Coded Computation: Straggler Mitigation in Distributed Matrix Multiplication[1]

Ankit Kumar Misra   Dhruva Dhingra

EE 605: Error Correcting Codes,
Autumn 2022, IIT Bombay

December 5, 2022

[1] Yu et al., IEEE Transactions on Information Theory, 2020

Distributed Matrix Multiplication

- Matrix multiplication is a fundamental operation in data analytics and machine learning applications.

Distributed Matrix Multiplication

- Matrix multiplication is a fundamental operation in data analytics and machine learning applications.
- Often requires a lot more storage and computational power than a single machine can offer.

Distributed Matrix Multiplication

- Matrix multiplication is a fundamental operation in data analytics and machine learning applications.
- Often requires a lot more storage and computational power than a single machine can offer.
- This problem is solved by deploying the multiplication task over a large-scale distributed system, having several nodes.

The Straggler's Delay Bottleneck

- But what if some nodes are slower than others?

The Straggler's Delay Bottleneck

- But what if some nodes are slower than others?
- The slowest nodes, a.k.a. *stragglers*, impose a latency bottleneck.

The Straggler's Delay Bottleneck

- But what if some nodes are slower than others?
- The slowest nodes, a.k.a. *stragglers*, impose a latency bottleneck.
- Commonly tackled by adding redundant computations.

The Straggler's Delay Bottleneck

- But what if some nodes are slower than others?
- The slowest nodes, a.k.a. *stragglers*, impose a latency bottleneck.
- Commonly tackled by adding redundant computations.
- Naturally, error correcting codes can be applied to introduce 'efficient redundancy' in computation.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.
- Encoding functions $\mathbf{f} = (f_0, \ldots, f_{N-1})$ and $\mathbf{g} = (g_0, \ldots, g_{N-1})$, and class of decoding functions $\mathbf{d} = \{d_{\mathcal{K}}\}_{\mathcal{K} \subseteq \{0,1,\ldots,N-1\}}$.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.
- Encoding functions $\mathbf{f} = (f_0, \ldots, f_{N-1})$ and $\mathbf{g} = (g_0, \ldots, g_{N-1})$, and class of decoding functions $\mathbf{d} = \{d_{\mathcal{K}}\}_{\mathcal{K} \subseteq \{0,1,\ldots,N-1\}}$.
- Master sends $\tilde{A}_i = f_i(A)$ and $\tilde{B}_i = g_i(B)$ to worker i.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.
- Encoding functions $\mathbf{f} = (f_0, \ldots, f_{N-1})$ and $\mathbf{g} = (g_0, \ldots, g_{N-1})$, and class of decoding functions $\mathbf{d} = \{d_{\mathcal{K}}\}_{\mathcal{K} \subseteq \{0,1,\ldots,N-1\}}$.
- Master sends $\tilde{A}_i = f_i(A)$ and $\tilde{B}_i = g_i(B)$ to worker i.
- Worker i sends $\tilde{C}_i = \tilde{A}_i^T \tilde{B}_i$ to master.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s\times r}$ and $B \in \mathbb{F}^{s\times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.
- Encoding functions $\mathbf{f} = (f_0, \ldots, f_{N-1})$ and $\mathbf{g} = (g_0, \ldots, g_{N-1})$, and class of decoding functions $\mathbf{d} = \{d_{\mathcal{K}}\}_{\mathcal{K}\subseteq\{0,1,\ldots,N-1\}}$.
- Master sends $\tilde{A}_i = f_i(A)$ and $\tilde{B}_i = g_i(B)$ to worker i.
- Worker i sends $\tilde{C}_i = \tilde{A}_i^T \tilde{B}_i$ to master.
- Master estimates C using $\hat{C} = d_{\mathcal{K}}(\{\tilde{C}_i\}_{i\in\mathcal{K}})$ using $\tilde{C}_i$ values received from a subset $\mathcal{K}$ of workers.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s\times r}$ and $B \in \mathbb{F}^{s\times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.
- Encoding functions $\mathbf{f} = (f_0, \ldots, f_{N-1})$ and $\mathbf{g} = (g_0, \ldots, g_{N-1})$, and class of decoding functions $\mathbf{d} = \{d_{\mathcal{K}}\}_{\mathcal{K}\subseteq\{0,1,\ldots,N-1\}}$.
- Master sends $\tilde{A}_i = f_i(A)$ and $\tilde{B}_i = g_i(B)$ to worker i.
- Worker i sends $\tilde{C}_i = \tilde{A}_i^T \tilde{B}_i$ to master.
- Master estimates C using $\hat{C} = d_{\mathcal{K}}(\{\tilde{C}_i\}_{i\in\mathcal{K}})$ using $\tilde{C}_i$ values received from a subset $\mathcal{K}$ of workers.
- k-recoverable if $\hat{C} = C$ for all $\mathcal{K}$ s.t. $|\mathcal{K}| = k$.

