Straggler-free Coding for Concurrent Matrix Multiplications

Pedro Soto, Jun Li

Florida International University

ISIT 2020

Background

- Modern distributed computing systems process large-scale data by running multiple tasks on different nodes in parallel.

Background

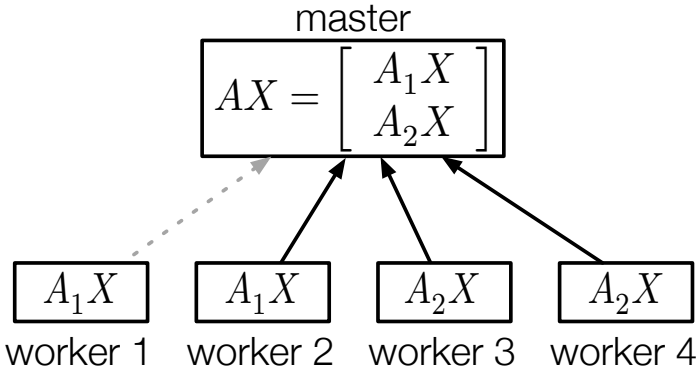
- Modern distributed computing systems process large-scale data by running multiple tasks on different nodes in parallel.
- However, stragglers are inevitable which lags the progress of the job significantly.

Background

- Modern distributed computing systems process large-scale data by running multiple tasks on different nodes in parallel.
- However, stragglers are inevitable which lags the progress of the job significantly.
- To mitigate stragglers, redundant tasks can be launched on additional nodes.

Background

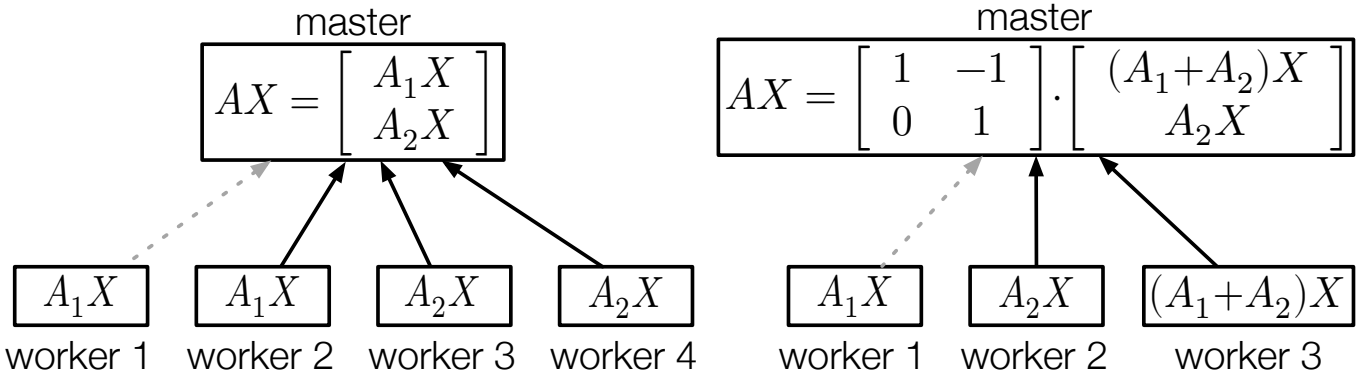
- Modern distributed computing systems process large-scale data by running multiple tasks on different nodes in parallel.
- However, stragglers are inevitable which lags the progress of the job significantly.
- To mitigate stragglers, redundant tasks can be launched on additional nodes.



(a) replicated task

Background

- Modern distributed computing systems process large-scale data by running multiple tasks on different nodes in parallel.
- However, stragglers are inevitable which lags the progress of the job significantly.
- To mitigate stragglers, redundant tasks can be launched on additional nodes.

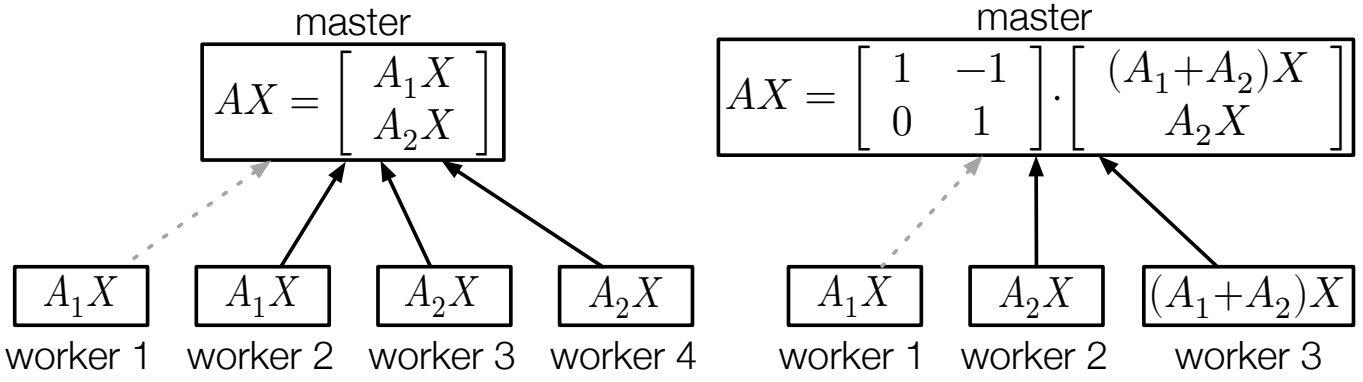


(a) replicated task

(b) coded task

Background

- Modern distributed computing systems process large-scale data by running multiple tasks on different nodes in parallel.
- However, stragglers are inevitable which lags the progress of the job significantly.
- To mitigate stragglers, redundant tasks can be launched on additional nodes.



(a) replicated task

(b) coded task

- Instead of replicating tasks, erasure coding allows creating **coded tasks** which tolerates the same number of stragglers with fewer additional tasks.

Motivation

- In this paper, we study a related problem with n concurrent matrix multiplications.

