Parallel Collaborative ADMM Privacy Computing and Adaptive GPU Acceleration for Distributed Edge Networks

Mengchun Xia, Zhicheng Dong, *Member, IEEE*, Donghong Cai, *Member, IEEE*, Fang Fang, *Senior Member, IEEE*, Lisheng Fan, *Member, IEEE*, and Pingzhi Fan, *Life Fellow, IEEE*

***Abstract*—Distributed computing has been widely applied in distributed edge networks for reducing the processing burden of high-dimensional data centralization, where a high-dimensional computational task is decomposed into multiple low-dimensional collaborative processing tasks or multiple edge nodes use distributed data to train a global model. However, the computing power of a single-edge node is limited, and collaborative computing will cause information leakage and excessive communication overhead. In this paper, we design a parallel collaborative distributed alternating direction method of multipliers (ADMM) and propose a three-phase parallel collaborative ADMM privacy computing (3P-ADMM-PC2) algorithm for distributed computing in edge networks, where the Paillier homomorphic encryption is utilized to protect data privacy during interactions. Especially, a quantization method is introduced, which maps the real numbers to a positive integer interval without affecting the homomorphic operations. To address the architectural mismatch between large-integer and Graphics Processing Unit (GPU) computing, we transforms high-bitwidth computations into low-bitwidth matrix and vector operations. Thus the GPU can be utilized to implement parallel encryption and decryption computations with long keys. Finally, a GPU-accelerated 3P-ADMM-PC2 is proposed to optimize the collaborative computing tasks. Meanwhile, large-scale computational tasks are conducted in network topologies with varying numbers of edge nodes. Experimental results demonstrate that the proposed 3P-ADMM-PC2 has excellent mean square error performance, which is closed to distributed ADMM without privacy-preserving. Compared to centralized ADMM and distributed ADMM implemented with Central Processing Unit (CPU) computation, the proposed scheme demonstrates a significant speedup ratio.**

***Index Terms*—Distributed ADMM, homomorphic encryption, privacy computing, GPU-accelerated computation, parallel encryption.**

I. Introduction

With the large-scale application of machine-type communication system, massive data needs to be processed effectively. Moreover, high-resolution images are widely used in medical imaging, remote sensing, intelligent driving, unmanned aerial vehicle, etc, due to the rapid development of digital image techniques. Large amounts of data are used for centralized or distributed model training to effectively serve intelligent applications. In particular, some distributed learning and computing frameworks have been proposed, which can effectively utilize distributed data without compromising the privacy. For example, federated learning [1], [2], secure multiparty computation [3] and edge or fog computing [4], [5]. However, distributed learning or computing [6] requires strong single-node computing power and additional parameter exchange overhead. Fortunately, most of the local data is image and video, which is sparse data [7]. Local learning models also need to be sparse before transmission and processing [8] to save transmission bandwidth. Besides, large-scale random access to network nodes leads to the sparsity of distributed network topology. Therefore, distributed computing combining sparse signal processing can be widely applied to solve local high-dimensional computation problems with sparsity prior information.

The Least Absolute Shrinkage and Selection Operator (LASSO) problem is generally applied to formulate the sparse signal recovery due to the sparse solution. Greedy algorithm, such as orthogonal matching pursuit (OMP) [9], can recover sparse signals well when the compressed-sensing restricted isometry property (RIP) [10] is satisfied, but the complexity of OMP-like algorithms is higher. In addition, the Bayesian estimation algorithm can be well designed by using sparse prior information. For the processing of massive distributed data, [9] proposes a large number of distributed collaborative computing algorithms to approximate the performance of the central computing algorithm, without the need for a large amount of data transmission overhead and the disclosure of local data privacy. Specially, the Alternating Direction Method of Multipliers (ADMM), as an effective distributed optimization method, has been widely applied in areas such as multi-robot collaboration [11], wireless communication control [12], [13], autonomous vehicle routing [14], image processing [15], [16], and smart grid operations [17]–[19]. The ADMM decomposes the LASSO problem into several more tractable subproblems, allowing these to be executed in parallel across multiple edge devices, and ultimately achieves consensus by iteratively updating the global solution. ADMM is particularly effective for large-scale datasets with structured sparsity in the models.

In distributed computing, such as distributed OMP (Dis.-

M. Xia and Z. Dong are with the College of Information Science and Technology, Tibet University, Lhasa, 850000, China (e-mail: mengchunxiavip@163.com; dongzc666@163.com).

D. Cai is with the College of Cyber Security, Jinan University, Guangzhou 510632, China (e-mail: dhcai@jnu.edu.cn).

F. Fang is with the Department of Electrical and Computer Engineering, Western University, London N6A 5B9, Canada (e-mail: fang.fang@uwo.ca).

L. Fan is with the School of Computer Science, Guangzhou University, Guangzhou 510006, China (e-mail: lsfan@gzhu.edu.cn).

P. Fan is with the Key Lab of Information Coding and Transmission, Southwest Jiaotong University, Chengdu 610031, China (e-mail: pzfan@swjtu.edu.cn).

TABLE I: Existing Distributed Privacy Computing Schemes

Computing Algorithm	Existing Schemes	Privacy Computing		Available GPU-Acceleration		Available Security	
		DP	HE	Matrix	ModExp	Global privacy	Local privacy
Dis.-OMP	SPriFed-OMP [20]	✓		✓		✓	
	CM-Pai.-OMP [21]		✓	✓	✓		✓
Dis.-ADMM	DP-ADMM [22]	✓		✓		✓	
	scheme from [23]		✓	✓	✓	✓	
FL	NbAFL [24]	✓		✓			✓
	scheme from [25]		✓	✓		✓	✓
MPL	PEA [26]	✓		✓			✓
	PE-MPDL [27]		✓	✓	✓		✓
Fog-Com.	MaxDiff [28]	✓		✓			✓
	EEDHSC [29]		✓	✓			✓
Ours scheme	3P-ADMM-PC2		✓	✓	✓	✓	✓

OMP), distributed ADMM (Dis.-ADMM), federated learning (FL), multi-party learning (MPL), and fog computing (Fog-Com.), multiple edge nodes collaborate to perform computations, significantly reducing computational overhead, especially for large-scale datasets. However, as shown in Table I, these schemes initially paid less attention to data privacy protection during node interactions. Although federated learning and multi-party learning consider the privacy protection of local data, there is still a risk of privacy leakage.

In distributed computing employing differential privacy as a privacy-preserving mechanism, such as in SPriFed-OMP [20], DP-ADMM [22], NbAFL [24], PEA [26], and MaxDiff [28], the addition of noise may compromise the sparsity of the original signal. Moreover, introducing noise in each gradient update can adversely affect convergence and the accuracy of the final solution. In multi-party settings, noise parameters must be meticulously designed. Therefore, in scenarios demanding high-precision privacy protection, the use of homomorphic encryption schemes becomes indispensable. In distributed computing utilizing homomorphic encryption for privacy protection, such as in CM-Paillier-OMP [21], schemes from [23], PE-MPDL [27], the substantial computational overhead induced by encryption results in inefficiency for large-scale matrix operations. Such methods are often limited to small-scale experiments due to the lack of effective acceleration schemes. Furthermore, deployment on resource-constrained edge devices remains particularly challenging. The scheme proposed in [25] combines federated learning with CKKS. However, CKKS [30] is an approximate decryption scheme with complex implementation, requiring relinearization or bootstrapping operations to reduce the impact of noise during homomorphic operations. The EEDHSC in [29] combines fog computing with the ECC-ElGamal encryption scheme. Although the ECC-ElGamal scheme [31] provides equivalent security to RSA encryption scheme with shorter key lengths, it supports only homomorphic multiplication operations. Additionally, there are homomorphic encryption schemes such as BFV [32] and BGN [33]. Similar to CKKS, BFV has complex implementation requiring relinearization or bootstrapping operations. BGN supports only once homomorphic multiplication. Besides, among the aforementioned homomorphic encryption schemes, Paillier, BGN, ECC-ElGamal, and BFV can only encrypt unsigned integers. However, the Paillier homomorphic encryption scheme [34] is simple to implement and supports a limited number of homomorphic additions and constant multiplications, which aligns well with the computational tasks in this paper. Therefore, considering these factors, Paillier homomorphic encryption is used in this paper to protect data privacy during interactions.

In recent years, designing customized Graphics Processing Unit (GPU) computing paradigms for specific computational scenarios has become a research hotspot. The core idea involves hardware-software co-design, mapping computational patterns onto GPU parallel architectures to unleash extreme performance. For instance, Yu et al. proposed the TwinPilots paradigm [35]. It dynamically schedules Transformer layer computations across GPU and CPU cores. An online load-balancing planner coordinates GPU data loading time with Central Processing Unit (CPU) computation time. This achieves efficient large language model inference on memory-constrained GPU servers. Wang et al. proposed the Q-GTC paradigm [21], which achieves a several-fold speedup compared to conventional CUDA cores by mapping quantized graph neural network operations onto GPU Tensor Cores to leverage their capability in executing matrix operations at specific bit-widths. Shivdikar et al. accelerated fully homomorphic encryption (FHE) [36] through GPU-based microarchitectural extensions. By introducing a CU-side hierarchical interconnect in the AMD CDNA GPU architecture, they managed to retain FHE ciphertexts in cache, eliminating redundant memory accesses

and reducing latency. Additionally, they designed a MOD unit and 64-bit integer arithmetic cores to provide native modular arithmetic support, accelerating the most frequent modular reduction operations in FHE, while optimizing throughput via pipelining. They also proposed a Locality-Aware Block Scheduler to enhance scheduling efficiency by leveraging the temporal locality of FHE primitive blocks, further optimizing computational performance. Riazi et al. proposed the HEAX specialized architecture [37], which designs a GPU-like parallel architecture tailored for homomorphic encryption. Its computing units and memory hierarchy are custom-built for cryptographic primitives, enabling hardware-level acceleration of homomorphic encryption operations. However, in the particular field of homomorphic encryption, its computational characteristics exhibit a fundamental architectural mismatch with mainstream GPU paradigms, making the direct application of existing paradigms challenging.

Facing the above-mentioned problems, in this paper, we consider a distributed edge privacy-computing network and propose a three-phase parallel collaborative ADMM privacy computing (3P-ADMM-PC2) algorithm that integrates distributed parallel ADMM with the Paillier homomorphic encryption for sparse signal processing. The proposed 3P-ADMM-PC2 decomposes a high-dimensional computational problem into multiple low-dimensional subproblems for collaborative processing by multiple edge nodes, while also protecting data privacy during node interactions. To enable encryption and decryption of real numbers and achieve homomorphic operations, a quantization method is proposed, which maps real numbers to an unsigned integer range without affecting the homomorphic properties of the encryption scheme. Compared with traditional quantization methods [23], our method does not require additional space to represent negative numbers. However, a vector encryption is required to reduce the computing deday in bigdata. Existing homomorphic encryption schemes show lower parallelism during vector encryption and decryption due to their computational characteristics [31]–[34], which is the reason these schemes incur significant additional computational overhead. If multiple elements within a vector can be encrypted in parallel, it would significantly reduce the computational overhead. Feasible existing approach is to deploy encryption operations on GPU or FPGA [38], [39] to achieve parallel encryption. However, according to the 754-2008 IEEE Standard for Floating-Point Arithmetic [40], each GPU core supports only 32 bits or 64 bits operations. To ensure security, a key length of at least 1024 bits is required. Modular exponentiation (ModExp) of such large integers cannot be directly implemented on GPU or FPGA. A GPU-accelerated computation scheme is proposed, which decomposes computational tasks to enable multiple GPU cores to collaboratively process a single task, thereby achieving parallel operations such as encryption overall. Finally, we propose a GPU-accelerated 3P-ADMM-PC2 scheme, which enables edge nodes to collaborate with the master node for encryption and decryption computations, thereby reducing the computational burden on the master node. Summary, the contributions of this paper are as follows:

- We propose a 3P-ADMM-PC2 algorithm for large-scale sparse signal processing, integrating distributed parallel ADMM with the Paillier homomorphic encryption. In particular, proposed 3P-ADMM-PC2 decomposes high-dimensional computational tasks into low-dimensional subtasks for parallel computation by multiple edge nodes, while protecting data privacy during node interactions.
- A quantization method and a GPU-accelerated computation scheme are proposed to address the plaintext limitations of homomorphic encryption and deploy encryption operations on GPU. To prevent overflow in the ModExp of large integers under long key length, the scheme transforms high-bitwidth computations into low-bitwidth matrix and vector operations. This approach fully utilizes the multi-core characteristics of GPU to achieve parallel privacy-preserving computations.
- To reduce the computational burden on the master node, we propose a GPU-accelerated 3P-ADMM-PC2 scheme. The master node and edge nodes perform ModExp in parallel on two smaller spaces, respectively. Compared with direct computation in a large space, our approach further reduces computational overhead and enables collaborative encryption and decryption between the master node and edge nodes.

The remainder of this paper is organized as follows. Section II introduces the system model and problem formulation. Section III introduces our proposed parallel collaborative ADMM privacy computing. Section IV details our proposed adaptive GPU accelerated 3P-ADMM-PC2. Section V presents experimental results and Section VI summarizes our work.

