symmetry

MDPI

Article

Newton-like Polynomial-Coded Distributed Computing for Numerical Stability

Mingjun Dai [1,2,*], Xiong Lai [2], Yanli Tong [2] and Bingchun Li [1]

[1] School of Computer Science and Technology, Kashi University, Kashi 844000, China; mathchun-ks@163.com
[2] Guangdong Provincial Engineering Center for Ubiquitous Computing and Intelligent Networking, College of Electronic and Information Engineering, Shenzhen University, Shenzhen 518060, China; 2110436133@email.szu.edu.cn (X.L.); tongyanli2019@email.szu.edu.cn (Y.T.)
* Correspondence: mjdai@szu.edu.cn

Abstract: For coded distributed computing (CDC), polynomial code is one prevalent encoding method for CDC (called Poly-CDC). It suffers from poor numerical stability due to the Vandermonde matrix serving as the coefficient matrix which needs to be inverted, and whose condition number increases exponentially with the size of the matrix or equivalently with the number of parallel worker nodes. To improve the numerical stability, especially for large networks, we propose a Newton-like polynomial code (NLPC)-based CDC (NLPC-CDC), with a design dedicated for both matrix–vector and matrix–matrix multiplications. The associated proof of the constructed code possesses a (n,k)-symmetrical combination property (CP), where symmetrical means the worker nodes have identical computation volume, CP means the k-symmetrical original computing tasks are encoded into $n(n \geq k)$-symmetrically coded computing tasks, and the arbitrary k resulting from the n-coded computing tasks can recover the intended computing results. Extensive numerical studies verify the significant numerical stability improvement of our proposed NLPC-CDC over Poly-CDC.

Keywords: symmetrical combination property; coded distributed computing (CDC); Newton interpolation polynomial (NIP) encoding; numerical stability

Citation: Dai, M.; Lai, X.; Tong, Y.; Li, B. Newton-like Polynomial-Coded Distributed Computing for Numerical Stability. *Symmetry* **2023**, *15*, 1372. https://doi.org/10.3390/sym15071372

Academic Editors: Gabriel Ciobanu and José Carlos R. Alcantud

Received: 11 May 2023
Revised: 27 June 2023
Accepted: 30 June 2023
Published: 6 July 2023

1. Introduction

Matrix multiplications (especially large matrix multiplications) are widely adopted in data mining, machine learning, etc. [1]. A single computing node may not be able to handle large or frequent matrix multiplications [2–6]. Due to the development of 5G/6G communication techniques [7] and network virtualizations [8] that provide efficient and reliable data delivery, especially in future 6G communication networks, the strict requirements of a high data rate and computational complexity require frequent processing of large matrices [9,10], and distributed computing (DC) based on partitioning computing tasks could effectively alleviate this problem. By introducing proper redundant computing [11,12], coded distributed computing (CDC) effectively tackles the straggler problem of DC [13,14]. Prevalent CDC designs aims at obtaining the (n,k)-symmetrical combination property (CP): k original symmetrical computing tasks are encoded into $n(n \geq k)$-coded computing tasks, and the arbitrary k resulting from the n-coded computing tasks can recover the intended computing results [15], where symmetrical means the worker nodes have identical computation volume. Based on this symmetrical assumption, the worker nodes can be viewed as having identical but independent capabilities, and the design focus of CDC can be directed towards achieving (n,k) CP.

Prevalent encoding methods for CDC are based on polynomial encoding methods [16–18], and is therefore called Poly-CDC, where the work in [16] set the fundamental/framework for Poly-CDC, while the work in [17,18] reported variants within this framework. More specifically, the the authors of [16] designed the fundamental encoding

coefficients which laid the actual framework, and the authors in [17,18] modified the encoding coefficients within this framework. The drawback lies in poor the numerical stability during the associated decoding stage: The coefficient matrix of the polynomial method is a Vandermonde matrix whose condition number increases exponentially with the size of the matrix [19–21], and the decoding process inevitably involves the interpolation of polynomials, eventually evolving into the inversion of the coefficient matrix. Note that the size of the matrix may be large by network slicing techniques [22], and a large condition number means that a small perturbation will cause a larger perturbation after decoding [23,24], thus affecting the robustness of the decoding results. This makes large networks have extremely poor numerical stability.

In order to obtain higher numerical stability, based on Newton-like polynomial code (NLPC), we propose an NLPC-based CDC (NLPC-CDC). This encoding is designed for both matrix–vector and matrix–matrix multiplications. An associated proof that the constructed code possesses a (n, k)-symmetrical combination property (CP) is provided. Extensive numerical studies verify that this method improves numerical stability significantly.

Preliminaries are given in Section 2. Afterwards, our proposed NLPC-CDC is elaborated upon in Section 3. Numerical comparisons are executed in Section 4. Finally, conclusions are drawn in Section 5.

2. Preliminaries

We first formulate the matrix–vector and matrix–matrix multiplication within the CDC framework. Afterwards, we briefly describe the Newton interpolation polynomial (NIP) encoding method.

2.1. Matrix–Vector Multiplication within the CDC Framework

The first task is to calculate the matrix–vector multiplication $\boldsymbol{Ax}$, where matrix $\boldsymbol{A} \in \mathrm{R}^{H\times W}$, column vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_W) \in \mathrm{R}^{W\times 1}$, and R stands for the real number field. For this task, the computing system consists of a master node and N-symmetrical worker nodes, as shown in Figure 1, where symmetrical means the worker nodes have identical computation capabilities. Under this symmetrical assumption, codes that have symmetical CP can be brought into play to tackle stragglers.

