Received August 31, 2020, accepted September 14, 2020, date of publication September 25, 2020, date of current version October 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3026839

Dynamic and Static Energy Efficient Scheduling of Task Graphs on Multiprocessors: A Heuristic

MANOJ KUMAR[1], LAKHWINDER KAUR[1], AND JAGPREET SINGH[2]
[1]Department of Computer Science and Engineering, Punjabi University, Patiala 147002, India
[2]Department of Information Technology, Indian Institute of Information Technology Allahabad, Prayagraj 211015, India

Corresponding author: Manoj Kumar (manoj@pbi.ac.in)

This work was supported by the Department of Science and Technology (DST), India, under Grant ECR/2018/002794.

ABSTRACT For energy efficient scheduling of task graphs on multiprocessors, dynamic voltage and frequency scaling (DVFS) and duplication are two widely used techniques. DVFS is generally used to utilize the execution slack by lowering the voltage and frequency of a task to decrease the dynamic energy consumption. Whereas duplication decreases the schedule length and communication energy consumption by replicating certain dependent tasks to avoid communication delays. However, while making decisions on DVFS and duplication for a task, the static energy consumption is mostly overlooked. With chip technologies reducing to a few nano meters, static energy consumption due to leakage current has become important. This article proposes a novel polynomial time heuristic that uses both DVFS and duplication to optimize static energy consumption along with dynamic and communication energy when scheduling task graphs on heterogeneous multiprocessors. The proposed list scheduling algorithm also balances schedule length with energy consumption using proposed normalized difference parameters while making scheduling decisions for a particular task. The results demonstrate the ability of the proposed algorithm to decrease the overall energy consumption with an improved or comparable schedule length as compared with other algorithms in various scenarios.

INDEX TERMS Scheduling, duplication, DVFS, multiprocessors, heterogeneity.

I. INTRODUCTION

The static scheduling of task graphs or Directed Acyclic Graphs (DAGs) on heterogeneous multiprocessors is an NP-Hard problem [1], [2]. This problem gained more attention in the research community with the increase in energy consumption of multiprocessors [3], [4]. However, most of the published literature deals with optimizing performance (schedule length or makespan) along with only dynamic power consumption [4], [5]. Dynamic voltage and frequency scaling (DVFS) is one widely employed technique to reduce the dynamic power consumption by running certain tasks on low voltage-frequency pairs. Since, the dynamic power consumption is directly proportional to the frequency and voltage, this decreases the dynamic energy consumption, however, increases the execution cost of the tasks and hence, increases the schedule length. Many scheduling algorithms uses DVFS only for slack reclamation i.e., in an already known schedule, tasks with idle slots (also called as slack) are run on low voltage/frequency to fill the idle slot. By doing this, the energy consumption can be decreased without increasing the schedule length.

Along with the dynamic power consumption, the static power consumption has significantly increased as an implication of the Moore's law [6], [7]. Static power consumption is independent of the tasks or processes being run on the hardware. The major source of the static power is the leakage current which is because of the reverse bias current in a transistor even in an off state. The increasing chip density and hence chip technology dropping below 65*nm* is one of the major reasons of an increasing leakage current. Also, leakage current is directly proportional to the temperature of the device. Hence, if a processor does more computation, then, along with dynamic power consumption, the temperature also rises which increases the leakage current and eventually static power consumption.

Current processor manufacturers employ dynamic power management (DPM) to keep the processors on a low-power state whenever possible to reduce the static power consumption. The Advanced Configuration and Power Interface

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

(ACPI) standard [8], [9] defines various low power states ($S1$ to $S4$) in the standby mode. Each of these low power states save more power than its predecessor, however, requires more time (and energy) to bring the system back to the active state. Hence, a processor needs to be idle for a certain minimum time instance to save on static energy by going into the low power state. For every low power state, this time instance is called as a break-even time.

The DVFS uses the idle slots to decrease dynamic energy consumption whereas, in DPM, these idle slots can be optimized to decrease the static energy consumption. Hence, there is an interesting trade-off in using idle slots for either running tasks on low voltages/frequencies to save dynamic energy consumption or keeping the idle slots within break-even time to save more on static energy consumption. Another interesting trade-off of DVFS and duplication for optimizing computation and communication energy along with performance has been recently explored [10], [11]. The duplication of tasks in idle slots is done to reduce the inter-task communication delay to shorter schedule length [12] and also to reduce the communication energy consumption. Hence, idle slots can be used in three different ways as follows:

1) putting the processor into a sleep state to reduce static energy consumption and also temperature of the processor
2) duplicating tasks to reduce communication delay and energy consumption and also to shorten schedule length
3) use slack reclamation using DVFS to reduce computation energy consumption

This article proposes a polynomial time scheduling heuristic named DSEAS to optimize dynamic, static and communication energy consumption along with the schedule length when tasks graphs are scheduled on heterogeneous multiprocessor using DVFS and duplication. While deciding on dynamic voltage/frequency pair for a task or it's duplicate, DSEAS considers static energy consumption to make optimal decision and hence balance schedule length with the energy consumption. The DSEAS algorithm also uses an integer linear program (ILP) named OptILP-idle to optimally uses an idle slot in the partial schedule to decrease dynamic and static energy consumption. The results show that DSEAS is able to decrease the energy consumption with a better of comparable schedule length than the state-of-the-art algorithms. The rest of the article is organized as follows: Section II describes the related work. The task and system model is presented in section III. The DSEAS algorithm is presented in detail in section IV with OptILP-idle in section IV-A. The experimental results are discussed in section V followed by conclusions and future directions in section VI.

II. RELATED WORK

Static scheduling of DAGs on multiprocessors with only makespan objective is an NP-hard problem [1], [2]. To deal with hardness, researchers have proposed optimal ILPs [13]–[15], meta-heuristics [16]–[19] and low-complexity heuristics [20]–[22]. The Heterogeneous Earliest Finish Time First (HEFT) [20] is one of the first very popular polynomial time list scheduling algorithm, which sorts tasks in a DAG in to a list and later greedily schedule tasks from the list one at a time to reduce the finish time. The HEFT algorithm was later extended with duplication [21] to further shorten the makespan. The last decade has seen the focus with scheduling of task graphs been shifted from performance (reducing makespan) to optimizing energy consumption and also to control the thermal properties of the system. To reduce the dynamic energy consumption, Dynamic Voltage/ Frequency Scaling (DVFS) has been widely utilized [3], [23]–[26]. A detailed survey on energy aware scheduling is presented in [3]. Kappiah *et al.* [23] effectively reduced the processor energy consumption by applying slack reclamation to the MPI programs. They experimentally evaluated the approach without modifying the programs. Son *et al.* [25] simultaneously reduced the voltage on the processors and the network links to utilize the available slack. Rizvandi *et al.* [24] proposed the concept of scheduling certain tasks with integrated max-min voltages to effectively reclaim the slack. Their results were found to be close to the optimal solution.

There are a few efforts to simultaneously reduce the makespan as well as the energy consumption. Lee and Zomaya [26] proposed a polynomial time Energy Conscious Scheduling (ECS) algorithm using DVFS, which uses a parameter that simultaneously considers the energy and the schedule length to make task scheduling decisions. However, it is seen that ECS sacrifices the makespan to reduce the power consumption [11]. Duplication has also been used to decrease the energy consumption [27], [28]. Zong *et al.* have proposed an algorithm that duplicates a task only if it keeps the increase in energy and schedule length below a threshold level. An extension of HLD, named Enery Aware Minimizing Duplication (HLD-EAMD), was proposed [28]. HLD-EAMD works in two steps. First, it runs the HLD algorithm to decrease the makespan with duplication and then, it removes the redundant duplications to decrease the energy consumption. However, both of these algorithms do not uses DVFS and perform poorly in the case of low communication costs. Scheduling with duplication has also been implemented using an Mixed Integer Linear Program (MILP). Bender [13] described an MILP to reduce the makespan with duplication. Tosun and Suleyman [29] used DVFS and duplication for scheduling independent tasks using an MIP. In [10], authors proposed an MILP formulation using DVFS and duplication to optimize both of these objectives together. The formulation has proved to be very effective, however, only for smaller instances of the scheduling problem. A normalization based heuristic with duplication and DVFS is proposed in [11] but it does not optimizes static energy consumption.

