2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

HEART: A Heterogeneous Energy-Aware Real-Time scheduler

SANJAY MOULIK*, RAJESH DEVARAJ† and ARNAB SARKAR†
*Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati, Assam, India
Email: sanjay@iiitg.ac.in
†Department of Computer Science and Engineering
Indian Institute of Technology Guwahati, Assam, India
Email: {d.rajesh, arnabsarkar}@iitg.ac.in

***Abstract*—Devising energy efficient scheduling strategies for real-time periodic tasks on heterogeneous platforms is a challenging as well as a computationally demanding problem. As a consequence, today we face a scarcity of low-overhead real-time energy aware scheduling techniques which are applicable to heterogeneous platforms. Hence, *this paper proposes a low-overhead heuristic approach called, HEART, for DVFS enabled energy-aware scheduling of a set of periodic tasks executing on a heterogeneous multi-core system*. The proposed approach first applies *deadline partitioning scheme* to obtain a set of distinct time-slices. For each such time-slice, HEART conducts the following *three phase* operation: First, it computes the fragments of the execution demands of all tasks on different processing cores of the platform. Next, it generates a schedule of each task on one or more processing cores such that total execution demands of all tasks are satisfied. Finally, HEART applies DVFS on all processing cores to minimise the energy consumed by the system. Experimental studies show that our scheme is able to significantly improve acceptance ratios for task sets, and energy savings of the platform, compared to the state-of-the-art.**

***Keywords*-Real-time Systems; Multi-cores; Heterogeneous Platforms; Scheduling; Periodic tasks; Heuristic scheme**

I. Introduction

Heterogeneous Platforms [1], [2]: Over the years, the industry is witnessing a significant shift in the nature of processing platforms in real-time embedded systems. For example, a modern System-on-Chip platform contains multi-core processors with specialized digital signal processing cores, graphics processing cores, customizable FPGAs, ASIP, ASIC, etc. Processing Platforms with such varying types of computing elements are called *heterogeneous* (or unrelated) platforms. *On such a platform, the same piece of code may need different amounts of execution time on different processing cores*.

Scheduling on Heterogeneous Platforms: Traditionally, schedulers for real-time tasks on multiprocessors/multi-cores can be broadly classified into following three categories: *non-migrative*, *intra-migrative* and *fully-migrative*. *In non-migrative scheduling, a task is allotted to a specific core and is never allowed to migrate to any other core*. Such an approach has the benefit of converting the multiprocessor scheduling problem to a set of uniprocessor ones. Hence, simple uniprocessor scheduling algorithms such as Earliest Deadline First (EDF), Rate Monotonic (RM) [3] etc. may be used. In such an approach, there is no cost for inter-processor task migrations. However, this scheme suffers from the complexity involved in task partitioning among processors and low resource utilisation. On the other hand, *intra-migrative scheduling allows tasks to migrate among similar processors only*. Once tasks are allocated to processors, the scheduling problem boils down to a group of homogeneous multiprocessor scheduling problems. For scheduling of homogeneous multi-cores, optimal online approaches such as *Pfair* [4], *ERFair* [5], *DPFair* [6] etc. may be used. Compared to a non-migrative scheme, this approach is marked by higher achievable resource utilisations and lower partitioning related overheads. *Fully-migrating approach allows a task to migrate among all of the processors in the platform*, and hence offers better schedulability than its two previous counterparts. According to [1], this strategy can be further extended to optimally schedule periodic tasks also. However, the resulting schedule is likely to incur a very high number of preemptions and inter-processor migrations, which makes this strategy impractical. Recently, Chwa et al. [7] extended the *DPfair* scheduling strategy and proposed *Hetero-Fair*, a global optimal algorithm for the scheduling of periodic tasks on heterogeneous multi-cores with two-types (for example, ARM's big.LITTLE [1]; referred to as *two-type* platform) of processing cores.

Energy-Aware Scheduling: Apart from temporal constraints, energy efficiency has become a primary design constraint in modern real-time computing systems. Typically, two schemes have been employed for energy management purpose. One is *Dynamic Power Management* (DPM), where particular parts of a system are turned off strategically when the processors are in idle state. The other is *Dynamic Voltage and Frequency Scaling* (DVFS), which reduces the power dissipation by exploiting the relation between the power consumption and supply voltage. In this work, we consider *the problem of fully-migrative scheduling of real-time tasks on heterogeneous multi-cores under a DVFS scheme*. The objective is to *minimise energy consumption, while satisfying both resource and timing constraints* of real-time tasks.

