Task Offloading and Resource Allocation in CPU-GPU Heterogeneous Networks

Chenyu Gong[1,2,3], Mulei Ma[1,2,3], Liantao Wu[1], Wenxiang Liu[4], Yong Zhou[1], and Yang Yang[1]

[1]School of Information Science and Technology, ShanghaiTech University
[2]Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences
[3]University of Chinese Academy of Sciences
[4]College of Electronic Science and Technology, National University of Defense Technology
[1]{gongchy, wult, maml, zhouyong, yangyang}@shanghaitech.edu.cn
[4]liuwenxiang8888@163.com

GLOBECOM 2022 - 2022 IEEE Global Communications Conference | 978-1-6654-3540-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/GLOBECOM48099.2022.10000976

***Abstract*—With the massive use of GPU, task scheduling under CPU-GPU clusters has become an indispensable research topic. Unlike existing models, we propose an innovative framework that users offload their tasks in CPU-GPU heterogeneous Edge Clusters (ECs) instead of general-purpose CPU clusters. The framework takes full advantage of the GPU's powerful parallel computing capabilities. Specifically, we decompose each user task into sequential segments and parallel segments, which can be offloaded to CPUs and GPUs of the ECs, respectively. By discretizing the GPU's computing capability, we formulate a Mixed Integer Nonlinear Programming (MINLP), which involves jointly optimizing the task offloading decision, the uplink transmission power of users, and computing resource allocation. To tackle this challenging problem, we propose a Joint Simulated Annealing and Convex Optimization (JSAC) based algorithm to minimize the total overhead consisting of delay and energy consumption. Our experimental simulation results demonstrate that the JSAC algorithm can make full use of GPU's powerful parallel computing capability via allocating GPU resources effectively. In particular, the JSAC algorithm achieves optimal performance in terms of system overhead, number of beneficial UEs, and speedup.**

***Index Terms*—Task scheduling, Heterogeneous Networks, CPU-GPU, Simulated Annealing**

I. Introduction

As 5G technology advances and artificial intelligence (AI) becomes commercially available on a large scale, user demand for computing continues to grow. Emerging computing-intensive and delay-sensitive applications (such as virtual reality, augmented reality, autonomous driving, and intelligent interaction) cannot deliver a high Quality of Experience (QoE) [1] due to the distance of cloud computing centers and limited local computing resources. In light of this, Multi-Access Edge Computing (MEC) [2] concept emerged. Unlike cloud computing [3] and local computing, MEC provides better services by offloading user tasks to the edge of the network. Edge nodes are closer to users than cloud computing centers with more powerful computing capabilities than local devices [4]. Specifically, they allow for faster completion of tasks through a trade-off between communication and computing, i.e. making up for transmission delays with savings in computing time.

The GPU's powerful floating-point computing capabilities significantly reduce the training time of deep neural networks (DNNs) compared to CPUs [5]. Meanwhile, high-performance GPUs produced by NVIDIA and AMD have met the computing needs of many applications, such as Computer Vision (CV) and Natural Language Processing (NLP). As a result, an increasing number of CPU-GPU clusters are being used for DNN training and inference [6]. At the same time, Kubernetes [7], an open-source container orchestration system, was introduced to enable orchestration on heterogeneous devices. The above points suggest that task scheduling in CPU-GPU heterogeneous networks will become a natural trend.

Since the concept of MEC was proposed, it has attracted a great deal of attention. A series of approaches and algorithms have been developed to address the challenges of task scheduling and enhance different global performance metrics in MEC scenarios. On the one hand, users offload tasks in homogeneous networks consisting of homogeneous devices with the same computing capability. Reference [8] defined a problem of offloading dependent tasks to homogeneous edge nodes with service caching and designed an efficient convex programming based algorithm to solve it. Xiao et al. [9] proposed a new heuristic algorithm for periodic task scheduling in the environment of homogeneous multi-core processors. On the other hand, users offload tasks in heterogeneous networks consisting of devices with different computing capabilities. Works [10], [11] adopted distributed game theoretic approach for achieving efficient computing offloading in the heterogeneous environment. Tran et al. [12] formulated a Mixed Integer Nonlinear Programming (MINLP) and proposed to decompose it into several sub-problems. Deep reinforcement learning was adopted in [13]–[16] for determining the offloading strategies. However, all the works above only used general-purpose computing devices, i.e., CPUs, without considering the case of joint CPU-GPU usage.

*The corresponding author is Liantao Wu.

