# EA-HRT: An Energy-Aware scheduler for Heterogeneous Real-Time systems

Sanjay Moulik
IIIT Guwahati
India -781015
e-mail: sanjay@iiitg.ac.in

Rishabh Chaudhary
IIIT Guwahati
India -781015
rishabh@iiitg.ac.in

Zinea Das
IIIT Guwahati
India -781015
zinea@iiitg.ac.in

Arnab Sarkar
IIT Guwahati
India - 781029
arnabsarkar@iitg.ac.in

**Developing energy-efficient schedulers for real-time heterogeneous platforms executing periodic tasks is an onerous as well as a computationally challenging issue. *This research presents a heuristic strategy named, EA-HRT, for DVFS based energy-aware scheduling of a set of periodic tasks executing on a heterogeneous multicore platform*. Initially it calculates the execution demands of every task on each of the different type of cores. Then, it simultaneously allocates each task on available cores and selects operating frequencies for the concerned cores such that the summation of execution demands of all tasks are met as well as there is minimum change in energy consumption for the system. Experimental results show that our proposed strategy is not only able to achieve appreciable energy savings with respect to state-of-the-art (2% to 37% on average) but also enables significant improvement in resource utilization (as high as 57%).**

## I. Introduction

Computing systems with specialized components like multicore CPUs with specialized graphics processing cores, specialized signal processing cores, specialized floating-point units, customizable FPGAs, etc., are called heterogeneous (or unrelated) processing systems [1], [2]. *On such a system, the same program may require different amounts of execution time on different processing cores*. Nowadays, many of the real-time embedded systems like PDAs, laptops, mobiles etc. depend upon battery as their principal source of energy. Hence, these devices are not only judged by their real-time functional performance but also their efficiencies in terms of energy management. A lot of research has been conducted towards energy management of devices based on homogeneous platforms at various levels of abstraction starting from firmware and hardware to architectural, system and even application levels. Typically, two energy management schemes are employed at the operating system level: i) *Dynamic Power Management* (DPM) [3], [4], [5], where certain segments of a system are strategically turned off when the cores are in idle state, and ii) *Dynamic Voltage and Frequency Scaling* (DVFS) [3], [6], which minimizes power consumption by exploiting the relation between the operating voltage/frequency and power consumption, while satisfying both temporal and resource constraints of a set of real-time tasks. Although, the nature of processing elements in these devices have changed from homogeneous to heterogeneous but there is a scarcity of energy-efficient real-time schedulers for heterogeneous platforms. An energy-aware partitioning and scheduling algorithm for standby-sparing systems has been presented in [7]. It is essentially a primary-backup approach for dual-core platform where the primaries and back-ups of tasks are always mapped to distinct cores. Another scheduling technique for standby-sparing systems has been proposed in [8]. They presented a DVFS based energy-aware strategy to schedule fixed-priority real-time tasks. *Although, these strategies have attempted to address the problem of energy-aware scheduling in heterogeneous platforms, they restricted the number of processing core types to only two.*

In recent years, some research works have been focused on generic heterogeneous platforms, having an arbitrary number of core types. Authors in [9] proposed an energy-aware partitioning strategy called the *Least Loss Energy Density* (LLED) algorithm. Although, this strategy is able to use DVFS and DPM mechanisms to reduce energy consumption in the system, mapping of tasks onto cores is computationally expensive and this restricts its practical applicability. Hence, the system suffers from low resource utilization. In [10], authors have proposed two heuristics based on the *Earliest Deadline First* (EDF) algorithm to minimize energy consumption in heterogeneous multicore platforms.

**Our Work**: Although, these works attempt to address the problem of energy-aware scheduling for heterogeneous platforms, they do not allow migration of tasks within the system, often leading to significantly low resource utilizations. However, *efficient resource utilization is a critical design criteria in many embedded systems as it helps in minimizing the number of required resources, thereby reducing design cost.* Thus, with efficient energy management and resource usage as primary objectives, we propose, EA-HRT, a DVFS enabled energy-aware scheduling strategy which offers high resource utilizations in heterogeneous multicore platforms. The salient features of the proposed strategy can be summarized as follows:

- While performing task-to-core allocation, EA-HRT not only considers execution and energy demands of the tasks under consideration but also considers the current operating frequencies of the cores in the system.
- Using an efficient task-to-core allocation strategy, EA-HRT is able to achieve high resource utilization and thus deliver significant performance improvement over a state-of-the-art[11] approach, while incurring only a bounded number of inter-core task migrations.
- By employing an efficient energy saving heuristic, EA-HRT is able to deliver appreciable energy savings compared to the state-of-the-art[11].

