symmetry

MDPI

Article

Dynamic Priority Real-Time Scheduling on Power Asymmetric Multicore Processors

Basharat Mahmood [1], Naveed Ahmad [2], Majid Iqbal Khan [1] and Adnan Akhunzada [3,*]

[1] Department of Computer Science, COMSATS University Islamabad, Islamabad 46000, Pakistan; basharatmahmood@comsats.edu.pk (B.M.); majid_iqbal@comsats.edu.pk (M.I.K.)
[2] Department of Computing, National University of Computer and Emerging Sciences, FAST-NU, Islamabad 46000, Pakistan; naveed.ahmad@nu.edu.pk
[3] Faculty of Computing and Informatics, University Malaysia Sabah, Kota Kinabalu 88400, Malaysia
* Correspondence: adnan.akhunzada@ums.edu.my

Abstract: The use of real-time systems is growing at an increasing rate. This raises the power efficiency as the main challenge for system designers. Power asymmetric multicore processors provide a power-efficient platform for building complex real-time systems. The utilization of this efficient platform can be further enhanced by adopting proficient scheduling policies. Unfortunately, the research on real-time scheduling of power asymmetric multicore processors is in its infancy. In this research, we have addressed this problem and added new results. We have proposed a dynamic-priority semi-partitioned algorithm named: Earliest-Deadline First with C=D Task Splitting (EDFwC=D-TS) for scheduling real-time applications on power asymmetric multicore processors. EDFwC=D-TS outclasses its counterparts in terms of system utilization. The simulation results show that EDFwC=D-TS schedules up to 67% more tasks with heavy workloads. Furthermore, it improves the processor utilization up to 11% and on average uses 14% less cores to schedule the given workload.

Keywords: real-time scheduling; EDF scheduling; semi-partitioned scheduling; power asymmetric multicore processors

Citation: Mahmood, B.; Ahmad, N.; Khan, M.I.; Akhunzada, A. Dynamic Priority Real-Time Scheduling on Power Asymmetric Multicore Processors. *Symmetry* **2021**, *13*, 1488. https://doi.org/10.3390/sym13081488

Academic Editor: Iver H. Brevik

Received: 12 July 2021
Accepted: 8 August 2021
Published: 13 August 2021

1. Introduction

The use of real-time systems has grown rapidly due to their assorted application areas ranging from simple household electronics to fully automated industrial control systems [1,2]. These systems are characterized by temporal constraints, and the fulfillment of these constraints is considered as necessary as executing the tasks correctly. The temporal correctness can be effectively achieved by designing efficient task scheduling policies [1,3]. A real-time scheduler decides the execution order of time dependent tasks. Its essential goal is to schedule tasks so that they can meet their timing constraints. Proficient scheduling approaches not only improve the system utilization but can also be integrated with other power management techniques to attain high power efficiency. Dynamic voltage and frequency scaling (DVFS) [4] and memory shut-down [5] are instances of such improvements. On DVFS enabled processors, supplied voltage and clock frequency are dynamically adjusted depending upon the current workload. In this way, the system consumes less power when the workload is on lower side. Similarly, in memory shut-down technique unused memory is dynamically shut-down. This also results in reduced energy consumption. Both DVFS and memory shut-down can be effectively integrated with real-time scheduling to achieve energy efficiency.

In recent times, real-time applications have grown extensively in complexity. Multicore processors are considered more favorable for implementing such complex and processing intensive applications due to their proficiency in terms of energy consumption and heat

generation [6]. Multicore processors are fundamentally categorized as homogeneous, heterogeneous, or power asymmetric (also known as uniform or single-ISA heterogeneous) [6]. In homogeneous multicore processors, all of the cores have similar functional and processing capabilities while a heterogeneous multicore processor may contain cores with different functional and processing capabilities [7]. On the other hand, processing cores in a power asymmetric multicore processor are similar in functional capabilities but they may differ in their processing capabilities [7–9]. Power asymmetric processors are viewed as better in terms of energy consumption when contrasted with its other counterparts [8,9].

Research on single-processor real-time scheduling is considered developed yet there is as yet a generous space for research on multiprocessor/multicore scheduling [10,11]. The existing multicore real-time scheduling approaches are categorized as partitioned, global, or semi-partitioned [11,12]. In partitioned scheduling, the given workload is divided into m subsets (where m is the number of cores) such that each subset k is feasible on corresponding core k. During execution each subset is executed on its assigned core. This task to core binding is permanent and no task can migrate during the execution [11,12]. On the other hand, in global scheduling, all of the tasks are placed in a single prioritized queue and the scheduler assigns them to cores according to their priorities. Therefore, during execution tasks can migrate from one core to another [11,12]. Generally, global scheduling is considered superior to partitioned scheduling in terms of system schedulability but suffers from high runtime overheads. Semi-partitioned scheduling is presented as a compromise between pure partitioned and global scheduling in order to reduce the runtime overheads associated with the global scheduling and to improve the performance of partitioned scheduling. The semi-partitioned scheduling extends the partitioned scheduling by allowing a small number of tasks to migrate, which results in improved system utilization [11,12].

Although a lot of work has been done in multicore scheduling but still none of the existing approaches achieve optimal performance. The best-known utilization bound for both global and partitioned scheduling algorithms is 50%, while the semi-partitioned scheduling improves it up to 65% [11,13]. Furthermore, most of the results are based on the homogenous multicore processors whereas research on power asymmetric multicore scheduling is still in its infancy. The power consumption has become a main challenge for future embedded system designs; therefore, it is much needed to consider the power-efficient power asymmetric multicore processors while addressing the real-time scheduling problem [11]. In this paper, we have considered the dynamic priority real-time scheduling of power asymmetric multicore processors and proposed a semi-partitioned scheduling algorithm named Earliest-deadline First with C=D Task Splitting (EDFwC=D-TS). EDFwC=D-TS algorithm allocates the tasks to cores in decreasing order of their utilizations while the cores are sorted in descending order of their processing power. It utilizes the C=D heuristic to split tasks. The simulation results show that EDFwC=D-TS outclasses its counterparts and provides better system utilization.

The rest of the paper is organized as follows. Section 2 presents the existing work that is closely related to the addressed problem. System and task models are given in Section 3. In Section 4 the EDFwC=D-TS algorithm is presented. Experimental evaluation of the proposed work is presented in Section 5. Section 6 presents the evaluation of the EDFwC=D-TS algorithm while our work is concluded in Section 7.

2. Related Work

In this section, we present the existing work that is generally pertinent to the addressed problem. Since semi-partitioned scheduling utilizes uniprocessor schedulability analysis while doling out tasks to cores, we therefore first discuss some significant results on uniprocessor dynamic-priority real-time scheduling. Subsequently, we discuss the most significant existing literature on dynamic-priority semi-partitioned real-time scheduling and power asymmetric multi-processor scheduling.

2.1. Uniprocessor Scheduling

In 1973, Liu and Layland did the pioneer and the most influential work in real-time scheduling theory (presented in [14]). Under the dynamic-priority category, they proposed an optimal algorithm named Earliest-deadline First (EDF) [14]. EDF assigns the highest priority to the task that has the least absolute deadline. Liu and Layland have proved that EDF achieves 100% system utilization, i.e., any task-set Γ can be feasibly scheduled on a single processor system using the EDF algorithm if $U(\Gamma) \leq 1$.

Baruah et al. [15] derived an exact schedulability test known as Processor Demand Analysis (PDA) for sporadic arbitrary relative deadline tasks. According to the PDA a task-set Γ is EDF schedulable if:

$$h(t) \leq t \ \ \forall t > 0 \tag{1}$$

where $h(t)$ is the function that computes the maximum CPU time required by all tasks which have both arrival times and relative deadlines in the interval of length t where $h(t)$ is given by Equation (2):

$$h(t) = max\left(0, \sum_{j=1}^{n} \left\lfloor \frac{t + P_j - D_j}{P_j} \right\rfloor\right) \times C_j \tag{2}$$

where P_j is the period, D_j is the relative deadline, and C_j is the worst-case execution time of the task j. Since the value of t may be very large, it may take a lot of time to determine the feasibility of a task-set using PDA. In [15], Baruah et al. determined an upper bound L_a on the value of t given as follows:

$$L_a = \max\left\{D_1, \ \ldots, D_n, \max_{1 \leq i \leq}\{P_i - D_i\}\frac{U}{1-U}\right\} \tag{3}$$

In Equation (3), U represents the system utilization factor of the given workload. Therefore, the feasibility condition for task-set under PDA is given by Equation (4):

$$h(t) \leq t \ \ \forall t < L_a \tag{4}$$

Ripoll et al. [16] further reduced the upper bound on the maximum value of t that is given by Equation (5):

$$\sum_{i=1}^{n}\{P_i - D_i\}\frac{U_i}{1-U} \tag{5}$$

Ripoll et al. [16] and Spuri [17] derived a recursive function to determine the upper bound (L_b) on the value of t given by Equation (6):

$$s^{q+1} = \sum_{i=1}^{n} \left\lceil \frac{s^q}{P_i} \right\rceil C_i \tag{6}$$

The initial value of S^q is set equal to $\sum_{i=1}^{n} C_i$. The recurrence S^{q+1} is solved until it gives the same value in two consecutive iterations.

Zhang and Burns [18] presented the Quick-convergence Processor-demand Analysis (QPA) to efficiently determine the feasibility of task-sets. The QPA reduces the calculation effort exponentially. It starts by selecting the upper (L) and lower bound (d_{min}) on the value of t where $L = \min(L_a, L_b)$ and d_{min} is equal to minimum relative deadline of the task $\tau_i \in \Gamma$. Next, starting from $t = L$, QPA computes $h(t)$ for t and if it is found less than t then it is replaced with $h(t)$. This process continues as long as t reaches to d_{min} or $h(t)$ is found greater than t. Zhang et al. further extended their work to perform sensitivity analysis on EDF scheduling [19].

2.2. Semi-Partitioned Multi-Processor Scheduling

Anderson et al. introduced the semi-partitioned scheduling, presented in [20]. They introduced the notion of task-splitting to improve the system utilization and proposed the EDF-fm algorithm for scheduling recurrent soft real-time tasks on multiprocessor systems. Under the hard dynamic priority semi-partitioned category, EDF with task splitting and K processors in a Group (EKG) is a well-known algorithm [21]. EKG classifies the tasks into heavy and light. Each of the heavy tasks is assigned to a separate processor while the light tasks are sequentially allocated to the remaining processors.

In [22], Kato et al. presented the Ehd2-SIP. It assigns tasks to processors sequentially starting from the first processor. If the utilization of the current task τ_i is less than or equal to the remaining capacity of the current processor P_m i.e., $U(\tau_i) \leq U_b - U(P_m)$ then τ_i is assigned to P_m. Otherwise, it is split into two portions τ_i' and τ_i'' where τ_i'' is assigned to P_m while τ_i'' is assigned to $P_{(m+1)}$. Kato et al. presented the Earliest Deadline Deferrable Portion (EDDP) algorithm in [23]. EDDP first classifies the tasks as light or heavy. A task τ_i is considered as heavy if:

$$U_i > 4\sqrt{2} - 5 \cong 65\%$$

All of the remaining tasks are considered light tasks. Heavy tasks are assigned to dedicated processors while the light tasks are sequentially assigned to the remaining processors. EDDP algorithm achieves a utilization bound of 65%. Kato et al. further extend their results on semi-partitioned scheduling and proposed Earliest Deadline and Highest Priority Split (EDHS) algorithm [24]. EDHS algorithm assigns global and highest static priority to migratory tasks while the priorities to fixed tasks are assigned using EDF algorithm. Earliest Deadline First with Window-constrained Migration (EDF-WM) algorithm, presented in [25], aims at improving the system schedulability with reduced context switching cost. EDF-WM algorithm assigns tasks to processors on first-fit basis. When a task is not feasible on any processors then it is split across more than one processor.

Burn et al. addressed the dynamic priority semi-partitioned scheduling of periodic tasks on identical processors and proposed the C=D heuristic for task-splitting [26]. When a task τ_i is required to be split into sub-tasks τ_i' and τ_i'', the deadline of τ_i' is set equal to its worst-case execution time. In this way, τ_i' always has the highest priority on its assigned core. This reduces the task-splitting penalty.

