ETA-HP: An Energy and Temperature-Aware Real-time Scheduler for Heterogeneous Platforms

Yanshul Sharma[1], Shounak Chakraborty[2] and Sanjay Moulik[3*]

[1*]Department of Computer Science and Engineering, Indian Institute of Information Technology (IIIT), Guwahati, India.
[2]Department of Computer Science (IDI), Norwegian University of Science and Technology (NTNU), Trondheim, Norway.
[3]Department of Computer Science and Engineering, Indian Institute of Information Technology (IIIT), Guwahati, India.

*Corresponding author(s). E-mail(s): sanjay@iiitg.ac.in;
Contributing authors: yanshul.sharma@iiitg.ac.in;
shounak.chakraborty@ntnu.no;

Abstract

Modern real-time systems are based on heterogeneous multicore platforms, which help them productively meet the applications' diverse and high computational requirements. Managing the energy and temperature of these computational platforms has become a topic of inconceivable enthusiasm for researchers and specialists over recent years. This paper presents a heuristic technique, named ETA-HP, for energy and temperature efficient scheduling of a set of real-time periodic tasks on a DVFS empowered heterogeneous multicore system. The proposed strategy operates in four stages, namely *Deadline Partitioning*, *Task-to-Core Allocation*, *Temperature-Aware Scheduling*, and *Energy-Aware Scheduling*. Our empirical analysis shows that with a variation in system workload from **50%** to **100%**, ETA-HP can schedule more tasks (**2.52%** on an average) compared to the state-of-the-art while achieving **7.29%** average energy savings with 9.59 °C reduction in the average temperature of our considered heterogeneous chip-multiprocessor consisting **4** in-order and **4** out-of-order cores.

Keywords: Real-time, Heterogeneous Platform, DVFS, Temperature, Energy

1 Introduction

The logical notion of correctness, and the timely manner in which results are produced, are the most crucial real-time system features. Based on the approach of the schedulers deployed in such systems, they can be categorised as either a *partitioned* or a *global* schedulers [1]. All partition-oriented schedulers have a separate queue for each core, and when a task arrives into a system, it is allotted to any one of such private queues. Such schedulers are constrained that tasks cannot migrate to queues of other cores, but they also benefit from the fact that there is no migration cost involved for such schedulers. Nevertheless, the *partitioning* process of a set of tasks into available cores is an *NP-hard* problem [2]. On the contrary, all global schedulers have a single task queue for all cores, allowing migrations. Such schedulers can provide higher utilisation than partition-oriented schedulers, but they often suffer from a very high migration cost.

To overcome these two categories of schedulers' shortcomings, the research fraternity has started to focus on schedulers based on a hybrid approach. Such schedulers are known as *semi-partitioned* schedulers, and they can be further divided into two sub-categories. The schedulers from the first sub-category deal with the systems where cores are divided into disjoint groups based on their features. A task is allotted to a particular group of cores and is constrained to have migrations among that group's cores only. On the other hand, the schedulers from the second sub-category break the timeline into intervals, often called slices or frames. In each frame, they allow restricted migrations, and the tasks are allowed to synchronise at every interval boundary. Schedulers belonging to the latter sub-category [3, 4] offer high resource utilisation.

To meet the varying demands of modern applications, most of the current real-time systems have started using heterogeneous multicore platforms [5] such as ARM's big.LITTLE, Nvidia Tegra, etc. Hence, to better utilise such platforms, these systems need to adapt to them. On such a platform, the same coding lines may need dissimilar time duration to execute over non-identical cores, making scheduling on such a platform more challenging. As an essential driving force of social development and world economic growth in the 21^{st} century, the ICT (information and communication technology) industry consumes 10% of the global power consumption [6]. The majority of the advanced gadgets are fundamentally founded on batteries, making effective utilisation of energy in these gadgets vital. Consequently, energy management has become a subject of incredible enthusiasm for researchers and specialists in the recent past.

An uncontrolled ascent in temperature swells cooling costs as well as diminishes a system's productivity. A study in [7] has found that the life expectancy of a chip can be diminished by up to half with a temperature surge of $10-15\,^{\circ}\mathrm{C}$ over conventional working temperature. Hence, thermal hotspots play a vital role in the performance of modern-day gadgets. Further, a vast majority of these gadgets are essentially founded on batteries, making the proficient utilisation of power in these gadgets vital. Therefore, analysts and practitioners have started focusing on these aspects together in recent years. Unfortunately, all the literary works have not focused adequately on proficient resource utilisation alongside energy and temperature control in the gadgets. Achieving efficient resource utilisation has always been a cherished aim for designers of these gadgets, which helps them to get proper cost-to-performance value. Therefore, we present an energy and temperature-aware low-overhead scheduler based on a semi-partitioned strategy that can provide efficient resource utilisation in these devices. The major contributions of our work can be summarised as:

- Development of an efficient scheduling strategy which allots tasks on available cores of different types, so that not only tasks meet their deadlines, but the migration overhead is also bounded.
- For each core, we modify the prepared task schedule such that temperatures remain balanced for all cores.
- We apply the Dynamic Voltage/Frequency Scaling (DVFS) technique on each core to reduce their dynamic energy consumption. DVFS reduces power dissipation by scaling down the processor's operation frequency along with its supply voltage.
- Empirical analysis shows that our presented strategy not only outperforms the state-of-the-art [8] technique in terms of resource utilisation but also offers better energy and thermal management in the system.

As we would like to think, the proposed strategy fits precisely to the platforms that have cores with different micro-architectures but have identical ISA, like big.LITTLE® or the Helio X20®.

Paper Organisation

After presenting the relevant related research attempts in the next section (Section 2), we discuss the specifications in Section 3 before detailing the proposed mechanism, ETA-HP, in Section 4. We further analyse our proposed algorithms in Section 5. The empirical studies are presented next in Section 6 together with the analytical discussions. Finally, before concluding the paper in Section 8, we have further summarised the ETA-HP mechanism along with its potential future research avenues in Section 7.

2 Related Works

Researchers have started focusing on the problem of energy management for multicore heterogeneous platforms in recent years. In [9], the authors have proposed a scheduler for cloud-based platforms which tries to reduce energy consumption and minimise the makespan. To reduce the energy consumption, the authors have designed an effective model, where based on the characteristics of the tasks, they are either allocated to GPUs or the CPUs. As the work has considered platforms having only two types of cores, the proposed strategy cannot be applied to platforms that have more core types. Another scheduling strategy for platforms comprised of cores of two-types and having continuous available frequency, has been proposed in [10]. The scheduler uses a constant approximation technique along with the DVFS to achieve it's objective. In another work [11], a scheduler for stochastic tasks has been proposed for platforms having DVFS capability. The task's stochastic processing durations have been derived from separate probability distributions. This work not only tries to minimise makespan but energy consumption in the system as well. In [12], the authors have suggested a scheduling strategy for platforms with DVFS capability and having an arbitrary number of cores. The proposed algorithm works in two phases. In the first phase, the clusters of cores are formed based on their specifications and the tasks are allotted. In the second phase, the per-core DVFS is applied to reduce the energy consumption.

The problem of temperature management for multicore, heterogeneous platforms is also getting a lot of attention in recent times. A temperature-aware scheduling strategy has been presented in [13], which is an extension of a basic scheduler, HETERO-FAIR [14]. However, the work is targeted towards platforms having only two types of cores. Another strategy for similar platforms was proposed [15], in which, at the beginning, the strategy profiles applications to extract their thermal characteristics, and then they assign these tasks on the available cores based on the extracted thermal features of the tasks. In another work [16], authors have presented a partition-oriented strategy. The authors have used heterogeneous configurable cache systems to manage the cores' temperatures. In [17], the authors have presented a partition-oriented scheduler that works in two phases. Initially, the strategy assigns tasks on the available cores and then tries to schedule them under a specified temperature constraint. Next, they try to reduce the makespan.

Few works have focused on the temperature and energy aspects of heterogeneous multicore platforms together as well. One such work was presented in [18], where constant monitoring of the system workload is performed. Whenever the workload gets below a threshold, the work focuses more on the energy aspect, and when the workload is higher than the specified threshold, it focuses more on the management of core temperatures. However, this strategy is based on the first category of semi-partitioned schedulers discussed in the previous section. Hence, it offers low utilisation because of the absence of migrations across the core groups. A partition-oriented strategy has been presented in [19]. Initially, this strategy analysed the various task-to-core assignments' energy consumption and then choose the assignment that leads to minimum energy consumption. Then the strategy applies a heuristic to reduce the cores' temperatures. As there can be many possible task-to-core assignments, this strategy is not suitable for real-time systems. Another partition-oriented strategy has been presented in [20]. At first, the strategy considers the energy and thermal aspects while assigning tasks to cores. Then, a fluid scheduling mechanism [21] is applied on each core to prepare task schedules. A strategy that works under a specified thermal threshold has been presented in [22]. Here, the authors profiled temperature characteristics of all the tasks offline and then categorised the tasks based on their (thermal) profiling. At last, they have applied DVFS at the individual cores.