Our Work: In recent years, researchers have explored energy-efficient scheduling techniques for multi-cores [1], [8], [9]. However, only a few works have been targeted towards heterogeneous platforms. The *MaxMin* algorithm (proposed by Awan et al. [10]) is lucrative since it does energy aware partitioning of real-time tasks on a heterogeneous multi-core platform. However, being *non-migrative* in nature, this scheme may lead to very poor resource utilisations [11], [12]. In this work, we propose a DVFS based scheduler for heterogeneous platforms called, HEART,

2380-6923/19/$31.00 ©2019 IEEE
DOI 10.1109/VLSID.2019.00100

which provides slightly improved energy-efficiency and significantly better resource utilisations compared to *MaxMin* while incurring only a restricted number of inter processor task migrations. Our *contributions* can be summarized as follows:

- Development of a resource allocation strategy called, *COMPUTE-SCHEDULE*, which conducts task-to-core allocation and schedules the tasks in such a way that task migrations and preemptions are minimized, on a given heterogeneous multi-core platform.
- Given a task execution schedule for each processing core, a DVFS based energy minimisation strategy called, *COMPUTE-EA-SCHEDULE*, has been devised. It attempts to reduce dynamic power consumption by appropriately scaling the operating frequencies of the heterogeneous cores.
- Analysis conducted using extensive simulation based experiments show that our proposed scheme is not only able to significantly improve acceptance ratios of task sets on a given heterogeneous platform but also reduce dynamic energy consumption in the system with respect to the state-of-the-art [10].

II. Specifications

System Model: The system under consideration consists of a set of *n* periodic tasks $T = \{T_1, T_2, ..., T_n\}$ to be scheduled on *m* heterogeneous multi-cores $V = \{V_1, V_2, ..., V_m\}$ which can operate on discrete normalised set of frequencies $F = \{f_1, f_2, ..., f_{max}\}$, such that, f_{max} represents normalised frequency of 1 and all other frequencies lie between 0 and 1. Each instance of a periodic task T_i has an *execution requirement* of $e_{i,j}$ (when executing at f_{max}), *period* p_i, and *utilisation* $u_{i,j} = e_{i,j}/p_i$, on core V_j. We assume task deadlines to be *implicit*, that is, same as its period p_i.

Power Model: In this work, we have adopted the analytical core energy model presented in [8]. The dynamic power consumption P in a system with DVFS capability is directly proportional to the operating frequency f and the square of the supply voltage ν (i.e. $P \propto f\nu^2$). The supply voltage is again linearly proportional to the operating frequency. Hence, the expression for power consumption may be represented as: $P_f = c \times f^3$, where, f represents the operating frequency of the core, P_f represents the power consumption in the system with operating frequency f and c denotes the constant of proportionality.

III. Proposed Scheduling Scheme

The proposed scheduling strategy HEART (Heterogeneous Energy-Aware Real-Time scheduler) is a three-level hierarchical resource allocation mechanism. At the first level, it applies deadline partitioning (similar to *DPFair* [6]) to *compute a set of time-slices*, where a single time-slice is the interval between two consecutive deadlines corresponding to the set of ready tasks. Next, within each time-slice, HEART *schedules tasks onto the processing cores*, such that each task receives its appropriate execution share (when all processing cores are operating at f_{max}). Finally, it tries to *reconfigure the operating frequencies in each core* such that overall energy consumption of the platform is minimized while ensuring that this reconfiguration does not cause overloads in the system.

By following the above steps, we present our proposed scheduling strategy, HEART (Algorithm 1), for the energy aware scheduling of real-time tasks on heterogeneous multi-cores. It takes the task set T, the heterogeneous platform V, and the frequency set F as inputs and computes the schedule for each ready task T_i, over the time-slice. HEART first determines the set of ready tasks, and computes a time-slice according to the *deadline partitioning* (DP) [6] scheme (Lines 1 and 2). Within the time-slice, HEART computes the schedule matrix SM_k. For this purpose, it internally uses COMPUTE-SCHEDULE (Algorithm 2), and COMPUTE-EA-SCHEDULE (Algorithm 5).