## II. Specifications

We have considered a system which comprises of a set of $n$ periodic tasks $T = \{T_1, T_2, ..., T_n\}$ to be scheduled on a



978-1-7281-4123-7/20/$31.00 

| Symbol | Description |
|---|---|
| $T$ | Task set $\{T_1, T_2, \ldots, T_n\}$ |
| $V$ | Heterogeneous multicore platform $\{V_1, V_2, \ldots, V_m\}$ |
| $T_i$ | $i^{th}$ task. |
| $e_i$ | Execution requirement of task $T_i$ |
| $p_i$ | Period of task $T_i$ |
| $u_{i,j}$ | Utilization of task $T_i$ on core $V_j$ |
| $U$ | *Utilization matrix*, $U_{[n \times m]}$ |
| $H$ | *Hyper-period*. It is defined as the least common multiple of all task periods, $H = lcm(p_1, p_2, \ldots, p_n)$ |
| $G_k$ | $k^{th}$ frame in the interval $[0, H]$ |
| $AM$ | *Allocation Matrix* of size $[n \times m]$ for $G_k$ |
| $SM$ | *Schedule Matrix* of size $[n \times m]$ at $G_k$ |
| $sh[i][j]$ | Share required by task $T_i$ at $G_k$ on $V_j$ |
| $F$ | Discrete normalized set of available frequencies |
| $fr$ | Frequency level matrix of size $[m \times 1]$ for cores at $G_k$ |

TABLE I: Important Terminologies

**Algorithm 1:** EA-HRT

**Input:** Task set $T$, Platform $V$, Frequency Set $F$
**Output:** A set of schedule tables ($\forall V_j \in V$ in $[0, H]$)

1 Compute hyperperiod $H = lcm(p_1, p_2, \ldots, p_n)$
2 Using *deadline-partitioning*, compute frame $G_k$
3 Let $G$ be the set of frames $G = \{G_1, G_2, \ldots\}$ and $fr[m \times 1]$ be frequency level matrix for cores
4 **for** ***each*** *frame* $G_k \in G$ **do**
5 Let $AM_{[}n \times m]$ and $SM_{[}n \times m]$ be the allocation matrix and schedule matrix at $G_k$
6 Initialize $AM$ and $SM$ to $\phi$ and $fr$ to 1
7 EA-ALLOCATE ($T$, $V$, $F$, $G_k$, $AM$, $fr$)
8 EA-SCHEDULE ($T$, $V$, $AM$, $SM$)

set of *m* heterogeneous multicores $V = \{V_1, V_2, ..., V_m\}$. All the cores are able to operate on a discrete normalized set of frequencies $F = \{f_1, f_2, ..., f_{max}\}$, such that, $f_{max}$ represents the normalized frequency of 1 and all other frequencies lie between $(\frac{f_1}{f_{max}})$ and 1. We have used the frequency set, which is available in *Nexus 4* with quad-core *Snapdragon S4 Pro processor*, to conduct the experiments. Each occurrence of a periodic task $T_i$ is associated with an *execution requirement* of $e_i$ (at $f_{max}$), *period* $p_i$ and *utilization* $u_{i,j}$ on core $V_j$. We assume implicit task deadlines, i.e. same as its period $p_i$. We have adopted the energy model presented in [12] for our work. The dynamic power consumption $P$ of a DVFS enabled core is directly proportional to the square of the supply voltage $\nu$ and the operating frequency $f$. Again, the supply voltage is linearly proportional to the operating frequency. Therefore, the expression for power consumption may be represented as: $P = c \times f^3$, where, *c* denotes the constant of proportionality.

## III. Proposed Scheduling Strategy

The presented scheduling strategy *EA-HRT* comprises of a two-level hierarchical resource allocation strategy. At the outer level, EA-HRT employs deadline partitioning (similar to *DPFair* [13]) in order to *find a set of frames*, where each frame is the time gap between two consecutive deadlines corresponding to the set of ready tasks. At the inner level, within each frame, EA-HRT *schedules tasks onto available cores* in such a way that each task meets its execution demand and operating frequencies of the cores are scaled appropriately. The proposed scheduling strategy, EA-HRT (Algorithm 1), takes a task set $T$, a heterogeneous platform $V$, and a frequency set $F$ as inputs. It starts by computing set of frames $G$ within hyper-period $H$, according to the *deadline partitioning* scheme. Next, for each frame $G_k \in G$, *EA-HRT* computes the allocation matrix $AM_{[n \times m]}$. In order to do so, the proposed strategy internally uses EA-ALLOCATE (Algorithm 2) and EA-SCHEDULE (Algorithm 4).

