An interplay of energy and temperature minimization techniques for heterogeneous multiprocessor systems

Yanshul Sharma
IIIT Guwahati, India
yanshul.sharma@iiitg.ac.in

Swati Gupta
IIIT Guwahati, India
swati.gupta21b@iiitg.ac.in

Sanjay Moulik
IIIT Guwahati, India
sanjay@iiitg.ac.in

Abstract—**Real-time embedded systems are designed to perform specific functions in real-time, with a microcontroller, memory, and input/output devices. The scheduler is a critical component that manages resource allocation and schedules jobs based on priority and available resources. Multiprocessor platforms improve performance, scalability, redundancy, and flexibility, with different approaches to scheduling, such as global, partitioned, and semi-partitioned. Minimizing dynamic energy consumption and processor temperatures is essential for improving battery life and reliability and meeting power and thermal constraints in applications such as mobile devices, aerospace, and defense systems. There are many energy and temperature management techniques, but their effect on each other has not been studied in detail. Hence, we want to employ a few of those techniques and want to observe their impacts. In this work, we first propose a basic semi-partitioned scheduler for heterogeneous multiprocessor systems which supports the execution of real-time jobs. Then, we apply well-known energy and temperature minimization techniques over the proposed scheduler to study their impact on the system. To conduct our experiments, we use benchmark programs whose characteristics have been extracted using various simulators.**

Index Terms—**Real-time, Multiprocessor, Scheduler, Frequency Scaling, Bin Packing**

I. Introduction

A real-time embedded system is a computer system designed to perform specific functions in real-time, meaning that the system must respond to external events within a specific time frame. These systems are often used in industrial control systems, medical devices, and automotive systems. Real-time embedded systems typically consist of a microcontroller or microprocessor, memory, and various input/output devices. They are designed to handle specific jobs and operate under strict time constraints to deliver consistent, reliable performance in a time-critical environment.

A scheduler is a critical component in real-time embedded systems that allocates system resources and schedules jobs based on their priority and available resources. It plays a key role in ensuring time-critical operations are completed within their deadline, preventing any job from monopolizing system resources, providing predictable system behaviour, and preventing interference between jobs. The scheduler is responsible for deciding which job should be executed next, allowing the system to operate efficiently and reliably, even in environments with limited resources and time-critical operations. Without a scheduler, real-time embedded systems would be unable to operate predictably, efficiently, and safely, making them a crucial component for various applications such as industrial control systems, medical devices, and automotive systems.

Multiprocessor platforms have become increasingly important for real-time systems as they offer several benefits that improve performance, scalability, redundancy, and flexibility. Multiprocessor platforms allow multiple jobs to run simultaneously on separate processors, reducing overall response time and increasing throughput. Additional processors can be added as needed to handle additional jobs, ensuring the system can keep up with increasing workloads. Critical jobs can be executed on multiple processors, ensuring the system can continue operating even if one or more processors fail. Dynamic resource allocation ensures that the most critical jobs are given priority and that resources are used efficiently. Overall, multiprocessor platforms enable real-time systems to operate reliably and efficiently, making them ideal for use in a wide range of applications. There are several different approaches to multiprocessor scheduling, including global, partitioned, and semi-partitioned scheduling.

Global scheduling [1] involves treating all processors as a single pool of resources, with jobs being assigned to any available processor based on their priority and the available resources. This approach offers good performance, but it may not be suitable for systems with critical jobs, as there is no guarantee that a job will be assigned to a specific processor. Partitioned scheduling [2] involves dividing the set of processors into separate partitions, with each partition assigned to a specific job or group of jobs. This approach ensures that critical jobs are assigned to specific processors, ensuring predictable performance and reducing the risk of interference between jobs. Semi-partitioned multiprocessor scheduling [3], [4] is a hybrid approach that combines the benefits of global and partitioned scheduling. It offers several advantages over these two approaches, making it a popular choice for real-time systems. One of the key advantages of semi-partitioned scheduling is its ability to provide a balance between predictability and performance. In this scheduling approach, the timeline may be divided into chunks, and in each chunk of time slots, the progress of every job is monitored. There is another variation of the semi-partitioned approach whereby reserving some processors for critical jobs while allowing non-critical jobs to be assigned to any available processor. The scheduler ensures that critical jobs are assigned to specific processors while still providing the flexibility to optimize performance by using any available processor for non-critical jobs.

