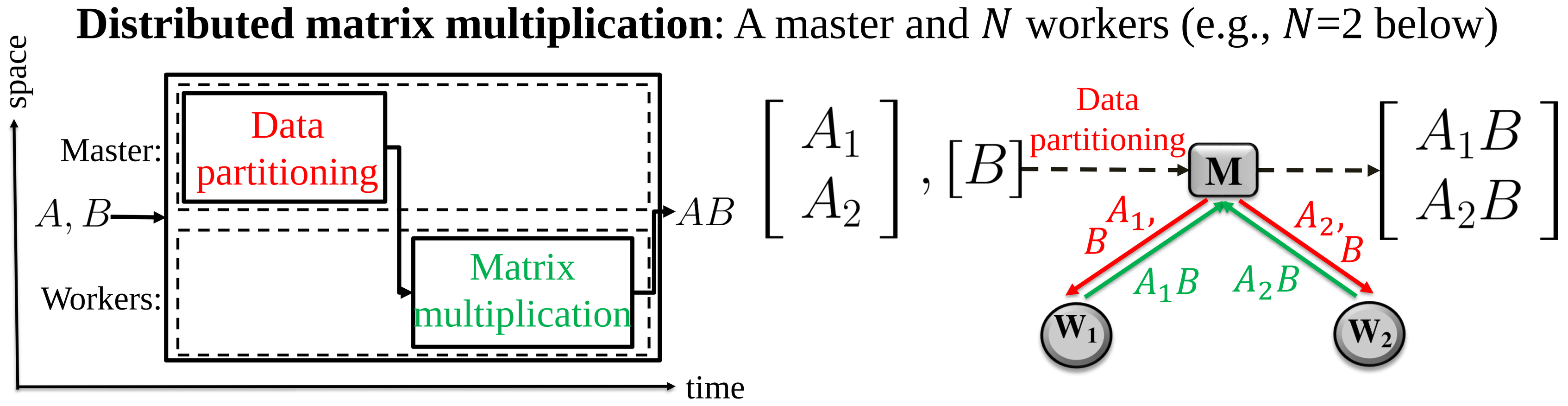
Coded matrix multiplication is a technique to enable straggler-resistant multiplication of large matrices in distributed computing systems. In this work, we first present a conceptual framework to represent the division of work amongst processors in coded matrix multiplication as a cuboid partitioning problem. This framework allows us to unify existing methods and motivates new techniques. Building on this framework, we apply the idea of hierarchical coding (Ferdinand & Draper, 2018) to coded matrix multiplication. The hierarchical scheme we develop is able to exploit the work completed by all processors (fast and slow), rather than ignoring the slow ones, even if the amount of work completed by stragglers is much less than that completed by the fastest workers. On Amazon EC2, we achieve a 37% improvement in average finishing time compared to non-hierarchical schemes.

Introduction: Distributed matrix multiplication

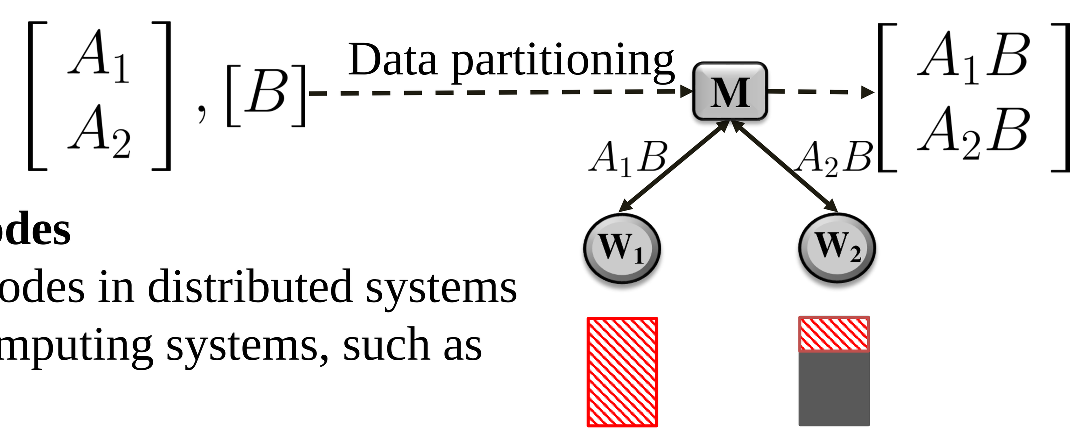
Goal: Compute $C = AB$, where $A \in \mathbb{R}^{(N_x \times N_z)}$, $B \in \mathbb{R}^{(N_z \times N_y)}$, and this product
requires $\mathcal{O}(N_x N_z N_y)$ basic operations
utilizes $\mathcal{O}(N_x N_z + N_x N_y + N_z N_y)$ memory