All of the above works do not optimise static energy consumption. Niu and Quan [30] proposed a hard real-time algorithm to reduce dynamic and static energy consumption.

TABLE 1. Notations for task, processor, network, and energy model. Notations with capital letters are constant and with small letters are variables.

Notation	**Definition**
F, P	Set of tasks in the application DAG and Set of Processors respectively
G	Set of Edges in the application DAG
$S[m]$	Set of idle states on processor m to calculate static energy consumption
$S[m, s]$	Idle state s on processor m, The state $S[m, 0]$ on processor m is called as an active-idle state
V_k^m	The k^{th} voltage level on processor m
$E_b[m], E_i[m]$	Busy (at V_{max}^m) and active-idle energy consumption of a processor m for a unit time
$E_s^a[m]$	Unit active-static energy consumption of processor m
$E_s^i[m, s]$	Unit idle-static energy consumption of processor m at idle state s
$E_s^p[m, s]$	A penalty in terms of energy consumption to bring processor m from passive-idle state s to active-idle state
$E_b[m, k]$	Energy consumption when processor m is busy at voltage level k for unit time
$Bt[m, s]$	Break-even time of idle state s on processor m
$E_c[m, m']$	Unit communication energy consumption for sending data between processor m and m'
$e_b[i, m, k]$	Busy (or direct) energy consumption of i^{th} task on processor m at voltage level k
$e_i[i, m, s]$	Idle energy consumption of i^{th} task on processor m at idle state s
$T[i, m, k]$	Execution cost (T) of the i^{th} task on processor m at voltage k.
$C[i, j]$	Communication cost from i^{th} task to j^{th} task
f_{max}	Makespan of the task graph
$d[i, m]$	$d \in \mathbb{B}^{F \times P}$ Assignment Indicator Variable to show if a copy of task i is assigned to processor m or not
$d^k[i, m]$	$d^v, in\ \mathbb{R}^{F \times P \times \lvert V \rvert}$, if $d[i, m] = 1$ then copy of task i executes on processor m with fraction $d^k[i, m]$ on voltage level k
e_{dyn}^P	Dynamic energy consumption of processors
e_{sta-a}^P, e_{sta-i}^P	Static-active and static-idle energy consumption of processors
e^C	Energy consumption for communicating data among tasks allocated to various processors
$ec[i, m]$	Combined execution cost of the i^{th} task on processor m
ENB and ENI	Network interface card busy and idle energy consumption respectively
ES	Energy consumption of a network switch
$m, m', i, j, k, k', g, s,$	Indexes - Processors: m, m'; Tasks: i, j; Voltage Level: k, k'; Edges: g; Idle-levels: s

The main idea of their paper is to shift scheduled tasks in such a way that the small idle slots are merged together to make bigger ones with a constraint that none of the task misses its deadline. They used DVFS to finish tasks earlier so that slightly bigger idle slots can be obtained. However, their solution is only for independent tasks. A similar approch to combine execution slacks is proposed in [31]. The authors have used multiple sleep states and also proposed an offline method to calculate break-even time for each sleep state. Chen and Thiele [32] also looked at a similar problem in real-time system with independent tasks. Their approach is to find optimal frequency to run a task such that there is a possibility to push the processors into a dormant state. In [33] authors proposed an algorithm to shutdown the under utilised processors and limit the number of processors along with DVFS to reduce the static energy. Ma *et al.* considered dependent tasks and used clustering based scheduling along with duplication and dynamic power management techniques. They assume that DVFS is not available in the cluster system. An optimal ILP based solution for scheduling hard real-time and mixed-criticality tasks on multiprocessors to optimize static energy consumption is proposed in [34]. This work uses multiple sleep states as utilized in our work. But, they do not use DVFS and the algorithm is designed for independent tasks only. More recently, Kaur *et al.* [35] proposed a duplication based approach combined with dynamic power methods to reduce dynamic as well as static power consumption for dependent tasks. However, their work also does not utilize DVFS. In this paper, we propose a DVFS and duplication based polynomial time heuristic that also focuses on static power consumption along with the dynamic power consumption and balances power consumption with performance i.e., decreasing the makespan.

III. TASK AND SYSTEM MODEL

We define the task and system model used in this study in this section. The system model used in this research is similar to the one in [11], [26], [34]. Table 1 describe all the notations used in this work.

A. TASK, PROCESSOR AND NETWORK MODEL

In static version of the scheduling problem, the information about the task graph which is represented as a directed acyclic graph (DAG), is available in advance. The application DAG (F) is required to be scheduled on a set P of heterogeneous processing elements (PEs). There are N nodes (represented as a set F) of the DAG which represents N non-preemptive tasks. A set $G \in F \times F$ of directed edges between the nodes in the DAG defines dependency among tasks. An edge between two tasks *i* and *j* means that the task *j* can not begin its execution until the task *i* has finished and communicated required data to task *j*. Since, we employ duplication, there can be multiple copies of a task on different processors. In case copies of tasks i and j are scheduled on different

TABLE 2. Equations used to describe the energy model.

$$en_{total} = en^P_{dyn} + en^P_{sta-a} + en^P_{sta-i} + en^C \quad \text{(E)}$$

$$en^P_{dyn} = \sum_{m\in P}\sum_{i\in F}\Big(\sum_{k\in V^m} \underbrace{T[i,m,V^m_k]\cdot d^k[i,m]\cdot E_b[m,k]}_{*}\Big) \quad \text{(E1)}$$

$$en^P_{sta-a} = \sum_{m\in P}\sum_{i\in F}\Big(\sum_{k\in V^m} \underbrace{T[i,m,V^m_k]\cdot d^k[i,m]\cdot E^a_s[m]}_{*}\Big) \quad \text{(E2)}$$

$$en^P_{sta-i} = \sum_{m\in P}\ \sum_{idle\_slots\in m}\Big((len_{idle\_slot} - Bt[m,s]) * E^i_s[m,s] + E^p_s[m,s]\Big) \quad \text{(E3)}$$

$$en^C_N = |P|\cdot ENI\cdot f_{max} + 2\cdot(ENB - ENI)\cdot \underbrace{\sum_{i\in F}\sum_{m\in P}\Big(d[i,m]\cdot \sum_{e_{ij}\in E}(1-d[j,m])\cdot C[i,j]\Big)}_{*} \quad \text{(E4)}$$

$$en^C_S = N_{switch}\cdot f_{max}\cdot ES \quad \text{(E5)}$$

processors then $C[i,j]$ ($\in \mathbb{R}^{F\times F}$) gives the time required to transfer data from the i^{th} task to the j^{th} task. Both the tasks, if scheduled on same processor, are assumed to exchange data through shared memory, which does not incur any cost. $T[i,m] \in \mathbb{R}^{F\times P}$ represents the execution cost of the i^{th} task on the m^{th} processor at the highest voltage (V^m_{max}) and frequency pair. The V^m_{max} represents the maximum voltage of the m^{th} processor.