Table 1: Comparison of Related Works

Related Work	Resource Efficient	Energy Aware	Temperature Aware	Platform
[3]	✓	✓	X	Based on single-core type (Homogeneous)
[4]	✓	✓	X	Based on single-core type (Homogeneous)
[8]	✓	✓	X	Generic number of core types
[9]	X	✓	X	Based on two-type of cores
[10]	X	✓	X	Based on two-type of cores
[11]	X	✓	X	Generic number of core types
[12]	✓	✓	X	Generic number of core types
[13]	X	X	✓	Based on two-type of cores
[15]	X	X	✓	Based on two-type of cores
[16]	X	X	✓	Generic number of core types
[17]	X	X	✓	Generic number of core types
[18]	X	✓	✓	Generic number of core types
[19]	X	✓	✓	Generic number of core types
[20]	X	✓	✓	Generic number of core types
[22]	X	✓	✓	Generic number of core types
ETA-HP	✓	✓	✓	Generic number of core types

A summary of the comparisons of the related works has been provided in Table 1. As stated earlier, resource utilisation along with energy and temperature management have become major design criteria for the manufacturers of modern devices. Unfortunately, none of these prior research attempts have focused adequately on the system's utilisation aspect while performing energy/thermal management, as it can be observed from Table 1.

3 Specifications

System Model: The system under consideration is composed of a set of periodic tasks τ having $\mid \tau \mid$ tasks, i.e. $\tau = \{\tau_1, \tau_2, ...\}$, and a heterogeneous multicore platform $\Pi = \{\Pi_1, \Pi_2, ...\}$ having $\mid \Pi \mid$ cores. Each core may run on a frequency which is chosen from the frequency set $\digamma = \{\digamma_1, \digamma_2, ..., \digamma_{max}\}$, such that, $\digamma_{max}$ represents normalised frequency of 1 and all other frequencies lie between 0 and 1. Every task τ_i is related with a $(2\times \mid \Pi \mid +1)$ tuple $\tau_i \langle u_{i,1}, u_{i,2}, \ldots, u_{i,|\Pi|}, p_i, \Gamma_{ss}^{i,1}, \Gamma_{ss}^{i,2}, \ldots, \Gamma_{ss}^{i,|\Pi|}\rangle$, where,

- $u_{i,j}$ is the *utilisation* of τ_i on Π_j,
- p_i is the fixed inter-arrival time (i.e., *period*) as well as the deadline of τ_i, and

- $\Gamma_{ss}^{i,j}$ is its *steady state temperature* on Π_j. The steady state temperature of a task represents the Π_j's core temperature that is reached when the task executes continuously on Π_j for an adequately prolonged stretch of time, possibly over multiple instances.

At any given instant, rp_i denotes the remaining period of the current instance of τ_i.

Symbol	Description
τ	Task set
τ_i	i^{th} task
p_i	Period of i^{th} task
$\mid \tau \mid$	Number of tasks in τ
Π	Core set
Π_j	j^{th} core
$\mid \Pi \mid$	Number of cores in Π
$\digamma$	Operating frequency set for cores
$u_{i,j}$	Utilisation of τ_i on Π_j
R_k	k^{th} frame
$shr_{i,j,k}$	Share of τ_i on Π_j in R_k
$\Gamma_{ss}^{i,j}$	Steady state temperature of τ_i on Π_j
UF	Utilisation Factor of a task set
LT_1	List of all tasks in a frame
LT_2	List of tasks requiring migration in a frame

Table 2: Important Terminologies

Power Model: The power model used in our work is based on a recent prior work [4]. The dynamic power consumption P in a platform with DVFS capacity is directly related to the level of operating frequency (say $\digamma_k$) and the square of the supply voltage υ_k (i.e. $P \propto \digamma_k \upsilon_k^2$). The supply voltage is again linearly proportional to the operating frequency. Hence, the expression for power consumption may be represented as: $P = c \times \digamma_k^3$, where, c is the proportionality constant.

Temperature Model: The rate of change of temperature [23, 24] of a core is modelled through the following equation:

$$\frac{d\Gamma(t)}{dt} = \frac{P(t)}{C} - \frac{(\Gamma(t) - \Gamma_{amb})}{r \times C}, \tag{1}$$

where, $\Gamma(t)$ and Γ_{amb} are the core's temperature at time t and ambient temperature, respectively. $P(t)$ is the power consumption at time t, while r and C are thermal resistance and thermal capacitance of the core, respectively. Scaling $\Gamma(t)$ such that Γ_{amb} is zero (i.e., by replacing $\Gamma(t)$ - Γ_{amb} with $\Gamma(t)$), Equation 1 gets transformed as:

$$\frac{d\Gamma(t)}{dt} = aP(t) - b\Gamma(t), \tag{2}$$

where, $a = 1/C$ and $b = 1/rC$.

For a core with multiple voltage/frequency levels, the gross power consumption of a core when running at the k^{th} voltage level, can be modelled as [23]:

$$P(k) = (C_0(k) \times v_k + C_1(k) \cdot \Gamma(t) \cdot v_k) + C_2 \times v_k^3 \tag{3}$$

where C_2 is a constant and $C_0(k)$ and $C_1(k)$ are the constants dependent on the voltage level v_k. Accordingly, the rate of temperature variation can be formulated from Equation 2 and Equation 3 as:

$$\frac{d\Gamma(t)}{dt} = a[C_0(k) \times v_k + C_2 \cdot v_k^3] + a \cdot C_1(k) \cdot \Gamma(t) \cdot v_k - b \cdot \Gamma(t)$$

$$\Rightarrow \frac{d\Gamma(t)}{dt} = A(k) - B(k)\Gamma(t), \tag{4}$$

where $A(k) = a \cdot (C_0(k) \times v_k + C_2 \times v_k^3)$ and $B(k) = b - a \cdot C_1(k) \cdot v_k$.

For an interval $[t_0, t_e]$ in which τ_i is executing on Π_j, if the core temperature is Γ_0 at time t_0, the temperature Γ_e at the end of the interval at time t_e is given by [25]:

$$\Gamma_e = \frac{A(k)}{B(k)} + (\Gamma_0 - \frac{A(k)}{B(k)})e^{-B(k)(t_e - t_0)}$$

$$\Gamma_e = \Gamma_{ss}^{i,j}(k) + (\Gamma_0 - \Gamma_{ss}^{i,j}(k))e^{-B(k)(t_e - t_0)} \tag{5}$$

where $\Gamma_{ss}^{i,j}$ is the steady state temperature of task τ_i on core Π_j and is dependent on the overall power consumption of τ_i. The steady state temperature of a task represents the core's temperature that is reached when the task executes continuously on the core for a sufficiently long time, possibly over multiple instances.

Nowadays, many embedded systems are equipped with several levels of operating frequencies for the cores. The running of cores at a reduced frequency results in lower temperatures and leads to long execution times. Therefore, a scheduler has to carefully select a group of frequencies for the cores in real-time systems. We have obtained the execution requirements for two programs from PARSEC [26] benchmark, namely, *Bodytrack* and *Canneal*, by executing them in gem5 [27] for an Alpha21364 processor (32 nm technology) for various operating frequencies. We further used McPAT [28] and HotSpot 6.0 [29] simulators to get power traces and steady state temperatures, respectively, for the individual tasks while executing them on a system having the same configuration. The results are shown in Figure 1.