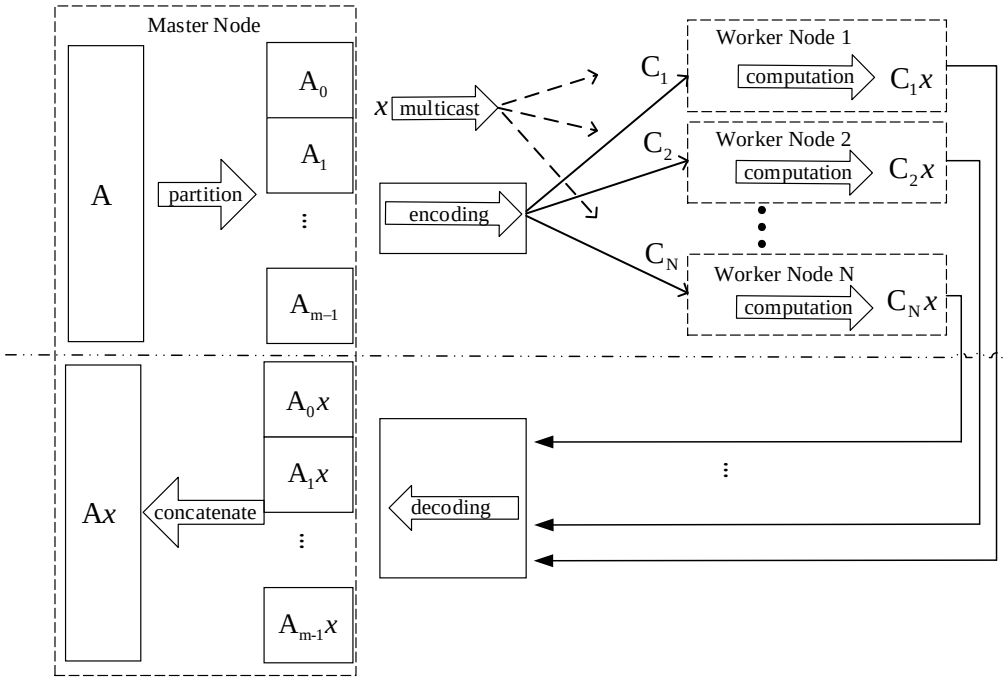


Figure 1. Matrix–vector multiplication within the CDC framework.

As H is very large, it is necessary to partition matrix $\boldsymbol{A}$ into m sub-matrices, including $A_0 \in \mathrm{R}^{(H/m)\times W}$ through $A_{m-1} \in \mathrm{R}^{(H/m)\times W}$. Encoding the m sub-matrices into N matrices is denoted by C_1 through C_N.

The master node delivers vector $\boldsymbol{x}$ to all worker nodes. Worker node $i \in \{1, 2, \ldots, N\}$ calculates the matrix–vector multiplication $C_i\boldsymbol{x}$ and delivers the calculation result back to the master node. The master node can then execute decoding to recover the intended calculation result provided that enough results have been collected.

2.2. Matrix–Matrix Multiplication within the CDC Framework

The next task is to calculate the matrix–matrix multiplication $\boldsymbol{AB}$, where the matrix $\boldsymbol{A} \in \mathrm{R}^{H\times W}$, $\boldsymbol{B} \in \mathrm{R}^{W\times L}$. For this task, the computing system consists of a master node and N worker nodes, as shown in Figure 2. The matrix $\boldsymbol{A}$ is split horizontally into m sub-matrices, including $A_0 \in \mathrm{R}^{(H/m)\times W}$ through $A_{m-1} \in \mathrm{R}^{(H/m)\times W}$. Matrix $\boldsymbol{B}$ is then split vertically into q sub-matrices, including $B_0 \in \mathrm{R}^{W\times(L/q)}$ through $B_{q-1} \in \mathrm{R}^{W\times(L/q)}$. We suppose H and L are divisible by m and q, respectively (otherwise, $\boldsymbol{A}$ and $\boldsymbol{B}$ are filled with zeros so that the number of rows of $\boldsymbol{A}$ and the number of columns of $\boldsymbol{B}$ are multiples of m and q).

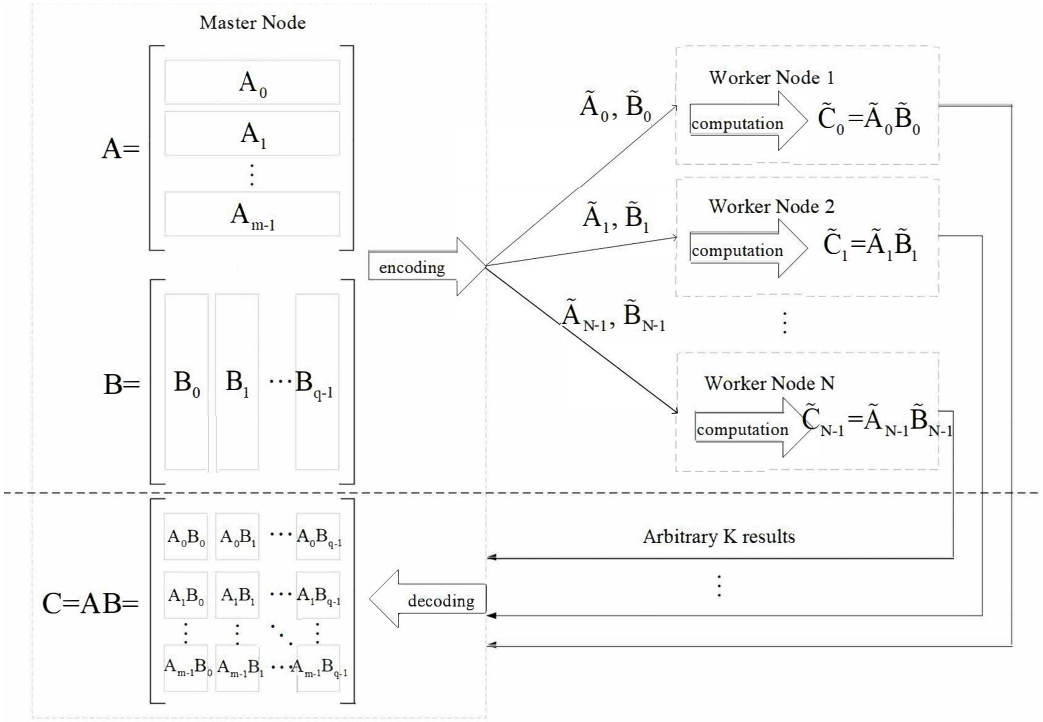


Figure 2. Matrix–matrix multiplication within the CDC framework.

The intended calculated matrix $\boldsymbol{C}$ becomes

$$\boldsymbol{C} = \boldsymbol{AB} = \begin{bmatrix} A_0B_0 & A_0B_1 & \cdots & A_0B_{q-1} \\ A_1B_0 & A_1B_1 & \cdots & A_1B_{q-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m-1}B_0 & A_{m-1}B_1 & \cdots & A_{m-1}B_{q-1} \end{bmatrix} \triangleq \begin{bmatrix} C_0 & C_1 & \cdots & C_{q-1} \\ C_q & C_{q+1} & \cdots & C_{2q-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{(m-1)q} & C_{(m-1)q+1} & \cdots & C_{mq-1} \end{bmatrix}. \quad (1)$$

Calculating $\boldsymbol{C}$ is equivalent to calculating C_0 through C_{mq-1} simultaneously. We set the recovery threshold $K = mq$, directly allocate C_0 through C_{mq-1} to K-symmetrical worker nodes (worker node i calculates C_i), and the calculation will encounter the problem of stragglers.

