IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 February 2023, accepted 9 April 2023, date of publication 12 April 2023, date of current version 19 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3266478

RESEARCH ARTICLE

Multiobjective Approach to Schedule DAG Tasks on Voltage Frequency Islands

SANCHIT[1], NAVJOT SINGH[1], (Senior Member, IEEE), AND JAGPREET SINGH[2]
[1]Indian Institute of Information Technology Allahabad, Allahabad, Uttar Pradesh 211015, India
[2]Indian Institute of Technology Ropar, Ropar, Punjab 140001, India

Corresponding author: Sanchit (tyagisanchit5@gmail.com)

ABSTRACT Scheduling a Directed Acyclic Graph (DAG) on voltage frequency islands involves dividing the available processing units into multiple islands with varying voltage and frequency levels and then mapping the tasks of the DAG to the islands while minimizing the makespan and overall energy consumption. In this research work, a novel DAG task scheduling model is introduced with the assistance acquired from the deep learning paradigm. The proposed model includes four major phases: (a) DAG modelling, (b) Voltage frequency island partitioning, (c) core temperature prediction and (d) scheduling optimization. Initially, the Directed Acyclic Graph (DAG) model is designed. The nodes of DAG represent tasks, and the edges represent dependencies between tasks. Then, in the Voltage frequency island partitioning, the available processing units are into multiple voltage frequency islands. This can be done based on the power consumption of each unit and the task requirements. Subsequently, the Recurrent Neural Network (RERNN) is trained to predict the core temperature of each voltage frequency island based on the multi-objectives like execution time, makespan, and overall energy consumption. Then, the scheduling of the DAG on the voltage frequency islands is optimized using the Self-Improved Pelican Optimization Algorithm (SI-POA). The proposed SI-POA model is an extended version of the standard POA model. The SI-POA model is inspired by the behaviour by the natural behaviour of pelicans during hunting. In the scheduling optimization phase, the SI-POA algorithm optimizes the scheduling of the DAG on the voltage frequency islands while taking into account the predicted core temperature of each island based on the Recalling-enhanced recurrent neural network (RERNN) model. The goal is to minimize the makespan and overall energy consumption of the DAG while keeping the core temperature of each island within safe limits.

INDEX TERMS Scheduling, makespan, voltage frequency island (VFI), reliability, energy consumption, directed acyclic graph (DAG), temperature.

I. INTRODUCTION

Grid computing is regarded as the foundation of the next generation of distributed computing, which organizes the sharing of massive amounts of resources and the resolution of issues in dynamic multi-institutional virtual groups [1]. People can collaborate with one another and share all resources via the Internet without surrendering local autonomy across corporate, institutional, and geographic barriers. Grid workflow is described as the coordination of a collection of discrete actions carried out across a network of resources in a predetermined sequence in order to achieve a significant and complex objective [2]. Directed Acyclic Graph (DAG) is currently widely utilized in modelling scientific computational workflows, particularly large-scale computing-intensive or data-intensive Grid applications, including high-energy physics, geophysics, astronomical, medical image processing, and informatics. two well-known Grid workflow solutions based on DAG [3]. Due to the importance of task scheduling for the performance of applications, it is a well-studied problem. A number of methods have been developed to schedule the nodes of the DAG onto the heterogeneous machines [4]. Applications are often depicted by means of a directed acyclic graph (DAG). List scheduling-based heuristics are among those that offer good schedules at a fair price.

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

The semiconductor industry is now integrating uniprocessors (cores) into a single chip to create multicore systems that can run computationally intensive applications within a given deadline due to the growing need for quicker computations in high-performance computing (HPC) clusters. High consumption of energy becomes detrimental to the performance and reliability of system components as the number of chip cores rises with application size [5]. Applications with different processing demands have been shown to have lower energy efficiency when running on a device with Dynamic Voltage and Frequency Scaling (DVFS), where all cores operate at the same V/F level [6]. Researchers have adopted the use of Voltage/Frequency Islands (VFIs), where a VFI consists of one or even more cores that run at the same V/F level and whose V/F levels may be different from other VFIs, to address multi/manycore energy efficiency trying to run applications with heterogeneous computational and energy requirements [7]. The most energy-efficient architecture is a single-core VFI, but as the number of VFIs increases, so does the complexity of the per-core V/F regulators and the synchronizations between cores to resolve memory accesses in multicore systems with shared memory.

The multicore system is divided into many islands so that each island can operate at a different V/F level in order to achieve acceptable energy saving with lesser construction cost and runtime overhead [8]. Whether they are single-core or multiple-core, VFIs' V/F levels can be either statically established at design time or tweaked while the tasks are running. The static V/F level assignment is suited for applications whose workloads do not vary noticeably during run-time in addition to low-cost, low-overhead hardware design. Additionally, the benefits of the VFI partitions' energy-saving capabilities are not compromised by the off-chip or on-chip high package cost in system architectures with small-scale cores and VFIs [9]. At the expense of chip area and complexity, as well as potential overhead in execution energy and time usage caused by complex V/F controllers/regulators used for dynamically adjusting cores' V/F levels during the application run-time, the dynamically tuned VFIs compensate for differing application characteristics that are not handled by the static approach [10]. By figuring out the V/F levels of the tasks that are given to the cores for execution, the energy efficiency of multi-core systems can be further increased [11]. Task-core mapping and V/F level assignments are essential for resolving restricted optimization issues, regardless of whether the system VFIs are created at design time or VFI partitions are optimized per application.

This paper investigates the multi-core platform scheduling of a group of irregular DAG tasks with implicit deadlines. As far as we are aware, this is the first study that solves the problem of power usage when scheduling many DAG activities on multi-core [12], [13]. We assume that a DAG task always uses all of the cores assigned to it, which results in non-negligible power usage. We enable the removal or reduction of the number of lightly laden cores in order to mitigate this effect. Cores that are not needed can be entirely turned off after merging [14]. The cores won't be used if the average case execution times are normally low relative to the worst-case execution times (WCET).

The major contribution of this research work is:

- To design a new DAG Scheduling model for efficient scheduling of periodic tasks into multiple voltage frequency islands.
- To Implement the SI-POA algorithm to optimize the scheduling of the DAG on the voltage frequency islands.
- To Train a Recurrent Neural Network (RERNN) to predict the core temperature of each voltage frequency island based on the considered multi-objectives like execution time, makespan, and overall energy consumption.
- To optimize the scheduling of the DAG on the voltage frequency islands using SI-POA.
- The proposed Self-Improved Pelican Optimization Algorithm (SI-POA) is an extended version of the standard Pelican Optimization Algorithm (POA) model.

The chapters were arranged in the manner described below: The basic introduction is presented in Chapter I, the theoretical background of the literature review conducted for this research work is presented in Chapter II, an overview of the proposed methodology is presented in Chapter III; the proposed algorithm is used in Chapter IV, the same experiment's results are summarized in Chapter V, and the research work conclusions are presented in Chapter VI.

II. LITERATURE REVIEW

In this section, a few relevant works on the Scheduling of DAG on Voltage frequency islands are examined. Tariq et al. [15] have experimented using Voltage Frequency Island (VFI)-based heterogeneous NoC-MPSoCs with re-timing coupled with DVFS for real-time streaming applications. This paper develops an energy-aware scheduler in this paper that takes conditional constraints into account. This paper presented forward a brand-new task-level re-timing method called R-CTG and merged it with a voltage scaling method name ALI-EBAD that was based on non-linear programming. Without sacrificing energy efficiency, the R-CTG technique seeks to reduce the latency brought on by re-timing.