Parallelization: Serial matrix multiplication impractical → need to parallelize

Distributed matrix multiplication: A master and N workers (e.g., N=2 below)



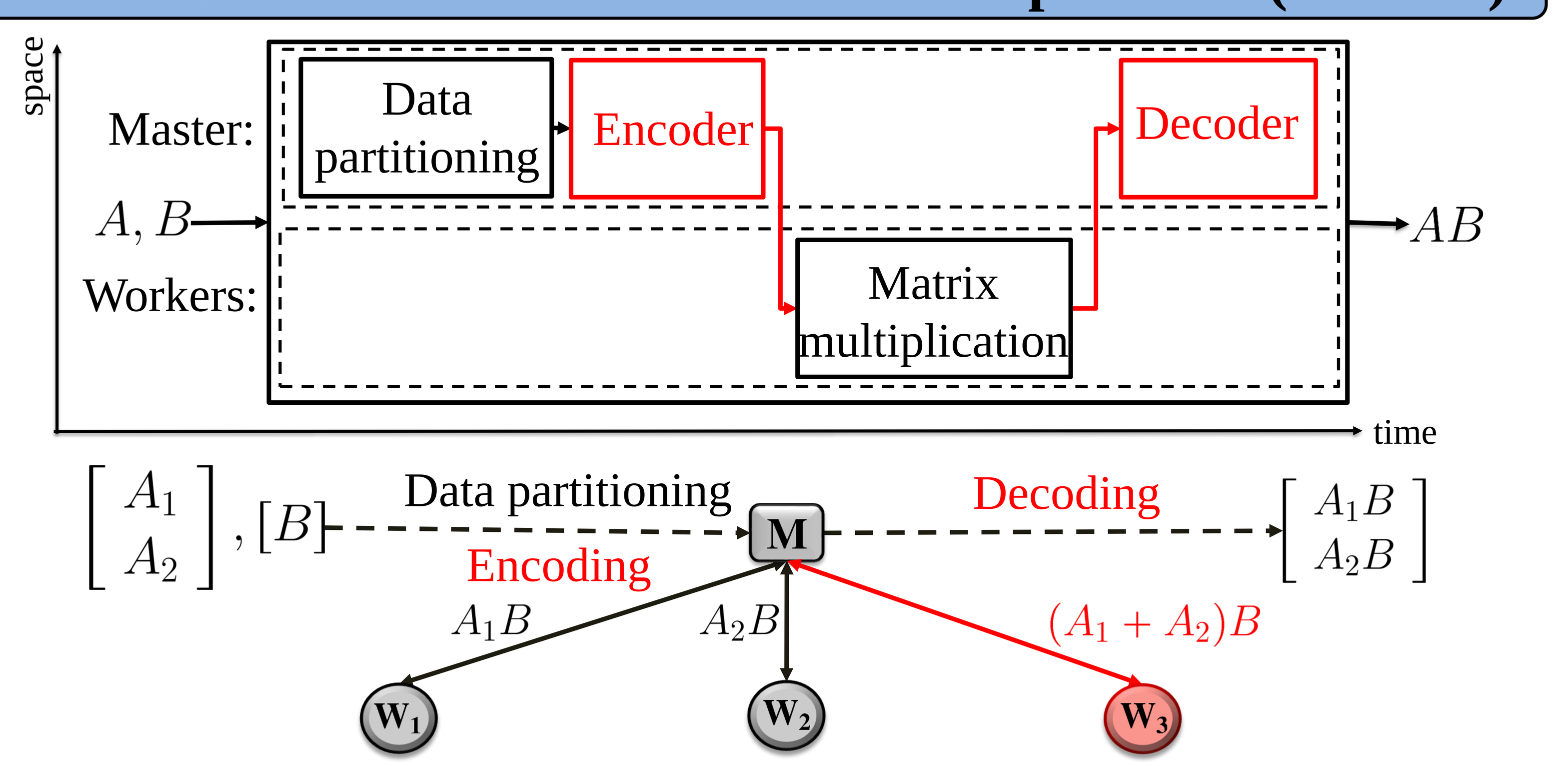
Challenge: Stragglers delay computation



Straggling compute nodes

- unpredictably slow nodes in distributed systems
