Dual Entangled Polynomial Code: Three-Dimensional Coding for Distributed Matrix Multiplication

Pedro Soto, Jun Li, and Xiaodi Fan

School of Computing and Information Sciences, Florida International University

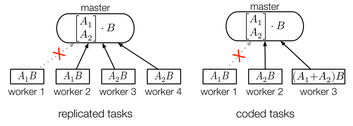
FIU Engineering & Computing
School of Computing & Information Sciences

Large-scale Matrix Multiplication

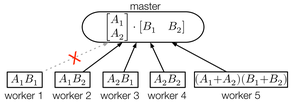
- Matrix multiplication is a fundamental operation in various machine learning algorithms. As the size of dataset growing rapidly, it is inevitable to run matrix multiplication on multiple servers, where each server runs a task calculating a submatrix of the result.
- However, it is well known that servers in a distributed infrastructure are subject to various faulty behaviors.
  - It can be observed that virtual machines on Amazon EC2 may be 5× slower than others of the same type.
  - It is also reported that a cluster in Facebook with thousands of server experienced 10s-100s failures on a daily basis.
- A task running on servers affected by such a faulty server becomes a straggler. A straggler may significantly affect the performance of distributed matrix multiplication as the job depends on the results of all tasks.

Tolerating Stragglers with Coding

- Conventionally, stragglers can be tolerated by running replicated tasks on multiple servers.
  - Replicated tasks require a lot of resources. To tolerate r stragglers, each task needs to be replicated on $r+1$ servers.



- On the other hand, we can run additional coded tasks which compute the multiplication of coded matrices.
  - Compared to replication, we can tolerate the same number of stragglers with much fewer additional coded tasks.
  - By dividing input matrices in more dimensions, we can reduce the size of tasks.



Background: Entangled Polynomial Code

- Given two input matrices A and B, such two matrices can be divided in three dimensions, including the rows of A, the columns/rows of A/B, and the columns of B:

$$A=\begin{bmatrix} A_{0,0} & \cdots & A_{0,z-1} \\ \vdots & \cdots & \vdots \\ A_{x-1,0}\cdots & & A_{x-1,z-1} \end{bmatrix}, \text{ and } B=\begin{bmatrix} B_{0,0} & \cdots & B_{0,y-1} \\ \vdots & \cdots & \vdots \\ B_{z-1,0}\cdots & & B_{z-1,t-1} \end{bmatrix}.$$

- Entangled polynomial code is the state-of-the-art three-dimensional coding for matrix multiplication. A EP code encodes A and B into a task $T(\delta)$ that multiplies $\Sigma_{i=0}^{x-1}\tilde{A}_i(\delta)\delta^{yzi}$ and $\Sigma_{j=0}^{y-1}\tilde{B}_j(\delta)\delta^{zj}$, where $\tilde{A}_i(\delta)=\Sigma_{l=0}^{z-1}A_{i,l}\delta^l$ and $\tilde{B}_j(\delta)=\Sigma_{l=0}^{z-1}B_{z-1-l,j}\delta^l$.
- For example, when $x=y=z=2$, we have $A=\begin{bmatrix}A_{0,0}\,A_{0,1}\\A_{1,0}\,A_{1,1}\end{bmatrix}$ and $B=\begin{bmatrix}B_{0,0}\,B_{0,1}\\B_{1,0}\,B_{1,1}\end{bmatrix}$, and $T(\delta)$ can be written as the sum of

$$\begin{array}{l} A_0B_0\delta^0 \to A_{0,0}B_{1,0}\delta^0+(A_{0,0}B_{0,0}+A_{0,1}B_{1,0})\,\delta^1+A_{0,1}B_{0,0}\delta^2 \\ A_0B_1\delta^2 \to A_{0,0}B_{1,1}\delta^2+(A_{0,0}B_{0,1}+A_{0,1}B_{1,1})\,\delta^3+A_{0,1}B_{0,1}\delta^4 \\ A_1B_0\delta^4 \to A_{1,0}B_{1,0}\delta^4+(A_{1,0}B_{0,0}+A_{1,1}B_{1,0})\,\delta^5+A_{1,1}B_{0,0}\delta^6 \\ A_1B_1\delta^6 \to A_{1,0}B_{1,1}\delta^6+(A_{1,0}B_{0,1}+A_{1,1}B_{1,1})\,\delta^7+A_{1,1}B_{0,1}\delta^8 \end{array}$$