Wu et al. [16] have presented a path-relinking-enhanced estimation of the distribution method (EDA). To benefit from the understanding of previous research, an effective hybrid scheme, including list scheduling heuristics, was created. Additionally, a particular probability model was constructed to explain the relative position relationships between the task pairs, and the task processing permutations are created by sampling such a model. The local search approach was designed using path-relinking-based knowledge to maximize the utilization of EDA.

Tariq et al. [17] have experimented with real-time streaming application scheduling on edge devices while taking energy considerations into account. To begin with, a cutting-edge re-timing-based technique was created to convert the

dependent workload into an independent task model in order to free up resources and the processors' unused slack with a potential minimal preamble. A unique population-based method called ARSH-FATI was also introduced, which, in contrast to the known population-based optimization algorithms, can dynamically switch among exploitative and explorative search modes at run-time for optimization technique.

Guo et al. [18] have investigated energy-efficient real-time scheduling of sporadic parallel activities with bounded deadlines, with each task represented as a directed acyclic graph (DAG). This paper takes into account a clustered multi-core platform, where each processor in a cluster operates at the same speed at all times. In order to reduce the anticipated long-term energy consumption, a novel concept called speed profile was developed to simulate run-time fluctuations in energy consumption by task and by cluster. No existing work, as far as we're aware, takes energy-sensitive real-time scheduling of DAG activities with tight deadlines into account, much alone on a clustered multi-core platform.

Pournazarian et al. [19] have suggested that DGs with inverter interfaces, with the exception of PHEVs, implement an intelligent droop control for MGs voltage and frequency regulation. The proposed droop control can smoothly adjust the frequency and voltage of MG and be independent of the MG line's parameters. The voltage and frequency of MGs were then controlled by PHEVs using a revolutionary technique that was then put forth. In the V2G mode, this method was used on the PHEV parking lots. Compared to the other prior technique, this one has a stronger ability to maintain the stability of MG even in dire situations.

Safari et al. [20] has presented a comprehensive overview of the recent research initiatives that utilize fault-tolerance methods while taking into account timing, power/energy, and temperature from the design standpoint of real-time embedded systems. The job mapping/scheduling policies for fault-tolerance real-time embedded systems are examined and categorized in accordance with the aims and limitations they are thought to have. Additionally, the hardware models, application models, and fault-tolerance strategies used are considered additional dimensions of the classification provided.

Huang et al. [21] have experimented with the dynamic scheduling of activities described by directed acyclic graphs (DAGs), an NP-hard issue with only heuristic solutions, which was the subject of this study. There were two steps to our contributions: 1) Assuming that the allocation of DAG nodes to processors was known, this paper suggests optimal energy allocation (OEA) and search-based OEA (SOEA), the first optimal approaches that minimize energy usage while rewarding the reliability demand homogeneous and heterogeneous systems, respectively. 2) This paper introduces a scheduling algorithm, out-degree scheduling (ODS), which allocates the DAG nodes based on their out-degrees and takes energy consumption into consideration.

Hajiamini et al. [22] have experimented in order to reduce the task set (application) execution time (makespan) for a certain energy budget. A task scheduling and VFI partitioning problem were developed in this work. To generate per-core, per-task dynamic V/F levels in a fine-grain VFI-based system with single-core islands, the restricted optimization problem was first defined with integer linear programming (ILP). Next, mixed integer linear programming (MILP) was used to develop static task scheduling for coarse-grain VFI-based systems, where an island can contain many cores operating at the same V/F level while taking into account the energy budget and task set precedence limitations.

Roeder et al. [23] have experimented with reducing total energy usage; this paper provides an off-line scheduling approach based on forward list scheduling for dependent multi-version tasks. Our heuristic allows apps to dynamically adjust voltage and frequency while they are running since it takes into consideration Dynamic Voltage and Frequency Scaling (DVFS). The given examples of how multi-version task models and an energy-conscious scheduler might be advantageous. We note that using the most energy-efficient version for each activity does not result in the application using the least amount of energy overall.

Li et al. [24] have presented an energy/thermal aware work scheduling strategy by taking into account both thermal and energy considerations. The optimization is carried out from two angles: first, it balances the energy/thermal loads of processors by allocating tasks in an energy/thermal aware heuristic way and the workload of tasks by the deduced task-level deadlines; second, it reduces the amount of time among both parallel tasks that share the same successor task.

Guo et al. [18] have investigated energy-efficient real-time scheduling of sporadic parallel activities with bounded deadlines, with each task described as a directed acyclic graph (DAG). The author takes into account a clustered multi-core platform, where each processor in a cluster operates at the same speed at all times. In order to reduce the anticipated long-term energy consumption, a novel concept called speed-profile is developed to simulate run-time fluctuations in energy consumption by job and by cluster. No existing work, as far as we are aware, takes into account energy-conscious real-time scheduling of DAG jobs with tight deadlines, nor on a clustered multi-core platform.

A. MOTIVATION FOR RECENT RESEARCH WORK

Dynamic Voltage and Frequency Scaling is possible in most current systems (DVFS). There is a clock frequency for each activity, for each type of CPU or GPU, that minimizes energy consumption. Lower frequency results in longer duration, which raises static energy usage. While a higher frequency results in a shorter run time, the required higher voltage also increases the use of dynamic energy. The code affects this convex behaviour. With contemporary CPUs, only the core cluster may change the clock frequency (i.e. voltage island). Hence, based on the many tasks running on each

voltage island, we must choose the best frequency for that island. We utilize DVFS for various voltage islands to lower the overall energy consumption of an application made up of several tasks. Common scheduling challenges are now made more difficult by heterogeneous platforms, numerous versions, voltage islands, and DVFS. Schedulers must now choose which processing unit and how frequently to execute a task (version) in order to limit the amount of energy used overall.

III. SYSTEM DESCRIPTION

This paper concentrates on the scheduling of DAG tasks in the VFI model. The workflow for scheduling a DAG on voltage frequency islands to minimize the makespan, overall energy consumption, and reduce core temperature using the Self-Improved Pelican Optimization Algorithm (SI-POA) and Recurrent Neural Network (RERNN):

- **DAG modelling**: Model the Directed Acyclic Graph (DAG) that represents the workflow to be executed. This can be done using a graph representation where the nodes represent tasks, and the edges represent dependencies between tasks.
- **Voltage frequency island partitioning**: Divide the available processing units into multiple voltage frequency islands. This can be done based on the power consumption of each unit and the task requirements.
- **Core Temperature prediction using RERNN model**: Train a Recurrent Neural Network (RERNN) to predict the core temperature of each voltage frequency island based on the multi-objectives like execution time, makespan, overall energy consumption and core temperature.
- **DAG scheduling optimization with SI-POA**: Scheduling of the DAG on the voltage frequency islands is optimized using the Self-Improved Pelican Optimization Algorithm (SI-POA). The proposed SI-POA model is an extended version of the standard POA model. The SI-POA model is inspired by the behaviour by the natural behaviour of pelicans during hunting. In the scheduling optimization phase, the SI-POA algorithm optimizes the scheduling of the DAG on the voltage frequency islands while taking into account the predicted core temperature of each island based on the Recalling-enhanced recurrent neural network RERNN) model. The goal is to minimize the makespan and overall energy consumption of the DAG while keeping the core temperature of each island within safe limits.
- **Simulation and evaluation**: Simulate the optimized scheduling on a representative set of input DAGs and evaluate the performance of the algorithm in terms of makespan, energy consumption, and core temperature.