Minimizing dynamic energy consumption and processor temperatures is critical for real-time embedded systems for several reasons [5], [6]. Firstly, reducing dynamic energy consumption is essential to maximize the battery life of mobile and portable devices, such as smartphones, tablets, and wearables [7], [8]. These devices are typically powered by batteries that have a limited capacity, and reducing dynamic energy consumption can significantly extend the battery life, improving the user experience and reducing the need for frequent charging. Secondly, reducing processor temperatures is critical to ensure the reliability and longevity of the processor. High temperatures can cause damage to the processor, leading to system failures and reduced lifespan [9], [10]. By minimizing processor temperatures, real-time embedded systems can operate more reliably and have a longer lifespan. Thirdly, minimizing dynamic energy consumption and processor temperatures is essential for applications that have strict thermal and power constraints, such as aerospace and defense systems. These systems must operate reliably in harsh environments and have limited power and cooling resources, making it essential to minimize energy consumption and processor temperatures to ensure optimal performance and reliability. Overall, minimizing dynamic energy consumption and processor temperatures is essential for real-time embedded systems to improve battery life, ensure reliability and longevity, and meet strict power and thermal constraints.

Contributions: Although there is a lot of work in literature which addresses energy and temperature management. However, most of them are targeted towards homogeneous multicore platforms. In addition, there are many techniques for these optimizations, but their effect on each other has not been studied in detail. Hence, we want to employ a few of those techniques and want to observe their impacts. The following are the contributions of the proposed work:

1) We propose a basic scheduler for heterogeneous multiprocessor systems with a generic number of processor types. The execution requirement of each job varies on the processors of such systems, which makes scheduling on such systems very challenging.
2) We extract job characteristics from Parsec benchmark programs using gem5 simulator [11], McPAT [12], and HotSpot [13] simulators.
3) We employ various techniques over the proposed scheduler to make it energy and temperature cognizant.
4) We compare the performances of these strategies under various system configurations.

II. Background

The energy consumption and processor temperatures can be minimized at various levels in the system. However, in our work, we are only concerned about the techniques which can be employed at the software level for the processors.

Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) are techniques employed in computer processors/processors to reduce energy consumption while maintaining performance [14], [15]. DVFS adjusts the voltage and clock frequency of a processor based on workload demands, striking a balance between performance and energy efficiency. By lowering the voltage and frequency during low workloads and increasing them during intensive jobs, DVFS optimizes power consumption. DPM extends beyond DVFS to encompass various strategies for managing power usage throughout the system. It involves techniques such as power gating, where idle components are turned off or put into low-power states, and clock gating, which selectively stops the clock signal to inactive parts of the processor. Job scheduling algorithms intelligently distribute workloads among processors, minimizing the need for all processors to be active simultaneously. Together, DVFS and DPM optimize power consumption by adapting to workload variations and performance needs, resulting in improved energy efficiency without sacrificing performance. These techniques play a crucial role in reducing energy consumption in modern processors and systems.

When a processor enters sleep mode, it consumes significantly less power compared to its active state. However, there is an overhead involved in transitioning between sleep and active states. This overhead includes the time it takes to wake up the processor, restore its state, and resume normal operation. The *break-even time* [14] can be defined as the duration it takes for the power savings achieved during sleep mode to compensate for the power consumed during the transition to and from sleep mode. In other words, it's the point where the energy savings from being in a low-power state surpass the energy expended during the transition. To optimize power management, system designers and developers aim to balance the time spent in sleep modes with the energy savings achieved and the responsiveness required for the system to wake up and resume its jobs.

Combining idle time slots and DVFS can be a powerful approach to controlling processor temperatures also. DVFS can be employed to reduce voltage and frequency during heavy workloads, which mitigates heat generation. Further, as stated before, each job has its own temperature characteristics. In embedded systems, most of the jobs which need execution are known beforehand. Hence, they may be studied and based on it, intelligent schedulers may be designed to reduce energy consumption.