- We can see that $T(\delta)$ is a polynomial of δ whose rank is 8. Therefore, with 9 tasks whose δs are distinct, we can interpolate the coefficients of $T(\delta)$ which contain the submatrices in $A\cdot B$.
- In particular, given any valid i,j

$$\tilde{A}_i(\delta)\tilde{B}_j(\delta)=\sum_{t=0}^{2z-2}\left(\sum_{l=\max(0,t-z+1)}^{\min(z-1,t)}A_{i,l}B_{z-1-t+l,j}\right)\delta^t,$$

where the coefficient of δ^t, if $t=z-1$, is $\Sigma_{l=0}^{z-1}A_{i,l}B_{l,j}$.
- In general, we can find that the exponents of other undesired terms will be "entangled", in order to save the degree of $T(\delta)$.

exponents in $\tilde{A}_i\tilde{B}_j$	$(0,0)$	$(0,1)$	$\cdots$	$(x-1,y-2)$	$(x-1,y-1)$
	0	z	$\cdots$	$(xy-2)z$	$(xy-1)z$
	$z-2$	$2z-2$	$\cdots$	$(xy-1)z-2$	$xyz-2$
	$z-1$	$2z-1$	$\cdots$	$(xy-1)z-1$	$xyz-1$
	z	$2z$	$\cdots$	$(xy-1)z$	xyz
	$2z-2$	$3z-2$	$\cdots$	$xyz-2$	$xyz+z-2$

- The rank of $T(\delta)$ is $xyz+z-2$, leading to a recovery threshold (the number of tasks required for decoding) of $xyz+z-1$.

Motivation

- Existing coding schemes have been focusing on the size of tasks, without considering the communication overhead incurred by sending the results of tasks back to the master, and the decoding overhead at the master.
- We propose dual entangled polynomial codes, where each task will execute two matrix multiplications with the same size as entangled polynomial codes, while the the number of required tasks for decoding is reduced to $\frac{3}{4}xyz+\frac{1}{2}z-1$.

Dual Entangled Polynomial Code

- When z is even, we define $\tilde{A}_i(\delta)=\Sigma_{l=0}^{\frac{z}{2}-1}A_{i,l}\delta^l+\Sigma_{l=\frac{z}{2}}^{z-1}A_{x-1-i,l}\delta^l$ and $\tilde{B}_j(\delta)=\Sigma_{l=0}^{\frac{z}{2}-1}B_{\frac{z}{2}-1-l,j}\delta^l+\Sigma_{l=\frac{z}{2}}^{z-1}B_{\frac{3}{2}z-1-l,y-1-j}\delta^l$. Still, consider the coefficient of δ^t in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$, we can see that $\Sigma_{l=0}^{\frac{z}{2}-1}A_{i,l}B_{l,j}$ and $\Sigma_{l=\frac{z}{2}}^{z-1}A_{x-1-i,l}B_{l,y-1-j}$ appear as coefficients with $t=\frac{z}{2}-1$ and $t=\frac{3}{2}z-1$.

t	$(0,0)$	$\cdots$	$(\frac{x}{2}-1,y-1)$	$(\frac{x}{2},0)$	$\cdots$	$(x-1,y-1)$
$\frac{z}{2}-1$	l_0	$\cdots$	l_2	l_3	$\cdots$	l_1
$\frac{3}{2}z-1$	l_1	$\cdots$	l_3	l_2	$\cdots$	l_0

- Hence, we define $T_1(\delta)=\left(\Sigma_{i=0}^{\frac{x}{2}-1}\tilde{A}_i(\delta)\delta^{\alpha i}\right)\cdot\left(\Sigma_{j=0}^{y}\tilde{B}_j(\delta)\delta^{\beta j}\right)$ and $T_2(\delta)=\left(\Sigma_{i=\frac{x}{2}}^{x-1}\tilde{A}_i(\delta)\delta^{\alpha i}\right)\cdot\left(\Sigma_{j=0}^{y}\tilde{B}_j(\delta)\delta^{\beta j}\right)\cdot\delta^{-(\frac{x}{2}-1)i-(y-1)j}$.
- A task will then be the sum of two polynomials, *i.e.*, $T_1(\delta)+T_2(\delta^{-1})$. For example, when $x=y=z=2$, we have $T_1(\delta)=(A_{0,0}\delta^0+A_{1,1}\delta^1)(B_{0,0}\delta^0+B_{1,1}\delta^1+B_{0,1}\delta^2+B_{1,0}\delta^3)$ and $T_2(\delta^{-1})=$ $(A_{1,0}\delta^0+A_{0,1}\delta^{-1})(B_{0,0}\delta^0+B_{1,1}\delta^{-1}+B_{0,1}\delta^{-2}+B_{1,0}\delta^{-3})\delta^4$. Then we can list the coefficients of δ in T_1 and T_2:

	δ^0	δ^1	δ^2	δ^3	δ^4	δ^5
$T_1(\delta)$	$A_{0,0}B_{0,0}$	noise	$A_{1,1}B_{1,1}$	$A_{0,0}B_{0,1}$	noise	$A_{1,1}B_{1,0}$
$T_2(\delta^{-1})$	$A_{0,1}B_{1,0}$	noise	$A_{1,0}B_{0,1}$	$A_{0,1}B_{1,1}$	noise	$A_{1,0}B_{0,0}$

- Therefore, we can recover the submatrices in $A\cdot B$ from T_1+T_2 with 6 tasks.