In [27], Anderson et al. extends the *E*DF-fm algorithm presented in [20] and proposed Earliest Deadline First with Optimal Semi-partitioned (EDF-os) scheduling algorithm. EDF-fm algorithm restricts the migrating tasks to have utilization less than 0.5. It assigns high-priority to migrating tasks while the priorities to fixed tasks are assigned in EDF manner.

2.3. Power Asymmetric Multiprocessor Scheduling

The problem of power asymmetric multiprocessor scheduling was first addressed in [28]. In [28], Baruah studied the dynamic priority scheduling of periodic real-time tasks on power asymmetric multicore processors with integer boundary constraint on task preemptions and showed that the general problem is intractable.

In [29], Funk et al. has provided an online algorithm based on global EDF scheduling. They have derived sufficient condition to determine the EDF feasibility of a set of tasks on a power asymmetric processor provided that this task set is known to be feasible on some different power asymmetric processor. This algorithm suffers from high runtime cost due to task migrations.

In [30], the fixed priority real-time scheduling of periodic tasks on power asymmetric processors is considered and a sufficient test to determine the feasibility of a set of tasks is derived. This test works well for task sets with low utilization but fails to determine schedulability of high utilizations task sets. Andersson et al. has addressed the real-time scheduling problem of sporadic tasks and proposed a partitioned dynamic priority algorithm named EDF-DU-IS-FF [31]. The EDF-DU-IS-FF algorithm partitions the task set based on the processor capacity and at runtime; these partitioned tasks are executed in

EDF fashion. The same authors have discussed the fixed priority scheduling of sporadic tasks in [32] and proposed the RM-DU-IS-FF algorithm. This algorithm first partitions the task set using the L.L bound and then these tasks are executed in RM fashion. This algorithm fails to fully utilize the processor capacity due to the usage of sufficient test during partitioning stage and as a result does not perform well at higher system utilization levels.

In [33], Cucu et al. has shown that, according to the global fixed-priority scheduling, any schedule of asynchronous periodic task sets that is feasible on a power asymmetric processor becomes periodic after a specific moment in time. They have determined that point and have provided a feasibility interval for such systems. In [34], a sufficient test for global EDF scheduling of sporadic task system on a power asymmetric multicore processor is presented. The scheduling of soft real-time periodic tasks on power asymmetric multicore processor is studied in [35]. The authors have argued that there are deficiencies in the Linux system for supporting real-time periodic tasks. They discussed the way to provide better performance for the soft workload in the presence of hard workload using deferrable servers.

Chen et al. presented the online-scheduling algorithms PG and PCG for scheduling periodic tasks on a power asymmetric multicore processor [36]. These algorithms assign tasks with largest remaining execution time to the fastest processor. PG and PCG algorithms incur high runtime cost in the form of context switches and task migrations. In [37], A-S algorithm is proposed as an improvement over the PCG algorithm to reduce the runtime costs. A-S algorithm reduces the task preemptions and task migrations up to 90% and 87% respectively. Risat et al. has discussed the RM based global scheduling of periodic implicit-deadline tasks in [38]. They have provided a set of schedulability conditions based on easily computable task-set parameters for providing better utilization along with maintaining the feasibility. They have showed that their conditions provide better performance than other counterparts. Jung et al. studied the scheduling of harmonic real-time tasks in [39]. They have proposed a RM based partitioned approach. They first partition the task set based on processor capacity using the harmonic bound and then split the remaining tasks if required.

3. System and Task Models

We have considered a power asymmetric multicore processor having m cores. The processing power of these cores is defined by the set $S = \{S_1, S_2, \ldots, S_m\}$, where S_i is the processing power of the ith core and $S_i = S_j$ *or* $S_i \neq S_j \; \forall \, i, j \in S$. The total processing power of the system can be calculated by $S = \sum_{i=1}^{m} S_i$.

We have considered the standard real-time task model to characterize the workload of the system. The system workload is represented by the set Γ, which consists of n real-time periodic tasks. Each task τ_i is characterized by its worst-case execution requirements, i.e., the number of *CPU* cycles required by τ_i to complete its execution in the worst-case (C_i); its minimum inter-arrival time (P_i), i.e., its period; and its relative deadline (D_i). The time instant at which the first job of a task is released is known as its phase and is denoted by ϕ_i. We assume that all of the tasks in Γ follow the implicit deadline model, i.e., $P_i = D_i \; \forall \, i \in \Gamma$. In addition, we assume that all of the tasks are synchronous, i.e., $\phi_i = 0 \; \forall \, i \in \Gamma$.

Furthermore, we have considered another task parameter, i.e., the additional offset. The additional offset of a task τ_i denoted by δ_i is the amount of time for which τ_i remains blocked and is not considered for scheduling, i.e., during the time interval $r_i + \delta_i$ (where r_i is the release time of τ_i) remains blocked. Initially, the additional offset of all of the tasks is zero, i.e., $\delta_i = 0 \; \forall \, i \in \Gamma$. Thus, we can define a task τ_i by a 4-tuple $(\delta_i, C_i, P_i, D_i)$. Typical examples of this kind of workload include sensory data acquisition system, air traffic control systems, environment monitoring system, etc. These systems have to periodically execute certain tasks and the completion of those tasks within a specified time is mandatory. Any delay in the completion of these tasks may have catastrophic consequences.

Due to the difference in processing speed of different cores, a task takes a different amount of time on different cores to complete its execution. The time a task $\tau_i \in \Gamma$ takes to complete its execution on core $j \in S$ can be calculated by Equation (7):

$$T(i,j) = \frac{C_i}{S_j} \tag{7}$$

The fraction of the processor time required by a task to complete its execution is known as its *system utilization factor.* The system utilization factor of a task $\tau_i \in \Gamma$ on core $j \in S$ is calculated using the Equation (8):

$$U(i,j) = \frac{T(i,j)}{P_i} = \frac{\left(C_i/S_j\right)}{P_i} \tag{8}$$

While the system utilization factor of a task $\tau_i \in \Gamma$ on the processor S is given by Equation (9):

$$U(i) = \frac{T(i,j)}{P_i} = \frac{\left(C_i/\sum_{i=1}^{m} S_i\right)}{P_i} \tag{9}$$

Now, the total its system utilization factor of the task set Γ is given by Equation (10):

$$U(\Gamma) = \sum_{i=1}^{n} U(i) = \sum_{i=1}^{n} \frac{\left(C_i/\sum_{j=1}^{m} S_j\right)}{P_i} \tag{10}$$

A task set Γ can only be considered for scheduling if $U(\Gamma) \leq 1$ and any task set with $U(\Gamma) > 1$ can never be feasible with any scheduling algorithm. The used notations are given in Table 1.

Table 1. Used notations and their meanings.

Notation	Meanings
Γ	Task-set representing the workload
τ_i	Task $i \in \Gamma$
C_i	*CPU* cycles required by task i in worst-case
D_i	Relative deadline of the task i
P_i	Period of the task i
S	Set defining the power asymmetric multicore processor
S_i	Processing power of the core i
$T(i,j)$	Time required by task i to complete its execution on core j
$U(i,j)$	System utilization factor of the task i on core j
$\Theta(n)$	Value of the *L.L* bound for n tasks
δ_i	Additional-offset of task i

4. Dynamic Priority Semi-Partitioned Scheduling of Real-Time Tasks on Power-Asymmetric Multicore Processors

In this section, we present our dynamic-priority algorithm named EDFwC=D-TS (EDF Scheduling with C=D task-splitting) for scheduling real-time tasks on power-asymmetric multicore processors. EDFwC=D-TS works in two phases: task-allocation and scheduling. In task-allocation phase, the given workload is distributed among the processor cores, while in scheduling phase, tasks are scheduled using the EDF algorithm on each core. In the following sections, we discuss the EDFwC=D-TS scheduling algorithm in detail.

4.1. Task-Allocation in EDFwC=D-TS Scheduling

In this section, we discuss the task-allocation phase under the EDFwC=D-TS scheduling. Task-allocation deals with the mapping of tasks to cores. In task-allocation, a given set of tasks Γ is divided into at most M (M is the number of cores in S) subsets such that each subset Γ_i is EDF-feasible on the corresponding core S_i i.e., $U(\Gamma_i) \leq 1$. Task-allocation in EDFwC=D-TS assigns tasks to a core with the provision that its capacity is fully utilized. When the capacity of a core gets exceeded due to the assignment of a task, a task is split over multiple cores using the C=D heuristic.

Task-allocations scheme under EDFwC=D-TS scheduling is given by Algorithm 1. In Algorithm 1, it is assumed that tasks in Γ are sorted in descending order of their utilization, i.e., $U_i > U_j \; \forall i < j$. Furthermore, it is also assumed that the processor cores are sorted in descending order of their speed, i.e., $S_i > S_j \; \forall i < j$.

Algorithm 1 starts by assigning the tasks to the first core, i.e., the fastest core. It first calculates the system utilization of the next task τ_i on the current core m i.e., $U(\tau_i, m)$ and then adds it to the total system utilization of the tasks which are already assigned to the core m i.e., $U(\Gamma_m)$; here Γ_m represents the set of tasks which are already assigned to the core m (Line 3–4).

Now, if after adding $U(\tau_i, m)$ the total system utilization of Γ_m i.e., $U(\Gamma_m)$ remains less than the EDF-bound that is 1, then τ_i is removed from Γ and it is added in Γ_m (Lines 5–7). Similarly, τ_i is also removed from Γ and is assigned to Γ_m in cases when the addition of $U(\tau_i, m)$ to $U(\Gamma_m)$ makes it equal to 1. Furthermore, $U(\Gamma_m) = 1$ means that the capacity of the core m is fully utilized and therefore the next core should be considered for the allocation of remaining tasks. For this the index of current core, i.e., m is increased by 1 (Line 8–11).

In cases when the addition of $U(\tau_i, m)$ to $U(\Gamma_m)$ results in $U(\Gamma_m)$ getting greater than 1, i.e., the core's capacity gets exceeded, then Algorithm 1 examines the remaining unassigned tasks in Γ to find a task that can be accommodated on core m, i.e., a task τ_i with $U(\tau_i, m) \leq 1 - U(\Gamma_m)$ (Lines 12–27). If no such unassigned task exist in Γ then the last task in Γ, i.e., $\tau_{|\Gamma|}$, is added in Γ_m and task-splitting is performed (Lines 28–31). This means that a task τ_i from Γ_m is selected for splitting into two subtasks τ_i' and τ_i'' such that τ_i' is assigned to core m while τ_i'' is assigned to some other core k where $k > m$. The task-splitting under EDFwC=D-TS is discussed in Section 4.2. In the end, Algorithm 1 decides the feasibility of task-set (Lines 34–38). This decision is made on the number of cores used. If the number of used cores is greater than the total number of cores, i.e., M, then the task-set is declared not feasible.

4.2. Splitting Tasks in EDFwC=D-TS Scheduling

In this section, we discuss the task-splitting under the EDFwC=D-TS scheduling. Task-splitting chooses a task $\tau_i \in \Gamma_m$ to be split into two subtasks τ_i' and τ_i'' such that τ_i' is feasible on core m, i.e., $U(\Gamma_m) + U(\tau_i'') - U(\tau_i) \leq 1$. τ_i' is allocated to the core m while τ_i'' is allocated to the slowest core where it is feasible. For this purpose, the feasibility of τ_i'' is tested on S_M, i.e., the core having lowest processing power. If the system utilization of τ_i'' exceeds than the residual capacity of the S_M then the next core, i.e., S_{M-1} is considered. This process continues unless τ_i'' is feasibly assigned to some core. This indexing of cores is important to reduce the task-splitting penalty (as shown in Theorem 4).