Motivation

- In this paper, we study a related problem with n concurrent matrix multiplications.
- In other words, we need to multiply n matrices $A_1, \ldots, A_n$ with $B_n, \ldots, B_n$, *i.e.*, $A_1 B_1, \ldots, A_n B_n$ in parallel.

Motivation

- In this paper, we study a related problem with n concurrent matrix multiplications.
- In other words, we need to multiply n matrices $A_1, \ldots, A_n$ with $B_n, \ldots, B_n$, *i.e.*, $A_1B_1, \ldots, A_nB_n$ in parallel.

In this paper, we propose a coding framework for distributed computing of concurrent (*i.e.*, batch) matrix multiplications.

Model

- Each worker performs one coded multiplication of the form $\tilde{A}(x) \cdot \tilde{B}(x)$.

Model

- Each worker performs one coded multiplication of the form $\tilde{A}(x) \cdot \tilde{B}(x)$.
- The n matrix multiplications are encoded into $\tilde{A}(x)$ and $\tilde{B}(x)$ as

$$\tilde{A}(x) = \sum_{i=1}^{n} A_{M_i} x^{P_i}, \text{ and } \tilde{B}(x) = \sum_{i=1}^{n} B_{N_i} x^{Q_i},$$

achieving the optimal encoding complexity based on polynomials.

Model

- Each worker performs one coded multiplication of the form $\tilde{A}(x) \cdot \tilde{B}(x)$.
- The n matrix multiplications are encoded into $\tilde{A}(x)$ and $\tilde{B}(x)$ as

$$\tilde{A}(x) = \sum_{i=1}^{n} A_{M_i} x^{P_i}, \text{ and } \tilde{B}(x) = \sum_{i=1}^{n} B_{N_i} x^{Q_i},$$

  achieving the optimal encoding complexity based on polynomials.
- Therefore, a coded task can be generated as

$$\tilde{C}(x) = \tilde{A}(x)\tilde{B}(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{M_i} B_{N_j} x^{P_i+Q_j}$$

Model

- Each worker performs one coded multiplication of the form $\tilde{A}(x) \cdot \tilde{B}(x)$.
- The n matrix multiplications are encoded into $\tilde{A}(x)$ and $\tilde{B}(x)$ as

$$\tilde{A}(x) = \sum_{i=1}^{n} A_{M_i} x^{P_i}, \text{ and } \tilde{B}(x) = \sum_{i=1}^{n} B_{N_i} x^{Q_i},$$

  achieving the optimal encoding complexity based on polynomials.
- Therefore, a coded task can be generated as

$$\tilde{C}(x) = \tilde{A}(x)\tilde{B}(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{M_i} B_{N_j} x^{P_i+Q_j}$$

- Minimize the degree of $\tilde{C}(x)$ by carefully choosing values of M_i, N_i, P_i, Q_i, $i = 1, \ldots, n$.

Toy Example

- Suppose that there are two matrix multiplications $A_1 B_1$ and $A_2 B_2$ in parallel.

Toy Example

- Suppose that there are two matrix multiplications $A_1 B_1$ and $A_2 B_2$ in parallel.
- A possible way to add coded tasks for the two jobs is to embed the two matrix multiplications into one larger job as:

$$\hat{A} \cdot \hat{B} \triangleq \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_2 \end{bmatrix} = \begin{bmatrix} A_1 B_1 & A_1 B_2 \\ A_1 B_2 & A_2 B_2 \end{bmatrix}.$$

Toy Example

- Suppose that there are two matrix multiplications $A_1 B_1$ and $A_2 B_2$ in parallel.
- A possible way to add coded tasks for the two jobs is to embed the two matrix multiplications into one larger job as:

$$\hat{A} \cdot \hat{B} \triangleq \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_2 \end{bmatrix} = \begin{bmatrix} A_1 B_1 & A_1 B_2 \\ A_1 B_2 & A_2 B_2 \end{bmatrix}.$$

- To generate coded tasks, we can apply polynomial codes by encoding $\hat{A}$ and $\hat{B}$ as

$$\tilde{A}(x) = A_1 x^0 + A_2 x^2, \text{ and } \tilde{B}(x) = B_1 x^0 + B_2 x^1,$$

Toy Example

- Suppose that there are two matrix multiplications $A_1 B_1$ and $A_2 B_2$ in parallel.
- A possible way to add coded tasks for the two jobs is to embed the two matrix multiplications into one larger job as:

$$\hat{A} \cdot \hat{B} \triangleq \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_2 \end{bmatrix} = \begin{bmatrix} A_1 B_1 & A_1 B_2 \\ A_1 B_2 & A_2 B_2 \end{bmatrix}.$$

- To generate coded tasks, we can apply polynomial codes by encoding $\hat{A}$ and $\hat{B}$ as

$$\tilde{A}(x) = A_1 x^0 + A_2 x^2, \text{ and } \tilde{B}(x) = B_1 x^0 + B_2 x^1,$$

- A worker then multiply $\hat{A}(x)$ and $\hat{B}(x)$:

$$\tilde{C}(x) = A_1 B_1 x^0 + A_1 B_2 x^1 + A_2 B_1 x^2 + A_2 B_2 x^3,$$

which can be decoded from any 4 tasks with different values of x.

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	P_1+Q_1	P_1+Q_2
A_{M_2}	P_2+Q_1	P_2+Q_2

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{2,1\}$
$P=\{0,1\}, Q=\{0,2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,1\}$

(d) a feasible coding scheme with the optimal degree

- The previous scheme is suboptimal because $\tilde{C}(x)$ can be decoded into four matrix multiplications:

x^0	x^1	x^2	x^3
A_1B_1	A_1B_2	A_2B_1	A_2B_2

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	P_1+Q_1	P_1+Q_2
A_{M_2}	P_2+Q_1	P_2+Q_2

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{2,1\}$
$P=\{0,1\}, Q=\{0,2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,1\}$

(d) a feasible coding scheme with the optimal degree

- The previous scheme is suboptimal because $\tilde{C}(x)$ can be decoded into four matrix multiplications:

x^0	x^1	x^2	x^3
A_1B_1	A_1B_2	A_2B_1	A_2B_2

- However we only need A_1B_1 and A_2B_2.