### A. EA-ALLOCATE

The pseudo-code for EA-ALLOCATE has been shown in Algorithm 2. It tries to assign each task $T_i \in T$ onto the set of available cores $V$ in such a way that their execution requirements within the given frame $G_k$ are satisfied and operating frequencies of the cores are scaled appropriately. For this purpose, it first computes the share matrix $sh_{[n \times m]}$ for the task set (Line 3). The share $sh[i][j]$ required by each task $T_i \in T$ at $G_k$ on the $V_j^{th}$ processing core, as follows:

$$sh[i][j] = \lceil u_{i,j} \times |TS_k| \rceil \quad (1)$$

Next, it tries to find an unconsidered task $T_i$ whose share value is minimum in the share matrix (at some core) (Line 5). Then, it attempts to find a core $V_j$, where $T_i$ can be fully allocated in the frame at current frequency level $fr[j]$ of $V_j$ (Line 6). Since, there are discrete frequency levels in the system, there may be a scenario when there are empty slots at a core at it's current operating frequency level. Hence, we may be able to allocate $T_i$ using those idle slots. If such a core is found then EA-ALLOCATE simply updates $AM$ with the scaled shared values (Line 8). If such a core is not found then it computes the frequency level for each core, at which $T_i$'s corresponding shares can be allocated (Line 10). Then, it selects the core and the frequency which leads to minimum change in energy consumption for the system (Line 14 and 15) or else it adds $T_i$ to the list of migrating tasks $L_1$ at end (Line 12). This strategy carries on till all the tasks have been considered. Next, it calls ALLOCATE-MIGRATE(3) to allocate the migrating tasks from list $L_1$.

### B. ALLOCATE-MIGRATE

The pseudo-code of ALLOCATE-MIGRATE has been shown in Algorithm 3. It attempts to schedule tasks which require more than one core for their execution. First, it extracts a task $T_i$ from the list $L_1$ and creates a non-decreasing order sorted list $L_2$ (based on shares) to keep track of the normalized unallocated shares of $T_i$ (Lines 2 to 4), where each node in $L_2$ $\langle T_i, j, sh[i][j] \rangle$ stores $T_i$'s share $sh[i][j]$ at core $V_j$. Then, it extracts the first element (say, $\langle T_i, j, sh[i][j] \rangle$) from $L_2$ and computes the unused capacity $uc_j$ of $V_j$ (Line 8) at frequency $f_{max}$. If $uc_j$ is non-zero, then ALLOCATE-MIGRATE computes the unallocated shares of $T_i$ with respect to $V_j$ at $f_{max}$, i.e., $us_i$ (Line 10). While utilizing the unused capacity of $V_j$, there are two possibilities (Lines 11 to 17):

- $us_i > uc_j$: It signifies that unallocated shares of $T_i$ is greater than the unused capacity of $V_j$. Hence, $T_i$ is partially scheduled on $V_j$, the normalized unallocated shares of $T_i$ is updated and frequency level for $V_j$ is set at maximum.

**Algorithm 2:** EA-ALLOCATE

**Input:** $T$, $V$, $F$, $G_k$, $AM$, $fr$
**Output:** Allocation Matrix $AM$ and $fr$ *Frequency Levels* for all tasks

1 Let $sh_{[}n \times m]$ be share matrix at $G_k$
2 Initialize $fr[j] = 1$ for all $V_j$
3 Compute Share Matrix $sh$ for $T$ in $G_k$
4 **for** $i = 1$ *to* $n$ **do**
5  Find unconsidered $T_i$ with minimum share from $sh$
6  Find core $V_j$ where it can be allocated at $fr[j]$ level
7  **if** $V_j$ *is found* **then**
8  Update $AM[i][j] = sh[i][j]$
9  **else**
10  Find $fr[j]$ level for each core $V_j$ at which $T_i$ can be allocated with shares: $sh[i][j]$
11  **if** $T_i$ *cannot be scheduled at any core* **then**
12  Add $T_i$ at end of the migrating task list $L_1$
13  **else**
14  Choose a $V_j$ where it causes minimum energy change
15  Raise $fr[j]$ to appropriate level and set $AM[i][j] = sh[i][j]$
16 ALLOCATE-MIGRATE ($L_1$, $V$, $AM$, $fr$, $F$, $G_k$)
17 **return** $AM$ and $fr$