To make the choice of the task to split, each task $\tau_i \in \Gamma_m$ is split one after the other in increasing order of their deadlines using C=D heuristic and its feasibility is determined using the QPA algorithm until a feasible task is found. This process is performed in the following steps:

- A task $\tau_i : (0, C_i, P_i, D_i) \in \Gamma_m$ such that it has the minimum deadline among the tasks in Γ_m and is not previously tested is selected
- τ_i is split into two subtasks $\tau_i' : (C', P_i, D_i = C')$ and $\tau_i'' : (C_i - C', P_i, D\_i - C')$ using C=D heuristic such that $U(\Gamma_m) + U(\tau_i') - U(\tau_i) = 1$
- In Γ_m, τ_i is replaced by τ_i' and its feasibility is determined using QPA algorithm

- If Γ_m is feasible then it is allocated to the core m otherwise the next task is tested in the similar way
- If no task in Γ_m is found feasible then for each task τ_i the worst-case execution time of τ_i' is reduced and tested again
- This process continuous until a feasible task is found or the worst-case execution time of τ_i' for each task becomes equal to zero

Algorithm 1 Task-Allocation under the EDFwC=D-TS Algorithm

Input: (i) Set of n real-time implicit-deadline periodic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ where tasks are sorted in descending order of their utilization i.e., $U_i > U_j \; \forall i < j$. **(ii)** Single-ISA heterogeneous multicore processor $S = \{S_1, S_2, \ldots, S_M\}$ having M processing cores where processing cores are sorted in descending order of their processing speed i.e., $S_i > S_j \; \forall i < j$.
Output: Assigns tasks to cores; and returns success if Γ is feasible on S

1: $M = |S|;\ i = 1;\ m = 1;\ U\left(\Gamma_{m\ (m=1,\ 2,\ \ldots,\ M)}\right) = 0$
2: ***While*** $\Gamma \neq \varnothing \ AND \ m \leq M$ ***Do***
3: $U_i = \frac{C_i}{S_i \times P_i}$
4: $U(\Gamma_m) = U(\Gamma_m) + U_i$
5: ***If*** $U(\Gamma_m) < 1$ ***Then***
6: $\Gamma = \Gamma - \tau_i$
7: $\Gamma_m = \Gamma_m \cup \tau_i$
8: ***ElseIf*** $U(\Gamma_m) == 1$ ***Then***
9: $\Gamma = \Gamma - \tau_i$
10: $\Gamma_m = \Gamma_m \cup \tau_i$
11: $m = m + 1$
12: ***Else***
13: $U(\Gamma_m) = U(\Gamma_m) - U_i$
14: ***For*** $j = i + 1$ *to* $|\Gamma|$ ***Do***
15: $U(\Gamma_m) = U(\Gamma_m) + U_j$
16: ***If*** $U(\Gamma_m) < 1$ ***Then***
17: $\Gamma = \Gamma - \tau_i$
18: $\Gamma_m = \Gamma_m \cup \tau_i$
19: ***ElseIf*** $U(\Gamma_m) == 1$***Then***
20: $\Gamma = \Gamma - \tau_i$
21: $\Gamma_m = \Gamma_m \cup \tau_i$
22: $m = m + 1$
23: *break*;
24: ***Else***
25: $U(\Gamma_m) = U(\Gamma_m) - U_i$
26: ***End If***
27: ***End For***
28: $U(\Gamma_m) = U(\Gamma_m) + U_{|\Gamma|}$
29: $\Gamma = \Gamma - \tau_i$
30: $\Gamma_m = \Gamma_m \cup \tau_i$
31: $\Gamma_m = split - task(S_m, \Gamma_m, U(\Gamma_m))$
32: ***End If***
33: ***End While***
34: ***If*** $m \leq M$ ***Then***
35: ***return*** (*"Allocation is Successful"*)
36: ***Else***
37: ***return*** (*"Allocation Failed"*)
38: ***End If***

The task-splitting process used in EDFwC=D-TS is given in Algorithm 2. Since the total system utilization of tasks assigned to core m is greater than its capacity, i.e., $U(\Gamma_m) > 1$, Algorithm 2 first computes the fraction of $U(\Gamma_m)$ by which it exceeds the core's capacity (Line 2). Furthermore, due to the reason that the suitability of tasks for splitting is determined in ascending order of their deadlines, consequently tasks in Γ_m are sorted in ascending order of their deadlines (Line 3). In the next step, for each task the fraction of

worst-case execution requirements (C'_i) that can be allocated to core m given that this task is chosen for splitting is computed (Line 4–6).

Once these basic computations are made, Algorithm 2 splits each task $\tau_i \in \Gamma_m$ into two subtasks, τ'_i and τ''_i, one by one and determines their feasibility using the QPA Algorithm (Lines 9–22). To support the varying processing power of cores, the processor demand function is modified as given in Equation (11).

$$h(t) = Max\left(0, \sum_{j=1}^{|\Gamma_m|} \left\lfloor \frac{t + P_j - D_j}{P_j} \right\rfloor\right) \times \left(\frac{C'_i}{S_m}\right) \tag{11}$$

when the splitting of some task τ_i is found feasible on core m then τ_i is replaced with τ'_i in Γ_m and Γ_m is allocated to the core m. If none of the tasks is found feasible for splitting then the worst-case execution requirements (C'_i) of tasks is reduced using the recursive function presented in [16] (Lines 23–30). However, to support varying processing speed of cores some basic modifications are required as given in Equation (12).

$$C'_i(r+1) = \frac{1 - oth(t)}{\left\lfloor \frac{t + P_i - C'_i(r)}{P_i} \right\rfloor} \tag{12}$$

where

$$C'_i(r) = \frac{C'_i}{S_m}$$

and

$$oth(t) = \sum_{\tau_j \in \Gamma_m \wedge \tau_j \neq \tau_i} \left\lfloor \frac{t + P_j - D_j}{P_i} \right\rfloor$$

For the reduced C'_i of each task, the feasibility of task-splitting is determined again until some feasible task-splitting is found. If the reduced C'_i for all tasks reaches to 0 then the task-splitting is failed. In this case, the minimum utilization task $\tau_i \in \Gamma_m$ is removed from Γ_m and added to the set of unassigned tasks Γ (Lines 32–34). If the task-splitting is successful, i.e., a task $\tau_i \in \Gamma_m$ is split into two subtasks τ'_i and τ''_i then τ''_i, defined by $\left(C'_i/(S_m), C_i - C'_i, D_i - C'_i, P_i\right)$, is assigned to the slowest core where it is feasible (Lines 35–46). Here, it can be seen that the additional offset of τ''_i is increased from 0 to $\frac{C'_i}{S_m}$. This ensures that τ'_i and τ''_i never execute simultaneously. Finally, Algorithm 2 returns Γ_m (Line 47).

Analysis of Task-Splitting in EDFwC=D-TS Scheduling

Usually, partitioned scheduling fails to fully utilize the processor capacity. Consider the allocation of tasks to the core m. Suppose, Γ_m is the set of tasks that are already assigned to the core m. Further assume that none of the unassigned tasks can be assigned entirely to core m then $1 - U(\Gamma_m)$ capacity of the core m will remain unused. Similarly, the total wasted capacity on the processor S denoted by λ(S) can be calculated as given below:

$$\lambda(S) = \sum_{j=1}^{M} \left(1 - U\left(\Gamma_j\right)\right)$$

Semi-partitioned scheduling aims at using this wasted processor capacity to improve the system schedulability. In EDFwC=D-TS scheduling, during allocation of tasks to the core m, when τ_i is split the gained advantage can be written as:

$$U\left(\tau'_i\right) = \frac{C'_i}{P_i \times S_m}$$

Algorithm 2 Split-task process under EDFwC=D-TS Scheduling

Input: (i) Processing core having processing power S_m, which is currently being considered for task allocation. **(ii)** Set of tasks Γ_m where $\Gamma_m \subseteq \Gamma$ and contains the tasks which are assigned to core m by Algorithm 1. **(iii)** System utilization of Γ_m i.e., $U(\Gamma_m)$
Output: Selects a task $\tau_i \in \Gamma_m$ and splits it into two subtasks.

$flag = flase$
$udiff = U(\Gamma_m) - 1$
$sort\_increasing(\Gamma_m, Deadline)$ // sort tasks in Γ_m; in increasing order of deadline
For $i = 1$ *to* $|\Gamma_m|$ ***Do***
  $C_{temp}[i] = C_i - (S_m \times udiff \times P_i)$
End For
While $flag == false$ ***Do***
  $flag_{zero} = flase$
  For $i = 1$ *to* $|\Gamma_m|$ ***Do***
    $\Gamma_{temp} = \Gamma_m$
    $\Gamma_{temp}[i] \Leftarrow (0,\ C_{temp}[i],\ C_{temp}[i], P_i)$
    If $C_i \neq 0$ ***Then***
      $flag_{zero} = true$
      $Result = Q.P.A(\Gamma_{temp})$
      If $Result == Feasible$ ***Then***
        $\Gamma_m[i] \Leftarrow (0,\ C_{temp}[i],\ C_{temp}[i], P_i)$
        $\tau_i'' \Leftarrow \left(\frac{C_{temp}[i]}{S_m}, C_i - C_{temp}[i], D_i - C_{temp}[i], P_i\right)$
        $m = m + 1$
        $flag = true$
      End If
    End If
  End For
  If $result \neq feasible\ AND\ flag_{zero} == true$ ***Then***
    For $i = 1$ *to* $|\Gamma_m|$ ***Do***
      $C_{temp}[i] = Re - compute(C)$
    End For
  ElseIf $flag_{zero} == false$ ***Then***
    $flag == true$
  End ElseIf
  End If
  End While
  If $result \neq feasible$ ***Then***
    $\Gamma = \Gamma \cup minU(\Gamma_m)$
    $\Gamma_m = \Gamma_m - minU(\Gamma_m)$
  else
    $Assingflag = false$
    While $Assingflag == false\ \wedge M > m$ ***Do***
      $U(\Gamma_M) = U(\Gamma_M) + U_{\tau_i''}$
      If $U(\Gamma_M) > 1$ ***Then***
        $M = M - 1$
      Else
        $\Gamma_M = \Gamma_M \cup \tau_i''$
        $Assingflag = true$
      End If
      $M = M - 1$
    End While
    If $Assingflag = false$ ***Then***
      return "Task set is not feasible"
    End If
End If
return Γ_m

Task-splitting has non-zero penalty on the system. It is incurred in the form of increased system utilization of the split task. Suppose a task $\tau_i \in \Gamma_m$ is split into two subtasks $\tau_i' : \left(0, C_i', C_i'/S_m, P_i\right)$ and $\tau_i'' : \left(C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i\right)$, then the utilization of split tasks is always higher than τ_i. From now onwards we call it the task splitting penalty and it is given by Inequality 13:

$$U(\tau_i') + U(\tau_i'') > U(\tau_i) \tag{13}$$

If we replace $U(\tau_i')$, $U(\tau_i'')$, and $U(\tau_i)$ with their actual value then the task-splitting penalty ($\rho(\tau_i)$) can be written as given in equality 14:

$$\rho(\tau_i) = \frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m} \tag{14}$$

Now, if the advantage of task-split always remains greater than its overhead then we can say that task-splitting benefits the system. This condition is given by Inequality 15:

$$\frac{C_i'}{P_i \times S_m} > \frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m} \tag{15}$$

In the following, we prove that task-splitting always benefits the system.

Theorem 1. *The advantage gained due to task-splitting in EDFwC=D-TS scheduling is always greater than its overhead on the system.*

Proof of Theorem 1. We assume that currently tasks are being assigned to the core m. Furthermore, we assume that τ_i is selected to split into two subtasks, τ_i' and τ_i'', and τ_i' is assigned to the core m. To proof Theorem 1, we have to show that:

$$\frac{C_i'}{P_i \times S_m} > \frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m}$$

$$\Rightarrow \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m} + \frac{C_i'}{P_i \times S_m} - \frac{C_i'}{D_i \times S_m} < 0 \tag{16}$$

As in implicit-deadline task model $\mathrm{P_i} = \mathrm{D_i}$, the Inequality 16 can be written as:

$$\frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m} < 0$$

$$\Rightarrow \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i - C_i' + C_i'}{P_i \times S_m} < 0 \tag{1}$$

$$\Rightarrow \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i - C_i'}{P_i \times S_m} - \frac{C_i'}{P_i \times S_m} < 0 \tag{2}$$

$$\Rightarrow \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i - C_i'}{P_i \times S_m} < \frac{C_i'}{P_i \times S_m} \tag{17}$$

As $\frac{C_i - C_i'}{P_i \times S_m} < \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m}$, therefore by replacing a smaller value with a larger value:

$$\frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} < \frac{C_i'}{P_i \times S_m} \Rightarrow 0 < \frac{C_i'}{P_i \times S_m}$$

Since $\frac{C_i'}{P_i \times S_m} > 0$, therefore it is proved that the advantage gained due to the task-splitting remains always greater than its overhead on the system. □

Now, we show that the task-splitting under EDFwC=D-TS satisfies the necessary task-splitting condition, i.e., the split tasks can never execute simultaneously.