To tackle the straggler problem, an encoding strategy is applied on A_0 through A_{m-1} to obtain $N > K$-encoded sub-matrices, recorded as $\tilde{A}_0$ through $\tilde{A}_{N-1}$. Similarly, B_0 through B_{q-1} are encoded into $\tilde{B}_0$ through $\tilde{B}_{N-1}$.

The master node delivers the encoded pair $\left(\tilde{A}_i, \tilde{B}_i\right)$ to worker node $i \in \mathcal{N} \triangleq \{1, 2, \ldots, N\}$. Worker node $i \in \mathcal{N}$ then calculates the matrix–matrix multiplication $\tilde{C}_i = \tilde{A}_i \tilde{B}_i$ and delivers the calculation result back to the master node, which will then execute decoding to recover the original intended calculation result.

2.3. Newton Interpolation Polynomial (NIP) Encoding

As illustrated previously, symmetrical assumption leads the design to focus on CP encoding that can be directly applied in CDC systems. NIP may serve as a candidate for this encoding design.

NIP improves upon the Lagrange interpolation method by saving the number of multiplication and division operations. In addition, its base is not as complex as the Lagrange function, thus the solving process is much simpler than that of monomial based methods.

We suppose there are $n+1$ real interpolation points $x_0 \cdots x_n$ in interval $[a, b]$, and the selected base is

$$\pi_j(x) = \prod_{k=0}^{j-1}(x - x_k) = (x - x_0)(x - x_1) \cdots \left(x - x_{j-1}\right), \tag{2}$$

for $j = 0, 1, \ldots, n$. This base has the property $\pi_j(x_i) = 0$ for $\forall i < j$.

The polynomial form obtained by linear combination based on $\pi_j(x)$ is

$$\begin{aligned} c\sum_{j=0}^{n} a_j \pi_j(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots \\ + a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}), \end{aligned} \tag{3}$$

where a_j is the undetermined coefficient. We let $N_n(x)$ denote the above formula, which form is called the Newton interpolation polynomial (NIP).

By substituting $x_0 \cdots x_n$ into the above Newton polynomial for interpolation, and re-arranging into matrix form, we obtain a lower triangular matrix (4).

$$\begin{pmatrix} \pi_0(x_0) & \pi_1(x_0) & \cdots & \pi_n(x_0) \\ \pi_0(x_1) & \pi_1(x_1) & \cdots & \pi_n(x_1) \\ \cdots & \cdots & \cdots & \cdots \\ \pi_0(x_n) & \pi_1(x_n) & \cdots & \pi_n(x_n) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & x_n - x_0 & \cdots & (x_n - x_0) \cdots (x_n - x_{n-1}) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_n \end{pmatrix}. \tag{4}$$

The objective is to solve for the undetermined coefficients $a_0 \sim a_n$. It can be observed from the above formula that the interpolation coefficient solution matrix of the Newton polynomial is a lower triangular matrix.

3. Proposed NLPC-Based CDC (NLPC-CDC)

The processing is divided into matrix–vector and matrix–matrix multiplications.

3.1. NLPC-CDC for Matrix–Vector Multiplication

By applying NIP to encode A_0 through A_{m-1}, we obtain

$$\tilde{C}(x) = \sum_{i=0}^{m-1} A_i \varphi_i(x), \tag{5}$$

where $\varphi_0(x) = 1, \varphi_i(x) = (x - x_{i-1})\varphi_{i-1}(x)$. We expand $\tilde{C}(x)$ to

$$\tilde{C}(x) = A_0 + A_1(x - x_0) + A_2(x - x_0)(x - x_1) + \cdots + A_{m-1}(x - x_0)(x - x_1)\ldots(x - x_{m-2}). \tag{6}$$

The master node delivers the encoded package $\tilde{C}(x)$ and column vector $\boldsymbol{x}$ to the worker nodes. Worker node i performs multiplication of $\tilde{C}(x)\boldsymbol{x}$, and interpolates $\tilde{C}(x)$ at x_i. The matrix form of the encoded results is shown in (7).

$$\begin{bmatrix} \begin{pmatrix} \tilde{C}(x_0) \\ \tilde{C}(x_1) \\ \tilde{C}(x_2) \\ \vdots \\ \tilde{C}(x_{m-1}) \\ \vdots \\ \tilde{C}(x_n) \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \cdots & 0 \\ 1 & x_1 - x_0 & 0 & 0 & 0 \cdots & 0 \\ 1 & x_2 - x_0 & (x_2 - x_1)(x_2 - x_0) & 0 & 0 \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \cdots & \\ 1 & x_{m-1} - x_0 & (x_{m-1} - x_1)(x_{m-1} - x_0) & (x_{m-1} - x_2)(x_{m-1} - x_1)(x_{m-1} - x_0) & \cdots \cdots & (x_{m-1} - x_{m-2}) \cdots (x_{m-1} - x_0) \\ \vdots & \vdots & \vdots & \vdots & \vdots \; \vdots & \\ 1 & x_n - x_0 & (x_n - x_1)(x_n - x_0) & (x_n - x_2)(x_n - x_1)(x_n - x_0) & \cdots \cdots & (x_n - x_{m-2}) \cdots (x_n - x_0) \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ \vdots \\ A_{m-1} \end{pmatrix} \boldsymbol{x} \end{bmatrix}. \tag{7}$$

It can be seen from the above formula that to solve $\boldsymbol{Ax}$, only m values from the interpolation coefficient matrix need to be returned, which goes through inversion and multiplying by $\tilde{C}(x)$ obtains the final intended result. The interpolation coefficient matrix comprising m returned nodes is a lower triangular matrix, making the entire decoding matrix a sparse trapezoid and the recovery threshold $K = m$.