- $us_i \leq uc_j$: This signifies that unused capacity of core $V_j$ is sufficient enough to meet the unallocated demand of $T_i$. Hence, the task $T_i$ is allocated on $V_j$ and frequency level of $V_j$ is set appropriately. It may be noted that scaling of frequency might lead to parallel execution of a task at multiple cores. Hence, the algorithm verifies that summation of scaled shares of $T_i$ does not exceed 1 across the cores. Since, $T_i$'s allocation is completed, $us_i$ is reset to 0.

In a scenario when a task cannot be allocated across multiple cores (Line 18), EA-HRT declares that the scheduling of $T_i$ on $V$ is infeasible.

### C. EA-SCHEDULE (Algorithm 4)

After successful task allocation in a frame, EA-HRT invokes EA-SCHEDULE (4). The algorithm EA-SCHEDULE carefully schedules each task $T_i$ on it's allocated cores such that $T_i$ is not executed simultaneously on more than one core. In order to do so, it uses following set of guidelines [14]: i. Task are scheduled on free slots starting from left to right in a frame, ii. Always schedule task with highest number of migrations first, iii. A migrating task is scheduled in a stair-case fashion to avoid overlapping and, iv. Schedule of a non-migrating task is broken into multiple chunks when it cannot be scheduled continuously on it's allocated core. These guidelines helps in generation of a feasible schedule for a task sets on heterogeneous platforms.

### D. An Illustrative Example

Let us consider a platform comprising of a set of seven real-time periodic tasks, $T = \{T_1, T_2, ..., T_7\}$, to be scheduled on a heterogeneous multicore system having four cores, $V = \{V_1, \ldots, V_4\}$. The *Utilization Matrix* $U_{[7\times4]}$ is given in Table IIa. The *period* (as well as *deadline*) of each task is: $p_1 = 10$, $p_2 = 20$, $p_3 = 10$, $p_4 = 20$, $p_5 = 40$, $p_6 = 40$, $p_7 = 40$. According to *EA-HRT* (Algorithm 1), we first use *deadline partitioning* to compute the current frame $G_1 = [0, 10)$. Next, EA-HRT calls EA-ALLOCATE to allocate the task set onto the cores. EA-ALLOCATE starts by setting the frequency levels of all cores at minimum i.e. frequency for all cores is set at 0.25 (Refer Table I). The algorithm proceeds by computing the share matrix $sh_{[7\times4]}$ as shown in Table IIb. $T_7$ is the first task to be considered because it has the lowest share value of 1 at $V_3$. Hence, it is allocated to $V_3$ but we

**Algorithm 3:** ALLOCATE-MIGRATE

**Input:** $L_1$, $V$, $AM$, $fr$, $F$, $G_k$
**Output:** $AM$ and $fr$

1 Let $sum_i$ denotes summation of frequency scaled shares allocated for $T_i$ in the system
2 **while** $L_1$ *is not empty* **do**
3  Extract $T_i$ from $L_1$ and set $sum_i = 0$
4  Let $L_2$ be a non-decreasing order sorted list based on $T_i$'s shares on cores
5  Add $\langle T_i, j, sh[i][j]\rangle$ to $L_2$, for $j = 1$ to $m$
6  Let $us_i$ be the *normalized unallocated share* of $T_i$
7  **while** $L_2$ *is not empty* **do**
8  Extract-out the first entry $\langle T_i, j, sh[i][j]\rangle$ from $L_2$
9  Compute *unused capacity* of $V_j$ at $f_{max}$: $uc_j$
10  **if** $uc_j \neq 0$ **then**
11  Compute *unallocated share* of $T_i$ on $V_j$: $us_i$
12  **if** $us_i > uc_j$ **then**
13  Set $fr[j]$ to highest level i.e. $f_{max}$
14  Set $AM[i][j] = uc_j$ and $sum_i += (uc_j/fr[j])$
15  Update the normalized unallocated share of $T_i$: $us_i = (us_i - uc_i)/u_{i,j}$
16  **else if** $sum_i + uc_j - \lceil us_i \rceil \leq |G_k|$ **then**
17  Set $fr[j]$ to maximum level which avoids $sum_i + (uc_j - \lceil us_i \rceil) > |G_k|$
18  Set $AM[i][j] = (uc_j - \lceil us_i \rceil)$
19  **else**
20  Declare infeasible $T$
21 **return** $AM$ and $fr$