II. System Model and Problem Formulation

Consider a distributed edge privacy-computing network, including one master node and K incompletely trusted distributed nodes, where the master node borrows the computing power of other K distributed nodes to compute a LASSO problem securely. In particular, the LASSO problem is expressed as

$$\arg\min_{\mathbf{x}\in\mathbb{R}^N} \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2+\lambda\|\mathbf{x}\|_1, \tag{1}$$

where $\mathbf{A}\in\mathbb{R}^{M\times N}(M\ll N)$ is the compressed matrix, $\mathbf{x}\in\mathbb{R}^N$ is a sparse vector. The vector $\mathbf{x}\in\mathbb{R}^N$ represents the global sparse signal to be jointly estimated from the observed data. In classical compressed sensing, $\mathbf{x}$ may denote the sparse coefficient vector of a natural image under a wavelet transform basis. In power grid reconstruction, $\mathbf{x}$ could represent the admittance or impedance parameters of a transmission line. $\mathbf{y}\in\mathbb{R}^M$ is the observation vector, $\lambda\geq 0$ is a parameter. Note that the problem (1) can be further formulated as

$$\min_{\mathbf{x},\mathbf{z}\in\mathbb{R}^N} \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2+\lambda\|\mathbf{z}\|_1 \tag{2a}$$

$$\text{s.t.} \quad \mathbf{x}-\mathbf{z}=0. \tag{2b}$$

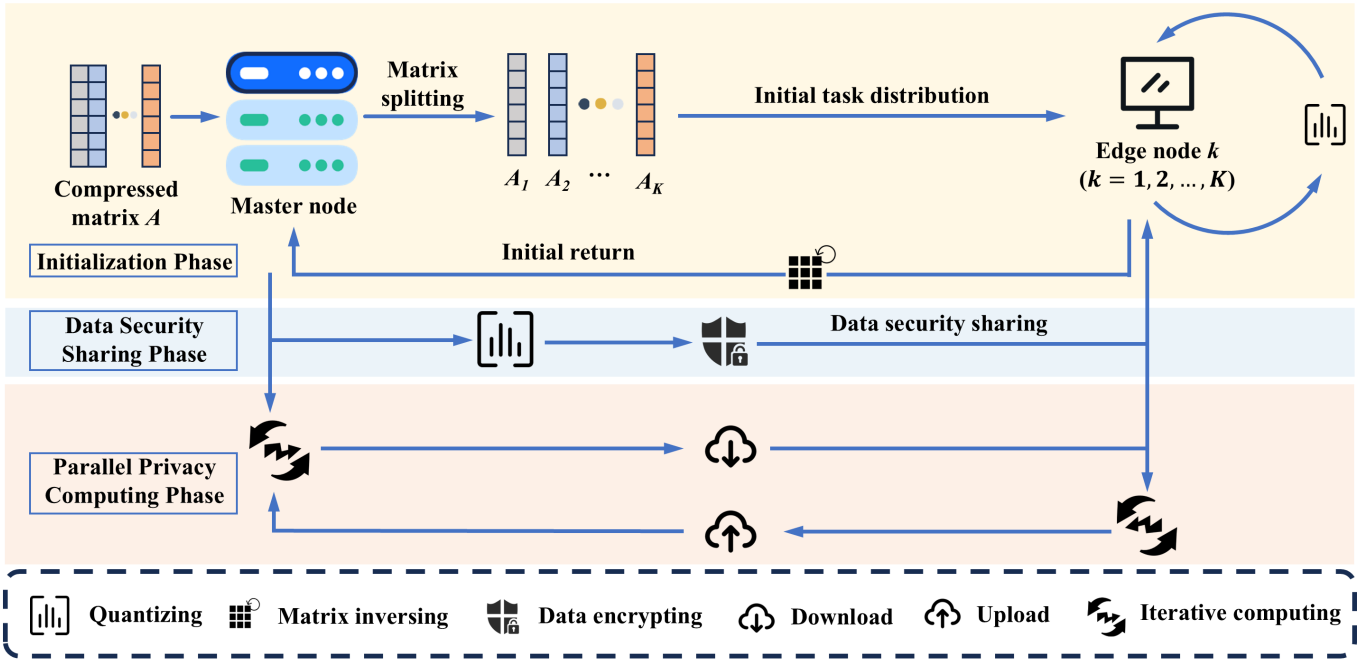


Fig. 1: Proposed 3P-ADMM-PC2 scheme.

Then the corresponding augmented Lagrangian function is expressed as

$$\mathcal{L}_\rho(\mathbf{x},\mathbf{z},\mathbf{u}) = \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{z}\|_1 + \mathbf{u}^T(\mathbf{x}-\mathbf{z}) + \frac{\rho}{2}\|\mathbf{x}-\mathbf{z}\|_2^2, \tag{3}$$

where $\mathbf{u}\in\mathbb{R}^N$ is a lagrangian multiplier vector, and $\rho\geq 0$ is a parameter. Based on the augmented Lagrangian function in (3), the iterative steps with respect to $\mathbf{x},\mathbf{z},\mathbf{v}$ for solving the LASSO problem can be expressed as

$$\mathbf{x}^{(t)} = (\mathbf{A}^T\mathbf{A}+\rho\mathbf{I}_N)^{-1}(\mathbf{A}^T\mathbf{y}+\rho(\mathbf{z}^{(t-1)}-\mathbf{v}^{(t-1)})), \tag{4a}$$
$$\mathbf{z}^{(t)} = S_{\frac{\lambda}{\rho}}(\mathbf{v}^{(t-1)}+\mathbf{x}^{(t)}), \tag{4b}$$
$$\mathbf{v}^{(t)} = \mathbf{v}^{(t-1)}+\mathbf{x}^{(t)}-\mathbf{z}^{(t)}, \tag{4c}$$

where $\mathbf{v}=\frac{1}{\rho}\mathbf{u}, \mathbf{I}_N\in\mathbb{R}^{N\times N}$ is a unit matrix, and $S_{\frac{\lambda}{\rho}}(\cdot)$ denotes the soft threshold function. It is important to point out that the complexity of (4a) is $\mathcal{O}(N^3)$. The complexity limits the application of ADMM, especially machine learning and large-scale signal processing.

To efficiently compute iterations of LASSO problem, the master node cooperates with K distributed incompletely trusted nodes. In particular, $\mathbf{x}$ is divided into $\mathbf{x}=(\mathbf{x}_1^T,\mathbf{x}_2^T,\cdots,\mathbf{x}_K^T)^T$, where $\mathbf{x}_k^T\in\mathbb{R}^{N_k}$, $N=\sum_{k=1}^K N_k$. The compressed matrix $\mathbf{A}$ is expressed as $\mathbf{A}=(\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K)$, where $\mathbf{A}_k\in\mathbb{R}^{M\times N_k}$. Then, the subproblem for solving $\mathbf{x}$ in (3) is expressed as

$$\begin{aligned}&\arg\min_{\mathbf{x}}\frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2+\mathbf{u}^T(\mathbf{x}-\mathbf{z})+\frac{\rho}{2}\|\mathbf{x}-\mathbf{z}\|_2^2\\ &=\arg\min_{\mathbf{x}}\frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2+\frac{\rho}{2}\left\|\mathbf{x}-\mathbf{z}+\frac{\mathbf{u}}{\rho}\right\|_2^2\\ &=\arg\min_{\mathbf{x}_k}\frac{1}{2}\left\|\mathbf{y}-\sum_{k=1}^K\mathbf{A}_k\mathbf{x}_k\right\|_2^2+\frac{\rho}{2}\sum_{k=1}^K\left\|\mathbf{x}_k-\mathbf{z}_k+\frac{\mathbf{u}_k}{\rho}\right\|_2^2,\end{aligned} \tag{5}$$

where we use the fact that $\mathbf{z}=(\mathbf{z}_1^T,\mathbf{z}_2^T,\cdots,\mathbf{z}_K^T)^T$ and $\mathbf{u}=(\mathbf{u}_1^T,\mathbf{u}_2^T,\cdots,\mathbf{u}_K^T)^T$.

Note that the first term of the objective function in (5) couplings K variables, resulting in a low efficiency solution. Therefore, the upper bound of the first term in (5) is given by

$$\left\|\mathbf{y}-\sum_{k=1}^K\mathbf{A}_k\mathbf{x}_k\right\|_2^2\leq\sum_{k=1}^K\left\|\frac{1}{K}\mathbf{y}-\mathbf{A}_k\mathbf{x}_k\right\|_2^2. \tag{6}$$

Problem (5) is further approximated as

$$\arg\min_{\mathbf{x}_k}\frac{1}{2}\sum_{k=1}^K\left\|\frac{\mathbf{y}}{K}-\mathbf{A}_k\mathbf{x}_k\right\|_2^2+\frac{\rho}{2}\sum_{k=1}^K\left\|\mathbf{x}_k-\mathbf{z}_k+\frac{\mathbf{u}_k}{\rho}\right\|_2^2, \tag{7}$$

where the variables are separable. Thus, problem (7) can be translated into K subproblems:

$$\arg\min_{\mathbf{x}_k}\frac{1}{2}\left\|\frac{\mathbf{y}}{K}-\mathbf{A}_k\mathbf{x}_k\right\|_2^2+\frac{\rho}{2}\left\|\mathbf{x}_k-\mathbf{z}_k+\frac{\mathbf{u}_k}{\rho}\right\|_2^2,\forall k. \tag{8}$$

Similar to (4a), the solution of (8) is expressed as

$$\mathbf{x}_k^{(t)} = (\mathbf{A}_k^T\mathbf{A}_k+\rho\mathbf{I}_{N_k})^{-1}(\mathbf{A}_k^T\mathbf{y}+\rho(\mathbf{z}_k^{(t-1)}-\mathbf{v}_k^{(t-1)})). \tag{9}$$

According to (9), (4b) and (4c), the LASSO problem in (1) can be solved by alternate iteration. To realize the synchronous

computation, the iterations (9), (4b) and (4c) are further expressed as follows:

$$\mathbf{x}_k^{(t)} = (\mathbf{A}_k^T\mathbf{A}_k + \rho\mathbf{I}_{N_k})^{-1}(\mathbf{A}_k^T\mathbf{y} + \rho(\mathbf{z}_k^{(t-1)} - \mathbf{v}_k^{(t-1)})), \forall k, \tag{10a}$$

$$\mathbf{z}^{(t)} = S_{\frac{\lambda}{\rho}}(\mathbf{v}^{(t-1)} + \mathbf{x}^{(t-1)}), \tag{10b}$$

$$\mathbf{v}^{(t)} = \mathbf{v}^{(t-1)} + \mathbf{x}^{(t-1)} - \mathbf{z}^{(t)}. \tag{10c}$$

It is noted that the updates of $\mathbf{x}_k^{(t)}$, $\mathbf{z}^{(t)}$, and $\mathbf{v}^{(t)}$ in the t-th iteration is based on the $(t-1)$-th iteration. As shown in Fig. 1, we propose a 3P-ADMM-PC2 scheme, which is summarized as follows:

- **Initialization Phase**: In this phase, the master node splits the computing task and sends them to different edge nodes. And the edge nodes return their initial calculated values to the master node.
  Master node: The master node splits $\mathbf{A}, \mathbf{z}, \mathbf{v}$ in (4a) into K parts, i.e., $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_K)$, $\mathbf{z} = (\mathbf{z}_1^T, \mathbf{z}_2^T, \cdots, \mathbf{z}_K^T)^T$, $\mathbf{u} = (\mathbf{u}_1^T, \mathbf{u}_2^T, \cdots, \mathbf{u}_K^T)^T$, and successively transmits $\alpha_k = \{\mathbf{A}_k^T\mathbf{A}_k, \rho\}$ to edge node $k, k = 1, 2, \cdots, K$.
  Edge node: Edge node k computes $\mathbf{B}_k = (\mathbf{A}_k^T\mathbf{A}_k + \rho\mathbf{I}_{N_k})^{-1} \in \mathbb{R}^{N_k \times N_k}$ and returns it to the master node. Meanwhile, edge node k saves the quantization of $\mathbf{B}_k\rho$ as $\bar{\mathbf{B}}_k$.
- **Data Security Sharing Phase**: The observation $\mathbf{y}$ in (1) is sensitive information. To prevent information leakage, the master node quantizes and encrypts the information containing $\mathbf{y}$ before sharing it.
  Master node: The master node quantizes and encrypts $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}) \in \mathbb{R}^{N_k}$, i.e.,

$$\hat{\boldsymbol{\alpha}}_k = f_{en}(\Gamma_1(\mathbf{B}_k\mathbf{A}_k^T\mathbf{y}) \in \mathbb{Z}^{N_k}, \forall k, \tag{11}$$

  where $\Gamma_i(\cdot)$ and $f_{en}(\cdot)$ are the quantization function and the encryption function, respectively.
  Edge node: The edge node k downloads and saves $\hat{\boldsymbol{\alpha}}_k$ for the computing task.
- **Parallel Privacy-Computing Phase**: In this phase, the information in (4a) is calculated and exchanged safely between the master node and the distributed edge nodes.
  Master node: At the master node, $\mathbf{z}^{(t)}$ and $-\mathbf{v}^{(t)}$ are updated according to (10b) and (10c) with $f_{de}(\hat{\mathbf{x}}^{(t)})$, where $f_{de}(\cdot)$ is the decryption function. Note that the updates of $\mathbf{z}^{(t)}, -\mathbf{v}^{(t-1)}$, and the collection of $\{\hat{\mathbf{x}}_k\}_{k=1}^K$. $-\mathbf{v}^{(t)}$ and $\hat{\mathbf{x}}^{(t)}$ require the $(t-1)$-th updates $\mathbf{z}^{(t-1)}$. To safely calculate $\mathbf{x}_k^{(t)}$ in (10a), $\mathbf{z}^{(t)}$ and $-\mathbf{v}^{(t)}$ are quantized and encrypted, resulting in

$$\hat{\mathbf{z}}_k^{(t)} = f_{en}(\Gamma_2(\mathbf{z}_k^{(t)})), \hat{\mathbf{v}}_k^{(t)} = f_{en}(\Gamma_2(-\mathbf{v}_k^{(t)})), \forall k, \tag{12}$$

  and $\hat{\mathbf{z}}_k^{(t)}, \hat{\mathbf{v}}_k^{(t)}$ are transmitted to edge node k.
  Edge node: At edge node k, a calculated value of $\mathbf{x}_k^{(t)}$ in (10a) is obtained based on $\hat{\mathbf{z}}_k^{(t-1)}$ and $\hat{\mathbf{v}}_k^{(t-1)}$, i.e.,

$$\hat{\mathbf{x}}_k^{(t)} = \hat{\boldsymbol{\alpha}}_k + \Gamma_2(\bar{\mathbf{B}}_k)(\hat{\mathbf{z}}_k^{(t-1)} + (-\hat{\mathbf{v}}_k^{(t-1)})). \tag{13}$$

  Then $\hat{\mathbf{x}}_k^{(t)}$ is uploaded to the master node.

