**ADVANCED TASK MAPPING AND SCHEDULING ALGORITHM FOR ENHANCED ENERGY EFFICIENCY ON MULTI-CORE EMBEDDED PLATFORMS**

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

**B. Tech**

**in**

**Electronics and Communication Engineering**

By

**G R Jeshnu Skandan (108121041)**

**Saravanakrishnan B (108121113)**

****

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**NATIONAL INSTITUTEOF TECHNOLOGY**

**TIRUCHIRAPALLI-620015**

**MAY 2024**

**ADVANCED TASK MAPPING AND SCHEDULING ALGORITHM FOR ENHANCED ENERGY EFFICIENCY ON MULTI-CORE EMBEDDED PLATFORMS**

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

**B. Tech**

**in**

**Electronics and Communication Engineering**

By

**G R Jeshnu Skandan (108121041)**

**Saravanakrishnan B (108121113)**

****

**ELECTRONICS AND COMMUNICATION ENGINEERING**

**NATIONAL INSTITUTEOF TECHNOLOGY**

**TIRUCHIRAPALLI-620015**

**MAY 2024**

**BONAFIDE CERTIFICATE**

This is to certify that the project titled ‘**Advanced Task Mapping and Scheduling Algorithm for Enhanced Energy Efficiency on Multi-Core Embedded Platforms**’ is a bonafide record of the work done by

**G R Jeshnu Skandan (108121041)**

**Saravanakrishnan B (108121113)**

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering** of the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI,** during the year 2024-2025.

**Dr. B. Naresh Kumar Reddy**  **Dr. M.Bhaskar**  
 Guide Head of the Department

Project Viva-voce held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Internal Examiner** **External Examiner**

**ABSTRACT**

Energy-efficient task scheduling has been an essential topic in recent times. Considering the complexity of AI and energy usage and the considerable task loads given to systems, efficient task scheduling plays a vital role in meeting deadlines and energy requirements. Energy consumption minimization is one of the essential requirements in scheduling tasks in heterogeneous multicore embedded systems. Dynamic Voltage and Frequency Scaling (DVFS) plays a significant role in energy consumption minimization, but the complexity of frequency allocation significantly affects the overall efficiency of the embedded system. Using techniques like DVFS helps to achieve better task scheduling, but the problem of task scheduling becomes an NP-Hard problem. Scheduling in heterogeneous systems involves mapping a task from the set (n1, n2,…, nN) to a processor U in set (u1, u2,…., uU) while implementing DVS to find the appropriate frequency for each of the tasks such that their timing constraints are met. The existing conventional scheduling techniques often consume more energy because of their high computation cost or sub-optimal frequency and task assignments. To address these problems, our work proposes a novel approach to assigning frequencies to each task and allocating tasks to various cores in a multicore processor. Our method introduces a less complex yet energy-efficient frequency assignment and task allocation strategy. The Frequency Assignment (FA) algorithm uses the binary search for frequency selection, which reduces the computational complexity to O (N \* log L), where N is the number of tasks and L represents the frequency levels. This guarantees that the frequency is allocated to each task optimally and consumes less energy. For task allocation, we proposed a Task Assignment (TA) algorithm based on Rank and Earliest finish time (EFT), which ensures that the tasks are assigned to available processor cores optimally while minimizing the overall execution time of the processor. This strategy minimizes energy consumption, distributes the workload evenly, and efficiently uses the available processing power. We compare our solution with other state-of-the-art algorithms to evaluate performance in various applications like Gaussian Elimination (GE) and a random task graph. Our numerical results demonstrate that the proposed scheduling algorithms perform significantly higher than the traditional state-of-the-art algorithms in terms of energy savings, task execution efficiency, and reduced computation complexity. The results showed that our method provides an efficient trade-off between complexity and performance, making it a good solution for the next-generation energy-aware embedded systems. The proposed work is implemented in Verilog on the Zynq Ultrascale+ MPSoc ZCU106 Evaluation Kit FPGA platform, and its performance has been validated.

Key words: Energy-efficient task scheduling, Dynamic Voltage and Frequency Scaling (DVFS), Heterogeneous multicore embedded systems, Task Assignment algorithm.