- observed in cloud computing systems, such as Amazon EC2

Solution: Distributed coded matrix multiplication (DCMM)



Decompose matrix multiplication into basic operations represented by unit cubes

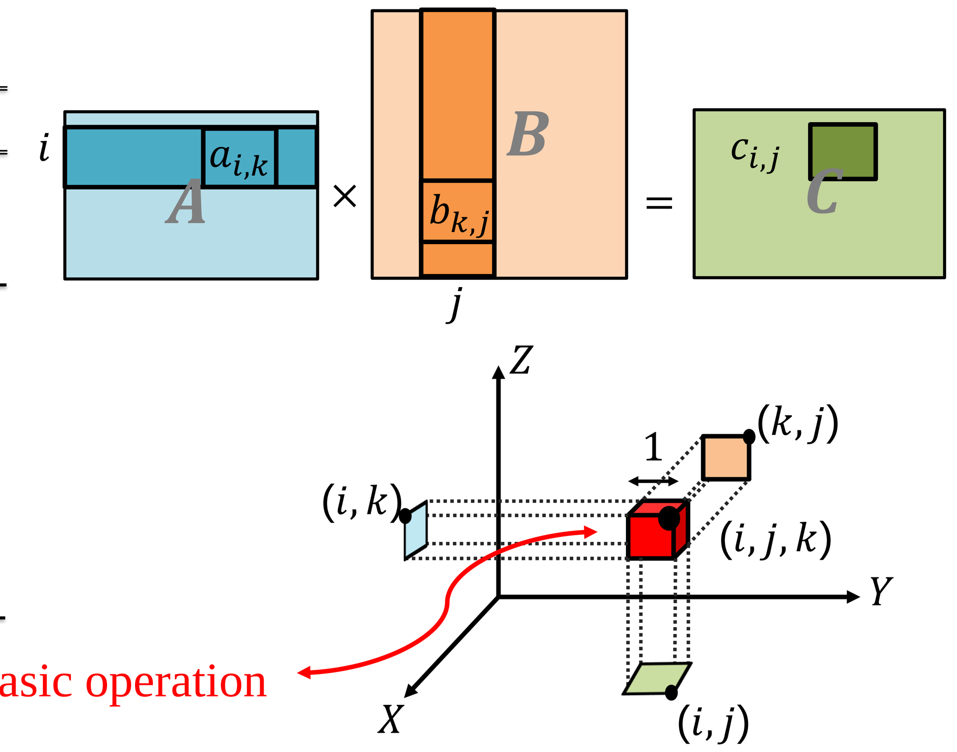
Algorithm 1: $[C] = \text{MatMult}(A, B)$