Note that $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\mathbf{z}_k^{(t)}$, and $-\mathbf{v}_k^{(t)}$ are encrypted respectively at master node. Each edge node estimates $\hat{\mathbf{x}}_k^{(t)}$ in (13) with ciphertexts $\hat{\boldsymbol{\alpha}}_k$, $\hat{\mathbf{z}}_k^{(t)}$, and $\hat{\mathbf{v}}_k^{(t)}$. It is important to point out that calculation of $\hat{\mathbf{x}}_k^{(t)}$ in (13) consists of the addition computation and the multiply-by-a-constant computation. In addition, $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\mathbf{z}_k^{(t)}$, and $-\mathbf{v}_k^{(t)}$ are real vector, whose elements are the real numbers. Especially, the positive-negative characteristic and floating-point types are challenge to encryption and efficient computing.

III. Parallel Collaborative ADMM Privacy Computing

In this section, we introduce proposed 3P-ADMM-PC2 scheme for distributed computation. In particular, a quantization is first designed by mapping the real number to a certain range of positive integers. Furthermore, we reveal that the designed quantizer does not affect the homomorphic properties of Paillier-based ADMM privacy computing.

A. Quantization of Floating Point Type Data

Note that the elements in $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\bar{\mathbf{B}}_k$, $\mathbf{z}_k^{(t)}$, $-\mathbf{v}_k^{(t)}$ are the floating point type data, which can not be encrypted directly. Thus, we first quantize $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\bar{\mathbf{B}}_k$, $\mathbf{z}_k^{(t)}$, $-\mathbf{v}_k^{(t)}$, to $\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))$, $\Gamma_2(\bar{\mathbf{B}}_k)$, $\Gamma_2(\mathbf{z}_k^{(t)})$, $\Gamma_2(-\mathbf{v}_k^{(t)})$, expressed as

$$\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) = \left\lfloor \frac{\Delta^2(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}) - (\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))_{\min}\mathbf{1}_{N_k})}{((\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))_{\max} - (\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))_{\min})^2} \right\rfloor, \tag{14a}$$

$$\Gamma_2(\bar{\mathbf{B}}_k) = \left\lfloor \frac{\Delta(\bar{\mathbf{B}}_k - (\bar{\mathbf{B}}_k)_{\min}\mathbf{1}_{N_k \times N_k})}{(\bar{\mathbf{B}}_k)_{\max} - (\bar{\mathbf{B}}_k)_{\min}} \right\rfloor, \tag{14b}$$

$$\Gamma_2(\mathbf{z}_k^{(t)}) = \left\lfloor \frac{\Delta(\mathbf{z}_k^{(t)} - (\mathbf{z}_k^{(t)})_{\min}\mathbf{1}_{N_k})}{(\mathbf{z}_k^{(t)})_{\max} - (\mathbf{z}_k^{(t)})_{\min}} \right\rfloor, \tag{14c}$$

$$\Gamma_2(-\mathbf{v}_k^{(t)}) = \left\lfloor \frac{\Delta(-\mathbf{v}_k^{(t)} - (-\mathbf{v}_k^{(t)})_{\min}\mathbf{1}_{N_k})}{(-\mathbf{v}_k^{(t)})_{\max} - (-\mathbf{v}_k^{(t)})_{\min}} \right\rfloor, \tag{14d}$$

where $\mathbf{1}_{N_k}$ is all one vector, $\Delta > 0$, $(\cdot)_{\max}$ denotes the maximum element of a vector or matrix, $(\cdot)_{\min}$ denotes the minimum element, and $\lfloor\cdot\rceil$ denotes rounding.

Remark 1: The quantization function $\Gamma_2(\cdot)$ maps the element from the N_k dimensional real number field $\mathbb{R}^{N_k}$ to an integer set $\{0, 1, 2, \cdots, \Delta\}^{N_k}$. Especially, the negative real numbers are mapped to non-negative integers, which can be encrypted by the Paillier algorithm. To ensure that the quantization scheme does not compromise the homomorphic properties of the subsequent encryption scheme, $\Gamma_1(\cdot)$, unlike $\Gamma_2(\cdot)$, maps the element from the N_k dimensional real number field $\mathbb{R}^{N_k}$ to an integer set $\{0, 1, 2, \cdots, \lfloor\frac{\Delta^2}{max-min}\rceil\}^{N_k}$.

B. Encryption and Distributed ADMM Privacy Computation

Note that the elements of $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\mathbf{z}_k^{(t)}$, $-\mathbf{v}_k^{(t)}$ are quantized to the positive integers, then the Paillier encryption and decryption algorithm can be used for the parallel collaborative ADMM privacy computing, where the key generation at the master node is presented as follows:

- Choose two large prime numbers p, q.
- Computing $n = pq, \epsilon = \mathrm{lcm}(p-1, q-1)$.
- Select an integer $g \in \mathbb{Z}^*_{n^2}, \gcd(r, n) = 1$.
- Select an integer $r \in \mathbb{Z}^*_n, \gcd(r, n) = 1$.
- Define the function $\mathrm{L}(x) = \frac{x-1}{n}$.
- Computing $\mu = (\mathrm{L}(g^\epsilon \bmod n^2))^{-1} \bmod n$.

where $\mathrm{lcm}(a, b)$ denotes the least common multiple of a and b, and $\gcd(a, b)$ denotes the greatest common divisor of a and b. The public key is (n, g) and the private key is (ϵ, μ).

At the master node, the quantized $\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))$, $\Gamma_2(\mathbf{z}_k^{(t)})$, $\Gamma_2(-\mathbf{v}_k^{(t)})$ in (11) and (12) are encrypted as follows:

$$\hat{\boldsymbol{\alpha}}_k = f_{en}(\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))) = g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))} r^n \bmod n^2, \tag{15a}$$

$$\hat{\mathbf{z}}_k^{(t)} = f_{en}(\Gamma_2(\mathbf{z}_k^{(t)})) = g^{\Gamma_2(\mathbf{z}_k^{(t)})} r^n \bmod n^2, \tag{15b}$$

$$-\hat{\mathbf{v}}_k^{(t)} = f_{en}(\Gamma_2(-\mathbf{v}_k^{(t)})) = g^{\Gamma_2(-\mathbf{v}_k^{(t)})} r^n \bmod n^2, \tag{15c}$$

where r denotes distinct random masks, and for brevity, r is consistently used to represent them in this context.

It is important to point out that the Paillier algorithm is considered as a encryption function in (11) and (12) due to the homomorphic operations of ciphertexts are involved in (13). However, the quantization error will impact the estimation accuracy of $\hat{\mathbf{x}}_k^{(t)}$ in (13), which is calculated based on ciphertext operations. In particular, calculation of $\hat{\mathbf{x}}_k^{(t)}$ in (13) consists of the addition computation and the multiply-by-a-constant computation. Then we have the following homomorphism definitions of the parallel privacy computing phase of proposed 3P-ADMM-PC2 scheme.

Definition 1: For any two ciphertexts $\mathbf{c}_1 = f_{en}(\Gamma_2(\mathbf{z}_k^{(t)}))$, $\mathbf{c}_2 = f_{en}(\Gamma_2(-\mathbf{v}_k^{(t)}))$, the addition operation $\oplus$ between ciphertexts is defined as

$$\begin{aligned} f_{en}(\Gamma_2(\mathbf{z}_k^{(t)})) \oplus f_{en}(\Gamma_2(-\mathbf{v}_k^{(t)})) &= \mathbf{c}_1\mathbf{c}_2 \bmod n^2 \\ &= f_{en}(\Gamma_2(\mathbf{z}_k^{(t)}) + \Gamma_2(-\mathbf{v}_k^{(t)}) \bmod n). \end{aligned} \tag{16}$$

Definition 2: For $\Gamma_2(\bar{\mathbf{B}}_k), \mathbf{c} = f_{en}(\Gamma_2(\mathbf{z}_k^{(t)}) + \Gamma_2(-\mathbf{v}_k^{(t)}))$, the multiplication operation is defined as

$$\begin{aligned} \Gamma_2(\bar{\mathbf{B}}_k) \otimes f_{en}(\Gamma_2(\mathbf{z}_k^{(t)}) + \Gamma_2(-\mathbf{v}_k^{(t)})) &= \mathbf{c}^{\Gamma_2(\bar{\mathbf{B}}_k)} \bmod n^2 \\ = f_{en}(\Gamma_2(\bar{\mathbf{B}}_k) \cdot (\Gamma_2(\mathbf{z}_k^{(t)}) + \Gamma_2(-\mathbf{v}_k^{(t)})) \bmod n). \end{aligned} \tag{17}$$

Based on Definition 1 and Definition 2, $\hat{\mathbf{x}}_k^{(t)}$ in (13) is calculated as

$$\begin{aligned} &\hat{\mathbf{x}}_k^{(t)} = \hat{\boldsymbol{\alpha}}_k \oplus \Gamma_2(\bar{\mathbf{B}}_k) \otimes (\hat{\mathbf{z}}_k^{(t-1)} \oplus (-\hat{\mathbf{v}}_k^{(t-1)})) \\ &= \hat{\boldsymbol{\alpha}}_k \oplus (f_{en}(\Gamma_2(\bar{\mathbf{B}}_k) \cdot (\Gamma_2(\mathbf{z}_k^{(t)}) + \Gamma_2(-\mathbf{v}_k^{(t)})) \bmod n)) \\ &= f_{en}(\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) + \Gamma_2(\bar{\mathbf{B}}_k)(\Gamma_2(\mathbf{z}_k^{(t)}) \\ &+ \Gamma_2(-\mathbf{v}_k^{(t)})) \bmod n). \end{aligned} \tag{18}$$

It is worth noting that the information for encryption in (18) is given by

$$\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) + \Gamma_2(\bar{\mathbf{B}}_k)(\Gamma_2(\mathbf{z}_k^{(t)}) + \Gamma_2(-\mathbf{v}_k^{(t)})) \bmod n \tag{19}$$

containing two additions and one multiplication. Note that the elements of $\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))$, $\Gamma_2(\bar{\mathbf{B}}_k)$, $\Gamma_2(\mathbf{z}_k^{(t)})$, $\Gamma_2(-\mathbf{v}_k^{(t)})$ are nonnegative integers. The elements of $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\bar{\mathbf{B}}_k$, $\mathbf{z}_k^{(t)}$, $-\mathbf{v}_k^{(t)}$ are real numbers. To evaluate the estimation performance of decryption and inverse quantization in (18), we have the following theorem, which simplifies $\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})$, $\bar{\mathbf{B}}_k$, $\mathbf{z}_k^{(t)}$, $-\mathbf{v}_k^{(t)}$ to $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \in \mathbb{R}^N, \mathbf{B} \in \mathbb{R}^{N\times N}$.

Theorem 1: For quantization $\Gamma_1, \Gamma_2, \Gamma_3$, and $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \in \mathbb{R}^N, \mathbf{B} \in \mathbb{R}^{N\times N}$, the quantization calculation in (19) can be given by

$$\begin{aligned} &\Gamma_1(\mathbf{u}_3) + \Gamma_2(\mathbf{B})(\Gamma_2(\mathbf{u}_1) + \Gamma_2(\mathbf{u}_2)) = \\ &\left\lfloor \Delta^2 \frac{\mathbf{u}_3 + \mathbf{B}(\mathbf{u}_1 + \mathbf{u}_2) - (2\mathbf{B}\mathbf{1}_N + \mathbf{u}_1 + \mathbf{u}_2 + 1)z_{\min}}{(z_{\max} - z_{\min})^2} \right. \\ &\left. + \Delta^2 \frac{2z_{\min}^2 \mathbf{1}_N}{(z_{\max} - z_{\min})^2} \right\rfloor, \end{aligned} \tag{20}$$

which can be further approximate by

$$\begin{aligned} &\mathbf{u}_3 + \mathbf{B}(\mathbf{u}_1 + \mathbf{u}_2) \approx \\ &\frac{(\Gamma_1(\mathbf{u}_3) + \Gamma_2(\mathbf{B})(\Gamma_2(\mathbf{u}_1) + \Gamma_2(\mathbf{u}_2)))(z_{\max} - z_{\min})^2}{\Delta^2} \\ &+ (2\mathbf{B}\mathbf{1}_N + \mathbf{u}_1 + \mathbf{u}_2 + 1)z_{\min} - 2z_{\min}^2 \mathbf{1}_N. \end{aligned} \tag{21}$$

Proof: Without loss of generality, we use the close set $[z_{\min}, z_{\max}]$ to approximate the open set $\mathbb{R}$. Then we have

$$[z_{\min}, z_{\max}]^N \subset \mathbb{R}^N, \tag{22}$$

$$[z_{\min}, z_{\max}]^N \times [z_{\min}, z_{\max}]^N \subset \mathbb{R}^{N\times N}. \tag{23}$$

The quantization of $\mathbf{u}_1 \in \mathbb{R}^N$ can be expressed as

$$\Gamma_2(\mathbf{u}_1) = \left\lfloor \Delta \frac{\mathbf{u}_1 - z_{\min}\mathbf{1}_N}{z_{\max} - z_{\min}} \right\rfloor, \tag{24}$$

the quantization of $\mathbf{u}_2 \in \mathbb{R}^N$ and $\mathbf{B} \in \mathbb{R}^{M\times N}$ is the same as that of $\mathbf{u}_1$. The quantization of $\mathbf{u}_3 \in \mathbb{R}^N$ can be expressed as

$$\Gamma_1(\mathbf{u}_3) = \left\lfloor \Delta^2 \frac{\mathbf{u}_3 - z_{\min}\mathbf{1}_N}{(z_{\max} - z_{\min})^2} \right\rfloor. \tag{25}$$