3.2. NLPC-CDC for Matrix–Matrix Multiplication

We suppose $H = L$, matrix $\boldsymbol{A}$ and $\boldsymbol{B}$ can both be partitioned into m blocks in the following manner:

$$\boldsymbol{A} = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{m-1} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} B_0 & B_1 & \ldots & B_{m-1} \end{bmatrix}. \tag{8}$$

Applying encoding over these two matrices in the following manner gives:

$$\tilde{A} = \sum_{i=0}^{m-1} A_i \varphi_i(x), \tag{9}$$

$$\tilde{B} = \sum_{j=0}^{m-1} B_j \varphi_{jm}(x), \tag{10}$$

where $\varphi_0(x) = 1, \varphi_i(x) = (x - x_{i-1})\varphi_{i-1}(x)$. Thus, the coded matrices have the following expansions:

$$\begin{aligned} \tilde{A} &= \textstyle\sum_{i=0}^{m-1} A_i \varphi_i(x) \\ &= A_0 + A_1(x - x_0) + A_2(x - x_0)(x - x_1) \\ &\quad + \cdots + \\ &\quad A_{m-1}(x - x_0)(x - x_1)\ldots(x - x_{m-2}), \end{aligned} \tag{11}$$

$$\begin{aligned} \tilde{B} = &\quad \textstyle\sum_{j=0}^{m-1} B_j \varphi_{jm}(x) \\ = &\quad B_0 + B_1(x - x_0)(x - x_1)\ldots(x - x_{m-1}) \\ &+ B_2(x - x_0)(x - x_1)\ldots(x - x_{2m-1}) + \cdots \\ &+ B_{m-1}(x - x_0)(x - x_1)\ldots\left(x - x_{m(m-1)}\right). \end{aligned} \tag{12}$$

The master node delivers the encoded pair $(\tilde{A}, \tilde{B})$ to the worker nodes. Each worker node calculates

$$\tilde{C} = (\tilde{A}, \tilde{B}) = \sum_{i=0}^{m-1} A_i \varphi_i(x) \sum_{j=0}^{m-1} B_j \varphi_{jm}(x) = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} A_i B_j \varphi_i(x) \varphi_{jm}(x). \tag{13}$$

We divide it into m parts, such that $\tilde{C}$ can be written as shown in (14).

$$\begin{pmatrix} 1 & (x-x_0) & (x-x_0)(x-x_1) & \dots & (x-x_0)(x-x_1)\dots(x-x_{m-2}) \end{pmatrix} \begin{pmatrix} A_0B_0 \\ A_1B_0 \\ A_2B_0 \\ \vdots \\ A_{m-1}B_0 \end{pmatrix} = M^1 c^1. \tag{14}$$

Note that (13) can be re-expressed as

$$\tilde{C} = Mc = \begin{pmatrix} M^1 & M^2 & M^3 & \dots & M^m \end{pmatrix} \begin{pmatrix} c^1 \\ c^2 \\ c^3 \\ \vdots \\ c^m \end{pmatrix}, \tag{15}$$

where M^1 denotes the interpolation coefficient term of the first m terms of $\tilde{C}$, and c^1 denotes the coefficient to be calculated for the first m terms of $\tilde{C}$. Similarly, the remaining items are processed.

We re-write (15) as $\tilde{C} = \sum M^i c^i, i \in \{1, 2, \cdots, m\}$. By interpolating $\tilde{C}$ with $x_0 \cdots x_n$, we obtain

$$\begin{aligned} \tilde{C}(x_0) &= A_0B_0, \\ \tilde{C}(x_1) &= A_0B_0 + A_1B_0(x_1 - x_0), \\ &\cdots \\ \tilde{C}(x_n) = A_0B_0 + A_1B_0(x_n - x_0) + A_2B_0(x_n - x_0)(x_n - x_1) + \cdots + &\ A_{m-1}B_{m-1}(x_n - x_0)^2(x_n - x_1)^2 \dots \\ &(x_n - x_{m-2})^2 \dots \left(x_n - x_{m(m-1)-1}\right). \end{aligned} \tag{16}$$

We divide $n+1$ interpolation points $x_0 \cdots x_n$ into $m+1$ sets $x^j, j \in \{1, 2, \cdots, m, m'\}$, $x^1 \sim x^m$ containing m points, where the data point of x^1 is $x_0 \cdots x_{m-1}$, the data point of x^m is $x_{m(m-1)} \cdots x_{m^2-1}$, and the remaining $n - m^2 + 1$ points $(x_{m^2} \cdots x_n)$ are placed in $x^{m'}$. Let M^{ij} denote the result of interpolation of M^i at x^j.

After all interpolated results are processed in the above manner, they can be written in matrix form as shown in (17). Thus, similar to the Newton polynomial, the matrix for solving interpolation coefficients can also be treated as a trapezoid, called the Newton-like interpolation polynomial (NLIP). We have the following theorem, whose proof is deferred to the Appendix A as the proof is lengthy and does not hinder understanding the whole idea.

$$\begin{pmatrix} \tilde{C}(x_0) \\ \vdots \\ \tilde{C}(x_{m-1}) \\ \vdots \\ \tilde{C}(x_{m^2-1}) \\ \vdots \\ \tilde{C}(x_n) \end{pmatrix} = \begin{pmatrix} \tilde{C}(x^1) \\ \tilde{C}(x^2) \\ \tilde{C}(x^3) \\ \vdots \\ \tilde{C}(x^{m-1}) \\ \tilde{C}(x^m) \\ \tilde{C}(x^{m'}) \end{pmatrix} = \begin{pmatrix} M^{11} & 0 & & & & \\ M^{12} & M^{22} & 0 & & & \\ M^{13} & M^{23} & M^{33} & 0 & & \\ & \vdots & & \ddots & 0 & \\ M^{1(m-1)} & M^{2(m-1)} & M^{3(m-1)} & \cdots & M^{(m-1)(m-1)} & 0 \\ M^{1m} & M^{2m} & M^{3m} & \cdots & M^{(m-1)m} & M^{m^2} \\ M^{1m'} & M^{2m'} & M^{3m'} & \cdots & M^{(m-1)m'} & M^{mm'} \end{pmatrix} \begin{pmatrix} c^1 \\ c^2 \\ c^3 \\ \vdots \\ c^{m-1} \\ c^m \end{pmatrix}. \quad (17)$$

Theorem 1. *For any $n(n \geq 2)$, there is an NIP matrix of n rows and k columns, as shown in (18). Randomly selecting k rows from this matrix forms a $k \times k$ sub-matrix. This sub-matrix is invertible, meaning the Newton interpolation matrix has (n,k)-symmetrical CP.*

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & x_2 - x_0 & (x_2 - x_1)(x_2 - x_0) & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & 0 \\ 1 & x_{k-1} - x_0 & (x_{k-1} - x_1)(x_{k-1} - x_0) & (x_{k-1} - x_2)(x_{k-1} - x_1)(x_{k-1} - x_0) & \cdots & \cdots & (x_{k-1} - x_{k-2}) \cdots (x_{k-1} - x_0) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & x_n - x_0 & (x_n - x_1)(x_n - x_0) & (x_n - x_2)(x_n - x_1)(x_n - x_0) & \cdots & \cdots & (x_n - x_{k-2}) \cdots (x_n - x_0) \end{pmatrix}. \quad (18)$$

4. Numerical Study

Experiments were divided into matrix–vector and matrix–matrix multiplication, where both the condition number and relative error performance was compared among the prevalent Poly-CDC and our proposed NLPC-CDC. Since the work in [16] is fundamental, our proposed NLPC is also fundamental, hence comparing these two fundamentals is needed, laying the foundation for future variant comparisons.