This work was supported in part by the National Key Research and Development Program of China under grant 2019YFB1803304

The area of CPU-GPU cooperation has been studied. Reference [17] adopted a multiplexing approach that improves the utilization of CPU and GPU during the computing of machine learning tasks. Reference [18] investigated the computing offloading process in CPU-GPU heterogeneous processors and improved the efficiency of the whole cluster, but it ignores the resource allocation process. In our work, we consider the joint computing offloading and resource allocation in CPU-GPU heterogeneous networks to achieve lower system overhead and higher GPU utilization. Specifically, **the main contributions** of this paper are summarized as follows.

- Considering the user task at the code level, we decompose each task into sequential and parallel segments, which can be offloaded to CPUs and GPUs, respectively. Based on the resource sharing technology of the GPU, we discretize the GPU computing capability so that computing resource allocation is formulated as integer programming.
- In CPU-GPU heterogeneous networks, we model task scheduling as a MINLP problem to minimize the total overhead consisting of delay and energy consumption. The MINLP is decomposed so that computing offloading and resource allocation can be optimized alternately, which leads to the Joint Simulated Annealing and Convex Optimization (JSAC) based algorithm. It is a simulated annealing algorithm that requires the participation of the Bisection method and CVX.
- We carry out numerical simulations to evaluate the performance of the proposed solution, which is shown to be optimal in terms of system overhead, number of beneficial UEs, and speedup compared with traditional approaches.

II. System Model and Problem Formulation

A. Overview

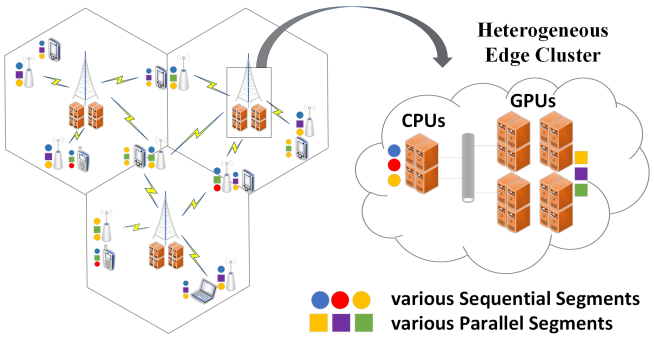

Fig. 1. CPU-GPU Heterogeneous Network

In this section, we consider a multi-user, multi-cluster scenario as illustrated in Fig. 1. At the center of the cell, each base station (BS) is equipped with an edge cluster (EC) composed of CPUs and GPUs to provide services to resource-limited mobile users. Each user's task can be executed locally or offloaded to an EC nearby via the wireless channel. Similar to many other works, such as [11], the transmission of final results can be ignored since the size of results is negligible compared to the task size, as well as the downlink bandwidth is usually larger than the uplink bandwidth. This section starts with the task model, followed by the EC model. The computing and communication model will be described next. The final section will present the problem formulation.

B. Task model

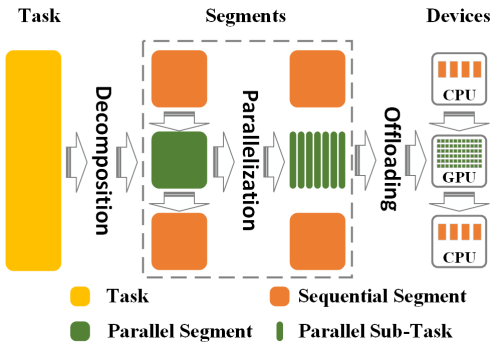

Fig. 2. Task model

DNN training and inference require quite considerable computing resources. GPU can meet the requirement due to its powerful parallel computing capability. In light of this, we adopt a task model consisting of both CPUs and GPUs instead of only using general-purpose CPUs. Reference [19] proposed a task model based on the dependencies between sub-tasks. Similarly, as illustrated in Fig. 2, a user task can be decomposed into three segments, which are divided into two types, i.e., two sequential segments and one parallel segment [18]. The task starts with a sequential segment executed on the CPU, which is a data pre-processing phase. Then we use a GPU to execute a parallel segment that can be parallelized into several sub-tasks. Tasks being offloaded to the GPU decreases the computing delay due to its powerful computing capability. The whole process ends with the result aggregation phase with the second sequential segment running on the CPU.

For expression and analysis, we assume that there are N users, denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$. Each user has a task A_n, i.e., $A_n = (z_n; f_{n,1}; f_{n,2}; f_{n,3}; b_n), n \in \mathcal{N}$, where $z_n \in \mathbb{N}^+$ denotes the size (bit) of the task , $f_{n,1}, f_{n,3} \in \mathbb{R}$ denotes floating point of operations (FLOPs) required for the sequential segment at the beginning and the end, $f_{n,2} \in \mathbb{R}$ denotes FLOPs required for the parallel segment, and $b_n \in \mathbb{Z}$ denotes the number of parallel sub-tasks. The task can be executed locally or offloaded to the EC as a whole.