Input: $A \in \mathbb{R}^{N_x \times N_z}$, $B \in \mathbb{R}^{N_z \times N_y}$

Output: $C \in \mathbb{R}^{N_x \times N_y}$

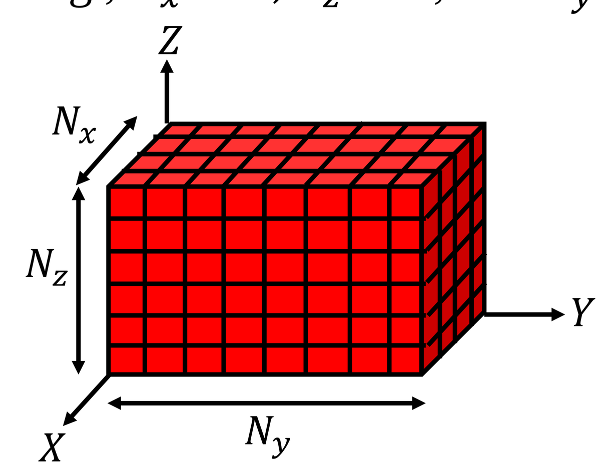
1: *forall* $i \in [N_x], j \in [N_y]$:
2: $c_{i,j=0}$
3: *for* $k \in [N_z]$:
4: $c_{i,j} = c_{i,j} + a_{i,k} b_{k,j}$
5: *end for*
6: *end forall*

The (i, j, k)-th basic operation



View the product AB as a rectangular cuboid

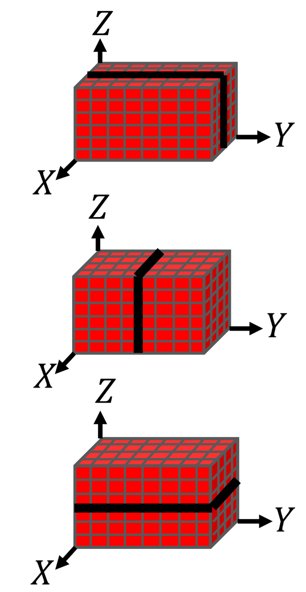
E.g., $N_x = 4, N_z = 6$, and $N_y = 8$



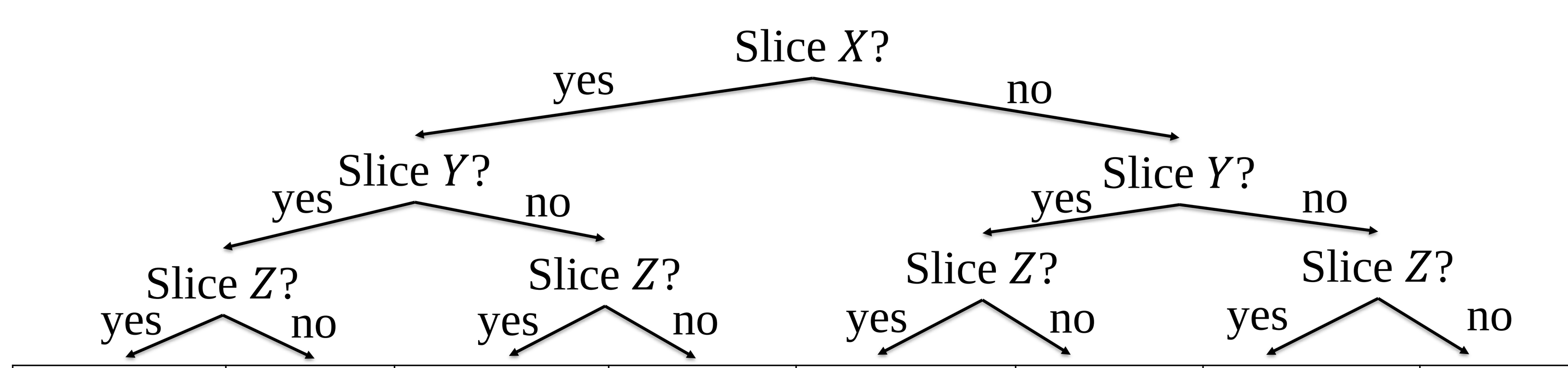
$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} B \leftrightarrow$ *X axis slicing*

$A[B_1\ B_2] \leftrightarrow$ *Y axis slicing*

$[A_1\ A_2] \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \leftrightarrow$ *Z axis slicing*