4.1. Matrix–Vector Multiplication

Condition number: The size of matrix $\boldsymbol{A}$ and column vector $\boldsymbol{x}$ are 5040×20 and 20×1, respectively. We set the number of blocks m to be from 8 to 20. We set the interpolation point to be a random number in $[0,3]$. By averaging over 50 realizations, the condition number of the Poly-CDC and NLPC-CDC versus m are plotted in Figure 3 for $N = 2K$ and $N = 10K$, respectively. The observations show that compared to Poly-CDC, NLPC-CDC reduces the condition number by over 10^{10} and 10^8 for scenario $N = 2K$ and $N = 10K$, respectively.

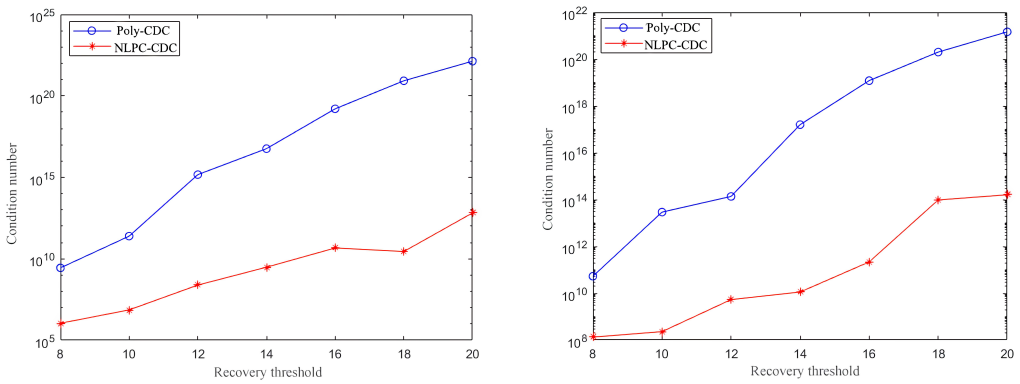
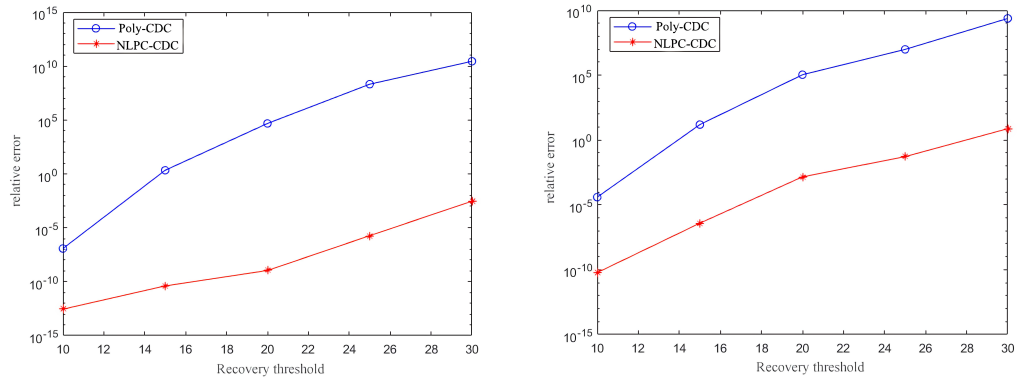


Figure 3. Condition number of matrix–vector multiplication. (**a**) $N = 2K$. (**b**) $N = 10K$.

Relative error: The size of matrix $\boldsymbol{A}$ and column vector $\boldsymbol{x}$ are 900×20 and 20×1, respectively. We set m to range from 10 to 30. The relative error of the results is plotted in Figure 4. The observations show that NLPC-CDC significantly outperforms Poly-CDC.

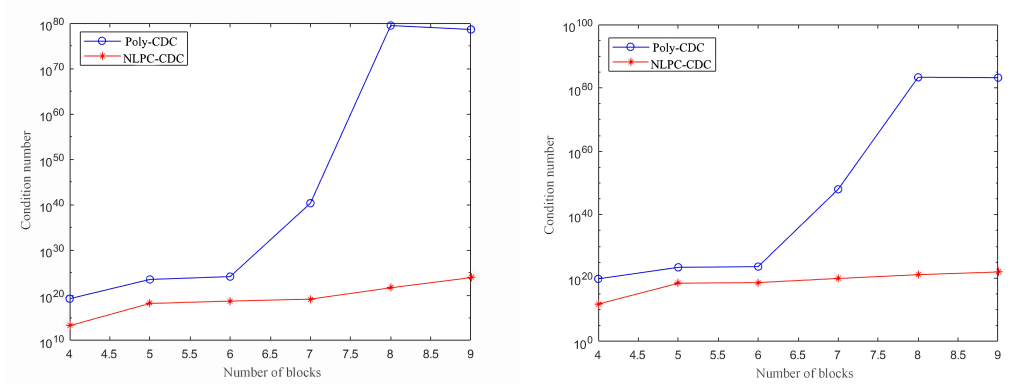


(**a**) (**b**)

Figure 4. Relative error of matrix–vector multiplication. (**a**) $N = 2K$. (**b**) $N = 10K$.

4.2. Matrix–Matrix Multiplication

Condition number: The size of matrix $\boldsymbol{A}$ and $\boldsymbol{B}$ are 2520×10 and 10×2520, respectively. We set the number of blocks m to be from four to nine. A random number in $[0, 3]$ is selected as the interpolation point. By averaging over 50 realizations, the condition numbers of Poly-CDC and NLPC-CDC vs. m are plotted in Figure 5, for $N = 2K$ and $N = 10K$, respectively. The observations show that the condition number of Poly-CDC increases rapidly when the number of blocks exceeds six, while the condition number of NLPC-CDC grows very slowly for both scenarios, $N = 2K$ and $N = 10K$, indicating significant improvement of NLPC-CDC over Poly-CDC.