Each processor in this model can run on multiple voltage and frequency pairs, where V^m_k represent the k^{th} voltage/frequency pair on the m^{th} processor. As energy consumed while transiting between various voltage pairs is small, we do not consider this in our study. Each of the processors, when not in use, can be in one the idle states s where $s \in S[m]$. The set $S[m]$ define idle states on a processor m. We consider four different idle states as have been used in [35]. The state $S[m,0]$ represents active-idle state on a processor m. In an active-idle state, a processor is idle i.e., it is not executing any tasks, however, all of the components of the processor are in active (power-on) state. Generally, a processor is in active-idle state when it is idle for a short duration of time while waiting for some communication to finish. In case a processor is idle for relatively large duration of time then some parts of the processor can be powered-off to bring the processor to a state such that static energy consumption can be reduced. These states are called as passive-idle states. We use three passive-idle states viz. standby, dormant, and shutdown as has been extensively used in the literature [35]. We describe the amount of static energy consumed during these states in the section III-B.

For inter-process communication, we assume a contention free fully connected topology. Each of the processors have a network interface card (NIC) which is used to connect the processor to a network. Here, we assume homogeneous network cards throughout the system, where *ENB* and *ENI* are the network interface busy and idle energy consumption respectively. It has been noticed that a switch consumes almost similar energy in busy as well as idle mode [36]. Hence in busy and idle modes, a unified parameter *ES* is used to describe energy consumption of a switch.

B. ENERGY MODEL

The total energy consumption for scheduling task graphs on heterogeneous multiprocessors is defined in equation **E**[1] in table 2, where, en^P_{dyn} and en^P_{sta} stands for dynamic and static energy consumption of processors respectively. We explain how to evaluate all of these energy components in the following sections.

1) DYNAMIC ENERGY

The dynamic energy consumption (en^P_{dyn}) can be evaluated by calculating the amount of time processors (or cores) are busy executing the tasks. The part $*$ in Equation **E1** gives the energy consumption of a task i on processor m.

In our heuristic, a task is allowed to be executed using integrated voltage-frequency pairs i.e., some fraction on one voltage and the remaining on the other. It is reported in literature [24] that the slack time can be optimally utilized when a task is allowed to be executed on integrated voltages. In part $*$ in Equation **E1**, $d^k[i,m]$ represents fraction of time task i executes on voltage-frequency pair k and $E_b[m,k]$ is the unit busy energy consumption of processor m on V^m_k. $E_b[m,k]$ is evaluated as:

$$E_b[m,k] = \beta_m V^2\cdot f$$
$$\beta_m = \frac{E_b[m]}{((V^m_{max})^2\cdot f)}$$

The β_m is evaluated from the fact that the capacitive (dynamic) power consumption of a processor is defined as $E_b = ACV^2f$, where A: the number of switches per clock cycle, C: the total capacitance load, V, f: voltage and frequency.

2) STATIC ENERGY

A processor consumes static energy consumption while it is busy executing tasks as well as when it is idle. However, when busy (or active), the static energy consumption depends on the thermal properties of the system. Since, in this

[1]We label the equations with symbol **E** for easy understanding that these equations belong to energy model

work, we are not focusing on thermal modeling, we assume a fixed active-static energy consumption for a processor m i.e., $E_s^a[m]$. Hence, total active-static energy consumption of all the processors is:

The equation **E2** is exactly similar to the equation **E1** with a minor difference that now we multiply the execution cost of tasks with unit active-static energy consumption ($E_s^a[m]$). The crucial part of saving static energy consumption is when a processor is in idle state. Depending on the amount of time a processor is idle i.e., the size of the idle slot, a processor can be put into one of the idle states as discussed in section III. The unit idle-static energy consumption ($E_s^i[m, active-idle]$) of a processor decreases as we send a processor to a deeper idle state i.e., we can decrease the static energy consumption:

$$\begin{aligned} E_s^i[m, active-idle] &> E_s^i[m, standby] > E_s^i[m, dormant] \\ &> E_s^i[m, shutdown]. \end{aligned}$$

However, the break-even time ($Bt[m, active - idle]$) to push a processor to a deeper idle state and energy penalty ($E_s^p[m, active-idle]$) to bring a processor to active-idle state from a deeper idle state increases as we move to a deeper idle state as described below:

$$\begin{aligned} Bt[m, active-idle] &< Bt[m, standby] < Bt[m, dormant] \\ &< Bt[m, shutdown] \\ E_s^p[m, active-idle] &< E_s^p[m, standby] < E_s^p[m, dormant] \\ &< E_s^p[m, shutdown] \end{aligned}$$

Assuming an idle slot with a size len, we can find an idle state from $s \in \{shutdown, dormant, standby\}$ for which $l > Bt[m, s]$. Then, the static energy consumption for this idle slot is $(len - Bt[m, s]) * E_s^i[m, s] + E_s^p[m, s]$. The total idle-static energy (en_{sta-i}^P) consumption can be found using equation **E3**.

3) COMMUNICATION ENERGY

The communication energy is mostly consumed by network interface cards (en_N^C) and the switches (en_S^C). Hence, the total communication energy ((en^C)) consumption is calculated as $en^C = en_N^C + en_S^C$ as used in [36]. For en_N^C, we need to evaluate the amount of time network cards are involved in communication during the entire schedule length. The part $*$ in equation **E4** calculates the total communication time between all of the processors in the entire schedule length. If task i and task j have an edge ($e_{ij} = 1$) and $d[i, m] = 1$ and $d[j, m] \neq 1$ then there is a communication for $C[i, j]$ times between processor m where task I is allocated and another processor where task j is allocated.

To simplify the calculation of en_N^C, we assume that all of the network cards consume idle energy consumption for the entire makespan ($|P| \cdot ENI \cdot f_{max}$), where ENI is the unit idle energy consumption of the network cards. Later, after calculating the total communication time in part $*$ in equation **E4**, we add twice the busy network energy consumption for this amount to account for sender and receiving processors and subtract the similar amount of idle energy consumption which was added earlier. Equation **E5** is used to calculate energy consumption of switches.

In equation **E5**, ES is the unit energy consumption of switches. Also, depending on the number of processors, we can calculate the number of switches (N_{switch}) as follows:

$$N_{switch} = \begin{cases} 1, & \text{if } |P| \leq N_{port} \\ \left\lfloor \dfrac{|P|}{N_{port}} \right\rfloor + 1, & \text{otherwise} \end{cases}$$

```
Topologically sort tasks in $F$ based on
    decreasing b-level and store in a list;

for (i in list) {
  $m' = m_0$ ; $V_{k'}^{m'} = V_{min}^{m_0}$;
  for (m ∈ P) {
    set dup=false;
    for ($V_k^m \in V^m$) {
      if ($IF[i, m, V_k^m, m', V_{k'}^{m'}]$ > 0) {
        $m' = m$ ; $V_{k'}^{m'}$ = $V_k^m$;
        dup=false;
      }
      else {
        if ($\exists MIIP[i]$) {
          $V_l^{m'}$ = least voltage for which
              $EST[MIIP[i], m'] + T[MIIP[i], m', V_l^{m'}]$
              $< EST[i, m']$
          if ($\exists V_l^{m'}$) {
            $V_{dup}^{m'} = V_l^{m'}$;
            if {$IF[i, m, V_k^m, m', V_{k'}^{m'}] > 0$} {
              dup=true;
              $m' = m$ ; $V_{k'}^{m'} = V_k^m$ ;
            }
          }
        }
      }
    }
    if (dup=true)
      Schedule[$MIIP[i], m', V_{dup}^{m'}$];

    Schedule[$i, m', V_{k'}^{m'}$];
  }
  Identify and Remove redundant duplications;
  for (every idle slot) {call OptILP();}
}
```

LISTING 1. Algorithm for DSEAS (input: F; output:schedule).