As the temperature of a core is dependent on the operating frequency of the core and the task executing on it, running a core on a lower frequency results in a lower core temperature. We used the following equation to relate the operating frequency of the system with the steady state temperature of a task:

$$\Gamma_{ss}^{i,j}(k) = \beta \times \digamma_k \times \Gamma_{ss}^{i,j}(\digamma_{max}) \tag{6}$$

where β is a constant, $\digamma_k$ is the k^{th} level of operating frequency and $\Gamma_{ss}^{i,j}(k)$ denotes the steady state temperature of τ_i on Π_j at k^{th} level of operating

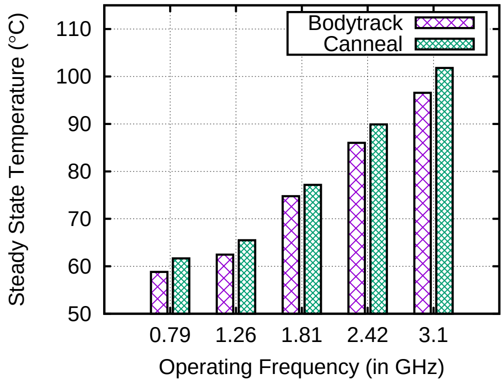


Fig. 1: Effect of Operating Frequency on Steady State Temperature

frequency. Table 2 lists some of the important terms which have been used in this proposed work. The subsequent section presents a detailed description of our proposed strategy, ETA-HP.

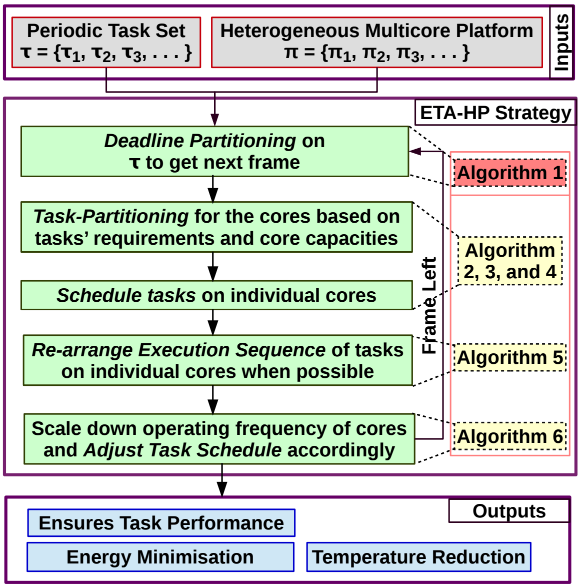

Periodic Task Set
τ = {τ1, τ2, τ3, . . . }
Heterogeneous Multicore Platform
π = {π1, π2, π3, . . . }
Inputs
ETA-HP Strategy
Deadline Partitioning on τ to get next frame
Algorithm 1
Task-Partitioning for the cores based on tasks' requirements and core capacities
Algorithm 2, 3, and 4
Schedule tasks on individual cores
Frame Left
Re-arrange Execution Sequence of tasks on individual cores when possible
Algorithm 5
Scale down operating frequency of cores and Adjust Task Schedule accordingly
Algorithm 6
Outputs
Ensures Task Performance
Energy Minimisation
Temperature Reduction

Fig. 2: Scheduling Strategy of ETA-HP

4 Proposed Scheduling Scheme: ETA-HP

Our proposed scheduling strategy, *ETA-HP* (Algorithm 1), is a four-level hierarchical scheduling strategy. Initially, the technique of deadline partitioning [4] is used by the strategy to find frames, where an individual frame is a consecutive group of time-slots addressing a span between two sequential deadlines corresponding to the set of ready tasks. Within a frame, each task is scheduled for appropriate times (Algorithms: 2, 3, and 4) on available cores. It may be noted that a task may be scheduled on multiple cores within a frame, but it is not scheduled on more than one core at any individual time slot. Then ETA-HP tries to rearrange the execution order of the tasks on the cores such that the core temperatures remain balanced (Algorithm 5). At last, it finds a suitable operating frequency for every core based on the workload allotted in the previous step (Algorithm 6). A summarised scheduling strategy of ETA-HP has been presented in Figure 2.

4.1 ETA-HP (Algorithm 1)

The proposed algorithm, ETA-HP, progresses in the system frame by frame. To do so, it proceeds by finding the size of the next frame by using *Deadline Partitioning* [4], in which the frames are restricted by the periods of all ready tasks. It is achieved by searching the task with the earliest period among all task instances.

$$\mid R_k \mid = min\{rp_1, rp_2, ..., rp_{|\tau|}\} \tag{7}$$

where R_k is the k^{th} frame. Within R_k, all elements of the *Basic Allocation*

Algorithm 1 ETA-HP

```
Input: $\tau$, $\Pi$, $\digamma$
Output: Final Schedule
1 while true do
2     Using deadline-partitioning, compute next frame (say $k^{th}$) $R_k$
3     Let $BAM_k$ be the Basic Allocation Matrix for $R_k$
4     Initialise all entries of the Basic Allocation Matrix $BAM_k$ to $\emptyset$
5     TENTATIVE-SCHEDULE ($\tau$, $\Pi$, $R_k$, $BAM_k$)
6     if scheduling of $\tau$ is feasible on $\Pi$ then
7         TA-ALLOCATE ($\tau$, $R_k$, $BAM_k$)
8         EA-ALLOCATE ($\tau$, $R_k$, $BAM_k$, $\digamma$)
9 return Final Schedule
```

Matrix (BAM), that maintains the order and length of the task execution on the available cores, are initialised to $\emptyset$ (Line 4). Next, TENTATIVE-SCHEDULE() is called to allocate tasks on cores and construct a tentative schedule (Line 5). Suppose the function can build a workable schedule in the previous step. In that case, it calls functions TA-ALLOCATE() (Line 7) and EA-ALLOCATE() (Line 8) to make the schedule temperature-aware and energy-aware, respectively.

4.2 TENTATIVE-SCHEDULE (Algorithm 2)

Since every task may have different utilisation on different cores of a heterogeneous platform, it is challenging to schedule them. Initially, the function TENTATIVE-SCHEDULE() finds the shares for each task on all cores in the ensuing frame (Line 4 - Line 7) using the equation:

$$shr_{i,j,k} = \lceil u_{i,j} \times \mid R_k \mid \rceil \tag{8}$$

Algorithm 2 TENTATIVE-SCHEDULE

```
Input: $\tau$, $\Pi$, $R_k$, $BAM_k$
Output: Basic Allocation Matrix $BAM_k$
1  Let $LT_1$ and $LT_2$ be sorted lists of tasks based on their share values
2  Initialise $LT_1 = \emptyset$ and $LT_2 = \emptyset$
3  {Find task shares in $R_k$}
4  for $i = 1 :\mid \tau \mid$ do
5      for $j = 1 :\mid \Pi \mid$ do
6          $shr_{i,j,k} = \lceil u_{i,j} \times \mid R_k \mid \rceil$
7          $LT_1 = LT_1 \cup \{\langle i, j, shr_{i,j,k} \rangle\}$
8  ALLOCATE-FIXED ($LT_1$, $LT_2$, $BAM_k$, $\Pi$)
9  ALLOCATE-MIGRATE ($\tau$, $LT_2$, $BAM_k$, $\Pi$)
10 return $BAM_k$
```

The share value of a task indicates its proportional demand in a frame to it's total requirement. These values are kept in the *share matrix shr* of size $\mid \tau \mid \times \mid \Pi \mid$. Also, this function maintains a sorted list LT_1 of non-decreasing order based on the shared values. Each element of LT_1 is having a format: $\langle i, j, shr_{i,j,k} \rangle$, where i is task ID, j is core ID and $shr_{i,j,k}$ is the task's share value for the frame R_k. At last, function ALLOCATE-FIXED() (Line 8) and function ALLOCATE-MIGRATE() (Line 9) are called to schedule tasks on the available cores.

4.3 ALLOCATE-FIXED (Algorithm 3)

Every task which can be completely designated on any single core is called a *fixed/non-migrating* task. The function ALLOCATE-FIXED() tries to schedule such non-migrating tasks. It uses a sorted list LT_2 (in non-decreasing order) based on the shared values of the tasks on cores to keep information about the tasks that cannot be fully allocated on any single core. The format of elements of the list LT_2 is similar to the list LT_1. The function starts its operation by iterating over the sorted list LT_1 (Line 1 - Line 7). At start of every iteration, it extricates an element $\langle i, j, shr_{i,j,k} \rangle$ from the front of LT_1 (Line 2). Then it verifies whether Π_j can execute τ_i for $shr_{i,j,k}$ in the ensuing frame. When possible, the function updates BAM_k with beginning and end time for τ_i on Π_j (Line 4). A task can only start running on a core after the previously allocated task on Π_j concludes its execution. Next, the function deletes all nodes corresponding to τ_i from LT_1 (Line 5). In a scenario, when Π_j is unable to run τ_i for $shr_{i,j,k}$ time-slots in the ensuing frame, $\langle i, j, shr_{i,j,k} \rangle$ is added to the list of migrating tasks LT_2 (Line 7). The function ALLOCATE-FIXED returns BAM_k and LT_2 to the function TENTATIVE-SCHEDULE() (Line 8).