III. Specifications

A. System Model

We consider a job set $J = \{J^1, J^2, \ldots, J^n\}$ having n periodic jobs and a heterogeneous multiprocessor platform $P = \{P^1, P^2, \ldots, P^m\}$ having m processor types. For simplicity, we assume that each processor type has a single processor. Although, this assumption can be easily generalized by employing any optimal homogeneous scheduler for each processor type in the case of multiple processors. Each instance of a periodic job J^i is associated with a $(m+3)$ tuple, $\langle c^i, \tau^i_{ss}, d^i, r^{i1}, r^{i2}, \ldots, r^{im} \rangle$, where c^i, τ^i_{ss} and d^i are the execution requirement, steady state temperature, and the deadline (as well as the period) of J^i in the system and r^{ij}, is the rate of execution of J^i on P^j. Hence, each job J^i can further be related to the terms: u^i $(= c^i/d^i)$, which represents utilization of J^i in the system and, u^{ij} $(= c^i/(d^i \times r^{ij}))$, which represents utilization of J^i on a processor P^j. Each of the processors can operate at a discrete frequency level which is chosen from the normalized operating frequency set $\mu = \{\mu^1, \mu^2, \ldots, \mu^l\}$, where all the levels are having values between 0 (excluding) and 1 with $\mu^l = 1$. The processors can also be put in DPM mode, where they will not perform any operation.

B. Power Specifications

We have utilized the analytical processor energy model introduced in a previous work [5]. The overall power consumption (Γ) of a processor based on CMOS technology comprises two components: dynamic power consumption (γ_d) and static power consumption(γ_s).

$$\Gamma = \gamma_d + \gamma_s \tag{1}$$

The dynamic power consumption (γ_d) of CMOS circuits is determined by the following equation:

$$\gamma_d = C_{eff} \times V_{dd}^2 \times \mu^j \tag{2}$$

Here, V_{dd} represents the supply voltage, C_{eff} denotes the average switched capacitance per cycle, and μ^j indicates the clock frequency. The primary contributors to the static current in a standard inverter arise from reverse bias junction current and sub-threshold conduction. As a result, the static power consumption (γ_s) can be calculated using the following expression:

$$\gamma_s = (V_{dd} \times I_{subn} + |V_{bs}| \times I)L_g \tag{3}$$

In this equation, V_{bs} represents the body bias voltage, I_{subn} denotes the sub-threshold leakage current, I signifies the reverse bias junction current in the NMOS device, and L_g represents the number of devices in the circuit. Using this energy model, we can determine the corresponding clock frequency (μ^j), dynamic power (γ_d), and static power (γ_s) for different voltage levels.

C. Thermal Model

For an interval $[t_0, t_e]$ in which J^i is executing on the processor P^j, if the processor 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 [16]:

$$\Gamma_e = \tau_{ss}^{ij} + (\Gamma_0 - \tau_{ss}^{ij})e^{-B(t_e - t_0)} \tag{4}$$

where B is a constant depending upon power consumption in the system. τ_{ss}^{ij} represents the steady state temperature of job J^i, which relies on the overall power consumption of J^i. The steady state temperature of a job refers to the temperature reached by a processor when a job runs continuously on it for a significant duration, possibly spanning multiple instances. By reducing the operating frequency of a processor, the rate at which the temperature changes decreases, thereby resulting in a lower steady state temperature for the job. To establish the relationship between the operating frequency of a heterogeneous multiprocessor platform and the steady state temperature of a job, we employed the following equation [16]:

$$\tau_{ss}^{ij}(k) = \beta * \mu^k * \tau_{ss}^{ij}(\mu^l) \tag{5}$$

where β is a constant, $\tau_{ss}^{ij}(k)$ denote the steady state temperature of J^i on P^j at k^{th} level of operating frequency.

IV. The Basic Scheduler

The proposed hierarchical scheduler comprises three phases. The overall progress of individual jobs is monitored in *Phase-1*. The algorithm performs the job-to-processor assignment in *Phase-2* along with preparation on the initial schedule. It is done for jobs which can be fully assigned on single processors. In *Phase-3*, only those jobs are assigned across multiple processors, which the algorithm was unable to do in the previous phase.