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	P_1+Q_1	P_1+Q_2
A_{M_2}	P_2+Q_1	P_2+Q_2

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{2,1\}$
$P=\{0,1\}, Q=\{0,2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,1\}$

(d) a feasible coding scheme with the optimal degree

- The previous scheme is suboptimal because $\tilde{C}(x)$ can be decoded into four matrix multiplications:

x^0	x^1	x^2	x^3
A_1B_1	A_1B_2	A_2B_1	A_2B_2

- However we only need A_1B_1 and A_2B_2.
- The tasks A_1B_2 and A_2B_1 are noise coefficients in $\tilde{C}(x)$.

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	P_1+Q_1	P_1+Q_2
A_{M_2}	P_2+Q_1	P_2+Q_2

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{2,1\}$
$P=\{0,1\}, Q=\{0,2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,1\}$

(d) a feasible coding scheme with the optimal degree

- The previous scheme is suboptimal because $\tilde{C}(x)$ can be decoded into four matrix multiplications:

x^0	x^1	x^2	x^3
A_1B_1	A_1B_2	A_2B_1	A_2B_2

- However we only need A_1B_1 and A_2B_2.
- The tasks A_1B_2 and A_2B_1 are noise coefficients in $\tilde{C}(x)$.
- To minimize noise coefficients, we use the figure in (a) to represent the exponents of all coefficients.

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	$P_1 + Q_1$	$P_1 + Q_2$
A_{M_2}	$P_2 + Q_1$	$P_2 + Q_2$

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M = \{1, 2\}, N = \{1, 2\}$
$P = \{0, 1\}, Q = \{0, 2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M = \{1, 2\}, N = \{2, 1\}$
$P = \{0, 1\}, Q = \{0, 2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M = \{1, 2\}, N = \{1, 2\}$
$P = \{0, 1\}, Q = \{0, 1\}$

(d) a feasible coding scheme with the optimal degree

- The sub-optimal solution from the previous example corresponds to a degree table in (b).

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	$P_1 + Q_1$	$P_1 + Q_2$
A_{M_2}	$P_2 + Q_1$	$P_2 + Q_2$

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M = \{1, 2\}, N = \{1, 2\}$
$P = \{0, 1\}, Q = \{0, 2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M = \{1, 2\}, N = \{2, 1\}$
$P = \{0, 1\}, Q = \{0, 2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M = \{1, 2\}, N = \{1, 2\}$
$P = \{0, 1\}, Q = \{0, 1\}$

(d) a feasible coding scheme with the optimal degree

- The sub-optimal solution from the previous example corresponds to a degree table in (b).
- The optimal coding scheme is shown in (d), corresponding to:

x^0	x^1	x^2
$A_1 B_1$	$A_1 B_2 + A_2 B_1$	$A_2 B_2$

Why is it sub-optimal?

	B_{N_1}	B_{N_2}
A_{M_1}	P_1+Q_1	P_1+Q_2
A_{M_2}	P_2+Q_1	P_2+Q_2

(a) a general illustration of the coding scheme

	B_1	B_2
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,2\}$

(b) a coding scheme based on the polynomial code

	B_2	B_1
A_1	0	2
A_2	1	3

$M=\{1,2\}, N=\{2,1\}$
$P=\{0,1\}, Q=\{0,2\}$

(c) another feasible coding scheme

	B_1	B_2
A_1	0	1
A_2	1	2

$M=\{1,2\}, N=\{1,2\}$
$P=\{0,1\}, Q=\{0,1\}$

(d) a feasible coding scheme with the optimal degree

- Another sub-optimal scheme in (c):

x^0	x^1	x^2	x^3
A_1B_2	A_2B_2	A_1B_1	A_1B_2

- The optimal scheme in (d):

x^0	x^1	x^2
A_1B_1	$A_1B_2+A_2B_1$	A_2B_2

Algorithm

- The algorithm design breaks down into two steps

Algorithm

- The algorithm design breaks down into two steps

> The 1st step is: “What is the optimal placement of the highlighted entries in the degree table?”

Algorithm

- The algorithm design breaks down into two steps

> The 1st step is: “What is the optimal placement of the highlighted entries in the degree table?”

  - This is essentially the non-attacking rooks placement that minimizes the degree.

Algorithm

- The algorithm design breaks down into two steps

> The 1st step is: "What is the optimal placement of the highlighted entries in the degree table?"

  - This is essentially the non-attacking rooks placement that minimizes the degree.

> The 2nd step is: "Given a degree table with highlighted entries (*i.e.*, non-attacking rook placement) what is the optimal degree placement?"

Algorithm

- The algorithm design breaks down into two steps

> The 1st step is: "What is the optimal placement of the highlighted entries in the degree table?"

  - This is essentially the non-attacking rooks placement that minimizes the degree.

> The 2nd step is: "Given a degree table with highlighted entries (*i.e.*, non-attacking rook placement) what is the optimal degree placement?"

- Surprisingly it is easier to attack the second problem first and then find the solution to the first problem.

Algorithm Design

- The placement of the highlighted entries (*i.e.*, rook placement) corresponds to the sequences $(M_1, ..., M_n)$ and $(N_1, ..., N_n)$.

Algorithm Design

- The placement of the highlighted entries (*i.e.*, rook placement) corresponds to the sequences $(M_1, ..., M_n)$ and $(N_1, ..., N_n)$.
- The placement of the degrees in the table corresponds to the sequences $(P_1, ..., P_n)$ and $(Q_1, ..., Q_n)$.