IV. ALGORITHM:DSEAS

The proposed algorithm uses greedy approach as used in [11], [26]. We name our algorithm as Dynamic and Static Energy Aware Scheduler (DSEAS). DSEAS algorithm is significantly different than the other state-of-the-art algorithms since it explores the interplay of DVFS, duplication and DPM to optimize performance, dynamic, static and communication energy consumption at the same time. The main objective is to generate schedules which balances energy consumption with makespan. Algorithm 1 describe the pseudo-code of our heuristic. The algorithm begins by topological sorting of the nodes of the task graph with decreasing values of the b-level.

TABLE 3. Evaluating improvement factor IF.

$IF[i,m,V_k^m,m',V_{k'}^{m'}] = \frac{e_a[i,m',V_{k'}^{m'}] - e_a[i,m,V_k^m]}{e_a[i,m,V_k^m]} + \frac{EFT[i,m',V_{k'}^{m'}] - EFT[i,m,V_k^m]}{T[i,m,V_k^m]}$	**(P1)**
$e_b[i,m,V_k^m] = T[i,m,V_k^m] \times E_b[m,V_k^m] + T[i,m,V_k^m] \times E_a^s[m]$	**(P2)**
if $\exists s \in S$with lowest $E_s^i[m,s]$ such that $Bt[m,s] < T[i,m,V_k^m]$ then $e_i[i,m,V_k^m] = (T[i,m,V_k^m] - Bt[m,s]) * E_s^i[m,s] + E_s^p[m,s]$	**(P3)**
$e_a[i,m,V_k^m]$ **without duplication** $e_a[i,m,V_k^m] = e_b[i,m,V_k^m] - e_i[i,m,V_k^m]$	**(P4)**
$e_a[i,m,V_k^m]$ **with duplication of** $MIIP[i]$ **at voltage** V_{dup}^m $e_a[i,m,V_k^m] = e_b[i,m,V_k^m] + e_b[MIIP[i],m,V_{dup}^m] - e_i[i,m,V_k^m] - e_i[MIIP[i],m,V_{dup}^m]$	**(P5)**

The *b*-level (or bottom-level) for each node is calculated as follows:

$$blevel[i] = \mu^e[i] \qquad \forall i \in F, succ(i) = \phi \tag{1}$$

$$blevel[i] = \mu^e[i] + \max_{j \in succ(i)} \left(c[i,j] + blevel[j]\right) \tag{2}$$

In the equations above, $\mu^e[i] = \left(\frac{\sum_{m \in P} T[i,m,max]}{|P|}\right)$ is taken as the average execution cost of task i on V_k^m where $m \in P$ and $succ(i)$ gives all of the successors of task i. The *b*-level measures the largest path from a task to a leaf task and is widely used in literature for sorting task in a graph to a list for list scheduling heuristics. The tasks are selected for allocation and scheduling based on decreasing values of *b*-level. Initially, for every task, we select any random processor m_0 and the minimum voltage level on m_0(i.e., $V_{min}^{m_0}$). Later, we iterate through every processor (step-5) and all of voltage/frequency pairs (step-7) to see which processor and voltage level is more appropriate for a particular task. To decide this, we make use of a parameter named as Improvement Factor (`IF`). Equation **P1** in table 3 is used to calculate `IF`. The parameter `IF` is defined as the improvement in makespan and energy consumption when a new voltage-frequency pair and a processor is selected for a task as compared to the current mapping. If a task i is currently allocated to a processor m' with voltage $V_{k'}^{m'}$ and we want to see if processor m and voltage level V_k^m should be preferred then the improvement factor should be positive i.e., `IF` should be positive (steps 7-9). The distinct feature of `IF` is that it makes use of affective energy consumption ($e_a[i, m, V_k^m]$) which consider dynamic as well as static energy consumption while scheduling a task with or without duplication. To be the best of authors knowledge this kind of parameter is not used by any known greedy algorithm.

The `IF` in makespan is evaluated as the normalized difference in the earliest finish time (EFT) of task i on $(m', V_{k'}^{m'})$ and (m, V_k^m). Similarly, the `IF` in the energy consumption is evaluated by taking into account the affective energy consumption ($e_a[i, m, V_{k'}^{m'}]$) of the task i on $(m', V_{k'}^{m'})$ and (m, V_k^m). Importantly, we take in to account the affective energy consumption in comparison to only busy energy consumption as used in [11], [26]. The idea behind using affective energy consumption is that in case a task does not execute in a particular slot (i.e., no busy energy consumption) then there is still some idle static energy consumption. Hence, we trade-off dynamic and static energy consumption while scheduling tasks. To calculate the affective energy consumption (equation **P4**), we subtract the idle energy consumption of task i on (m, V_k^m) from the busy energy consumption i.e., $e_b[i, m, V_k^m]$. The $e_b[i, m, V_k^m]$ and $e_i[i, m, V_k^m]$ are calculated in equations **P2** and **P3** respectively.

The $EFT[i, m, V_k^m]$ is the time the task i can finish the earliest while executing on processor m on voltage V_k^m. This also depends on the data arrival time from parents of task i. In case `IF` is negative (steps 12-23) then we try to duplicate the most important immediate parent (`MIIP`) of task i which delays its execution the most on processor m. This can reduce the EFT, however, can also increase the energy consumption, hence, we also consider the duplicated parent while calculating the affective energy consumption as in equation **P5**. In case duplication leads to `IF` > 0 then we schedule this task with duplication (step 26) otherwise without duplication (step 28). Once all of the tasks are scheduled then we remove any redundant duplication in step 27 to create more idle slots in the schedule. Finally, for each idle slot, we optimize the dynamic and static energy consumption by running an optimal ILP as described in section IV-A. The complexity of DSEAS is $|O(|F|log(|F|) + (|F| + |E|) * |P| * V_{max}^2)|$, where V_{max}^2 is the maximum number of voltage levels on a processor. The DSEAS time complexity is a V_{max} factor more than the [26] and [11] because we decide on a voltage level for the duplicated task as well. This helps us to achieve better results without increasing much on the time complexity. For large task graphs, a restricted number of voltage-frequency pairs can be used [15] to reduce the time complexity.

A. OPTILP-IDLE

As highlighted earlier, an idle slot can be used either to reduce the dynamic energy consumption (by running tasks on low voltage/frequency pairs) or to decrease the static energy consumption (by putting the processor to a deep idle state). We propose an integer linear program (ILP) named as OptILP-idle for optimized use of the idle slots. Figure 1 gives the basic idea of the problem which we solve using the ILP.

TABLE 4. Equations in OptILP-idle.