Algorithm 3 ALLOCATE-FIXED

```
Input: $LT_1$, $LT_2$, $BAM_k$, $\Pi$
Output: $BAM_k$ (Basic Allocation Matrix with fixed tasks), $LT_2$ (Sorted list
        of migrating tasks)
1 while $LT_1$ is not empty do
2     Get the first element of $LT_1$: $\langle i, j, shr_{i,j,k} \rangle$
3     if $shr_{i,j,k}$ can be fully allocated on $\Pi_j$ for $shr_{i,j,k}$ then
4         Assign start and end times of $\tau_i$ on $\Pi_j$, i.e., $BAM_k[i][j] =$
          $\langle \text{start\_time}(\tau_i), \text{end\_time}(\tau_i) \rangle$
5         Remove all entries of $\tau_i$ from $LT_1$ and $LT_2$
6     else
7         $LT_2 = LT_2 \cup \{\langle i, j, shr_{i,j,k} \rangle\}$
8 return $BAM_k$, $LT_2$
```

4.4 ALLOCATE-MIGRATE (Algorithm 4)

The function ALLOCATE-MIGRATE considers each element of LT_2 one by one. During each pass, it extricates an element (say $\langle i, j, shr_{i,j,k} \rangle$) from front of the list (Line 3) and moves all elements of τ_i from LT_2 to LT_3 (Line 4). As the elements in LT_2 were sorted in order, the entries in LT_3 are also sorted based on the shared values of τ_i on the available cores. Next, it iterates through the list LT_3, where each node of LT_3 contains the execution requirement of τ_i on a specific core-type. The list is sorted in non-decreasing order of the execution requirements of the task, i.e. the degree of favouritism of τ_i to core-types. In each iteration of the loop, it extracts the current element of LT_3 (say $\langle i, j, shr_{i,j,k} \rangle$) and tries to schedule τ_i on Π_j for a certain duration. Let $ns_{i,j}$ represents the *normalised unallocated share* of τ_i on Π_j and rc_j be the remaining capacity of Π_j in R_k. When the algorithm tries to allocate τ_i on Π_j, the following two situations may arise:

Algorithm 4 ALLOCATE-MIGRATE

```
Input: $\tau$, $LT_2$, $BAM_k$, $\Pi$
Output: $BAM_k$ Basic Allocation Matrix
1  while $LT_2$ is not empty do
2      Let $LT_3$ be an empty sorted list of tasks based on share values
3      Extract the entry $\langle i, j, shr_{i,j,k} \rangle$ from $LT_2$
4      Move every element corresponding to $\tau_i$ from $LT_2$ to $LT_3$
5      Let $ns_{ij}$ be the normalized unallocated share of $\tau_i$ on $\Pi_j$
6      while $LT_3$ is not empty do
7          Extract the first element of $LT_3$: $\langle i, j, shr_{i,j,k} \rangle$
8          Let $rc_j$ be the remaining spare capacity of $\Pi_j$
9          if $\Pi_j$ is the first core for allocation of $\tau_i$ then
10             $ns_{i,j} = shr_{i,j,k}$
11         else
12             Let $\Pi_q$ be the latest core on which $\tau_i$ was scheduled
13             $ns_{i,j} = ns_{i,q} \times u_{i,j}/u_{i,q}$
14         if $ns_{i,j} > rc_j$ then
15             Update $BAM_k[i][j]$ to schedule $\tau_i$ on $\Pi_j$ for duration $rc_j$
16             $ns_{i,j} = ns_{i,j} - rc_j$
17         else
18             Update $BAM_k[i][j]$ to schedule $\tau_i$ on $\Pi_j$ for duration $rc_j - \lceil ns_{i,j} \rceil$
19             break
20     if ($ns_i \neq 0$) or ($\tau_i$ is allocated on more than one core in parallel) then
21         $\tau$ cannot be scheduled on $\Pi$
22 return $BAM_k$
```

- $rc_j < ns_{i,j}$: In such a scenario, the function partially allocates τ_i on Π_j for rc_j time-slots and updates $ns_{i,j}$.
- $ns_{i,j} \leq rc_j$: In this scenario, the function fully allocates τ_i on Π_j for $ns_{i,j}$ time-slots and deletes the list LT_3.

After allotting τ_i on different cores, the function tries to schedule them on the allocated cores. For this purpose, ETA-HP uses the following heuristic strategy, which helps it to prevent the execution of τ_i on more than one core at the same time slot (Line 9 - Line 19). On the first core, where τ_i has been partially allotted, the task is scheduled from the start of the R_k. On subsequent cores, τ_i is scheduled at the time slot after it finishes its execution on the last partially allotted core. The checkpoints indicating starting and ending of execution on each core maintained in BAM_k. If there is a lack of capacity in the available multicore platform to schedule τ_i, the function concludes scheduling of τ is not possible on Π. Otherwise, BAM_k, which consists of a partially constructed schedule for R_k, is returned.

4.5 TA-ALLOCATE (Algorithm 5)

In literature [24], it has been found that executing hot and cold tasks alternatively on cores can help us maintain a better temperature control at the cores. Hence, ETA-HP uses this idea on each core within every frame. For each core, Π_j, TA-ALLOCATE() constructs the list LT_4 containing non-migrating tasks allocated on the core (Line 2). Then each task of LT_4 is classified as either a cool or a hot task by comparing its steady state temperature with the average steady state temperature of the task set Γ^{ss}_{avg} (Line 3). Next, it constructs the list of tasks LT_5 by alternatively extracting the hottest and coolest task from LT_4 (Line 4). After that, it extricates tasks one by one from LT_5 and schedules them in the time slots where a migrating task is not scheduled. At last, BAM_k is updated for the current frame R_k.

Algorithm 5 TA-ALLOCATE

```
Input: $\tau$, $R_k$, $BAM_k$
Output: $BAM_k$ for $R_k$
1 for each core $\Pi_j \in \Pi$ do
2     Create $LT_4$, a sorted list of non-migrating tasks which have been scheduled
        on $\Pi_j$ based on their $\Gamma^{i,j}_{ss}$
3     Classify every task (say $\tau_i$) in $LT_4$ as either hot task if ($\Gamma^{i,j}_{ss} \geq \Gamma^{ss}_{avg}$; else
        classify it as a cold task
4     Create $LT_5$ by alternately extricating the hottest and the coolest task from
        $LT_4$
5     Extract tasks from $LT_5$ one by one and schedule them on $\Pi_j$ for the time-
        slots in which a migrating task is not scheduled
6     Update $BAM_k$ with the schedule for the core $\Pi_j$
7 return $BAM_k$
```

4.6 EA-ALLOCATE (Algorithm 6)

The function EA-ALLOCATE() considers cores of the platform one at a time. It calculates the spare capacity of the core under consideration (say Π_j). Then the function finds the most suitable frequency that is needed to execute the tasks allocated on the core. This may be calculated as:

$$\digamma_{opt} = \frac{\Sigma_{\tau_i \epsilon |FIXED|} shr_{i,j,k}}{| R_k | - \Sigma_{\tau_i \epsilon |MGR|} shr_{i,j,k}} \tag{9}$$

where $| MGR |$ and $| FIXED |$ are sets of migrating and fixed tasks which have been allotted on the core. As the required frequency $\digamma_{opt}$ may not be always present in a core with discrete levels of frequency, the next higher level available frequency is selected in such a case. Its execution length is adjusted for each fixed task with respect to $\digamma_{opt}$ of the core.