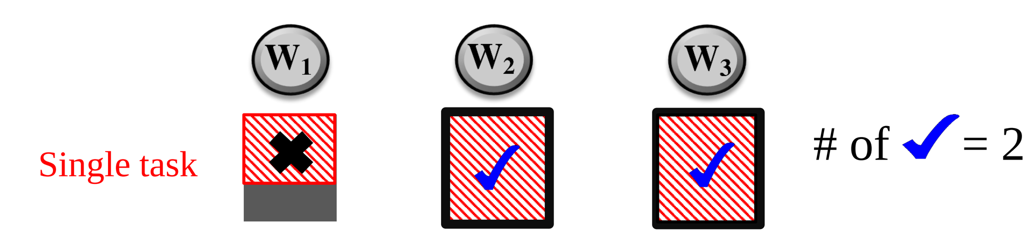
Cuboid visualization of data partitioning phase in DCMM



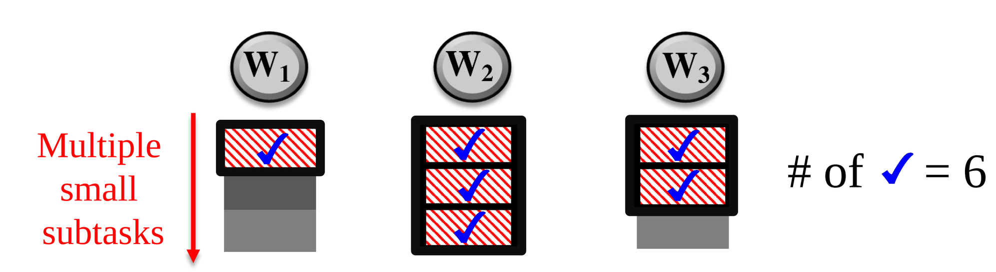
$\{X,Z,Y\}$	$\{X,Y\}$	$\{X,Z\}$	$\{X\}$	$\{Z,Y\}$	$\{Y\}$	$\{Z\}$	$\{\}$
PolyDot, Ent-Poly	Poly, Product	PolyDot, Ent-Poly	MatVec, Poly,	PolyDot, Ent-Poly	VecMat, Poly,	MatDot	Rep

Novel strategies to exploit stragglers

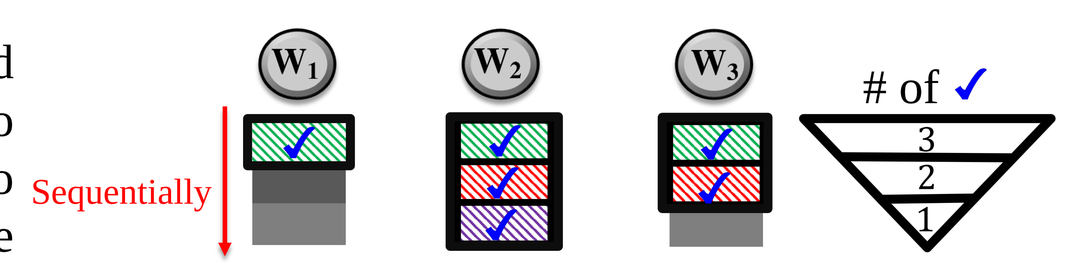
Standard approach: Throw away half-completed work.



Sum-rate coding [Kiani et al. ISIT'18]: (i) Split into smaller subtasks, and (ii) code across all subtasks.



Hierarchical coding [Ferdinand & Draper ISIT'18]: (i) Split into smaller subtasks, (ii) collect into layers of subtasks, and (iii) code with each layer.