The quantized $\Gamma_2(\mathbf{u}_1)$, $\Gamma_2(\mathbf{u}_2)$ perform an addition operation, resulting in

$$\Gamma_2(\mathbf{u}_1) + \Gamma_2(\mathbf{u}_2) = \left\lfloor \Delta \frac{\mathbf{u}_1 + \mathbf{u}_2 - 2z_{\min}\mathbf{1}_N}{z_{\max} - z_{\min}} \right\rfloor. \tag{26}$$

Then, the quantized $\Gamma_2(\mathbf{B})$ performs the multiplication operation with (26), we have

$$\begin{aligned} &\Gamma_2(\mathbf{B})(\Gamma_2(\mathbf{u}_1) + \Gamma_2(\mathbf{u}_2)) = \\ &\left\lfloor \Delta^2 \frac{\mathbf{B}(\mathbf{u}_1 + \mathbf{u}_2) - (2\mathbf{B}\mathbf{1}_N + \mathbf{u}_1 + \mathbf{u}_2)z_{\min} + 2z_{\min}^2 \mathbf{1}_N}{(z_{\max} - z_{\min})^2} \right\rfloor. \end{aligned} \tag{27}$$

Then, the quantized $\Gamma_1(\mathbf{u}_3)$ performs the addition operation with (27), resulting in (20). Furthermore, the result in (21) can be obtained by (20). ■

The obtained results in Theorem 1 reveal that the quantized positive integers do not affect the homomorphic operation of Paillier encryption and the exact value of the original $\mathbf{u}_3 + \mathbf{B}(\mathbf{u}_1 + \mathbf{u}_2)$ can be separated from the computational result in (20). Especially, the proposed quantization method can be used in conjunction with the Paillier homomorphic encryption

scheme, which can be applied in iterative computations under privacy protection. According to Theorem 1, quantizing (13) results in

$$\begin{aligned}\Gamma(\mathbf{x}_k^{(t)}) &= \Gamma_1\left(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})\right) \\ &\quad + \Gamma_2(\bar{\mathbf{B}}_k)\left(\Gamma_2(\mathbf{z}_k^{(t-1)}) + \Gamma_2(-\mathbf{v}_k^{(t-1)})\right) \\ &= \Bigg\lfloor \Delta^2 \frac{\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}) - z_{\min}\mathbf{1}_N}{(z_{\max}-z_{\min})^2} \\ &\quad + \Delta^2 \frac{(\bar{\mathbf{B}}_k - z_{\min}\mathbf{1}_N)(\mathbf{z}_k^{(t-1)} - \mathbf{v}_k^{(t-1)} - 2z_{\min}\mathbf{1}_N)}{(z_{\max}-z_{\min})^2} \Bigg\rceil. \end{aligned} \tag{28}$$

C. Decryption and Inverse Quantization

In each iteration round, the edge node simultaneously sends $\hat{\mathbf{x}}_k^{(t)}$ to the master node, which can be decrypted as

$$\Gamma(\mathbf{x_k^{(t)}}) = \mathrm{L}((\hat{\mathbf{x}}_k^{(t)})^{\epsilon} \bmod n^2)\mu \bmod n. \tag{29}$$

According to (21), the inverse quantization of $\Gamma(\mathbf{x_k^{(t)}})$ is given by

$$\begin{aligned}\mathbf{x}_k^{(t)} &\approx \frac{\Gamma(\mathbf{x}_k^{(t)})}{\Delta^2}(z_{\max}-z_{\min})^2 \\ &+ [2\bar{\mathbf{B}}_k\mathbf{1}_N + (\mathbf{z}_k^{(t-1)} - \mathbf{v}_k^{(t-1)}) + 1]z_{\min} - 2z_{\min}^2\mathbf{1}_N. \end{aligned} \tag{30}$$

Remark 2: The quantization loss depends on Δ when other parameters are unchanged, which reduces with the increase of Δ. For example, the loss of quantization is 10^{-15} for a Δ that can be represented in 45 bits. Besides, selecting a larger key for encryption and decryption is aimed at ensuring security, while a larger Δ of quantization also reduces the precision loss. However, the computational overhead required for modular exponentiation in (15)-(19) with larger Δ increases due to both key and plaintext are larger integers.

Remark 3: For a security Δ, the maximum plaintext length is 49 bits. Treating all plaintexts as 49 bits, the homomorphic computation based on (10a) yields a result with a maximum length of $99+\log_2 N_k$ bits. Thus the iterative computation will not be overflowed from $[0, \mathrm{n})$.

With the obtained $\mathbf{x}_k^{(t-1)}$, the master note updates $\mathbf{z}^{(t+1)}$ and $\mathbf{v}^{(t+1)}$ in (10b) and (10c), respectively, i.e.,

$$\mathbf{z}^{(t+1)} = S_{\frac{\lambda}{\rho}}\left\{\mathbf{v}^{(t)} + \Gamma^{(-1)}(f_{de}(\mathbf{x})^{(t)})\right\}, \tag{31}$$

$$\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} + \Gamma^{(-1)}(f_{de}(\mathbf{x})^{(t)}) - \mathbf{z}^{(t)}, \tag{32}$$

where $\mathbf{x} = (\mathbf{x}_1^T, \mathbf{x}^T, \cdots, \mathbf{x}_K^T)^T$, $\Gamma^{(-1)}(\cdot)$ is the inverse quantization function and $f_{de}(\cdot)$ is the decryption function.

Finally, the proposed 3P-ADMM-PC2 algorithm is summarized in Algorithm 1. However, there exist numerous computations of encryption and decryption involving ModExp of large integers, which constitute the primary computational overhead.

Algorithm 1 Proposed 3P-ADMM-PC2

Input: $\mathbf{A} \in \mathbf{R}^{M\times N}, \mathbf{y} \in \mathbf{R}^M$
Output: $\mathbf{x}^{(t)}$

1: Initializing $\mathbf{x}, \mathbf{z}, \mathbf{v}$.
2: Selection of appropriate ρ, λ, Δ.
3: Specifying the minimum and maximum values of the quantizing.
4: Generating private $key(p, q)$.
5: $n \leftarrow pq$.
6: Generating g, r.
7: $\epsilon \leftarrow lcm(p-1, q-1)$.
8: $\mu \leftarrow (\mathrm{L}(g^{\epsilon} \bmod n^2))^{-1} \bmod n$.
9: Master node : Spliting $\mathbf{A}$ by column; Sending α_k, minimum and maximum values, ρ, Δ to edge node k.
10: Edge node k : Computing $\mathbf{B}_k$; Sends $\mathbf{B}_k$ to master node.
11: Master node : Computing $\hat{\boldsymbol{\alpha}}_k$; Sends $\hat{\boldsymbol{\alpha}}_k$ to edge node k. Edge node k: Quantizes $\mathbf{B}_k\rho$ as $\bar{\mathbf{B}}_k$.
The first iteration is the same as the calculation in the loop
12: **for** $t = 1$ **to** $iter_{max}$ **do**
13: Edge node k: Sending $\hat{\mathbf{x}}_k^{(t-1)}$ to master node.
Master node: Sending $\hat{\mathbf{z}}_k^{(t-1)}$, $-\hat{\mathbf{v}}_k^{(t-1)}$ to edge node k.
14: Edge node k: Computing $\hat{\mathbf{x}}_k^{(t)} \leftarrow \hat{\boldsymbol{\alpha}}_k \oplus \Gamma(\bar{\mathbf{B}}_k) \otimes (\hat{\mathbf{z}}_k^{(t-1)} \oplus (-\hat{\mathbf{v}}_k^{(t-1)}))$.
Master node: Decrypting and inverse quantizing $\hat{\mathbf{x}}_k^{(t)}$, and splicing $\hat{\mathbf{x}}_k^{(t)}$ to get $\mathbf{x}^{(t-1)}$, then computing $\mathbf{z}^{(t)} \leftarrow S_{\frac{\lambda}{\rho}}(\mathbf{v}^{(t-1)} + \mathbf{x}^{(t-1)})$ and $\mathbf{v}^{(t)} \leftarrow \mathbf{v}^{(t-1)} + \mathbf{x}^{(t-1)} - \mathbf{z}^{(t)}$.
15: **end for**
16: **return** $(\mathbf{x}^{(t)})$

IV. Adaptive GPU Accelerated 3P-ADMM-PC2

In this section, we propose an adaptive GPU accelerated 3P-ADMM-PC2 for large key space, which has at least 1024 bits key length. The encryption, the decryption and the homomorphic operations of proposed 3P-ADMM-PC2 refer to ModExp in large key space. In particular, the ModExp computation task in large key space is first decomposed into multiple ModExp computation subtasks in small key spaces. Then, the computational subtasks in smaller space are distributed to multiple streaming multiprocessors (SM) of GPU, which are processed in parallel. In addition, the ModExp of each subtask in SM is translated into integer multiplication and shift operation with adaptive bit width.

To decompose the ModExp of proposed 3P-ADMM-PC2 in large key space into multiple subtask in smaller spaces, the following lemmas are considered.

Lemma 1: In a system of univariate linear congruences, if the moduli are pairwise coprime, then for any integers, the system admits a solution, and the general solution is given by [41]

$$x_{CRT} = \sum_{i=1}^{n} a_i t_i M_i \bmod M. \tag{33}$$

Lemma 2: If p^2, q^2 are mutually prime, we have [42]

$$(q^2)^{-1} \pmod{p^2} q^2 + (p^2)^{-1} \pmod{q^2} p^2 \bmod n^2 = 1. \tag{34}$$

By leveraging the Lemma 1, the computations in the encryption process (EP) performed in the large space n^2, are decomposed into two smaller spaces p^2, q^2. Consequently, the calculations for (15) are optimized as follows:
$\mathbb{Z}_{n^2} \rightarrow \mathbb{Z}_{p^2 \times q^2}$:

$$g^{'} = g \bmod p^2, \tag{35a}$$
$$g^{''} = g \bmod q^2, \tag{35b}$$
$$\Gamma_1^{'}(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) = \Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) \bmod \varphi(p^2), \tag{35c}$$
$$\Gamma_1^{''}(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) = \Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y})) \bmod \varphi(q^2), \tag{35d}$$
$$\Gamma_2^{'}(\mathbf{z}_k^{(t)}) = \Gamma_2(\mathbf{z}_k^{(t)}) \bmod \varphi(p^2), \tag{35e}$$
$$\Gamma_2^{''}(\mathbf{z}_k^{(t)}) = \Gamma_2(\mathbf{z}_k^{(t)}) \bmod \varphi(q^2), \tag{35f}$$
$$\Gamma_2^{'}(-\mathbf{v}_k^{(t)}) = \Gamma_2(-\mathbf{v}_k^{(t)}) \bmod \varphi(p^2), \tag{35g}$$
$$\Gamma_2^{''}(-\mathbf{v}_k^{(t)}) = \Gamma_2(-\mathbf{v}_k^{(t)}) \bmod \varphi(p^2), \tag{35h}$$

where $\varphi(\cdot)$ is the Euler's totient function. The corresponding calculation results are given by

$$\left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{'} = g^{'\Gamma_1^{'}(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))} \bmod p^2, \tag{36a}$$
$$\left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{''} = g^{''\Gamma_1^{''}(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))} \bmod q^2, \tag{36b}$$
$$\left(g^{\Gamma_2(\mathbf{z}_k^{(t)})}\right)^{'} = g^{'\Gamma_2^{'}(\mathbf{z}_k^{(t)})} \bmod p^2, \tag{36c}$$
$$\left(g^{\Gamma_2(\mathbf{z}_k^{(t)})}\right)^{''} = g^{''\Gamma_2^{''}(\mathbf{z}_k^{(t)})} \bmod q^2, \tag{36d}$$
$$\left(g^{\Gamma_2(-\mathbf{v}_k^{(t)})}\right)^{'} = g^{'\Gamma_2^{'}(-\mathbf{v}_k^{(t)})} \bmod p^2, \tag{36e}$$
$$\left(g^{\Gamma_2(-\mathbf{v}_k^{(t)})}\right)^{''} = g^{''\Gamma_2^{''}(-\mathbf{v}_k^{(t)})} \bmod q^2. \tag{36f}$$

According to Lemma 1, we have

$$\begin{aligned} g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))} \bmod n^2 = &\left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{'} (q^2)^{-1}(\bmod p^2) q^2 \\ &+ \left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{''} (p^2)^{-1}(\bmod q^2) p^2. \end{aligned} \tag{37}$$

Using Lemma 2, (37) is further expressed as

$$\begin{aligned} g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))} \bmod n^2 = &\left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{'} + [\left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{''} \\ &- \left(g^{\Gamma_1(\mathbf{B}_k(\mathbf{A}_k^T\mathbf{y}))}\right)^{'}](p^2)^{-1}(\bmod q^2) p^2. \end{aligned} \tag{38}$$

Similarly, (15b) and (15c) can be further calculated by

$$\begin{aligned} (g^{\Gamma_2(\mathbf{z}_k^{(t)})}) \bmod n^2 = &(g^{\Gamma_2(\mathbf{z}_k^{(t)})})^{'} + [(g^{\Gamma_2(\mathbf{z}_k^{(t)})})^{''} \\ &- (g^{\Gamma_2(\mathbf{z}_k^{(t)})})^{'}](p^2)^{-1}(\bmod q^2) p^2, \end{aligned} \tag{39}$$

$$\begin{aligned} (g^{\Gamma_2(-\mathbf{v}_k^{(t)})}) \bmod n^2 = &(g^{\Gamma_2(-\mathbf{v}_k^{(t)})})^{'} + [(g^{\Gamma_2(-\mathbf{v}_k^{(t)})})^{''} \\ &- (g^{\Gamma_2(-\mathbf{v}_k^{(t)})})^{'}](p^2)^{-1}(\bmod q^2) p^2. \end{aligned} \tag{40}$$

Furthermore, the EP calculations for μ and $\mathbf{c}^{\epsilon} \bmod n^2$ in the decryption phase can also be accelerated similar to that in encryption phase.