Algorithm 6 EA-ALLOCATE

```
Input: $\tau$, $R_k$, $BAM_k$, $\digamma$
Output: $BAM_k$ for $R_k$
1 for each core $\Pi_j \in \Pi$ do
2     Find remaining capacity of $\Pi_j$: $rc_j$
3     if $rc_j \neq 0$ then
4         Find the frequency $f_{opt} \in \digamma$ for $\Pi_j$
5         Update $BAM_k$ for all non-migrating tasks allocated on $\Pi_j$
6 return $BAM_k$
```

5 Analysis of the algorithm

In this section, we will analyse each of the functions used in the algorithm one by one:

- **Function ALLOCATE-FIXED()**: There are $\mid \tau \mid \times \mid \Pi \mid$ elements in the list LT_1. We sequentially consider the elements. Getting an element from LT_1 will have $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid \times log\ (\mid \tau \mid . \mid \Pi \mid))$ time-complexity. The computation of the time-slots for each task's execution on a core has a complexity of $\mathcal{O}(1)$. Adding a migrating task to L_2 has $\mathcal{O}(1)$ complexity too. A 2D array may be used to keep task-to-core allocations. This operation will also have $\mathcal{O}(1)$ complexity for each task. Therefore, the time complexity of the function ALLOCATE-FIXED() becomes $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid \times log\ (\mid \tau \mid \cdot \mid \Pi \mid))$.
- **Function ALLOCATE-MIGRATE()**: It consists of $\mid \tau \mid \times \mid \Pi \mid$ passes, one each for a migrating task contained in LT_2. In each pass, the function may try to allocate a part of the task on the available cores. This operation will have a complexity of $\mathcal{O}(\mid \Pi \mid)$. Therefore, the function ALLOCATE-MIGRATE() has a complexity of $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid^2)$.
- **Function TENTATIVE-SCHEDULE()**: The function has to compute share values for all tasks for every core of the platform. This operation will have a complexity of $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid)$. Next, adding an entry to LT_1 will have $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid \times log\ (\mid \tau \mid \cdot \mid \Pi \mid))$ complexity, of a priority queue is used to implement LT_1. Then functions ALLOCATE-FIXED() and ALLOCATE-MIGRATE() are called for scheduling of the tasks on available cores. Hence, time complexity of the function TENTATIVE-SCHEDULE() becomes $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid^2 + \mid \tau \mid . \mid \Pi \mid \times log\ (\mid \tau \mid \cdot \mid \Pi \mid))$.
- **Function TA-ALLOCATE()**: The construction of LT_4 for every core may be done in TENTATIVE-SCHEDULE() and this operation will have a complexity of $\mathcal{O}(\mid \tau \mid)$ for every core. Similarly, construction of LT_5 will have a time complexity of $\mathcal{O}(\mid \tau \mid)$ for every core. Therefore, the complexity of the function TA-ALLOCATE() becomes $\mathcal{O}(\mid \tau \mid \cdot \mid \Pi \mid)$.
- **Function EA-ALLOCATE()**: Based on the workload, the function needs to calculate $\digamma_{opt}$. This operation can be carried out in a constant time. After calculating the required frequency, each non-migrating task's schedule must be readjusted accordingly on every core, which will have a complexity

of $\mathcal{O}(\mid \tau \mid)$. Therefore, the function EA-ALLOCATE() will have a time complexity of $\mathcal{O}(\mid \tau \mid . \mid \Pi \mid)$.

- **Function ETA-HP()**: Calculating size of the subsequent frame will have a complexity of $\mathcal{O}(\mid \tau \mid)$, since it has considered the period of every ready task. Then the functions TENTATIVE-SCHEDULE(), TA-ALLOCATE() and EA-ALLOCATE() are called to carry out their operations. Therefore, the function ETA-HP() will have a time-complexity of $\mathcal{O}(\mid \tau \mid . \mid \Pi \mid \times(\mid \Pi \mid + log\ (\mid \tau \mid . \mid \Pi \mid)))$.

Assuming the number of tasks to be much higher than the number of cores in a system, the overall complexity of ETA-HP becomes $\mathcal{O}(\mid \tau \mid \times log\ (\mid \tau \mid))$ per frame.

It may be noted that TA-ALLOCATE (Algorithm 5) only rearranges the execution of tasks on the available cores and EA-ALLOCATE (Algorithm 6) extends task executions in frames by reducing the operating frequencies of the cores. But the operating frequencies are lowered such that none of the tasks misses their execution requirements in a frame. Hence, these two algorithms do not hamper the feasibility of task allocation, which has been done by the TENTATIVE-SCHEDULE (Algorithm 2).

Migration Overheads: In ETA-HP, a task is allotted to multiple cores in a frame when it cannot be fully allotted on any single core. It is done to avoid executing such a task on more than one core simultaneously in any time slot. From the scheduling strategy which has been explained in the previous section, we may infer that in a frame, a core belonging to a platform may be allotted a group of fixed tasks along with either: i. zero migrating tasks, ii. several migrating tasks who finish their execution of the frame on this core, iii. a single migrating task which finishes its execution of the frame on some other core, and iv. a single migrating task that finishes its execution of the frame at some other core and several migrating tasks that finish their execution of the frame on this core. Hence, we may infer that the cores that have been allotted non-migrating tasks and migrating tasks that finish at some other cores do not incur any migration in the frame. When a migrating task does not finish its execution of the frame on a specific core (say Π_j) and is allotted on that core, Π_j will not have spare capacity to allot any more fixed or migrating tasks. Therefore, the number of migrations on Π_j with such a migrating task is constrained to only one. At most, there can be $\mid \Pi \mid -1$ such cores with such migrating tasks which do not finish their execution on the cores. Hence, the number of migrations using such a scheduling mechanism is bounded by $\mid \Pi \mid -1$ in a frame.

Arrivals and Departures of Dynamic Tasks: In a dynamic system, a periodic task may arrive or depart at any time instant. When a new task (let us say τ_i) arrives, ETA-HP first verifies the arrived task's deadline. If there is sufficient time to complete its execution by beginning it after the next frame, this task's scheduling is then delayed till the beginning of the subsequent frame. If not, then the system stops its work from the current time slot and recalculates the set of new frames by considering the newly arrived task's deadline. Next, it constructs a new schedule for the tasks. When an executing task finishes its execution and leaves the platform, its entry is removed from the list of ready tasks and is not considered for scheduling from the following frames. The proposed scheduler, ETA-HP, may be modified easily for sporadic tasks as well. In such a case, only the deadlines of current instances of executing tasks in the system should be considered for deadline partitioning, and the deadlines of subsequent instances will not be used to find the successive frames unless they start their execution in the system.

Assuring Task Deadlines: In ETA-HP, the execution of tasks in the system progresses frame by frame. Each core is allotted a maximum normalised workload of one within each frame, i.e., the sum of the task shares allocated to a core in a frame should be less than or equal to the core's processing capacity in a frame. The scheduler follows a two-phase task-core allocation rule. In the first phase, the allocation is carried out for all those tasks which can be fully allocated on a single core. In the second phase, all those tasks which need migrations across available cores of the platform are allocated on multiple cores, but the scheduler ensures that they are not executed in parallel on multiple cores in any time slot. When it cannot allot any task on available cores without violating the core capacity or parallel execution constraints, that task is rejected from execution in the system. Hence, the boundary of frames acts as pseudo-deadlines for the tasks. If all tasks meet their execution requirements within each frame, they are guaranteed to meet their actual deadlines. By breaking the execution of tasks into multiple frames, the scheduler can divide task scheduling into multiple sub-problems, where each sub-problem is solved by preparing a task schedule for a frame. Hence, using such a mechanism, ETA-HP ensures that all tasks meet their deadlines.

6 Experimental Set Up and Results

We have compared the performance of ETA-HP against three strategies, namely, i. TARTS [24], ii. HEARS [8], and iii. TA-SS [20]. TARTS is a semi-partitioned approach-based temperature-aware scheduler for homogeneous platforms. It performs scheduling based on the heuristic of the hottest task on the coolest core, but it is oblivious to the system's dynamic energy consumption. HEARS is a DVFS based energy-aware heuristic scheduler for heterogeneous platforms having a variable number of core types. It examines the present levels of the cores' operating frequencies while performing task-to-core allotment to minimise energy consumption in the system. On the other hand, TA-SS is an energy and temperature-aware scheduler for heterogeneous multicore platforms. All the energy/temperature efficient schedulers for heterogeneous platforms follow either global or partitioned strategies. Hence, they offer low resource utilisation and are inappropriate for comparison. As a salient feature of ETA-HP is resource utilisation along with energy and thermal efficiency, the performance of our algorithm has been compared against the following algorithms: i. TARTS, which is only a temperature-aware scheduler, so we applied the DVFS technique to make it energy-aware, ii. HEARS, which is an energy-aware semi-partition oriented scheduler but is temperature-ignorant and, iii. TA-SS, which is an energy and temperature-aware partition-oriented scheduler, so we applied the deadline partitioning technique over it to make it semi-partitioned.