Algorithm 1: BASIC-SCHEDULE

```
Input: $J$, $P$
Output: Final Schedule
1  while true do
       // Phase-1: Basic Computations
2      Find next window $W_k$ using Equation 6
3      Let $S_k[n \times m]$ be the Schedule Matrix for $W_k$
4      Let $LT_1$ and $LT_2$ be sorted lists of jobs based on their
         workload in $W_k$
5      for $i = 1 : n$ do
6          for $j = 1 : m$ do
7              Compute $q_k^{ij}$ using Equation 7
8              $LT_1 = LT_1 \cup \{\langle i, j, q_k^{ij} \rangle\}$
       // Phase-2: Basic Job Assignment and
          Schedule Preparation
9      while $LT_1$ is not empty do
10         Get the first element of $LT_1$: $\langle i, j, q_k^{ij} \rangle$
11         if $q_k^{ij}$ can be fully allocated on $P^j$ then
12             Assign start and end times of $J^i$ on $P^j$, i.e., $S_k[i][j]$
                 = $\langle$start_time($J^i$), end_time($J^i$)$\rangle$
13             Remove all entries of $J^i$ from $LT_1$ and $LT_2$
14         else
15             $LT_2 = LT_2 \cup \{\langle i, j, q_k^{ij} \rangle\}$
       // Phase-3: Migrating Job Assignment and
          Schedule Preparation
16     while $LT_2$ is not empty do
17         Let $LT_3$ be an empty list of jobs
18         Get the first entry $\langle i, j, q_k^{ij} \rangle$ from $LT_2$
19         Move all entries of $J^i$ from $LT_2$ to $LT_3$
20         Starting from the first node of $LT_3$, assign $J^i$ on
             processors using Next-Fit Bin Packing Algorithm
```

A. Description of the basic scheduler

The algorithm called BASIC-SCHEDULE, is designed to generate a schedule for a set of jobs and processors. It begins with the inputs J (the set of jobs) and P (the set of processors) and aims to produce the final schedule as output. The algorithm operates within an infinite loop, indicating that it will continue scheduling jobs indefinitely. Each iteration of the loop performs several steps to generate a schedule. The pseudo-code for the basic scheduler used in our work is presented in Algorithm 1. In the following section, we elaborate on the phases of the algorithm.

- **Phase-1:** In this phase, the overall progress of the jobs is monitored. To do so, the algorithm breaks the timeline into multiple chunks of time slots using the technique of *Deadline Partitioning* [17], which are called *windows*. The use of such a technique allows the scheduler to keep track of jobs' progress at the end of every window. To compute the next window W_k, the scheduler finds the minimum of all job deadlines:

$$|W_k| = min\{d^1, d^2, \ldots, d^n\} \tag{6}$$

  Next, the workload or quota of all individual jobs J^i on a processor P^j in the current window are computed as:

$$q_k^{ij} = u^{ij} \times |W_k| \tag{7}$$

  If all jobs complete the execution of their computed quota in the window, there will be no lag for any job. Hence, boundaries of individual windows also act as mini-deadlines. The algorithm initializes a schedule matrix $S_k[n \times m]$ that will hold the schedule for the jobs within W_k. The matrix is initially empty, with all its entries set to $\emptyset$. Next, the algorithm creates two sorted job lists based on their quota values, LT_1 and LT_2, which will be used for job scheduling. LT_1 and LT_2 are initially empty. The algorithm iterates through all the jobs in J and processors in P, calculating and storing the share values in LT_1.