After optimizing the ModExp in the encryption and decryption processes, the computation overhead is reduced, and computations on the two smaller key spaces $\mathbb{Z}_{p^2}, \mathbb{Z}_{q^2}$, can be performed in parallel. However, the EP in large key space $\mathbb{Z}_{n^2}$ is decomposed into the smaller key spaces $\mathbb{Z}_{p^2}, \mathbb{Z}_{q^2}$, practical implementation remains challenging due to the limited computational power of edge nodes. For example, the key space $\mathbb{Z}_{n^2}^{4096bits}$ is merely decomposed into smaller spaces $\mathbb{Z}_{p^2}^{2048bits}, \mathbb{Z}_{q^2}^{2048bits}$ for a 2048 bits length key, which are still difficult to handle. In addition, the EP computations for a plaintext vector encrypting or decrypting always are processed one by one, increasing the computational overhead.

A. Parallel Computing in Streaming Multiprocessor

Due to the limited number of CPU cores, ModExp operations in large key spaces exhibit low parallelism on CPU. To implement in the hardware devices with high parallelism, such as GPU or FPGA, ModExp operations should be optimized due to each GPU core supports only 32 bits or 64 bits operations. Note that the EP operation mainly includes multiplication of large integers and modular operations. Thus we present the GPU-accelerated deployment process of the proposed 3P-ADMM-PC2 scheme as follows.

1) Multiplication of Large Integers: The decimal large integers can be expressed by the corresponding polynomial coefficient vectors, i.e.,

$$Int_{1,(10)} \rightarrow Vector_1 = [Vec_{1,L-1}, ..., Vec_{1,0}]_{(base)}, \tag{41}$$

$$Int_{2,(10)} \rightarrow Vector_2 = [Vec_{2,L-1}, ..., Vec_{2,0}]_{(base)}. \tag{42}$$

Then the product of two numbers can be written as

$$\begin{aligned} Int_{1,(10)} \cdot Int_{2,(10)} &\Leftrightarrow Vector_1 \cdot Vector_2 \\ &= \sum_{i+j=k} Vec_{1,i} Vec_{2,j}, \end{aligned} \tag{43}$$

where $Vec_{1,i}, Vec_{2,j} \in \{0, 1\}, k = 0, 1, ..., 2L-2$.

Different bases can be chosen to allow multiple GPU cores to compute the product of each bit in parallel, and the results are stored in shared memory. Multiple computational tasks can even be assigned to a single GPU core, which stores the results in shared memory after computation to reduce access frequency and thus decrease computation time. Note that the multiplication of $Int_{1,(10)} \times Int_{2,(10)}$ at one base is equivalent to the multiplication of two polynomials, with $Vec_{1,i}, Vec_{2,i}, i = 0, 1, ..., L-1$ representing the coefficients of the respective polynomials. To further reduce the complexity, $Vec_{1,i}$ and $Vec_{2,i}$ are transformed into point-value representations using FFT, and performs dot product. The obtained results are subsequently converted back to coefficient representation using IFFT, i.e.,

$$Vec_{i,FFT} = FFT([Vec_{i,L-1}, ..., Vec_{i,0}]_{(base)}), i = 1, 2, \tag{44}$$

$$Vec_{res}[k]_{FFT} = Vec_{1,k} Vec_{2,k}, \tag{45}$$

$$Vec_{res}[k] = IFFT(Vec_{res}[k]_{FFT}), k = 0, ..., L-1. \tag{46}$$

In the computation of FFT and IFFT, ξ serves as a primitive root on the cyclotomic polynomial, where $\xi = e^{-\frac{2\pi i}{L}}$, i denotes the imaginary unit, and L must be a power of 2.

Algorithm 2 Proposed GPU-accelerated EP Computation

Input: $g, m_{base2}, n^2_{base}, n^2$.
Output: $g^m \bmod n^2$.

$T_{base} \leftarrow 1$
$length \leftarrow len(n^2_{base})$
$R \leftarrow \lfloor \frac{1<<2length}{n^2} \rfloor$
$R_{base\_FFT} \leftarrow FFT(R_{base})$
$n^2_{base\_FFT} \leftarrow FFT(n^2_{base})$
for $i = 0$ **to** $len(m_{base2})$ **do**
  if $m_{base2\_i} == 1$ **then**
    $T_{base\_FFT} \leftarrow FFT(T_{base})$
    $g_{base\_FFT} \leftarrow FFT(g_{base})$
    $a_{base\_FFT} \leftarrow T_{base\_FFT} \times g_{base\_FFT}$
    $Q_{base} \leftarrow IFFT(a_{base\_FFT} \times R_{base\_FFT}) >> 2length$
    $T_{base} \leftarrow IFFT(a_{base\_FFT} - FFT(Q_{base}) \times n^2_{base\_FFT})$
    if $T_{base} >= n^2_{base}$ **then**
      $T_{base} \leftarrow T_{base} - n^2_{base}$
    end if
  end if
  $g_{base\_FFT} \leftarrow FFT(g_{base})$
  $b_{base\_FFT} \leftarrow g_{base\_FFT} \times g_{base\_FFT}$
  $P_{base} \leftarrow IFFT(b_{base\_FFT} \times R_{base\_FFT}) >> 2length$
  $g_{base} \leftarrow IFFT(b_{base\_FFT} - FFT(P_{base}) \times n^2_{base\_FFT})$
  if $g_{base} >= n^2_{base}$ **then**
    $g_{base} \leftarrow g_{base} - n^2_{base}$
  end if
end for
return $(T_{base} \rightarrow T_{10})$ ▷ Decimal conversion

Remark 4: To ensure that L is a power of 2, the base can be chosen as 2, 4, 16, 256, 65536. Then, a 4096 bits large integer will be converted into vectors of lengths 4096, 2048, 1024, 512, 256, respectively. Furthermore, the dimensions of the Discrete Fourier Transform (DFT) matrix used in the subsequent FFT process are determined. And the inversion operation of DFT matrix for IFFT is performed only once.

B. ModExp Computing Task Decomposition

1) Modular Operation: To deploy ModExp in GPU, it is necessary to address the issue of overflow beyond the bit width of CUDA cores. Barrett Reduction replaces division with multiplication and shift operations. However, Barrett Reduction also involves large integer multiplication in its intermediate steps, which similarly overflowing the CUDA cores bit width. Based on aforementioned large integer multiplication, the large integer ModExp is optimized and decomposed. And our proposed GPU-accelerated EP computation is summarized in Algorithm 2. The detailed computational architecture is illustrated in Fig. 2.

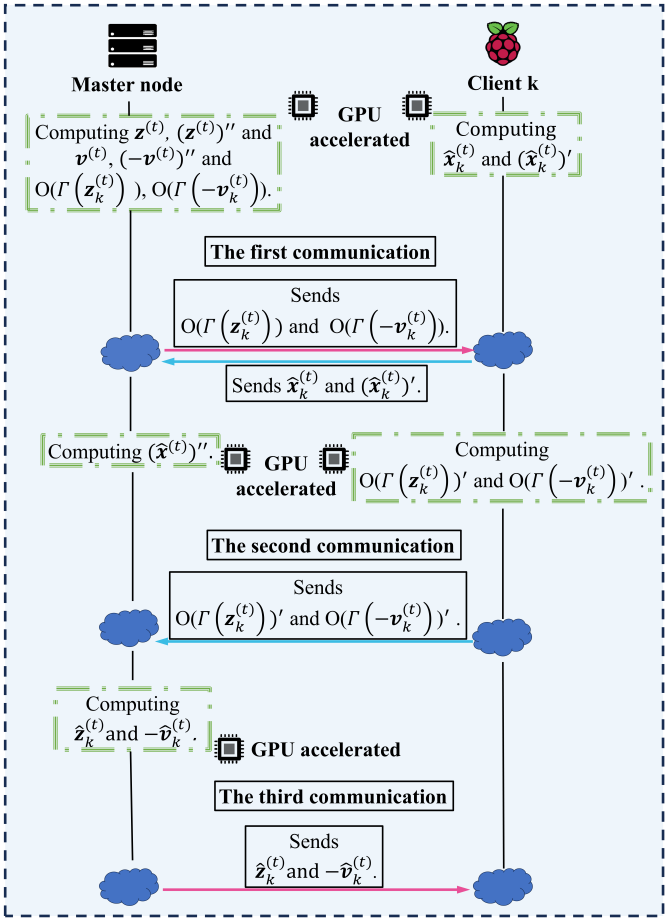


Fig. 3: The illustration of proposed GPU-accelerated 3P-ADMM-PC2.

C. Proposed GPU-accelerated 3P-ADMM-PC2

Although GPU acceleration can be employed to expedite the encryption and decryption of plaintext vectors, the master node bears a substantial computational burden. Consequently, we optimize the compute burden of distributed nodes by sacrificing some communication overhead. A GPU-accelerated 3P-ADMM-PC2 algorithm is proposed, which involves leveraging edge nodes to collaboratively encrypt and decrypt data with the master node. During each iteration, each edge node communicates with the master node three times, exchanging different parameters, achieving a computation overhead balance for collaborative encryption and decryption, as shown in Fig. 3.

Note that the calculation on large space $\mathbb{Z}_{n^2}$ can be decomposed into two smaller spaces $\mathbb{Z}_{p^2}, \mathbb{Z}_{q^2}$, and the results from these smaller spaces can be linearly combined back to the large space through Lemma 1 and Lemma 2, enabling parallel computations across two smaller spaces. Consequently, in our proposed GPU-accelerated 3P-ADMM-PC2 algorithm, the computations on $\mathbb{Z}_{q^2}$ are executed by the master node, while those on $\mathbb{Z}_{p^2}$ are handled by the edge nodes. For one complete iteration of computation, the master node communicates with edge node k three times. In the initialization parameter phase, the master node sends p^2 and $\varphi(p^2)$ to edge node k for subsequent computations.

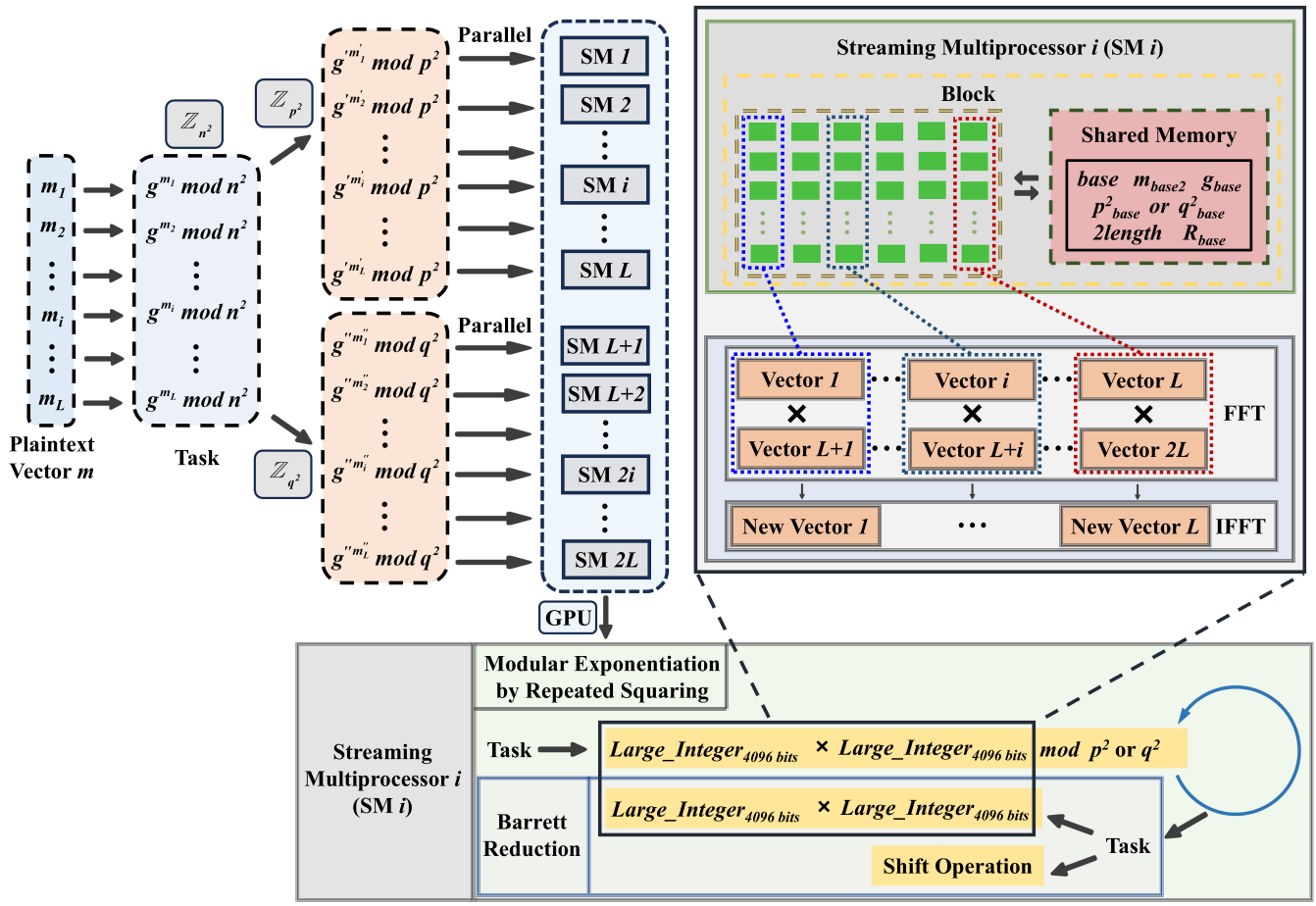


Fig. 2: GPU-accelerated ModExp computation in large key space.

