Coded Computing Meets Quantum Circuit Simulation: Coded Parallel Tensor Network Contraction Algorithm

Jin Lee  
University of California, Santa Barbara  
hojin@ucsb.edu

Zheng Zhang  
University of California, Santa Barbara  
zhengzhang@ece.ucsb.edu

Sofía González-García  
University of California, Santa Barbara  
sofiagonzalezgarcia@ucsb.edu

Haewon Jeong  
University of California, Santa Barbara  
haewon@ucsb.edu

Abstract—Parallel tensor network contraction algorithms have emerged as the pivotal benchmarks for assessing the classical limits of computation, exemplified by Google’s demonstration of quantum supremacy through random circuit sampling. However, the massive parallelization of the algorithm makes it vulnerable to computer node failures. In this work, we apply coded computing to a practical parallel tensor network contraction algorithm. To the best of our knowledge, this is the first attempt to code tensor network contractions. Inspired by matrix multiplication codes, we provide two coding schemes: 2-node code for practicality in quantum simulation and hyperedge code for generality. Our 2-node code successfully achieves significant gain for $f$-resilient number compared to naive replication, proportional to both the number of node failures and the dimension product of sliced indices. Our hyperedge code can cover tensor networks out of the scope of quantum, with degraded gain in the exchange of its generality.

I. INTRODUCTION

The search for quantum advantage hinges on the fact that the dynamics of quantum computers are hard to simulate with classical resources [1]–[3], given the exponential growth of the state space with the number of qubits. Tensor networks are arguably the most important classical tool for simulating quantum circuits. Algorithms such as tensor network contractions [4]–[13], together with contraction order optimization [14] and parallelization by tensor index slicing [12] lie at the heart of determining a beyond-classical threshold in quantum computer experiments. Google’s quantum supremacy work relied on this classical benchmark [15], [16]. However, the scope of tensor network applications extends far beyond quantum circuit simulation. They are widely used as ansätze for many-body wavefunctions: Matrix Product States (MPS) for one-dimensional (1d) systems [17], [18], and their higher dimensional generalization in the form of Projected Entangled Pair States (PEPS) [19]. Moreover, tensor networks have found potential applications in machine learning [20]–[23], quantum chemistry [24]–[26] and other optimization problems [27], [28].

Despite all the great success and extensive recognition however, tensor network simulation faces reliability challenges due to the massive parallelization for efficiency. To effectively simulate state-of-the-art quantum circuits, tensor network contraction algorithms must be distributed across millions of classical machines. Distributing workloads to billions of threads inevitably increases the rate of errors. At the same time, as we are incorporating more diverse devices with different computational capacities (e.g., computation-specific accelerators, quantum processors), the variance in node response time significantly increases. Managing such uncertainties entirely at the operating systems (OS) level puts too much strain on the already overloaded OS.

In this paper, we propose a node failure tolerant scheme applied at the algorithm level by applying coded computing to tensor network contraction. Coded computing, also known as algorithm-based fault tolerance (ABFT), has proposed innovative ways to add redundancy in the computation using error-correcting codes. Coded computing strategies were developed for essential numerical algorithms such as matrix multiplication [29]–[34], FFT [35], [36], and matrix factorization [37], [38]. Matrix multiplication, in particular, has been a focal point of extensive research as it is an important backbone of scientific and machine learning applications.

Our contributions of this paper are for both tensor network and coded computing community. For the tensor network community, we provide a first work on parallel tensor network contraction algorithm that can be resilient to stragglers and failures. For the coded computing community, we provide elegant generalization of matrix multiplication codes, to the extremely expressive high-dimensional linear algebraic framework of tensor network.

II. BACKGROUND AND NOTATIONS

In this section, we provide a concise overview of tensor network contraction and its parallel algorithms, as well as the notations employed throughout the paper. For a more extensive review of this subject, refer to [39]–[42].