**ACKNOWLEDGEMENT**

We would like to express our heartfelt gratitude to all those who have supported and guided us throughout the course of this project. Without their valuable input, encouragement, and assistance, the successful completion of this work would not have been possible.

First and foremost, we extend our sincere thanks to **Dr. B. Naresh Kumar Reddy**, our project guide, for his constant support, insightful feedback, and guidance throughout the project. His mentorship was instrumental in shaping our ideas and steering our efforts in the right direction.

We would also like to acknowledge our internal reviewers—**Dr. G. Lakshminarayanan**, and **Dr. V. Sudha**—for their valuable suggestions and constructive feedback during the review sessions, which helped us refine and improve our project.

Our sincere appreciation also goes to the **Department of Electronics and Communication Engineering, NIT Trichy**, and the ECE Lab facilities for providing us with the resources and environment necessary to carry out our research and development activities effectively.

Last but not least, we thank our parents and friends for their unwavering encouragement, emotional support, and belief in us throughout this journey.

**TABLE OF CONTENTS**

**Title** Page No.

**ABSTRACT** ........................................................................................................... iv

**ACKNOWLEDGEMENTS** ................................................................................... vi

**TABLE** **OF CONTENTS** ....................................................................................... vii

**LIST OF TABLES**................................................................................................... ix

**LIST OF FIGURES**................................................................................................. x

**CHAPTER 1 INTRODUCTION**

1.1 General Motivation ……………………………………………………….. 1

1.2 Our Contribution ………………………………………………………….. 2

**CHAPTER 2 LITERATURE REVIEW**

2.1 Related works …………………………………………………………………. 3

**CHAPTER 3 PRELIMINARIES AND PROBLEM FORMULATION**

3.1 System and Application Model ……………………………………………….. 6

3.2 Power and Energy Model ……………………………………………………... 9

3.3 Constraints …………………………………………………………………….. 10

3.4 Problem Description …………………………………………………………… 11

**CHAPTER 4 PROPOSED WORK**

4.1 Frequency Assignment …………………………………………………………. 12

4.2 Task Assignment ……………………………………………………………….. 14

**CHAPTER 5 SIMULATION RESULTS AND HARDWARE IMPLEMENTATION**

5.1 Vivado simulation output ………………………………………………………. 17

5.2 Complexity ……………………………………………………………………… 18

5.3 Energy …………………………………………………………………………… 19

5.3.1 Motivational Application ……………………………………………… 19

5.3.1 Gaussian Elimination Application …………………………………….. 23

5.4 Hardware Implementation ………………………………………………………. 27

**CHAPTER 6 SUMMARY AND CONCLUSION**

6.1 Summary ………………………………………………………………………… 28

6.2 Conclusion and Future Work ……………………………………………………. 29

**REFERENCES**

**LIST OF TABLES**

**Table No. Title Page No.**

3.1 WCETs of Tasks on Different Processors with the Maximum   
Frequencies of Motivating Parallel Application……………………………... 8

5.1. Complexities of different algorithms ……………………………………….. 18

5.3. Energy consumption at different deadlines ………………………………… 21

5.4. Reduction in energy consumption over other algorithms (%) ……………….. 21

5.5 GDES vs. Proposed algorithm using motivational example from Fig. 3.2 …. 22

5.6 ISAECC vs. Proposed algorithm using motivational example from Fig. 3.2.. 22

5.7. Task assignment using the proposed algorithm for the motivating example.. 23

5.8. Percentage reduction in energy consumption of proposed vs   
other algorithms for varying deadlines………………………………………. 25