Frequency (in MHz)	**Frequency** (Normalised)	**Frequency (in MHz)**	**Frequency** (Normalised)
900	0.3	2100	0.7
1200	0.4	2400	0.8
1500	0.5	2700	0.9
1800	0.6	3000	1.0

Table 3: Available Frequency

6.1 Experimental Set Up

All simulation instances for the concerned algorithms have been executed for 100000 time-slots with task sets having specific *utilisation factors. Utilisation Factor (UF)* is characterised as the proportion between the *summation over the average utilisation of tasks* and the *number of available cores,* i.e. $UF = \frac{\sum_{i=1}^{|\tau|} avg_{j=1}^{|\Pi|}(u_{i,j})}{|\Pi|}$. The randomly generated utilization values have been scaled fittingly for creating task sets with a specific UF value. The ambient temperature for the simulations has been set at $25°C$. We simulated 50 separate test cases for each set of input parameters, and then the average of these cases has been taken as the final result. We conducted two sets of experiments to evaluate and compare the algorithms' results. Benchmark programs have been used in the first set to analyse the algorithms' performance in real-life situations. In contrast, detailed simulation-based experiments using synthetic task sets have been carried out in the second set to validate the algorithms' efficiency over varying situations that may be encountered. We varied the steady state temperatures of each task τ_i on different cores at random in the range of $\Gamma_{ss}^i \pm \alpha$, where the value of α lies in the range of 0% to 10% of Γ_{ss}^i. Table 3 lists the operating frequencies that we used in our experiments. The β value has been chosen randomly from the range of -0.05 to 0.0 and sets arbitrarily for each task to calculate $\Gamma_{ss}^{i,j}(k)$.

Program	**Execution Requirement (in ms)**	**Steady-State Peak Temperature (in $^{\circ}C$)**
Bodytrack	3824	85
Canneal	1007	80
Dedup	6455	91
Fluidanimate	4090	81
Freqmine	11082	84
Streamcluster	6156	68
Swaptions	4535	76
x264	1203	85

Table 4: Task Specifications for Benchmark Programs.

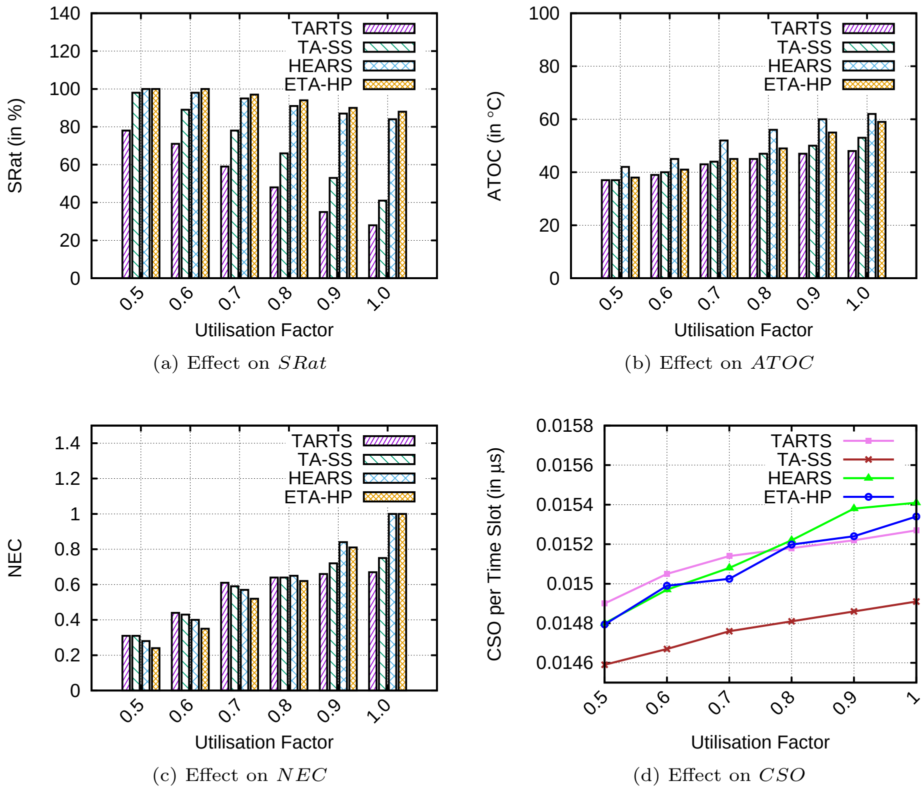
Framework for Benchmarks: The PARSEC [26] benchmark suite (with a large input set) has been used to substantiate efficiencies of the algorithms over different real-life scenarios that may arise. Other than *Swaptions* (compute intensive) and *Streamcluser* (memory intensive), all of these considered benchmark applications are mixed workloads, i.e. both memory and compute intensive, and have different executional requirements in terms of computation and working sizes at the caches. In [30], a thorough discussion of the procedure for calculating each program's execution specifications as well as it's steady state temperature is presented, and the results have been listed in Table 4[1]. We received periodic performance traces from Gem5 [27] simulator for an 8-core based heterogeneous chip-multiprocessor (considering 32nm CMOS technology), where each of the faster 4 Out-of-Order cores can operate at a maximum frequency of 3.0GHz, and each of the 4 smaller In-Order cores can have a maximum frequency of 1.8GHz. The normalised frequencies available for faster cores range from 0.6 to 1.0, while the normalised frequencies available for slower cores range from 0.3 to 0.6 (ref. Table 3). Note that, for each of our cores (both in-order and out-of-order), we have considered Alpha 21364 ISA.

Simulation Setup: The periodic performance traces from Gem5 is fed to McPAT [28] for power traces. These derived power traces are now provided to HotSpot 6.0 [29] along with the floorplan of the heterogeneous chip to get the transient as well as steady state temperatures for the individual tasks (Γ^i_{ss}). Note that the floorplan for the entire chip is generated by the *Hotfloorplan* module of Hotspot 6.0 at the beginning of the simulation by obtaining area details from the McPAT. Each task set consists of 20 tasks, whose instances have been selected randomly from the 8 benchmark applications (repeated to prepare the task set) listed in Table 4.

Framework for Synthetic Tasks: The simulation framework utilised in this work considers randomly created task sets with sizes (n) ranging from 20 to 100. A standard distribution of $\sigma^e = 100$ and $\mu^e = 20$ were used to produce

[1]Based on the execution requirements of individual tasks, we simulate PARSEC application (continuous execution in RoI) accordingly (by specifying the execution span in Gem5 simulator) and obtain the respective execution requirements.

the task execution specifications. For the tasks, steady state temperatures were chosen randomly from a uniform distribution ranging from 40 °C to 120 °C. We have varied the number of cores in the platforms from 2 to 10, and the considered systems have a high UF value of 0.9.



(a) Effect on *SRat* (b) Effect on *ATOC*

(c) Effect on *NEC* (d) Effect on *CSO*

Fig. 3: Benchmark Program Results
(Effect of variation in UF values)

6.2 Experimental Results

We have considered the following four metrics to analyse the performance of the algorithms:
i. *Average Temperature Of Cores* (*ATOC*) denotes the ratio of the summation of the average temperatures of the cores measured at the end of all frames to the total number of frames.
ii. *Success Ratio* (*SRat*) denotes the percentage of tasks that have been successfully scheduled, and therefore, it also gives a measure of the task feasibility analysis of the algorithms.

iii. *Normalised Energy Consumption* (*NEC*) denotes the normalised energy consumption in the system. It is the ratio of the actual energy consumption of the cores operating at the computed operating frequency F_{opt} to the maximum possible energy consumption of the cores, i.e. operating at F_{max}.
iv. *Context-Switch Overhead* (*CSO*) gives a measure of the context-switch overheads incurred by the algorithms.

6.2.1 Benchmark Program Results

We have studied the effect of variation in Utilisation Factor (UF) on Average Temperature Of Cores ($ATOC$), Success Ratio ($SRat$), Normalised Energy Consumption (NEC) and Context-Switch Overhead (CSO) for the algorithms on platforms having 8 cores.