Theorem 2. *Given a task τ_i , split into two subtasks $\tau_i' : (0, C_i', C_i'/S_m, P_i)$ and $\tau_i'' : (C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i)$ using the Algorithm 2. If τ_i' is assigned to core m where $m < M$ and τ_i'' is assigned to some other core k where $m < k \leq M$ then τ_i' and τ_i'' can never execute simultaneously.*

Proof of Theorem 2. Since the task-splitting under EDFwC=D-TS assigns an additional-offset equal to C_i'/S_m to τ_i'' , τ_i'' remains in blocked state for C_i'/S_m for an amount of time after its release. To show that τ_i' and τ_i'' can never execute simultaneously, we have to prove that τ_i' is always completed before the additional-offset of τ_i'' , i.e., the worst-case response time of τ_i' always remains less than or equal to the additional-offset of τ_i'' . It can be written as:

$$R(\tau_i') \leq \delta_i$$
$$\Rightarrow R(\tau_i') \leq \frac{C_i'}{S_m} \tag{18}$$

The worst-case response time of a task is equal to the sum of its execution time and the interference from high priority tasks before its completion. Since τ_i' has the highest priority, therefore its worst-case response time always remains equal to its worst-case execution time, i.e.,

$$R(\tau_i') = \frac{C_i'}{S_m} \tag{19}$$

By comparing Equations (18) and (19), it is clear that the necessary task-splitting condition is satisfied in EDFwC=D-TS scheduling. □

In Theorem 2, we have showed that due to the assignment of additional offset δ_i to the split task, τ_i'' ensures that τ_i' and τ_i'' never execute simultaneously, since, during δ_i time interval, τ_i'' remains blocked. Therefore, it is required to ensure that it does hurt its deadline. We prove this property in Theorem 3.

Theorem 3. *Given a task τ_i, split into two subtasks $\tau_i' : (0, C_i', C_i'/S_m, P_i)$ and $\tau_i'' : (C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i)$ using Algorithm 2. If τ_i' is assigned to core m where $m < M$ and τ_i'' is assigned to some other core k where $m < k \leq M$ then the assignment of additional offset (δ_i) to τ_i'' does not affect its schedulability.*

Proof of Theorem 3. We assume that a task τ_i is split into two subtasks, $\tau_i' : \left(0, C_i', C_i'/S_m, P_i\right)$ and $\tau_i'' : \left(C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i\right)$. Further assume that τ_i' is allocated to core m while τ_i'' is allocated to some other core k where $m < k \leq M$. Since, τ_i'' is feasible on core k, therefore:

$$R(\tau_i'') \leq D_i - \frac{C_i'}{S_m}$$
$$\Rightarrow I_{hp(i)} + C_i - C_i' \leq D_i - \frac{C_i'}{S_m} \tag{20}$$

Now, to show that $\tau_i^{''}$ remains schedulable after the assignment of additional offset (δ_i) we have to prove that:

$$\left(\tau_i^{''}\right) + \delta_i \leq D_i \Rightarrow I_{hp(i)} + C_i - C_i' + \frac{C_i'}{S_m} \leq D_i$$

$$\Rightarrow I_{hp(i)} + C_i - C_i' \leq D_i - \frac{C_i'}{S_m} \tag{21}$$

By comparing Inequalities 20 and 21 it is proved that the assignment of additional offset to $\tau_i^{''}$ does not affect its deadline. □

After proving that the task-splitting under EDFwC=D-TS maintains operational accuracy, now we show that the indexing of processor cores used in EDFwC=D-TS task-splitting reduces the task-splitting penalty.

Theorem 4. *Given a Single-ISA heterogeneous multicore processor S having M cores, the indexing of cores in descending order of their processing speed reduces the task-splitting penalty.*

Proof of Theorem 4. Suppose the task τ_i is split into two subtasks $\tau_i' : \left(0, C_i', C_i'/S_m\ , P_i\right)$ and $\tau_i^{''} : \left(C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i\right)$ using Algorithm 2. Further assume that τ_i' is assigned to core m where $m < M$ and $\tau_i^{''}$ is assigned to some other core k where $m < k \leq M$. From (14), the task-splitting penalty is given by:

$$\rho(\tau_i) = \frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m} \tag{22}$$

Given a single-ISA heterogeneous multicore processor S has M cores, the indexing of cores in descending order of their processing speed reduces the task-splitting penalty.

That is, $\rho(\tau_i)$ time on core k is wasted due to task-splitting. Now, assume that S_k is the processing speed of k^{th} core when processor cores are sorted in descending order of their processing speeds; while S_k^* is its processing speed when processor cores are sorted in ascending order of their processing speeds then:

$$S_k^* \geq S_k$$

Now, the $\rho(\tau_i)$ when the processor cores are sorted in descending order of their processing speeds is given by (23):

$$\rho(\tau_i) \times S_k \tag{23}$$

Furthermore, the $\rho(\tau_i)$ when the processor cores are sorted in ascending order of their processing speeds is given by (24):

$$\rho(\tau_i) \times S_k^* \tag{24}$$

Since $S_k^* \geq S_k$, therefore by comparing (20) and (21) it is proved that the task-splitting penalty is lower when processor cores are sorted in descending order of their processing speed. □

Task-splitting under EDFwC=D-TS assigns the $\tau_i^{''}$ to the lowest index core where it is feasible. We can show by using the Theorem 4 that this approach helps to minimize the task-splitting overhead. We prove this in Theorem 5.

Theorem 5. *Given a task τ_i is split into two subtasks $\tau_i' : \left(0, C_i', C_i'/S_m, P_i\right)$ and $\tau_i^{''} : (C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i)$ using Algorithm 2, if τ_i' is assigned to core m where $m < M$ and $\tau_i^{''}$ is assigned to some other core k where $m < k \leq M$ then the task-splitting penalty is minimized if $k = M$ provided that the processor cores are indexed in descending order of their processing speed.*

Proof of Theorem 5. Assume that the task $\tau_i \in \Gamma_m$ is split into two subtasks $\tau_i' : (0,\ C_i', C_i'/S_m, P_i)$ and $\tau_i^{''} : (C_i'/S_m, C_i - C_i', D_i - C_i'/S_m, P_i)$ using Algorithm 2. Further assume that

τ_i' is assigned to the core m where $m < M$ and τ_i'' is assigned to some other core k where $m < k \leq M$. From (14), the time wasted on core k due to task-splitting is given by (25):

$$\rho(\tau_i) = \left(\frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{(D_i - \frac{C_i'}{P_i \times S_m}) \times S_m} - \frac{C_i}{P_i \times S_m}\right) \times S_k \tag{25}$$

As the processor cores are sorted in descending order of their processing speed, therefore:

$$\min_{k \to M}(S_k) \tag{26}$$

From (13):

$$U(\tau_i') + U(\tau_i'') > U(\tau_i) \Rightarrow \frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} > \frac{C_i}{P_i \times S_m}$$

$$\Rightarrow \frac{C_i'}{D_i \times S_m} + \frac{C_i - C_i'}{\left(D_i - \frac{C_i'}{P_i \times S_m}\right) \times S_m} - \frac{C_i}{P_i \times S_m} > 0 \tag{27}$$

By comparing (25) and (27) it is clear that $\rho(\tau_i)$ keeps on increasing as S_k increases. Furthermore, (26) shows that S_k decreases as $k \to M$ and it is minimum when $k = M$. Therefore, it can be concluded that $\rho(\tau_i)$ decreases as $k \to M$ and it is minimum when $k = M$. □

4.3. The EDFwC=D-TS Scheduling Algorithm

After discussing the task-allocation, now we present the EDFwC=D-TS scheduling algorithm. In EDFwC=D-TS scheduling given by Algorithm 3, the given workload is first distributed among the processor cores using the Algorithm 1 (Lines 1–5). On each core j, initial task priorities are assigned using the EDF algorithm (Lines 6–7). At time t = 0, the highest priority task is executed on each core (Lines 9–10). Whenever a task is completed, the highest priority ready task is executed next (Lines 11–12). Similarly, when a task is released its absolute deadline is compared with the absolute deadline of the currently executing task and the task with earliest absolute deadline is selected for execution (Lines 13–16).

4.4. Working Example

In this section, in order to illustrate the working of EDFwC=D-TS algorithm, we apply it on an example task-set. We have assumed a power asymmetric multicore processor having 3 cores, defined by the set S = {2.0 GHz, 1.5 GHz, 1.0 GHz}. The workload consists of 10 synchronous periodic implicit-deadline real-time tasks given by Table 2.

Table 2. Example Task-set.

Task (τ_i)	CPU Cycles Required in Worst-Case (C_i)	Task Period (P_i)	Task Deadline (D_i)	System Utilization Factor ($\frac{C_i}{D_i}$)
τ_1	4×10^9	6 s *	6 s	0.6667
τ_2	3×10^9	5 s	5 s	0.6
τ_3	6×10^9	12 s	12 s	0.5
τ_4	6×10^9	12 s	12 s	0.5
τ_5	9×10^9	20 s	20 s	0.45
τ_6	12×10^9	30 s	30 s	0.40
τ_7	2×10^9	6 s	6 s	0.3333
τ_8	5×10^9	15 s	15 s	0.3333
τ_9	4×10^9	15 s	15 s	0.2667
τ_{10}	1×10^9	4 s	4 s	0.25

* s stands for seconds.

Algorithm 3: EDFwC=D-TS Scheduling Algorithm

Input: (i) Set of tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with $P_i \geq P_j \; \forall \; i > j$ **(ii)** a Single-ISA heterogeneous multicore processor $S = \{S_1, S_2, \ldots, S_M\}$ with $S_i \geq S_j \; \forall i < j$
Output: Schedule Γ on S

```
rv = Task − Allocation(Γ, S) // partition task-set using Algorithm 3.1
If rv == "Feasible" Then
For j = 1 to M Do
        Core_j ⇐ Γ_j        // Assign Γ_j ⊆ Γ to core j
End For
On each core j :
        Assign priorities to each τ_i ∈ Γ_j according to EDF Algorithm
  While (System Remains Active) Then
        t = 0    // At beginning
                Core_j ⇐ τ_h(p)
        On task completion
                Core_j ⇐ τ_h(p)
        On task release
                If d(τ_rel) < d(τ_cur) Then
                        Core_j ⇐ τ_rel
                End If
End While
Else
Print "Fail to Schedule"
End If
```

As EDFwC=D-TS algorithm assumes that processor cores are sorted in descending order of their speeds, therefore tasks are first allocated to the core having highest processing speed, i.e., 2.0 GHz. Initially, the utilization of core 1 is set to 0, i.e., $U_1 = 0$. The system utilization factor of τ_1 on core 1 is 0.3333 (calculation is given below):

$$\frac{C_1}{P_1 \times S_1} = \frac{4 \times 10^5}{6 \times 2 \times 10^5} = 0.3333$$

After allocating τ_1 to core 1, its utilization becomes 0.3333 ($U_1 = 0 + 0.3333$). Since $U_1 < 1$, therefore, τ_1 is feasibly allocated to core 1. Next, the τ_2 is considered for allocation on core 1. Its system utilization factor on core 1 is 0.3. After allocating τ_2 to core 1, U_1 turns out to be 0.6333 ($U_1 = 0.3333 + 0.3 = 0.6333$). Since U_1 is still less than the capacity of the core 1, i.e., 1, therefore τ_2 is also feasible on core 1. Subsequently, τ_3 is assigned to core 1 after τ_2. The system utilization factor of τ_3 on core 1 is 0.25 and after allocating τ_3 to core 1 U_1 grows to 0.8833. The next task considered for allocation to the core 1 is τ_4. The system utilization factor of τ_4 on core 1 is 0.25. After allocating τ_4 to core 1, its utilization becomes 0.1333. Since U_1 gets greater than the capacity of core 1, therefore τ_4 is not feasible on core 1.