5.9. ISAECC vs. Proposed algorithm using Gaussian Elimination application …. 26

5.10. GDES vs. Proposed algorithm using Gaussian Elimination application ……. 26

**LIST OF FIGURES**

**Figure No. Title Page No.**

3.1. Architecture of heterogeneous embedded system using CAN……………… 6

3.2. Motivating example of a DAG-based parallel application …………………. 8

4.1. Representation of the critical path ………………………………………….. 12

5.1. Waveform showing the assigned frequency and   
total energy (Vivado simulator) …………………………………………….. 17

5.2. Processor and frequency assignment output ………………………………... 17

5.3. Scheduling Gantt chart for the motivational application   
using the proposed algorithm ……………………………………………….. 19

5.4. Comparison of energy consumption for fixed deadline ……………………. 20

5.5. Comparison of energy consumption at different deadlines …………………. 21

5.6. DAG of the Gaussian Elimination application for m = 5 …………………… 23

5.7. Scheduling Gantt chart for the Gaussian elimination application   
using the proposed algorithm ……………………………………………….. 24

5.8. Comparison of energy consumption ………………………………………… 24

5.9 Comparison of energy consumption at different deadlines …………………. 25

**CHAPTER 1**

**INTRODUCTION**

**1.1 GENERAL MOTIVATION**

Computers have changed substantially over the last 70 years to satisfy the ever-increasing demands of information technology (IT) in both business and education. Because of their strong processing and data storage capacities, modern computing systems are crucial for facilitating scientific and industrial processes. But even though performance has significantly improved, power consumption has also increased significantly. This increase in energy usage leads to serious economic, environmental, and technical challenges. A common method to deal with this problem is power-aware software design. A popular method called Dynamic Voltage and Frequency Scaling (DVFS) adjusts the trade-off between performance and energy consumption in real time. This has led to a great deal of research on power-aware task scheduling on processors that can run at different frequencies and voltages.

Heterogeneous distributed systems are increasingly being deployed across a wide spectrum of platforms, ranging from small-scale embedded devices like laptops and smartphones to large-scale infrastructures such as grids, clusters, clouds, and service-oriented systems. The ever-increasing demand for devices with increased processing power has only increased the requirement of heterogeneous systems with NoCs to ensure the bottleneck does not affect the processing power. To allow for efficient usage of processing power, we need to make sure the task scheduling is done properly. It is also a requirement that the task scheduling be energy efficient as the remaining energy can be used for other applications in the NoC. By having energy-efficient tasks scheduled in a NoC, it is possible to achieve higher processing power that users demand. Significant research has been conducted in this area, and the primary method used is DVFS [1]. Dynamic Voltage and Frequency Scaling (DVFS) enables energy-efficient scheduling by simultaneously reducing a processor’s supply voltage and frequency. The frequency at which a task is executed is reduced such that the deadline is still met, allowing for reduced energy consumption while timing constraints are still met.

In real-time embedded systems, meeting deadlines is a critical requirement; failure to meet hard deadlines can result in system malfunction and potentially severe outcomes. Therefore, hard real-time applications must strictly adhere to their timing constraints. The number of parallel applications with precedence-constrained tasks—such as fast Fourier transform and Gaussian elimination—is increasing in heterogeneous distributed environments. These applications are commonly modeled using structures like directed acyclic graphs (DAGs), hybrid DAGs (HDAGs), and task interaction graphs (TIGs). In this context, the parallel application is represented using a DAG, where each node denotes a task and each edge signifies communication between tasks.

**1.2 OUR CONTRIBUTIONS**

1. For the frequency allocation problem, we use a novel binary search approach. By using binary search, we reduce the complexity of the greedy algorithm from O(N\*L) to O(N\*log L), where N is the number of tasks, and L is the number of frequency levels.
2. For the task assignment phase, we employ a rank calculation strategy to find the minimum Earliest Finish Time (EFT) and allocate the appropriate processor to the task.
3. The proposed work is compared with state-of-the-art works like ISAECC, GDES, and Greedy approach algorithms in simulation as well as hardware for energy consumption.
4. The comparison is done on the Zynq Ultrascale+ MPSoC board using applications like Gaussian Elimination (GE), and random task graph, and the proposed work has higher energy savings for a given application deadline compared to the other related works.