Algorithm 1: HEART

Input: Task set T, Platform V, Frequency Set F
Output: Schedule Matrix SM_k at time-slice TS_k
1 Let $\{T_1, T_2, \ldots, T_n\}$ be set of tasks ready for execution
2 Using *deadline-partitioning*, compute time-slice TS_k
3 Let SM_k be the *Schedule Matrix* at TS_k and initialize all its $[n \times m]$ entries to $\emptyset$
4 COMPUTE-SCHEDULE (T, V, TS_k, SM_k)
5 **if** *task set T is feasible on platform V* **then**
6   COMPUTE-EA-SCHEDULE (TS_k, SM_k, F)

A. COMPUTE-SCHEDULE

It attempts to allocate each task $T_i \in T$ onto the heterogeneous platform V such that their execution requirement within the given time slice TS_k is satisfied. It first computes the shares required by each task in TS_k, and inserts them into the list L_1 (Lines 2 to 5). Then, it sorts L_1 in *non-decreasing* order of shares. Subsequently, it invokes SCHEDULE-NON-MIGRATE (Algorithm 3) to assign tasks which can be fully allocated on any one of the available processing core and then adds rest of the tasks in the list L_2. Finally, it invokes SCHEDULE-MIGRATE (Algorithm 4) to assign migrating tasks (in the list L_2).

Algorithm 2: COMPUTE-SCHEDULE

Input: T, V, TS_k, SM_k
Output: Schedule Matrix SM_k
1 Initialize the lists $L_1 = \emptyset$, $L_2 = \emptyset$
  {COMPUTE-SHARES-REQUIRED by tasks at TS_k}
2 **for** $i = 1$ *to* n **do**
3   **for** $j = 1$ *to* m **do**
4         $sh_{i,j,k} = \lceil u_{i,j} \times |TS_k| \rceil$
5         $L_1 = L_1 \cup \{\langle i, j, sh_{i,j,k} \rangle\}$
6 Sort L_1 in *non-decreasing* order of $sh_{i,j,k}$
7 SCHEDULE-NON-MIGRATE (L_1, L_2, SM_k, V)
  SCHEDULE-MIGRATE (L_2, SM_k, V);
8 **return** SM_k;

1) SCHEDULE-NON-MIGRATE (Algorithm 3): It attempts to assign start and end times to tasks which can be fully allocated onto any one of the m cores in the platform. Once a task T_i is scheduled, then all its entries are deleted

Algorithm 3: SCHEDULE-NON-MIGRATE

Input: L_1, L_2, SM_k, V
Output: SM_k (Schedule matrix with non-migrating tasks), L_2 (Sorted list of migrating tasks)

1 **while** *L_1 is not empty* **do**
2   Extract the first element $\langle i, j, sh_{i,j,k} \rangle$ from L_1
3   **if** *τ_i can be allocated fully on V_j for $sh_{i,j,k}$* **then**
4   Assign start and end times of τ_i on V_j, i.e., $SM_k[i][j]$ = ⟨start_time(T_i), end_time(T_i)⟩
5   Delete all entries of T_i from L_1 and L_2
6   **else**
7   Tentatively insert $\langle i, j, sh_{i,j,k} \rangle$ at end of L_2
8 **return** SM_k, L_2

from both the lists L_1 and L_2 (Lines 3 to 5). Suppose T_i cannot be fully scheduled, then all its entries are moved to the list L_2 (Line 7).

2) SCHEDULE-MIGRATE (Algorithm 4): It attempts to schedule tasks which require more than one core for their execution. First, it creates a list L_3 from L_2 to keep track of the normalized unallocated share of T_i (Lines 2 to 4). Then, it extracts the first element (say, $\langle i, j, sh_{i,j,k} \rangle$) from L_3 and computes the unused capacity uc_j of V_j (Line 7). If uc_j is non-zero, then SCHEDULE-MIGRATE computes the unallocated share of T_i with respect to V_j, i.e., us_i (Line 9). While utilizing the unused capacity of V_j, there are two possibilities (Lines 10 to 16):

- $us_i > uc_j$: This implies that the unallocated share of T_i is greater than the unused capacity of core V_j. Hence, T_i is partially scheduled on V_j and the normalized unallocated share of T_i is updated.
- $us_i \leq uc_j$: This implies that the 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. Since, T_i's allocation is completed, us_i is reset to 0 and all entries of T_i is deleted from L_3.

While scheduling a migrating task T_i, HEART attempts to ensure that T_i does not execute on multiple processing cores simultaneously (Line 17). In case of overlapped/parallel execution of T_i on multiple distinct cores, HEART declares that the scheduling of T_i on V is infeasible. Otherwise, it returns the schedule matrix SM_k, consisting of schedules for all ready tasks in the current time slice TS_k.

