

## RESEARCH ARTICLE

# Optimizing Task Orchestration for Distributed Real-Time Electromagnetic Transient Simulation

QI GUO<sup>1,2,3</sup>, (Senior Member, IEEE), HAIPING GUO<sup>1,2,3</sup>, YUANHONG LU<sup>1,2,3</sup>,  
TIANYU GUO<sup>1,2,3</sup>, (Member, IEEE), JIE ZHANG<sup>1,2,3</sup>, JINGYUE ZHANG<sup>1,2,3</sup>, HUI LUO<sup>1,2,3</sup>,  
LIBIN HUANG<sup>1,2,3</sup>, AND YANJUN ZHAO<sup>4</sup>

<sup>1</sup>State Key Laboratory of HVDC, Electric Power Research Institute, China Southern Power Grid, Guangzhou 510663, China

<sup>2</sup>Guangdong Provincial Key Laboratory of Intelligent Operation and Control for New Energy Power System, Guangzhou 510663, China

<sup>3</sup>CSG Key Laboratory for Power System Simulation, Electric Power Research Institute, China Southern Power Grid, Guangzhou 510663, China

<sup>4</sup>China Southern Power Grid Company Ltd., Guangzhou 510663, China

Corresponding author: Tianyu Guo (guoty@csg.cn)


This work was supported in part by the Science and Technology Project of China Southern Power Grid under Grant ZBKJXM20220068, and in part by the National Natural Science Foundation of China under Grant U23B6008.

**ABSTRACT** Transient simulation in power engineering is crucial as it models the dynamic behavior of power systems during sudden events like faults or short circuits. Electromagnetic transient simulations involve multiple coordinated tasks. Traditional simulations are centralized and struggle to meet scalability requirements. To achieve these goals, distributed electromagnetic transient simulation has emerged as a new trend. Nevertheless, the distributed electromagnetic transient simulation introduces network communication. Achieving real-time simulation across distributed nodes poses the challenge of minimizing communication costs. In this paper, our proposal focuses on optimizing the task orchestration to reduce communication costs. Specifically, in the electromagnetic transient simulation, these tasks has certain communication pattern where the communicated objects of each task are pre-defined. We represent the pattern as a graph, with tasks represented as nodes and communications as edges. Furthermore, we propose to use graph partition with the objective of minimal communication costs and fine tune the partitions with the resource requirements of each distributed node. The experimental results demonstrate that our proposal has strength in achieving high-performance electromagnetic transient simulation.

**INDEX TERMS** Electromagnetic transient simulation, distributed system, task orchestration.

## I. INTRODUCTION

Real-time simulation of electromagnetic transients offers a powerful tool for understanding and mitigating the challenges posed by the integration of new energy sources [1]. It allows for the dynamic assessment of power system behavior under varying conditions, enabling the development and validation of effective control strategies [2], [3]. In the face of dynamic changes brought about by renewable energy generation, energy storage, and other emerging technologies, real-time simulation provides insights into system stability, fault analysis, and protection coordination [4], [5]. The ability

The associate editor coordinating the review of this manuscript and approving it for publication was Ning Kang .

to capture transient responses in real-time is paramount for ensuring the reliability and resilience of power systems. Real-time simulations facilitate a proactive approach to addressing potential issues, allowing for the timely implementation of corrective measures. By providing a dynamic and accurate representation of power system behavior, real-time electromagnetic transient simulation contributes to the efficient integration and optimal utilization of new energy sources, fostering the ongoing evolution of power systems towards sustainability and adaptability [6], [7].

Within the domain of electromagnetic transient simulation, centralized methodologies have long been the predominant choice. Yet, the limitations of these centralized models, particularly in terms of scalability, are becoming increasingly

evident [8]. The complexity of modern power systems, coupled with the demand for large-scale simulations, is pushing the limits of traditional centralized approaches. This has led to a growing recognition in the industry of the need for a shift towards distributed transient simulation, which offers a more scalable and viable solution. This shift represents a significant evolution in simulation strategies, driven by the need to accommodate the complexities arising from integrating diverse energy sources and advanced technologies into contemporary power systems [9], [10]. Distributed transient simulation, characterized by its decentralized and parallel processing capabilities, addresses the scalability issues inherent in traditional methods. It caters to the expanding scope of transient simulations and aligns with the broader trend of exploiting distributed computing for improved computational efficiency [11], [12].

Communication is fundamental, serving as the critical element that facilitates seamless interaction and synchronization among distributed components [13], [14] in the realm of distributed real-time electromagnetic transient simulation. The importance of efficient and effective communication is particularly highlighted when striving to achieve accurate, real-time results in simulating electromagnetic transients. This complex interplay involving data exchange, coordination, and synchronization, managed through communication protocols, constitutes the core of a robust distributed electromagnetic transient simulation system [15], [16]. Therefore, a deep understanding and optimization of these communication mechanisms are essential to guarantee the success and reliability of real-time simulations [17] in this intricate and ever-evolving field.

In electromagnetic transient simulation, it is common practice to view each task as process in operating system and assign individual tasks to designated CPU cores. This approach helps to minimize the performance issues associated with process switching. However, with the increasing complexity of electromagnetic transient simulations, there arises a need to distribute these tasks across multiple computing nodes. This distribution strategy is crucial to maintain exclusive access of each task to a CPU core, thereby optimizing processing efficiency. One key aspect of this approach is the communication between tasks, which is vital for maintaining synchronization. Nonetheless, this inter-task communication also significantly impacts the overall efficiency of the simulation. In these simulations, each task adheres to a specific communication pattern. Given this scenario, optimizing the coordination of tasks across the distributed computing nodes becomes imperative [18], [19]. We propose a task orchestration algorithm aimed at addressing the aforementioned challenge. In essence, the algorithm partitions simulation tasks based on their individual communication patterns. Subsequently, each partition is allocated to a computing node based on its CPU requirements.

The primary contribution of this paper can be summarized as follows:

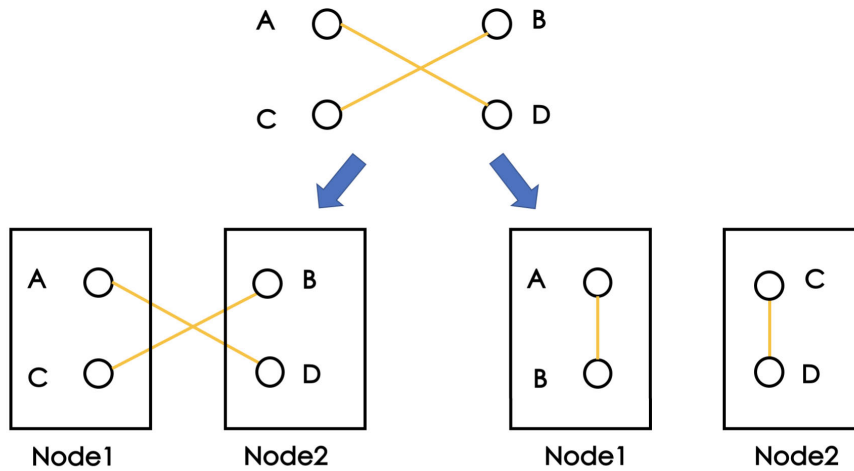
- We abstract the orchestration of simulation tasks as graph partition problem. The tasks can be viewed as vertices and are partitioned according to the amount of computing nodes.
- We carefully design a task orchestration algorithm based on communication patterns of tasks to achieve minimum communication cost.
- The experimental results demonstrate that our proposal has advantages in realizing high-performance simulation.