**LITERATURE REVIEW**

**2.1 RELATED WORKS**

Recently, much attention has been given to energy-efficient task scheduling in embedded systems. Many of the published works in the literature consider energy efficiency when it comes to frequency allocation and task scheduling. Biao Hu et al. [4] proposed a task assignment algorithm that uses DVFS to schedule the task optimally for different processors. In addition, it employs a task migration algorithm, which focuses on migrating the tasks in a particular processor to some other processor so that the processor can be deactivated to conserve energy. However, this technique leads to considerable computation complexity as the tasks must be reallocated continuously between the processors. Abhishek Mishra et al. [5] introduced the Energy-Efficient Load Scheduling Algorithm (EELSA), which minimizes energy consumption in multicore processors by optimally scheduling core speeds and voltages using Dynamic voltage scaling to complete tasks within a given deadline. The algorithm begins by assigning the largest possible energy consumption as an initial upper limit and then employing integer linear programming (ILP) to check the feasibility. Then, it uses a binary search to iteratively refine the energy constraint while employing ILP at each step to find the feasible solution and return it to minimize energy while meeting the deadline. Ernest Antolak et al. [6] combines FPGA-based implementation with heuristic scheduling to provide an energy-efficient scheduling method for real-time embedded systems. It mainly focuses on minimizing energy consumption while meeting deadline constraints. The proposed algorithms BLIS II, COTAS II, and STODER II allocate tasks to processors based on task frequency, energy constraints, and load balancing. Instead of analyzing the slack and deadline margins, the author introduced the task processing frequency (TF) parameter, which combines the complexity of a task and its deadline. This parameter makes the task mapping efficient and simpler and also guarantees the system's time predictability.

Huang et al. [7] proposed a variable weighted energy pre-assignment algorithm for the NP-Hard task scheduling problem. The focus was on minimizing the schedule length of the tasks given an energy constraint. Once an initial solution is found, a new suitable frequency is reached by recalculating the parameters so that energy is saved. This saved energy is then used to further minimize the schedule length of the application. Ultimately, this algorithm mainly prioritizes schedule length minimization over energy minimization. An algorithm for energy-aware task scheduling that considers energy efficiency, timing constraints, and probability guarantee is introduced in [8]. The authors suggested a dynamic programming model that reduces the search space for the task assignment problem, which solves only the probability constraints. Then, the timing and dependency of the tasks are considered. The final task-scheduling algorithm finds a balance between energy and timing constraints to make sure that an optimal solution is reached. ISAECC [9] introduces a weighted energy pre-assignment strategy that minimizes schedule length and focuses on reliability maximization. The algorithm makes use of the given energy constraint to minimize the schedule length as much as possible. The pre-assignment strategy is extended to develop a reliability maximization algorithm. In [10], two approaches are incorporated to find the optimal solution to task scheduling. A non-DVFS approach is combined with a global DVFS approach while considering energy and timing constraints. The non-DVFS approach is based on variable deadline slack for energy efficiency, while the global DVFS approach aims to find frequencies considering global energy efficiency task scheduling.

In [11], Xinmei Li et al. formulated an ILP problem for maximizing QoS. Two separate algorithms for frequency and task assignments in the multicore processor are proposed to solve the ILP. It used an optional cycle adjustment step which utilizes the idle time and remaining energy to execute optional subtasks to improve the Quality of Service (QoS). However, the FA algorithm makes use of a greedy approach to allocating frequencies to each task, which leads to significantly higher energy consumption. The greedy approach will also lead to a higher complexity of O(N\*L), which, in turn, will create a larger system overhead. This motivates us to come up with a FA and TA algorithm that makes use of binary search for frequency assignments and rank based approach that makes use of EFT for task allocation, which significantly reduces the energy consumption as well as the complexity of frequency assignments to O(N\*logL) as compared to O(N\*L) while using greedy approach.

**3.** **PRELIMINARIES AND PROBLEM FORMULATION**

**3.1 SYSTEM AND APPLICATION MODEL**

Our study considers a distributed embedded architecture with several processor cores in the same controller area network (CAN) bus, as given in fig 3.1. Each core contains a central processing unit, non-volatile memory and a random-access memory (RAM). In this architecture, a task that has been executed in one core can send a message to its successor task that has been running on different processor cores. For example, task n1 running in core u2 can send message m1,5 to its successor task n5 running in core u4, as shown in the fig. 3.1. Let U {u1; u2; . . . uU} represent a set of heterogeneous multiprocessors, where |U| represents the size of processor set U.