$ec[i,m] = \Big(T[i,m,V^m_{max}] \cdot d^{max}[i,m] + T[i,m,V^m_{min}] \cdot d^{min}[i,m]\Big)$		(I1)
if $e_{ij} = 1$ and task i is giving data to task j		
$ec[i,m] \leq min(DAT[i_m, j_{m'}] - C[i,j], len_{idle\_slot})$		(I2)
otherwise		
$ec[i,m] \leq len_{idle\_slot}$		(I2a)
$en^i_{dyn} = \Big(T[i,m,V^m_{max}] \cdot d^{max}[i,m] \cdot E_b[m,max] + T[i,m,V^m_{min}] \cdot d^{min}[i,m] \cdot E_b[m,min]\Big)$		(I3)
$en^i_{sta-a} = \Big(ec[i,m] \cdot E^a_s[m]\Big)$		(I4)
$M \cdot (x[s]-1) \leq len_{idle\_slot} - ec[i,m] - Bt[m,s] \leq x[s] \cdot M$	$\forall s \in S[m]$	(I5)
$y[s] \leq x[s]$	$\forall s \in S[m]$	(I6)
$\sum_{s \in S[m]} y[s] = 1$	$\forall s \in S[m]$	(I7)
$\Big((len_{idle\_slot} - Bt[m,s]) * E^i_s[m,s] + E^p_s[m,s]\Big) + (y[s]-1) * M \leq en^i_{sta-i}$	$\forall s \in S[m]$	(I8)

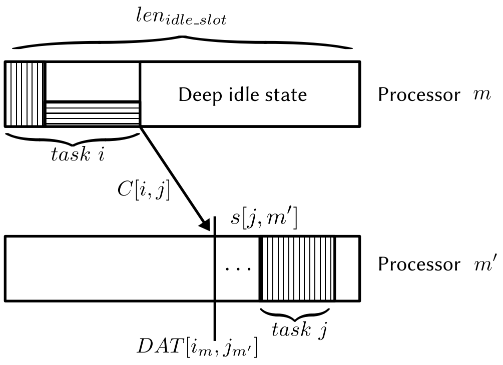


FIGURE 1. Idle slot optimized for static and dynamic energy consumption.

Every idle slot other than at the beginning of a processor is preceded by a scheduled task (for ex. task i). We re-define the size of the idle slot here, which now begins at the start of this task i and finish at the end of the actual idle slot as shown in the figure 1. The problem *Task-in-Idle-Slot* is described as follows:

*Problem 1: **Task-in-Idle-Slot***: Given an idle slot on processor m with size len_{idle_slot} and a task i to be scheduled from the beginning of the slot, minimize the total dynamic and static energy consumption such that the final schedule remains feasible.

The objective of the OptILP-idle is as follows:

$$\texttt{minimise} \quad en^i_{dyn} + en^i_{sta-i} + en^i_{sta-a}$$

where, en^i_{dyn} and en^i_{sta-a} are the dynamic and active static energy consumption of task i on processor m. The en^i_{sta-i} represents the idle static energy consumption of the idle slot that follows task i on processor m. Table 4 describe the constraints required to optimally solve the ***Task-in-Idle-Slot*** problem. As shown in figure 1, the task i is executed by running it on some fraction ($d^{max}[i,m]$) of time on maximum voltage/frequency level (vertical strips) and some fraction ($d^{min}[i,m]$) on minimum voltage/frequency level (horizontal strips) to optimally utilize the idle slot and eventually the dynamic energy consumption. Hence, as in equation **I1**, the execution cost of task i on processor m ($ec[i,m]$) is calculated by the time i runs on max and min voltage/frequency pairs. In case any task j depends on task i for data, the $ec[i,m]$ is upper bounded by a known communication cost ($C[i,j]$) and the largest possible data arrival time ($DAT[i_m, j_{m'}]$) from i to j. The fractions $d^{max}[i,m]$ and $d^{min}[i,m]$ makes the problem interesting since they are used to calculate the total dynamic energy consumption (en^i_{dyn}) of task i on processor m as in equation **I3**. Equation $I4$ gives the total active static energy consumed by task i (en^i_{sta-i}).

Finally, the size of the remaining idle slot, $len_{idle\_slot} - ec[i,m]$ is used to decide a deep idle state to minimize the static energy consumption. Equations **I5**-**I8** achieve this objective of deciding a deep idle state. A variable $x[s]$ is set to 1 if based on the size of the remaining idle slot and the break-even time of state s ($Bt[m,s]$), it is possible to shift processor m to idle state s (equation **I5**). Equations **I6** and **I7** force the ILP solver to select only one of the possible idle states in variable $y[s]$. A state for which $y[s] = 1$ is used to calculate en^i_{sta-i}. The overall minimization objective makes sure that the state that minimize en^i_{sta-i} is selected in $y[s]$.

V. EXPERIMENTAL RESULTS

In this section, we compare DSEAS algorithm with three other state-of-the-art algorithms viz: NormEAS [11], C-SEED [35] and ECS+idle [26]. Table 5 briefly compare these algorithms with their properties.

TABLE 5. State-of-art algorithms compared with DSEAS.

Algorithm	DVFS	Duplication	Static-energy
ECS+idle	Yes	No	No
NormEAS	Yes	Yes	No
C-SEED	No	Yes	Yes
DSEAS	Yes	Yes	Yes

Table 6 provides details of the parameters used to generate widely used real task graphs: LU Decomposition (LUD) [16], Fast Fourier Transform (FFT) and Random task graphs [4]. Approximately ten thousand graphs have been generated by varying graph parameters as listed in table 6 using the tool, Task Graph Generator [37]. The parameters are set same as [11], [26], [35]. Other than these graphs, we also use three

Processor Type	Dynamic Power (Max Frequency)	Power Status	Static Power (W)	Break-even time (ms)	Energy penalty (J)
AMD Turion MT-34	25W (1.8 GHz)	Active-idle	7.5	0	0
		Standby	3.75	0.1	0.1875
		Dormant	0.75	2	6.75
		Shutdown	0.000075	10	37.49
AMD Opteron 2218	95W (2.6 GHz)	Active-idle	28.5	0	0
		Standby	14.25	0.1	0.7125
		Dormant	2.85	2	25.65
		Shutdown	0.000285	10	142.49

(a)

Level	AMD Turion MT-34		AMD Opteron 2218	
	Voltage	Relative Speed (%)	Voltage	Relative Speed (%)
Vmax	1.20	100	1.30	100
1	1.10	80	1.25	92
2	1.00	60	1.20	84
3	0.90	44	1.15	69
4			1.10	38

(b)

FIGURE 2. (a) Processor types with dynamic and static energy consumption (b) voltage-relative speed pairs.

TABLE 6. Graph parameters.

Parameter	Values
Graph Type	LUD, FFT, Random
Number of Processors	2, 4, 8, 16, 32, 64
Communication to Computation Ratio (CCR)	0.1, 0.2, 0.5, 1, 2, 5 10
Number of Nodes	[5, 600]

application task graphs Robot Control (RC), Sparse matrix Solver (SMS), SPEC fpppp (SPECF) from Standard Task Graph Set (STG) [38]. The STG set defines graphs only with execution costs, the communication cost is generated to maintain various CCR values as shown in table 6.

All the algorithms have been implemented in C++. We have used CPLEX ILP Solver [39] to solve OptILP-idle which takes only a few seconds to optimally solve. Figure 2(a) [35] shows the two processor types we use in this study with voltage pairs for both the processors in 2(b) [11], [26]. It is important to mention that the proposed algorithm does not depend on a particular processor architecture and these two processor types are only selected because of the availability of the technical specifications (specifically idle-states parameters) of the processors from research papers. We take equal number of both types of processors while varying number of processors as described in table 6.

For comparison, we have used two performance metrics: SLR (Schedule Length Ratio) and ECR (Energy Consumption Ratio) [26] as used in other relevant studies. The SLR is computed as follows:

$$SLR = \frac{makespan}{\sum_{i \in CP} \min_{m \in P} \left(T[i, m, V^m_{max}] \right)}$$

where, CP is a set of tasks on the critical path of the application task graph. The ECR is evaluated as:

$$ECR = \frac{en_{total}}{\sum_{i \in CP} (\min_{m \in P}\{T[i, m, V^m_{max}]\} \cdot EBV[m, V^m_{max}]) + en^c_{cp}}$$

where en^c_{cp} is the communication energy for Critical Path (CP). The lower the values of SLR and ECR, the better the algorithm has performed. Next, we compare the algorithms by varying CCR, number of tasks and number of processors and then summarize the results based on various graph types.