Hierarchical coded matrix multiplication

Phase 1: Data partitioning and cuboid visualization

step 1: split job into L layers of computation

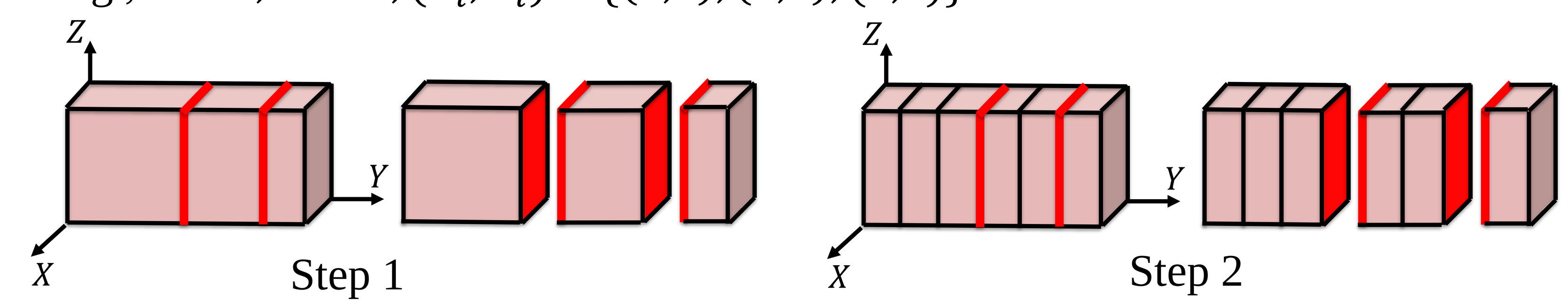
step 2: divide lth layer into K_l sub-computations

Phase 2: Data encoding and distribution. Encode lth layer using an (N, K_l) code.

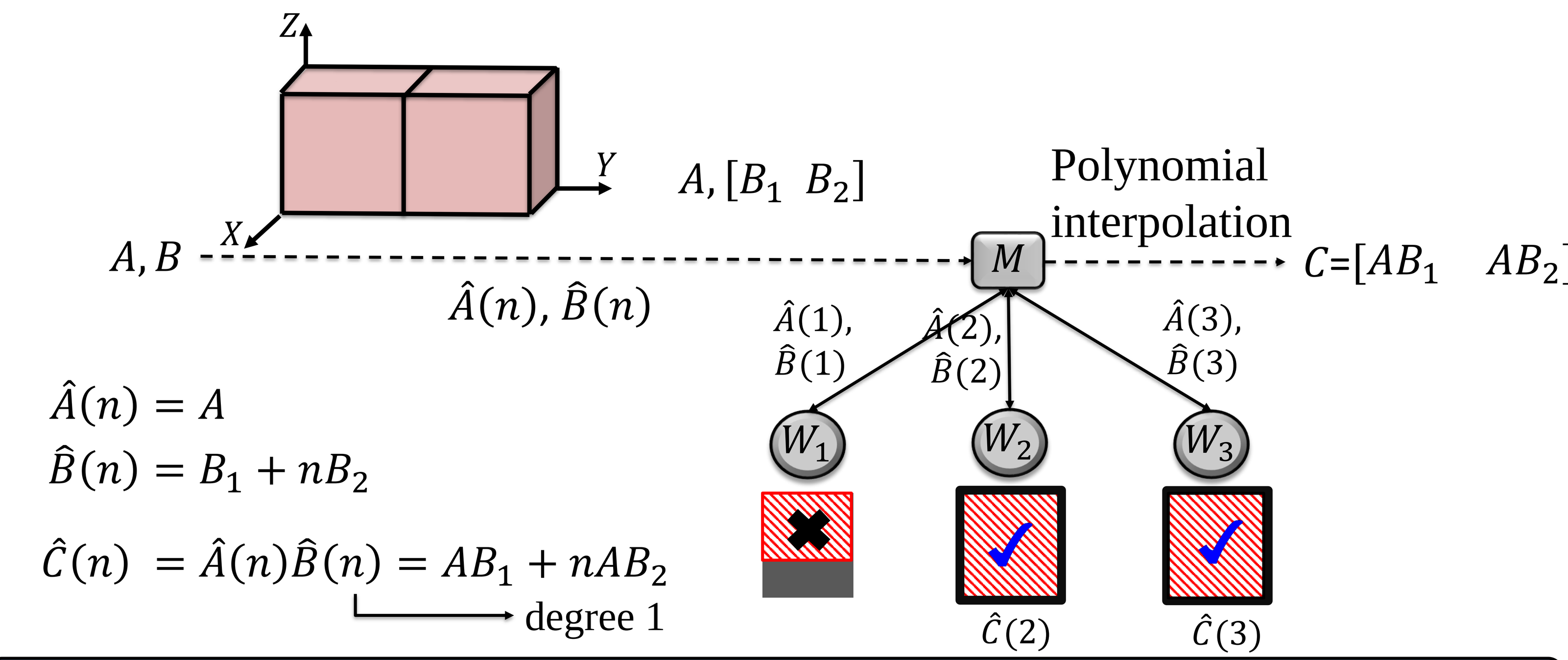
Phase 3: Worker computation and decoding. Any R_l encoded completed tasks can be decoded to recover lth layer of desired matrix product.