A diagram of a diagram

AI-generated content may be incorrect.

*Fig.3.1. Architecture of heterogeneous embedded system using CAN*

1. The set of nodes in the directed acyclic graph (DAG) is represented by N, where each node ni is an element of N that represents a task. The predecessor set of tasks ni is represented by pred(ni), whereas the successor set of tasks ni is represented by succ(ni). nentry denotes the set of tasks that do not have any predecessor, while nexit denotes the tasks that do not have any successor task.

2. During the analysis phase, the worst-case execution time (WCET) of each task running on different processors is determined [9]. The WCET is the time needed to execute the task in a processor at maximum frequency. As a result, when a task is being executed at a maximum frequency in a specific processor, its WCET is the maximum execution time among all the execution time values when run at maximum frequency. Since the processor is heterogeneous, each task will have different WCET values on different processors. We introduce a matrix W of size |N| x |U|, where wi,y denotes the worst case execution time of task ni running on the processor uy with maximum frequency.

3. The bus helps with passing the messages, i.e. communication between tasks mapped into different processors. We represent the Worst-case response time (WCRT) between task ni and nj as ci,j ∈ C, if tasks ni and nj are not assigned to the same processors. Therefore, when a message is passed between two tasks that are not mapped to the same processor, the WCRT is the maximum response time out of all the real-time values that could be possible. When two tasks run on the same processor, the communication time will be 0. All the Worst-case response times of the messages are determined during the analysis phase [10].

4. The deadline for application G is denoted as D(G), and it should be equal to or larger than the minimum schedule length of an application [11] when they are executed at maximum frequency using a well-known scheduling algorithm. There are several types of scheduling: static or dynamic and preemptive or non-preemptive. The application can also be hard or soft [12]. This paper considers non-preemptive static scheduling for a hard real-time application.

In our study, we have considered the parallel application shown in Fig.3.2, where each node represents a task, and the number on the edges represents the communication time between the tasks running in different processors. For example, the weight value 14 between tasks n1 and n3 denotes the communication time c1,3 between tasks n1 and n2 when not assigned to the same processor. The table shows the worst-case execution time of each task in 3 different processors. The weight value of 16 of n1 and u2 in the table shows the WCET denoted by w1,2 = 16. So, in short, this example considers 10 tasks {n1, n2,…n10} executed in 3 processors {u1, u2, u3}.

The frequencies must be sorted in ascending order for binary search of frequency levels. This helps perform binary search on the L levels of V/F pairs. Since the energy will increase with increasing frequency, we can find the minimum energy taken for the task by finding the frequency and calculating the energy by sorting in ascending order. For example, consider two frequencies, fi and fj. Assuming frequency fi is greater than a frequency fj (fi > fj). From the dynamic power formula (2), we can say that Ei > Ej where Ei and Ej are the dynamic energy consumption of the respective frequencies fi and fj for a given processor u. Therefore, any frequency greater than fj need not be considered. This reduces the range of search space due to the monotonically increasing nature of energy with frequency.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | u1 | u2 | u3 |
| n1 | 15 | 17 | 10 |
| n2 | 14 | 20 | 19 |
| n3 | 12 | 14 | 20 |
| n4 | 14 | 9 | 18 |
| n5 | 13 | 14 | 11 |
| n6 | 14 | 17 | 10 |
| n7 | 8 | 16 | 12 |
| n8 | 6 | 12 | 15 |
| n9 | 19 | 13 | 21 |
| n10 | 22 | 8 | 17 |

A diagram of a network

AI-generated content may be incorrect.

*Fig. 3.2. Motivating example of a DAG-based parallel application*

*TABLE 3.1. WCETs of Tasks on Different Processors with the Maximum Frequencies of the Motivating Parallel Application*

The deadlines for the tasks are determined by the length of the critical path, and we calculate that the number of execution cycles of each task lie in the range determined by the WCET of the application.