Algorithm Design

- The placement of the highlighted entries (*i.e.*, rook placement) corresponds to the sequences $(M_1, ..., M_n)$ and $(N_1, ..., N_n)$.
- The placement of the degrees in the table corresponds to the sequences $(P_1, ..., P_n)$ and $(Q_1, ..., Q_n)$.

$$\tilde{A}(x) = \sum_{i=1}^{n} A_{M_i} x^{P_i} \qquad \tilde{B}(x) = \sum_{i=1}^{n} B_{N_i} x^{Q_i}$$
$$\tilde{C}(x) = \tilde{A}(x)\tilde{B}(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{M_i} B_{N_j} x^{P_i+Q_j}$$

Algorithm Design

- The placement of the highlighted entries (*i.e.*, rook placement) corresponds to the sequences $(M_1, ..., M_n)$ and $(N_1, ..., N_n)$.
- The placement of the degrees in the table corresponds to the sequences $(P_1, ..., P_n)$ and $(Q_1, ..., Q_n)$.

$$\tilde{A}(x) = \sum_{i=1}^{n} A_{M_i} x^{P_i} \qquad \tilde{B}(x) = \sum_{i=1}^{n} B_{N_i} x^{Q_i}$$
$$\tilde{C}(x) = \tilde{A}(x)\tilde{B}(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{M_i} B_{N_j} x^{P_i+Q_j}$$

- We aim to minimize the the degree of the polynomial $\tilde{C}(x)$, *i.e.*, $\max_{i,j}\{P_i + Q_j\}$, but still having $A_i B_i$, $i = 1, \ldots, n$, decodable.

Optimal Placement of the Degrees

	B_3	B_1	B_2	B_4
A_1	0	4	7	9
A_2	1	5	8	10
A_3	2	6	9	11
A_4	3	7	10	12

(a) Placement 1

	B_1	B_4	B_2	B_3
A_1	0	1	5	7
A_2	1	2	6	8
A_3	2	3	7	9
A_4	3	4	8	10

(b) Placement 2

- The idea is to maximize the overlap of (noise) entries between two adjacent columns.

Optimal Placement of the Degrees

	B_3	B_1	B_2	B_4
A_1	0	4	7	9
A_2	1	5	8	10
A_3	2	6	9	11
A_4	3	7	10	12

(a) Placement 1

	B_1	B_4	B_2	B_3
A_1	0	1	5	7
A_2	1	2	6	8
A_3	2	3	7	9
A_4	3	4	8	10

(b) Placement 2

- The idea is to maximize the overlap of (noise) entries between two adjacent columns.
- The degree placements (not the rooks placement!) is optimal for the two examples to the right.

Optimal Placement of the Degrees

	B_3	B_1	B_2	B_4
A_1	0	4	7	9
A_2	1	5	8	10
A_3	2	6	9	11
A_4	3	7	10	12

(a) Placement 1

	B_1	B_4	B_2	B_3
A_1	0	1	5	7
A_2	1	2	6	8
A_3	2	3	7	9
A_4	3	4	8	10

(b) Placement 2

- The idea is to maximize the overlap of (noise) entries between two adjacent columns.
- The degree placements (not the rooks placement!) is optimal for the two examples to the right.
- By the definition of polynomial multiplication, the numbers are forced to increase up in each column.

Optimal Placement of the Degrees

	B_3	B_1	B_2	B_4
A_1	0	4	7	9
A_2	1	5	8	10
A_3	2	6	9	11
A_4	3	7	10	12

(a) Placement 1

	B_1	B_4	B_2	B_3
A_1	0	1	5	7
A_2	1	2	6	8
A_3	2	3	7	9
A_4	3	4	8	10

(b) Placement 2

- Placement 2 is optimal because the two overlaps $(1, 2, 3)$ and $(7, 8)$ are maximal. Furthermore the middle two columns are incapable of having overlap

Optimal Placement of the Degrees

	B_3	B_1	B_2	B_4
A_1	0	4	7	9
A_2	1	5	8	10
A_3	2	6	9	11
A_4	3	7	10	12

(a) Placement 1

	B_1	B_4	B_2	B_3
A_1	0	1	5	7
A_2	1	2	6	8
A_3	2	3	7	9
A_4	3	4	8	10

(b) Placement 2

- Placement 2 is optimal because the two overlaps $(1, 2, 3)$ and $(7, 8)$ are maximal. Furthermore the middle two columns are incapable of having overlap
- Similarly, Placement 1 is optimal because the two overlaps (7) and $(9, 10)$ are maximal. Furthermore the first two columns are incapable of having overlap

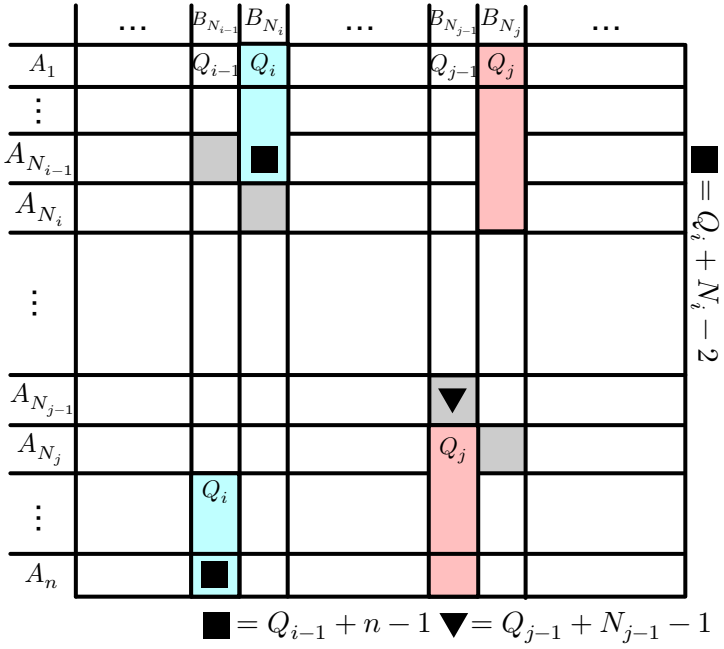
Algorithm for the First Step

Algorithm 1 Find Optimal Degree Q

Input: $N = (N_1, ..., N_n)$

Output: $Q = (Q_1, ..., Q_n)$

1: $Q_1 = 0$
2: **for** $i \leftarrow 2$ to n **do**
3: **if** $n - N_{i-1} \leq N_i - 1$ **then**
4: $Q_i \leftarrow Q_{i-1} + N_{i-1}$
5: **else**
6: $Q_i \leftarrow Q_{i-1} + n - N_i + 1$
7: **end if**
8: **end for**