(L: # layers, N: # workers, K_l: information dimension, R_l: recovery threshold)

E.g., $L = 3, N = 3, (K_l, R_l) \in \{(3,3), (2,2), (1,1)\}$



Example: Standard polynomial coding approach (poly)



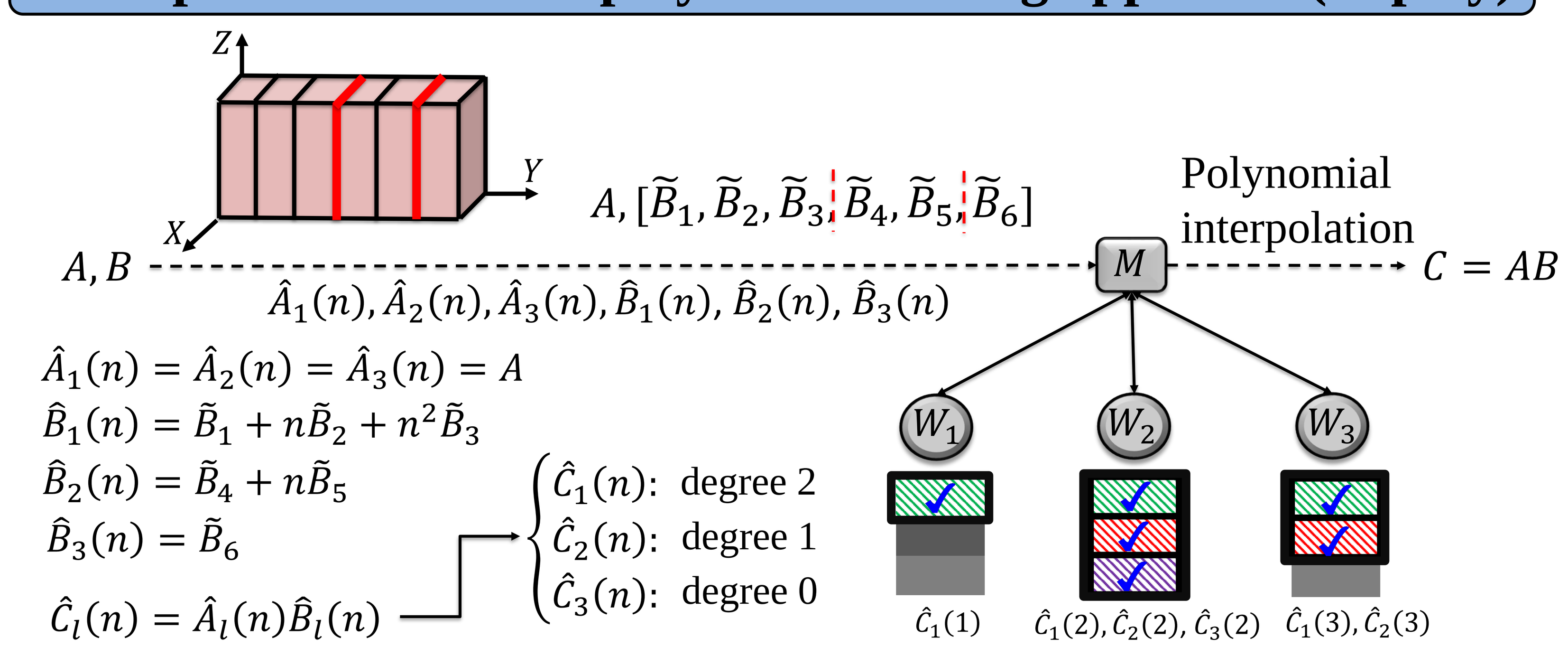
$\hat{A}(n) = A$

$\hat{B}(n) = B_1 + nB_2$

$\hat{C}(n) = \hat{A}(n)\hat{B}(n) = AB_1 + nAB_2$

degree 1

Example: Hierarchical polynomial coding approach (H-poly)



$\hat{A}_1(n) = \hat{A}_2(n) = \hat{A}_3(n) = A$

$\hat{B}_1(n) = \tilde{B}_1 + n\tilde{B}_2 + n^2\tilde{B}_3$

$\hat{B}_2(n) = \tilde{B}_4 + n\tilde{B}_5$

$\hat{B}_3(n) = \tilde{B}_6$