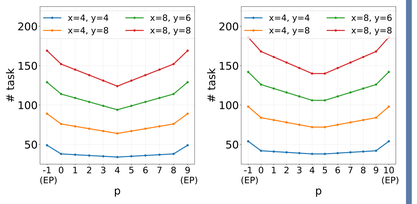
Generalization

- We now remove the original requirement of z being even. Instead, we define $\tilde{A}_i(\delta)=\Sigma_{l=0}^{p}A_{i,l}\delta^l+\Sigma_{l=p+1}^{z-1}A_{x-1-i,l}\delta^l$, and $\tilde{B}_j(\delta)=\Sigma_{l=0}^{p}B_{p-l,j}\delta^l+\Sigma_{l=p+1}^{z-1}B_{p+z-l,y-1-j}\delta^l$, where the value of p can be chosen between 0 and $z-2$.
  - If $p=\frac{z}{2}-1$, $\tilde{A}_i(\delta)$ and $\tilde{B}_j(\delta)$ are equivalent as the original construction on the left.
  - If $p=-1$ or $p=z-1$, they become equivalent as in entangled polynomial codes.
- Therefore, in $\tilde{A}_i(\delta)\tilde{B}_j(\delta)$, desired coefficients will appear as coefficients in terms of δ^p and δ^{p+z}.
- With a general value of p, we still construct a task as $T_1(\delta)+T_2(\delta^{-1})$. We can then obtain the exponents of desired coefficients as follows:

exponent in $\tilde{A}_i\tilde{B}_j$	$(0,0)$	$(0,1)$	$\cdots$	$(\frac{x}{2}-1,y-1)$
0	0	$2z-p-1$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)$
p	p	$2z-1$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)+p$
$p+z$	$p+z$	$3z-1$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)+p+z$
$2z-2$	$2z-2$	$4z-p-3$	$\cdots$	$(\frac{xy}{2}-1)(2z-p-1)+2z-2$

The picture above only show half cases of $i<\frac{x}{2}$, and the other half cases are center symmetrical to the left part.

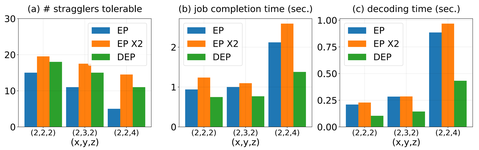
- The recovery threshold is $(\frac{xy}{2}-1)(2z-p-1)+2z-1$ when $p<\frac{z}{2}$, and $(\frac{xy}{2}-1)(p+z+1)+2z-1$ otherwise.
  - The recovery threshold is minimized when $p=\frac{z}{2}-1$.
  - The recovery threshold is also symmetric to the values of p around $\frac{z}{2}-1$.



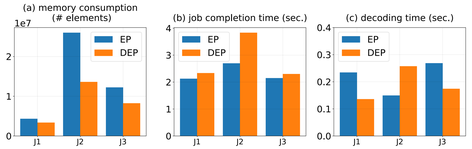
- Even in the worst cases where $p=0$ or $p=z-2$, the corresponding recovery threshold is also better than entangled polynomial codes.

Evaluation

- Implementation is based on OpenMPI. Two coded matrices in both $T_1(\delta)$ and $T_2(\delta^{-1})$ are stored on n workers, which upload the result of $T_1(\delta)+T_2(\delta^{-1})$ to a master. The master will start decoding once the number of results exceed the corresponding recovery threshold.
- Setup:
  - Running jobs of coded matrix multiplication on virtual machines on Microsoft Azure. All virtual machines are of type `B1s`, with 1 vcpu and 1 GB of memory.
  - We also run another scheme (EP x2) which simply runs two tasks with EP codes on each worker.
- We run jobs to multiply two matrices AB, where the sizes of A and B are 3600×200 and 200×3600. Each job is repeated for 20 times.



  - DEP saves the recovery threshold, leading to an increase of tolerable stragglers by 120%.
  - We observe that the major bottleneck of workers is sending their results to the master, and the doubled computation on each task only leads to marginal overhead. Moreover, the decoding overhead is also saved by DEP.
- We now run three more jobs encoded with EP and DEP codes. This time, the number of workers and the number of stragglers to tolerate are fixed.



  - In this configuration, DEP codes allow us to split the input matrix into more partitions, lowering the memory consumption of each task.
  - With saved memory, the job completion time still remains similar to EP codes, except when the value of z is increased in the job J2.
  - DEP codes can still save the decoding time by up to 42.0% in the job J1 and J3, thanks to its higher number of partitions.