A. IMPACT OF CCR

Figure 3 shows the impact of varying CCR on SLR (a) and ECR (b). For low values of CCR (<1) i.e., when the computation cost is higher than the communication cost, an only duplication based algorithm C-SEED suffers a little with the energy consumption because of not using DVFS, with limited duplication, it is able to reduce the SLR though. Similarly, ECS+idle sacrifices SLR by running tasks on low voltage/frequency pairs to reduce the dynamic energy consumption and hence reducing ECR than C-SEED. NormEAS does better than the two by using duplication and DVFS both and finding a nice balance between SLR and ECR. However, NormEAS does not do anything to reduce the static energy consumption. Especially, when the number of tasks are higher, even for CCR$<$1, still there are idle slots, which are utilised by DSEAS to reduce the static energy consumption along with dynamic energy consumption by using DVFS. The DSEAS algorithm also employs selective duplication to reduce the SLR than other algorithms and hence, reduces the communication energy consumption as well. Overall, the DSEAS performs the best among other algorithms.

With increasing values of CCR ($\geq$1), communication begins to dominate, and hence, SLR and ECR increases for ECS+idle for not using duplication and any method to reduce the static and communication energy consumption. Since, computation is lower than communication, DVFS impact to reduce only the dynamic energy consumption is negligible. Infact, using only duplication and dynamic power management (DPM) methods to reduce static energy consumption works in favour of C-SEED. With duplication, C-SEED reduces the communication and hence reduces SLR and communication energy consumption. Duplication increases the dynamic energy consumption, however, savings on communication energy consumption dominates dynamic energy

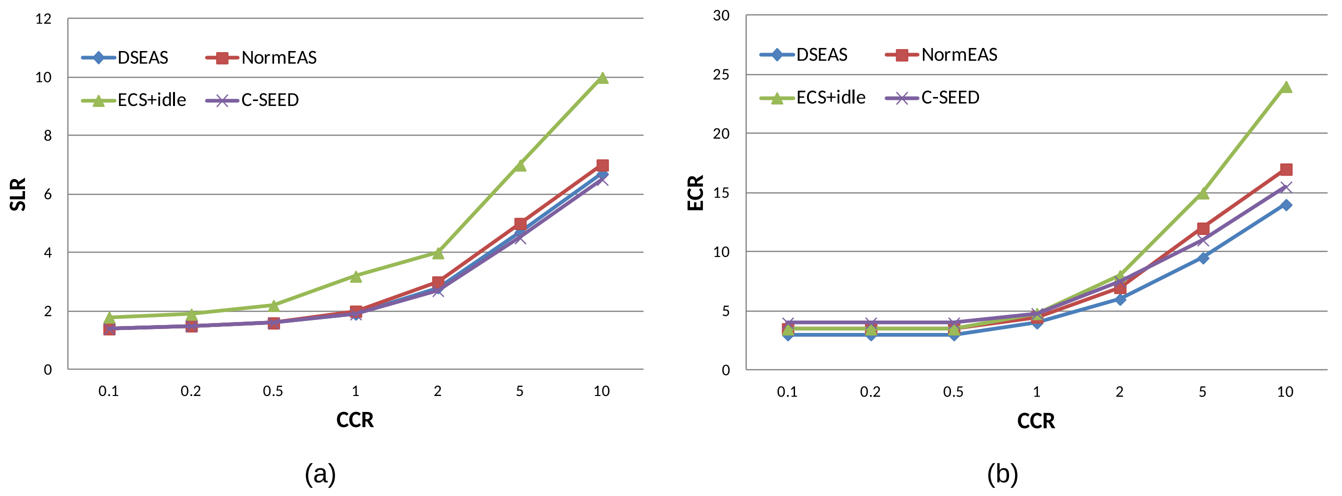


FIGURE 3. Impact of varying CCR on (a) SLR and (b) ECR.

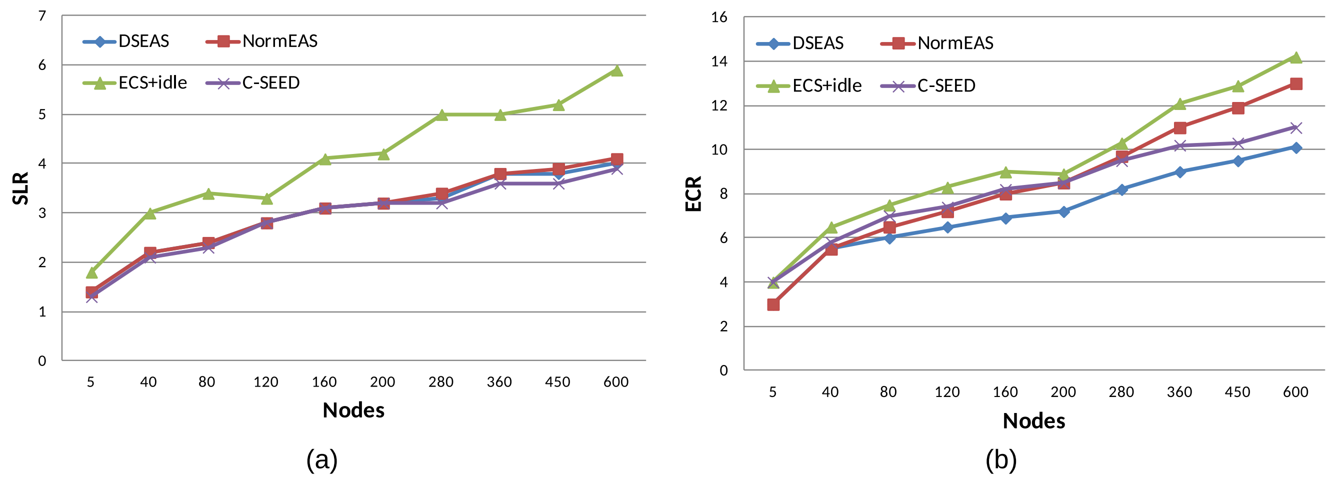


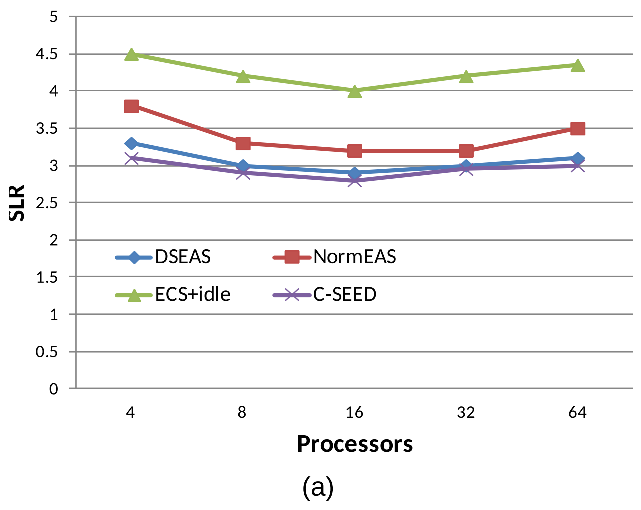
FIGURE 4. Impact of varying number of tasks in the application DAG on (a) SLR and (b) ECR.

consumption. Another, interesting result is that the ECR of C-SEED is lower than NormEAS for CCR ($\geq$2). This is attributed to DPM techniques used by C-SEED to put the processors into deep idle states when the idle slots are big enough. With increasing CCR, the idle slots starts getting bigger and hence saving static energy consumption dominates dynamic energy consumption, since, NormEAS also employs DVFS and duplication to reduce dynamic energy consumption and communication energy consumption.