Modelling the Problem

- Input matrices $A \in \mathbb{F}^{s \times r}$ and $B \in \mathbb{F}^{s \times t}$.
- One master node and N worker nodes, each of which can store $1/pm$ fraction of A and $1/pn$ fraction of B.
- Encoding functions $\mathbf{f} = (f_0, \ldots, f_{N-1})$ and $\mathbf{g} = (g_0, \ldots, g_{N-1})$, and class of decoding functions $\mathbf{d} = \{d_{\mathcal{K}}\}_{\mathcal{K} \subseteq \{0,1,\ldots,N-1\}}$.
- Master sends $\tilde{A}_i = f_i(A)$ and $\tilde{B}_i = g_i(B)$ to worker i.
- Worker i sends $\tilde{C}_i = \tilde{A}_i^T \tilde{B}_i$ to master.
- Master estimates C using $\hat{C} = d_{\mathcal{K}}(\{\tilde{C}_i\}_{i \in \mathcal{K}})$ using $\tilde{C}_i$ values received from a subset $\mathcal{K}$ of workers.
- k-recoverable if $\hat{C} = C$ for all $\mathcal{K}$ s.t. $|\mathcal{K}| = k$.
- Recovery threshold $K(\mathbf{f}, \mathbf{g}, \mathbf{d})$ is smallest k s.t. k-recoverable.

Redundant Code

- Divide $A \in \mathbb{F}^{s \times r}$ into r_1 matrices of size $\mathbb{F}^{s \times r/r_1}$. Divide $B \in \mathbb{F}^{s \times t}$ into r_2 matrices of size $\mathbb{F}^{s \times t/r_2}$.

Redundant Code

- Divide $A \in \mathbb{F}^{s \times r}$ into r_1 matrices of size $\mathbb{F}^{s \times r/r_1}$. Divide $B \in \mathbb{F}^{s \times t}$ into r_2 matrices of size $\mathbb{F}^{s \times t/r_2}$.
- Give each worker the task of computing the product of one submatrix of A with another submatrix of B.

Redundant Code

- Divide $A \in \mathbb{F}^{s\times r}$ into r_1 matrices of size $\mathbb{F}^{s\times r/r_1}$. Divide $B \in \mathbb{F}^{s\times t}$ into r_2 matrices of size $\mathbb{F}^{s\times t/r_2}$.
- Give each worker the task of computing the product of one submatrix of A with another submatrix of B.
- There are N workers and $r_1 \times r_2$ unique computations. As soon as we have $r_1 \times r_2$ unique results of the form $A_i^T B_j$, we can interpolate the complete matrix.

Redundant Code

- Divide $A \in \mathbb{F}^{s \times r}$ into r_1 matrices of size $\mathbb{F}^{s \times r/r_1}$. Divide $B \in \mathbb{F}^{s \times t}$ into r_2 matrices of size $\mathbb{F}^{s \times t/r_2}$.
- Give each worker the task of computing the product of one submatrix of A with another submatrix of B.
- There are N workers and $r_1 \times r_2$ unique computations. As soon as we have $r_1 \times r_2$ unique results of the form $A_i^T B_j$, we can interpolate the complete matrix.
- Thus, the redundancy is $\frac{N}{r_1 r_2}$

Linear Codes

$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,m-1} \\ \vdots & \ddots & \vdots \\ A_{p-1,0} & \cdots & A_{p-1,m-1} \end{bmatrix}, \quad B = \begin{bmatrix} B_{0,0} & \cdots & B_{0,n-1} \\ \vdots & \ddots & \vdots \\ B_{p-1,0} & \cdots & B_{p-1,n-1} \end{bmatrix}$$