**Algorithm 4:** EA-SCHEDULE

**Input:** $T$, $V$, $AM$, $SM$
**Output:** Schedule Matrix $SM$

1 **for** $i = 1$ *to* $n$ **do**
2  From $AM$, select $T_i$ with the highest number of migrations (handle tie-brakes arbitrarily)
3  **for** *each* $V_j$ *(*$j$ = 1 *to* $m$*) and* $AM[i][j] \neq 0$ **do**
4  Schedule $T_i$'s allocated share on $V_j$ according to EA-HRT guidelines
5 **return** $SM$;

| $U_{[7\times4]}$ | $V_1$ | $V_2$ | $V_3$ | $V_4$ |
|---|---|---|---|---|
| $T_1$ | 0.7 | 0.5 | 0.9 | 1.2 |
| $T_2$ | 0.4 | 0.5 | 0.8 | 0.8 |
| $T_3$ | 1.0 | 0.8 | 0.7 | 0.4 |
| $T_4$ | 0.9 | 1.3 | 1.0 | 0.6 |
| $T_5$ | 0.6 | 0.6 | 1.2 | 1.4 |
| $T_6$ | 1.0 | 0.6 | 0.7 | 1.7 |
| $T_7$ | 1.0 | 0.6 | 0.1 | 1.7 |

(a) Utilization Matrix

| $sh_{[7\times4]}$ | $sh_{i,1}$ | $sh_{i,2}$ | $sh_{i,3}$ | $sh_{i,4}$ |
|---|---|---|---|---|
| $T_1$ | 7 | 5 | 9 | 12 |
| $T_2$ | 4 | 5 | 8 | 8 |
| $T_3$ | 10 | 8 | 7 | 4 |
| $T_4$ | 9 | 13 | 10 | 6 |
| $T_5$ | 6 | 6 | 12 | 14 |
| $T_6$ | 10 | 6 | 7 | 17 |
| $T_7$ | 10 | 6 | 1 | 17 |

(b) Share Matrix

| $AM_{[7\times4]}$ | $V_1$ | $V_2$ | $V_3$ | $V_4$ |
|---|---|---|---|---|
| $T_1$ | 0 | 5 | 0 | 0 |
| $T_2$ | 4 | 0 | 0 | 0 |
| $T_3$ | 0 | 0 | 0 | 4 |
| $T_4$ | 0 | 0 | 0 | 6 |
| $T_5$ | 6 | 0 | 0 | 0 |
| $T_6$ | 0 | 5 | 2 | 0 |
| $T_7$ | 0 | 0 | 1 | 0 |

(c) Allocation Matrix

TABLE II: Example 1

don't need to change the frequency for $V_3$ because the current operating frequency of $V_3$ is sufficient to execute share of $V_3$. Energy consumption ($E$) for the system remains at same level of 0.06 (= $4 \times 0.25^3$). Then $T_2$ is considered because it has the lowest share value of 4 at $V_1$ among remaining unconsidered tasks. Hence, $T_2$ is allocated to $V_1$ and $V_1$ is set at frequency 0.4, which is the nearest available frequency (from Table I) sufficient to complete execution of shares of $T_2$. Energy consumption ($E$) for the system becomes 0.109 (=$0.4^3 + 3 \times 0.25^3$). Next, $T_3$ is considered with share value of 4 on $V_4$. The frequency of $V_4$ is set to 0.4 and E becomes 0.158 ($2 \times (0.4^3 + 0.25^3)$). Then, $T_1$ is allocated to $V_2$ with frequency 0.53, $T_4$ to $V_4$ with frequency set at $f_{max}$ i.e. 1 and $T_5$ to $V_1$ with frequency $f_{max}$. Next, we find that $T_6$ cannot be allocated fully at any single core. Hence, it is added to the migrating task list $L_1$. Since, no task is remaining for consideration, so ALLOCATE-MIGRATE is called with $L_1$. ALLOCATE-MIGRATE extracts $T_6$ from $L_1$ and creates a sorted list $L_2$ for $T_6$. Next, it starts iterating on all nodes of $L_2$. It starts by extracting $\langle 6, 2, 6\rangle$ from $L_2$. But, it will find that remaining capacity $uc_2$ for $V_2$ at $f_{max}$ is only 5. Hence, it allocates $T_6$ on $V_2$ for 5 time-slots and set operating frequency of $V_2$ at $f_{max}$. The remaining share of $T_6$ becomes 1 (=6-5). Then, the algorithm extracts next node $\langle 6, 3, 7\rangle$ from $L_2$. The normalized unallocated share $us_6$ of $T_6$ is computed as: $\lceil 1 \times \frac{7}{6} \rceil = 2$. It finds that the current frequency of $V_3$ is not sufficient to execute $T_7$ and remaining share of $T_6$. Hence, $T_6$ is allocated to $V_3$ with raised frequency of 0.4 for 2 time-slots. The allocation matrix $AM$ for $G_1$ has been shown in Table IIc. Next, EA-HRT calls EA-SCHEDULE to schedule the tasks on the cores with respect to $AM$. The final schedule is shown in Figure 1. Therefore, the overall percentage of fractional power saved in the system is: P = $\frac{4-(1.0^3+1.0^3+0.4^3+1.0^3)}{4}$ = 23.4%.