$\hat{C}_l(n) = \hat{A}_l(n)\hat{B}_l(n)$

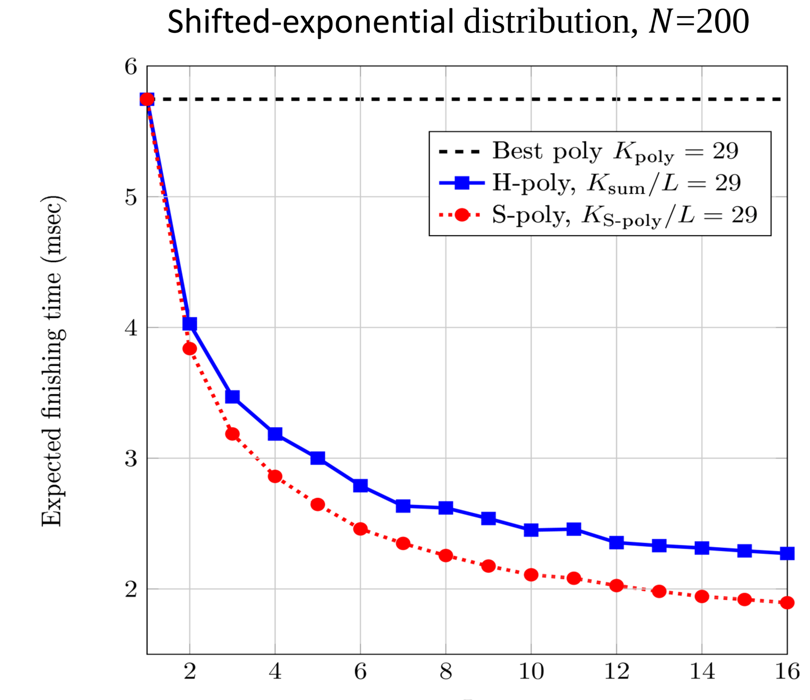
$\hat{C}_1(n)$: degree 2
$\hat{C}_2(n)$: degree 1
$\hat{C}_3(n)$: degree 0

$\hat{C}_1(1)$ $\quad$ $\hat{C}_1(2), \hat{C}_2(2), \hat{C}_3(2)$ $\quad$ $\hat{C}_1(3), \hat{C}_2(3)$

Results and conclusion: H-poly vs. S-poly vs. Poly

H-poly exploits work done by all workers, including stragglers.
H-poly realizes 60% improvement in expected finishing time compared to poly.
H-poly has lower decoding time compared to sum-rate polynomial coding (S-poly).

Shifted-exponential distribution, N=200



EC2 experiment, $N_x = N_z = N_y = 1000$