B. COMPUTE-EA-SCHEDULE

If task allocation is successful, then HEART invokes COMPUTE-EA-SCHEDULE (Algorithm 5). It may be noted that the objective of this phase is to re-assign start and finish times for all tasks on their already allocated processing cores such that operating frequencies of the cores get reduced from their maximum values f_{max}, so that energy consumption of the system can be minimized. COMPUTE-EA-SCHEDULE first finds the spare capacity uc_j of each processing core V_j. Then, it uses uc_j to reconfigure the operating frequency of V_j from f_{max} to f_{opt} (Lines 1 to 5). It may be noted that this decrease in the frequency

Algorithm 4: SCHEDULE-MIGRATE

Input: L_2, SM_k, V
Output: SM_k (*Schedule Matrix* for all tasks)

1 **while** *L_2 is not empty* **do**
2   Create a list L_3 and initialize it to $\emptyset$
3   Extract all entries of T_i from L_2 and move to L_3
4   Let us_i be the *normalized unallocated share* of T_i
5   **while** *L_3 is not empty* **do**
6   Extract-out the first entry $\langle i, j, sh_{i,j,k} \rangle$ from L_3
7   Compute *unused capacity* of V_j: uc_j
8   **if** $uc_j \neq 0$ **then**
9   Compute *unallocated share* of τ_i on V_j: us_i
10   **if** $us_i > uc_j$ **then**
11   Update $SM_k[i][j]$ to schedule τ_i on V_j for the duration uc_j
12   Update the normalized unallocated share of T_i: $us_i = (us_i - uc_i)/u_{i,j}$
13   **else**
14   Update $SM_k[i][j]$ to schedule τ_i on V_j for the duration $uc_j - \lceil us_i \rceil$
15   Reset us_i to 0
16   Delete all entries of T_i from L_3
17   **if** *($us_i \neq 0$) or (T_i executes on multiple processing cores simultaneously)* **then**
18   Declare the scheduling of T on V is infeasible
19 **return** SM_k

leads to the increase in execution windows of the tasks. Such an increase in execution windows, in turn, leads to the possibility of parallel execution of tasks on multiple cores. Therefore, Algorithm 5 re-schedules tasks to avoid parallel execution (Lines 6 to 9). Finally, it returns SM_k which contains the energy-aware schedule for the current time-slice.

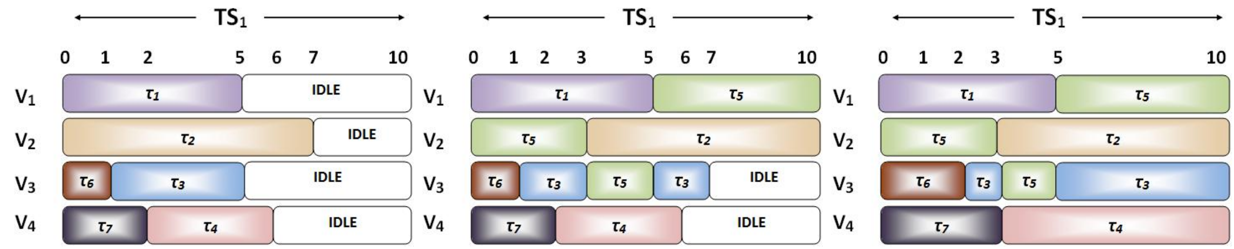
Algorithm 5: COMPUTE-EA-SCHEDULE

Input: TS_k, SM_k, Frequency Set F
Output: Schedule Matrix SM_k at time-slice TS_k

1 **for** *each processor $V_j \in V$* **do**
2   Compute unused capacity of V_j: uc_j
3   **if** $uc_j \neq 0$ **then**
4   Compute the frequency $f_{opt} \in F$ for V_j
5   Update SM_k for all tasks scheduled on V_j
6 **for** *each migrating task $T_i \in T$* **do**
7   **if** *there exists a overlap in their execution* **then**
8   Re-schedule T_i to avoid parallel execution
9   Allocate the unused time to non-migrating tasks
10 **return** SM_k

C. An Illustrative Example

Let us consider a system consisting of a set of seven real-time periodic tasks, $T = \{T_1, T_2, ..., T_7\}$, to be scheduled



(a) Schedule for migrating tasks (b) Schedule for all tasks (c) Final energy-aware schedule for all tasks

Figure 1: An example to illustrate our proposed algorithm HEART.

on a heterogeneous multi-core platform consisting of four cores, $V = \{V_1, \ldots, V_4\}$. The *Utilisation Matrix* $U_{[7\times4]}$ is:

$U_{[7\times4]}$	V_1	V_2	V_3	V_4
T_1	0.5	1.3	0.8	1.2
T_2	0.8	0.7	1.0	1.1
T_3	1.1	0.8	0.4	0.9
T_4	1.2	0.9	0.8	0.4
T_5	0.9	1.1	1.1	1.4
T_6	0.9	0.6	0.1	0.8
T_7	1.2	0.4	0.7	0.2

The *period* (as well as *deadline*) of each task is: $p_1 = 10$, $p_2 = 20$, $p_3 = 10$, $p_4 = 20$, $p_5 = 40$, $p_6 = 40$ and $p_7 = 40$. According to ***HEART*** (Algorithm 1), we first use *deadline partitioning* to compute the current time-slice. Since, all tasks are ready for execution, the first time-slice TS_1 = $[0, 10)$. In order to compute the schedule matrix SM_1 at TS_1, HEART calls Algorithm 2. First, it creates the lists L_1 and L_2 and initializes them to $\emptyset$. Then, it **computes the required shares** for each task $T_i \in T$ at TS_1. The computed shares are as follows:

$sh_{[7\times4]}$	$sh_{i,1,1}$	$sh_{i,2,1}$	$sh_{i,3,1}$	$sh_{i,4,1}$
T_1	5	13	8	12
T_2	8	7	10	11
T_3	11	8	4	9
T_4	12	9	8	4
T_5	9	11	11	14
T_6	9	6	1	8
T_7	12	4	7	2

Now, Algorithm 2 sorts L_1 based on computed shares in *non-decreasing* order, i.e., The sorted list L_1 at time-slice TS_1 is ($\langle i, j, sh_{i,j,1}\rangle$): $\{\langle 6, 3, 1\rangle$, $\langle 7, 4, 2\rangle$, $\langle 3, 3, 4\rangle$, $\langle 4, 4, 4\rangle$, $\langle 7, 2, 4\rangle$, $\langle 1, 1, 5\rangle$, $\langle 6, 2, 6\rangle$, $\langle 2, 2, 7\rangle$, $\langle 7, 3, 7\rangle$, $\langle 1, 3, 8\rangle$, $\langle 2, 1, 8\rangle$, $\langle 3, 2, 8\rangle$, $\langle 4, 3, 8\rangle$, $\langle 6, 4, 8\rangle$, $\langle 3, 4, 9\rangle$, $\langle 4, 2, 9\rangle$, $\langle 5, 1, 9\rangle$, $\langle 6, 1, 9\rangle$, $\langle 2, 3, 10\rangle\}$, $\langle 2, 4, 11\rangle$, $\langle 3, 1, 11\rangle$, $\langle 5, 2, 11\rangle$, $\langle 5, 3, 11\rangle$, $\langle 1, 4, 12\rangle$, $\langle 4, 1, 12\rangle$, $\langle 7, 1, 12\rangle$, $\langle 1, 2, 13\rangle\}$. Then, Algorithm 2 computes the schedule for non-migrating tasks.

Scheduling Non-migrating Tasks: Algorithm 3 extracts the first element from the list L_1, i.e., $\langle 6, 3, 1\rangle$ (task T_6 requires a share of 1 unit on V_3). The task T_6 can be fully allocated on V_3 and hence, the schedule matrix $SM_1[6][3]$ is updated as $\langle 0, 1\rangle$. Since, T_6 has been completely scheduled on V_3, all entries of T_6, i.e., $\langle 6, 1, 9\rangle$, $\langle 6, 2, 6\rangle$ and $\langle 6, 4, 8\rangle$ have been removed from L_1. Now, the above process is repeated by extracting the first element from L_1, i.e., $\langle 7, 4, 2\rangle$. Since, V_4 can fully accommodate T_7, it has been scheduled on V_4 with $SM_1[7][4]$ = $\langle 0, 2\rangle$. Similarly, T_3 is scheduled (according to $\langle 3, 3, 4\rangle$) on V_3, T_4 is scheduled (according to $\langle 4, 4, 4\rangle$) on V_4, T_1 is scheduled (according to $\langle 1, 1, 5\rangle$) on V_1 and T_2 is scheduled (according to $\langle 2, 2, 7\rangle$) on V_2.