## II. BACKGROUND

### A. ELECTROMAGNETIC TRANSIENT SIMULATION

Electromagnetic transient simulation is a method of simulating the electromagnetic transient processes in power systems ranging from microseconds to seconds through numerical methods [20], [21]. The simulation of electromagnetic transient processes must take into account the nonlinear characteristics of the power system. For instance, when analyzing transmission lines, it is necessary to consider their distributed parameter characteristics and the electromagnetic aspects within generators, including non-linear features. Therefore, accurate modeling of the various characteristics of different models is essential for precisely representing the relevant components in the power system, thereby accurately capturing the dynamic characteristics of the power system. The world's first commercial digital real-time simulator (Real Time Digital Simulator, RTDS) [22], developed by the Manitoba HVDC Research Centre in Canada, is based on the classic electromagnetic transient solution theory established by H.W. Dommel in 1969. This simulator provided technical support for the hardware-in-the-loop testing of converter controllers in early high-voltage direct current (HVDC) transmission systems [12].

The development of real-time simulators has revolutionized research in power systems, yet the field faces several evolving challenges. The integration of large-scale renewable energy sources adds complexity to the dynamic analysis of power systems, necessitating a delicate balance between simulation efficiency, stability, and accuracy. Critical to this is the selection of appropriate numerical integration algorithms for effectively solving nonlinear stiff systems. Additionally, the incorporation of numerous power electronic devices, each with distinct dynamic characteristics, presents a dilemma: smaller simulation step sizes enhance accuracy but increase computational load, while larger steps may reduce precision. Furthermore, as power systems expand in scale and diversity, ensuring the efficiency and accuracy of simulators becomes more challenging. Simulators must not only handle complex and voluminous data for accurate mathematical modeling but also maintain real-time performance without overwhelming the hardware's computational resources. These challenges underscore the need for continued advancement in simulation methodologies to keep pace with the rapidly evolving power system landscape.



**FIGURE 1.** The motivation of our proposal. Different task orchestrations on computing nodes result in different communication cost.

**B. EXECUTION ON COMPUTER SYSTEM**

The electromagnetic transient simulation can be conceptualized as a collaborative effort involving multiple tasks that jointly execute computational operations. In order to effectively implement this simulation within a computer system, these tasks are best abstracted as distinct processes. The core of the computation hinges on the intricate synchronization among these processes, ensuring a cohesive and efficient simulation workflow. The specific procedure is listed in Algorithm 1.

**Algorithm 1** Electromagnetic Transient Simulation

```

Input: Task id task_id, iteration steps steps, time interval interval
Output: Simulation Results
1: bind_core(task_id)
2: clock ← clock_synchronization()
3: for i ← 1, 2, ..., steps do
4:   wait_until(clock + i * interval)
5:   receive()
6:   execute()
7:   send()
8: end for
    
```

The function presents the execution of tasks in real-time electromagnetic transient simulations. Instantiated with parameters such as *task\_id*, *steps*, and *interval*, this function describes a coarse simulation steps. The *task\_id* parameter uniquely identifies the computational task, *steps* enumerates the temporal iterations of the simulation, and *interval* indicates the time span between these iterations.

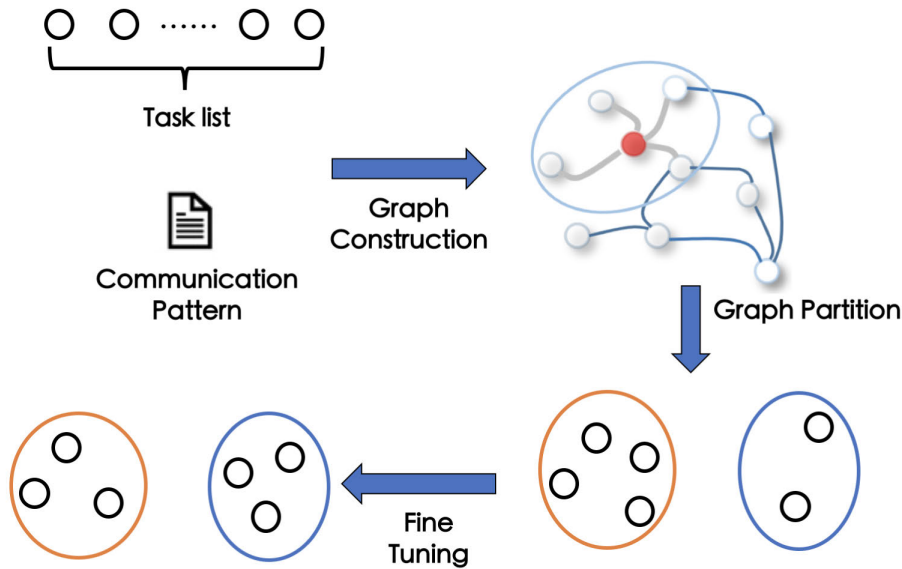
Commencing with the *bind\_core(task\_id)* procedure, the function initially allocates a specific CPU core to the computing process, a critical step in implementing real-time simulation. This CPU binding is essential as it reduces the overhead associated with process switching, particularly

when the operating system handles clock interrupts, thereby significantly enhancing efficiency. Another crucial step in the process is clock synchronization, which ensures that all tasks commence their computations simultaneously. This synchronization is vital for maintaining the coherence and accuracy of the real-time simulation.

Entering the concrete simulation, the function encompasses a loop, methodically crafted to execute over a pre-defined number of steps, each symbolizing a discrete temporal step of the simulation. The *wait\_until(clock + i \* interval)* command within each iteration is pivotal, enforcing the real-time constraints essential for precise modeling. Each loop iteration is further characterized by a series of functions—*receive()*, *execute()*, and *send()*. These functions collectively manage the input data gathering, execution of core simulation computations, and scatter of results, respectively.

**C. MOTIVATION**

The simulation procedure of each task in *compute* function would bring communication cost through *receive()* and *send()*. During simulation, each task has certain communication pattern that it would only send data to other fixed tasks. As depicted in the upper section of Figure 1, there are four tasks denoted as A, B, C, and D. The communication pattern is such that tasks A and D engage in mutual communication, while tasks B and C also communicate with each other. We consider to perform a two-node distributed simulation. As illustrated at the bottom of Figure 1, there are two potential orchestrations for the four tasks. If tasks A and C are allocated to one node and tasks B and D to another, this configuration will result in intensive communications during the simulation. Conversely, if tasks A and B are assigned to one node and tasks C and D to the other, the situation is markedly different, leading to a scenario with no inter-node communication. Consequently, based on our analysis,



**FIGURE 2.** The overview of our proposal. Our proposal has three steps: (a) Graph Construction, which involves compiling a task list and their communication patterns. (b) Graph Partitioning to suit a distributed computing environment. (c) Fine-tuning to adapt partitions to meet the computing resource needs of each task.

it is evident that different task orchestrations lead to varying levels of communication costs, which can adversely affect the performance of real-time simulations. Therefore, it is imperative to develop a task orchestration algorithm that minimizes communication overhead to the greatest extent possible, thereby facilitating the realization of efficient real-time simulation.

### III. METHODOLOGY

This section offers a comprehensive insight into our strategy for achieving efficient communication synchronization, consisting of three fundamental components: graph construction, graph partitioning, and fine-tuning, illustrated in Figure 2. For brevity, we post the significant notations of this paper in Table 1.

#### A. GRAPH CONSTRUCTION

Electromagnetic transient simulation involves a sequence of tasks represented as  $T = \{t_1, t_2, \dots, t_n\}$ . In implementation, each task is assigned to a dedicated process within the operating system. Each process is then bound to a CPU core, ensuring that inter-process scheduling and context switching do not impact the simulation. Moreover, during the simulation, these tasks have to communicate with each other to achieve computation collaboratively. These communications have certain pattern  $B \in \mathcal{R}^{n \times n}$ .  $B_{i,j}$  indicates the data volume to be transmitted from task  $t_i$  to task  $t_j$ .