After $\hat{\mathbf{x}}_k^{(t)}$ and $\mathbf{z}^{(t)}$, $\mathbf{v}^{(t)}$ are computed by the edge node k and the master node, the edge node k and the master node then compute $(\hat{\mathbf{x}}_k^{(t)})' = \hat{\mathbf{x}}_k^{(t)} \bmod p^2$ and $(\mathbf{z}^{(t)})'' = \Gamma(\mathbf{z}^{(t)}) \bmod \varphi(q^2), (-\mathbf{v}^{(t)})'' = \Gamma(-\mathbf{v}^{(t)}) \bmod \varphi(q^2)$. Three communication rounds of our proposed GPU-accelerated 3P-ADMM-PC2 algorithm can be presented as follows:

1) The First Communication Round: The master node sends $\mathbf{O}(\Gamma(\mathbf{z}_k^{(t)})), \mathbf{O}(\Gamma(-\mathbf{v}_k^{(t)}))$ to edge node k, where $\mathbf{O}(\cdot)$ is the obfuscation function. To ensure the privacy of the data, master node allows the edge nodes k to collaborate in the encryption, the data needs to be obfuscated and sent. At the same time, the edge node k sends $\hat{\mathbf{x}}_k^{(t)}, (\hat{\mathbf{x}}_k^{(t)})'$ to the master node.

After the first communication round, master node computes $(\hat{\mathbf{x}}_k^{(t)})'' = \hat{\mathbf{x}}_k^{(t)} \bmod q^2$, and then computes $\mathbf{x}_k^{(t)}$ according to Lemma 1. Edge node k computes $(\mathbf{O}(\Gamma(\mathbf{z}_k^{(t)})))' = \mathbf{O}(\Gamma(\mathbf{z}_k^{(t)})) \bmod \varphi(p^2), (\mathbf{O}(\Gamma(-\mathbf{v}_k^{(t)})))' = \mathbf{O}(\Gamma(-\mathbf{v}_k^{(t)})) \bmod \varphi(p^2)$.

2) The Second Communication Round: Edge node k sends $(\mathbf{O}(\Gamma(\mathbf{z}_k^{(t)})))', (\mathbf{O}(\Gamma(-\mathbf{v}_k^{(t)})))'$ to master node.

After the second communication round, master node calculates $\hat{\mathbf{z}}_k, -\hat{\mathbf{v}}_k$. according to (39) and (40).

3) The Third Communication Round: The master node sends $\hat{\mathbf{z}}_k, -\hat{\mathbf{v}}_k$ to edge node k.

Our proposed GPU-accelerated 3P-ADMM-PC2 algorithm is summarized in Algorithm 3. The large integer ModExp of Algorithm 3 is performed according to Algorithm 2.

Remark 5: The master node has only sent p^2 to the edge nodes in our proposed GPU-accelerated 3P-ADMM-PC2 algorithm, and the edge nodes only know the p^2, $\varphi(p^2)$ without other parameters of the Paillier encryption and decryption. Thus the privacy of the data can still be guaranteed.

D. Complexity and Convergence Analysis

The complexity of Algorithm 3 is primarily determined by each large integer ModExp operation, making its essence the complexity of Algorithm 2. In Algorithm 2, (44)(45)(46) reduces the complexity of the multiplication calculation in (43) from $\mathcal{O}(L^2)$ to $\mathcal{O}(LlogL)$. Ultimately, the overall complexity of Algorithm 2 is optimized to $\mathcal{O}(len(m_{base2})(5LlogL+2L))$, and further approximated to $\mathcal{O}(len(m_{base2})LlogL)$.

According to the convergence theory of approximate ADM-M [43] [44], if the errors are bounded, the algorithm converges to a neighborhood of the optimal solution. In the distributed scenario presented in the paper, matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ is partitioned by columns into $[\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_K]$ based on the total number K of edge nodes. During computation, the data is quantized into positive integers, introducing quantization noise on the order of 10^{-14}. Additionally, the matrix partitioning neglects off-diagonal blocks, which introduces a partitioning

Algorithm 3 Proposed GPU-accelerated 3P-ADMM-PC2

Input: $\mathbf{A} \in \mathbf{R}^{M\times N}, \mathbf{y} \in \mathbf{R}^{M}$
Output: $\mathbf{x}^{(t)}$

1: Initializing $\mathbf{x}, \mathbf{z}, \mathbf{v}$.
2: Selection of appropriate ρ, λ, Δ.
3: Specifying the minimum and maximum values of the quantizing.
4: Generating private $key(p, q)$.
5: $n \leftarrow pq$.
6: Generating g, f, a.
7: $\epsilon \leftarrow lcm(p-1, q-1)$.
8: $\mu \leftarrow (\mathrm{L}(g^{\epsilon} \bmod n^2))^{-1} \bmod n$.
9: Master node: Spliting $\mathbf{A}$ by column; Sending α_k, minimum and maximum values, ρ, Δ to edge node k.
10: Edge node k: Computing $\mathbf{B}_k$; Sending $\mathbf{B}_k$ to master node.
11: Master node: Computing $\hat{\boldsymbol{\alpha}}_k$; Sends $\hat{\boldsymbol{\alpha}}_k$ to edge node k.
Edge node k: Quantizing $\mathbf{B}_k\rho$ as $\bar{\mathbf{B}}_k$.
12: **for** $t = 1$ **to** $iter_{max}$ **do**
13: Edge node k: Sending $\hat{\mathbf{x}}_k^{(t-1)}, (\hat{\mathbf{x}}_k^{(t-1)})'$ to master node.
Master node: Sending $\mathbf{O}(\Gamma(\mathbf{z}^{(t)})), \mathbf{O}(\Gamma(-\mathbf{v}^{(t)}))$ to edge node k.
14: Edge node k: Computing $(\mathbf{O}(\Gamma(\mathbf{z}^{(t)})))'$ and $(\mathbf{O}(\Gamma(-\mathbf{v}^{(t)})))'$.
Master node: Computing $(\hat{\mathbf{x}}_k^{(t-1)})''$ and $\mathbf{x}^{(t-1)}$.
15: Edge node k: Sending $(\mathbf{O}(\Gamma(\mathbf{z}^{(t)})))'$ and $\mathbf{x}^{(t-1)}$ to master node.
16: Master node: Computing $\hat{\mathbf{z}}_k, -\hat{\mathbf{v}}_k$.
17: Master node: Sending $\hat{\mathbf{z}}_k, -\hat{\mathbf{v}}_k$ to edge node k.
18: Edge node k: Computing $\hat{\mathbf{x}}_k^{(t-1)}, (\hat{\mathbf{x}}_k^{(t-1)})'$.
Master node: Computing $\mathbf{z}^{(t)}, (\mathbf{z}^{(t)})''$, $\mathbf{v}^{(t)}, (-\mathbf{v}^{(t)})''$.
19: **end for**
20: **return** $(\mathbf{x}^{(t)})$

error. In practical algorithms, matrix $\mathbf{A}$ is a Gaussian matrix. Its column vectors $\mathbf{a}_i$ and $\mathbf{a}_j, i \neq j$ are random vectors, and their inner product $\mathbf{a}_i^T\mathbf{a}_j$ is a random variable satisfying the following conditions:

$$E[\mathbf{a}_i^T\mathbf{a}_j] = 0, \tag{47}$$

$$\mathrm{Var}(\mathbf{a}_i^T\mathbf{a}_j) = M. \tag{48}$$

where $E[\cdot]$ is expected value and $Var(\cdot)$ is the variance.Therefore, the variance of the normalized inner product $\frac{\mathbf{a}_i^T\mathbf{a}_j}{M}$ is $\frac{1}{M}$. When M is large, the correlation between columns becomes negligible, indicating approximate orthogonality. For the segmentation error, if the columns of matrix A exhibit low correlation, the segmentation error remains bounded. Meanwhile, the quantization noise is on the order of 10 and is stably bounded. Therefore, the overall error remains small and bounded, ensuring the practical convergence of the algorithm.

V. Experimental Results And Analysis

In this section, the proposed 3P-ADMM-PC2 scheme is first deployed in a hardware environment, and then the precision loss of the quantization as well as the mean squared error (MSE) are evaluated and compared. Furthermore, the proposed GPU-accelerated 3P-ADMM-PC2 scheme is detailed analyzed and evaluated. Finally, a typical application example for power network reconstruction is presented.

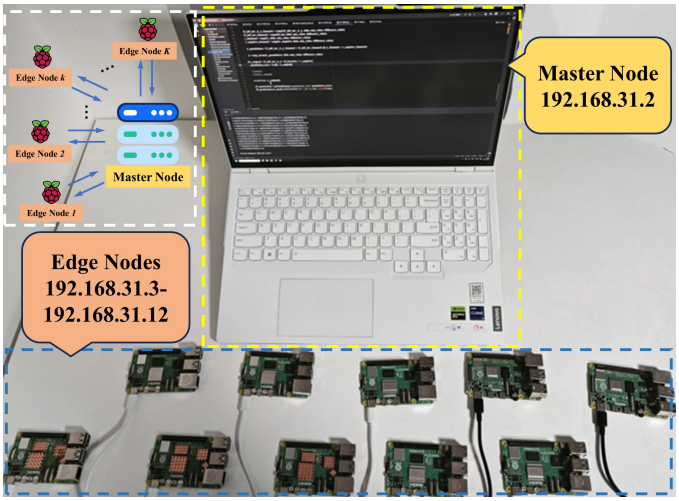


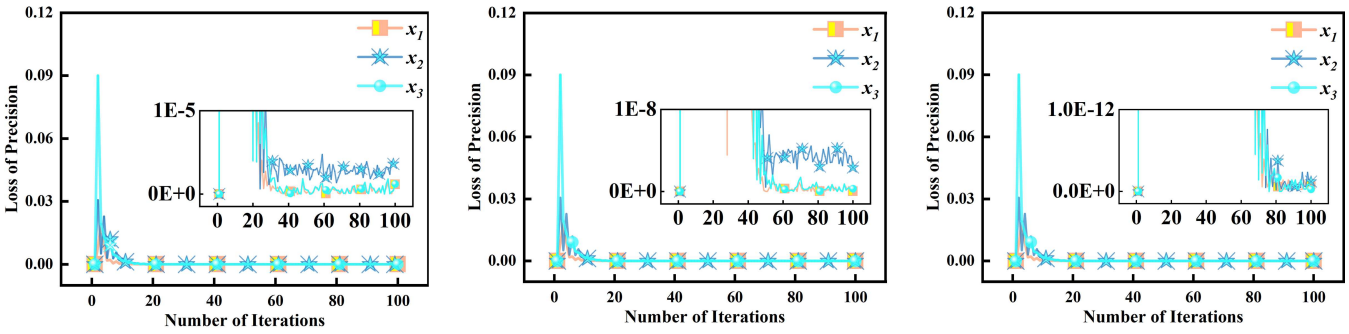
Fig. 4: Hardware configuration and network environment for distributed computing.

As shown in Fig. 4, the hardware environment utilized in the experiments comprises a master node equipped with an Intel Core i9-13900HX CPU and 16GB RAM, along with an NVIDIA RTX 4060 8GB RAM GPU. The edge nodes consist of three Raspberry Pi 5 development boards, where each node is equipped with an Cortex-A76 64-bit SoC, 800MHz VideoCore VII GPU (supporting OpenGL ES 3.1 and Vulkan 1.2), and an 8G LPDDR4X-4267 SDRAM.

A. Quantization Error and MSE of Proposed 3P-ADMM-PC2

Fig. 5 illustrates the quantization precision loss of the proposed 3P-ADMM-PC2. Without loss of generality, we consider $\mathbf{A} \in \mathbb{R}^{3\times 3}, \mathbf{A} \sim \mathcal{CN}(0, 1)$. The precision loss is defined as the absolute value of the difference between the estimates of the unencrypted Dis.-ADMM and proposed 3P-ADMM-PC2 for $\mathbf{x} = (x_1, x_2, x_3)^T$. As shown in Fig. 5, as Δ increases from 10^5 to 10^{15}, the quantization precision loss decreases from 10^{-6} to 10^{-16}. And the quantization precision loss can be approximated as $\frac{1}{10\Delta}$.

In a local area network (LAN), we use the hardware environment in Fig. 4 to deploy distributed computing algorithms. To facilitate a performance comparison with centralized ADMM, we selected the maximum dimension that centralized ADMM can handle, denoted as $\mathbf{A} \in \mathbb{R}^{3000\times 27000}$, $\mathbf{A} \sim \mathcal{CN}(0, 1)$. The computational tasks with $\mathbf{A}_k \in \mathbb{R}^{3000\times 9000}, k = 1, 2, 3$, are assigned to 3 edge nodes. We compare the MSE of the centralized ADMM (Cen.-ADMM), Dis.-ADMM, DP-ADMM, and proposed 3P-ADMM-PC2. In the proposed 3P-ADMM-PC2, the key length is 2048 bits with $\Delta = 10^{15}$. The choice of $\mathbf{A}$'s dimension is based on the dimension of $\mathbf{A}^T\mathbf{A} + \rho\mathbf{I}_N$. When inverting $\mathbf{A}^T\mathbf{A} + \rho\mathbf{I}_N$ with dimension 27000×27000, approximately 14 GB of memory is required, which is the limit of available memory for Cen.-ADMM. In Fig. 6, the Cen.-ADMM shows the best MSE performance.



(a) λ=1, ρ=1, Δ=10^5 (b) λ=1, ρ=1, Δ=10^{10} (c) λ=1, ρ=1, Δ=10^{15}

Fig. 5: Precision loss of the quantization scheme with different Δ values.