A. TASK DECOMPOSITION

Task decomposition is a well-known method for simplifying the scheduling analysis of concurrent real-time jobs. In our method, the first stage is task decomposition, which breaks

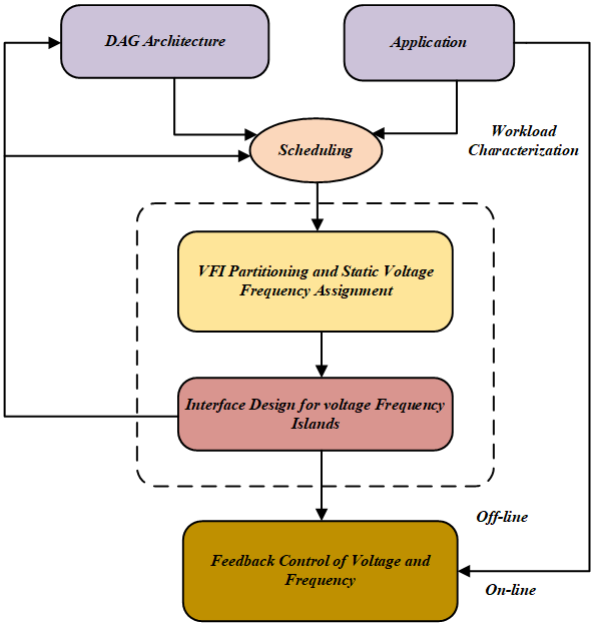

FIGURE 1. Scheduling for directed acyclic graph (DAG).

down each DAG task τ_i node N_i^l into a separate sub-task τ_i^l with a release offset b_i^l, deadline f_i^l, and execution requirement c_i^l. All dependencies (expressed by DAG edges) are honoured when setting release dates and deadlines. Decomposition assures that the DAG can be scheduled if all of the subtasks are schedule-able. In order to be thorough, we provide a brief explanation of job decomposition in this subsection, along with an illustration. The strategy employed by Saifullah et al. [25] is changed significantly by us. Initially, we carried out the assignment decomposition using the methods outlined below. We create a vertical line at each time instant when a node starts or ends for every node starting from the beginning, assuming the work is executed on an infinite number of cores. The DAG is divided into segments by these vertical lines, and each segment contains an equal number of nodes that execute. Now, portions of various nodes within a segment can be thought of as parallel execution threads and threads inside a segment can begin only when threads within the preceding segment have completed their executions. Now, we'll claim that the task's segmented structure has been transformed into a synchronized form and mark it with the symbol τ_i^{syn}. A node's allocated time is calculated by first allocating time to each segment and then adding the times assigned to each segment individually.

There may be a slack in which all processors are idle since the minimal makespan, $L_i \leq T_i$, occurs at the end of each period (which is typically energy inefficient). By increasing each section by a common link for task $\frac{T_i}{L_i}$, we distribute such idle time uniformly. The scheduling window (b_i^l, f_i^l) on top of the processor assignment M_i^l (i.e., a node-to-processor mapping) provided by task decomposition is where each node N_i^l of a task τ_i resolve be scheduled.

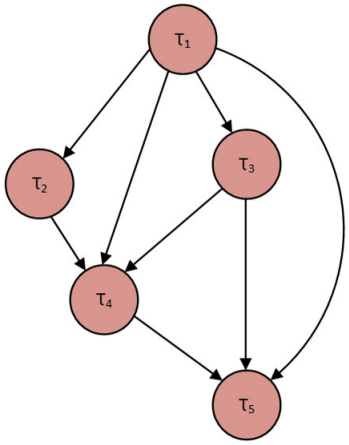

FIGURE 2. DAG application model.

B. DAG APPLICATION MODEL

When N is a collection of nodes, and E is a collection of edges, $G = (N, E)$ is used to represent a DAG task model. Each directed edge $e_{i,j} \in E$ provides an executive order such that the sub-task τ_j can only begin once the sub-task τ_i is finished. Each node $\tau_i \in N$ signifies a sub-task of the DAG. In accordance with this, it is said that τ_i is τ_j's immediate predecessor and that τ_j is τ_i's, immediate successor. We refer to the group of τ_i's immediate predecessors and successors, respectively, as $pre(\tau_i)$ and $succ(\tau_i)$. The communication expense between the sub-tasks τ_i and τ_j is also represented by a weight $w_{i,j}$ that is added to each edge $e_{i,j}$. It is usual practice to ignore the communication cost between two sub-tasks when they are assigned to the same processor.

Figure 2 depicts a sample DAG task with five sub-tasks $\tau_1, \tau_2, \tau_3, \tau_4, \tau_5$, where tasks $\tau_2, \tau_3, \tau_4, \tau_5$ can only begin until task τ_1 has finished and are carried out concurrently. The computation matrix for this example on a three-processor system also provides the highest frequency f_{max} at which each sub-task was completed on each processor. Different processors on a hybrid platform offer various processing capabilities. In the following sections on the efficiency and reliability model, the processor parameters are explained. It has been common practice in the literature to employ the DAG job in Figure 2 with the calculation matrix and processor parameters. It is also mentioned in the later sections of this work and used as an example to illustrate a point.

C. ENERGY MODEL

Processor power dissipation is mostly made up of static consumption, frequency-dependent dynamic consumption, and frequency-independent dynamic consumption. The primary one of them is frequency-dependent dynamic power consumption, which is expressed by

$$P = \xi C V^2 F, \tag{1}$$

where C is denoted as the loading capacitance, V is denoted as the supply voltage, and F is denoted as the clock frequency, ξ is an activity factor. Considering that F, we obtain $P \propto CF^{\alpha}$, where α is roughly 3. In order to facilitate discussion, we model a processor's frequency-dependent dynamic power consumption is denoted as CF^{α} and its frequency-independent dynamic and static power consumption is denoted as P^*. Afterwards, a processor's overall power use is denoted as u_k

$$P = P^* + CF^{\alpha} \tag{2}$$

We normalize a processor's frequency so that $F_{\max} = 1$ for ease. When the frequency range is F_i ($F_i \leq F_{\max}$), and T_{τ_i} is the efficient implementation of the sub-task τ_i on a processor operating at its maximum frequency $F_{\max}$, the implementation time is calculated as follows:

$$t_{\tau i} = T_{\tau i} \times \frac{Fmax}{Fi} = T_{\tau i} \times \frac{1}{Fi}. \tag{3}$$

The sum of the processor's power consumption and the execution time gives the amount of energy needed to accomplish τ_i.

$$E_{\tau i}(F_i) = P \times t_{\tau i} = \left(P^* + CF_i^{\alpha}\right) \times T\tau iF \tag{4}$$

The sum of all sub-tasks equals the energy consumption of a DAG task.

$$E_G(F) = E_{\tau 1}(F_1) + E_{\tau 2}(F_2) + \cdots + E_{\tau n}(F_n), \tag{5}$$

where $F = (F_1, F_2, \ldots, F_n)$ denoted as a vector.

D. RELIABILITY MODEL

The probability that a task is executed without error is meant by task reliability. Like many previous studies, ours focuses on the prevalent transient faults that are related to processing frequency and may be modelled by the exponential distribution given below.

$$\lambda(F) = \lambda_f \times 10^{\frac{D(1-F)}{1-Fmin}} \tag{6}$$

where D is denoted as a hardware-related constant, $F_{\min}$ is denoted as the lowest frequency that can be used, and λf is denoted as the average number of errors per second at the highest frequency.

The potential for errors increases as task execution time increases. The reliability of a sub-task τ_i that is performed on a processor with frequency F_i can be computed using equation

$$R_{\tau_i}(Fi) = e^{-\lambda(Fi) \times \frac{T\tau i}{Fi}} \tag{7}$$

The successful, error-free execution of each sub-task is a need for a dependable DAG task. Consequently, a DAG task's reliability $R(G)$ is equal to the sum of all its sub-tasks.

$$R_G(F) = R_{\tau_1}(F_1) \times R_{\tau_2}(F_2) \times \cdots \times R_{\tau_n}(F_n). \tag{8}$$

E. THERMAL MODEL

A thermal model is a mathematical model that represents the temperature distribution in a system or component as a function of time and space. Thermal models are used to predict the response of a system or component to heat input and to determine the temperature distribution under various operating conditions. They can be used to design heat exchangers, predict the temperature distribution in electronic components, and optimize the performance of thermal systems. Thermal models can be based on analytical or numerical methods and may include assumptions and approximations depending on the complexity of the system being modelled.