Based on the communication pattern denoted as  $B$ , these tasks can be abstracted into a graph  $G(V, E, W)$ , abbreviated as  $G$ . Each task is represented a vertex in the graph, and if two tasks have communication link, there is an edge between them. Each task has varying communication data volumes,

**TABLE 1.** Significant notations.

Notations	Description
$T$	Task list in electromagnetic transient simulation
$n$	Number of tasks in electromagnetic transient simulation
$t_i$	The $i$ -th task in $T$
$B \in \mathcal{R}^{n \times n}$	Task communication pattern
$G(V, E, W)$	Graph constructed from $T$ and $B$
$N$	Total number of computing nodes
$P$	Partitions derived from graph $G$
$G_i$	The $i$ -th partition in $P$ ( $1 \leq i \leq N$ )
$n_i$	The number of vertex in $G_i$
$\hat{G}(\hat{V}, \hat{E}, \hat{W})$	The hyper graph constructed from partitions $P$
$M_i$	The $i$ -th computing node
$c_i$	The number of CPU cores in the $i$ -th computing node

and we normalize their transmitted data volumes to a range between 0 and 1, representing communication costs, using the following formula. We use the communication costs as the weight of graph.

$$W_{i,j} = \frac{B_{i,j}}{\max\{B\}} \quad (1)$$

The specific construction is described in Algorithm 2. The time complexity of the algorithm is  $O(n^2)$

#### B. GRAPH PARTITION

After completing the process outlined in Section III-A to transform the simulation tasks into a graph, we derive the weighted directed acyclic graph  $G$ . In large-scale electromagnetic transient simulation, it is inevitable to distribute these tasks across various computing nodes to meet the CPU core requirements of each task. Specifically, our proposal divide graph  $G$  into  $N$  partitions  $P = \{G_1, G_2, \dots, G_N\}$

**Algorithm 2** Graph Construction

**Input:** Task list  $T = \{t_1, t_2, \dots, t_n\}$ , communication pattern  $B$

**Output:** Graph  $G(V, E, W)$

```

1:  $V \leftarrow T$ 
2:  $m \leftarrow \max\{B\}$ 
3: for  $i \leftarrow 1, 2, \dots, n$  do
4:   for  $j \leftarrow 1, 2, \dots, n$  do
5:     if  $B_{i,j} > 0$  then
6:       add edge  $\langle t_i, t_j \rangle$  into  $E$ 
7:        $W_{i,j} \leftarrow B_{i,j}/m$ 
8:     end if
9:   end for
10: end for

```

according to the number of computing nodes. We can build a hyper graph  $\hat{G}(\hat{V}, \hat{E}, \hat{W})$  over these partitioned subgraphs  $\{G_1, G_2, \dots, G_N\}$  based on the following algorithm:

**Algorithm 3** Hyper Graph Construction

**Input:** task graph  $G$ , partition  $P = \{G_1, G_2, \dots, G_N\}$

**Output:** hyper graph  $\hat{G}(\hat{V}, \hat{E}, \hat{W})$

```

1:  $\hat{W} \leftarrow 0$ 
2: for  $i \leftarrow 1, 2, \dots, N$  do
3:   for  $j \leftarrow 1, 2, \dots, N$  do
4:      $\forall u \in G_i$ , and  $\forall v \in G_j$   $\{u$  and  $v$  are nodes in  $G_i$  and  $G_j\}$ 
5:     if  $W_{u,v} > 0$  then
6:        $\hat{W}_{i,j} \leftarrow \hat{W}_{i,j} + W_{u,v}$ 
7:     end if
8:   end for
9: end for

```

Therefore, the communication cost of the partition  $P$  is formulated as follows:

$$C(P) = \sum_i^N \sum_j^N \hat{W}_{i,j} \quad (2)$$

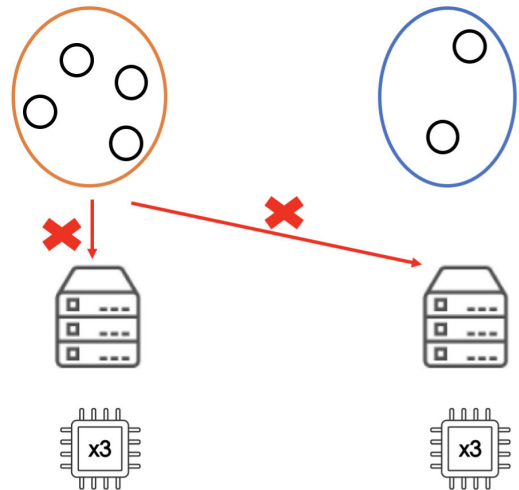
and the purpose of graph partitioning is to identify a partition  $P^*$  that minimizes the above equation:

$$P^* = \arg \max_P C(P) \quad (3)$$

The graph partition is proved as NP-hard problem by many previous researches [23], [24]. Fortunately, METIS [25], [26] is the state-of-the-art tool known for its ability to efficiently partition large-scale graphs while aiming for balanced partitions, minimizing edge cuts, and ensuring scalability. It employs a multilevel approach and offers high-quality partitioning for diverse graph structures. Thus, we resort to METIS to solve the equation 3.

**C. FINE TUNING**

Upon performing graph partitioning, we obtain the optimized partitions, denoted as  $P^*$ . The subsequent step involves



**FIGURE 3.** The mismatch between partitions and computing resources. One partition consists of 4 tasks, while the other has 2 tasks. However, there are two computing nodes, each with 3 CPU cores. Consequently, the partition with 4 tasks cannot be assigned to any nodes, as each task requires a dedicated CPU core.

allocating computing nodes to these partitions. However, a perfect alignment between the divided partitions and the computing nodes may not always be achievable. For instance, consider a scenario where the number of tasks in partition  $G_i (i = 1, 2, \dots, N)$  is represented by  $n_i$ , and the number of CPU cores in computing node  $M_j (j = 1, 2, \dots, N)$  is denoted as  $c_j$ . It is possible that a given  $G_i$  may not match any  $M_j$  if  $n_i > c_j$ , a situation illustrated in Figure 3.

We introduce a fine tuning algorithm to adjust the partition according to available CPU cores of computing nodes. Initially, we apply the best fit algorithm [27] to achieve the assignment of partitions. The specific procedure is shown in Algorithm 4. The best fit assignment first arranges the partitions  $G_1, G_2, \dots, G_N$  (Line 1). Then, it seeks a computing node  $M_k$  with the lowest  $c_m$  that satisfies the condition  $c_m \geq |\hat{G}_i|$  to accommodate the arranged  $\hat{G}_i$  (Lines 6 ~ 14). Nevertheless, the best fit method might not allocate the partitions optimally to computing nodes since a partition could contain more tasks than the available CPU cores in any computing node. This results in the partition cannot be match to current computing resources.

From the analysis above, the next step involves adjusting the partitions. The central idea of this adjustment is transferring tasks from the larger partition to the smaller one. It's crucial to note that this task transfer will influence the communication costs, and the primary concern is to minimize these costs as much as possible. To execute the task transfer as described, we adhere to the principle that a task can only be transferred to a partition if a communication link exists between them, illustrated in Figure 4.

*Theorem 1:* The above task transfer scheme is able to maintain minimum increase in communication overhead.