A. Tensor Networks and contraction

Tensors are multidimensional arrays that extend the concepts of one-dimensional vectors and two-dimensional matrices to higher dimensions, and the tensor network formalism represents linear operations between tensors using graph structures. A tensor network consists of nodes and edges, where each node represents a tensor, and each edge connected to the node represents the index of the tensor. Diagrammatic representations of one-dimensional tensor $\mathbf{A}$, two-dimensional tensor $\mathbf{B}_{i,j}$, and three-dimensional tensor $\mathbf{C}_{i,j,k}$ are shown in Fig. 1. We use bold text with its index labels given in subscript for tensors. E.g., $\mathbf{B}_{i,j}$ is a two-dimensional tensor with indices labeled as $i$ and $j$; note that this is not a scalar entry $B[i, j]$. $\mathbf{C}_{i,j,k}$ is a three-dimensional tensor and $\mathbf{C}_{1,j,k}$ is a two-dimensional tensor when the first index is fixed to $i = 1$.

An edge connecting two nodes in a tensor network represents a contraction operation, which generalizes an inner product of the dimension—pairwise multiplication followed by summation. For instance, matrix multiplication can be represented as...
two nodes as shown in Fig. 2(c). We call an index shared by hyperedges terms edge and index interchangeably in this paper. In tensor after the contraction and the final tensors are indexed only by Note that the indices of closed edges disappear throughs in quantum circuit simulation [6], [7], [43].

We say that the tensor network is sliced by the index $k$, and each $\sigma_{k}$ is a sliced partition of the original tensor network. Chen and Zhang [12] proposed an algorithm that distributed the computation of each sliced partition to a separate compute node and summing them up at the end to obtain $\sigma_{\text{final}}$. For computing the sliced partition $\sigma_{k}$, no inter-node communication is required, which makes the algorithm embarrassingly parallel. In Figure 3 we show the diagrammatic representation of slicing: the corresponding edge for the slicing index $k$ is erased and the connected tensors $B_{i,j,k}$, $D_{k,l}$, $E_{k,m}$ are replaced with sliced subtensors (e.g., $B_{i,j,k}$, $D_{1,l}$, $E_{1,m}$).

For the rest of the paper, we further simplify the notations by abstracting out all parts not related to the slicing operation as $T_{\ldots}$, where ‘\ldots’ denotes all indices other than the slicing index. E.g., [3] can be written as:

$$\sigma_{k} = \sum_{i,j,l,m,n} A_{i,j,k} B_{i,j,k} D_{k,l} E_{k,m} F_{l,m,n} G_{m,n,g}.$$ (3)

Tensors $A, C, F, G$ that are not connected to the edge $k$ are all abstracted into a big tensor $T_{\ldots}$. Finally, we say two edges are adjacent if they are connected with the same node and non-adjacent if they do not share a node.

### III. System Model and Problem Statement

The goal of this paper is to develop coded computing strategies for parallel tensor network contraction for a given tensor network $G(V,E)$. For parallel tensor network contraction, we adopt Chen & Zhang’s algorithm [12] described in Section II-B. We assume a computing model similar to [29], which consists of a master compute node (master) that distributes computation inputs and assign tasks to worker compute nodes (workers). Workers then perform the assigned computation job and report the result back to the master. With Chen & Zhang’s algorithm [12], each worker will compute a sliced tensor network $\sigma_{k}$ and the master node will aggregate the outputs to obtain $\sigma_{\text{final}} = \sum_{k} \sigma_{k}$ as shown in Fig. 4.

An important distinction from the model given in [29] is that for high-dimensional tensor contraction, it is often compute-bound rather than storage-bound. Thus, instead of a storage constraint, we impose that each worker in the coded computing...
scheme cannot have computational costs exceeding the original uncoded algorithm, i.e., when a worker performs coded computation on encoded tensor network $\tilde{\sigma}_k$, it should satisfy the following:

$$\text{comp. complexity}(\tilde{\sigma}_k) \leq \text{comp. complexity}(\sigma_k).$$

(4)

Computation and memory requirements in tensor network contraction is fully determined by the network topology\(^\text{1}\) and thus having the same topology with same tensor dimensions implies the same computation cost.