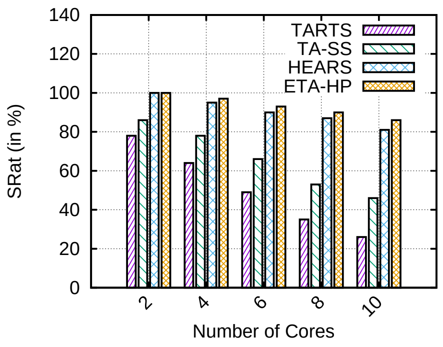
Effect on SRat: In this experiment, we observed the variation in SRat values with the increase in workload. As stated before, the metric SRat helps us evaluate the algorithms' efficiency in terms of the ratio of the number of tasks they were able to schedule to the total number of tasks that they were supposed to schedule. We may observe from Figure 3a, the SRat values decrease with an increase in UF values for all the algorithms. This phenomenon may be attributed to increased workload, resulting in a higher probability of tasks requiring migration within the frames. Although TARTS is a semi-partitioned temperature-aware scheduler, it primarily targets homogeneous multicore platforms. Hence, it is oblivious towards task-core affinity, which leads to inefficient usage of computing resources as a task may be scheduled on a core with a higher execution requirement than other cores. Further, at higher utilisation values, it may require task migrations which needs to be handled carefully on heterogeneous multicore platforms. Therefore, TARTS gradually fails to schedule the task sets when the system workload increases. TA-SS is based on partition oriented scheduling approach, and hence, it inherits the disadvantage of low resource utilisation of the partitioned approach. We employed the deadline partitioning concept to make it semi-partitioned. Still, it only allows migration at the frame boundaries, which leads to lower SRat values than other semi-partitioned algorithms based on heterogeneous platforms. On the other hand, HEARS and ETA-HP allow migrations within frames and are based on the semi-partitioned approach. Still, they might fail to schedule all the tasks at the higher workloads when tasks may require executions on more than one core simultaneously, which is not allowed. The task set is rejected from further scheduling in such a scenario, resulting in reduced SRat values for HEARS and ETA-HP. Also, HEARS performs task-to-core allocation based on the minimum increase in energy consumption for every step. In contrast, ETA-HP performs task-to-core allocation based on task-to-core affinities, which helps ETA-HP to achieve better SRat values than HEARS. From Figure 3a, we may observe that the SRat values decrease from 78% to

28%, 98% to 41%, 100% to 84% and 100% to 88% with the utilisation variation from 0.5 to 1.0, for TARTS, TA-SS, HEARS and ETA-HP, respectively.

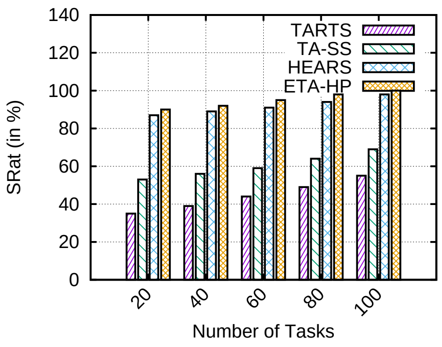
Effect on ATOC: We also analyse the effect of variation in UF on the average temperature of the cores. The UF values have been varied from 0.5 to 1.0. As we can observe from the Figure 3b, the ATOC values rise in the system for all the algorithms with a rise in UF values. This is due to the fact that with an increase in the system workload, there are fewer idle slots in a frame. So, the cores get lesser time to cool down. Also, at a lesser workload, all the algorithms use DVFS to scale down the operating frequencies of the cores, which leads to further reduction of the core temperatures. The heuristic used in HEARS is only focused on energy management and is not temperature-aware. So, the temperature of the cores is highest when they execute the HEARS algorithm. TA-SS is only focused on managing core temperatures in a heterogeneous platform and does not care about other performance criteria. So, it performs better than HEARS and ETA-HP. The TARTS algorithm is primarily focused on managing core temperatures in a homogeneous multi-core platform. Hence, TARTS and TA-SS are able to better manage the core temperatures than HEARS and ETA-HP. But as stated in the results of the last experiment, the SRat values for TARTS and TA-SS algorithms are very low at a higher workload. Hence, they schedule fewer tasks in the system and get more idle time-slots, which they use to lower the core temperatures further. Therefore, TARTS and TA-SS are able to achieve lower ATOC values, although at the cost of lower SRat values. Still, ETA-HP is able to better manage the temperatures of the cores with its heuristic temperature-aware scheduling strategy than HEARS and fetch slightly inferior results than TARTS and TA-SS. From Figure 3b, we can observe that the ATOC values increase from 37.09 °C to 48.17 °C, 33.07 °C to 53.11 °C, 42.27 °C to 62.03 °C and 38.91 °C to 59.63 °C with the utilisation variation from 0.5 to 1.0, for TARTS, TA-SS, HEARS and ETA-HP, respectively.

Effect on NEC: We further observed the variation in NEC values with the increase in workload, depicted in Figure 3c. According to this figure, the NEC values increase with an increase in UF values for all the algorithms. This phenomenon may be attributed to the fact that all the algorithms can run the cores at lower operating frequencies at a lower workload. But as the UF values increase in the system, the algorithms have to choose a higher frequency to run the required workload, resulting in higher energy consumption in the system. As discussed earlier, the SRat values of TARTS and TA-SS are lower than ETA-HP and HEARS, which means that TARTS and TA-SS cannot run all the input tasks at a higher workload. So, they consume lesser energy even at a high prescribed system workload. HEARS checks the current levels of operating frequencies of the cores while performing task-to-core allotment. In some cases, it leads to higher energy consumption for tasks that are allotted later because previous tasks were allotted by only checking the local minimum. However, ETA-HP performs task-to-core allocation first based on the execution requirement of the tasks and then applies the DVFS technique on individual cores to reduce energy consumption, which leads to better energy efficiency in the system. From Figure 3c, we may observe that the NEC values increase from 0.31 to 0.67, 0.31 to 0.75, 0.28 to 1.0 and 0.24 to 1.0 with utilisation variation from 0.5 to 1.0, for TARTS, TA-SS, HEARS and ETA-HP, respectively.

Effect on CSO: We assumed that the timing delay for each context-switch corresponds to 5.24 μs [31], which represents the average value of a context-switch on a multicore system under typical workloads. To find the total delay caused by the context-switches in our experiment, we computed the number of context-switches for each run and then multiplied it with the delay due to a single context-switch (5.24μs [31]). Then we computed the average overheads for context-switches per time slot (in μs) for all the algorithms. Figure 3d illustrates the context-switch/migration overhead increases for all algorithms with an increase in the UF values. However, the rising curves are less steep for TARTS and TA-SS algorithms, as they schedule a lesser number of tasks, which is not desirable in a system. Also, TA-SS is based on the partitioned oriented scheduling approach. Although we applied deadline partitioning to make it semi-partitioned, it only allows migration at frame boundaries and not within frames. This leads to very low CSO values for TA-SS. It may be noted that although ETA-HP can schedule a higher number of tasks than the rest of the algorithms, it is still able to achieve CSO values on par with TARTS and HEARS, because the number of migrations within each frame is bounded.



(a) Effect of variation in | Π |



(b) Effect of variation in | τ |

Fig. 4: Synthetic Task Set Results (Effect on SRat values)

6.2.2 Synthetic Task Set Results

We have measured the variation in the number of cores ($| \Pi |$) and the number of tasks ($| \tau |$) on the SRat of the algorithms on systems having a high utilisation factor of 0.9.

Effects on the number of cores ($| \Pi |$): We varied the number of cores from 2 to 10 towards analysing the effects. As discussed before, TARTS is targeted for the homogeneous platforms, TA-SS only allows migrations at frame boundaries, whereas HEARS and ETA-HP do not allow simultaneous executions of the same task on multiple cores within a frame. As the number of cores increase while the utilisation value and the number of tasks remain constant, it signifies that the tasks will have longer execution duration within frames. It will lead to a situation where there will be more need for migrations and the algorithms have to carefully schedule migrating tasks to avoid simultaneous execution on more than one core. As it is not always possible, the SRat values decrease. Hence, at a high workload of 0.9, the SRat values for all algorithms decrease with an increase in $| \Pi |$ values, which can be observed in Figure 4a. However, with a better heuristic, ETA-HP is able to perform better than other algorithms. In particular, the SRat value decreases from 78% to 26%, 86% to 46%, 100% to 81% and 100% to 86% with an increase in $| \Pi |$ values from 2 to 10, for TARTS, TA-SS, HEARS and ETA-HP, respectively.
Effect of the number of tasks ($| \tau |$): We further varied the number of tasks from 20 to 100 on a system having a fixed set of 8 cores and shows the results in Figure 4b. We can observe from Figure 4b that SRat values increase progressively for all algorithms while increasing the number of tasks. This phenomenon may be attributed to the fact that an increase in the number of tasks while having a constant number of cores and fixed workload, leads to a decrease in the average execution requirements of the individual tasks, that further results in a decrease in the number of migrations for the strategies. Hence, all the strategies are able to have better SRat values with an increase in $| \tau |$ values. In particular, the SRat value increases from 35% to 55%, 53% to 69%, 87% to 98% and 90% to 100% with an increase in $| \tau |$ values from 20 to 100, for TARTS, TA-SS, HEARS and ETA-HP, respectively.