(**a**) (**b**)

Figure 5. Condition number of matrix–matrix multiplication. (**a**) $N = 2K$. (**b**) $N = 10K$.

Relative error: The relatives error of the results are plotted in Figure 6. The observations show that NLPC-CDC significantly outperforms Poly-CDC.

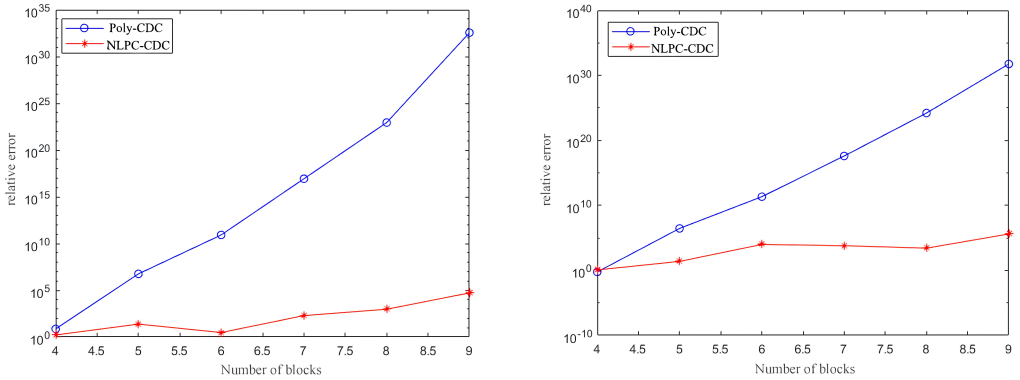


Figure 6. Relative error of matrix–matrix multiplication. (**a**) $N = 2K$. (**b**) $N = 10K$.

5. Conclusions

To improve numerical stability, a novel NLPC-based CDC was proposed. A detailed design were executed for both matrix–vector and matrix–matrix multiplications. An associated proof that the constructed code possesses an (n, k)-symmetrical combination property (CP) was provided. A series of numerical studies verified that our proposed NLPC-CDC significantly outperforms Poly-CDC in terms of the achieved condition number or relative error, and the improvement was over 10^4 for all cases. This work adopted random interpolation points, without careful selection of real interpolation points. Therefore, future work may consider selecting appropriate interpolation points to obtain the best numerical results, which variants may outperform the Poly-CDC variants [17,18]. In addition, as the encoding procedure is complicated and tedious, we may consider designing a simple encoding method that can maintain numerical stability.

Author Contributions: Conceptualization, M.D., Y.T. and B.L.; Methodology, M.D. and X.L.; Validation, X.L. and Y.T.; Investigation, X.L.; Writing—original draft, M.D.; Writing—review & editing, B.L.; Supervision, M.D. and B.L.; Project administration, M.D. and B.L.; Funding acquisition, M.D. and B.L. All authors have read and agreed to the published version of the manuscript.

Funding: The research was jointly supported by research grants from the Natural Science Foundation of China (62071304), Guangdong Basic and Applied Basic Research Foundation (2020A1515010381), Basic Research foundation of Shenzhen City (20200826152915001, 20220809155455002), and Natural Science Foundation of Shenzhen University (00002501). The Program of the Science and Technology Bureau of Kashi (No. KS2022083) and Education Department of Xinjiang Uygur Autonomous Region (XJEDU2022P080).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Proof of Theorem 1. Invertibility of a matrix is equivalent to the determinant of arbitrary square when a sub-matrix is non-zero. We prove this by the mathematical induction method.

(1) We first consider the case where $k = 2$. In this case, determinant $\begin{vmatrix} 1 & x_i - x_0 \\ 1 & x_j - x_0 \end{vmatrix} = x_j - x_i, i \neq j$. Therefore, $\begin{vmatrix} 1 & 0 \\ 1 & x_1 - x_0 \end{vmatrix} = x_1 - x_0$ is non-zero, and hence we prove the determinant is non-zero.

(2) We assume that the determinant is non-zero for an arbitrary $(k-1) \times (k-1)$ Newton-interpolation matrix, namely

$$\begin{vmatrix} 1 & x_i - x_0 & (x_i - x_0)(x_i - x_1) & \cdots & (x_i - x_0)(x_i - x_1)\dots(x_i - x_{k-3}) \\ 1 & x_j - x_0 & (x_j - x_0)(x_j - x_1) & \cdots & (x_j - x_0)(x_j - x_1)\dots(x_j - x_{k-3}) \\ 1 & x_k - x_0 & (x_k - x_0)(x_k - x_1) & \ddots & (x_k - x_0)(x_k - x_1)\dots(x_k - x_{k-3}) \\ \vdots & & & & \\ 1 & x_p - x_0 & (x_p - x_0)(x_p - x_1) & \cdots & (x_p - x_0)(x_p - x_1)\dots(x_p - x_{k-3}) \end{vmatrix} = D_{k-1} \neq 0, \tag{A1}$$

where $x_i, x_j, x_k \cdots x_p$ takes actual values at random from $x_0 \cdots x_n$. We let Q denote the above matrix within the determinant.

(3) We prove that the determinant of an arbitrary $k \times k$ Newton-interpolation matrix is non-zero.