C. Edge cluster model

We denote M ECs by $\mathcal{M} = \{1, 2, \cdots, M\}$. Each EC is equipped with multiple CPUs and GPUs. About GPU, resource sharing has already been a topic to be studied. On the one hand, resource sharing requires resource isolation. NVIDIA has made a scheme [20] to divide computing resources equally and achieve resource isolation completely. On the other hand, the parallel mode of the GPU is also an issue to be considered.

Parallel mode is the way that multiple tasks run on the same GPU at the same time. There are two types currently: (i) Time-slice Multiplexing. It refers to dividing time slices so that different tasks occupy a separate time slice. In this mode, tasks are actually concurrent rather than parallel, as only one task is running at the same time. Gaia [21], KubeShare [22], etc. adopt this mode. (ii) Merge Sharing. It refers to merging multiple tasks into a single context so that multiple tasks can be run simultaneously. This mode is in the true sense of parallelism. The most representative one is MPS [23] of NVIDIA. Here we adopt the second parallel mode, Merge Sharing. Based on the two aspects above, we divide a full GPU into several parts. Each part is regarded as a non-parallelizable GPU that provides service to a parallel sub-task. Then we make the following assumptions.

- The computing resources of each GPU in the same EC are the same.
- Multiple GPUs in the same EC can run simultaneously. Each GPU can be assigned to only one parallel sub-task, and the utilization can reach 100%.

Based on the above assumptions, we denote EC m as $C_m = (b_m^{cpu}; f_m^{cpu}; p_m^{c-wk}; p_m^{c-stb}; b_m^{gpu}; f_m^{gpu}; p_m^{g-wk}; p_m^{g-stb}), \forall m \in \mathcal{M}$, where $b_m^{cpu} \in \mathbb{Z}^+$ denotes the number of CPU in EC m, $f_m^{cpu} \in \mathbb{R}^+$ denotes CPU computing capability, i.e., floating point of per second (FLOPS), $p_m^{c-wk} \in \mathbb{R}^+$ denotes CPU work power (W), and $p_m^{c-stb} \in \mathbb{R}^+$ denotes CPU standby power (W). The parameters of the GPU are the same.

D. Computing model

Users can execute their tasks locally or offload them to ECs. We denote offloading decision by $x_{n,m} \in \{0,1\}, n \in \mathcal{N}, m \in \mathcal{M} \cup \{0\}$, where $x_{n,m} = 1$ means user n offload task to EC m or executes its task locally if $m = 0$. $\mathcal{X} \subseteq \mathbb{N}^{N,M+1}$ denotes the offloading variables of all users. We assume that user n has only one CPU locally whose computing capability is f_n^{cpu} and computing power is p_n^{cpu}.If user n decides not to offload, both the sequential and parallel segments will be executed by the local CPU. The computing delay and energy consumption of local computing are expressed as:

$$T_{n,0}^{comp} = \frac{f_{n,1} + f_{n,2} + f_{n,3}}{f_n^{cpu}} \tag{1}$$

$$E_{n,0}^{comp} = T_{n,0}^{comp} p_n^{cpu} \tag{2}$$

When decides to offload the task to EC m, the user n will be allocated a CPU and $b_{n,m}^{gpu}$ GPUs. $b_{n,m}^{gpu} \in \mathcal{B}, n \in \mathcal{N}, m \in \mathcal{M}$,and $\mathcal{B}$ is computing resource allocation variables of all users. We denote the set of users offloaded to EC m by $\mathcal{N}_m$, and the number of the set is expressed as $k_m = |\mathcal{N}_m|$. In addition, there are several restrictions: (i)$0 \leq \sum_{n \in \mathcal{N}_m} b_{n,m}^{gpu} \leq b_m^{gpu}, \forall m \in \mathcal{M}$, which means that the number of GPUs allocated by EC m is no more than the total GPUs. (ii) $0 \leq k_m \leq b_m^{cpu}, \forall m \in \mathcal{M}$, which indicates that the total number of tasks offloaded to EC m is no more than the total number of CPUs because one task must occupy one CPU. We assume that the task occupies the device entirely during computing. The CPU and GPU computing delay of user n associated with EC m can be expressed as:

$$T_{n,m}^{comp,cpu} = \frac{f_{n,1} + f_{n,3}}{f_m^{cpu}} \tag{3}$$

$$T_{n,m}^{comp,gpu} = \left\lceil \frac{b_n}{b_{n,m}^{gpu}} \right\rceil \frac{f_{n,2}}{b_n f_m^{gpu}} \tag{4}$$

where the latter item represents the computing delay consumed by the parallel sub-task on one GPU, and the former item represents the rounds of the whole parallel segment that needs to be executed on $b_{n,m}^{gpu}$ GPUs. To sum up, the computing delay of user n is :

$$T_n^{comp} = \sum_{m \in \mathcal{M} \cup \{0\}} x_{n,m} (T_{n,m}^{comp,cpu} + T_{n,m}^{comp,gpu}) \tag{5}$$