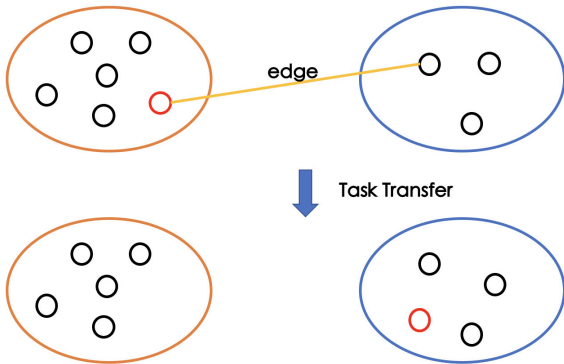
*Proof of Theorem 1:* There two partitions  $G_i$  and  $G_j$ , and task  $u \in G_i$ . We assume  $\forall v \in G_j$ , task  $u$  and  $v$  has no

**Algorithm 4** Best Fit Assignment

**Input:** partition  $P = \{G_1, G_2, \dots, G_N\}$ , computing nodes  $\{M_1, M_2, \dots, M_N\}$  with CPU cores  $\{c_1, c_2, \dots, c_N\}$

**Output:**  $A$ : assignment of partition

- 1:  $\{\hat{G}_1, \hat{G}_2, \dots, \hat{G}_N\} \leftarrow$  sort  $\{G_1, G_2, \dots, G_N\}$  with ascending number of nodes.
- 2:  $explore[1 \dots N] \leftarrow False$
- 3: **for**  $i \leftarrow 1, 2, \dots, N$  **do**
- 4:    $best \leftarrow \infty$
- 5:    $k \leftarrow -1$
- 6:   **for**  $j \leftarrow 1, 2, \dots, N$  **do**
- 7:     **if**  $c_j \geq n_i$  and  $c_j < best$  and  $explore[j] == False$  **then**
- 8:        $best \leftarrow c_j$
- 9:        $k \leftarrow j$
- 10:     **end if**
- 11:   **end for**
- 12:   **if**  $best < \infty$  **then**
- 13:      $explore[k] \leftarrow True$
- 14:      $A[i] \leftarrow k$
- 15:   **end if**
- 16: **end for**



**FIGURE 4.** The process of task transfer. If there is a communication link between two tasks in separate partitions, one task can be transferred to the other partition.

communication link. If we move task  $u$  to partition  $G_j$ , the increased communication overhead is

$$H_1 = \sum_{k \in G_i} (W_{u,k} + W_{k,u}) \quad (4)$$

However, if we assume there exists task  $v \in G_j$  that has communication link with task  $u$ , the increased communication overhead is

$$H_2 = \sum_{k \in G_i} (W_{u,k} + W_{k,u}) - (W_{u,v} + W_{v,u}) \quad (5)$$

It is obvious that  $H_2 < H_1$ , and from the above analysis, the task transfer scheme can maintain the minimum increased communication overhead. ■

Theorem 1 establishes the strategy for partition adjustment. Subsequently, the decision on which tasks to transfer is

**Algorithm 5** Fine Tuning

**Input:** partition  $P = \{G_1, G_2, \dots, G_N\}$ , computing nodes  $\{M_1, M_2, \dots, M_N\}$  with CPU cores  $\{c_1, c_2, \dots, c_N\}$ , incomplete assignment  $A$ , initial temperature  $T_{max}$ , cooling rate  $\alpha$ , minimum temperature  $T_{min}$

**Output:** adjusted partition  $P'$  and complete assignment  $A$

- 1: **for all** partition  $G_i (1 \leq i \leq N)$  that  $\forall c_j \in \{c_1, c_2, \dots, c_N\}, n_i > c_j$  **do**
- 2:    $T' \leftarrow T_{max}$
- 3:   **while**  $T' \geq T_{min}$  **do**
- 4:     Select a partition  $G_j (j \neq i)$  that  $\hat{W}_{i,j} > 0$  and  $c_{A[j]} > n_j$ .
- 5:     Choose a task  $u$  in  $G_i$  that has communication link in the task of  $G_j$
- 6:     If task  $u$  has not been moved from  $G_j$ , transfer task  $u$  to partition  $G_j$  and obtain new partition  $P'$ .
- 7:     Compute  $C(P')$  via equation 2.
- 8:      $\Delta E \leftarrow C(P') - C(P)$
- 9:     **if**  $\Delta E \leq 0$  **then**
- 10:       Accept this transfer
- 11:     **else**
- 12:       Accept this transfer with the probability  $e^{-\Delta E/T_{max}}$
- 13:     **end if**
- 14:     **if** Accept this transfer **then**
- 15:        $P \leftarrow P'$
- 16:     **end if**
- 17:      $T' \leftarrow \alpha \cdot T'$
- 18:   **end while**
- 19: **end for**
- 20: **for all** new partition  $G'_i (1 \leq i \leq N)$  that  $A[i] == \emptyset$  **do**
- 21:    $best \leftarrow \infty$
- 22:    $k \leftarrow -1$
- 23:   **for**  $j \leftarrow 1, 2, \dots, N$  **do**
- 24:     **if**  $c_j \geq n_i$  and  $c_j < best$  and no partition is allocated to  $M_j$  **then**
- 25:        $best \leftarrow c_j$
- 26:        $k \leftarrow j$
- 27:     **end if**
- 28:   **end for**
- 29:    $A_i \leftarrow k$
- 30: **end for**

made to fulfill the CPU resource requirements of each partition while minimizing the increase in communication costs. The time complexity of the vanilla algorithm is  $O(n^N)$ , making it time-consuming. To achieve effective task transfer, Algorithm 5 employs simulated annealing [28], [29]. We first enumerate each oversized partition (Line 1), and then use simulated annealing strategy to transfer the tasks in selected partition to others. The algorithm according to the equation 2 to determine whether to accept this transfer. Upon accepting this transfer, the task will not be moved back. Consequently, we obtain a new partition, denoted as  $P'$ , and proceed to

the next phase of the process. After the task transfer, the algorithm (Line 20 ~ 26) assigns computing nodes to those partitions which were not previously allocated, owing to their unmet computing requirements.

#### D. PUT EVERYTHING TOGETHER

According to the algorithms described in Section 2 to III-C, the pipeline of our task orchestration is presented in Figure 5. We begin by applying Algorithm III-A with task lists and communication patterns as inputs. Subsequently, we use METIS to generate graph partitions. We then employ Algorithm 4 to formulate a tentative allocation plan. In case any partition is not assigned to a computing node, we invoke Algorithm 5 to adjust partitions and derive new task allocations, ensuring each task occupies a CPU core with minimal communication overhead. After determining each task's CPU core, these tasks can commence on simulation.

#### IV. EVALUATION

Our experiment was conducted using a hardware setup consisting of four nodes interconnected to form a distributed system. Within this setup, we performed two performance evaluation tests to assess system efficiency and scalability. Additionally, we employed five concurrent processes to efficiently utilize the computational resources and enable parallel execution of a simulation task. These simulation tasks were conducted using five different models, allowing us to evaluate the effectiveness and accuracy of our approach. The experimental results obtained from this hardware configuration provide valuable insights into the performance and capabilities of our system.

##### A. HARDWARE EQUIPMENT

Our experimental setup involved four dual-socket nodes, each powered by Intel Xeon Gold 6346 processors with a 3.1GHz operating frequency. These nodes were each furnished with 64 GBytes of main memory, distributed as 32 GBytes per socket. Network connectivity was established using Mellanox ConnectX-5 VPI NICs, offering 100 GBit/s bandwidth via FDR InfiniBand. A Mellanox SB7800 switch was used for node interconnections, providing high-speed, reliable communication. To mitigate any NUMA-related performance inconsistencies, all experiments were conducted on sockets directly connected to the network cards. For precise time synchronization, critical for ensuring coordinated data exchange during simulations, the Beidou Time Service [30] was employed, maintaining accurate timing across the distributed system nodes.

##### B. SOFTWARE SETUP

In our software configuration, we employed KylinSec, a real-time operating system designed by the National University of Defense Technology [31]. This system, tailored for the x86\_64 architecture, operates on a kernel derived from Linux version 5.10.0-60.18.0.50.kb7.ky3.x86\_64. To facili-

tate RDMA communication, we integrated key libraries and drivers, including librdmacm, ibverbs, and mlx5, enabling efficient data transfer via Remote Direct Memory Access (RDMA). For enhanced experiment performance, the software was compiled using GCC (GNU Compiler Collection) version 10.3.1 with the "-O2" optimization flag. This strategic amalgamation of specific OS components, RDMA support, and compiler settings is instrumental in achieving compatibility and high-performance outcomes in our setup.