Now, $\langle 5, 1, 9\rangle$ becomes the first element of the list L_1. According to this, if T_5 is allocated on V_1, it requires a share of 9 units. However, V_1 does not have the capacity to accommodate T_1 and hence, the element $\langle 5, 1, 9\rangle$ has been inserted into L_2. Similarly, the other three elements corresponding to the task T_5 has been moved to L_2, i.e., $\langle 5, 2, 11\rangle$, $\langle 5, 3, 11\rangle$ and $\langle 5, 4, 14\rangle$. Since, the list L_1 becomes empty, the execution moves from Algorithm 3 to Algorithm 4. The schedule matrix $SM_{1[7\times4]}$ for non-migrating tasks at TS_1, is as follows:

$SM_{1[7\times4]}$	V_1	V_2	V_3	V_4
T_1	$\langle 0, 5\rangle$	0	0	0
T_2	0	$\langle 0, 7\rangle$	0	0
T_3	0	0	$\langle 0, 4\rangle$	0
T_4	0	0	0	$\langle 2, 6\rangle$
T_5	0	0	0	0
T_6	0	0	$\langle 0, 1\rangle$	0
T_7	0	0	0	$\langle 0, 2\rangle$

Scheduling of Migrating Tasks: It may be observed that T_5 could not be allocated in the earlier phase and therefore it must be partitioned into multiple chunks and scheduled on more than one core using the SCHEDULE-MIGRATE algorithm (Algorithm 4). It first moves all entries corresponding to T_5 from list L_2 to L_3. Then, it extracts the first element from L_3, i.e., $\langle 5, 1, 9\rangle$, and computes the unused capacity of V_1. Since, there is a residual spare capacity of 5 on V_1, T_5 has been partially allocated on V_1, $SM_1[5][1] = \langle 5, 10\rangle$. The unallocated share of T_5 becomes $9 - 5 = 4$. This has been normalized as: $us_5 = 4/0.9$ = 4.4 (approx).

Now, Algorithm 4 extracts the next element $\langle 5, 2, 11\rangle$ from L_3 and checks the spare capacity of V_2. Since, the spare capacity of V_2 (i.e. 3), is not sufficient enough to accommodate the unallocated share of τ_5 (i.e., $4.4 * 1.1$ = 4.8 (approx.)), it has been partially allocated on V_2. That is, $SM_1[5][2] = \langle 0, 3\rangle$. The unallocated share of T_5 is $4.8 - 3$ = 1.8. This has been normalized as: us_5 = 1.8/1.1 = 1.6 (approx). It may be noted that T_2 has already been scheduled at V_2 from 0 to 7. To avoid the overlap with T_5, the schedule of T_2 is updated as: $SM_1[2][2] = \langle 3, 10\rangle$.

Next, Algorithm 4 extracts the element $\langle 5, 3, 11\rangle$ from L_3 and checks the spare capacity of V_3. Since, the spare capacity of V_3 (i.e. 5) is sufficient enough to accommodate the unallocated share of T_5 (i.e., $1.6 * 1.1$ = 1.76), it has been allocated on V_3. That is, $SM_1[5][3] = \langle 3, 5\rangle$. The tasks that are already scheduled on V_3 are then adjusted to avoid overlaps. The gantt chart representation of the schedule matrix $SM_{1[7\times4]}$ (including migrating tasks) for time-slice TS_1, is depicted in Figure 1b.

Energy-aware scheduling: As we can observe from the schedule matrix SM_k (in Figure 1a), V_1 and V_2 do not have any spare capacity in the current time-slice. Hence, COMPUTE-EA-SCHEDULE allows V_1 and V_2 at run at their highest available frequency and we are not able to reduce any power consumption at these cores. On the other hand, V_3 has a spare capacity of 3 time slots. This can be utilised to lower the operating frequency of V_3. However, the execution window of T_5 will overlap with its own execution on other cores, if the operating frequency is decreased during its execution. Hence, the spare capacity of 3 units is shared only among the non-migrating tasks T_3 and T_6 which together require 5 time units at frequency f_{max}. Therefore, the operating frequency of V_3 during the execution of T_3 and T_6 can be reduced by at most 37.5% (= $1-((4+1)/8)\times 100$ = $(1-0.625)\times 100$). The percentage fractional power saved in V_3 is: $\frac{[10-(0.625^3\times 8+1^3\times 2)]}{10}\times 100$ = 60.48%. The modified execution shares of T_3 and T_6 becomes 6 $(= 4/0.625)$ and 2 $(= 1/0.625)$, respectively.