Computing devices consume energy not only during working but also during standby. So the CPU and GPU computing energy consumption of user n associated with EC m can be expressed as follows.

$$E_{n,m}^{comp,cpu} = T_{n,m}^{comp,cpu} p_m^{c-wk} + T_{n,m}^{comp,gpu} p_m^{c-stb} \tag{6}$$

$$E_{n,m}^{comp,gpu} = T_{n,m}^{comp,gpu} p_m^{g-wk} + T_{n,m}^{comp,cpu} p_m^{g-stb} \tag{7}$$

To sum up, user n's computing energy consumption is :

$$E_n^{comp} = \sum_{m \in \mathcal{M} \cup \{0\}} x_{n,m} (E_{n,m}^{comp,cpu} + E_{n,m}^{comp,gpu}) \tag{8}$$

E. Communication model

The communication model is divided into two parts, the uplink process and the transmission of intermediate results within the EC. Especially, the latter can be ignored because the bandwidth inside the EC is very large compared with the uplink. Therefore, we only focus on the uplink process.

We consider the system with OFDMA as the multiple access scheme in the uplink, in which the total bandwidth W_0 is divided into J equal sub-bands of size $W = W_0/J$. $\mathcal{N}^j$ is the set of users who occupy sub-band j. We denote the maximum transmission power of user n as P_n. Then, the transmission power allocation variables can be expressed as $\mathcal{P} = \{p_n | 0 \leq p_n \leq P_n, n \in \mathcal{N}\}$. Meanwhile, the channel gain matrix can be represented by $\mathcal{H} = \{h_{n,m} = d_{n,m}^{-\gamma} | n \in \mathcal{N}, m \in \mathcal{M}\}$, where $d_{n,m}$ denotes the distance from user n to EC m and γ denotes the path loss factor. From Shannon's formula, we can get the transmission rate from user n to EC m [11]:

$$R_{n,m}(\mathcal{X}, \mathcal{P}) = W log_2(1 + \eta_{n,m}) \tag{9}$$

where $\eta_{n,m}$ represents the Signal-to-Interference-plus-Noise Ratio (SINR) from n to m, which can be expressed by the following:

$$\eta_{n,m} = \frac{p_n h_{n,m}}{\sigma^2 + \sum_{r \in \mathcal{N}^j \setminus \{n\}} p_r h_{r,m}} \tag{10}$$

where σ^2 is the noise power and j is the sub-band that user n occupies, i.e., $n \in \mathcal{N}^j$. The second term in the denominator represents the inter-cell interference generated between users under OFDMA. If user n decides to execute the task locally,

there will be no transmission delay and energy consumption, i.e., $T_{n,0}^{trans} = 0, E_{n,0}^{trans} = 0$. We express the transmission delay and energy consumption of user n as follows:

$$T_n^{trans} = \sum_{m \in \mathcal{M}} x_{n,m} \frac{z_n}{R_{n,m}(\mathcal{X}, \mathcal{P})} \tag{11}$$

$$E_n^{trans} = p_n T_n^{trans} \tag{12}$$

F. Problem formulation

According to (5)(11) and (8)(12), total delay and energy consumption of user n is :

$$T_n = T_n^{comp} + T_n^{trans} \tag{13}$$

$$E_n = E_n^{comp} + E_n^{trans} \tag{14}$$

So, the overhead of N users is :

$$Y(\mathcal{X}, \mathcal{B}, \mathcal{P}) = \sum_{n \in \mathcal{N}} Y_n = \sum_{n \in \mathcal{N}} (\lambda_t T_n + \lambda_e E_n) \tag{15}$$

where λ_t and λ_e specify user n's preference on delay and energy consumption, respectively. We formulate a system overhead minimization problem,i.e.,

$$\min_{\mathcal{X}, \mathcal{B}, \mathcal{P}} Y(\mathcal{X}, \mathcal{B}, \mathcal{P}) \tag{16a}$$

$$\text{s.t. } x_{n,m} \in \{0, 1\}, \forall n \in \mathcal{N}, m \in \mathcal{M} \tag{16b}$$

$$\sum_{m \in \mathcal{M} \cup \{0\}} x_{n,m} = 1, \forall n \in \mathcal{N} \tag{16c}$$

$$0 \leq p_n \leq P_n, \forall n \in \mathcal{N} \tag{16d}$$

$$0 \leq k_m \in \mathbb{Z} \leq b_m^{cpu}, \forall m \in \mathcal{M} \tag{16e}$$

$$0 \leq \sum_{n \in \mathcal{N}_m} b_{n,m}^{gpu} \in \mathbb{Z} \leq b_m^{gpu}, \forall m \in \mathcal{M} \tag{16f}$$