We assume that in the encoded algorithm, the tensor network $G$ is sliced multiple times at $n$ edges, each of which is a $m_i$-node edge with its index dimension $L_i$ ($i = 1, 2, \ldots, n$). This produces $N = \prod_{i=1}^{n} L_i$ sliced tensor networks and each of them will be distributed to individual workers. After all $N$ workers complete their computation, the master aggregates and obtains $\sigma_{\text{final}}$. We further make an assumption that all sliced edges are non-adjacent to each other.

We assume that there can be $f$ worker failures and we cannot recover any partial result from the failed worker. We define $f$-resilient number as the number of total workers required in the system to retrieve $\sigma_{\text{final}}$ in case of arbitrary $f$ node failures. We will compare our coding strategy with a naive replication strategy, which has a $f$-resilient number of $N(f + 1)$, as we need at least $(f + 1)$ replicas of each worker. We now define our problem below.

**Problem Statement.** For a tensor network contraction parallelized by slicing $n$ non-adjacent indices, each of which is $m_i$-node and $L_i$-dimensional ($i = 1, 2, \ldots, n$), our aim is to design a coded computing strategy that can be resilient to any $f$ failures while maintaining the same computational complexity per worker and achieve a $f$-resilient number, desired final outcome will always be retrieved.

\(^1\)This is assuming that the algorithm does not optimize for any sparsity in the tensor data.

Figure 4: (a) System model for distributed parallel tensor network contraction without coding. Failure on any computational node will result as a failure of entire system (b) System model for distributed coded parallel tensor network contraction. Even with some node failures, as long as the number of successful computational nodes is over $f$-resilient number, desired final outcome will always be retrieved.
networks commonly used in quantum simulation, which are the main interest of contemporary tensor network studies. The second code is for general tensor networks, applicable for any tensor network as long as sliced indices are non-adjacent to each other, which is also a practical assumption.

A. 2-node Code for Quantum Simulation

While \( m \)-node hyperedges can represent general tensor operations, in many practical applications, most edges are conventional 2-node edges. For instance, tensor networks for quantum simulation often follow the Projected Entangled-Pair States (PEPS) structure [19], which form a 2D-grid graph that require only 2-node edges. PEPS is also commonly used in condensed matter physics [45]–[50]. We thus first develop a coding scheme specifically designed for slicing 2-node indices, which we refer to as the 2-node code.

**Theorem 1.** For a tensor network contraction parallelized by slicing at \( n \) non-adjacent indices, each of which is 2-node and \( L_i \)-dimensional \((i = 1, 2, \ldots, n)\), \( f \)-resilient number of following is achievable:

\[
f + 2 \prod_{i=1}^{n} L_i - 1. \tag{6}
\]

The gain compared to replication for \( f \) failures is:

\[
\Delta = (\prod_{i=1}^{n} L_i - 1)(f - 1). \tag{7}
\]

**Proof.** We define template polynomials for \( i = 1, \ldots, n \):

\[
p^{(1)}_i(x) = 1 + x + x^2 + \ldots + x^{L_i-1}, \tag{8}
\]

\[
p^{(2)}_i(x) = x^{L_i-1} + x^{L_i-2} + x^{L_i-3} + \ldots + 1,
\]

and then encoding polynomials for index \( i \) are \( p^{(1)}_i\left(\prod_{j=1}^{i-1} L_j\right) \) with each coefficient substituted with the sliced tensors. E.g., for \( i = 2 \), let us call the two tensors connected to this index as \( B^{(1)} \) and \( B^{(2)} \). Then, the encoded tensor polynomials are:

\[
\tilde{B}^{(1)}_i(x) = B^{(1)}_{L_i-1} x^{L_i-1} + \ldots + B^{(1)}_{L_i-3} x^{L_i-3} + \ldots + 1,
\]

\[
\tilde{B}^{(2)}_i(x) = B^{(2)}_{L_i-1} x^{L_i-2} + \ldots + B^{(2)}_{L_i-3} x^{L_i-3} + \ldots + 1.
\]

Note that all of the desired terms of the product \( \tilde{B}^{(1)}_i(x) \tilde{B}^{(2)}_i(x) \) are aligned in the coefficient of \( x^{L_i(L_i-1)} \).