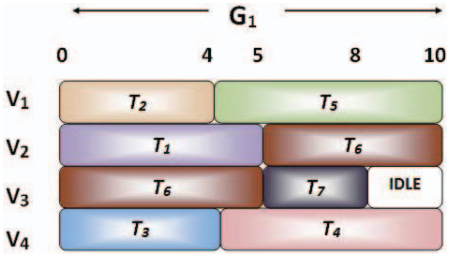



Fig. 1: Final energy-aware schedule for Example

## IV. Experimental Framework and Results

The proposed algorithm *EA-HRT* has been implemented and compared against *MM-M*, a variation of the *Maximum Minimum (MM)* [11] algorithm. MM is an energy-aware task partitioning scheme for periodic tasks executing on a DVFS enabled heterogeneous multicore platform. Before presenting our experimental framework and analysis of results in detail, we provide a brief overview of MM and MM-M algorithms.

**Overview of Maximum Minimum [11] Algorithm:** MM is an energy-aware heuristic task allocation scheme for heterogeneous multicore platforms. In order to perform its operation, the algorithm utilizes a specification named *Energy Density* $ED_{ij}$, which is the consumption rate of dynamic energy consumption rate for $T_i$ at the maximum available operating frequency on core $V_j$. MM is based on a three-phase hierarchical framework. In the initial phase, the algorithm computes the *Maximum Energy Density* $ED_i^{max}$ (=$max_{j=1}^{m}\{ED_{ij}\}$) and the *Minimum Energy Density* $ED_i^{min}$ (=$min_{j=1}^{m}\{ED_{ij}\}$) for each task $T_i$, over all cores and computes the difference $ED_i^{diff}$ (= $ED_i^{max} - ED_i^{min}$). Next, each task $T_i$ added to a list $LT_{ED}$, which is kept sorted in non-increasing order of $ED_i^{diff}$. In the second phase, each task in $LT_{ED}$ (starting with the first) is allocated to its most preferred core such that it's entire execution demand can be satisfied on that core. In the last phase, MM finds a suitable operating frequency for each core based on workloads assigned in the previous phase.

**Modified Maximum Minimum (MM-M) Algorithm**: Once a task has been allocated to a core, the basic MM algorithm does not allow it's migration to other available cores. It has been observed in literature that bin packing schemes which does not allow inter-core task migrations may lead to very poor resource utilizations. Hence, we have used a modified version of MM algorithm called *MM-M*, which embeds the basic MM algorithm over a deadline partitioning framework. At the beginning of each frame, the proportional execution demands for each task within the frame is computed by MM-M. Next, the basic MM algorithm is applied within each frame to perform energy-aware task allocation on heterogeneous cores. The system re-synchronizes globally at the end of every frame. Using such a mechanism, MM-M is able to allow migration of a task at end of all frames. Therefore, MM-M is able to deliver significantly better resource utilizations compared to basic MM. The experimental framework has been presented in the next section.