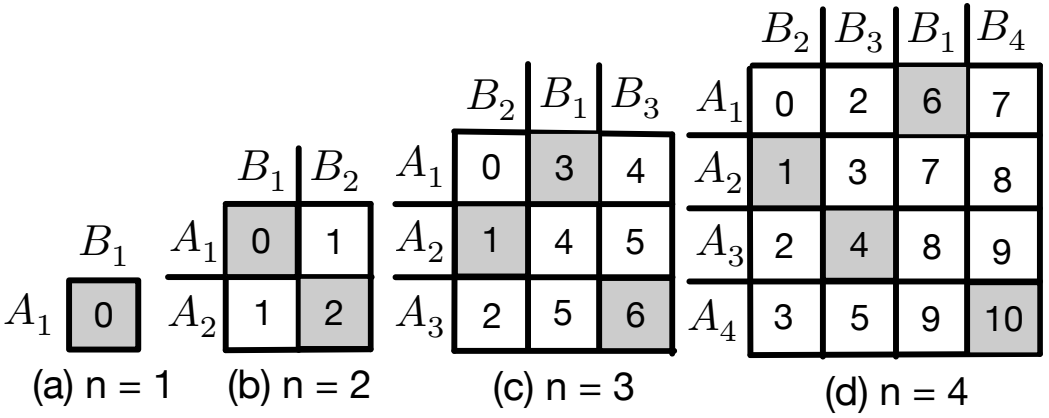


- All degree placements must satisfy:

$$Q_i \geq Q_{i-1} + \max\{N_{i-1}, n - N_i + 1\}$$

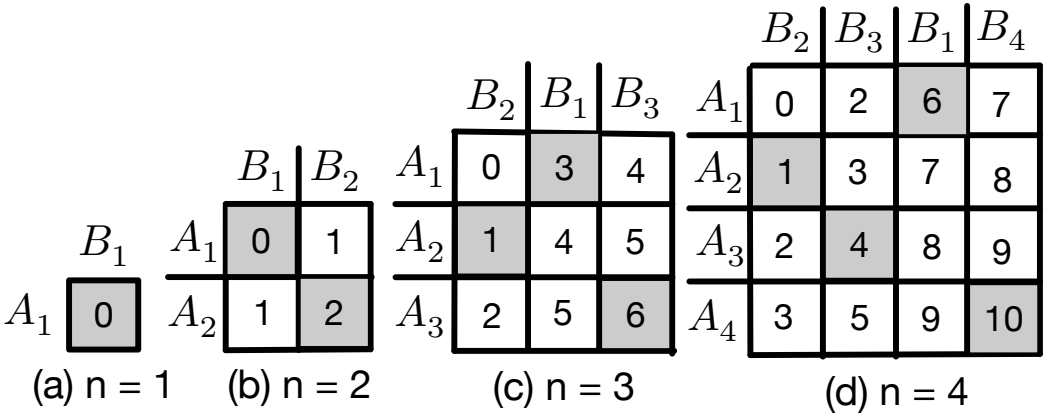
- Our algorithm is optimal with respect to this constraint

Optimal Rook Placement



- The optimal rook placements are constructed recursively.

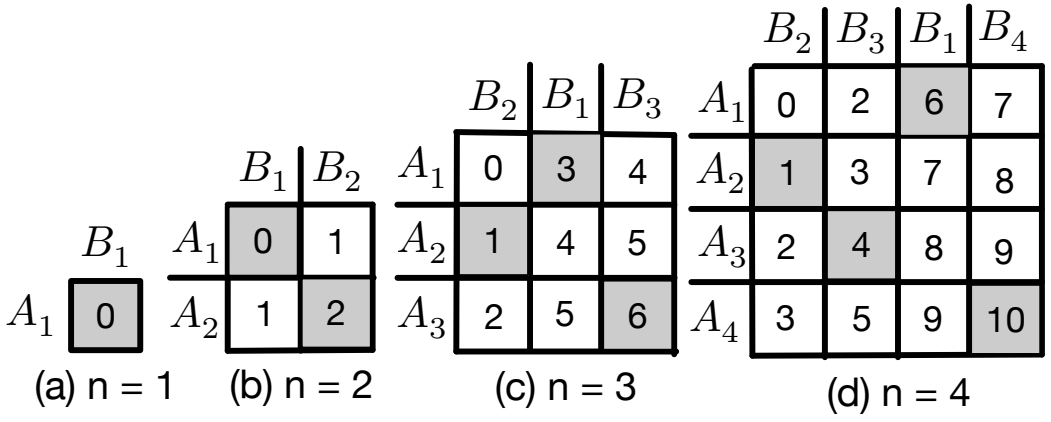
Optimal Rook Placement



(a) n = 1 (b) n = 2 (c) n = 3 (d) n = 4

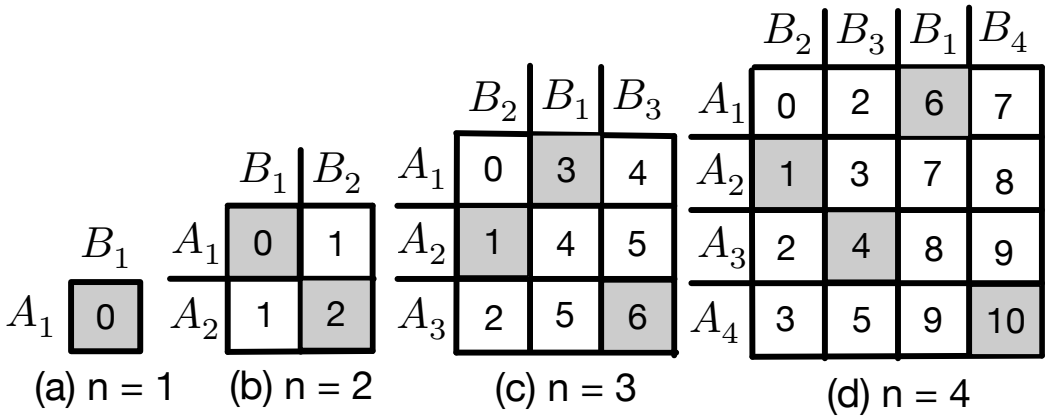
- The optimal rook placements are constructed recursively.
- The base case(s) are shown in the (a) and (b).