The thermal resistance among two processors, P_k and P_m, is denoted as $R_{k,m}$. $R_{k,m} = \infty$ if there is no heat flow between P_k and P_m. R_k is the portion of P_k's thermal resistance that allows heat to escape to the surrounding air. Let $\theta_k(t)$ and $P_k(t)$ represent the temperature and power usage of P_k at time t, respectively. The heat transmission process of P_k can therefore be explained as follows using Fourier's law:

$$\frac{d\theta_k(t)}{dt} = \frac{P_k(t)}{C_k} - \frac{\theta_k(t) - \theta_a}{R_k C_k} - \sum_{P_m \in P} \frac{\theta_k(t) - \theta_m(t)}{R_{km} C_k} \tag{9}$$

There won't be any heat transfer between CPUs when their thermal demands are balanced. Consequently, the thermal model given by Eq. (9) can be changed into:

$$\frac{d\theta_k(t)}{dt} = \frac{P_k(t)}{C_k} - \frac{\theta_k(t) - \theta_a}{R_k C_k} \tag{10}$$

Moreover, the temperature function can be determined as follows by mathematical translation and deduction:

$$\theta_k(t) = C_k \left(\theta_k(t) - \frac{R_k \beta_k + \theta_a}{1 - R_k \alpha_k} \right) \tag{11}$$

$$\theta_k(t) = \int_0^t P_d^k(u) e^{\lambda_k (u-t)} du + \theta_k(0) e^{-\lambda_k t} \tag{12}$$

$$\lambda_k = \frac{1}{R_k C_k} - \frac{a_k}{C_k} \tag{13}$$

F. TASK SCHEDULING AND VFI PARTITIONING

The Pareto frontier for the makespan-energy tradeoff is provided by the VFI partitions, which assign the best V/F levels per core. The hardware and space overhead of voltage regulators, which increase chip design complexity in light of current technological scaling, counteract the energy savings benefit of such VFI partitioning. As a result, this section offers a method with less design complexity at the expense of a reasonable makespan-energy budget tradeoff.

The energy and execution times for the subtasks in the first scheme were determined by completing the ILP. Furthermore, it is believed that all cores with allocated subtasks share the same V/F level and are clustered in a single VFI after the task scheduling and VFI partitioning problems have been solved. As a result, M ($M < N$)VFIs exist, each with a different V/F level. The definition of this issue is as follows: Assign and schedule subtasks to cores so that the makespan is minimized, keeping in mind the energy consumption determined by the ILP and the subtasks' dependencies. Given the V/F levels, related energy consumptions, and execution timings of the subtasks. According to the aforementioned definition, subtasks' start times and core assignments are variable; hence this issue is defined using mixed integer linear programming (MILP):

$$S_{i,j} + t_{i,j} \leq MS \quad \forall \tau_{i,j} \in T \tag{14}$$

MS is denoted as the makespan, $S_{i,j}$ is denoted as the sub-task start time. According to Equation (14), the application's makespan is determined by which sub-task completes execution on which core last.

$$\sum_{c=1}^{N} X_{(i,j),c} = 1 \quad \forall \tau_{i,j} \in T \tag{15}$$

$X_{(i,j)}, c$ returns an integer number, and $X_{(i,j)}, c = 1$, if sub-task $\tau_{i,j}$ is assigned to core c. Equation (15) makes sure that each core is given just one sub-task.

$$X_{(i,j),c} + X_{(k,l),c} \leq 1 \quad \forall \tau_{i,j}, \tau_{k,l} \in T, \quad \forall c \in CA_{(i,j),(k,l)} = 0 \tag{16}$$

Two sub-tasks with differing V/F levels cannot be allocated to the same core, according to equation (16).

$$S_{i,j} + t_{i,j} \leq S_{k,l} + \left(2 - X_{(i,j),c} - X_{(k,l),c}\right) \cdot M + \left(1 - Y_{(i,j),(k,l)}\right) \cdot M \quad \forall \tau_{i,j}, \tau_{k,l} \in T, \quad \forall c \in C, \; M \in Z+, \; A_{(i,j),(k,l)} = 1 \tag{17}$$

$Y_{(i,j),(k,l)}$ is an integer, and $Y_{(i,j),(k,l)}$' $= 1$ if sub-task $\tau_{i,j}$runs before subtask $\tau_{k,l}$ when they are both allocated to the same core. $A_{(i,j),(k,l)}$ is integer number, $A_{(i,j),(k,l)} = 1$ if sub-task $\tau_{i,j}$ and $\tau_{k,l}$ have the same V/F level. Two sub-tasks addressed to a single core cannot overlap, according to Equation (17). Additionally, this constraint guarantees that the V/F level of any two sub-tasks issued to the same core is the same.

$$Y_{(i,j),(k,l)} + Y_{(k,l),(i,j)} = 1 \quad \forall \tau_{i,j}, \forall_{k,l} \in T \tag{18}$$

According to equation (18), only one sub-task in a pair of sub-tasks comes before the other. The sub-task precedence links are held by equation (18).

$$S_{i,j} + t_{i,j} \leq S_{i,l} \quad \forall B \tau_{i,j}, \; \tau_{i,l} \in T, \; B_{(i,j),(i,l)} = 1 \tag{19}$$

$B_{(i,j),(i,l)}$ is integer number, $B_{(i,j),(i,l)} = 1$ if sub-task $\tau_{k,l}$ is dependent on sub-task $\tau_{i,j}$, sub-task execution time is denoted as t, energy consumption is denoted as E. If they are scheduled to the same core or VFI, the sub-task pair for the same task τ_i in case (19) must maintain dependency.

$$S_{i,j}, S_{k,l}, S_{i,l} \in R + \forall \tau_{i,j}, \tau_{k,l}, \tau_{i,l} \in T \tag{20}$$

$$X_{(i,j),c}, X_{(k,l),c}, Y_{(i,j),(k,l)} = \{0, 1\}, \quad \forall \tau_{i,j}, \tau_{k,l} \in T \tag{21}$$

According to equations (20) and (21), the start time and allocation variables for the sub-tasks are, respectively, real values and 0/1 integers.

G. VFI PARTITIONING

Voltage-frequency Island (VFI) partitioning is the process of dividing a VFI into multiple smaller sub-islands, each with its own voltage and frequency settings. This allows for more fine-grained control over the power consumption and performance of the processors within the VFI. VFI partitioning can be done in several ways, such as manual partitioning, where an engineer manually divides the VFI into sub-islands, or automatic partitioning, where an algorithm is used to divide the VFI into sub-islands based on certain criteria. One common approach for automatic VFI partitioning is to use an algorithm that minimizes the total energy consumption of the VFI while still meeting performance constraints, such as deadlines for tasks in a directed acyclic graph (DAG). This can be done by iteratively generating and evaluating different partitioning candidates and selecting the best one based on energy consumption and performance metrics.