##### C. COMMUNICATION EXPERIMENT

We conducted performance evaluation tests to measure the communication time required in the distributed system utilized by our task orchestration method. The code is revised as follows:

---

##### Algorithm 6 Performance Evaluation of Communication

---

**Input:** Task id  $task\_id$ , iteration steps  $steps$ , time interval  $interval$

**Output:** Statistics  $avg$ ,  $min$ , and  $max$

```

1: initialize  $cost[1 \dots interval]$ 
2:  $bind\_core(task\_id)$ 
3:  $clock \leftarrow clock\_synchronization()$ 
4: for  $i \leftarrow 1, 2, \dots, steps$  do
5:    $wait\_until(clock + i * interval)$ 
6:    $t_1 \leftarrow time()$ 
7:    $receive()$ 
8:    $send()$ 
9:    $t_2 \leftarrow time()$ 
10:   $cost[i] \leftarrow t_2 - t_1$ 
11: end for
12:  $avg \leftarrow mean(cost)$ 
13:  $min \leftarrow min(cost)$ 
14:  $max \leftarrow max(cost)$ 

```

---

We remove the function `execute()`, and reserve `receive()` and `send()` for statistics of communication cost.  $t_1$  and  $t_1$  record the start time and end time of communication, thus  $cost$  is computed by  $t_2 - t_1$ . We collect the communication cost of each iteration, and  $avg$ ,  $min$  and  $max$  respectively record the average, minimum and maximum communication cost of the total  $steps$  iterations.

- 1) **Communication between Four Tasks on Two Nodes:** We measured the time required for communication of  $m$  KB between four tasks in the distributed system with two nodes. The value of  $m$  varied from 0.5 to 2 KB in steps of 0.5 KB. The communication pattern of these four tasks (denoted as A, B, C, D) are presented in Table 2.  $m$  indicates there are  $m$  KB data transfer between two tasks in each iteration.
- 2) **Communication Among Eight Tasks on Four Nodes:** We also evaluated the time required for communication of  $m$  KB among all eight tasks in the distributed system with four nodes. Similarly, the value of  $m$  varied from 0.5 to 2 KB in steps of 0.5 KB. The communication

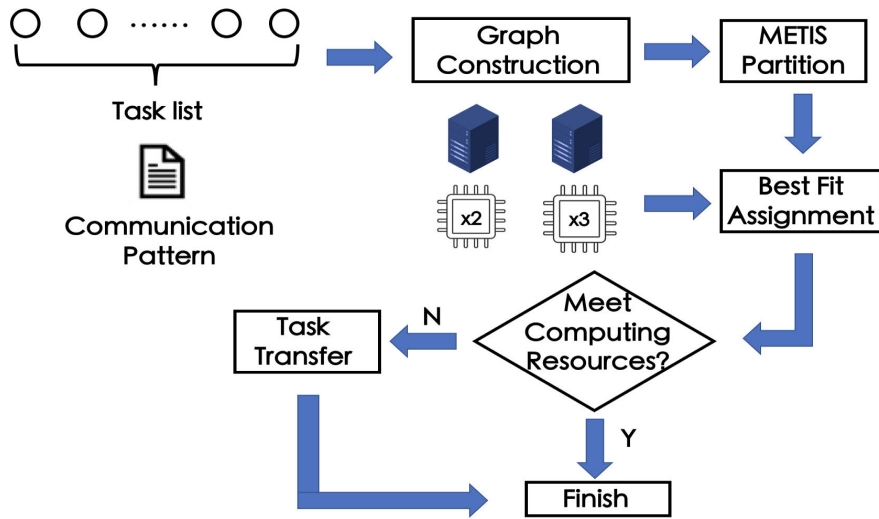


FIGURE 5. The pipeline of our task orchestration.

TABLE 2. Communication pattern of four tasks.

	A	B	C	D
A	0	0	m	0
B	0	0	0	m
C	m	0	0	0
D	0	m	0	0

pattern of these eight tasks (denoted as A, B, C, D, E, F, G, H) are presented in Table 3. m indicates there are m KB data transfer between two tasks in each iteration.

We present the averaged metrics of all tasks in Table 4. Our iteration steps are set to  $10^6$ , and from the results, it is evident that, through our task orchestration method, the communication costs are minor, especially in the four-task scenario where communication is confined to the same node.

To demonstrate the effectiveness of our proposed orchestration method, we compare it with random orchestration under identical experimental conditions. The results are depicted in Figure 6. In the case of four tasks distributed across two nodes, our proposed method exhibits significantly lower communication costs compared to random orchestration. This is attributed to our task orchestration method’s capability to place tasks involving mutual communication into the same computing node, while random orchestration may fail to eliminate cross-node communication. Similarly, in the more complex scenario with eight tasks distributed across four nodes, our method still outperforms random orchestration.

**D. SIMULATION EXPERIMENT**

In Figure 7, we depict the performance experiment of our proposed orchestration method applied to a real simulation in a distributed environment with two nodes. This simulation involves five tasks that engage in mutual communication. These tasks correspond to five computational functions: Angle, Gain, Cos, Sin, and Add. Each task is assigned to a dedicated CPU core. To validate our task orchestration

TABLE 3. Communication pattern of eight tasks.

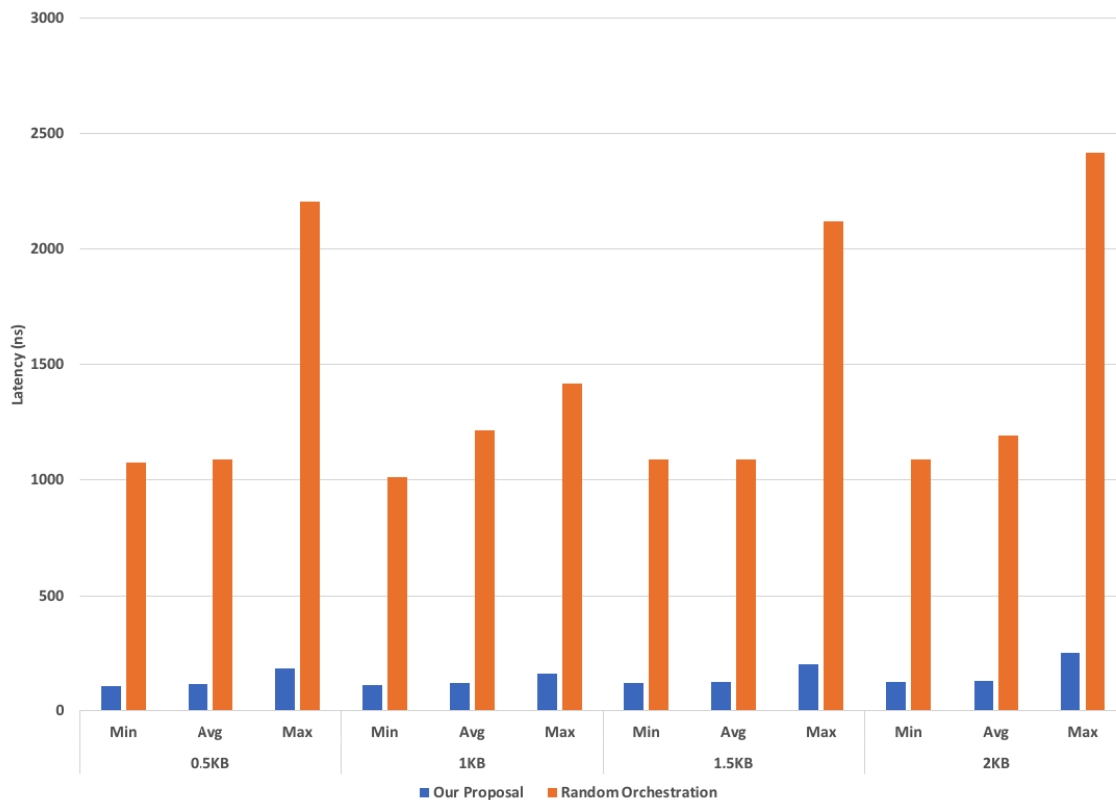
	A	B	C	D	E	F	G	H
A	0	0	m	0	0	0	m	0
B	0	0	0	m	m	0	0	0
C	m	0	0	0	0	0	0	m
D	0	m	0	0	0	m	0	0
E	0	m	0	0	0	0	0	0
F	0	0	0	m	0	0	0	0
G	m	0	0	0	0	0	0	0
H	0	0	m	0	0	0	0	0

method, we impose constraints, allowing for two and three CPU cores to be available in the two nodes, respectively. The data size of each task sending to its receiver is m KB in every iteration (The value of m varied from 0.5 to 2 KB in steps of 0.5 KB as above).