Now, the next task, i.e., τ_5, is considered for allocation to core 1. The utilization factor of τ_5 on core 1 is 0.225 and if this task is allocated to core 1 then U_1 becomes 1.1083. Therefore, τ_5 is also not feasible on core 1. We continue to test the feasibility of remaining tasks. Since none of the remaining tasks, i.e., from τ_6 to τ_{10}, is feasible on core 1, therefore τ_{10} which has the lowest utilization factor, is added to Γ_1 and task splitting is performed.

Task-splitting under EDFwC=D-TS scheduling (given by Algorithm 2) selects a task from Γ_1 that is most suitable for splitting, i.e., maximizes the U_1 without hurting the system feasibility. The suitability of tasks for splitting is determined in decreasing order of their deadlines. Since τ_{10} has the least deadline among the tasks in Γ_1, therefore its suitability for splitting is determined first. For this, τ_{10} is split into two subtasks: τ'_{10} and τ''_{10} such that $U(\Gamma_1 - \tau''_2) = 1$. Subtasks of τ_{10} are: $\tau'_{10} : (0, 0.932272 \times 10^5, 0.466136, 4)$ and $\tau''_{10} : (0.466136, 0.067728 \times 10^5, 3.533864, 4)$. Now, τ_{10} is replaced with τ'_{10} in Γ_1 and the feasibility of Γ_1 is determined using the QPA Algorithm.

The QPA Algorithm determines the feasibility of Γ_1 in the following steps:

- The value of L is calculated using Equation (3); L = 59.99204
- Value of d_{min} is set to 0.466136
- The initial value of t is assigned using $t = \max\{\text{d\_i} \mid \text{d\_i} < \text{L}\}$; $t = 56.466136$

Value of $h(t)$ is calculated against each value of t (calculations are given in Table 3).

Table 3. Value of $h(t)$ against each value of t.

t	$h(t)$	t	$h(t)$
56.46136	53.49204	28.729088	25.229088
53.49204	49.525904	25.229088	24.762952
49.525904	47.559768	24.762952	23.262952
47.559768	42.093632	23.262952	17.796816
42.093632	40.127496	17.796816	13.83068
40.127496	37.66136	13.83068	11.86544
37.66136	36.16136	11.86544	6.398408
36.16136	35.695224	6.398408	4.432272
35.695224	30.695224	4.432272	0.466136
30.695224	28.729088		

Since at $t = 4.432272$ the value of $h(t)$ is 0.466136 and $t < d_{min}$, it leads to the conclusion that $\Gamma_1 = \{\tau_1, \tau_2, \tau_3, \tau'_{10}\}$ is feasible on core 1. Now, τ''_{10} is assigned to the slowest core where it is feasible. Since core 3 is the slowest core and currently no task is assigned to it, therefore $U_3 = 0$. The system utilization factor of τ''_{10} on core 3 is 0.016. Therefore, it is feasible on core 3. Similarly, the remaining tasks are allocated to core 2 and 3. The final task allocation is given in Table 4.

Table 4. Final task allocation.

Core	Allocated Tasks	Processor Utilization
1	$\tau_1 : (0,\ 4 \times 10^9,\ 6,\ 6)$, $\tau_2 : (0,\ 3 \times 10^9,\ 5,\ 5)$, $\tau_3 : (0,\ 6 \times 10^9,\ 12,\ 12)$, $\tau'_{10} : (0,\ 932272 \times 10^9,\ 0.466136,\ 4)$	1
2	$\tau'_4 : (0, 4.571592 \times 10^9,\ 3.047728,\ 12)$, $\tau_5 : (0,\ 9 \times 10^9,\ 20,\ 20)$, $\tau_6 : (0,\ 12 \times 10^9,\ 30,\ 30)$, $\tau_9 : (0,\ 4 \times 10^9,\ 15,\ 15)$	1
3	$\tau''_4 : (3.047728,\ 1.428408 \times 10^9,\ 8.952272,\ 12)$, $\tau_7 : (0,\ 2 \times 10^9,\ 6,\ 6)$, $\tau_8 : (0,\ 5 \times 10^9, 15, 15)$, $\tau''_{10} : (0.466136,\ 0.067728 \times 10^9,\ 3.533864,\ 4)$	0.8454

Once the task-allocation phase is completed, tasks are scheduled on each core using the EDF algorithm. On core 1, at $t = 0$, the first job of τ_1, τ_2, τ_3, and τ'_{10} is ready for execution. Since τ'_{10} has the least absolute deadline, therefore it is executed first. The worst-case execution time of τ'_{10} on core 1 is 0.466136 s $\left(\frac{C'_{10}}{S_1} = \frac{0.9322272 \times 10^9}{2 \times 19^9} = 0.466136\right)$. At $t = 0.466136$, the first job of τ'_{10} is completed. Since, at $t = 0.466136$, τ_2 has the least absolute deadline among the ready tasks, therefore its first job is started. At $t = 1.9666136$, $J_{2,1}$ is completed and τ_1 is started. The WCET of τ_1 on core 1 is 2; therefore, it is completed at $t = 3.9666136$. τ_3 is the only ready task at $t = 3.9666136$, therefore its first job is started. At $t = 4$, second job of τ'_{10} ($J'_{10,2}$) is released. Since, $J'_{10,2}$ has earlier absolute deadline, i.e., 8 than currently executing task $J_{3,1}$ therefore, $J_{3,1}$ is preempted and $J'_{10,2}$ is executed. The execution of tasks continues in the similar way. The Gantt chart showing the execution of tasks on core 1 during the first hyper-period is given in Figure 1.

The Gantt chart showing the execution of tasks on core 2 is given in Figure 2 while is Figure 3 shows the execution of tasks on core 3.

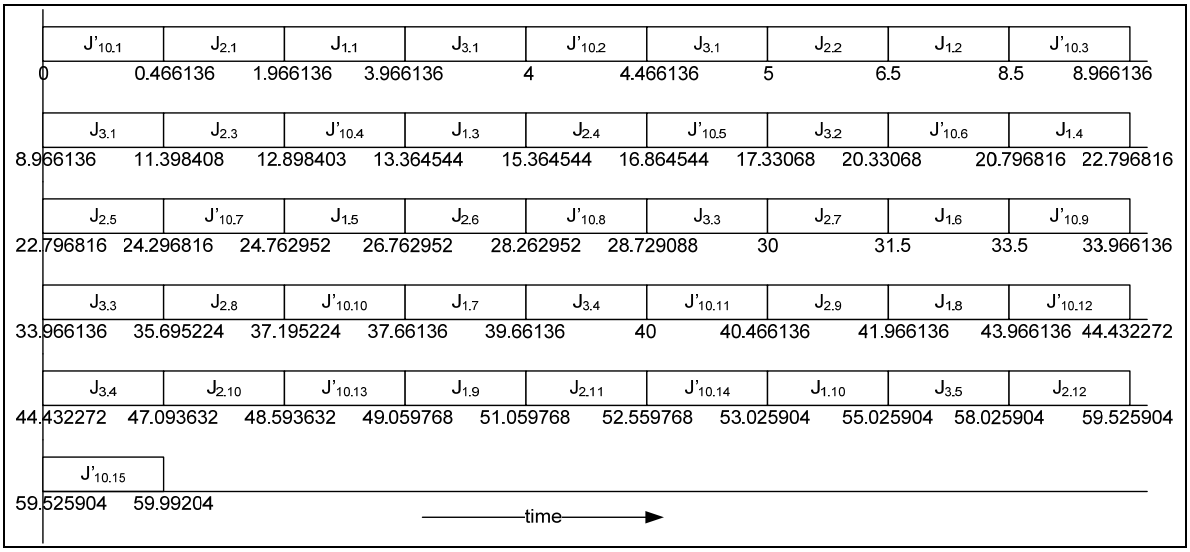

Figure 1. Execution of tasks on core 1 ($S_1 = 2\text{GHz}, \Gamma_1 = \{\tau_1,\ \tau_2,\ \tau_3,\ \tau'_{10}\}$).

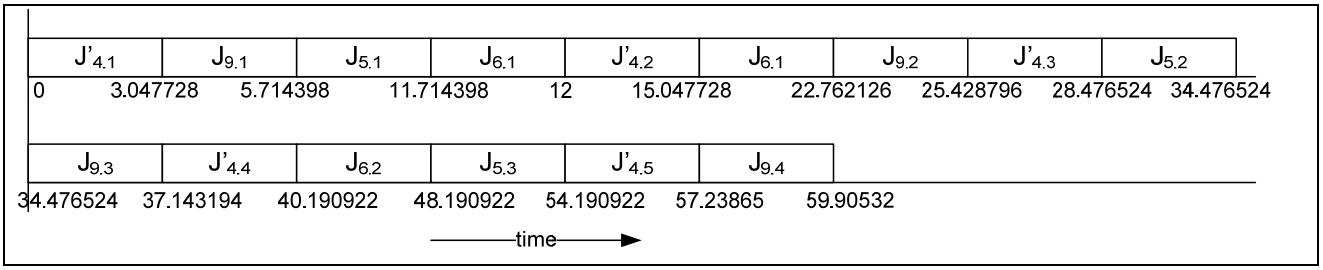

Figure 2. Execution of tasks on Core 2 ($S_2 = 1.5$ GHz, $\Gamma_2 = \{\tau'_4, \tau_5,\ \tau_6\}$).

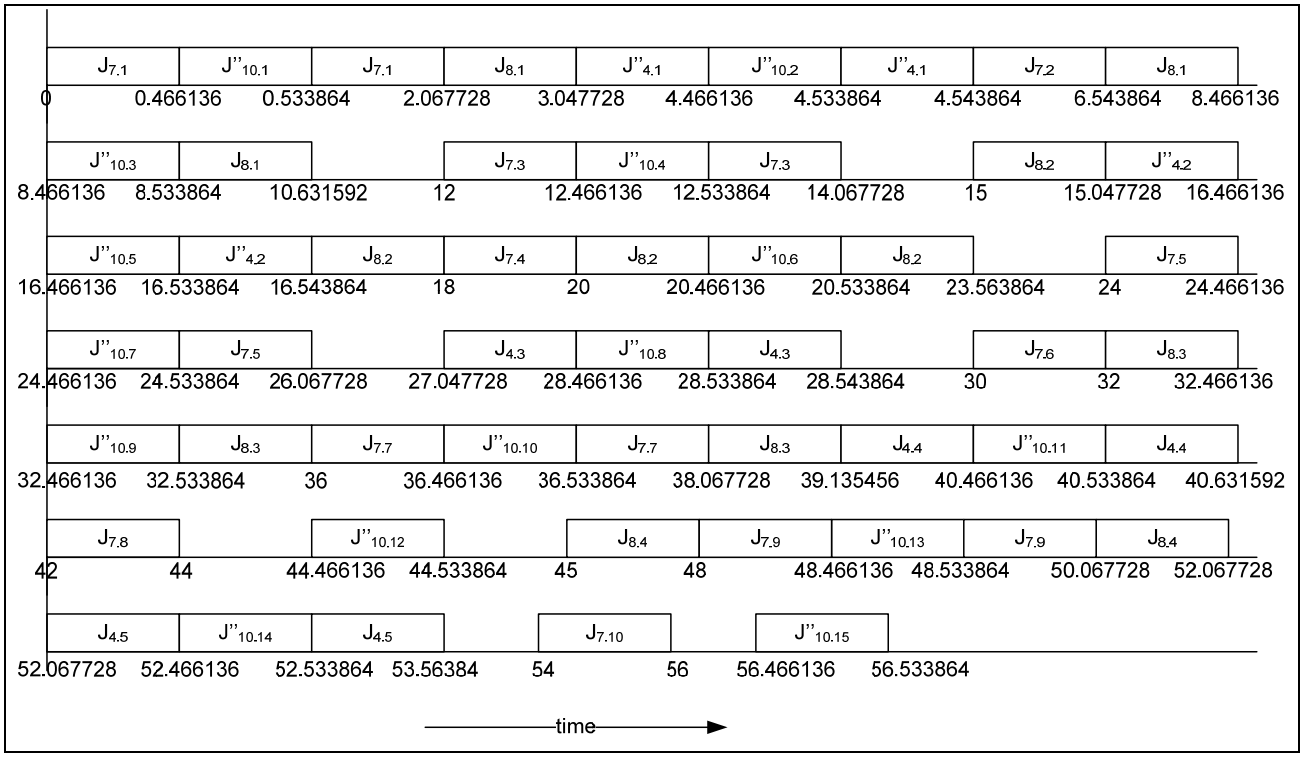

Figure 3. Execution of tasks on Core 3 ($S_3 = 1.0$ GHz, $\Gamma_3 = \{\tau''_4\ , \tau_8,\ \tau''_{10}\}$).