### A. Experimental Set Up

We ran all our simulations for a total of 100000 time-slots on platforms having 4 processing cores. A term named

*Utilization Factor (UF)* has been introduced to get a measure of resource utilization for a given task set. It is defined as $\frac{\sum_{i=i}^{n} avg_{j=1}^{m}(u_{i,j})}{m}$, where $\sum_{i=i}^{n} avg_{j=1}^{m}(u_{i,j})$ is the total average utilization of the tasks over $m$ cores. Initially, we have randomly generated utilization values for each task in the task sets. Next, we have scaled these randomly generated values in order to create task sets with a specific $UF$. For each set of input parameters, we ran our simulation 50 times and then performed an average of the outputs to achieve the final result. In order to compare the performance of our algorithm against MM-M, we have introduced two metrics namely, *Acceptance Ratio* (*ARat*) and *Normalized Power Consumption* (*NPow*). *ARat* measures ratio of number of task sets that have been successfully scheduled by the algorithms against total number of task sets submitted to them. On the other hand, *NPow* gives a measure of the normalized power consumption in the system. To validate the efficacy of the algorithms over varied scenarios that may be encountered, we have used a set of 14 tasks from Parsec [15] and Mälardalen [16] benchmarks. A detailed discussion on the procedure for measuring execution requirements of each program is presented in [17] and the results have been listed in Table II. These values have been obtained with the help of Gem5 [18] for an Intel Xeon processor, 65 nm CMOS technology, operating at 1.5 GHz. For creating task sets with a specific $UF$, periods of these tasks have been generated appropriately.

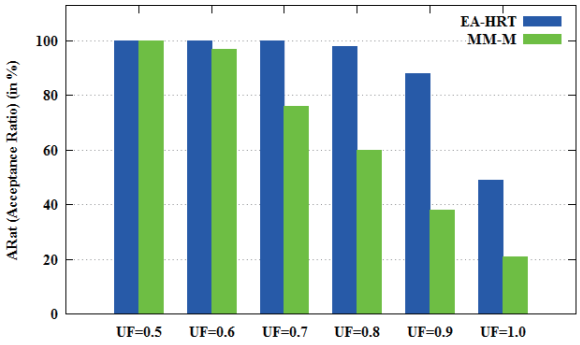
TABLE III: Execution Requirements of programs for Parsec[15] and Mälardalen benchmarks[16]

| Application | Execution Time (in ms) | Application | Execution Time (in ms) |
|---|---|---|---|
| body | 3120 | stream | 11820 |
| can | 12300 | swap | 34500 |
| fluid | 960 | x264 | 60 |
| freq | 1440 | bsort | 9 |
| duff | 73 | edn | 62 |
| lms | 146 | ndes | 8340 |
| qurt | 130 | select | 135 |

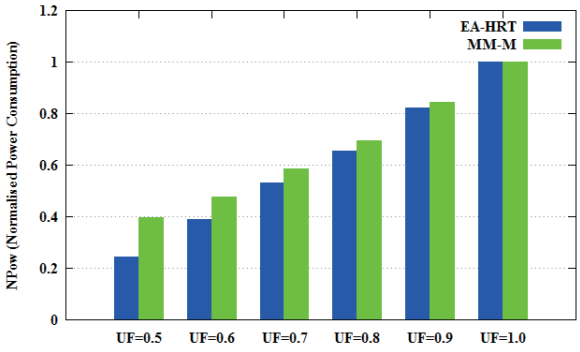
### B. Experimental Results

We have obtained the *ARat* and *NPow* values for different values of utilization factors ($UF$), in order to compare and analyze performance of both the algorithms. Next, we present a detailed analysis of our experimental results.

**1. *Effect on ARat values***: To conduct this experiment, we have varied the utilization factors of the task sets from 0.5 to 1.0. From the Figure 2a, we may observe that both the algorithms exhibit similar performance upto UF value of 0.6. However, EA-HRT is able to outperform MM-M progressively as utilization factor is increased above 0.6. This characteristic may be attributed to non-migration policy of MM-M within frames. When the utilization factors of the task sets are low, the probability that the tasks require migration is less in the system. But as the utilization factor increases for the task sets, this probability becomes progressively higher. As EA-HRT is fully-migrative in nature, it is able to outperform MM-M for higher UF values. And hence, MM-M shows poorer performance compared to EA-HRT. In particular, ARat value reduces from 100% to 21% and 100% to 41% for MM-M and EA-HRT, respectively.