Optimal Rook Placement



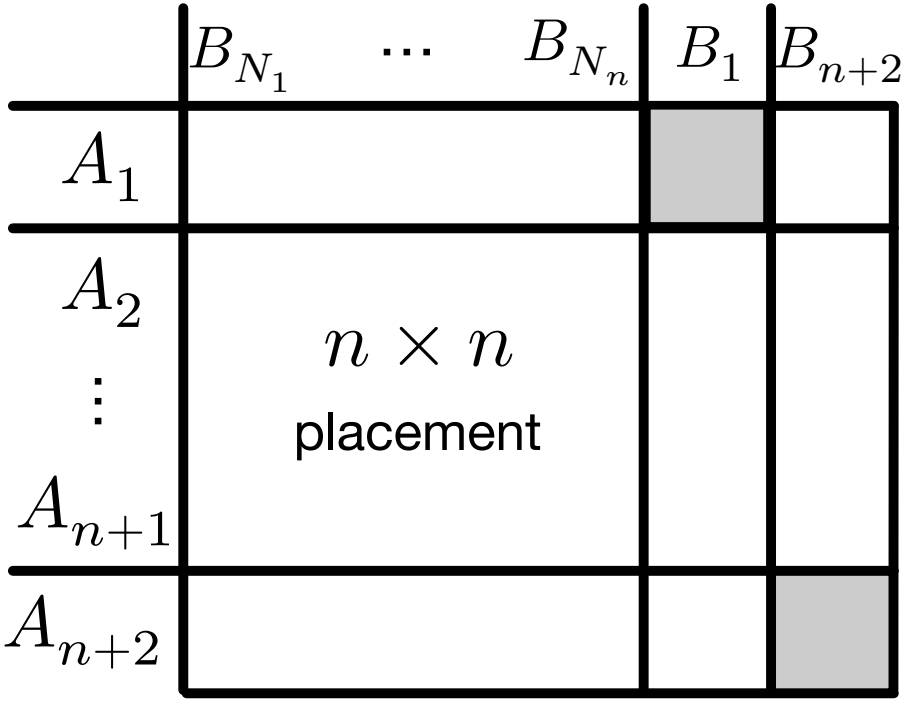
- The optimal rook placements are constructed recursively.
- The base case(s) are shown in the (a) and (b).
- The rook placement for $n = 3$ is constructed from the placement for $n = 1$.

Optimal Rook Placement



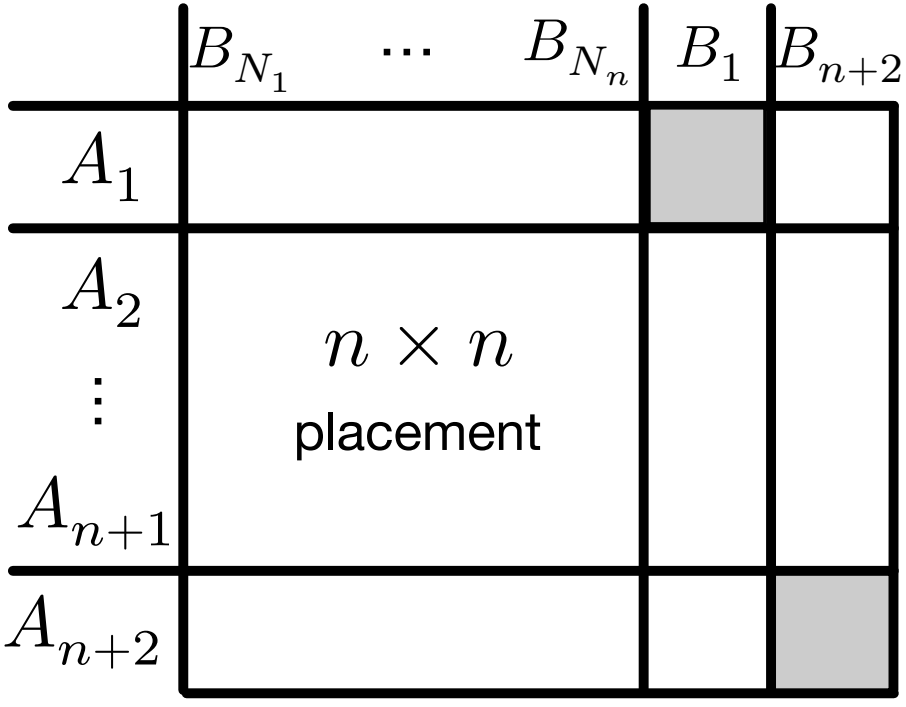
- The optimal rook placements are constructed recursively.
- The base case(s) are shown in the (a) and (b).
- The rook placement for $n = 3$ is constructed from the placement for $n = 1$.
- Likewise the placement for $n = 4$ is constructed from the placement for $n = 2$.