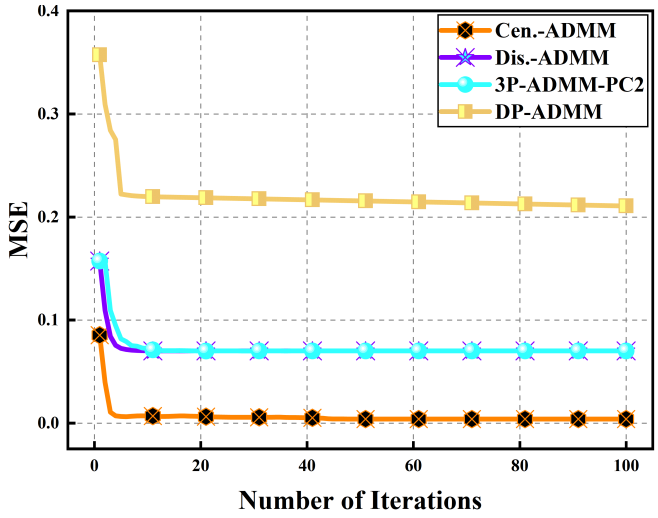
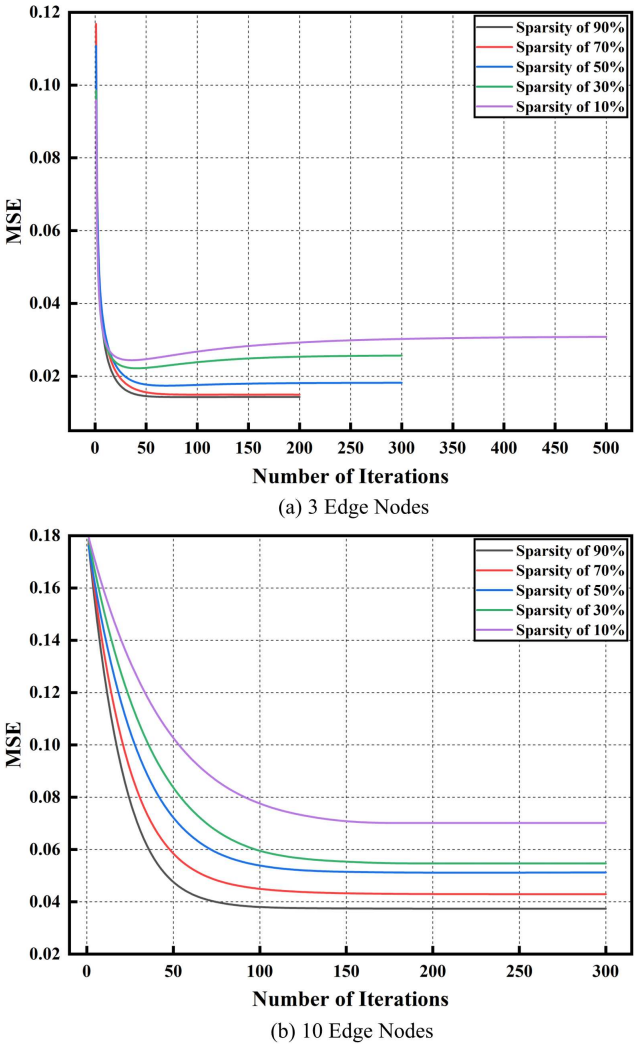


Fig. 6: MSE of various methods.

The Dis.-ADMM has an MSE approximately 0.07 higher than that of Cen.-ADMM. The DP-Dis.-ADMM has an MSE approximately 0.2 higher than that of Cen.-ADMM. The MSE of the proposed 3P-ADMM-PC2 differs from that of the Dis-ADMM by approximately 10^{-14}. The MSEs of both schemes almost overlap, indicating that the quantization precision loss is negligible.

To evaluate the performance of the proposed 3P-ADMM-PC2 under different numbers of edge nodes, we compared the MSE for configurations with 3 and 10 edge nodes. Simultaneously, to accommodate large-scale sparse signal processing, we increased the dimension of $\mathbf{x}^{(t)}$ to $\mathbf{x}^{(t)} \in \mathbb{R}^{65536}$. Accordingly, the dimension of matrix $\mathbf{A}$ was expanded to $\mathbf{A} \in \mathbb{R}^{10000\times 65536}$. The computational tasks $\mathbf{A}_k \in \mathbb{R}^{10000\times 21846}, k = 1, 2, 3$, and $\mathbf{A}_k \in \mathbb{R}^{10000\times 6554}, k = 1, 2, \cdots, 10$, are distributed to 3 and 10 edge nodes, respectively. As shown in Fig. 7, we compared the convergence rate and MSE of the algorithm under conditions of 3 and 10 edge nodes, respectively, at sparsity levels of 10%, 30%, 50%, 70%, and 90% for $\mathbf{x}^{(t)}$ It can be observed that when the sparsity is low, the algorithm converges faster with a lower MSE. As sparsity increases, the convergence speed decreases significantly, and the MSE rises accordingly. This aligns with the characteristics of ADMM, which introduces



(a) 3 Edge Nodes

(b) 10 Edge Nodes

Fig. 7: MSE with 3 and 10 edge nodes under varying sparsity levels.

an L1 regularization term to promote sparsity directly through a soft-thresholding shrinkage operator. When the original data is relatively sparse, the algorithm converges more quickly and approximates the true solution more easily, resulting in smaller errors. The number of edge nodes also impacts the algorithms performance: increasing the number of edge nodes accelerates the overall computation but slightly degrades performance. Under the same sparsity level, the configuration with 3 edge nodes achieves lower errors. In practical applications, it is necessary to balance the requirements of computational speed and accuracy.

B. Performance Evaluation of Proposed GPU-accelerated 3P-ADMM-PC2

1) Throughput: The core computation of proposed GPU-accelerated 3P-ADMM-PC2 is proposed GPU-accelerated EP computation in Algorithm 2, which involves modular multiplication (ModMult) and ModExp calculations for large numbers. Operations Per Second (OPS) is used to demonstrate the throughout of main computations on the master node and edge nodes for proposed GPU-accelerated 3P-ADMM-PC2. TABLE II shows the OPS for the computations with different key lengths, calculated on the CPU and GPU of the master node and edge nodes. Nine thousand unsigned integer plaintexts within the range $[0, n)$ were selected for each trial, repeated ten times, and the averages computed, with the modulus for the modulo operation set to n^2. It is observed that as the length of the public key n increases, the computational load increases, leading to more frequent communication between GPU's threads and a corresponding decrease in throughput.

2) Total Computation Time: From the beginning of iteration to stable convergence, the process can be divided into two phases: pre-computation and iterative calculation. Each iteration includes local computation and communication. Therefore, the total computation time is given by

$$T_{total} = T_{pre} + (T_{loc} + T_{comm}) \times Iterations, \tag{49}$$

where T_{pre} denotes the pre-computation time, T_{loc} and T_{comm}represents the local computation time and communication times for each iteration round. When $\mathbf{A} \in \mathbb{R}^{3000\times27000}$ and 3 edge nodes, Fig. 8 displays T_{pre} and T_{total} for different key lengths using the Cen.-ADMM, Dis.-ADMM, the CPU-Dis.-ADMM with CPU based encryption and decryption, and the proposed GPU-accelerated 3P-ADMM-PC2. Both Cen.-ADMM and Dis.-ADMM do not involve encryption and decryption computations, thus the changes of key lengths do not affect the computation time. The Dis.-ADMM has the shortest T_{pre} and T_{total} among all schemes but lacks privacy protection measures during data exchange. For 2048 bits key length, the difference in T_{total} between the CPU-Dis.-ADMM and the Cen.-ADMM is small. The T_{total} for the CPU-Dis.-ADMM is significantly larger than that for the Cen.-ADMM for 4096 bits key length. Thus, the distributed computing scheme using CPU for encryption and decryption becomes impractical. Moreover, the proposed GPU-accelerated 3P-ADMM-PC2 demonstrates better acceleration effects for T_{total} compared to the Cen.-ADMM across different key lengths. Additionally, the proposed GPU-accelerated 3P-ADMM-PC2 provides greater security than the Dis.-ADMM during data exchange.

3) Calculation Delay: When $\mathbf{A} \in \mathbb{R}^{3000\times27000}$ and 3 edge nodes, Fig. 9 illustrates the waiting time between nodes during iterative calculations in the proposed GPU-accelerated 3P-ADMM-PC2 with different key lengths. Without loss of generality, we analyze the calculation delay for the 30th and the 80th iterations. As shown in Fig. 9 (a), the master node completes local computations at 9427 seconds and 16721 seconds for the 30th and the 80th iterations, respectively. The edge nodes finish their computations approximately at 9439 seconds and 16733 seconds, respectively. The master node waits approximately 12 seconds for the edge nodes to complete each iteration. Edge nodes mutually wait about 1.5 seconds for each other. The waiting time for the master node is primarily due to the difference in hardware conditions between the master node and edge nodes. The waiting time among edge nodes is caused by uneven plaintext lengths. Especially, as the key length increases, the mutual waiting time between nodes also increases.

To further analyze the acceleration performance and feasibility of the proposed 3P-ADMM-PC2 framework, we evaluated the latency characteristics of both GPU and CPU implementations under larger-scale sparse problems and a greater number of edge nodes. When the dimension of matrix $\mathbf{A}$ is increased to $\mathbf{A} \in \mathbb{R}^{10000\times65536}$ with 10 edge nodes, the latency per iteration for both GPU and CPU computations under different key lengths is shown in Table III, IV and V.

As shown in Table III, when the key length is 1024 bits, the total algorithm runtime using GPU-based encryption and decryption is 11697 seconds, while that using CPU-based encryption and decryption is 29858 seconds. When the number of edge nodes is fixed, the dimensionality and data type of the transmitted parameters remain consistent. Therefore, the communication overhead differs only marginally between GPU and CPU implementations, with variations primarily arising from real-time network fluctuations. Due to the superior computational power of the master node compared to the edge nodes, the master completes its current computational task earlier. This results in a waiting latency of approximately 5 seconds when using GPU encryption and decryption, and about 7 seconds when using CPU encryption and decryption. Additionally, due to uneven plaintext lengths and varying real-time hardware loads, the completion time among edge nodes differs by approximately 1 second with GPU encryption and decryption, and about 2 seconds with CPU encryption and decryption. Overall, the computational overhead per iteration using GPU-based encryption and decryption is significantly lower than that of CPU-based encryption and decryption.

As shown in Table IV, when the key length is 2048 bits, the computational overhead per iteration remains lower with GPU-based encryption and decryption than with CPU-based encryption and decryption. Interestingly, although the master node possesses superior computational power compared to the edge nodes, it faces growing challenges as the key length and task dimensionality increase. In each iteration, the edge nodes now complete their tasks earlier than the master node. When

TABLE II: Throughput of CPU and GPU computations on the master node and edge node with different key lengths.

Throughput	ModMult(KOPS)			ModExp(OPS)			GPU-accelerated EP computation(OPS)		
Length of key n (bits)	1024	2048	4096	1024	2048	4096	1024	2048	4096
Matser Node (CPU)	207.95	42.37	**16.97**	810.2	113.38	**16.56**	1352.4	211.59	**31.29**
Master Node(GPU)	4062.66	1544.13	**1414.34**	79352.5	31080.16	**26594.01**	137807.85	65662.72	**50053.47**
Edge Node (CPU)	**41.27**	11.17	3.09	53.59	**7.26**	1.01	119.69	**22.59**	4.49
Edge Node (GPU)	**272.39**	67.23	5.05	9729.45	**2553.25**	260.98	22626.51	**7957.05**	701.75

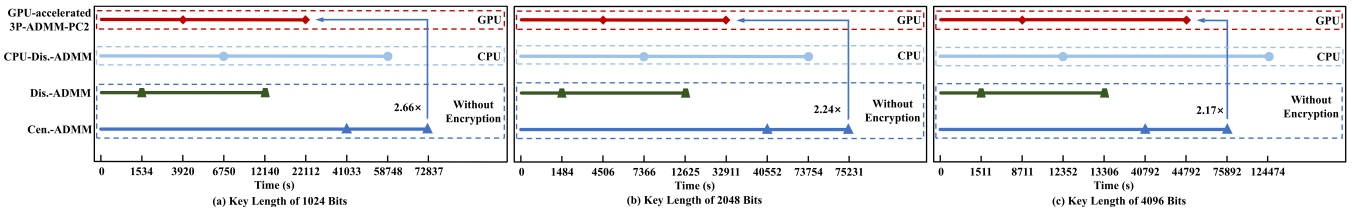


Fig. 8: Computational time of various schemes with different keys.

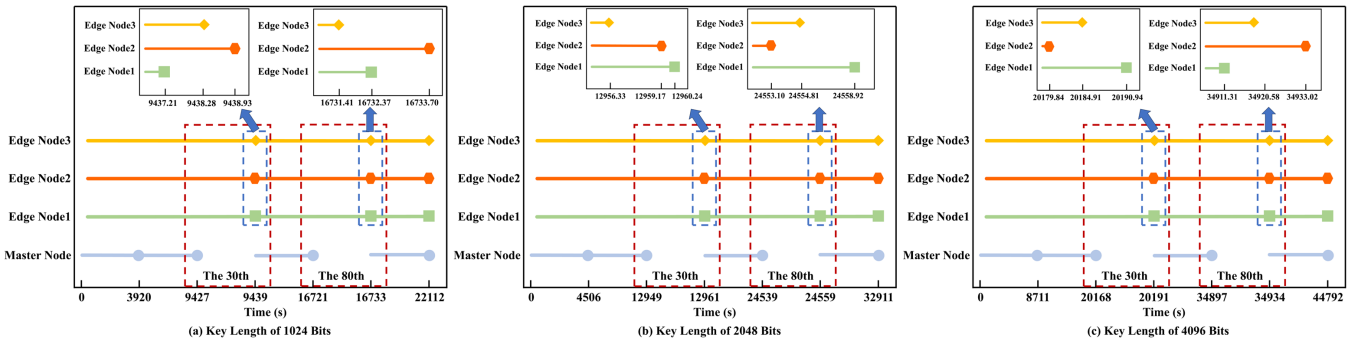


Fig. 9: The computational latency among nodes of proposed GPU-Accelerated 3P-ADMM-PC2.

TABLE III: Latency comparison of encryption and decryption using GPU vs. CPU with 10 edge nodes and 1024-bit key length.