The constraints in the formulation above can be explained as follows: constraints (16b) and (16c) imply that each task can be either executed locally or offloaded to at most one EC; constraint (16d) specifies the transmission power budget of each user; constraint (16e) indicates that the total number of tasks offloaded to EC m is no more than the total number of CPUs; constraint (16f) means that the number of GPUs allocated by EC m is no more than the total number of GPUs.

III. Joint Simulated Annealing and Convex Optimization Based Algorithm

A. Problem Analysis

Equation (16) indicates that this is a MINLP. By exploiting the structure and the constraints of the problem, we transform the problem as:

$$Y(\mathcal{X}, \mathcal{B}, \mathcal{P}) = Y_1(\mathcal{X}, \mathcal{P}) + Y_2(\mathcal{X}, \mathcal{B}) + Y_3(\mathcal{X}) \tag{17a}$$

$$Y_1(\mathcal{X}, \mathcal{P}) = \sum_{n \in \mathcal{N}} \sum_{m \in \mathcal{M}} x_{n,m} \frac{(\lambda_t + \lambda_e p_n) z_n}{R_{n,m}(\mathcal{X}, \mathcal{P})} \tag{17b}$$

$$Y_2(\mathcal{X}, \mathcal{B}) = \sum_{n \in \mathcal{N}} \sum_{m \in \mathcal{M}} x_{n,m} \alpha_1 T_{n,m}^{comp,gpu} \tag{17c}$$

$$Y_3(\mathcal{X}) = \sum_{n \in \mathcal{N}} \sum_{m \in \mathcal{M}} x_{n,m} \alpha_2 T_{n,m}^{comp,cpu} \tag{17d}$$

Where $\alpha_1 = \lambda_t + \lambda_e (p_{n,m}^{c-wk} + p_{n,m}^{g-stb})$,and $\alpha_2 = \lambda_t + \lambda_e (p_{n,m}^{g-wk} + p_{n,m}^{c-stb})$. By observing (17b) and (17c), we can decompose the MINLP into several sub-problems in order to optimize $\mathcal{P}$ and $\mathcal{B}$ separately. According to (17a), the problem (16) is equivalent to:

$$\min_{\mathcal{X}} \left(\min_{\mathcal{B}, \mathcal{P}} Y(\mathcal{X}, \mathcal{B}, \mathcal{P}) \right) \tag{18a}$$

$$\text{s.t. (16b)} - \text{(16f)} \tag{18b}$$

Then, we can decompose it to three problems:

$$\mathcal{Q}_1 : \begin{cases} \min_{\mathcal{X}} Y^*(\mathcal{X}, \mathcal{B}, \mathcal{P}) \\ \text{s.t. (16b)} - \text{(16c)} \end{cases}$$

$$\mathcal{Q}_2 : \begin{cases} \min_{\mathcal{P}} Y_1(\mathcal{X}, \mathcal{P}) \\ \text{s.t. (16d)} \end{cases} \qquad \mathcal{Q}_3 : \begin{cases} \min_{\mathcal{B}} Y_2(\mathcal{X}, \mathcal{B}) \\ \text{s.t. (16e)} - \text{(16f)} \end{cases}$$

where $Y^*(\mathcal{X}, \mathcal{B}, \mathcal{P}) = Y_1^*(\mathcal{X}, \mathcal{P}) + Y_2^*(\mathcal{X}, \mathcal{B}) + Y_3(\mathcal{X}), \forall \mathcal{X}$.

B. Algorithms

As shown in Fig. 3, the blue part represents the decomposition of the problem, the green part represents the corresponding solution, and the gray part represents the parameters. The specific analysis is below.