5. Experimental Evaluation

In this section, we have evaluated the effectiveness of *EDFwC=D-TS* algorithm through. We have developed our simulator in Java. Our simulator generates synthetic task-sets using the UUniFast algorithm [40] and then performs required analysis on these task-sets. Detail of experimental set-up and performed analysis is given in subsequent sections.

5.1. Experimental Setup

5.1.1. Task and System Parameters

We have measured the performance of *EDFwC=D-TS* algorithm through simulations on randomly generated synthetic task-sets. We have generated a total of 10^5 task-sets for each experiment. Each task-set contained 8 to 64 tasks, i.e., $|\Gamma| \in [8-64]$. Task parameters are set as follows: $C_i \wedge D_i \wedge P_i \in [10-100]$. Values of task parameter are set in such a way that the total system utilization factor remained between 0.90 and 1.0, i.e., $U \in \{0.85, \ldots, 1.0\}$. To execute these tasks, we have considered a power asymmetric multicore processor. First, we have executed the tasks on 2-core processor and then the experiments are repeated on 4-core, 6-core, and 8-core processors. The task-set and system parameters are summarized in Table 5.

Table 5. Task and system parameters for simulations.

Task Parameters			
Total Task-Sets	**Task-Set Cardinality (n)**	**Task Parameters (C_i, P_i, D_i)**	**System Utilization Factor (U)**
10^5	{8, . . . , 64}	$\{10 \times 10^9, \ldots, 100 \times 10^9\}$, {10, . . . , 100}, {10, . . . , 100}	{0.90, . . . , 1.0}
Processor Specification			
2	{1.01 GHz, 3.1 GHz}		
4	{1.01 GHz,1.5 3 GHz, 2.1 GHz, 3.1 GHz}		
6	{1.01 GHz, 1.53 GHz, 1.81 GHz, 2.1 GHz, 2.7 GHz, 3.1 GHz}		
8	{1.01 GHz, 1.53 GHz, 1.26 GHz, 1.81 GHz, 2.1 GHz, 2.42 GHz,2.7 GHz, 3.1 GHz}		

5.1.2. Comparing Algorithms

The performance of the EDFwC=D-TS algorithm is compared with the following algorithms

- EDF-Partitioned: assigns tasks to a core to its full capacity on First-Fit basis using the EDF utilization bound, i.e., U $\leq$ 1
- EDF-DU-IS-FF: referred to the algorithm presented in [31] that assigns tasks to cores on First-Fit basis using EDF utilization bound assuming that cores are sorted in-order of increasing speed while the tasks are arranged in order of decreasing utilization.

5.1.3. Metrics Used for Comparison

The performance of the above-mentioned algorithms is compared on the basis of following matrices

- Processor Utilization: refers to the ability of an algorithm to utilize the capacity of available processor cores
- No. of cores used: is the number of cores used to feasibly schedule the provided workload
- Schedulability: is the ability of an algorithm to feasibly schedule the workload using the available processing cores

5.2. Simulation Results

This segment presents the experiments, and obtained results, conducted to evaluate the performance of EDFwC=D-TS and its counterparts.

5.2.1. Processor Utilization

In this experiment, we have measured the ability of the aforementioned algorithms to utilize the processor capacity. First, we have evaluated these algorithms on a 2-core processor defined by $S = \{1.01\text{ GHz}, 3.1\text{ GHz}\}$. We have generated 25,000 task-sets as per the above-mentioned parameters. Total system utilization of these task-sets is kept between 0.90 and 1.0, i.e., $U(\Gamma) \in [0.90, \ldots, 1.0]\ \forall \Gamma$ as per S. Each task-set consisted of 4 to 16 tasks. For each task-set, we have determined the schedulability of these task-sets on S using each of the algorithms. If Γ is not found schedulable under certain algorithm then we have used an extra core with processing power of 1.53 GHz to make it schedulable. In the end, we calculated the average of system utilization of the work assigned to each core using the following formula:

$$Processor\ Utilization = \frac{\sum_{i=1}^{n=number\ of\ core\ used} U_i}{n}$$

For the purpose of comparison, we have calculated the average processor utilization under each algorithm. We have repeated the same experiment for 4-core, 6-core, and 8-core processors. The obtained results are shown in Figure 4.

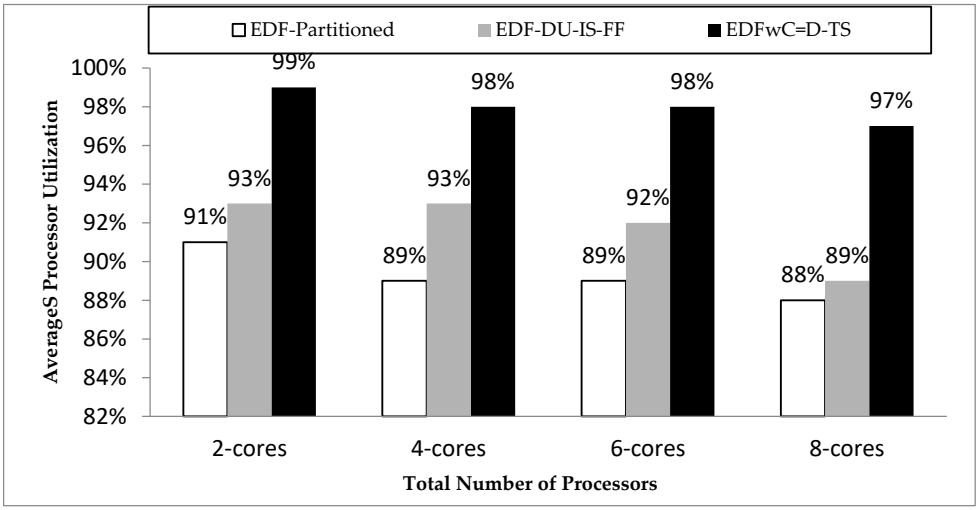

Figure 4. Average processor utilization ($m = 2, 4, 6, 8$; $n \in [4 - 64]$; $U \in [0.90 - 1.0]$).

Figure 4 shows that EDFwC=D-TS algorithm utilized the 99% of the processor capacity on 2-core processor while EDF-DU-IS-FF and EDF-Partitioned algorithms utilized 93% and 91% processor capacity respectively. This shows that EDFwC=D-TS algorithm utilized up to 6% and 8% more processor capacity as compared to EDF-DU-IS-FF and EDF-Partitioned algorithms respectively. Similar dominance of EDFwC=D-TS over its counterparts on 4-core, 6-core, and 8-core processors is obvious where it utilized up-to 9% more processor capacity.

5.2.2. Number of Cores Used

In this experiment, we have compared the EDFwC=D-TS, EDF-DU-IS-FF, and EDF-Partitioned algorithms on the basis of number of cores required by each algorithm to feasibly schedule the given workload. First, we have generated 25,000 task-sets having system utilization 1.0, i.e., $U(\Gamma) = 1.0\ \forall\ \Gamma$ on a 2-core processor defined by $S = \{1.01\text{ GHz}, 3.1\text{ GHz}\}$. Due to the heavy workload, none of the algorithms guarantee to

feasibly schedule all of the task-sets using processing cores provided by S. Therefore, we assume that the S has an extra core having speed of 1.53 GHz that will be used only if the task-set is not schedulable with two cores.

Considering this set-up, we have recorded the number of cores required by each algorithm to feasibly schedule a task-set on S. To make a comparison, we have calculated the average of the cores used by each algorithm using the following formula:

$$Average\ number\ of\ cores\ used = \frac{\sum_{i=1}^{n=Total\ Number\ of\ Task-sets} Number\ of\ cores\ used}{Total\ number\ of\ task-sets}$$

The same experiment is repeated for 4-core, 6-core, and 8-core processors. The obtained results are shown in Figure 5. In Figure 5, the number of processing cores used to determine the workload is given on the x-axis while the average number of cores used by each algorithm is given on the y-axis. An algorithm is considered better if the average number of cores used by it remains close to the total number of cores used to define the workload. For 2-cores, the EDFwC=D-TS used on average 2.10 cores while EDF-DU-IS-FF and EDF-Partitioned algorithms use 2.34 and 2.42 cores respectively. This shows that EDFwC=D-TS uses 14% less cores than EDF-DU-IS-FF while 19% less cores than EDF-Partitioned algorithm. Similarly, it is easy to observe that EDFwC=D-TS outperforms its counterparts on 4-core, 6-core, and 8-core processors and uses up to 13%, 8%, and 9% fewer cores than EDF-DU-IS-FF algorithm respectively while it uses 16%, 9%, and 11% fewer cores than EDF-partitioned algorithm.

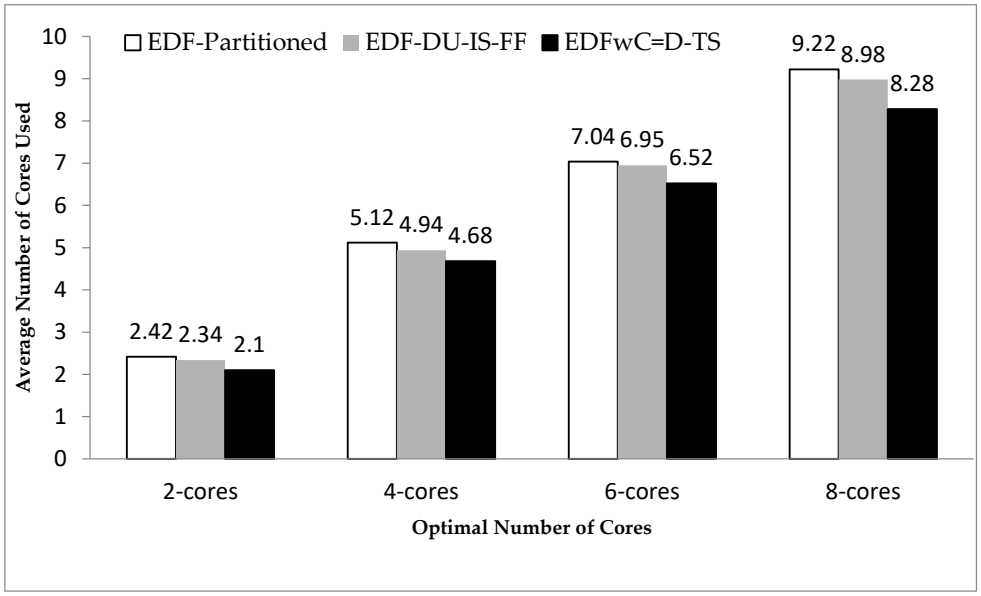

Figure 5. Average number of cores used ($m \in [2, 4, 6, 8]$, $n \in [4 - 64]$ and $U = 1.0$).

5.2.3. Schedulability

Schedulability is major metric used to compare the performance of real-time scheduling algorithms. This is defined as the ability of algorithms to schedule task-sets feasibly. In this experiment, we have compared EDFwC=D-TS, EDF-DU-IS-FF, EDF-Partitioned in schedulability perspective. To begin with, we assess the performance of these algorithms on 4-core power-asymmetric multicore processor defined by the set $S = \{1.01\ GHz, 1.53\ GHz, 2.1\ GHz, 3.1\ GHz\}$. We have generated 10^5 synthetic task-sets. Each task-set contained 16 to 32 tasks. Task parameters were set as per the criteria defined in Table 5. The system utilization of each task-set was kept between 0.90 and 1.0. We have determined the feasibility of each task-set on S and recorded the results. For comparison, we have counted the total

task-sets against each system utilization level and determined the percentage of feasible task-sets under each algorithm. The obtained results are given by Figure 6.