- **Phase-2:** Moving to the second phase, the algorithm enters a loop that continues until LT_1 becomes empty. Within this loop, the algorithm processes the jobs in LT_1 one by one. For each job J^i in LT_1, the algorithm checks if the entire quota q_k^{ij} of the job can be allocated to a single processor P^j. If it is feasible, the algorithm assigns the start and end times of the job on the assigned processor, recorded as $S_k[i][j]$. It then removes the corresponding job entries from both LT_1 and LT_2, indicating that the job has been fully allocated. If the entire quota of a job cannot be allocated to a single processor, the job is added to LT_2 for further processing in subsequent iterations.
- **Phase-3:** In this phase, the algorithm continues processing jobs that were not fully allocated in the previous phase. Within a loop that runs until LT_2 becomes empty, the algorithm creates an empty list LT_3 to store jobs temporarily. The algorithm retrieves the first entry, denoted as $\langle i, j, q_k^{ij} \rangle$, from LT_2 and moves all entries of the corresponding job J^i from LT_2 to LT_3. It then proceeds to assign the jobs in LT_3 to available processors using the Next-Fit Bin Packing algorithm [18], starting from the first node of LT_3.

After completing the third phase, the algorithm returns to the beginning of the loop to repeat the process for the next window W_k. This loop continues indefinitely, allowing for the continual scheduling of jobs on processors until the final schedule is obtained.

B. Techniques Employed

We have used Algorithm 1 to prepare the basic assignments of jobs on available processors. Over the algorithm, we have employed various energy and temperature minimization techniques which are used at the processor level to propose the following strategies:

- **Strategy 1:** In this approach, we have put the processors in sleep mode whenever idle time slots are available on processors. It may be noted that we have considered two time slots as the break-even time (refer to Section II). During the idle time slots, the processors in sleep mode not only save energy but also cools down.
- **Strategy 2:** We have segregated the jobs as hot and cold jobs in this approach [19]. We computed the average steady state temperature of the input job set on a reference processor (say P^j) and then considered individual jobs one by one. If a job J^i is having its τ_{ss}^{ij} less than the computed average steady state temperature of the job set, it is classified as a cold job, or else a hot job. After the segregation process is over and the jobs have been assigned to individual processors, we rearrange the job execution of processors in such a way that every hot job is followed by a cold job (if available). It is a well-known temperature management technique for processors which keeps the processor temperatures balanced [20]–[22].
- **Strategy 3:** We have employed DVFS on individual processors. Based on the workload assigned to individual processors by Algorithm 1, we have scaled the voltage/frequency such that there is no deadline miss for any job in a window, and the workload gets completed. As stated in Section II, DVFS not only helps in reducing energy consumption in the system by reducing operating voltage/frequency on individual processors but also reduces steady state temperatures of individual jobs (refer Equation 5). It ultimately leads to reduced processor temperature also [6], [16].
- **Strategy 4:** In this approach, we have used both DVFS and job resequencing based on temperature characteristics, i.e. we have used Strategy 2 along with Strategy 3. Thus it is a hybrid approach which uses two techniques together.

V. Experimental Set Up and Results

In this section, we discuss the configurations used for our experiments, followed by a discussion on the performance of various strategies (refer to Section IV-B).

Job	$\langle Requirement, Temp. \rangle$ (in ms and $^\circ C$)	Job	$\langle Requirement, Temp. \rangle$ (in ms and $^\circ C$)
x264	$\langle 1203, 85 \rangle$	Body	$\langle 3824, 85 \rangle$
Canneal	$\langle 1007, 80 \rangle$	Swap	$\langle 4535, 76 \rangle$
Dedup	$\langle 6455, 91 \rangle$	Stream	$\langle 6156, 68 \rangle$
Freq	$\langle 11082, 84 \rangle$	Fluid	$\langle 4090, 81 \rangle$

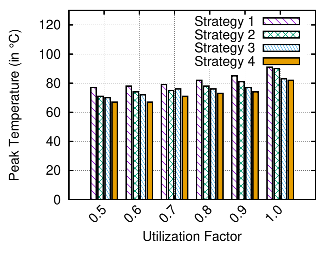
TABLE I: Job Specifications for Benchmark

A. Experimental Set Up

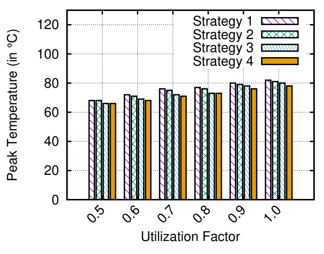
We performed simulations for a duration of one million time slots to evaluate our algorithms. The job sets used in the simulations were assigned predefined utilization factors (UF). The UF is calculated as: $UF = \frac{\sum_{i=1}^{n} avg_{j=1}^{m}(u^{ij})}{m}$. To generate job sets with specific UF values, we appropriately scaled the randomly generated utilization values. For each set of input parameters, we conducted 50 simulation runs on different test cases and considered the average of these runs as the final outcome.