We randomly choose interpolation point x_x from $x_0 \cdots x_n$ in the following manner and the chosen point should be distinct from $x_i, x_j, x_k \cdots x_p$ of Q. We then add one row and one column for Q in the following manner:

$$\begin{vmatrix} 1 & x_x - x_0 & (x_x - x_0)(x_x - x_1) & \cdots & (x_x - x_0)(x_x - x_1)\dots(x_x - x_{k-3}) & (x_x - x_0)(x_x - x_1)\dots(x_x - x_{k-2}) \\ & & & & & (x_i - x_0)(x_i - x_1)\dots(x_i - x_{k-2}) \\ & & & & & (x_j - x_0)(x_j - x_1)\dots(x_j - x_{k-2}) \\ & & Q & & & (x_k - x_0)(x_k - x_1)\dots(x_k - x_{k-2}) \\ & & & & & \vdots \\ & & & & & (x_n - x_0)(x_n - x_1)\dots(x_n - x_{k-2}) \end{vmatrix} \tag{A2}$$

$$= \begin{vmatrix} 1 & x_x - x_0 & (x_x - x_0)(x_x - x_1) & & (x_x - x_0)(x_x - x_1)\dots(x_x - x_{k-3}) & (x_x - x_0)(x_x - x_1)\dots(x_x - x_{k-2}) \\ 1 & x_i - x_0 & (x_i - x_0)(x_i - x_1) & \dots & (x_i - x_0)(x_i - x_1)\dots(x_i - x_{k-3}) & (x_i - x_0)(x_i - x_1)\dots(x_i - x_{k-2}) \\ 1 & x_j - x_0 & (x_j - x_0)(x_j - x_1) & & (x_j - x_0)(x_j - x_1)\dots(x_j - x_{k-3}) & (x_j - x_0)(x_j - x_1)\dots(x_j - x_{k-2}) \\ 1 & x_k - x_0 & (x_k - x_0)(x_k - x_1) & \ddots & (x_k - x_0)(x_k - x_1)\dots(x_k - x_{k-3}) & (x_k - x_0)(x_k - x_1)\dots(x_k - x_{k-2}) \\ \vdots & & \vdots & & & \\ 1 & x_p - x_0 & (x_p - x_0)(x_p - x_1) & \dots & (x_p - x_0)(x_p - x_1)\dots(x_p - x_{k-3}) & (x_p - x_0)(x_p - x_1)\dots(x_p - x_{k-2}) \end{vmatrix}. \tag{A3}$$

Through elementary operations on this matrix, we can eliminate the first row except for the first element because

$$\begin{vmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & x_i - x_x & (x_i - x_x)(x_i + x_x - x_1 - x_0) & & \left(x_i^{k-1} - x_x^{k-1}\right) - \left(x_i^{k-2} - x_x^{k-2}\right)L + \left(x_i^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_i - x_x)N \\ 1 & x_j - x_x & (x_j - x_x)(x_j + x_x - x_1 - x_0) & & \left(x_j^{k-1} - x_x^{k-1}\right) - \left(x_j^{k-2} - x_x^{k-2}\right)L + \left(x_j^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_j - x_x)N \\ 1 & x_k - x_x & (x_k - x_x)(x_k + x_x - x_1 - x_0) & \cdots & \left(x_k^{k-1} - x_x^{k-1}\right) - \left(x_k^{k-2} - x_x^{k-2}\right)L + \left(x_k^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_k - x_x)N \\ \vdots & & & & \\ 1 & x_p - x_x & (x_p - x_x)(x_p + x_x - x_1 - x_0) & \cdots & \left(x_p^{k-1} - x_x^{k-1}\right) - \left(x_p^{k-2} - x_x^{k-2}\right)L + \left(x_p^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_p - x_x)N \end{vmatrix}. \tag{A4}$$

Afterwards, we reduce the dimension to $(k-1)(k-1)$ as

$$\begin{vmatrix} x_i - x_x & (x_i - x_x)(x_i + x_x - x_1 - x_0) & & \left(x_i^{k-1} - x_x^{k-1}\right) - \left(x_i^{k-2} - x_x^{k-2}\right)L + \left(x_i^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_i - x_x)N \\ x_j - x_x & (x_j - x_x)(x_j + x_x - x_1 - x_0) & \cdots & \left(x_j^{k-1} - x_x^{k-1}\right) - \left(x_j^{k-2} - x_x^{k-2}\right)L + \left(x_j^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_j - x_x)N \\ x_k - x_x & (x_k - x_x)(x_k + x_x - x_1 - x_0) & & \left(x_k^{k-1} - x_x^{k-1}\right) - \left(x_k^{k-2} - x_x^{k-2}\right)L + \left(x_k^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_k - x_x)N \\ \vdots & & \ddots & \\ x_p - x_x & (x_p - x_x)(x_p + x_x - x_1 - x_0) & \cdots & \left(x_p^{k-1} - x_x^{k-1}\right) - \left(x_p^{k-2} - x_x^{k-2}\right)L + \left(x_p^{k-3} - x_x^{k-3}\right)M - \cdots \pm (x_p - x_x)N \end{vmatrix}. \tag{A5}$$

By extracting the common factor of each row, and letting T denote the common factor $(x_i - x_x)(x_j - x_x)(x_k - x_x)\dots(x_p - x_x)$, we obtain

$$T\begin{vmatrix} 1 & (x_i+x_x-x_1-x_0) & & \left(x_i^{k-2}+x_i^{k-3}x_x+\ldots+x_x^{k-2}\right)-\left(x_i^{k-3}+x_i^{k-4}x_x+\ldots+x_x^{k-3}\right)L+\cdots\pm N \\ 1 & \left(x_j+x_x-x_1-x_0\right) & \cdots & \left(x_j^{k-2}+x_j^{k-3}x_x+\ldots+x_x^{k-2}\right)-\left(x_j^{k-3}+x_j^{k-4}x_x+\ldots+x_x^{k-3}\right)L+\cdots\pm N \\ 1 & (x_k+x_x-x_1-x_0) & & \left(x_k^{k-2}+x_k^{k-3}x_x+\ldots+x_x^{k-2}\right)-\left(x_k^{k-3}+x_k^{k-4}x_x+\ldots+x_x^{k-3}\right)L+\cdots\pm N \\ \vdots & & \ddots & \\ 1 & \left(x_p+x_x-x_1-x_0\right) & \cdots & \left(x_p^{k-2}+x_p^{k-3}x_x+\ldots+x_x^{k-2}\right)-\left(x_p^{k-3}+x_p^{k-4}x_x+\ldots+x_x^{k-3}\right)L+\cdots\pm N \end{vmatrix} \tag{A6}$$

$$=T\begin{vmatrix} 1 & (x_i+x_x-x_1-x_0) & & x_i^{k-2}+x_i^{k-3}(x_x-L)+x_i^{k-4}\left(x_x^2-x_xL+M\right)+\cdots+x_x^{k-2}\pm N \\ 1 & \left(x_j+x_x-x_1-x_0\right) & \cdots & x_j^{k-2}+x_j^{k-3}(x_x-L)+x_j^{k-4}\left(x_x^2-x_xL+M\right)+\cdots+x_x^{k-2}\pm N \\ 1 & (x_k+x_x-x_1-x_0) & & x_k^{k-2}+x_k^{k-3}(x_x-L)+x_k^{k-4}\left(x_x^2-x_xL+M\right)+\cdots+x_x^{k-2}\pm N \\ \vdots & & \ddots & \\ 1 & \left(x_p+x_x-x_1-x_0\right) & \cdots & x_p^{k-2}+x_p^{k-3}(x_x-L)+x_p^{k-4}\left(x_x^2-x_xL+M\right)+\cdots+x_p^{k-2}\pm N \end{vmatrix}, \tag{A7}$$