Hardware Environment	**GPU**				**CPU**			
Computation Phase	**Initialization**	**Iterative Computation**			**Initialization**	**Iterative Computation**		
		30th	80th	100th		30th	80th	100th
Computation Completion Time (s)	1735	4679	9596	**11697**	4112	11818	24693	**29858**
Computational Latency (s)	1411	93	98	94	3771	253	266	260
Communication Latency (s)	294	8	7	-	289	8	8	-
Waiting Latency (s)	24	6	6	-	44	8	9	-

TABLE IV: Latency comparison of encryption and decryption using GPU vs. CPU with 10 edge nodes and 2048-bit key length.

Hardware Environment	**GPU**				**CPU**			
Computation Phase	**Initialization**	**Iterative Computation**			**Initialization**	**Iterative Computation**		
		30th	80th	100th		30th	80th	100th
Computation Completion Time (s)	4007	9107	17811	**20537**	6828	17141	33424	**41016**
Computational Latency (s)	3669	159	164	161	6462	351	364	353
Communication Latency (s)	292	11	10	-	297	10	12	-
Waiting Latency (s)	41	3	5	-	62	13	16	-

using GPU encryption and decryption, the edge nodes waits approximately 4 seconds for the master node, while the edge nodes wait about 2 seconds for each other. In contrast, with CPU encryption and decryption, the edge nodes experiences a waiting latency of around 9 seconds, and the edge nodes wait approximately 4 seconds for each other.

As shown in Table V, when the key length is 4096 bits, the computational overhead for encryption and decryption increases significantly. However, the computational cost per iteration remains lower with GPU-based encryption and decryption than with CPU-based encryption and decryption. It is noteworthy that the waiting latency among nodes increases with the key length. This is because the fluctuation in computational effort resulting from the non-uniform distribution of plaintext becomes more pronounced as the key length grows.

When the number of edge nodes is 3 and the dimension of $\mathbf{A}$ is $\mathbf{A} \in \mathbb{R}^{10000\times 65536}$, no scenario occurs where edge nodes wait for the master node, regardless of whether the key length is 1024, 2048, or 4096 bits. Therefore, in practical computational tasks, the balance among task scale, number of edge nodes, and disparities in hardware computational power directly affects both computational and waiting latency in each specific operation. When the number of edge nodes is large, the parallel computation across multiple edge nodes can even surpass the master node in speed. In future work, we will continue to investigate the constraints among task scale, number of edge nodes, and hardware capability gaps, and further optimize task allocation strategies to reduce overhead.

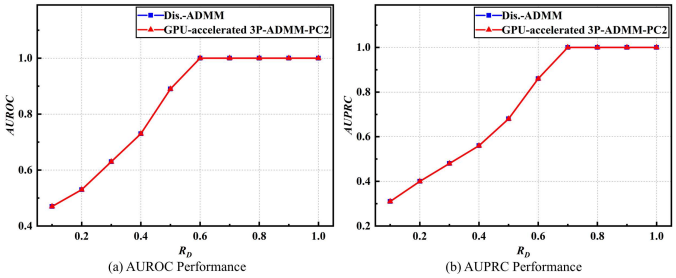
C. Application Example: Power Network Reconstruction

In the environment depicted in Fig. 4, consider a distributed edge privacy-computing network comprising one master node and three edge nodes. The goal is to recover the topology of a power network with N buses from M sets of observed data. For the adjacency matrix $\mathbf{a}_i$ of bus i, have the following optimization problem:

$$\arg\min_{\mathbf{a}_i\in\mathbb{R}^N} \frac{1}{2}\|\mathbf{I}_i - \mathbf{\Phi}_i\mathbf{a}_i\|_2^2 + \lambda\|\mathbf{a}_i\|_1, \tag{50}$$

where $\mathbf{I}_i \in \mathbb{R}^M$ represents the combination of current values at different time points for bus i; $\mathbf{\Phi}_i \in \mathbb{R}^{M\times N}(M \ll N)$ denotes the observation matrix of voltage differences and resistance values at different time points for bus i; and $\mathbf{a}_i \in \mathbb{R}^N$ is the adjacency vector for bus i. In a LAN, using the MATPOWER toolbox [1], a large scale real power network with 13569 buses is utilized, employing the local reconstruction method mentioned in [17]. For the high dimensionality of network, the centralized methods for network reconstruction is no longer feasible. The accuracy of the reconstruction is evaluated using the binary classification metrics area under the receiver operating characteristic (AUROC) and area under the precision-recall curve (AUPRC). In binary classification metrics, False Positive Rate and Recall are used as the x-axis, while True Positive Rate and Precision are used as the y-axis. The area under the curve represents the values of AUROC and AUPRC, respectively.

[1]https://matpower.org/



(a) AUROC Performance (b) AUPRC Performance

Fig. 10: The power grid reconstruction results of 13659 Buses, where $\Delta = 10^{15}$ and the key length is 2048 bits of GPU-accelerated 3P-ADMM-PC2.

Fig. 10 displays the accuracy of network reconstruction for the Dis.-ADMM scheme and the proposed GPU-accelerated 3P-ADMM-PC2 scheme under different data ratios (R_D). The proposed GPU-accelerated 3P-ADMM-PC2 scheme demonstrates high reconstruction accuracy. The AUROC and AUPRC of the Dis.-ADMM scheme coincide with those of the proposed GPU-accelerated 3P-ADMM-PC2 scheme, indicating once again that the accuracy loss of the proposed quantization method is negligible.

VI. Conclusion

This paper first proposed a parallel collaborative ADMM privacy computing algorithm, named 3P-ADMM-PC2, for distributed edge networks, where the edge nodes and master node complete the ADMM task calculation in three phases (3P), obtaining parallel collaborative privacy computing (PC2). Specially, the quantization method of proposed 3P-ADMM-PC2 addresses the limitations on plaintext for homomorphic encryption without affecting its homomorphic operations. Additionally, the ModExp of large integers has been successfully deployed on GPU across different platforms, enabling parallel encryption, decryption, and homomorphic computations, and facilitating multi-platform collaborative computing. To reduce the computational burden on the master node, a collaborative encryption and decryption algorithm between the master node and edge nodes, named GPU-accelerated 3P-ADMM-PC2, is finally proposed.

References

[1] C. Tan, D. Cai *et al.*, "Federated unfolding learning for csi feedback in distributed edge networks," *IEEE Trans. Commun.*, vol. 73, no. 1, pp. 410–424, 2025.
[2] B. Wu, F. Fang *et al.*, "Client selection and cost-efficient joint optimization for noma-enabled hierarchical federated learning," *IEEE Trans. Wireless Commun.*, 2024.
[3] X. Li *et al.*, "Publicly verifiable secure multi-party computation framework based on bulletin board," *IEEE Trans. Services Comput.*, 2024.
[4] H. Hua *et al.*, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1–35, 2023.
[5] X. Wang *et al.*, "Wireless powered mobile edge computing networks: A survey," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–37, 2023.
[6] L. Liu, J. Feng, X. Mu, Q. Pei, D. Lan, and M. Xiao, "Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15 513–15 526, 2023.
[7] C. Wang and L. Song, "An image encryption scheme based on chaotic system and compressed sensing for multiple application scenarios," *Inf. Sci.*, vol. 642, p. 119166, 2023.

TABLE V: Latency comparison of encryption and decryption using GPU vs. CPU with 10 edge nodes and 4096-bit key length.

Hardware Environment	**GPU**				**CPU**			
Computation Phase	**Initialization**	**Iterative Computation**			**Initialization**	**Iterative Computation**		
		30th	80th	100th		30th	80th	100th
Computation Completion Time (s)	6476	15021	29203	**34967**	13719	32904	66595	**79338**
Computational Latency (s)	6114	278	288	281	13314	644	681	659
Communication Latency (s)	295	13	11	-	297	12	11	-
Waiting Latency (s)	56	12	10	-	95	20	17	-

[8] C. Tan, D. Cai *et al.*, "Threshold-enhanced hierarchical spatial non-stationary channel estimation for uplink massive mimo systems," *IEEE Trans. wireless commun.*, vol. 23, no. 5, pp. 4830–4844, 2023.

[9] T. Wimalajeewa and P. K. Varshney, "Omp based joint sparsity pattern recovery under communication constraints," *IEEE Trans. Signal Process.*, vol. 62, no. 19, pp. 5059–5072, 2014.

[10] Q. Huang *et al.*, "Sparse learning model with embedded rip conditions for turbulence super-resolution reconstruction," *Comput. Methods Appl. Mech. Eng.*, vol. 425, p. 116965, 2024.

[11] T. Halsted, O. Shorinwa, J. Yu, and M. Schwager, "A survey of distributed optimization methods for multi-robot systems," *arXiv preprint arXiv:2103.12840*, 2021.

[12] C. Shen *et al.*, "Distributed robust multicell coordinated beamforming with imperfect csi: An admm approach," *IEEE Trans. signal process.*, vol. 60, no. 6, pp. 2988–3003, 2012.

[13] H. Liu, J. Zhang, Q. Wu, H. Xiao, and B. Ai, "Admm based channel estimation for riss aided millimeter wave communications," *IEEE Commun. Lett.*, vol. 25, no. 9, pp. 2894–2898, 2021.

[14] X. Zhang *et al.*, "Semi-definite relaxation-based admm for cooperative planning and control of connected autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 9240–9251, 2021.

[15] M. V. Afonso, J. M. Bioucas-Dias, and M. A. Figueiredo, "An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems," *IEEE Trans. Image Process.*, vol. 20, no. 3, pp. 681–695, 2010.

[16] M. S. Almeida and M. Figueiredo, "Deconvolving images with unknown boundaries using the alternating direction method of multipliers," *IEEE Trans. Image process.*, vol. 22, no. 8, pp. 3074–3086, 2013.

[17] Y. Liu, K. Huang, C. Yang, and Z. Wang, "Distributed network reconstruction based on binary compressed sensing via admm," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 4, pp. 2141–2153, 2023.

[18] Y. Yang, Y. Chen, G. Hu, and C. J. Spanos, "Optimal network charge for peer-to-peer energy trading: A grid perspective," *IEEE Trans. Power Syst.*, vol. 38, no. 3, pp. 2398–2410, 2022.

[19] T. Long, Q.-S. Jia, G. Wang, and Y. Yang, "Efficient real-time ev charging scheduling via ordinal optimization," *IEEE Trans. Smart Grid*, vol. 12, no. 5, pp. 4029–4038, 2021.

[20] Y.-C. Lin, G. Deng, and V. Prasanna, "A unified cpu-gpu protocol for gnn training," *Proc. ACM Int. Conf. Comput. Front.*, pp. 155–163, 2024.

[21] Y. Wang, B. Feng, and Y. Ding, "Qgtc: accelerating quantized graph neural networks via gpu tensor core," *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, pp. 107–119, 2022.

[22] Z. Huang, R. Hu, Y. Guo, E. Chan-Tin, and Y. Gong, "Dp-admm: Admm-based distributed learning with differential privacy," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1002–1012, 2019.

[23] C. Zhang, M. Ahmad, and Y. Wang, "Admm based privacy-preserving decentralized optimization," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 3, pp. 565–580, 2018.

[24] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3454–3469, 2020.

[25] T. Yang, X. Feng, S. Cai, Y. Niu, and H. Pen, "A privacy-preserving federated reinforcement learning method for multiple virtual power plants scheduling," *IEEE Trans. Circuits Syst. I Reg. Papers*, 2024.

[26] W. Ruan *et al.*, "Private, efficient, and accurate: Protecting models trained by multi-party learning with differential privacy," *IEEE Symp. Secur. Privacy (SP)*, pp. 1926–1943, 2023.

[27] M. Gong, J. Feng, and Y. Xie, "Privacy-enhanced multi-party deep learning," *Neural Netw.*, vol. 121, pp. 484–496, 2020.

[28] C. Piao, Y. Shi, J. Yan, C. Zhang, and L. Liu, "Privacy-preserving governmental data publishing: A fog-computing-based differential privacy approach," *Future Gener. Comput. Syst.*, vol. 90, pp. 158–174, 2019.

[29] S. Gupta *et al.*, "Energy-efficient dynamic homomorphic security scheme for fog computing in iot networks," *J. Inf. Security Appl.*, vol. 58, p. 102768, 2021.

[30] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, pp. 409–437, 2017.

[31] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[32] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," *CRYPTO*, pp. 868–886, 2012.

[33] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," *Proc. Theory Cryptogr. Conf.*, pp. 325–341, 2005.

[34] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*, pp. 223–238, 1999.

[35] C. Yu, T. Wang, Z. Shao, L. Zhu, X. Zhou, and S. Jiang, "Twinpilots: A new computing paradigm for gpu-cpu parallel llm inference," in *Proceedings of the 17th ACM International Systems and Storage Conference*, 2024, pp. 91–103.

[36] K. Shivdikar, Y. Bao *et al.*, "Gme: Gpu-based microarchitectural extensions to accelerate homomorphic encryption," in *Proc. 56th IEEE ACM Int. Symp. Microarchit.*, 2023, pp. 670–684.

[37] M. S. Riazi *et al.*, "Heax: An architecture for computing on encrypted data," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 1295–1309.

[38] B. Che *et al.*, "Unifl: Accelerating federated learning using heterogeneous hardware under a unified framework," *IEEE Access*, 2023.

[39] J. Dong, G. Fan, F. Zheng, T. Mao, F. Xiao, and J. Lin, "Tegras: An efficient tegra embedded gpu-based rsa acceleration server," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 16 850–16 861, 2022.

[40] IEEE, "Ieee standard for floating-point arithmetic, ieee std 754-2019 (revision of ieee 754-2008)," *Institute of Electrical and Electronics Engineers New York*, 2019.

[41] D. Pei, A. Salomaa, and C. Ding, *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.

[42] E. Bézout, *Théorie générale des équations algébriques*. Ph.-D. Pierres, 1779.

[43] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[44] J. Eckstein and D. P. Bertsekas, "On the douglas rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical programming*, vol. 55, no. 1, pp. 293–318, 1992.