The manycore system is divided into a predetermined number of VFIs using the nonlinear programming formulation shown below. This nonlinear problem formulation was initially an expensive computational integer programming problem. The values of the core-VFI mappings decision variables are relaxed to range between zero and one in order to lessen the complexity of the problem. Using a penalty function in the problem's objective, the solutions (core-VFI mappings) eventually converge to 0/1 integers.

$$\sum_{j=1}^{P}\sum_{i=1}^{N}\sum_{k=1}^{M}\left(\frac{\tau_{k,j}-t_{i,j}}{\tau_{k,j}}\right)\cdot X_{i,k} + \lambda\cdot\sum_{i=1}^{N}\sum_{k=1}^{M}x_{i,k}\cdot\left(1-X_{i,k}\right)\lambda\in R^{+} \tag{22}$$

$$t_{i,j}\cdot X_{i,k}\leq\tau_{k,j}1\leq j\leq P,\quad \forall c_i\in C,\ \forall i_k\in I \tag{23}$$

$$\sum_{i=1}^{N}X_{i,k}\geq\theta\quad\forall i_k\in I \tag{24}$$

$$\sum_{i=1}^{N}X_{i,k}\geq\theta\quad\forall ik\in I \tag{25}$$

$$X_{1,1}=1 \tag{26}$$

$$X_{j,k}\in[0,1]\quad\forall c_i\in C,\ \forall i_k\in I \tag{27}$$

where $\tau_{k,j}$ is the maximum execution time of cores that are located in a VFI i_k during the execution phase j, and $t_{i,j}$is the execution time of core $c_i$in execution phase j. The core c_i is placed in the VFI $i_k$is determined by $X_{i,k}$.θ is denoted as the minimum number of cores required for each VFI i_k. λ is denoted as a coefficient that penalizes the goal (Equation (22)) for giving the variables $x_{i,k}$non-binary values. The solutions to the issue ($X_{i,k}$) approach binary for bigger λs, avoiding the huge values of the second component in (22), which endangers the goal of our optimization problem.

The optimization objective (22) is to create the VFIs with cores whose run times are as close to their maximum execution times over the benchmark's execution phases as possible. The optimization seeks to reduce the percentage of time that a core spends idle after the execution phase is complete in comparison to the run time of the slowest core with identical computation time when these cores are in the same VFI, as demonstrated by the initial term of the objective. This fraction (the percentage of the core that is idle) serves as a weight when allocating a core to a VFI that also has cores with similar percentages of idle time during the application's execution phases. The solution to our optimization challenge involves grouping together the cores with the most comparable weights (idle percentages). However, because this article examines homogeneous multi-core systems/VFIs, each core's definition of the weighting technique is the same. The maximum execution time of the cores in the VFI is determined by constraint (23). In order to prevent exceptionally uneven VFI sizes that raise the benchmark's execution time or system energy consumption, constraint (24) mandates that each VFI have a minimal number of cores. The core c_i is contained in a single VFI i_k by constraint (25). During the optimization process, constraint (26) makes sure that the order of the VFI numbers remains the same for every experimented VFI partitioning solution. In constraint (27), $x_{i,k}$ ranges from 0 to 1. It makes sense that core c_i would be in VFI i_k as $x_{i,k}$ gets closer to one.

IV. PROPOSED HYBRID ALGORITHM

A. DAG SCHEDULING USING SI-POA

the scheduling of the DAG on the voltage frequency islands is optimized using the Self-Improved Pelican Optimization Algorithm (SI-POA). The proposed SI-POA model is an extended version of the standard POA model. In the scheduling optimization phase, the SI-POA algorithm optimizes the scheduling of the DAG on the voltage frequency islands while taking into account the predicted core temperature of each island based on the Recalling-enhanced recurrent neural network (RERNN) model.

B. PELICAN OPTIMIZATION ALGORITHM (POA)

Pelican Optimization Method (POA), a new stochastic algorithm that was organically inspired by the optimization algorithm, is presented. The primary inspiration for the POA's design was pelicans' typical hunting behaviour. This species thrives in social settings and flocks of several hundred pelicans [26].

Step 1: Initialization The first step of POA is the initialization of parameters; here, the input parameters are execution time (makespan), energy consumption, reliability and core temperature.

Step 2: Random Generation In this step of random generation, the input variables are generated at random.

$$Y=\begin{bmatrix}(S,T)^{11} & \cdots & \cdots & (S,T)^{1N}\\ (S,T)^{21} & \cdots & \cdots & (S,T)^{2N}\\ \vdots & \vdots & \vdots & \vdots\\ (S,T)^{m1} & \cdots & \cdots & (S,T)^{MN}\end{bmatrix} \tag{28}$$

Step 3: Fitness evaluation Based on the current best position, the parameters are initialized.

$$F = Min\,(T + M + E) \tag{29}$$

Step 4: Moving towards Prey (Exploration Phase)
Equation (30) is a mathematical formulation of the pelican's approach to the location of its prey,

$$X_{I,J}^{p1} = \begin{cases} X_{I,J} + rand \cdot \left(P_J - i \cdot X_{I,J}\right), & f_P < f_I; \\ X_{I,J} + rand \cdot \left(X_{I,J} - P_J\right), & Else, \end{cases} \tag{30}$$

Here, I denotes a random number which is equal to one or two, f_P denotes objective function value, location of prey in thej^{th} dimension is indicated as P_J, the new status of i^{th} pelican in j^{th} dimension based on phase 1 is denoted as $X_{I,J}^{p1}$ and it is resulting in equation (4),

$$x_I = \begin{cases} x_I^{p_1}, & f_I^{p_1} < f_I; \\ x_I, & Else, \end{cases} \tag{31}$$

Here, $f_I^{p_1}$ is represented as an objective function value based on phase 1 and the new status of the i^{th} pelican is represented as $x_I^{p_1}$. The flowchart of POA is shown in Figure 3.

Step 5: Choatic Map Based Water surface with wings (Exploitation Phase) Equation (32) is produced as a result of the pelican's hunting behaviour,

$$X_{I,J}^{p_2} = X_{I,J} + r \cdot \left(1 - \frac{T}{t}\right) \cdot (2 \cdot Rand - 1) \cdot X_{I,J} \tag{32}$$

Here, r is represented as constant, T is represented as the maximum number of iterations, $X_{I,J}^{p1}$ is represented as the new status of i^{th} pelican in j^{th} dimension based on phase 2 and which is modelled in Equation (33).

$$x_I = \begin{cases} x_I^{p_2}, & f_I^{p\,2} < f_I; \\ x_I, & Else, \end{cases} \tag{33}$$

Here, the new status of the i^{th}pelican is symbolized as $x_I^{p\,2}$ and the objective function value based on phase 2 is symbolized as$f_I^{p_2}$.

Step 6: Termination
Verify the termination criteria, and if the desired result is achieved, the procedure is concluded; otherwise, move on to step 3. The output of the algorithm is expressed as

$$\begin{bmatrix} e_f(t)^{11} & e_f(t)^{12} & \cdots & e_f(t)^{1n} \\ e_f(t)^{21} & e_f(t)^{22} & \cdots & e_f(t)^{2n} \\ \vdots & \vdots & \vdots & \vdots \\ e_f(t)^{m1} & e_f(t)^{m2} & \cdots & e_f(t)^{mn} \end{bmatrix} = \begin{bmatrix} (T,S)^{11} & (T,S)^{12} & \cdots & (T,S)^{1n} \\ (T,S)^{21} & (T,S)^{22} & \cdots & (T,S)^{21} \\ \vdots & \vdots & \vdots & \vdots \\ (T,S)^{m1} & (T,S)^{m2} & \cdots & (T,S)^{mn} \end{bmatrix} \tag{34}$$