Linear Codes

$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,m-1} \\ \vdots & \ddots & \vdots \\ A_{p-1,0} & \cdots & A_{p-1,m-1} \end{bmatrix}, \quad B = \begin{bmatrix} B_{0,0} & \cdots & B_{0,n-1} \\ \vdots & \ddots & \vdots \\ B_{p-1,0} & \cdots & B_{p-1,n-1} \end{bmatrix}$$

$$\tilde{A}_i = \sum_{j,k} A_{j,k} a_{ijk} \qquad \tilde{B}_i = \sum_{j,k} B_{j,k} b_{ijk}$$

Linear Codes

$$A = \begin{bmatrix} A_{0,0} & \cdots & A_{0,m-1} \\ \vdots & \ddots & \vdots \\ A_{p-1,0} & \cdots & A_{p-1,m-1} \end{bmatrix}, \quad B = \begin{bmatrix} B_{0,0} & \cdots & B_{0,n-1} \\ \vdots & \ddots & \vdots \\ B_{p-1,0} & \cdots & B_{p-1,n-1} \end{bmatrix}$$

$$\tilde{A}_i = \sum_{j,k} A_{j,k} a_{ijk} \qquad \tilde{B}_i = \sum_{j,k} B_{j,k} b_{ijk}$$

$$\hat{C}_{j,k} = \sum_{i \in \mathcal{K}} \tilde{C}_i c_{ijk}$$

Entangled Polynomial Code

- Assign a distinct $x_i \in \mathbb{F}$ to each worker i.

Entangled Polynomial Code

- Assign a distinct $x_i \in \mathbb{F}$ to each worker i.
- Worker i is given the following:

$$\tilde{A}_i = \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} A_{j,k} x_i^{j+kp} \qquad \tilde{B}_i = \sum_{j=0}^{p-1} \sum_{k=0}^{n-1} B_{j,k} x_i^{p-1-j+kpm}$$

Entangled Polynomial Code

- Assign a distinct $x_i \in \mathbb{F}$ to each worker i.
- Worker i is given the following:

$$\tilde{A}_i = \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} A_{j,k} x_i^{j+kp} \qquad \tilde{B}_i = \sum_{j=0}^{p-1} \sum_{k=0}^{n-1} B_{j,k} x_i^{p-1-j+kpm}$$

- Worker i returns

$$\begin{aligned}\tilde{C}_i &= \tilde{A}_i^T \tilde{B}_i \\ &= \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} \sum_{j'=0}^{p-1} \sum_{k='0}^{n-1} A_{j,k}^T B_{j',k'} x_i^{(p-1+j-j')+kp+k'pm}\end{aligned}$$

Entangled Polynomial Code

- $\tilde{C}_i$ is interpolation of polynomial $h(x)$ at $x = x_i$ where

$$h(x) = \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} \sum_{j'=0}^{p-1} \sum_{k='0}^{n-1} A_{j,k}^T B_{j',k'} x^{(p-1+j-j')+kp+k'pm}$$

Entangled Polynomial Code

- $\tilde{C}_i$ is interpolation of polynomial $h(x)$ at $x = x_i$ where

$$h(x) = \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} \sum_{j'=0}^{p-1} \sum_{k='0}^{n-1} A_{j,k}^T B_{j',k'} x^{(p-1+j-j')+kp+k'pm}$$

- $C_{k,k'}$ is the coefficient of $(p-1+kp+k'pm)$-th degree term.

Entangled Polynomial Code

- $\tilde{C}_i$ is interpolation of polynomial $h(x)$ at $x = x_i$ where

$$h(x) = \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} \sum_{j'=0}^{p-1} \sum_{k='0}^{n-1} A_{j,k}^T B_{j',k'} x^{(p-1+j-j')+kp+k'pm}$$

- $C_{k,k'}$ is the coefficient of $(p-1+kp+k'pm)$-th degree term.
- Degree of $h(x)$ is $pmn+p-2$. So, given evaluation of $h(x)$ at any distinct $pmn+p-1$ points, we can find the coefficients of $h(x)$ which are the required submatrices of the result.

Entangled Polynomial Code

- $\tilde{C}_i$ is interpolation of polynomial $h(x)$ at $x = x_i$ where

$$h(x) = \sum_{j=0}^{p-1} \sum_{k=0}^{m-1} \sum_{j'=0}^{p-1} \sum_{k='0}^{n-1} A_{j,k}^T B_{j',k'} x^{(p-1+j-j')+kp+k'pm}$$

- $C_{k,k'}$ is the coefficient of $(p - 1 + kp + k'pm)$-th degree term.
- Degree of $h(x)$ is $pmn + p - 2$. So, given evaluation of $h(x)$ at any distinct $pmn + p - 1$ points, we can find the coefficients of $h(x)$ which are the required submatrices of the result.