Optimal Rook Placement



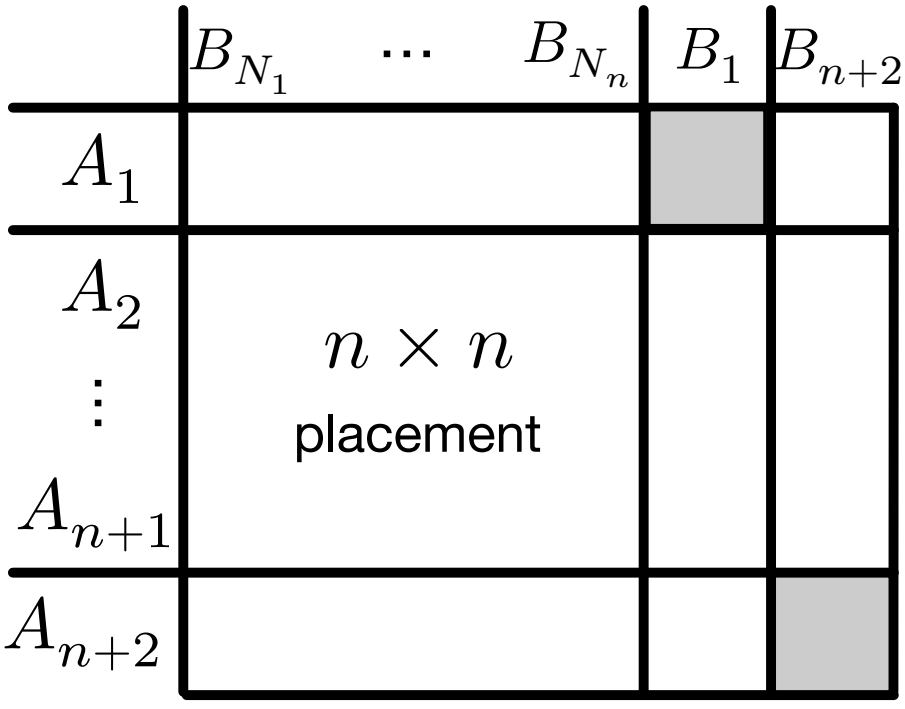
- Using the placements for $n \times n$ we can make a placement for $(n+2) \times (n+2)$

Optimal Rook Placement



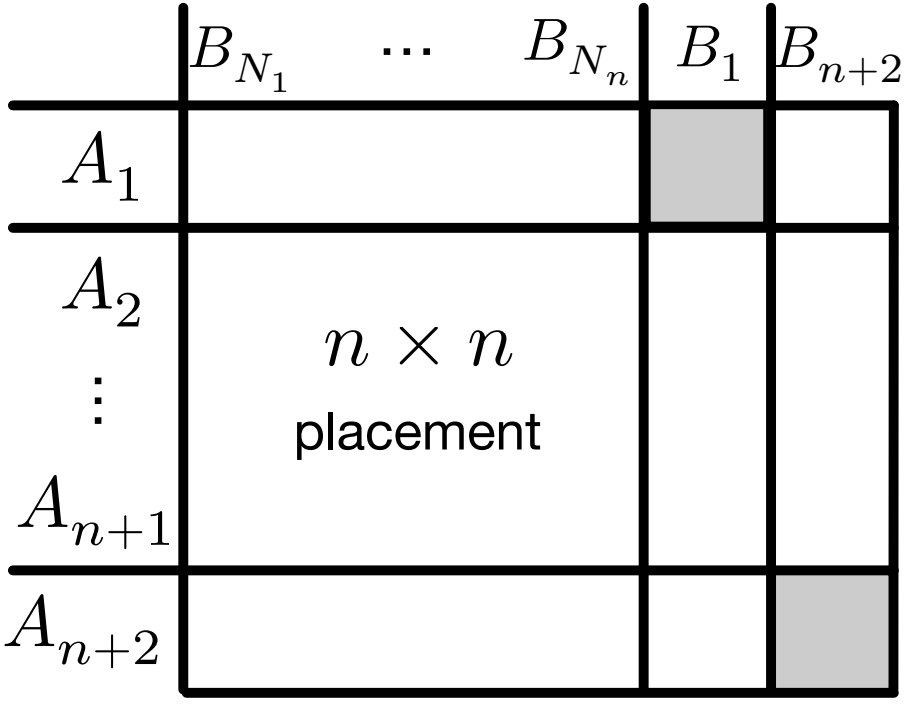
- Using the placements for $n \times n$ we can make a placement for $(n+2) \times (n+2)$
- The proof of optimality is given by considering any other placement and considering the highest and lowest" column entries

Optimal Rook Placement



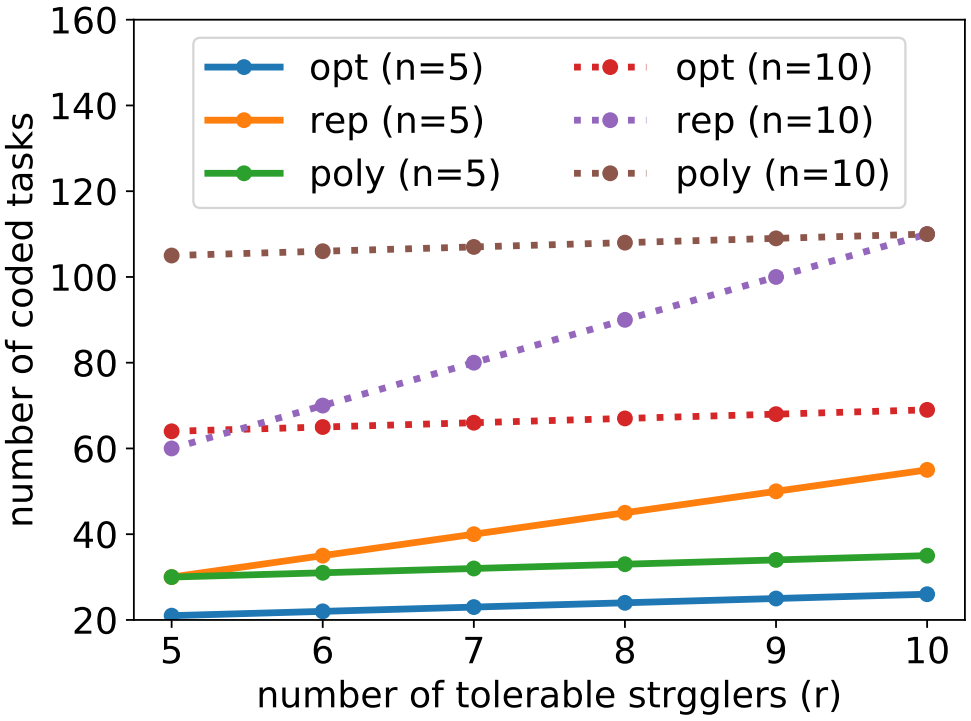
- Using the placements for $n \times n$ we can make a placement for $(n+2) \times (n+2)$
- The proof of optimality is given by considering any other placement and considering the highest and lowest" column entries
- We remove the corresponding rows and columns and analyze the amount of overlap in the submatrix left-over