7 Discussion

A critical factor affecting the performance of ETA-HP is the size of frames, which may vary over a wide range. We have used a dynamic deadline partitioning oriented approach, based on the nearest deadline among a set of active tasks at any given time, to determine the frames. A potential disadvantage of such an approach is that the size of a frame could be very small (large) if consecutive deadlines are very close to (far apart from) each other in time. If a frame is too small, the process for deciding which task should execute in the frame, as well as what share values should the included task receive, becomes considerably more involved. Additionally, as the major overhead incurred by ETA-HP is involved with the generation of the next frame's schedule at each frame boundary, the overall scheduling overhead associated with ETA-HP increases significantly when the average frame sizes are very small (making the number of frames large). On the other hand, as the admission control for a new task can only occur at frame boundaries, handling dynamic tasks become considerably more challenging when frame sizes are very large. This is because a new task may potentially arrive at any time in the middle of a frame and is only considered for admission at the end of the frame, which may be considerably later than its actual arrival time.

Due to the nature of the ETA-HP algorithm, a lower bound on frame sizes will lead to an obvious reduction in scheduling overheads. However, as frames are defined by all deadlines in the system, a natural tradeoff with such a lower bound is that the system now has to cope with bounded deadline violations. An upper bound on frame sizes, on the other hand, will lead to improvement in handling of dynamic task arrivals. However, a consequent tradeoff here is the increase in scheduling overhead. Detailed experimental analysis on the performance, timing and overheads associated with ETA-HP when frame sizes can vary within a fixed range of values, has not been performed as part of the current work, and will be taken up as future work.

8 Conclusion

With the technological advancement in modern embedded systems, the designers of the schedulers for these systems are faced with several challenges like performance, cost etc. In this work, we have presented a semi-partitioned heuristic scheduling strategy, ETA-HP, that performs energy and temperature-aware task scheduling on heterogeneous multicore platforms while providing an efficient utilisation of the resources. The overall working of the proposed scheduler is divided into the following four phases: task-partitioning, scheduling, temperature management, and energy management. Our experimental analysis shows that ETA-HP is not only able to improve success ratios for the task sets compared to HEARS [8] (2.52% on an average) but also improves average energy (7.29%) and reduces the average temperature of the underlying multicore by 9.59 °C.

Acknowledgement

This work is funded by Marie Curie Individual Fellowship (MSCA-IF), EU (Grant Number 898296).

References

[1] Saranya, N., Hansdah, R.C.: Dynamic partitioning based scheduling of real-time tasks in multicore processors. In: IEEE International Symposium on Real-Time Distributed Computing, pp. 190–197 (2015)

[2] Hoogeveen, J.A., van de Velde, S.L., Veltman, B.: Complexity of scheduling multiprocessor tasks with prespecified processor allocations. Discrete Applied Mathematics **55**(3), 259–272 (1994)

[3] Moulik, S., Sarkar, A., Kapoor, H.K.: DPFair scheduling with slowdown and suspension. In: International Conference on VLSI Design, pp. 43–48 (2018)

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

[5] Bertout, A., Goossens, J., Grolleau, E., Poczekajlo, X.: Workload assignment for global real-time scheduling on unrelated multicore platforms. In: Proceedings of the 28th International Conference on Real-Time Networks and Systems. RTNS 2020, pp. 139–148 (2020)

[6] Tafidis, P., Bandeira, J.: Interregional european cooperation platform to promote sustainable transport through ICT: An overview of best practices. In: Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments, pp. 255–260 (2017)

[7] Yeh, L.-T.L.-T.: Thermal management of microelectronic equipment : heat transfer theory, analysis methods and design practices. American Society of Mechanical Engineers (2002)

[8] Moulik, S., Chaudhary, R., Das, Z.: HEARS: A heterogeneous energy-aware real-time scheduler. Microprocessors and Microsystems **72**, 102939 (2020)

[9] Fatima, S., Vishwanath, V.M.: A heterogeneous dynamic scheduling minimized make-span for energy and performance balancing. In: Second International Conference on Advances in Electronics, Computers and Communications, pp. 1–7 (2018)

[10] Chau, V., Chu, X., Liu, H., Leung, Y.-W.: Energy efficient job scheduling with DVFS for CPU-GPU heterogeneous systems. In: Proceedings of the Eighth International Conference on Future Energy Systems, pp. 1–11 (2017)

[11] Sajid, M., Raza, Z.: Energy-efficient quantum-inspired stochastic Q-HypE algorithm for batch-of-stochastic-tasks on heterogeneous DVFS-enabled processors. Concurrency and Computation: Practice and Experience **31**(20), 5327 (2019)

[12] Moulik, S., Das, Z., Saikia, G.: CEAT: a cluster based energy aware scheduler for real-time heterogeneous systems. In: 2020 IEEE SMC, pp.

1815–1821 (2020)

[13] Tang, T.-C., Chen, Y.-S.: Thermal-aware mapreduce real-time scheduling in heterogeneous server systems. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, pp. 207–212 (2016)

[14] Chwa, H.S., Seo, J., Lee, J., Shin, I.: Optimal real-time scheduling on two-type heterogeneous multicore platforms. In: IEEE Real-Time Systems Symposium, pp. 119–129 (2015)

[15] Lee, Y., Shin, K.G., Chwa, H.S.: Thermal-aware scheduling for integrated cpus–gpu platforms. ACM Trans. Embed. Comput. Syst. **18**(5s) (2019)

[16] Alsafrjalani, M.H., Adegbija, T.: TaSaT: Thermal-aware scheduling and tuning algorithm for heterogeneous and configurable embedded systems. In: Great Lakes Symposium on VLSI, pp. 75–80 (2018)

[17] Cao, K., Zhou, J., Yin, M., Wei, T., Chen, M.: Static thermal-aware task assignment and scheduling for makespan minimization in heterogeneous real-time mpsocs. In: 2016 International Symposium on System and Software Reliability, pp. 111–118 (2016)

[18] Sharifi, S., Coskun, A.K., Rosing, T.S.: Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. In: 2010 15th Asia and South Pacific Design Automation Conference, pp. 873–878 (2010)

[19] Zhou, J., Wei, T., Chen, M., Yan, J., Hu, X.S., Ma, Y.: Thermal-aware task scheduling for energy minimization in heterogeneous real-time mpsoc systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**(8), 1269–1282 (2016)

[20] Li, T., Yu, G., Song, J.: Minimizing energy by thermal-aware task assignment and speed scaling in heterogeneous MPSoC systems. Journal of Systems Architecture **89** (2018). https://doi.org/10.1016/j.sysarc.2018.08.003

[21] Ahmed, R., Ramanathan, P., Saluja, K.K.: Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks under fluid scheduling model. ACM Trans. Embed. Comput. Syst. (2016)

[22] Wächter, E.W., de Bellefroid, C., Basireddy, K.R., Singh, A.K., Al-Hashimi, B.M., Merrett, G.: Predictive thermal management for energy-efficient execution of concurrent applications on heterogeneous multicores. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **27**(6), 1404–1415 (2019)

[23] Huang, H., Chaturvedi, V., Quan, G., Fan, J., Qiu, M.: Throughput maximization for periodic real-time systems under the maximal temperature constraint. ACM Trans. Embed. Comput. Syst. **13**(2s), 70–17022 (2014)

[24] Moulik, S., Sarkar, A., Kapoor, H.K.: TARTS: A temperature-aware real-time deadline-partitioned fair scheduler. Journal of Systems Architecture, 101847 (2020)

[25] Liu, S., Qiu, M., Gao, W., Tang, X.-j., Guo, B.: Hybrid of job sequencing and dvfs for peak temperature reduction with nondeterministic applications. In: Proceedings of IEEE International Conference on Computer and Information Technology. CIT '10, pp. 1780–1787 (2010)

[26] Bienia, C., Kumar, S., Singh, J.P., Li, K.: The parsec benchmark suite: Characterization and architectural implications (TR-811-08) (2008)

[27] Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., *et al.*: The gem5 simulator. ACM SIGARCH Computer Architecture News **39**(2), 1–7 (2011)

[28] Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In: MICRO, pp. 469–480 (2009)

[29] Zhang, R., Stan, M.R., Skadron, K.: HotSpot 6.0: Validation, acceleration and extension. In: University of Virginia, Tech. Report CS-2015-04 (2015)

[30] Bygde, S., Ermedahl, A., Lisper, B.: An efficient algorithm for parametric WCET calculation. In: IEEE RTCSA, pp. 13–21 (2009)

[31] Bastoni, A., Brandenburg, B., Anderson, J.: Cache-related preemption and migration delays: Empirical approximation and impact on schedulability, vol. 10, pp. 33–44 (2010)