In Figure 6, the x-axis represents the system utilization while the y-axis represents the percentage of feasible task-sets. EDF-partitioned scheduling performed well up to 93% system utilization while its performance reduces at an increasing rate, as the system utilization gets higher. At 94–95% system utilization level it feasibly schedules 78% tasks-sets, while at 96–97%, 98–99%, and 100% utilization levels the success rate drops up to 19%, 0%, and 0% respectively. In contrast, the EDF-DU-IS-FF algorithm schedules 100% task-sets with utilization 93% or less while at higher system utilization levels (94–95%, 96–97%, 98–99%, and 100%) its performance declines gradually and it achieves a success ratio of 86%, 27%, 3%, and 0%. At the same experimental set-up, the superior performance of EDFwC=D-TS algorithm is obvious. It successfully achieves 100% success ratio up-to 95% system utilization. However, at 96–97%, 98–99%, and 100% system utilization levels it schedules 78%, 43%, and 6% task-sets feasibly.

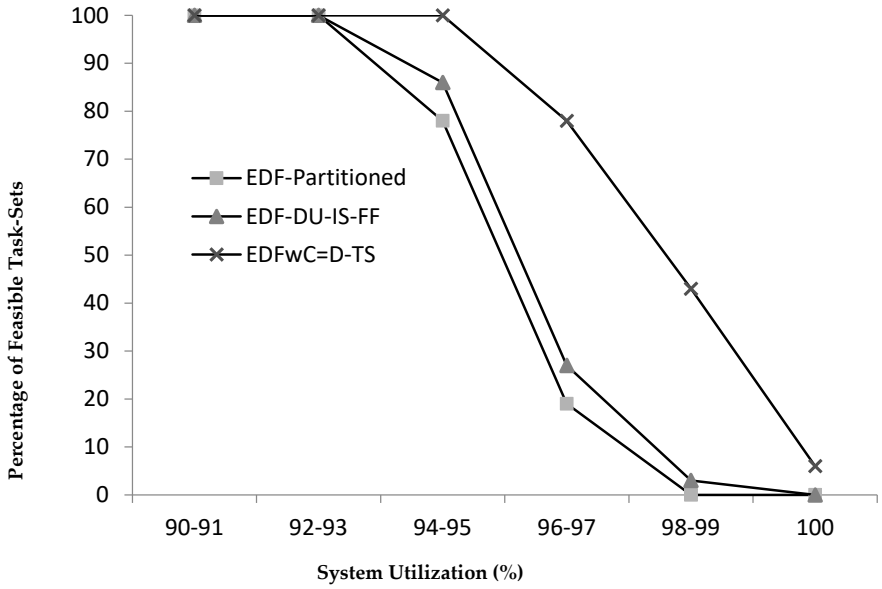

Figure 6. Feasible task-sets ($m = 4$, $n \in [16 - 32]$, $U \in [90\% - 100\%]$).

We repeated the same experiment for 6-core processor. For this experiment, system and workload specifications are set as given in Table 5. The obtained results are shown in Figure 7. It is obvious that all of the algorithms performed well at lower system utilization levels. For higher system utilization workload, EDFwC=D-TS dominates its counterparts. It can be seen that at 94–95% system utilization, it schedules 29% and 33% more task-sets as compared to EDF-DU-IS-FF and EDF-Partitioned algorithms respectively while at 96–97%, 98–99%, and 100% system utilization levels it schedules 51%, 29%, 5% and 57%, 29%, 5% more task-sets than EDF-DU-IS-FF and EDF-Partitioned algorithms respectively.

We further verified the dominance of EDFwC=D-TS by repeating the same experiment on 8-core processor (system and workload configuration is given in Table 5). We generated 10^5 task-sets with $\mid \Gamma \mid \in [32 - 64]$ following the task parameters as given in Table 5. The system utilization for these tasks-sets was kept between 0.90 and 1.0. The obtained results are given in Figure 8. It is easy to observe from Figure 8 that EDFwC=D-TS outclasses its counterparts in terms of schedulability at higher system utilization levels. The performance gain achieved is (38%, 46%, 13%, 2%) and (42%, 50%, 13%, 2%) at (94–95%, 96–97%, 98–99%, 100%) system utilization against EDF-DU-IS-FF and EDF-Partitioned algorithms respectively.

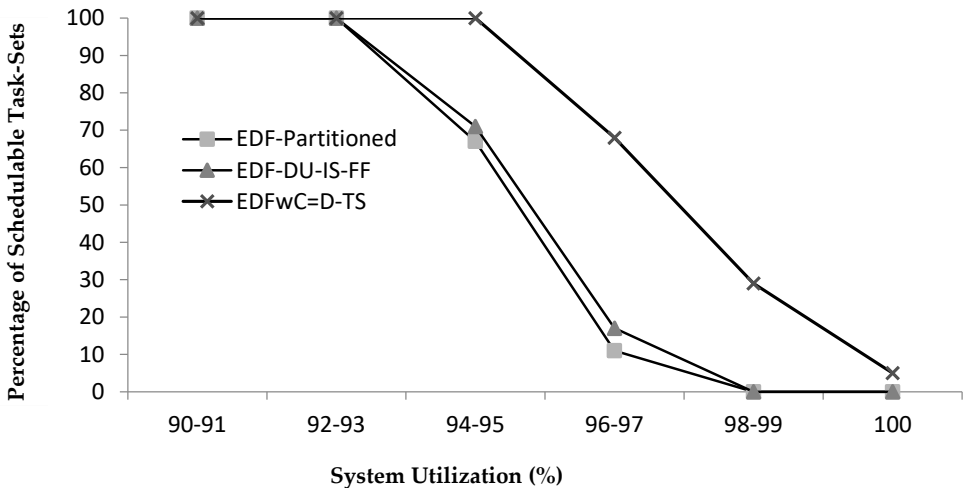

Figure 7. Feasible task-sets ($m = 6$, $n \in [24 - 48]$, $U \in [90\% - 100\%]$).

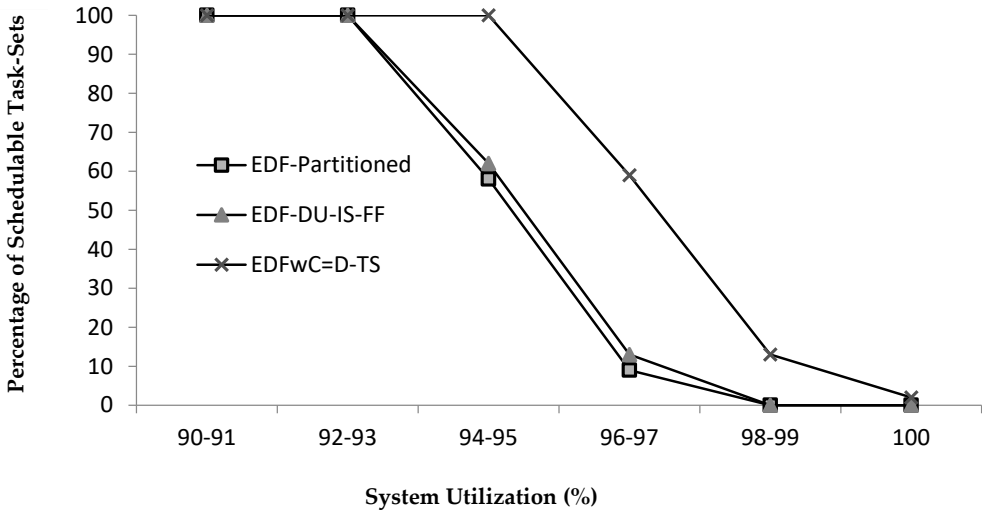

Figure 8. Feasible Task-sets ($m = 8$, $n \in [34 - 64]$, $U \in [56\% - 75\%]$).

We have summarized the results in Tables 6 and 7.

Table 6. Average number of cores used/Average processor utilization.

	EDF-Partitioned				EDF-DU-IS-FF				EDFwC=D-TS			
	2-cores	**4-cores**	**6-cores**	**8-cores**	**2-cores**	**4-cores**	**6-cores**	**8-cores**	**2-cores**	**4-cores**	**6-cores**	**8-cores**
Average No. of Cores Used	2.42	5.12	7.04	9.22	2.34	4.94	6.95	8.98	2.10	4.68	6.52	8.28
Average Processor Utilization	91%	89%	89%	88%	93%	93%	92%	89%	99%	98%	98%	97%

Table 7. Percentage of feasible task-sets.

System Utilization (%)	EDF-Partitioned			EDF-DU-IS-FF			EDFwC=D-TS		
	4-cores	**6-cores**	**8-cores**	**4-cores**	**6-cores**	**8-cores**	**4-cores**	**6-cores**	**8-cores**
90–91	100	100	100	100	100	100	100	100	100
92–93	100	100	100	100	100	100	100	100	100
94–95	78	67	58	86	71	62	100	100	100
96–97	19	11	9	27	17	13	78	68	59
98–99	0	0	0	3	0	0	43	29	13
100	0	0	0	0	0	0	6	5	2

6. Discussion of Experimental Results

In this section, we have discussed and evaluated the performance of EDFwC=D-TS algorithm.

Average Processor Utilization: It is aforementioned that partitioned scheduling usually fails to fully utilize the available processor capacity while semi-partitioned scheduling improves the processor utilization. EDFwC=D-TS algorithm expands the space for choosing the task to split. In this way a task is selected for splitting that maximizes the processor utilization. Furthermore, EDFwC=D-TS allocates the second portion of split task on slowest core where it is schedulable. We have shown that this reduces the overhead associated with existing semi-partitioned scheduling techniques. As a result, EDFwC=D-TS improves the processor utilization. We have conducted experiments to validate the superior performance of EDFwC=D-TS algorithm in terms of processor utilization.

In the first experiment, we have evaluated EDF-Partitioned, EDF-DU-IS-FF, and EDFwC=D-TS algorithms, in terms of their capabilities to utilize the processor capacity. The obtained results are given in Figure 4. It can be seen that EDFwC=D-TS achieves better processor utilization due to its better task assignment strategy as compared to other counterparts. On a 2-core processor, on average it uses 8% and 6% more processor capacity against EDF-Partitioned and EDF-DU-IS-FF algorithms respectively. Similarly, it achieves 10%, 5%, and 10% and 6%, 11%, and 8% better processor utilization on 4-core, 6-core, and 8-core processors than EDF-Partitioned and EDF-DU-IS-FF algorithms. This shows that EDFwC=D-TS dominates its counterparts in terms of processor utilization.

Average Number of Cores Used: Since EDFwC=D-TS algorithm utilizes the processor capacity in a better way as compared to its counterparts, this ability enables EDFwC=D-TS to schedule the given workload using less number of cores. While assigning tasks to mth core using partitioned scheduling if the utilization of remaining tasks is larger than the residual capacity of mth core, i.e., $S_m^{res} < \sum_{i=1}^{|remaining\ tasks|} U_i$ then at least one more core is required to schedule the tasks. Instead, in EDFwC=D-TS if $\sum_{j=1}^{m} S_i^{res} \geq \sum_{i=1}^{|remaining\ tasks|} U_i$ then no other core is required. Due to this reason, EDFwC=D-TS usually requires less number of cores than its counterparts to feasibly schedule the workload.

We have conducted the experiments to verify the ability of EDFwC=D-TS algorithm to schedule the given workload using less number of cores. The obtained results are given in Figure 5. To schedule heavy workload defined using two cores, in most of the cases EDFwC=D-TS successfully schedules the workload with two cores while in few cases it requires another core. As a result it uses 2.10 cores on average. Instead EDF-Partitioned and EDF-DU-IS-FF use 2.42 and 2.34 cores respectively to schedule the same workload. It shows that EDFwC=D-TS uses 13.22% less cores than EDF-Partitioned and 10.25% less cores than EDF-DU-IS-FF. Similarly, for the workload defined using 4-core, 6-core, and 8-core processors EDFwC=D-TS respectively uses 4.45%, 6.19%, and 7.8% less cores than EDF-DU-IS-FF and 8.59%, 7.39%, and 10.19% less cores than EDF-partitioned.