Interestingly, DSEAS is able to optimize the use of DVFS, duplication and deep idle states to significantly decrease the ECR than C-SEED. The DSEAS algorithm sacrifices a little on SLR (Figure 3(a)) for higher values of CCR but reduces the dynamic energy consumption far more than C-SEED. Also, DSEAS decision of whether to duplicate a task of not depends both on dynamic and static energy factors (refer Table 5, equation **P6**) which helps to do selective and optimized duplication and hence, does not blindly increases redundant duplications. This is also a distinctive factor of DSEAS as compared to NormEAS. Other than this, NormEAS, first allows to do redundant duplications and once all the tasks are scheduled then look for removing redundant duplications. We argue that this is not a good technique because allowing redundant duplications fill up the idle slots which can be used to either to schedule other tasks or used as idle slots to reduce static energy consumptions and DSEAS does exactly the same to reduce ECR than NormEAS.

B. IMPACT OF NUMBER OF TASKS

Another interesting way to look at the results is to compare algorithms on the basis of increasing number of tasks in an application DAG. Figure 4(a,b) shows the impact of number of tasks (or nodes) on SLR and ECR respectively. All duplication based algorithms except ECS+idle achieved similar SLR. This effect is attributed to duplicating tasks to reduce the schedule length, especially, when either communication cost or dependency among tasks increases. Only for high number of tasks, C-SEED is able to improve SLR by a small fraction



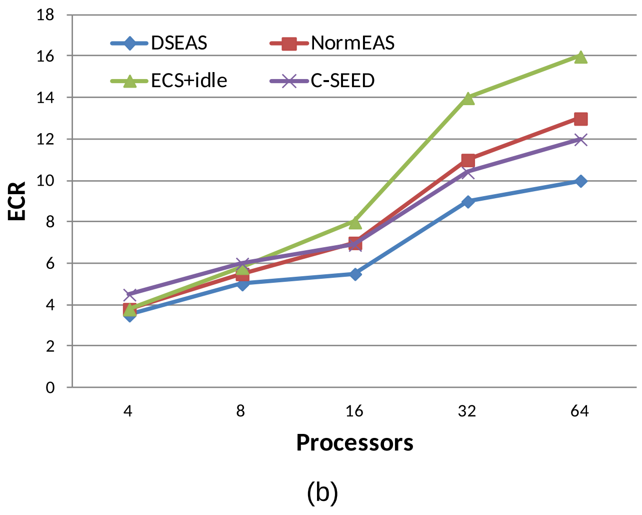


FIGURE 5. Impact of varying number of processors on (a) SLR and (b) ECR.

upon NormEAS and DSEAS. This is because of use of selective duplication and DVFS techniques in NormEAS and DSEAS to balance SLR with ECR as compared to C-SEED. In case of NormEAS, this can be seen for medium sized graphs (<200), where NormEAS is better than C-SEED. However, as the number of nodes in the graph increases (≥ 200), C-SEED is able to pull down ECR than NormEAS. Because, as the number of tasks increases, the dependency among tasks also increase, hence, the delay in executing these tasks increases the idle slots. Increasing size of the idle slots give chance to both C-SEED and DSEAS algorithms to reduce static energy consumption by putting processors into deep idle states.

The DSEAS algorithm is however better than both C-SEED and NormEAS for all the graph sizes while comparing ECR. Compared to NormEAS, DSEAS does duplication optimally by considering the static energy consumption along with dynamic energy consumption of the duplicated task. This step helps DSEAS to prevent redundant duplications and make optimal duplication choices. Secondly, is the optimization of static energy consumption, as in DSEAS every idle slot is optimally used to decrease dynamic as well as static energy consumption as explained in section IV-A. The downside of C-SEED is not using DVFS, which is efficiently used in DSEAS to decrease ECR.

C. IMPACT OF NUMBER OF PROCESSORS

Figure 5 shows the impact of increasing number of processors on SLR and ECR. With an increase in number of processors, the SLR decreases for all of the algorithms, however, increasing processors from 16 to 64, does not improve the SLR further. Because, for relatively small or medium sized graphs, 16 number of processors seems sufficient. The C-SEED algorithm reduces SLR slightly than DSEAS because of increased duplications which can be seen as C-SEED having higher ECR than DSEAS. Also, to save more dynamic energy consumption, DSEAS uses DVFS which slightly affect it SLR as compared to C-SEED. However, with increasing number of processors, the possibility of having (large) idle slot increases, which gives an opportunity to DSEAS and C-SEED to save on static energy consumption. As can be seen in ECR plots, from processors 16 to 64 C-SEED reduces the ECR than NormEAS. However, as is seen in other plots, Opt-ILP helps DSEAS to decrease the static energy further with ECR of DSEAS is significantly better than NormEAS. A without duplication ECS+idle does relatively well as compared to C-SEED for smaller number of processors by scarifying SLR. But clearly is not a good choice to achieve a balance of SLR and ECR. Whereas, DSEAS with an optimized mix of DVFS, duplication and dynamic power management techniques able to achieve a perfect balance of SLR and ECR.

TABLE 7. DSEAS percentage improvement in ECR over other algorithms.

Graph Type	Algorithm		
	ECS+idle	NormEAS	C-SEED
LUD	21.6	12.1	11.2
Random	29.4	18.7	16.6
FFT	26.4	16.5	14.3
RC	22.5	13.3	12.2
SMS	20.4	10.9	11.6
SPECF	25.4	17.8	15.3
Average	24.28	14.88	13.53

D. SUMMARY OF RESULTS

Tables 7 and 8 summarises and presents percentage improvement of DSEAS for ECR and SLR over other algorithms for various graph types. The impact of optimizing both dynamic and static energy consumption at the same time can be easily seen by comparing 14.88% and 13.53% improvement over NormEAS and C-SEED (uses DPM for static energy consumption) achieved by DSEAS. NormEAS reduces dynamic energy consumption for computation dominated task graphs

TABLE 8. DSEAS percentage improvement in SLR over other algorithms.

Graph Type	Algorithm		
	ECS+idle	NormEAS	C-SEED
LUD	23.3	6.7	-4.1
Random	31.1	8.5	-1.5
FFT	26.4	7.2	-2.1
RC	24.8	8.1	-1.2
SMS	22.4	6.1	-2.2
SPECF	29.1	7.7	-0.8
Average	26.16	7.38	-1.98

and small graph sizes with less dependencies as compared to C-SEED whereas C-SEED reduces static energy consumption for high CCR and large graph sizes. However, both of these algorithms are comparable with ECR. But, DSEAS focuses on both dynamic and static energy consumption and performed significantly better than these algorithms to reduce the energy consumption. For SLR, C-SEED does more duplication and reduces makespan on an average 1.98% than DSEAS. Whereas, DSEAS performs significantly better than NormEAS and ECS+idle that tries to balance energy consumption with schedule length. The results clearly shows that it is important to focus both of dynamic as well as static energy consumption and DSEAS does exactly the same.

VI. CONCLUSION

This paper talks about an ingenious polynomial time heuristic DSEAS for scheduling precedence tasks on heterogeneous multiprocessors with a focus on dynamic energy, static energy and communication energy consumption along with the makespan. The DSEAS algorithm optimizes the use of DVFS (for reduing dynamic energy consumption), duplication (for reducing schedule length and communication energy) and DPM (for reducing static energy consumption). The results exhibit that it is important to focus on all three energy consumption along with makespan. The proposed algorithm is able to generate balance schedules where schedule lengths are comparable or better and total energy consumption is always better than the state-of-the-art algorithms. In future, DSEAS can be made temperature aware as well to account the impact of energy consumption on temperature of the system and vice versa.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*, 1st ed. New York, NY, USA: W. H. Freeman, Jan. 1979.

[2] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.

[3] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1447–1464, Jul. 2013.