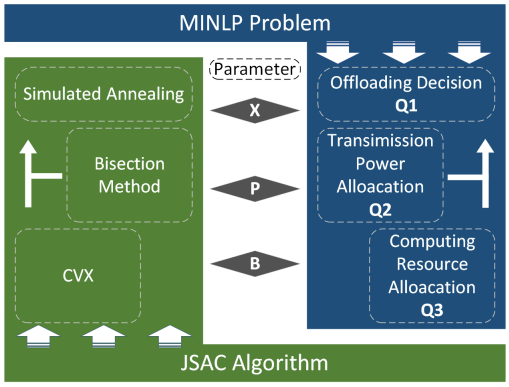

Fig. 3. Problem Decomposition

$\mathcal{Q}_1$ is a nonlinear integer programming problem that can be solved by some exact solutions, such as the branch-and-bound method, as well as some approximate solutions, such as heuristic algorithms. Because the branch-and-bound method can only solve integer linear programming problems, we choose a heuristic algorithm, simulated annealing. The algorithm starts from a higher initial temperature T, and decreases to the lowest temperature T_{min} with the parameter $\beta \in (0, 1)$. At each step, the algorithm selects a solution close to the current one by using the stochastic sampling method, measures its quality, and moves to it according to the temperature-dependent probabilities of selecting better or worse solutions.

Regarding $\mathcal{Q}_2$, according to the constraint of p_n, we approximate the transmission power allocation as a quasi-convex optimization problem which can be solved by the Bisection

Method [12]. About $\mathcal{Q}_3$, the problem is integer programming. We approximate it as nonlinear programming by relaxing integer variables associated with (4) as follows:

$$Y_2(\mathcal{X},\mathcal{B}) = \sum_{n\in\mathcal{N}}\sum_{m\in\mathcal{M}} x_{n,m}\frac{\alpha_1}{b^{gpu}_{n,m}}\frac{f_{n,2}}{f^{gpu}_m} \tag{19}$$

The problem (19) is transformed into a convex optimization problem about $b^{gpu}_{n,m}$. After finding the optimal solution $(b^{gpu}_{n,m})^*$ by a CVX solver, round it up. At each step of the $\mathcal{Q}_1$ update, the optimal solutions of $\mathcal{Q}_2$ and $\mathcal{Q}_3$ associated with $\mathcal{Q}_1$'s offloading decision is required. Our JSAC Algorithm is shown in Algorithm 1.

Algorithm 1: JSAC Algorithm

Input: $A_n, f_n, P_n, C_m; \mathcal{H}, W, \lambda_t, \lambda_e; T, T_{min}, \beta, L$
Output: $\mathcal{X}_{opt}, \mathcal{B}_{opt}, \mathcal{P}_{opt}$
Initialize: offloading decesion: $\mathcal{X}_{opt}$;
Get $\mathcal{B}_{opt}$ by CVX and $\mathcal{P}_{opt}$ by Bisection Method associated with $\mathcal{X}_{opt}$;
Get overhead Y_{opt} associated with $\mathcal{X}_{opt}$,$\mathcal{B}_{opt}$,$\mathcal{P}_{opt}$;
$Y, \mathcal{X}, \mathcal{B}, \mathcal{P} = Y_{opt}, \mathcal{X}_{opt}, \mathcal{B}_{opt}, \mathcal{P}_{opt}$;
while $T \geq T_{min}$ **do**
  while $L \geq 0$ **do**
    Get random neighborhood of $\mathcal{X}_{opt}$: $\mathcal{X}_{temp}$;
    Get $\mathcal{B}_{temp}, \mathcal{P}_{temp}, Y_{temp}$ associated with $\mathcal{X}_{opt}$;
    $\delta = Y_{temp} - Y$;
    if $\delta \leq 0$ **then**
      $Y, \mathcal{X}, \mathcal{B}, \mathcal{P} = Y_{temp}, \mathcal{X}_{temp}, \mathcal{B}_{temp}, \mathcal{P}_{temp}$;
      if $Y_{temp} \leq Y_{opt}$ **then**
        $Y_{opt}, \mathcal{X}_{opt}, \mathcal{B}_{opt}, \mathcal{P}_{opt} = Y_{temp}, \mathcal{X}_{temp}, \mathcal{B}_{temp}, \mathcal{P}_{temp}$;
      end
    else
      if $\exp(-\delta/T) \geq rand(0,1)$ **then**
        $Y, \mathcal{X}, \mathcal{B}, \mathcal{P} = Y_{temp}, \mathcal{X}_{temp}, \mathcal{B}_{temp}, \mathcal{P}_{temp}$;
      end
    end
    $L = L - 1$
  end
  $T = \beta T$
end

IV. Performance Evaluation

A. Simulation Setup

In this section, the results of the JSAC algorithm compared with other baseline algorithms are proposed. We consider a total of 10 cells, each with an EC, and users are randomly distributed in the cells. For computing devices, we assume standby power as one-quarter of computing power. Moreover, the computing power is proportional to the computing capability. The specific parameter settings are shown in Table I.