To analyze the performance of our algorithms in real-life scenarios, we utilized jobs from the PARSEC benchmark [23] running on systems with four types of processors. The measurement of execution requirements and steady state temperatures of each job can be found in [24], and the corresponding results are listed in Table I. These values are obtained using the following simulators: gem5 simulator [11], McPAT [12], and HotSpot [13]. The configurations are an ARMv8 processor operating at 3.0 GHz with 32 nm CMOS technology. The steady state temperatures (τ_k^{ij}) are randomly varied on different processors within the range of $\tau_k^{ij} \pm \alpha$, where α ranges from 0% to 10%.

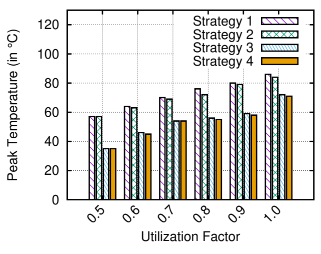
Each job set consisted of 20 jobs randomly selected from the eight benchmark jobs listed in Table I. The operating frequencies used in our experiments vary from 0.9 GHz to 3.0 GHz. For each job, a randomly chosen β value (used in Equation 5) ranging from 1 to 1.25 was assigned to calculate the steady state temperatures (τ_k^{ij}) for that specific job.



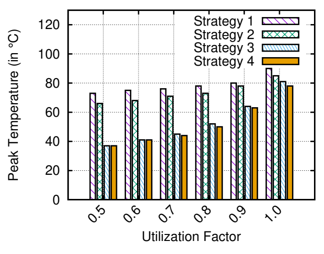
(a) Processor 1



(b) Processor 2



(c) Processor 3



(d) Processor 4

Fig. 1: Effect of Utilization Factor

B. *Experimental Results*

We compared the performance of Strategy 1, Strategy 2, Strategy 3, and Strategy 4 (refer Section IV-B) under various system configurations. The results of the experiments are discussed in this section.

1) Effect of Utilization Factors: In this experiment, the UF values range from 0.5 to 1.0 while the number of processors and the number of jobs remain at 4 and 20, respectively. The average steady state temperature for the job set is kept at $70°C$. By referring to Figure 1, we can infer that the peak temperature rises in proportion to the workload in the system. This phenomenon can be noticed for all four processors in the system. Further, this feature is true for all the concerned strategies. It may be attributed to the fact that at lower UF values, the number of idle time slots in a processor is more, which the concerned strategies use to cool down the processors. However, Strategy 2 fetches better results than Strategy 1 because it judiciously arranges jobs on processors based on job characteristics along with using idle time slots by allowing processors to enter sleep modes. Strategy 3 uses idle time slots to scale down the voltage/frequency, which further improves the results. However, Strategy 4 uses DVFS along with intelligent job sequencing to fetch the best results. This phenomenon can also be inferred from Figure 2, where we plotted the average temperature for the processors. In particular, the average temperatures of the processors rise from $52°C$ to $67°C$, $51°C$ to $66°C$, $37°C$ to $66°C$, and $37°C$ to $65°C$ for Strategy 1, Strategy 2, Strategy 3, and Strategy 4, respectively with the rise in UF values for the system.

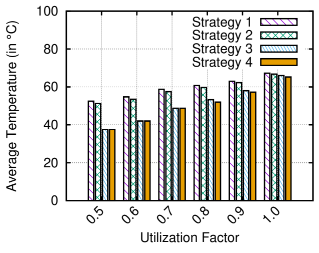


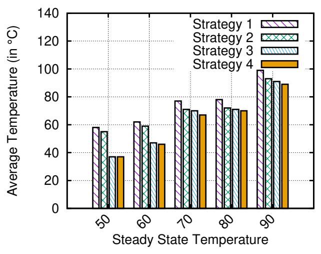
Fig. 2: Effect of Utilization Factor on Average Temperature of Processors