where L, M, N is a constant that is a linear combination of the interpolation points $x_0 \cdots x_n$. Through elementary transformations to eliminate these constants, we obtain

$$\begin{aligned}(x_i-x_x)\left(x_j-x_x\right)(x_k-x_x)\ldots(x_p-x_x)&\begin{vmatrix} 1 & x_i-x_0 & (x_i-x_0)(x_i-x_1) & \cdots & (x_i-x_0)(x_i-x_1)\ldots(x_i-x_{k-3}) \\ 1 & x_j-x_0 & \left(x_j-x_0\right)\left(x_j-x_1\right) & \cdots & \left(x_j-x_0\right)\left(x_j-x_1\right)\ldots\left(x_j-x_{k-3}\right) \\ 1 & x_k-x_0 & (x_k-x_0)(x_k-x_1) & \ddots & (x_k-x_0)(x_k-x_1)\ldots(x_k-x_{k-3}) \\ \vdots & & & & \\ 1 & x_p-x_0 & (x_p-x_0)(x_p-x_1) & \cdots & (x_p-x_0)(x_p-x_1)\ldots(x_p-x_{k-3}) \end{vmatrix} \\ &=(x_i-x_x)\left(x_j-x_x\right)(x_k-x_x)\ldots(x_p-x_x)D_{k-1}\neq 0\end{aligned} \tag{A8}$$

which completes the proof. □

References

1. Fan, M.; Hong, C.; Yingke, L. Blind Recognition of Forward Error Correction Codes Based on a Depth Distribution Algorithm. *Symmetry* **2021**, *13*, 1094. [CrossRef]
2. Abbas, N.; Zhang, Y.; Taherkordi, A. Mobile edge computing: A survey. *IEEE Internet Things J.* **2017**, *5*, 450–465. [CrossRef]
3. Shi, W.; Cao, J.; Zhang, Q. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
4. Zhu, H.; Luan, T.H.; Dong, M. Guest editorial: Fog computing on wheels. *Peer-Peer Netw. Appl.* **2018**, *11*, 735–737. [CrossRef]
5. Wang, T.; Zhou, J.; Liu, A. Fog-Based Computing and Storage Offloading for Data Synchronization in IoT. *IEEE Internet Things J.* **2018**, *6*, 4272–4282. [CrossRef]
6. Sisinni, E.; Saifullah, A.; Han, S. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Internet Things J.* **2018**, *14*, 4724–4734. [CrossRef]
7. Pokamestov, D. Adaptation of Signal with NOMA and Polar Codes to the Rayleigh Channel. *Symmetry* **2022**, *14*, 2103. [CrossRef]
8. Shen, X.; Gao, J.; Wu, W.; Li, M.; Zhou, C.; Zhuang, W. Holistic network intelligence for 6G. *IEEE Commun. Surv. Tutotrials* **2022**, *24*, 1–30. [CrossRef]
9. Li, J.; Dang, S.; Huang, Y.; Chen, P.; Qi, X.; Wen, M. Composite multiple-mode orthogonal frequency division multiplexing with index modulation. *IEEE Trans. Wirel. Commun.* **2022**, *22*, 3748–3761. [CrossRef]
10. Li, J.; Dang, S.; Wen, M.; Li, Q.; Chen, Y.; Huang, Y. Index Modulation Multiple Access for 6G Communications: Principles, Applications, and Challenges. *IEEE Netw.* **2023**, *37*, 52–60. [CrossRef]
11. Liu, N.; Li, K.; Tao, M. Code design and latency analysis of distributed matrix multiplication with straggling servers in fading channels. *China Commun.* **2021**, *18*, 15–29. [CrossRef]
12. Shin, D.-J.; Kim, J.-J. Cache-Based Matrix Technology for Efficient Write and Recovery in Erasure Coding Distributed File Systems. *Symmetry* **2023**, *15*, 872. [CrossRef]
13. Dean, J.; Barroso, L.A. The tail at scale. *Commun. ACM* **2013**, *56*, 74–80. [CrossRef]
14. Herault, T.; Hoarau, W. FAIL-MPI: How fault-tolerant is fault-tolerant MPI? In Proceedings of the IEEE International Conference on Cluster Computing, Barcelona, Spain, 25–28 September 2006.
15. Dai, M. SAZD: A low computational load coded distributed computing framework for IoT systems. *IEEE Internet Things J.* **2020**, *7*, 3640–3649. [CrossRef]

16. Yu, Q.; Maddah-Ali, M.A.; Avestimehr, A.S. Polynomial Codes: An Optimal Design for High-Dimensional Coded Matrix Multiplication. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–11.
17. Hasırcıoğlu, B.; Gómez-Vilardebó, J.; Gündüz, D. Bivariate Hermitian Polynomial Coding for Efficient Distributed Matrix Multiplication. In Proceedings of the IEEE Global Communications Conference, Taipei, Taiwan, 7–11 December 2020; pp. 1–6.
18. Hasırcıoğlu, B.; Gómez-Vilardebó, J.; Gündüz, D. Bivariate Polynomial Coding for Efficient Distributed Matrix Multiplication. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 814–829. [CrossRef]
19. Gautschi, W.; Inglese, G. Lower bounds for the condition number of vandermonde matrices. *Numer. Math.* **1987**, *52*, 241–250. [CrossRef]
20. Gautschi, W. How (un) stable are vandermonde systems. *Asymptot. Comput. Anal.* **1990**, *124*, 193–210.
21. Reichel, L.; Opfer, G. Chebyshev-vandermonde systems. *Math. Comput.* **1991**, *57*, 703–721. [CrossRef]
22. Shen, X.; Gao, J.; Wu, W. AI-assisted network-slicing based next-generation wireless networks. *IEEE Open J. Veh. Technol.* **2020**, *1*, 45–66. [CrossRef]
23. Quarteroni, A.; Sacco, R.; Saleri, F. *Numerical Mathematics*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 37.
24. Trefethen, L.N. *Approximation Theory and Approximation Practice*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2013; Volume 128.