In core V_4, the required frequency to execute the tasks T_7 and T_4 is $\frac{4+2}{10} = 0.6$. Hence, the savings in V_4 is: $\frac{[10-(0.6^3\times 10)]}{10}\times 100$ = 78.4%. Therefore, the overall percentage of fractional power saved in a system is: P = (60.48+78.4) / 4 = 34.72%. The modified share of $T_7 = 2/0.6 = 3$. Similarly, modified share of $T_4 = 4/0.6 = 7$. The final schedule for the given task set for TS_1 is shown in Figure 1c. Similarly, the schedule for subsequent time-slices can be computed.

IV. Experimental Set Up and Results

We have implemented the *HEART* algorithm and compared the same against *MaxMin-M*, a variation of the *MaxMin* [10] algorithm. MaxMin is a DVFS based energy aware task partitioning scheme for periodic tasks executing on a heterogeneous multi-core platform. The experimental framework used in our work is discussed next.

Table I: Available Frequency

Frequency (in MHz)	**Frequency** (Normalised)	**Frequency (in MHz)**	**Frequency** (Normalised)
384	0.25	1026	0.67
486	0.32	1134	0.75
594	0.39	1242	0.82
702	0.46	1350	0.89
810	0.53	1458	0.96
918	0.60	1512	1.0

Experimental Set Up: The performance of HEART has been evaluated and compared through a set of carefully chosen simulation based experiments. Each task set considered consists of 30 randomly generated hypothetical periodic tasks, whose execution requirements are generated from a Normal Distribution having standard deviation $\sigma_e = 10$ and mean $\mu_e = 50$. Entries in the utilisation matrix $(U_{[n\times m]})$ are also obtained from a Normal Distribution with standard deviation of $\sigma_u = 0.2$ and mean $\mu_u = 0.4$. In our experiments, we have used a parameter called *Utilisation Factor (UF)*, in order to obtain a measure of resource utilisation corresponding to a given task set. *UF* 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 summation of the average utilisation of the tasks over the m processing cores. For creating task sets with a specific UF, the randomly generated utilisation values have been scaled appropriately. Results are generated for various distinct utilisation factors. We ran all our simulations for a total execution time of 100000 time slots on systems having 4 processing cores. For each set of input parameters, we ran the simulation on 50 different test cases. We have used the frequency set (Table IV) available in *Nexus 4* with quad-core *Snapdragon S4 Pro processor*, to carry out the experiments. The following two metrics have been used to compare the performance of our proposed algorithm against that of MaxMin-M. The first metric, *ARat*, defines *acceptance ratio* as the number of task sets successfully scheduled by the system against the total number of task sets submitted to it. The second metric is *NPow*, which represents the *normalised power consumption* in the system. Before presenting the detailed experimental results, we now provide an overview of the MaxMin algorithm, a state-of-the-art task assignment strategy against which our proposed scheme HEART has been compared.

Overview of MaxMin [10]: MaxMin is a heuristic energy aware task allocating strategy targeted for heterogeneous processing platforms. It uses a parameter called *Energy Density* ED_{ij}, which is defined as the dynamic energy consumption rate of the i^{th} task at the highest operating frequency f_{max}, of processing core j. The algorithm MaxMin works in three phases. In the first phase, it finds 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 τ_i, over all processing cores and calculates the difference ED_i^{diff} (= $ED_i^{max} - ED_i^{min}$). Each task τ_i is then inserted into a list L_{ED}, which is sorted in non-increasing order of ED_i^{diff}. In the second phase, each task in L_{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, MaxMin finds a suitable operating frequency for each core based on workloads assigned in the previous phase.

Modified MaxMin (MaxMin-M) Algorithm: The basic MaxMin algorithm does not allow the execution of a single task to be allocated to multiple cores, where each core satisfies a distinct fraction of its total execution demand. The literature on bin packing strategies [11], [12] show that such a mechanism which does not allow inter-processor task migrations may lead to very poor resource utilisations. Hence, we have used a modified version of MaxMin called *MaxMin-M*, which embeds the strategy over a deadline partitioning framework. The MaxMin-M algorithm divides time into slices demarcated by the arrivals/departures of all tasks in the system (also known as *Deadline Partitioning* (DP)). At the beginning of each time-slice, MaxMin-M determines the proportional execution share for each task within the time-slice. Inside each time-slice, the basic MaxMin algorithm is applied for energy aware task allocation on heterogeneous cores. The system re-synchronises globally at the end of every time-slice. Such a strategy, enables MaxMin-M to allow migration at the end of every time-slice boundary