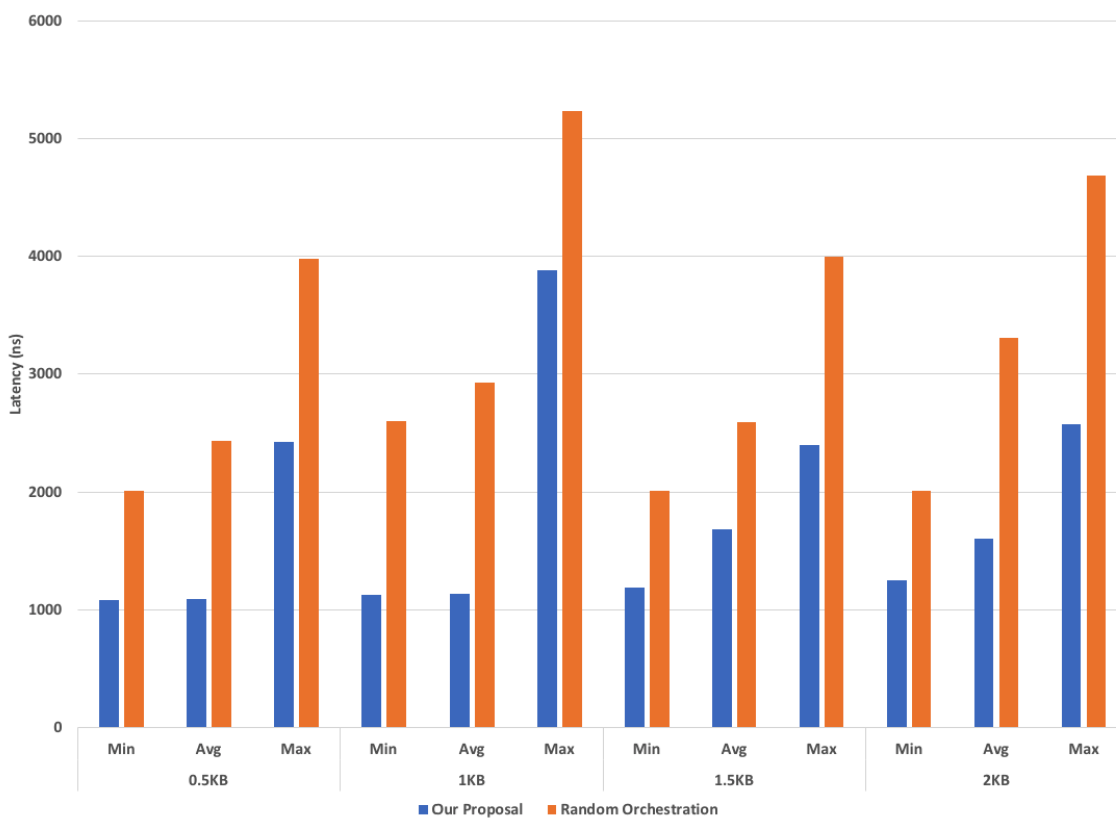
The Angle function generates an output that is transmitted to both the Gain and Sin functions. The Gain function receives the input from the Angle function and sends its output to the Cos function. Similarly, the Sin function receives the input from the Angle function and transmits its output to the Add function. The Cos function receives the input from the Gain function and performs the required computation, sending the results to the Add function. The Add function receives the results from both Sin and Cos functions and performs the necessary computation, recording the final result.

These tasks are expected to allocated to a CPU core within the two nodes, and we apply our task orchestration and random orchestration. After applying task orchestration methods, the results are presented in Figure 8. The left part of the Figure is our proposal. The Sin and Add functions are allocated to the node with two CPU cores while Angle, Gain and Cos functions are on the node with three CPU cores. Our task orchestration method has only twice communications during each iteration. The right part of the Figure is random orchestration, The Gain and Sin functions are on the node with two CPU cores, while Angle, Cos and Add functions are on the node with three CPU cores. This allocation will





(a) Performance evaluation of four tasks in two nodes



(b) Performance evaluation of eight tasks in four nodes

**FIGURE 6.** Performance evaluation of our proposed task orchestration and random orchestration.

TABLE 4. Communication costs of our task orchestration method in different test scenarios.

No.	Test Content	Steps	Min (ns)	Avg (ns)	Max (ns)
1	The communication time of data transfer of 0.5 KB between four tasks in the distributed system with two nodes.	$10^6$	104.6	115.4	184.6
	The communication time required for data transfer of 0.5 KB between eight tasks in the distributed system with four nodes.	$10^6$	1088	1091	2418
2	The communication time required for data transfer of 1 KB between four tasks in the distributed system with two nodes.	$10^6$	112.5	119.1	158.7
	The communication time required for data transfer of 1 KB between eight tasks in the distributed system with four nodes.	$10^6$	1126	1139	3877
3	The communication time required for data transfer of 1.5 KB between four tasks in the distributed system with two nodes.	$10^6$	118.4	125.3	201.4
	The communication time required for data transfer of 1.5 KB between eight tasks in the distributed system with four nodes.	$10^6$	1188	1690	2396
4	The communication time required for data transfer of 2 KB between four tasks in the distributed system with two nodes.	$10^6$	123.3	129.6	251.4
	The communication time required for data transfer of 2 KB between eight tasks in the distributed system with four nodes.	$10^6$	1258	1611	2575

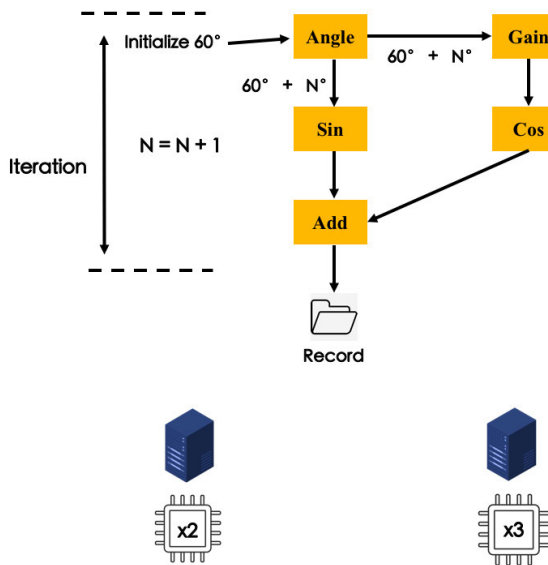


FIGURE 7. Simulation example: Five tasks—Angle, Gain, Cos, Sin, and Add—collaboratively participate in a simulation. Each of these tasks is allocated to a dedicated CPU core across two nodes. The available CPU cores of two nodes are 2 and 3, respectively.

result in four times cross-node communication, which are much larger than our proposed orchestration.

According to the results of task orchestration, we record the performance of both methods. In contrast to Section IV-C, we include the `execute()` cost in our statistics, and the code is shown in Algorithm 7.