We compare the proposed JSAC with the following baselines:

TABLE I
Simulation Parameters

Parameter	Value
z_n (task size of user n)	[50,500] MB
$f_{n,1/2/3}$ (FLOPs requirements of user n)	[100,1000] TFLOPs
b_n (number of the parallel sub-task)	[2,8]
f^{cpu}_n (computing capability of user n)	[6,14] GFLOPS
p^{cpu}_n (CPU power of user n)	[30,70] W
b^{cpu}_m (number of EC m's CPU)	8
f^{cpu}_m (EC m's CPU computing capability)	[40,60] GFLOPS
p^{c-wk}_m (CPU work-power of EC m)	[30,70] W
b^{gpu}_m (number of EC m's GPU)	[4,8,16,32,64]
f^{gpu}_m (EC m's GPU computing capability)	[40,60] GFLOPS
p^{g-wk}_m (GPU work-power of EC m)	[80,120] w
P_n (max transmission power of user n)	5 W
W (the size of sub-band)	2.5 MHz
$d_{n,m}$ (the distance from user n to EC m)	[10,100] m
γ (the path loss factor)	4
σ^2 (noise power)	10^{-10} W

- Local Computing (Local): every user chooses to execute the task locally.
- Random Offloading (Random): every user executes its task locally or offloads its task to a random EC.
- Greedy Offloading (Greedy): every user offloads its task to the EC with the highest computing capability.
- CATS Offloading (CATS) [24]: a game theory based distributed task scheduling algorithm.

B. Result

Fig. 4 illustrates the system overhead with a different number of users and demonstrates the best performance of the JSAC algorithm. Meanwhile, the significant improvement of performance between various algorithms and local computing indicates the GPU's powerful parallel computing capability. The beneficial UEs are those who can reduce its overhead compared with local computing, via offloading tasks to ECs, i.e., $\{n \in \mathcal{N} | Y_n(x_{n,0}=0) > Y_n(x_{n,0}=1)\}$, which is shown in Fig. 5. The percentage of beneficial UEs is up to 100% when there are 20 and 30 users associated with the JSAC algorithm. In addition, The percentage of beneficial UEs is inversely proportional to the total number of users, for users are allocated fewer resources.

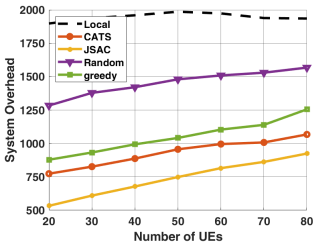

Fig. 4. System Overhead

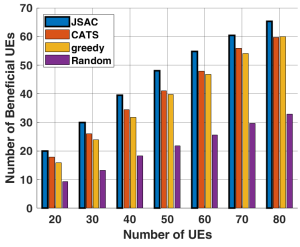

Fig. 5. Beneficial UEs

Fig. 6 shows the convergence rate of the JSAC algorithm for different numbers of users, which implies the algorithm's effectiveness. The speedup is a measure of

the multiple of multi-node parallel computing compared to one-node sequential computing, which is used to describe the effect of parallel computing, and it is defined by $T_n^{comp}(\text{local computing})/T_n^{comp}(\text{offloading})$. The overall Probability Mass Functions (PMF) and Cumulative Distribution Functions (CDF) of speedup are plotted as rectangular bars and dash curves, respectively, in Fig. 7. In the bar chart, the blue is shorter when the speedup is less than 4, while the blue is longer than the red in the case of speedup greater than 4. That indicates more users get a higher speedup under the JSAC algorithm. The blue curve is more concentrated to the right in the curves graph, which means users get a more significant average speedup by the JSAC algorithm. All of the above indicates that GPU can bring computing acceleration to the task, and the JSAC algorithm can make better use of the computing performance of GPU.

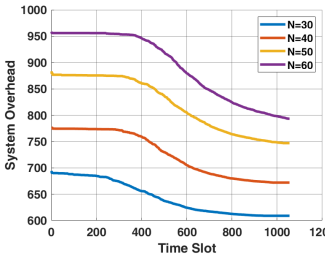

Fig. 6. Convergence Rate

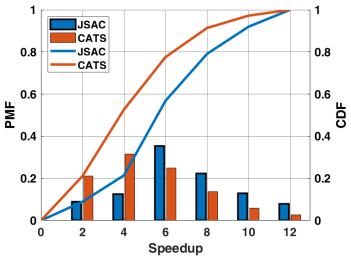

Fig. 7. Distribution of Speedup

V. Conclusions

In this paper, we propose a novel framework that users offload their tasks to CPU-GPU heterogeneous networks. Firstly, we decompose each task into sequential and parallel segments, then offload them to CPUs and GPUs, respectively, to exploit GPUs' powerful parallel computing capability. In addition, discretization of GPU computing capability leads to integer programming which is part of the MINLP. Specifically, we decompose the MINLP into three sub-problems nested with each other and solve them by the JSAC algorithm consisting of simulated annealing algorithm, bisection method, and CVX. Our experimental results indicate that the GPU can provide powerful parallel computing capability for user tasks and the JSAC algorithm can utilize GPUs to achieve optimal system performance effectively.