Theorem

$K_{entangled\text{-}poly} = pmn + p - 1.$

Why are Entangled Polynomial Codes Important?

Theorem

$K^*_{linear} = K_{entangled\text{-}poly}$.

Why are Entangled Polynomial Codes Important?

Theorem

$K^*_{linear} = K_{entangled\text{-}poly}$.

Theorem

If $\mathbb{F}$ *is a finite field, then* $\frac{1}{2}K_{entangled\text{-}poly} < K^* \leq K_{entangled\text{-}poly}$.

What we did

- We implemented entangled polynomial codes and redundant codes in Python.

What we did

- We implemented entangled polynomial codes and redundant codes in Python.
- Code at `https://github.com/ankitkmisra/Straggler-Mitigation-MatMul`.

What we did

- We implemented entangled polynomial codes and redundant codes in Python.
- Code at `https://github.com/ankitkmisra/Straggler-Mitigation-MatMul`.
- We simulated communication delays with two exponential distributions; a low-mean one for fast workers and a high-mean one for slow workers. $(p(x;\lambda) = \lambda e^{-\lambda x})$

What we did

- We implemented entangled polynomial codes and redundant codes in Python.
- Code at `https://github.com/ankitkmisra/Straggler-Mitigation-MatMul`.
- We simulated communication delays with two exponential distributions; a low-mean one for fast workers and a high-mean one for slow workers. ($p(x; \lambda) = \lambda e^{-\lambda x}$)
- A fraction $f < 1$ of workers were set to be slow.

What we did

- We implemented entangled polynomial codes and redundant codes in Python.
- Code at `https://github.com/ankitkmisra/Straggler-Mitigation-MatMul`.
- We simulated communication delays with two exponential distributions; a low-mean one for fast workers and a high-mean one for slow workers. ($p(x; \lambda) = \lambda e^{-\lambda x}$)
- A fraction $f < 1$ of workers were set to be slow.
- We observed computation times and errors in the final result, for both types of codes, by varying matrix sizes, number of partitions, number of workers, etc.

Practical Issues with Entangled Polynomial Code - Numerical Instability

- **Interpolation Error** - Lagrange Interpolation is numerically very unstable. As soon as interpolation degree crosses 20, error starts rising very quickly (Runge's phenomenon[2]).

[2] https://en.wikipedia.org/wiki/Runge's_phenomenon

[3] https://en.wikipedia.org/wiki/Chebyshev_nodes

Practical Issues with Entangled Polynomial Code - Numerical Instability

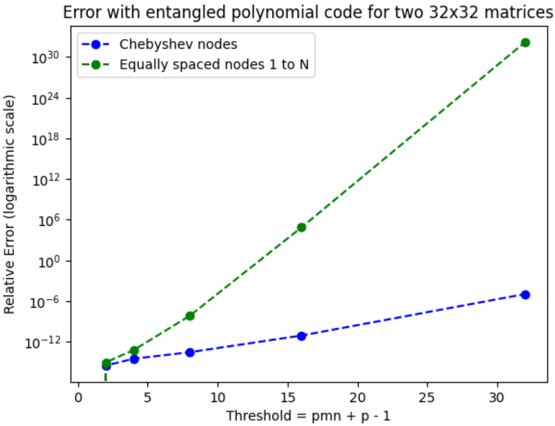
- **Interpolation Error** - Lagrange Interpolation is numerically very unstable. As soon as interpolation degree crosses 20, error starts rising very quickly (Runge's phenomenon[2]).
- **Solution** - Use Chebyshev Nodes[3] for evaluation points.

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right)$$

[2] https://en.wikipedia.org/wiki/Runge's_phenomenon

[3] https://en.wikipedia.org/wiki/Chebyshev_nodes

Practical Issues with Entangled Polynomial Code - Numerical Instability



Practical Issues with Entangled Polynomial Code - Preprocessing Time

- Encoding and Decoding for Entangled Polynomial Code is very expensive.

Practical Issues with Entangled Polynomial Code - Preprocessing Time

- Encoding and Decoding for Entangled Polynomial Code is very expensive.
- While multiplying a matrix with a scalar and summing over the block matrices is not expensive, calculating $x_i^{\mathbf{J}+\mathbf{K}p}$ for $\tilde{A}_i$ and $x_i^{p-1-\mathbf{J}+\mathbf{K}pm}$ for $\tilde{B}_i$ is a very expensive operation.