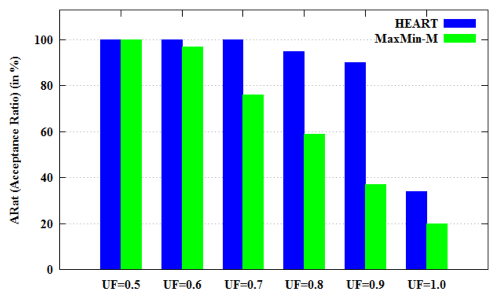


Figure 2: Effect on Acceptance Ratio (ARat)

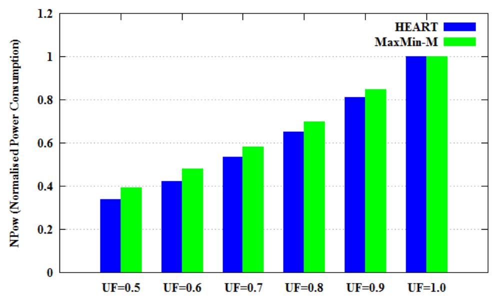


Figure 3: Effect on Normalised Power Consumption (NPow)

and hence deliver significantly better resource utilisations compared to basic MaxMin.

A. Experimental Results

We have conducted extensive simulation based experiments to evaluate the proposed scheduling scheme. To compare and analyse our results with that of MaxMin-M, we have measured *ARat* and *NPow* for different values of utilisation factors (UF). Detailed analysis of our experimental results is discussed below.

Experiment 1- ***Effect on Acceptance Ratio (ARat)***: In this experiment, we have varied UF between 0.5 and 1.0. We may observe from Figure 2 that at lower UF upto about 0.6, both MaxMin-M and HEART exhibits similar performance efficiencies. However, HEART progressively outperforms MaxMin-M as UF increases beyond 0.6. This phenomenon may be attributed to the non-migrative execution policy of MaxMin-M within time-slices. At lower UF values (below 0.6), the probability that MaxMin-M can successfully allocate entire execution shares of every task onto a single processing core, is very high. However, at higher UF values, the possibility of allocating every task onto cores without causing any migration becomes lower. And hence, MaxMin-M shows poorer performance compared to HEART. In particular, ARat reduces from 100% to 20% and 100% to 25% for MaxMin-M and HEART, respectively.

Experiment 2- ***Effect on Normalised Power Consumption (NPow))***: As the value of ARat for MaxMin-M is significantly lower than that of HEART, only those task sets which have been successfully completed by both the algorithms have been considered in this experiment. We may observe from Figure 3, that NPow is directly proportional to the UF of the system. This is because residual capacity in the system decreases with an increase in UF of the task set, thereby reducing the scope for lowering core operating frequencies. This leads to higher power consumption with an increase in UF. As discussed earlier, MaxMin-M sequentially allocates tasks to their most favoured processing cores in the order of their ED_{diff} values. On the other hand, HEART tries to directly allocate tasks to their most preferred processing cores in the order of decreasing energy affinities. This allows HEART to perform slightly better than MaxMin-M in most scenarios, where the system has some spare capacity so that most tasks get assigned to cores where their energy efficiencies are comparatively high. In particular, improvements in energy savings for HEART over MaxMin may be observed to be 14.10%, 12.08%, 8.18%, 7.04% and 4.25% for UF values 0.5, 0.6, 0.7, 0.8 and 0.9, respectively.

V. Conclusion

In this paper, we have proposed a low-overhead heuristic strategy called, *HEART*, for the energy-aware scheduling of a set of periodic tasks on a heterogeneous multi-core platform. Experimental studies show that our proposed scheduling scheme is able to significantly improve acceptance ratios for task sets and energy savings of the platform, compared to the state-of-the-art [10].

References

[1] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2015.

[2] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[3] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, vol. 24, 2011.

[4] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[5] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *12th Euromicro Conference on Real-Time Systems*. IEEE, pp. 35–43, 2000.

[6] 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.

[7] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Optimal real-time scheduling on two-type heterogeneous multicore platforms," in *Real-Time Systems Symposium*, pp. 119–129, 2015.

[8] 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.

[9] ——, "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.

[10] 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.

[11] S. Baruah and J. Carpenter, "Multiprocessor fixed-priority scheduling with restricted interprocessor migrations," in *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, July 2003, pp. 195–202.

[12] "Bin packing and machine scheduling," *American Mathematical Society*. [Online]. Available: http://www.ams.org/publicoutreach/feature-column/fcarc-packings1