[4] Z. Deng, Z. Yan, H. Huang, and H. Shen, "Energy-aware task scheduling on heterogeneous computing systems with time constraint," *IEEE Access*, vol. 8, pp. 23936–23950, 2020.

[5] U. U. Tariq, H. Wu, and S. A. Ishak, "Energy-efficient scheduling of tasks with conditional precedence constraints on MPSoCs," in *Towards Integrated Web, Mobile, and IoT Technology*, (Lecture Notes in Business Information Processing), T. A. Majchrzak, C. Mateos, F. Poggi, and T.-M. Grønli, Eds. Cham, Switzerland: Springer, 2019, pp. 115–145.

[6] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, Dec. 2003.

[7] G. Ananthanarayanan, S. R. Sarangi, and M. Balakrishnan, "Leakage power aware task assignment algorithms for multicore platforms," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 607–612.

[8] (2019). *(ACPI) Specification, Version 6.3*. [Online]. Available: https://uefi.org/sites/default/files/resources/ ACPI_6_3_final_Jan30.pdf

[9] (2018). *System Power States*. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/power/system-power-states

[10] J. Singh and N. Auluck, "DVFS and duplication based scheduling for optimizing power and performance in heterogeneous multiprocessors," in *Proc. High Perform. Comput. Symp.*, San Diego, CA, USA, 2014, p. 22.

[11] J. Singh, A. Gujral, H. Singh, J. U. Singh, and N. Auluck, "Energy aware scheduling on heterogeneous multiprocessors with DVFS and duplication," in *Proc. IEEE 17th Int. Conf. Parallel Distrib. Comput., Appl. Technol. (PDCAT)*, Dec. 2016, pp. 105–112.

[12] M. Orr and O. Sinnen, "Integrating task duplication in optimal task scheduling with communication delays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2277–2288, Oct. 2020.

[13] A. Bender, "MILP based task mapping for heterogeneous multiprocessor systems," in *Proc. EURO-DAC Eur. Design Autom. Conf. with EURO-VHDL Exhib.*, Los Alamitos, CA, USA, 1996, pp. 190–197.

[14] A. Davare, J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, "Classification, customization, and characterization: Using MILP for task allocation and scheduling," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2006-166, Dec. 2006.

[15] J. Singh, S. Betha, B. Mangipudi, and N. Auluck, "Contention aware energy efficient scheduling on heterogeneous multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1251–1264, May 2015.

[16] P. Chitra, R. Rajaram, and P. Venkatesh, "Application and comparison of hybrid evolutionary multiobjective optimization algorithms for solving task scheduling problem on heterogeneous systems," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2725–2734, Mar. 2011.

[17] A. J. Page, T. M. Keane, and T. J. Naughton, "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system," *J. Parallel Distrib. Comput.*, vol. 70, no. 7, pp. 758–766, Jul. 2010.

[18] S. G. Ahmad, C. S. Liew, E. U. Munir, T. F. Ang, and S. U. Khan, "A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 87, pp. 80–90, Jan. 2016.

[19] Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, I. Ahmad, and A. Ghafoor, "A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 66, no. 1, pp. 77–98, Jan. 2006.

[20] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[21] S. Bansal, P. Kumar, and K. Singh, "Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs," *J. Parallel Distrib. Comput.*, vol. 65, no. 4, pp. 479–491, Apr. 2005.

[22] S. Baskiyar and C. Dickinson, "Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication," *J. Parallel Distrib. Comput.*, vol. 65, no. 8, pp. 911–921, Aug. 2005.

[23] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs," in *Proc. ACM/IEEE SC Conf. (SC)*, Nov. 2005, p. 33.

[24] N. B. Rizvandi, J. Taheri, A. Y. Zomaya, and Y. C. Lee, "Linear combinations of DVFS-enabled processor frequencies to modify the energy-aware scheduling algorithms," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, May 2010, pp. 388–397.

[25] S. W. Son, K. Malkowski, G. Chen, M. Kandemir, and P. Raghavan, "Integrated link/CPU voltage scaling for reducing energy consumption of parallel sparse matrix applications," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2006, p. 8.

[26] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.

[27] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, "EAD and PEBD: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 360–374, Mar. 2011.

[28] J. Mei and K. Li, "Energy-aware scheduling algorithm with duplication on heterogeneous computing systems," in *Proc. ACM/IEEE 13th Int. Conf. Grid Comput. (GRID)*, Sep. 2012, pp. 122–129.

[29] S. Tosun, "Energy-and reliability-aware task scheduling onto heterogeneous MPSoC architectures," *J. Supercomput.*, vol. 62, pp. 265–289, Nov. 2011.

[30] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proc. Int. Conf. Compil., Archit., Synth. Embedded Syst.*, New York, NY, USA, 2004, pp. 140–148.

[31] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *Proc. 23rd Euromicro Conf. Real-Time Syst.*, Jul. 2011, pp. 92–101.

[32] J.-J. Chen and L. Thiele, "Expected system energy consumption minimization in leakage-aware DVS systems," in *Proc. 13th Int. Symp. Low power Electron. Design (ISLPED)*, 2008, pp. 315–320.

[33] P. de Langen and B. Juurlink, "Trade-offs between voltage scaling and processor shutdown for low-energy embedded multiprocessors," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, S. Vassiliadis, M. Bereković, and T. D. Hämäläinen, Eds. Berlin, Germany: Springer, 2007, pp. 75–85.

[34] V. Legout, M. Jan, and L. Pautet, "Scheduling algorithms to reduce the static energy consumption of real-time systems," *Real-Time Syst.*, vol. 51, no. 2, pp. 153–191, Mar. 2015.

[35] N. Kaur, S. Bansal, and R. K. Bansal, "Duplication-controlled static energy-efficient scheduling on multiprocessor computing system," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 12, p. e4124, Jun. 2017.

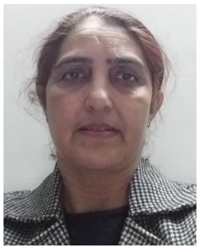
[36] X. Li, Y. Zhao, Y. Li, L. Ju, and Z. Jia, *An Improved Energy-Efficient Scheduling for Precedence Constrained Tasks Multiprocessor Clusters*. Cham, Switzerland: Springer, 2014, pp. 323–337.

[37] (2020). *Task Graph Generator*. [Online]. Available: http://taskgraphgen.sourceforge.net/

[38] (2020). *Standard Task Graph Set*. [Online]. Available: http://www.kasahara.cs.waseda.ac.jp/schedule/index.html

[39] (2020). *IBM ILOG CPLEX Optimization Studio, Version 12.10.0 Documentation*. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.10.0/COS_KC_home.html

MANOJ KUMAR received the master's degree from Guru Nanak Dev Engineering College (GNDEC), Punjab Technical University, Jalandhar, in 2010. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Punjabi University, Patiala, India. He is an Assistant Professor with the Department of Computer Science and Engineering, YCOE (Punjabi University Guru Kashi Campus Talwandi Sabo), Bathinda, India. His research interests include scheduling, distributed computing, and MANETs.



LAKHWINDER KAUR received the Ph.D. degree from the PTU, Jalandhar, India. She is a Professor in computer engineering with Punjabi University, Patiala, India. She has been in teaching since September 1992. She has published more than 50 research articles in various reputed international journals and conference proceedings. She has written three books. Her research interests include image processing, parallel computing, and computer networks.

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 at Pilani in 2009, and the Ph.D. degree in computer science and engineering from the Indian Institute of Technology at Ropar, India, in 2015. He is working as an Assistant Professor with the Indian Institute of Information Technology Allahabad, India, since 2015. His research interests include parallel and distributed systems, scheduling theory, high performance computing, and wireless sensor networks.