Likewise, the product of encoded tensors of index \( i \) is a polynomial of \( 2(L_i - 1) \prod_{j=1}^{i-1} L_j \) order which all desired terms are confined in \( (L_i - 1) \prod_{j=1}^{i-1} L_j \) order term. Hence, product of all encoded tensors is a polynomial of \( 2(\prod_{i=1}^{n} (L_i - 1)) \) order which all desired terms are confined in \( \prod_{i=1}^{n} (L_i - 1) \) order term. With any successful \( 2(\prod_{i=1}^{n} L_i - 2 + 1) \) encoded tensor networks each using different arbitrary constant \( \sigma \), which is the final outcome of the original tensor network. Please refer to Appendix A for further details on the proof. \( \square \)

The corresponding gain compared to naive replication is:

\[
\Delta = (\prod_{i=1}^{n} L_i - 1)(f + 1) - \prod_{i=1}^{n} \left( \frac{m^{L_i-1}}{m_i - 1} \right) + 1. \tag{12}
\]

**B. Hyperedge Code for General Case**

The tensor network framework is increasingly gaining recognition across various communities outside of the quantum field, and tensor network structures that incorporate hyperedges may find more use in the future. We thus provide a more general coding scheme beyond the 2-node code, suitable for any tensor network, which we refer to as the hyperedge code. We establish an achievable scheme for hyperedge codes in Theorem 2. The provided code works for any tensor network as long as two sliced edges are not adjacent to each other.

**Theorem 2.** For a tensor network contraction parallelized by slicing at \( n \) non-adjacent indices, each of which is \( m_i \)-node and \( L_i \)-dimensional \((i = 1, 2, \ldots, n)\), \( f \)-resilient number of following is achievable:

\[
f + 2 \prod_{i=1}^{n} \frac{(m_i^{L_i-1} - 1)}{(m_i - 1)}. \tag{11}
\]

The corresponding gain compared to naive replication is:

\[
\Delta = (\prod_{i=1}^{n} L_i - 1)(f + 1) - \prod_{i=1}^{n} \frac{(m^{L_i-1} - 1)}{(m_i - 1)} + 1. \tag{12}
\]

**Proof.** We define template polynomial for \( i = 1, \ldots, n \):

\[
p_i(x) = 1 + x + x^{m_i} + x^{m_i+m_i^2} + \ldots + x^{m_i+\ldots+m_i^{L_i-2}},
\]

and then the encoding polynomial for index \( i \) is \( p_i\left(\prod_{j=i}^{n} m_j^{L_j-1} \right) \) with each coefficient substituted with the sliced tensors. E.g., for \( i = 2 \), let us call the \( m_2 \) tensors...
connected to this index as $B(j)$ where $j = 1, \ldots, m_2$. Then, the encoded tensor polynomials are:

$$B(j)(x) = B_{L_2}(x) = B(j)_{L_2} + \cdots + B(j)_{L_2-1}x^{m_2-1} + \cdots + B(j)_{L_2-2}x^{m_2-2},$$

Note that every desired term of the product $\prod_{j=1}^{m_2} B(j)(x)$ is each confined in different order term only by itself, and the resulting polynomial is of degree $m_2-1$. Likewise, the product of encoded tensors of index $i$ is a polynomial of degree $m_1-1$. We will make an informal argument with as simple setting where all edges are not proven to be optimal nor tight.

C. Master Node Complexity

We provide an example for hyperedge code given in Theorem 2 when we have two sliced tensor networks each using different arbitrary constant $x$, all the desired outcomes of sliced tensor networks can be retrieved and the final outcome of the original tensor network can be acquired by their summation. Please refer to Appendix B for further details on the proof.