We choose the number of iterations for the simulation to be  $10^6$ . The performance evaluation is shown in Table 5

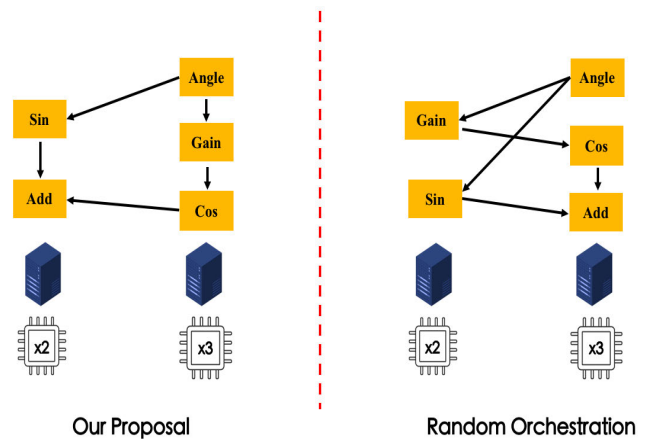


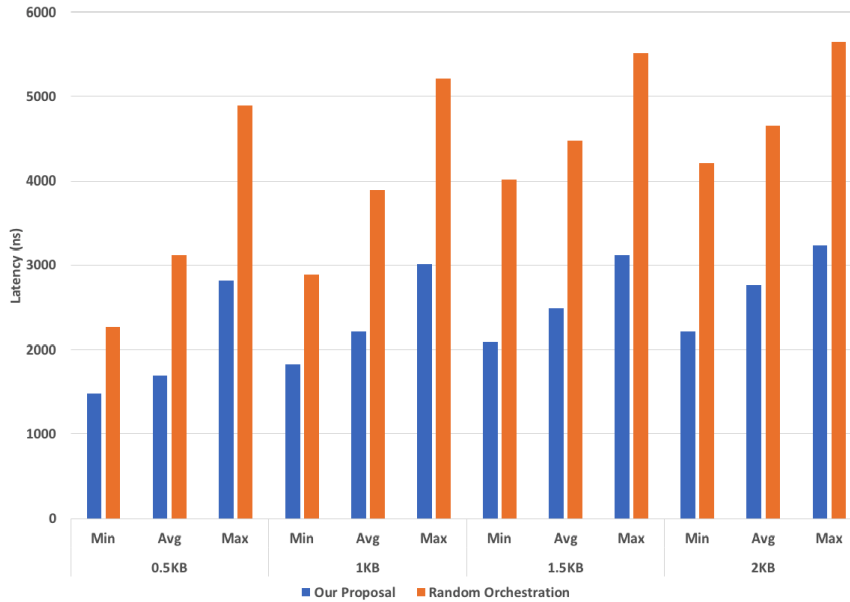
FIGURE 8. The results of orchestration through our proposal and random orchestration. Our task orchestration method has fewer communications than random orchestration.

and Figure 9. From the table and figure, it is obvious that our proposal outperforms random orchestration. Across all tested data transfer sizes, our method achieves lower communication and execution times.

In conclusion, the simulation experiment demonstrates the effectiveness and importance of our communication-efficient task orchestration in enabling synchronized and efficient interaction among the Angle, Gain, Cos, Sin, and Add functions. The optimized communication scheme ensures precise and timely data exchange, leading to accurate real-time simulations of electromagnetic transients within each time step.

**TABLE 5.** Performance of our task orchestration method and random orchestration.

Data Size	Method	Steps	Min(ns)	Avg(ns)	Max(ns)
0.5 KB	Our proposal	$10^6$	1488	1691	2818
	Random Orchestration	$10^6$	2271	3124	4892
1 KB	Our proposal	$10^6$	1826	2212	3012
	Random Orchestration	$10^6$	2892	3892	5212
1.5 KB	Our proposal	$10^6$	2088	2491	3123
	Random Orchestration	$10^6$	4022	4476	5512
2 KB	Our proposal	$10^6$	2212	2764	3233
	Random Orchestration	$10^6$	4212	4652	5652

**FIGURE 9.** The performance evaluation on simulation of 5 tasks within two nodes.**Algorithm 7** Performance Evaluation of Real Simulation

**Input:** Task id  $task\_id$ , iteration steps  $steps$ , time interval  $interval$

**Output:** Statistics  $avg$ ,  $min$ , and  $max$

```

1: initialize  $cost[1 \dots interval]$ 
2: bind_core( $task\_id$ )
3: clock  $\leftarrow$  clock_synchronization()
4: for  $i \leftarrow 1, 2, \dots, steps$  do
5:   wait_until(clock +  $i * interval$ )
6:    $t_1 \leftarrow$  time()
7:   receive()
8:   execute()
9:   send()
10:   $t_2 \leftarrow$  time()
11:   $cost[i] \leftarrow t_2 - t_1$ 
12: end for
13:  $avg \leftarrow$  mean( $cost$ )
14:  $min \leftarrow$  min( $cost$ )
15:  $max \leftarrow$  max( $cost$ )

```

**V. CONCLUSION**

Electromagnetic transient simulation plays an indispensable role in the analysis and design of power systems, offering

insights into the dynamic behavior of these complex systems. Recently, distributed electromagnetic transient simulation has emerged as a novel trend, particularly driven by the need to manage large-scale appliances and models within power systems. This approach, however, brings its own set of challenges to realize real-time simulation, one of them being the communication cost – a significant factor in distributed scenario. In such simulations, numerous tasks are required to perform computations collaboratively, which inevitably leads to communication overhead, especially for synchronization purposes between tasks.

In this paper, we propose a novel task orchestration method aimed at optimizing task allocation across different computing nodes, thereby minimizing communication costs. Our method views these tasks as nodes within a graph and consists of three components: Graph Construction, Graph Partition, and Fine Tuning. The Graph Construction involves mapping the tasks to a graph structure to represent the communication and dependency relationships. The Graph Partition component is designed to divide this graph into sub-graphs, strategically distributing tasks to different computing nodes in a way that minimizes inter-node communication. Lastly, the Fine Tuning phase addresses scenarios where certain sub-graphs might initially not be allocated to any

computing nodes. It does this by implementing a task transfer strategy, which ensures that all tasks are appropriately assigned to computing resources while maintaining minimum communication cost.

The effectiveness of our proposed method is validated through extensive experimental results. These results demonstrate our proposal can greatly reduce communication overhead.