2) Effect of Steady State Temperatures: In this experiment, the average steady state temperature of the job set range from $50°C$ to $90°C$ while the number of processors and the number of jobs remain at 4 and 20, respectively. The UF value for the system is kept at 0.5. From Figure 3, we can notice that the peak temperature rises for all the processors with the increase in average steady state temperature. As stated in Section II, the steady state temperature is the temperature which is attained on a processor if a job is run on it for an infinite duration. Since the experiments are run for a long duration of 10 million time slots, the processors are able to reach the steady state temperature values easily. However, as we can notice from Figure 4, the average temperature of the processors is lowest for Strategy 4, which can be attributed to its intelligent job sequencing and voltage/frequency scaling. In particular, the average temperatures of the processors rise from $39°C$ to $66°C$, $38°C$ to $65°C$, $24°C$ to $55°C$, and $24°C$ to $54°C$ for Strategy 1, Strategy 2, Strategy 3, and Strategy 4, respectively with the rise in steady state temperature of the job set.

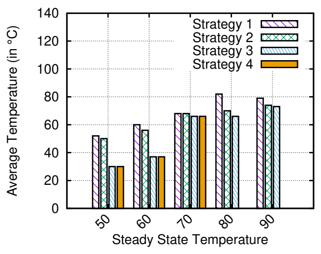
3) Effect on Normalized Energy Consumption: In this experiment, the average steady state temperature of the job set is kept at $70°C$, m at 4 and n at 20. The UF varied from 0.5 to 1.0. Since job sequence does not have any effect on the energy consumption of the system, Strategy 1 and Strategy 2 fetch similar results (refer Figure 5). This is also true for Strategy 3 and Strategy 4, which fare similar results. Hence, we only plotted the results of Strategy 1 and Strategy 3. In Strategy 1, the processors are put in sleep mode to conserve energy, whereas Strategy 3 judiciously scales down voltage/frequency to do the same. Hence, both strategies are able to fetch good results at lower workloads. But when the UF rises, both these strategies get a lesser scope to apply the energy conservation techniques, and therefore, the energy consumption in the system rises. Further, it can be noticed that the difference in results of the concerned strategies is very less at $UF = 1.0$. However, Strategy 4 is able to fetch the overall best results.

VI. Conclusion

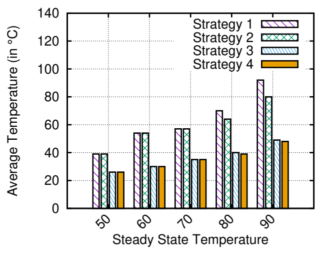
In real-time embedded systems, the scheduler is considered to be one of the most important parts of the system. With the new technological advancement, the schedulers are not only required to schedule jobs but reduce energy consumption and maintain processor temperatures within a threshold also. The problem becomes more significant when the system supports heterogeneous multiprocessors. In this paper, we propose a basic scheduler for such platforms. Further, we employ various well-known energy and temperature management techniques on top of the proposed scheduler. We extracted job characteristics of various Parsec benchmark programs using the following simulators: gem5, McPAT, and HotSpot. Then we provided them as inputs to our proposed strategies. Through our experiments, we demonstrated that the DVFS based strategy,



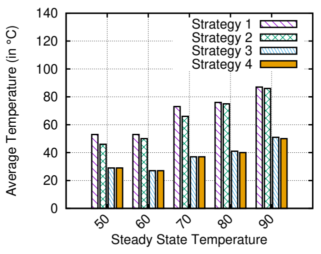
(a) Processor 1



(b) Processor 2



(c) Processor 3



(d) Processor 4

Fig. 3: Effect of Steady State Temperature

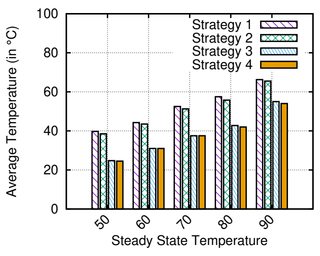


Fig. 4: Effect of Steady State Temperature on Average Temperature of Processors

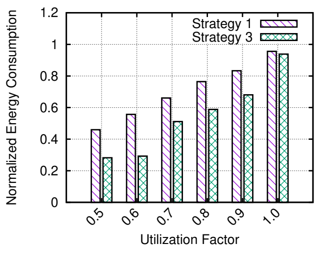


Fig. 5: Effect of Utilization Factor on Normalized Energy Consumption

which intelligently prepared the job sequence by alternatively executing hot and cool jobs, fetched the best result.