Example 2 (Hyperedge code). We provide an example for hyperedge code given in Theorem 2 when we have two sliced edges with $m_1 = 3, L_1 = 2$ and $m_2 = 4, L_2 = 2$. The contraction equation can be written as:

$$\sigma_{a,b} = \sum(A^{(1)}_a - A^{(2)}_a - A^{(3)}_a - A^{(4)}_a)B^{(1)}_bB^{(2)}_bB^{(3)}_bB^{(4)}_bT,$$

where $a \in \{1,2\}$ and $b \in \{1,2\}$ represent sliced indices. We encode the sliced tensors as

$$\tilde{A}^{(i)} = A^{(i)}_a - x$$

$$\tilde{B}^{(j)} = B^{(j)}_a + x^{1+3}$$

where $i = 1, 2, 3$ and $j = 1, 2, 3, 4$. $x$ can be substituted with a distinct evaluation point at each worker. After the local contraction operation, the result at each worker is:

$$\tilde{\sigma} = \sum(\tilde{A}^{(1)}\tilde{A}^{(2)}\tilde{A}^{(3)}\tilde{B}^{(1)}\tilde{B}^{(2)}\tilde{B}^{(3)}\tilde{B}^{(4)})T,$$

$$= \sigma_{1,1} + \sigma_{2,1}x^3 + \sigma_{1,2}x^4 + \sigma_{2,2}x^{19} + \cdots = \text{poly}(x^{19}).$$

Hence, with any $(19 + 1)$ successful workers with distinct evaluation points, we can perform polynomial interpolation to retrieve $\sigma_{\text{final}}$. The gain compared to naive replication is

$$\Delta = 4(f + 1) - (f + 20) = 3f - 16 \text{ (workers)},$$

which is consistent with Theorem 2.

Remark 2. Negative product term of $m_i$ in (12) of Theorem 2 degrades the gain of the hyperedge code. Intuitively, that is because hyperedge code confines all desired terms each in different order terms hence resulting polynomial requires higher degree for larger $\{m_i\}$. On the contrary, 2-node code confines all desired terms in a single term of the resulting polynomial, therefore its gain (7) does not suffer such loss. However, we emphasize that our codes and their upper bounds of $f$-resilient number are not proven to be optimal nor tight.

D. Existing matrix codes under our framework

Figure 5: Diagrammatic representation of (a) polynomial code and (b) MatDot code in tensor network formalism.

As the tensor network formalism is a general linear algebraic framework that also encompasses matrix multiplication, our work can be thought of as a generalized expansion of precedent matrix multiplication coding schemes. For instance, in the tensor network formalism, MatDot codes [29] slice a 2-node index between $A$ and $B$ (see Fig. 5), which corresponds to the case of $n = 1, m_1 = 2$, and $N = L_1$ for Theorem 1. In the formalism of our work, $f$-resilient number of the 2-node code is $f + 2L_1 - 1$, which is consistent with the $f$-resilient number of MatDot codes [29]. On the other hand, polynomial code [30] corresponds to slicing through open edges in tensor network formalism (see Fig. 5). This is a case we did not discuss in our work as open edges are not to be sliced as they do not reduce the complexity of the tensor network. Finally, $n$-matrix multiplication codes described in [29] correspond to a chain in tensor network formalism and slicing through all edges (with alternating index dimension of $s$ and $t$). While this gave a rise to high $f$-resilient number in the original work [29], our work shines a new light on this problem that in practical algorithms, coding overhead can be much lower than $n$-matrix codes as we only slice through a select few edges instead of all edges [12].

V. DISCUSSION AND FUTURE WORKS

This work is the first attempt at applying coded computing for reliable tensor network contraction on large-scale parallel machines. There are many more exciting future directions to explore. While we provide a coding scheme that works only for non-adjacent edges with some practical justification, making it more general to include both adjacent and non-adjacent edge slicing would be an interesting problem. Another crucial research question is a lower bound on the $f$-resilient number of coded tensor network algorithms. While we provide two different achievable schemes in Theorem 1 and Theorem 2, we do not know if the proposed approach is optimal or how far from optimality it is. Finally, as we are inspired by very practical applications, it would be valuable to implement the proposed coded computing strategies and perform an empirical evaluation on real-world high-performance computing systems.
Likewise, all the desired terms of $A_{s_1}^{(1)} A_{s_2}^{(1)} B_{s_2}^{(1)} B_{s_2}^{(1)}$ for $s_1 \in \{1, 2, ..., L_1\}$ and $s_2 \in \{1, 2, ..., L_2\}$ are confined in one coefficient of $(L_1 L_2 - 1)$ order term.