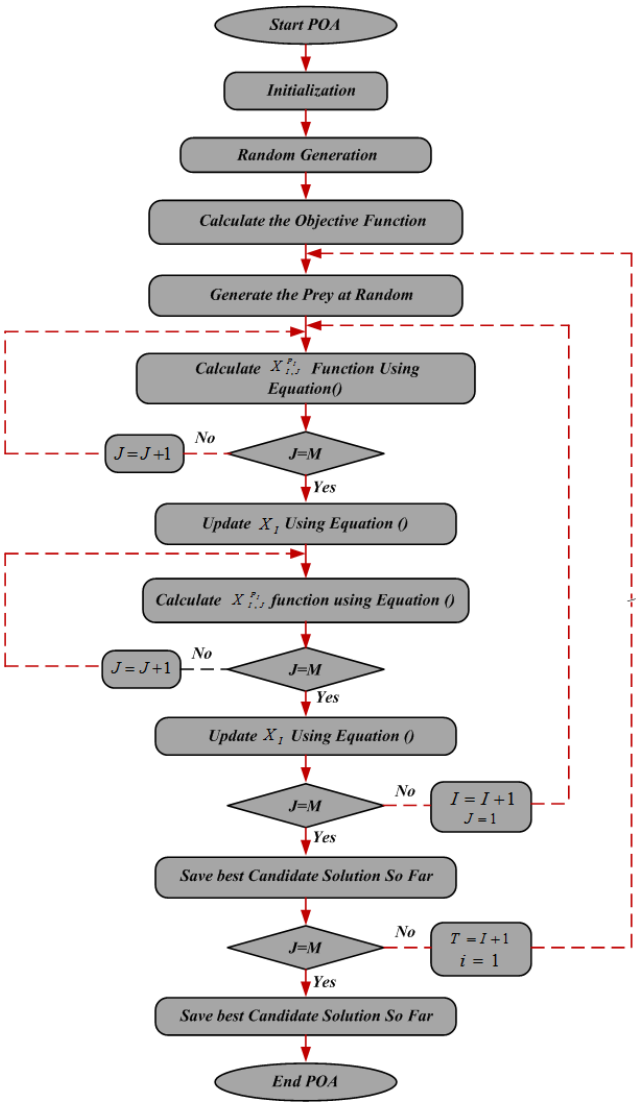

FIGURE 3. Flowchart of SI-POA.

C. RECALLING ENHANCED RECURRENT NEURAL NETWORK (RERNN)

RERNN is one of the artificial neural networks that use the radial function in the area of mathematical modelling. The Elman recurrent neural network has three layers compared to the six layers of the RERNN, which has selective memory. The input layer, state layer, memory layer, sum layer, hidden, delay, and output layer are the levels of RERNN [27]. The input node receives the system's input and also takes in the output of the hidden layer using a delay function. The memory layer can accommodate both the most recent state layer result and the preceding sum layer result. The primary role of the memory layer is to determine the size of the previous sum layer information for the following stage. The sum layer offers the ability to sum the final recurrent hidden outcome, the memory layer's result, and the current input. The hidden layer determines the output layer's ultimate probabilistic value. The delay layer propagates back the output of the currently active hidden layer.

Step 1. Initialization, We have given the output of the Pelican Search Optimization (PSO) equation (34) to the input of the Recalling Enhanced Recurrent Neural Network Algorithm (RERNN)

Step 2. Random Generation, The random vector generates random input parameters.

Step 3: Check the iteration, Procedure the data if the number of iterations is fewer than the maximum allowed; otherwise, halt the process.

Step 4. Output calculation, The error value is determined by,

$$E_k\,(n) = y_k\,(n) - \hat{y}k\,(n)\,, \quad k = 1.2, \ldots\ldots L \tag{35}$$

where, $\hat{y}k\,(n)$ is denoted as the desired output vector of its neuron in output, $y_k\,(n)$ is denoted as output vector of ith neuron in the output layer

Step 5: Find learning rate by generalized Armijo search approach, The generalized Armijo search strategy uses the following criteria, for example, to determine the learning rate,

$$e\left(m^k + L_R p^k\right) \leq e\left(m^k\right) + \alpha_1 L_R e_w^k \left(p^k\right)^t, \quad \alpha_1 \geq 0 \tag{36}$$

Step 6: Calculate the new weight The gradient descent algorithm is used to determine the new weight and is explained as,

$$m^{k+1} = m^k + L_R P^k \tag{37}$$

Step 7: Check the maximum iteration, Stop the process once the iteration is complete. If not, extend the iteration and move on to step 5.

Step 8: Calculate the direction, The direction of the learning process is estimated via the conjugate gradient descent algorithm.

$$P^k = -E_w^k + \beta P^{k-1} \tag{38}$$

$$\beta = \frac{\alpha E_w^k \left(P^{k-1}\right)^t}{P^{k-1}\left(P^{k-1} - E_w^k\right)}, \quad \alpha \in (0, 1) \tag{39}$$

Step 9: Error calculation, The total error is calculated using

$$E_T = \sum_{n=1}^{t} \sum_{k=1}^{n} ynk - \hat{y}nk \tag{40}$$

Step 10: Termination Verify the stopping requirements. **Step 2** should be taken if the stopping requirements have not been met after the maximum number of iterations.

V. RESULT AND DISCUSSION

The proposed model has been implemented in Python. Among the collected data, 80% of the data has been for training purposes, and the rest 20% of the data has been used for testing purposes. The graphical analysis and statically analysis are compared with various existing techniques such as Pelican Optimization Algorithm (POA), Crow Search Optimization (CSO), and Slap Swarm Optimization (SSO).

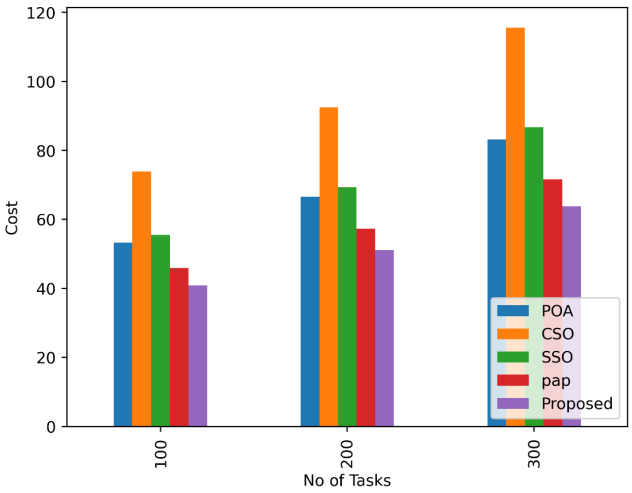

FIGURE 4. Assessment of cost employing various methods.

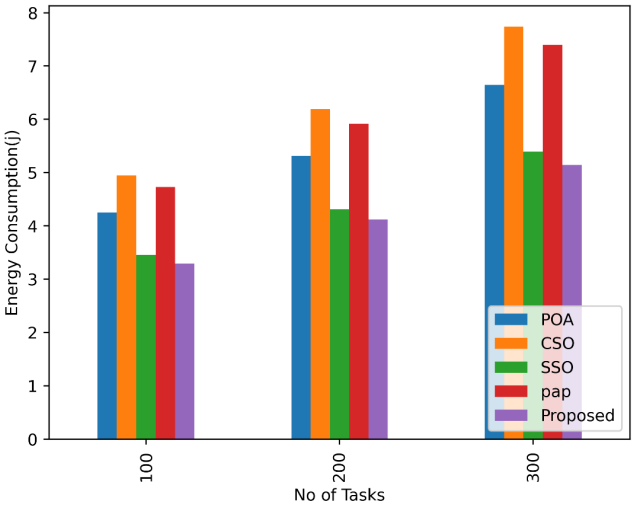

FIGURE 5. Valuation of energy consumption employing various methods.