## REFERENCES

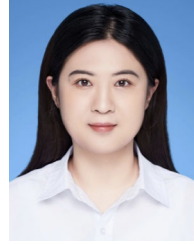
- [1] J. D. Smith, "Real-time simulation of electromagnetic transients," *Electric Power Syst. Res.*, vol. 45, no. 2, pp. 112–125, 2021.
- [2] L. Barbierato, E. Pons, E. F. Bompard, V. S. Rajkumar, P. Palensky, L. Bottaccioli, and E. Patti, "Exploring stability and accuracy limits of distributed real-time power system simulations via system-of-systems cosimulation," *IEEE Syst. J.*, vol. 17, no. 2, 2023.
- [3] H. Bai, C. Liu, E. Breaz, K. Al-Haddad, and F. Gao, "A review on the device-level real-time simulation of power electronic converters: Motivations for improving performance," *IEEE Ind. Electron. Mag.*, vol. 15, no. 1, pp. 12–27, Mar. 2021.
- [4] F. A. Silva, "Real-time electromagnetic transient simulation of AC–DC networks [book news]," *IEEE Ind. Electron. Mag.*, vol. 16, no. 3, pp. 101–102, Sep. 2022.
- [5] F. Guo, L. Herrera, R. Murawski, E. Inoa, C.-L. Wang, P. Beauchamp, E. Ekici, and J. Wang, "Comprehensive real-time simulation of the smart grid," *IEEE Trans. Ind. Appl.*, vol. 49, no. 2, pp. 899–908, Mar. 2013.
- [6] Q. Wang, J. Xu, K. Wang, P. Wu, W. Chen, and Z. Li, "Parallel electromagnetic transient simulation of power systems with a high proportion of renewable energy based on latency insertion method," *IET Renew. Power Gener.*, vol. 17, no. 1, pp. 110–123, Jan. 2023.
- [7] B. Bruned, J. Mahseredjian, S. Denetière, J. Michel, M. Schudel, and N. Bracikowski, "Compensation Method for parallel and iterative real-time simulation of electromagnetic transients," *IEEE Trans. Power Del.*, vol. 38, no. 4, pp. 2302–2310, 2023.
- [8] X. Maya, L. Garcia, A. Vazquez, E. Pichardo, J.-C. Sanchez, H. Perez, J.-G. Avalos, and G. Sanchez, "A high-precision distributed neural processor for efficient computation of a new distributed FxSMAP-L algorithm applied to real-time active noise control systems," *Neurocomputing*, vol. 518, pp. 545–561, Jan. 2023.
- [9] B. Cao, K. Su, L. Miao, L. Niu, Y. Song, and Z. Yu, "Real-time electromagnetic transient simulation for regional power grid based on cloudpss," in *Proc. 3rd Int. Symp. New Energy Electr. Technol.*, 2023, pp. 740–750.
- [10] F. A. Mourinho and T. M. L. Assis, "Impact of cascade disconnection of distributed energy resources on bulk power system stability: Modeling and mitigation requirements," *J. Modern Power Syst. Clean Energy*, vol. 11, no. 2, pp. 412–420, Mar. 2023.
- [11] J. A. Jardini and A. Gole, "Modelling and simulation studies to be performed during the lifecycle of HVDC systems," *Cigré WG B*, vol. 4, no. 563, pp. 34–38, 2013.
- [12] S. Debnath and J. Sun, "Fidelity requirements with fast transients from VSC-HVdc," in *Proc. 44th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2018, pp. 6007–6014.
- [13] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Cham, Switzerland: Springer, 2004, pp. 97–104.
- [14] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *Proc. 23rd Int. Conf. Distrib. Comput. Syst. Workshops*, May 2003, pp. 200–206.
- [15] H. Gharavi and B. Hu, "Scalable synchrophasors communication network design and implementation for real-time distributed generation grid," *IEEE Trans. Smart Grid*, vol. 6, no. 5, pp. 2539–2550, Sep. 2015.
- [16] Q. Mu, L. Niu, and Y. Cheng, "The communication methodology on the large-scale power system real-time simulation," in *Proc. 16th IET Int. Conf. AC DC Power Transmiss.*, vol. 2020, Jul. 2020, pp. 497–504.
- [17] W. Kang, K. Kapitanova, and S. Hyuk Son, "RDDS: A real-time data distribution service for cyber-physical systems," *IEEE Trans. Ind. Informat.*, vol. 8, no. 2, pp. 393–405, May 2012.
- [18] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *Proc. Annu. Tech. Conf.*, 2016, pp. 437–450.
- [19] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using RDMA efficiently for key-value services," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 295–306.
- [20] M. Griebel, T. Dornseifer, and T. Neunhoffer, *Proc. Numerical Simulation in Fluid Dynamics: A Practical Introduction*. Philadelphia, PA, USA: SIAM, 1998.
- [21] P. Luo, M. Luo, F. Li, X. Qi, A. Huo, Z. Wang, B. He, K. Takara, D. Nover, and Y. Wang, "Urban flood numerical simulation: Research, methods and future perspectives," *Environ. Model. Softw.*, vol. 156, Oct. 2022, Art. no. 105478.
- [22] R. Kuffel, J. Giesbrecht, T. Maguire, R. P. Wierckx, and P. McLaren, "RTDS—A fully digital power system simulator operating in real time," in *IEEE WESCANEX 95. Commun., Power, Computing. Conf. Proc.*, Jul. 1995, pp. 498–503.
- [23] P. A. Papp, G. Anegg, and A.-J.-N. Yzelman, "Partitioning hypergraphs is hard: Models, inapproximability, and applications," in *Proc. 35th ACM Symp. Parallelism Algorithms Architectures*, Jun. 2023, pp. 415–425.
- [24] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proc. 16th Annu. ACM Symp. Parallelism Algorithms Archit.*, Jun. 2004, pp. 120–124.
- [25] G. Karypis and V. Kumar, "METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," Univ. Digit. Conservancy, 1997. [Online]. Available: <https://hdl.handle.net/11299/215346> and <https://conservancy.umn.edu/items/2f610239-590c-45c0-bcd6-321036aaad56>
- [26] D. LaSalle, M. M. A. Patwary, N. Satish, N. Sundaram, P. Dubey, and G. Karypis, "Improving graph partitioning for modern graphs and architectures," in *Proc. 5th Workshop Irregular Appl., Archit. Algorithms*, Nov. 2015, pp. 1–4.
- [27] J. E. Shore, "On the external storage fragmentation produced by first-fit and best-fit allocation strategies," *Commun. ACM*, vol. 18, no. 8, pp. 433–440, Aug. 1975.
- [28] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Stat. Sci.*, vol. 8, no. 1, pp. 10–15, 1993.
- [29] D. Henderson, S. H. Jacobson, and A. W. Johnson, "The theory and practice of simulated annealing," in *Handbook Metaheuristics*. Boston, MA, USA: Springer, 2003, pp. 287–319.
- [30] Y. Xu, W. Wang, X. Yang, K. Deng, and Z. He, "Design and research of power system beidou timing and positioning module based on K-means clustering and gross error processing," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 8, no. 1, pp. 34–42, Mar. 2023.
- [31] M. Zhou, X. Hu, and W. Xiong, "OpenEuler: Advancing a hardware and software application ecosystem," *IEEE Softw.*, vol. 39, no. 2, pp. 101–105, Mar. 2022.



**QI GUO** (Senior Member, IEEE) received the B.E. and Ph.D. degrees from Tsinghua University, in 2000 and 2005, respectively. He is currently the Vice President of the Electric Power Research Institute, China Southern Power Grid. His research interests include control and simulation of renewable energy control, power system stability, and dc transmission.



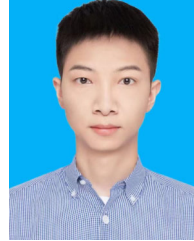
**HAIPING GUO** received the bachelor's and master's degrees from Xi'an Jiaotong University, in 2008 and 2011, respectively. He is currently a Professor-Level Senior Engineer. His research interests include renewable energy control and simulation, power system stability, and dc transmission technologies.



**JINGYUE ZHANG** received the bachelor's degree from North China Electric Power University, in 2020, and the master's degree from Huazhong University of Science and Technology, in 2023. Her research interests include microgrid energy management and reconstruction, and renewable energy simulation.



**YUANHONG LU** received the B.E. and Ph.D. degrees from Beijing Jiaotong University, in 2010 and 2016, respectively. He is currently a Senior Engineer with the Electric Power Research Institute, China Southern Power Grid. His research interest includes control and simulation of renewable energy control.



**HUI LUO** was born in Jiangxi, China, in 1998. He received the B.S. degree in electrical engineering from Central South University (CSU), in 2020, and the M.S. degree in electrical engineering from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2023. His research interests include renewable energy control and simulation, power quality control, active power filters, and PWM harmonic optimization.



**TIANYU GUO** (Member, IEEE) was born in Yunnan, China, in 1994. He received the B.S. and Ph.D. degrees from North China Electric Power University, China, in 2016 and 2021, respectively. His current research interests include control and simulation of renewable energy control, distributed generation and microgrids, and integrated energy systems.



**LIBIN HUANG** received the bachelor's and master's degrees from Huazhong University of Science and Technology, in 1998 and 2003, respectively. His research interests include renewable energy control and simulation, power system stability, and dc transmission technologies.



**JIE ZHANG** received the master's degree from Beijing University of Posts and Telecommunications, in 2019. He is currently an Engineer with the Electric Power Research Institute, China Southern Power Grid. His research interests include power system simulation and machine learning.



**YANJUN ZHAO** was born in 1987. Her research interests include power system analysis and control.

...