For the last index $n$, encoded tensors are

$$Z_n^{(1)} = \sum_{k=1}^{L_n} x(k-1) \prod_{i=1}^{n-1} L_i Z_{k,-}^{(1)}$$

$$Z_n^{(2)} = \sum_{k=1}^{L_n} x(k-n) \prod_{i=1}^{n-1} L_i Z_{k,-}^{(2)}$$

And their product is

$$Z_n^{(1)} Z_n^{(2)} = x(k-n-1) \prod_{i=1}^{n-1} L_i (Z_{k,-}^{(1)} Z_{k,-}^{(2)} + \cdots + Z_{k,-}^{(n)} Z_{k,-}^{(n)})$$

As a result, an encoded tensor network produces a polynomial at the end of the contraction, which confines all desired outcome of sliced tensor networks in one coefficient of $(\prod_{i=1}^{n} L_i - 1)$ order term.

$$\sigma = \sum (\tilde{A}_n^{(1)} \tilde{A}_n^{(2)} \tilde{B}_n^{(1)} \tilde{B}_n^{(2)} \tilde{Z}_n^{(1)} \tilde{Z}_n^{(2)}) T_n$$

$$= \sum \left( \prod_{s_1, s_2, ..., s_n} (A_{s_1}^{(1)} A_{s_2}^{(1)} B_{s_2}^{(2)} B_{s_2}^{(2)}) x(\prod_{i=1}^{n} L_i - 1) + \text{(other order terms)} \right) T_n$$

$$= \sigma_{\text{final}} x(\prod_{i=1}^{n} L_i - 1) + \text{(other order terms)} = \text{poly}(x(2n))$$

Hence, with any successful $(2\prod_{i=1}^{n} L_i - 2 + 1)$ encoded tensor networks each using different arbitrary constant $x$, coefficient of $(\prod_{i=1}^{n} L_i - 1)$ order term in the resulting polynomial can be retrieved, which is the final outcome of the original tensor network.

Therefore, for $f$ node failures, $f$-resilient number is

$$f + 2 \sum_{i=1}^{n} L_i - 1$$

for 2-node code.

**APPENDIX B**

**PROOF OF THEOREM 2**

Let us refer to the sliced indices by number $i = 1, 2, ..., n$. Each index $i$ is $L_i$ dimensional and shared by $m_i$ tensors. For convenience, let us denote the tensors sharing index 1 as $A_{(1)}^{(1)}, A_{(2)}^{(1)}, \ldots, A_{(m_1)}^{(1)}$, and tensors sharing index 2 as $B_{(1)}^{(1)}, B_{(2)}^{(1)}, \ldots, B_{(m_2)}^{(2)}$. We will denote tensors sharing index $n$ as $(Z_{(1)}^{(1)}, Z_{(2)}^{(2)}, \ldots, Z_{(m_n)}^{(n)})$. Each of the tensors is sliced by the selected index, for instance, $A_{(1)}^{(1)}$ is sliced as $A_{s_1}^{(1)}$ for fixed value $s_1 \in \{1, 2, ..., L_1\}$ index 1. Then, for fixed value $s_i$ for each index $i$, the equation for each sliced tensor network corresponding to the string $s_1s_2...s_n$ is

$$\sigma_{s_1s_2...s_n} = \sum_{j=1}^{m_1} A_{(j)}^{(1)} \prod_{k=1}^{m_2} B_{(k)}^{(2)} \prod_{l=1}^{m_n} Z_{(l)}^{(n)} T_n,$$
where each index $i$ spans by $s_i \in \{1, 2, \ldots, L_i \}$. Hence, there are $(N = \prod_{i=1}^{L_i} L_i)$ product terms in the summation we need to retrieve.