Optimal Rook Placement



- Using the placements for $n \times n$ we can make a placement for $(n+2) \times (n+2)$
- The proof of optimality is given by considering any other placement and considering the highest and lowest" column entries
- We remove the corresponding rows and columns and analyze the amount of overlap in the submatrix left-over
- We also factor in the difference of overlap given by a suboptimal "highest" and "lowest" rook placement

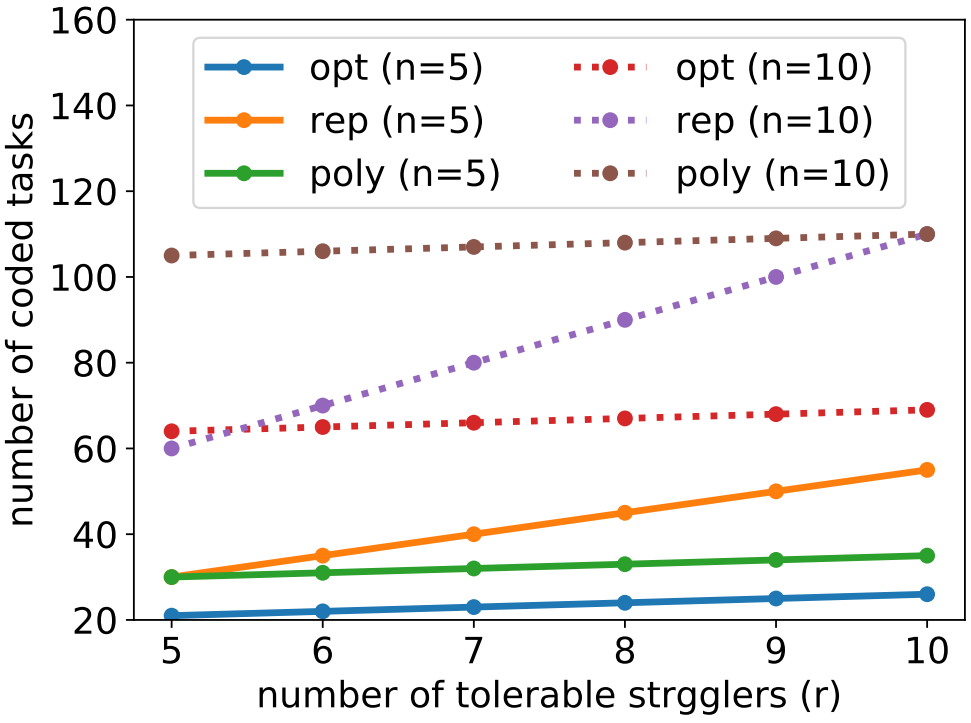
Evaluation



- Our coding scheme has a recovery threshold of:

$$R = \left\lfloor \frac{(n+1)^2 - 4}{2} \right\rfloor$$

Evaluation



- Our coding scheme has a recovery threshold of:

$$R = \left\lfloor \frac{(n+1)^2 - 4}{2} \right\rfloor$$

- Polynomial codes have a recovery threshold of:

$$R_{ep} = n^2$$

Comparison

Assume that the size of A_i is $x \times y$ and the size of B_i is $y \times z$, $i = 1, \ldots, n$.

Comparison

Assume that the size of A_i is $x \times y$ and the size of B_i is $y \times z$, $i = 1, \ldots, n$.

	encoding complexity	task complexity	recovery threshold
this paper	$O(n(xy + yz))$	$O(xyz)$	$O(n^2)$
Lagragian [1,2]	$O(n \log^2 n \log\log n(xy + yz))$	$O(xyz)$	$O(n)$

[1] Q. Yu, *et al.*, "Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy," in Workshop on Systems for ML and Open Source Software at NeurIPS 2018.

[2] Z. Jia, *et al.*, "Cross Subspace Alignment Codes for Coded Distributed Batch Computation," arXiv:1909.13873.

Comparison

Assume that the size of A_i is $x \times y$ and the size of B_i is $y \times z$, $i = 1, \ldots, n$.

	encoding complexity	task complexity	recovery threshold
this paper	$O(n(xy + yz))$	$O(xyz)$	$O(n^2)$
Lagragian [1,2]	$O(n \log^2 n \log\log n(xy + yz))$	$O(xyz)$	$O(n)$

[1] Q. Yu, *et al.*, "Lagrange Coded Computing: Optimal Design for Resiliency, Security, and Privacy," in Workshop on Systems for ML and Open Source Software at NeurIPS 2018.

[2] Z. Jia, *et al.*, "Cross Subspace Alignment Codes for Coded Distributed Batch Computation," arXiv:1909.13873.

Our coding scheme achieves the optimal encoding complexity among all polynomial-based coding schemes, with a higher recovery threshold.

Conclusions

- We propose a coding framework that can create coded tasks for concurrent matrix multiplications based on polynomial coding.
- Our coding scheme achieves the optimal encoding complexity among all polynomial-based coding schemes.
- We also minimize the degree of the polynomial to minimize the number of tasks required for decoding.