Schedulability: Better utilization of available processor capacity also results in better schedulability. When the given workload is not feasible using partitioned scheduling, i.e., some of the tasks cannot be assigned to any core, EDFwC=D-TS may produce feasible schedule for such workloads if the residual capacity utilized by it, is more than the system utilization of unassigned tasks, i.e., $\sum_{j=1}^{m} S_i^{res} \geq \sum_{i=1}^{|remaining\ tasks|} U_i$. It shows that EDFwC=D-TS is more prone to achieve better schedulability as compared to its counterparts.

We have evaluated EDF-Partitioned, EDF-DU-IS-FF, and EDFwC=D-TS algorithms through simulations to measure their abilities to feasibly schedule the given workload. On 4-core processors EDFwC=D-TS dominates its counterparts for heavy workload and schedules up-to 51% and 67% more task-sets than EDF-DU-IS-FF and EDF-Partitioned algorithms respectively for the workload having 96–97% system utilization. However, for

workload with low system utilization the performance of all algorithms is comparative (See Figure 6 for detailed results). A similar trend is observed when the simulations are performed on 6-core and 8-core processors (results are given in Figures 7 and 8 respectively). EDFwC=D-TS schedules 57% and 50% more task-set than EDF-Partitioned at 96–97% system utilization on 6-core and 8-core processors while this performance gain reaches up to 51% and 46% against EDF-DU-IS-FF.

7. Conclusions and Future Work

This research explores the dynamic-priority semi-partitioned scheduling of power asymmetric multicore processors and presented a novel algorithm: EDFwC=D-TS. EDFwC=D-TS algorithm introduces a two-round task-allocation policy. During task allocation, first a subset of task is assigned to a core and then, in the second round, task-splitting is performed is such a way that core utilization is maximized. The empirical analysis verifies the dominance of the EDFwC=D-TS algorithm over its counterpart. The obtained simulation results reveal that it schedules up to 67% more task-sets at higher system utilization. Furthermore, EDFwC=D-TS improves the processor utilization up to 11% while it also reduces the number of cores required to feasibly schedule the given workload up to 14%.

In this work we have evaluated the EDFwC=D-TS algorithm through simulations. However, in future we aim at validating the effectiveness of EDFwC=D-TS in real environments. Furthermore, we will also integrate the DVFS and memory shut-down approaches with EDFwC=D-TS scheduling to achieve energy efficiency. Additionally, we will also study the efficacy of the proposed work for parallel task models.

Author Contributions: Conceptualization, B.M.; formal analysis, N.A. and M.I.K.; funding acquisition, A.A.; investigation, B.M.; methodology, B.M. and N.A; project administration, M.I.K. and A.A.; software, A.A. and M.I.K.; supervision, M.I.K. and N.A.; validation, N.A.; writing—original draft, B.M.; writing—review and editing, B.M., N.A., M.I.K. and A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been fully funded by the University Malaysia Sabah.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: Authors declare no conflict of interest.

References

1. Selim, Z.; El-Attar, N.E.; Ghoneim, M.E.; Awad, W.A. Performance Analysis of Real-Time Scheduling Algorithms. In Proceedings of the ICICSE '20: 2020 International Conference on Internet Computing for Science and Engineering, Male, Maldives, 14–16 January 2020; pp. 70–75.
2. Zhao, S.; Chang, W.; Wei, R.; Liu, W.; Guan, N.; Burns, A.; Wellings, A. Priority Assignment on Partitioned Multiprocessor Systems with Shared Resources. *IEEE Trans. Comput.* **2020**, *70*, 1006–1018. [CrossRef]
3. Derafshi, D.; Norollah, A.; Khosroanjam, M.; Beitollahi, H. HRHS: A High-Performance Real-Time Hardware Scheduler. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 897–908. [CrossRef]
4. Seo, E.; Park, J.J.; Lee, J. Power Efficient Scheduling of Real-Time Tasks on Multicore Processors. *IEEE Trans. Parallel Distrib. Syst.* **2008**, *19*, 1541–1552.
5. Weisberg, P.; Wiseman, Y. Efficient memory control for avionics and embedded systems. *Int. J. Embed. Syst.* **2013**, *5*, 225–238. [CrossRef]
6. Maiza, C.; Rihani, H.; Rivas, J.M.; Goossens, J.; Altmeyer, S.; Davis, R.I. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Comput. Surv.* **2019**, *52*, 1–38. [CrossRef]
7. Bertout, A.; Goossens, J.; Grolleau, E.; Poczekajlo, X. Template schedule construction for global real-time scheduling on unrelated multiprocessor platforms. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 15 June 2020; pp. 2016–2221.
8. Yu, T.; Petoumenos, P.; Janjic, V.; Leather, H.; Thomson, J. COLAB: A Collaborative Multi-Factor Scheduler for Asymmetric Multicore Processors. In Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization, Seattle, WA, USA, 23–26 September 2019; pp. 268–279.
9. Choudhury, A.M.; Nur, K. Qualitative Study of Contention-aware Scheduling Algorithm for Asymmetric Multicore Processors. In Proceedings of the International Conference on Computing Advancements, Dhaka, Bangladesh, 10–12 January 2020; pp. 1–6.

10. Amina, M.; Kacem, Y.H.; Kerboeuf, M.; Mahfoudhi, A.; Abid, M. A design pattern-based approach for automatic choice of semi-partitioned and global scheduling algorithms. *Inf. Softw. Technol.* **2018**, *97*, 83–98.
11. Davis, R.I.; Burns, A. A survey of hard real-time scheduling for multiprocessor. *ACM Comput. Surv.* **2011**, *43*, 1–44. [CrossRef]
12. Guo, Z.; Yang, K.; Yao, F.; Awad, A. Inter-Task Cache Interference Aware Partitioned Real-Time Scheduling. In Proceedings of the SAC '20: 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March 2020; pp. 218–226.
13. Hassan, H.A.; Salem, S.A.; Mostafa, A.M.; Saad, E.M. Harmonic segment-based semi-partitioning scheduling for multi-core real-time systems. *ACM Trans. Embed. Comput. Syst.* **2016**, *15*, 29.
14. Liu, C.L.; Layland, J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM* **1973**, *20*, 40–61. [CrossRef]
15. Baruah, S.K.; Rosier, L.E.; Howell, R.R. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on One Processor. *J. Real-Time Syst.* **1990**, *4*, 301–324. [CrossRef]
16. Ripoll, I.; Crespo, A.; Mok, A.K. Improvement in feasibility testing for real-time tasks. *J. Real-Time Syst.* **1996**, *11*, 19–39. [CrossRef]
17. Spuri, M. *Analysis of Deadline Schedule Real-Time Systems*; Technical Report 2772; INRIA: Paris, France, 1996.
18. Zhang, F.; Burns, A. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Trans. Comput.* **2008**, *58*, 1250–1258. [CrossRef]
19. Zhang, F.; Burns, A.; Baruah, S.K. Sensitivity analysis of arbitrary deadline real-time systems with EDF scheduling. *J. Real-Time Syst.* **2011**, *47*, 224–252. [CrossRef]
20. Anderson, J.H.; Bud, V.; Devi, C. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In Proceedings of the 17th Euromicro Conference on Real-Time Systems, Balearic Islands, Spain, 6–8 July 2005; pp. 199–208.
21. Andersson, B.; Tovar, E. Multiprocessor scheduling with few preemptions. In Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Sydney, NSW, Australia, 11 September 2006; pp. 322–334.
22. Kato, S.; Yamasaki, N. Real-time scheduling with task splitting on multiprocessors. In Proceedings of the RTCSA '07: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Daegu, Korea, 4 September 2007; pp. 441–450.
23. Kato, S.; Yamasaki, N. Portioned EDF-based scheduling on multiprocessors. In Proceedings of the 8th ACM/IEEE International Conference on Embedded Software, Atlanta, GA, USA, 19–24 October 2008; pp. 139–148.
24. Kato, S.; Yamasaki, N. Semi-partitioning technique for multiprocessor real-time scheduling. In Proceedings of the WIP Session of the 29th Real-Time Systems Symposium (RTSS), Yokohama, Japan, 30 November 2008; p. 4.
25. Kato, S.; Yamasaki, N.; Ishikawa, Y. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In Proceedings of the ECRTS '09: 2009 21st Euromicro Conference on Real-Time Systems, Dublin, Ireland, 1–3 July 2009; pp. 249–258.
26. Burns, A.; Davis, R.I.; Wang, P.; Zhang, F. Partitioned EDF scheduling for multiprocessors using a C = D task splitting scheme. *Real-Time Syst.* **2012**, *48*, 3–33. [CrossRef]
27. Anderson, J.; Erickson, J.; Devi, U.C.; Casses, B. Optimal semi-partitioned scheduling in soft real-time systems. *J. Signal Process. Syst.* **2016**, *84*, 3–23. [CrossRef]
28. Baruah, S.K. Scheduling Periodic Tasks on uniform multiprocessors. *Inf. Process. Lett.* **2001**, *80*, 97–104. [CrossRef]
29. Funk, S.; Goossens, J.; Baruah, S.K. On-line Scheduling on Power asymmetric Multiprocessors. In Proceedings of the Real-Time Systems Symposium (RTSS), London, UK, 2–6 December 2001; pp. 183–192.
30. Baruah, S.K.; Goossens, J. Rate-Monotonic Scheduling on Power asymmetric Multiprocessors. *IEEE Trans. Comput.* **2003**, *52*, 966–970. [CrossRef]
31. Andersson, B.; Tovar, E. Competitive Analysis of Partitioned Scheduling on Uniform Multiprocessors. In Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium, Daegu, Korea, 21–24 August 2007; pp. 1–8. [CrossRef]
32. Andersson, B.; Tovar, E. Competitive Analysis of Static-Priority Partitioned Scheduling on Power asymmetric Multiprocessors. In Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu, Korea, 21–24 August 2007; pp. 111–119.
33. Cucu, L.; Goossens, J. Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Power asymmetric Multiprocessors. In Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation, Prague, Czech Republic, 20–22 September 2006; pp. 397–404.
34. Baruah, S.K.; Goossens, J. The EDF Scheduling of Sporadic Task Systems on Power asymmetric Multiprocessors. In Proceedings of the Real-Time Systems Symposium, Barcelona, Spain, 30 November–3 December 2008; pp. 367–374.
35. Calandrino, J.M.; Baumberger, D.; Li, T.; Hahn, S.; Anderson, J.H. Soft Real-Time Scheduling on Performance Asymmetric Multicore Platforms. In Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07), Bellevue, WA, USA, 3–6 April 2007; pp. 101–112.
36. Chen, S.Y.; Hsueh, C.W. Optimal Dynamic-Priority Real-Time Scheduling Algorithms for Power asymmetric Multiprocessors. In Proceedings of the Real-Time Systems Symposium, Barcelona, Spain, 30 November–3 December 2008; pp. 147–156.
37. Xiaojian, L.; Xiang, L. A-S Algorithm: An Optimal on-line Real-Time Scheduling Algorithm for Power asymmetric Multiprocessors. In Proceedings of the 3rd IEEE International Conference on "Computational Intelligence and Communication Technology" (IEEE-CICT 2017), Ghaziabad, India, 9–10 February 2017.

38. Pathan, R.M.; Jonsson, J. Parameterized Schedulability Analysis on Power asymmetric Multiprocessors. In Proceedings of the 39th International Conference on Parallel Processing, San Diego, CA, USA, 13–16 September 2010; pp. 323–332.
39. Jung, M.J.; Seong, Y.P.; Lee, C.H. Optimal RM Scheduling for Simply Periodic Tasks on Power asymmetric Multiprocessors. In Proceedings of the 2009 International Conference on Hybrid Information Technology (ICHIT '09), Daejeon, Korea, 27–29 August 2009; pp. 383–389.
40. Bini, E.; Buttazzo, G. Measuring the Performance of Schedulability Tests. *Real-Time Syst.* **2005**, *30*, 129–154. [CrossRef]