(a) Effect on *ARat* value




(b) Effect on *NPow* value

Fig. 2: Result Comparison with variation in $UF$ ($m$ = 4)

**2. *Effect on NPow values***: In this experiment, we have only considered the task sets which have been successfully scheduled by both the algorithms to compare the NPow values, since ARat value is significantly lower for MM-M with respect to EA-HRT at higher utilizations. From Figure 2b, we may observe that NPow value is directly proportional to the utilization factor of the task sets. This phenomenon may be attributed to the fact that spare capacity in the system decreases with an increase in utilization factor of the task sets, which provides lesser scope to lower operating frequencies to the cores in a system. Hence, power consumption in the system increases with an increase in utilization factor. In order to perform its operation, the algorithm MM-M uses a term called *Energy Density (ED)*, which provides a measure of dynamic energy consumption rate of a task on a core. MM-M allocates a task with highest difference in ED values across all cores first. On the other hand, EA-HRT tries to allocate tasks to cores where the change in energy consumption of the system is minimum. Such a heuristic allows EA-HRT to have lower normalized power consumption in systems than MM-M, when utilization factors for the task sets is low. From Figure 2b, the improvements in energy savings for EA-HRT over MM-M may be observed to be 37.08%, 18.15%, 7.51%, 5.18% and 2.38% for $UF$ values 0.5, 0.6, 0.7, 0.8 and 0.9, respectively

## V. Conclusion

In this work, we have presented, *EA-HRT*, which is a low-overhead scheduler for DVFS enabled heterogeneous mul-

ticore platforms. The proposed scheduler uses a two-phase hierarchical framework to achieve its goal. Initially, it uses deadline partitioning to divide execution of tasks into number of intervals. Next, it uses a heuristic strategy to allocate tasks onto available cores. Finally, it applies DVFS to reduce the energy consumption in the system. Our experimental analysis show that EA-HRT is able to significantly improve acceptance ratios for task sets and energy savings of heterogeneous multicore systems, compared to the state-of-the-art[11].

## REFERENCES


[1] S. Baruah, M. Bertogna, and G. Buttazzo, "Multiprocessor scheduling for real-time systems," 2015.

[2] S. Moulik, R. Devaraj, and A. Sarkar, "COST: A cluster-oriented scheduling technique for heterogeneous multi-cores," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2018, pp. 1951–1957.

[3] N. K. Jha, "Low power system scheduling and synthesis," in *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 259–263.

[4] S. Moulik, "HEALERS: A heterogeneous energy-aware low-overhead real-time scheduler," *IET Computers & Digital Techniques*, June 2019. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/iet-cdt.2019.0023

[5] S. Moulik, A. Sarkar, and H. K. Kapoor, "Dpfair scheduling with slowdown and suspension," in *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, Jan 2018, pp. 43–48.

[6] S. Moulik, R. Devaraj, and A. Sarkar, "HEART: A heterogeneous energy-aware real-time scheduler," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, Jan 2019, pp. 476–481.

[7] Y. wen Zhang, "Energy-aware mixed partitioning scheduling in standby-sparing systems," *Computer Standards and Interfaces*, vol. 61, pp. 129 – 136, 2019.

[8] V. Moghaddas, M. Fazeli, and A. Patooghy, "Reliability-oriented scheduling for static-priority real-time tasks in standby-sparing systems," *Microprocessors and Microsystems*, vol. 45, pp. 208 – 215, 2016.

[9] M. A. Awan and S. M. Petters, "Energy-aware partitioning of tasks onto a heterogeneous multi-core platform," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2013, pp. 205–214.

[10] S. Tosun, "Energy- and reliability-aware task scheduling onto heterogeneous mpsoc architectures," *The Journal of Supercomputing*, vol. 62, no. 1, pp. 265–289, Oct 2012.

[11] M. A. Awan, P. M. Yomsi, G. Nelissen, and S. M. Petters, "Energy-aware task mapping onto heterogeneous platforms using dvfs and sleep states," *Real-Time Systems*, vol. 52, no. 4, pp. 450–485, Jul 2016.

[12] S. Moulik, A. Sarkar, and H. K. Kapoor, "Energy aware frame based fair scheduling," *Sustainable Computing: Informatics and Systems*, vol. 18, pp. 66 – 77, 2018.

[13] S. Funk *et al.*, "Dp-fair: a unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Systems*, vol. 47, no. 5, p. 389, 2011.

[14] S. Moulik, R. Devaraj, and A. Sarkar, "Hetero-sched: A low-overhead heterogeneous multi-core scheduler for real-time periodic tasks," in *2018 IEEE 20th International Conference on High Performance Computing and Communications (HPCC)*, June 2018, pp. 659–666.

[15] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Oct 2008, pp. 72–81.

[16] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[17] S. Bygde, A. Ermedahl, and B. Lisper, "An efficient algorithm for parametric wcet calculation," in *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2009, pp. 13–21.

[18] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.