References

[1] Y. Yang, et al., "6G Network AI Architecture for Everyone-Centric Customized Services," *IEEE Network*, early access, doi: 10.1109/MNET.124.2200241, July 2022.

[2] L. Wu et al., "DOT: Decentralized Offloading of Tasks in OFDMA based Heterogeneous Computing Networks," *IEEE Internet of Things Journal*, early access, doi: 10.1109/JIOT.2022.3171555, 2022.

[3] M. Ke, Z. Gao, Y. Wu, X. Gao and K. -K. Wong, "Massive Access in Cell-Free Massive MIMO-Based Internet of Things: Cloud Computing and Edge Computing Paradigms," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 3, pp. 756-772, March 2021.

[4] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655-1674, Aug. 2019.

[5] N. -M. Ho and W. -F. Wong, "Tensorox: Accelerating GPU Applications via Neural Approximation on Unused Tensor Cores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 429-443, 1 Feb. 2022.

[6] A. Dhakal, S. G. Kulkarni and K. K. Ramakrishnan, "Machine Learning at the Edge: Efficient Utilization of Limited CPU/GPU Resources by Multiplexing," *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pp. 1-6, 2020.

[7] Kubernetes, [online] Available: https://kubernetes.io .

[8] G. Zhao, H. Xu, Y. Zhao, C. Qiao and L. Huang, "Offloading Tasks With Dependency and Service Caching in Mobile Edge Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777-2792, 1 Nov. 2021.

[9] S. Xiao, D. Li and S. Wang, "Periodic Task Scheduling Algorithm for Homogeneous Multi-core Parallel Processing System", *2019 IEEE International Conference on Unmanned Systems (ICUS)*, pp. 710-713 , 2019.

[10] X. Chen, L. Jiao, W. Li and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795-2808, October 2016.

[11] Y. Yang, Z. Liu, X. Yang, K. Wang, X. Hong and X. Ge, "POMT: Paired Offloading of Multiple Tasks in Heterogeneous Fog Networks," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8658-8669, Oct. 2019.

[12] T. X. Tran and D. Pompili, "Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856-868, Jan. 2019.

[13] Y. Han, S. Shen, X. Wang, S. Wang and V. C. M. Leung, "Tailored Learning-Based Scheduling for Kubernetes-Oriented Edge-Cloud System," *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1-10 , 2021.

[14] Y. Dai, K. Zhang, S. Maharjan and Y. Zhang, "Edge Intelligence for Energy-Efficient Computation Offloading and Resource Allocation in 5G Beyond," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12175-12186, Oct. 2020.

[15] S. Nath and J. Wu, "Dynamic Computation Offloading and Resource Allocation for Multi-user Mobile Edge Computing," *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1-6, 2020.

[16] S. Bi, L. Huang, H. Wang and Y. -J. A. Zhang, "Lyapunov-Guided Deep Reinforcement Learning for Stable Online Computation Offloading in Mobile-Edge Computing Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7519-7537, Nov. 2021.

[17] A. Dhakal, S. G. Kulkarni and K. K. Ramakrishnan, "Machine Learning at the Edge: Efficient Utilization of Limited CPU/GPU Resources by Multiplexing," *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pp. 1-6, 2020.

[18] N. Tsog, S. Mubeen, F. Bruhn, M. Behnam and M. Sjödin, "Offloading Accelerator-intensive Workloads in CPU-GPU Heterogeneous Processors," *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1-8, 2021.

[19] T. -W. Huang, D. -L. Lin, C. -X. Lin and Y. Lin, "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1303-1320, 1 June 2022.

[20] "NVIDIA Multi-Instance GPU User Guide", https://docs.nvidia.com/datacenter/tesla/mig-user-guide/#supported-profiles

[21] J. Gu, S. Song, Y. Li and H. Luo, "GaiaGPU: Sharing GPUs in Container Clouds," *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 469-476, 2018.

[22] Yeh, Tingan , H. Chen , and J. Chou . "KubeShare: A Framework to Manage GPUs as First-Class and Shared Resources in Container Cloud." *HPDC '20: The 29th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 173–184 , June 2020.

[23] "Multi-Process Service (nvidia.com)", https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[24] Z. Liu,K. Li,L. Wu,Z. Wang,Y.Yang. "CATS: Cost Aware Task Scheduling in Multi-Tier Computing Networks" , *Journal of Computer Research and Development*, vol. 57, no. 9, pp. 1810-1822, 2020.