For an index $i$, we define a template polynomial $p_i(x)$ to encode tensors sharing the corresponding index as

$$ p_i(x) = 1 + x + x^1 + m_1 + x^{1+m_1} + m_2 + x^{1+m_1+m_2} + \ldots + x^{1+m_1+\ldots+m_{L_i-2}} = \sum_{j=1}^{L_i} x^{m_{j-1}}. $$

For instance, encoding polynomial for index 1 is $p_1(x)$. That is, we encode each tensor sliced by the index 1 by putting the sliced tensors on each of the coefficients of encoding polynomial $p_1(x)$:

$$ \hat{A}^{(k)} = \prod_{j=1}^{L_i} A^{(k)}_{j=-m_{j-1}}^{m_{j-1}} (k = 1, 2, \ldots, m_1) $$

and

$$ A_{1+}^{(k)} + A_{2+}^{(k)} x + A_{3+}^{(k)} x^1 + m_1 + \ldots + A_{L_1+}^{(k)} x^{m_{L_1-1}}, $$

for $k \in \{1, 2, \ldots, m_1 \}$.

Then, the product of the encoded tensors results in a polynomial as below:

$$ \prod_{k=1}^{m_1} \hat{A}^{(k)} = \prod_{k=1}^{m_1} A_{1+}^{(k)} + x^{m_1} \prod_{k=1}^{m_1} A_{2+}^{(k)} + x^{m_1+m_2} \prod_{k=1}^{m_1} A_{3+}^{(k)} + \ldots $$

and

$$ + x^{m_1+m_2+m_3} \prod_{k=1}^{m_1} A_{1+}^{(k)} + \ldots + x^{m_1+m_2+\ldots+m_{L_i-2}} \prod_{k=1}^{m_1} A_{L_1+}^{(k)}. $$

+(other order terms) = $poly(x^{m_{L_i-1}+\ldots+m_1})$.

Each term of $\prod_{k=1}^{m_1} A_{s_i+}^{(k)}$ for $s_i \in \{1, 2, \ldots, L_i \}$ is confined on different order term only by itself. That is because order of individual term of the encoding polynomial $p_1(x)$ is larger than $\times m_1$ of previous order.

For index 2, we use encoding polynomial $p_2(x^{m_{L_i-1}})$ where

\begin{equation}
\begin{aligned}
\prod_{k=1}^{m_1} A_{s_i+}^{(k)} &= 1 + x + x^{1+m_2} + x^{1+m_2+m_3} + x^{1+m_2+m_3+m_4} + \ldots + x^{1+m_2+\ldots+m_{L_i-2}+m_{L_i}-1} \\
&= \sum_{j=1}^{m_2} x^{m_{j-1}+\ldots+m_{L_i-2}}.
\end{aligned}
\end{equation}

Same as for index 1, we encode each tensor of index 2 by putting the sliced tensors on each of the coefficients of encoding polynomial $p_2(x^{m_{L_i-1}})$:

$$ \tilde{B}^{(k)} = \prod_{j=1}^{m_2} B_{j=-m_{j-1}}^{m_{j-1}} (k = 1, 2, \ldots, m_2) $$

and

$$ B_{1+}^{(k)} + B_{2+}^{(k)} x + B_{3+}^{(k)} x^{1+m_2} + B_{4+}^{(k)} x^{1+m_2+m_3} + B_{5+}^{(k)} x^{1+m_2+m_3+m_4} + \ldots $$

Then the product of the encoded tensors for index 2 results in a polynomial as below:

$$ \prod_{k=1}^{m_2} B_{s_i+}^{(k)} = \prod_{k=1}^{m_2} B_{1+}^{(k)} + x^{m_2} \prod_{k=1}^{m_2} B_{2+}^{(k)} + x^{m_2+m_3} \prod_{k=1}^{m_2} B_{3+}^{(k)} + x^{m_2+m_3+m_4} \prod_{k=1}^{m_2} B_{4+}^{(k)}.$$