A. PERFORMANCE ANALYSIS: PROPOSED VS EXISTING METHOD

Assessment of cost employing various methods is shown in Figure 4. For task *count* = 100, the cost value recorded by the proposed work is 40%, which is better than *POA* = 54%, *CSO* = 78%, *SSO* = 57% and *pap* = 43%. For task *count* = 200, the cost value recorded by the proposed work is 45%, which is better than *POA* = 62%, *CSO* = 84%, *SSO* = 64% and *pap* = 51%. For task *count* = 300, the cost value recorded by the proposed work is 81.5%, which is better than *POA* = 80%, *CSO* = 115, *SSO* = 82% and *pap* = 64% the values are obtained for task 100. Compared to the existing method, the outcome features of the proposed are low in cost.

Figure 5 displayed the valuation of energy consumption employing various methods. For task count = 100, the proposed work's energy consumption value is 32%, better than POA's 41%, CSO's 50%, SSO's 48%, and pap's 48%. For task

TABLE 1. Statistical analysis of performance metrics for task 100.

	POA	CSO	SSO	pap	Proposed
Execution Time(ms)	1139.45	1372.547	1418.37	1210.46	1071.038
Response Time(ms)	0.044634	0.041181	0.039462	0.053922	0.036492
Energy Consumption(j)	4.253611	4.953611	3.452827	4.734282	3.292994
Cost	53.29372	73.9909	55.51908	45.85616	40.85341
Make span(ms)	1747.152	1830.206	1670.361	2299.109	1658.93

TABLE 2. Statistical analysis of performance metrics for task 200.

	POA	CSO	SSO	pap	Proposed
Execution Time(ms)	1424.312	1715.684	1772.962	1513.075	1338.798
Response Time(ms)	0.055792	0.051476	0.049328	0.067403	0.045615
Energy Consumption(j)	5.317014	6.192014	4.316033	5.917852	4.116242
Cost	66.61715	92.48862	69.39885	57.3202	51.06676
Make span(ms)	2183.94	2287.757	2087.951	2873.887	2073.662

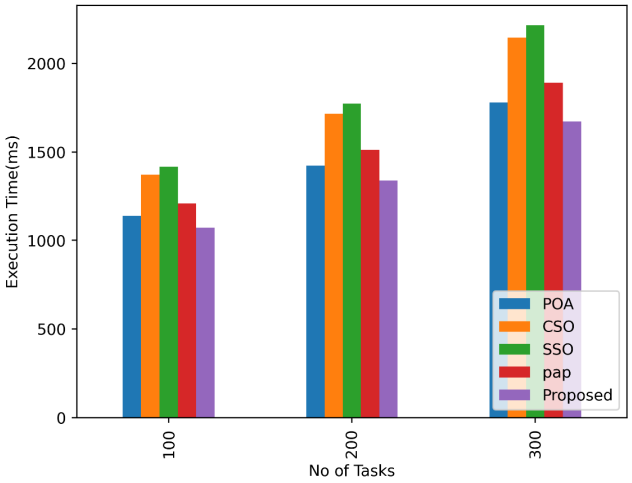

FIGURE 6. Estimate of execution time using various methods.

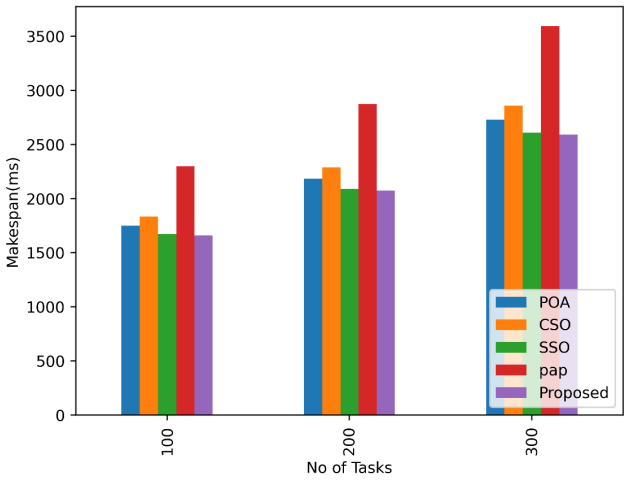

FIGURE 7. Evaluation of makespan using various methods.

count = 200, the proposed work's energy consumption value is 39%, better than POA's 51%, CSO's 61%, SSO's 40%, and pap's 58%. For task count = 300, the proposed work's energy consumption value is 51%, better than POA's 62%, CSO's 76%, SSO's 52%, and pap's 49%. The outcome feature of the proposed is low compared to the existing techniques in energy consumption.

Figure 6 demonstrates the estimate of execution time using various methods. The proposed work's execution time value for a task count of 100 is 1001, which is higher than the values for POA (1003), CSO (1450) SSO (1470), and pap (1200). The proposed work's execution time value for a task count of 200 is 1300, which is higher than the values for POA (1320), CSO (1600) SSO (1670), and pap (1490). The proposed work's execution time value for a task count of 300 is 1550, which is higher than the values for POA (1600), CSO (2001) SSO (2020), and pap (1950). Compared to the proposed technique, the outcome of the existing one is high in execution time.

Evaluation of makespan using various methods is illustrated in figure 7. In comparison to POA's 1700, CSO's 1800, SSO 1550, and pap's 2300, the proposed work's energy consumption value for task count = 100 is 1560. In comparison to POA's 2010, CSO's 2030, SSO 2000, and pap's 2800, the proposed work's energy consumption value for task count = 200 is 2000. In comparison to POA's 2550, CSO's 2600, SSO 2500, and pap's 3500, the proposed work's energy consumption value for task count = 300 is 2540. The outcome of the existing method is better in terms of makespan compared to the proposed method.

Analysis of response time using various methods is demonstrated in Figure 8. It is better than POA = 43%, CSO = 40%, SSO = 38%, and pap = 52% for task count = 100 since the proposed work's response time value is recorded as 45%. It is better than POA = 53%, CSO = 50%, SSO = 48%, and pap = 65% for task count = 100 since the proposed work's response time value is recorded as 43%. It is better than POA = 68%, CSO = 62%, SSO = 60%, and pap = 81% for task count = 300 since the proposed work's response time value is recorded as 55%. Compared to the proposed technique, the outcome of the proposed is low in response time.

TABLE 3. Statistical analysis of performance metrics for task 300.

	POA	CSO	SSO	pap	Proposed
Execution Time(ms)	1780.39	2144.605	2216.203	1891.344	1673.497
Response Time(ms)	0.06974	0.064345	0.06166	0.084253	0.057018
Energy Consumption(j)	6.646267	7.740017	5.395042	7.397315	5.145303
Cost	83.27143	115.6108	86.74856	71.65025	63.83345
Make span(ms)	2729.925	2859.696	2609.938	3592.358	2592.077

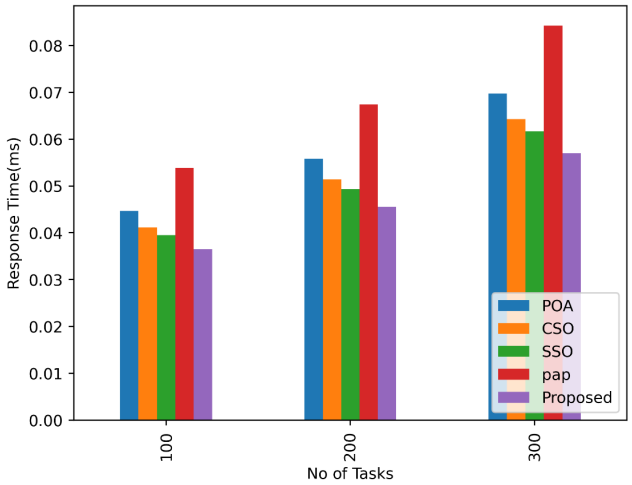

FIGURE 8. Analysis of response time using various methods.