REFERENCES

[1] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in *2009 30th IEEE Real-Time Systems Symposium*, 2009, pp. 469–478.

[2] Q. Zhao, M. Qu, Z. Gu, and H. Zeng, "Minimizing stack memory for partitioned mixed-criticality scheduling on multiprocessor platforms," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 2, mar 2022.

[3] L. Zeng, Y. Lei, and Y. Li, "A semi-partition algorithm for mixed-criticality tasks in multiprocessor platform," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, 2019, pp. 694–697.

[4] S. Moulik, R. Devaraj, A. Sarkar, and A. Shaw, "A deadline-partition oriented heterogeneous multi-core scheduler for periodic tasks," in *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2017, pp. 204–210.

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

[6] Y. Sharma, S. Chakraborty, and S. Moulik, "ETA-HP: an energy and temperature-aware real-time scheduler for heterogeneous platforms," *The Journal of Supercomputing*, vol. 78, no. 8, pp. 1–25, 2022.

[7] P. K. D. Pramanik, N. Sinhababu, B. Mukherjee, S. Padmanaban, A. Maity, B. K. Upadhyaya, J. B. Holm-Nielsen, and P. Choudhury, "Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage," *IEEE Access*, vol. 7, pp. 182 113–182 172, 2019.

[8] M. Tawalbeh, A. Eardley, and L. Tawalbeh, "Studying the energy consumption in mobile devices," *Procedia Computer Science*, vol. 94, pp. 183–189, 2016, the 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops.

[9] M. Arbabzadeh, G. M. Lewis, and G. A. Keoleian, "Green principles for responsible battery management in mobile applications," *Journal of Energy Storage*, vol. 24, p. 100779, 2019.

[10] S. L R, A. A, M. S. Ramkumar, G. Emayavaramban, K. Balachander, and P. Naveni, "Battery management system with iot for enhancement of battery life," in *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2021, pp. 1743–1750.

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

[12] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, Dec 2009, pp. 469–480.

[13] M. R. Stan, R. Zhang, and K. Skadron, "Hotspot 6.0: Validation, acceleration and extension," 2015.

[14] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, mar 2014.

[15] M. E. T. Gerards and J. Kuper, "Optimal dpm and dvfs for frame-based real-time systems," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, jan 2013.

[16] Y. Sharma and S. Moulik, "Cetas: A cluster based energy and temperature efficient real-time scheduler for heterogeneous platforms," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 501–509.

[17] J. Yang, X. Luo, and X. Long, "A discrete dp-wrap scheduling algorithm for multiprocessor systems," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, 2015, pp. 958–962.

[18] C. C. Lee and D. T. Lee, "A simple on-line bin-packing algorithm," *J. ACM*, vol. 32, no. 3, p. 562–572, jul 1985.

[19] S. Moulik and Z. Das, "TASOR: A Temperature-Aware Semi-Partitioned Real-time Scheduler," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 1578–1583.

[20] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 618–623.

[21] Y. Lee, K. G. Shin, and H. S. Chwa, "Thermal-aware scheduling for integrated cpus–gpu platforms," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, oct 2019.

[22] Y. Lee, H. S. Chwa, K. G. Shin, and S. Wang, "Thermal-aware resource management for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2857–2868, 2018.

[23] D. Chasapis, M. Casas, M. Moretó, R. Vidal, E. Ayguadé, J. Labarta, and M. Valero, "Parsecss: Evaluating the impact of task parallelism in the parsec benchmark suite," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, dec 2015.

[24] S. Bygde, A. Ermedahl, and B. Lisper, "An efficient algorithm for parametric WCET calculation," in *2009 IEEE RTCSA*, Aug 2009, pp. 13–21.