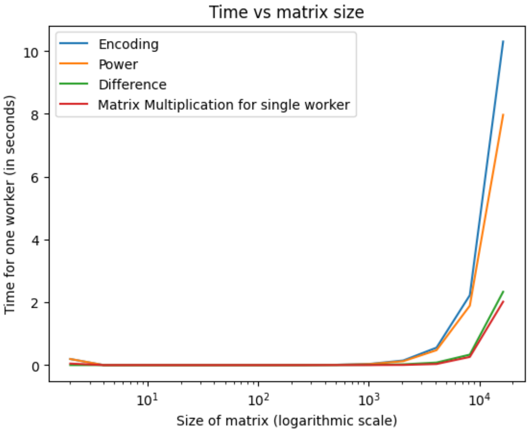
Practical Issues with Entangled Polynomial Code - Preprocessing Time

- Encoding and Decoding for Entangled Polynomial Code is very expensive.
- While multiplying a matrix with a scalar and summing over the block matrices is not expensive, calculating $x_i^{\mathbf{J}+\mathbf{K}p}$ for $\tilde{A}_i$ and $x_i^{p-1-\mathbf{J}+\mathbf{K}pm}$ for $\tilde{B}_i$ is a very expensive operation.
- In fact, this preprocessing time (even for a single worker) soon begins to eclipse the time taken for matrix multiplcation at a single worker.

Practical Issues with Entangled Polynomial Code - Preprocessing Time

- Encoding and Decoding for Entangled Polynomial Code is very expensive.
- While multiplying a matrix with a scalar and summing over the block matrices is not expensive, calculating $x_i^{\mathbf{J}+\mathbf{K}p}$ for $\tilde{A}_i$ and $x_i^{p-1-\mathbf{J}+\mathbf{K}pm}$ for $\tilde{B}_i$ is a very expensive operation.
- In fact, this preprocessing time (even for a single worker) soon begins to eclipse the time taken for matrix multiplcation at a single worker.
- Empirical analysis validates this result.

Practical Issues with Entangled Polynomial Code - Preprocessing Time



Execution times: Polynomial code vs. Redundant code

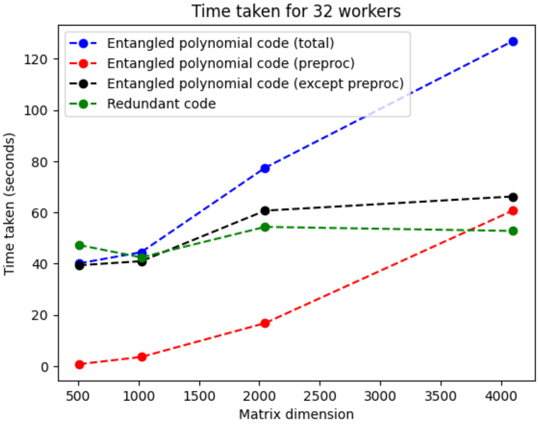


Figure: Caption

Conclusions

- Entangled polynomial codes are extremely effective for reconstruction from a minimal number of responsive workers.

Conclusions

- Entangled polynomial codes are extremely effective for reconstruction from a minimal number of responsive workers.
- But there are two problems:

Conclusions

- Entangled polynomial codes are extremely effective for reconstruction from a minimal number of responsive workers.
- But there are two problems:
  - Interpolation introduces errors, which increase exponentially with the number of evaluation points used.

Conclusions

- Entangled polynomial codes are extremely effective for reconstruction from a minimal number of responsive workers.
- But there are two problems:
  - Interpolation introduces errors, which increase exponentially with the number of evaluation points used.
  - Cost of preprocessing increases rapidly with matrix size, due to a large number of exponentiations, and causes a sharp increase in total computation time using entangled polynomial codes.

Conclusions

- Entangled polynomial codes are extremely effective for reconstruction from a minimal number of responsive workers.
- But there are two problems:
  - Interpolation introduces errors, which increase exponentially with the number of evaluation points used.
  - Cost of preprocessing increases rapidly with matrix size, due to a large number of exponentiations, and causes a sharp increase in total computation time using entangled polynomial codes.
- This renders entangled polynomial codes unfit for practical use, unless these issues are fixed.

Thanks!

Any questions?