B. OVERALL PERFORMANCE OF PROPOSED MODEL

Tables 1 to 3 demonstrate the Statistical Analysis of performance metrics for tasks 100, 200 and 300.

VI. CONCLUSION

This paper proposed the Scheduling of DAG on a Voltage frequency island based on the POA- RERNN approach. The proposed SI-POA approach is used to improve the speed of the convergence and the task scheduling of DAG on the VFI partitioning problem whose optimization goal is to minimize execution time (makespan), overall energy consumption, reliability and core temperature for a given energy budget. The SI-POA-RERNN approach solves the optimization problem. To achieve this, there are several approaches that can be taken. One approach is to use a POA algorithm that prioritizes tasks with higher energy requirements, as these tasks may be more likely to cause temperature spikes and can therefore benefit most from being scheduled on a VFI. Another approach is to use the RERNN algorithm that considers the dependencies between tasks, as this can help to minimize makespan and improve overall system efficiency. Additionally, using POA-RERNN algorithms that take into account the thermal properties of the system can help to minimize temperature increases and improve the overall performance of the system. In this work, we decompose the problem into two stages, as in one stage, the optimal solution can be computed. It is possible to solve the problem as a whole, which could lead to better results on one hand and makes it very challenging to devise effective approaches that can achieve these better results on the other hand due to the complexity. This is a promising future research direction to pursue.

REFERENCES

[1] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, p. 9.
[2] K. Li, "Energy efficient scheduling of parallel tasks on multiprocessor computers," *J. Supercomput.*, vol. 60, no. 2, pp. 223–247, May 2012.
[3] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela, "The global EDF scheduling of systems of conditional sporadic DAG tasks," in *Proc. 27th Euromicro Conf. Real-Time Syst.*, Jul. 2015, pp. 222–231.
[4] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *Proc. IEEE 33rd Real-Time Syst. Symp.*, Dec. 2012, pp. 63–72.
[5] C. Bazgan, B. Escoffier, and V. T. Paschos, "Poly-APX- and PTAS-completeness in standard and differential approximation," in *Proc. Int. Symp. Algorithms Comput.*, in Lecture Notes in Computer Science, vol. 3341, 2004, pp. 124–136.
[6] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing CPU energy in real-time systems with discrete speed management," *ACM Trans. Embedded Comput. Syst.*, vol. 8, no. 4, pp. 1–23, Jul. 2009.
[7] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *Proc. 25th Euromicro Conf. Real-Time Syst.*, Jul. 2013, pp. 225–233.
[8] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[9] L. Yuan, P. Jia, and Y. Yang, "Efficient scheduling of DAG tasks on multi-core processor based parallel systems," in *Proc. IEEE Region 10 Conf. (TENCON)*, Nov. 2015, pp. 1–6.
[10] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 3s, pp. 1–21, Mar. 2014.
[11] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–12.
[12] V. Devadas and H. Aydin, "Coordinated power management of periodic real-time tasks on chip multiprocessors," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 61–72.
[13] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Proc. 15th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2009, pp. 131–140.
[14] Y. Guo, D. Zhu, and H. Aydin, "Reliability-aware power management for parallel real-time applications with precedence constraints," in *Proc. Int. Green Comput. Conf. Workshops*, Jul. 2011, pp. 1–8.
[15] U. U. Tariq, H. Ali, L. Liu, J. Hardy, M. Kazim, and W. Ahmed, "Energy-aware scheduling of streaming applications on edge-devices in IoT-based healthcare," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 2, pp. 803–815, Jun. 2021.
[16] C.-G. Wu, L. Wang, and J.-J. Wang, "A path relinking enhanced estimation of distribution algorithm for direct acyclic graph task scheduling problem," *Knowl.-Based Syst.*, vol. 228, Sep. 2021, Art. no. 107255.
[17] U. U. Tariq, H. Ali, L. Liu, J. Panneerselvam, and J. Hardy, "Energy-efficient scheduling of streaming applications in VFI-NoC-HMPSoC based edge devices," *J. Ambient Intell. Hum. Comput.*, vol. 12, no. 11, pp. 9991–10007, Nov. 2021.
[18] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 156–168.

[19] B. Pournazarian, P. Karimyan, G. B. Gharehpetian, M. Abedi, and E. Pouresmaeil, "Smart participation of PHEVs in controlling voltage and frequency of island microgrids," *Int. J. Electr. Power Energy Syst.*, vol. 110, pp. 510–522, Sep. 2019.
[20] S. Safari, M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, and J. Henkel, "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues," *IEEE Access*, vol. 10, pp. 12229–12251, 2022.
[21] J. Huang, R. Li, X. Jiao, Y. Jiang, and W. Chang, "Dynamic DAG scheduling on multiprocessor systems: Reliability, energy, and makespan," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3336–3347, Nov. 2020.
[22] S. Hajiamini, B. Shirazi, H. Dong, and C. Cain, "Optimality of dynamic voltage/frequency scaling in many-core systems with voltage-frequency islands," *Sustain. Comput., Informat. Syst.*, vol. 24, Dec. 2019, Art. no. 100344.
[23] J. Roeder, B. Rouxel, S. Altmeyer, and C. Grelck, "Energy-aware scheduling of multi-version tasks on heterogeneous real-time systems," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, Mar. 2021, pp. 501–510.
[24] T. Li, T. Zhang, G. Yu, J. Song, and J. Fan, "Minimizing temperature and energy of real-time applications with precedence constraints on heterogeneous MPSoC systems," *J. Syst. Archit.*, vol. 98, pp. 79–91, Sep. 2019.
[25] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of DAGs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, Dec. 2014.
[26] P. Trojovský and M. Dehghani, "Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications," *Sensors*, vol. 22, no. 3, p. 855, Jan. 2022.
[27] P. Rajesh, S. Muthubalaji, S. Srinivasan, and F. H. Shajin, "Leveraging a dynamic differential annealed optimization and recalling enhanced recurrent neural network for maximum power point tracking in wind energy conversion system," *Technol. Econ. Smart Grids Sustain. Energy*, vol. 7, no. 1, p. 19, Apr. 2022.

SANCHIT received the B.Tech. degree in computer science and engineering from Uttar Pradesh Technical University, Uttar Pradesh, India, in 2013. He is currently pursuing the Ph.D. degree with the Department of Information Technology, Indian Institute of Information Technology Allahabad, Uttar Pradesh. His research interests include scheduling theory, distributed computing, and MANETs.

NAVJOT SINGH (Senior Member, IEEE) received the Ph.D. degree in computer science from the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi. He is currently an Assistant Professor with the Department of Information Technology, Indian Institute of Information Technology Allahabad. He has published articles in reputed journals, such as *Pattern Recognition*, *Digital Signal Processing*, *Applied Intelligence*, *Soft Computing*, *Multimedia Tools and Applications*, *Multimedia Systems*, *Knowledge and Information Systems*, and *Signal, Image and Video Processing*. His areas of specialization include image processing, computer vision, pattern recognition, machine learning, visual saliency, object detection, biomedical image analysis, and intelligent transportation systems. He was a recipient of the UGC-MANF Fellowship and has qualified for UGC-JRF, CSIR-JRF, and GATE examinations.

JAGPREET SINGH received the B.Tech. degree in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2003, the M.S. degree in software systems from the Birla Institute of Technology and Sciences, Pilani, in 2009, and the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Ropar, India, in 2015. He was an Assistant Professor with the Indian Institute of Information Technology Allahabad, from 2015 to 2022. He has been an Assistant Professor with the Indian Institute of Technology Ropar, since 2022. His research interests include parallel and distributed systems, scheduling theory, high-performance computing, and wireless sensor networks.