We have also introduced the term 𝛾i,k, which is the efficiency factor of a heterogeneous processor executing a task ni with mid voltage and frequency (vl, fl). Using this efficiency factor, the execution time of the task ni can be defined as,

The communication cost among the processors has been neglected in [11], but we have considered the finite communication time for accuracy in task scheduling. Also, note that the frequencies mentioned in this paper represent Voltage/ Frequency pairs.

The communication cost among the processors has been neglected in [11], but we have considered the finite communication time for accuracy in task scheduling. Also, note that the frequencies mentioned in this paper represent Voltage/ Frequency pairs.

**3.2 POWER AND ENERGY MODELS**

Since we are using a DVFS-enabled system, we define the power consumption of a processor working on voltage, frequency - V/F level (vl, fl) as

where Psta,l = Lg(VkIsub + |Vbs|Ij) is the static power when the processor is ready to execute tasks, and Pdyn,k = CeffVl2flis the dynamic power during the task execution.

It is evident from the equation that the static power is mainly from the subthreshold leakage current Isub, the reverse bias junction current Ij, and the body bias voltage Vbs*.* Similarly, the dynamic power is contributed by the assigned frequency and voltage and on the effective switching capacitance Ceff.

We are mainly focused on reducing dynamic energy, but we also take static energy into consideration while calculating the total energy. The static energy of application G is given by

Let us denote the dynamic energy consumption of task ni executing on processor uj with a frequency fl as

Where ti,l represents the task execution time given in eq. (1)

So, the total dynamic energy consumption of the application is calculated as the sum of the energy consumption while executing each task at their assigned processors.

The total energy consumption of the application is the sum of the static and dynamic energy consumption,

The total energy consumption in terms of static and dynamic power can be represented as,

**3.3 CONSTRAINTS**

In this section, the constraints of the task scheduling problem are outlined. The task assignment and scheduling are dependent on the timing constraints of the tasks, the energy constraint of the application, the task dependencies, and the frequencies for DVFS.

The energy consumption of the tasks in the DAG G must be lesser than the given energy budget, represented by the equation,

where Psta and Pdyn are given in equation (2).

Since tasks in a DAG are dependent on the execution of the previous task, there are task dependency constraints,

where tei represents the end time of the task ni and tsi+1 represents the start time of the task ni+1.

The timing constraints of each task must be taken into account, as each task may have varying deadlines. The deadline for a task ni given by Di, and the execution time of the task gives the following constraint,

Since all tasks must be allocated to a processor with a frequency, we also get the constraint,

This implies that there must be a processor assigned to each task.

For task non-preemption. We need to make sure that if two tasks are assigned to the same processor, their execution sequence is determined. This can be ensured by considering the task dependency constraint for any two different tasks, i.e.,

**3.4 PROBLEM DESCRIPTION**

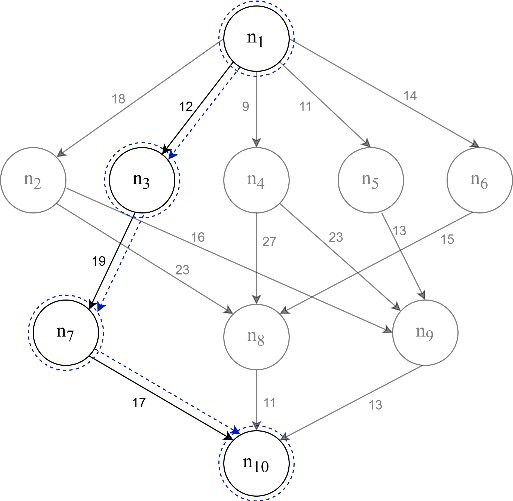
The problem description of this study is to determine the frequency and processor for each task such that it is less complex and the application consumes less energy, i.e., Etotal = Es + Ed should be minimized subject to the frequency, time, task non-preemptive, task dependency and energy constraints mentioned above (8) (9) (10) (11) (12).

**CHAPTER 4**

**PROPOSED WORK**

The proposed algorithm is split into two parts. First, the Frequency Assignment is done, followed by Task Assignment and Scheduling. The frequency assignment algorithm uses binary search to go through the energy consumption by each task when they are running at different frequencies. Then, once a minimum energy for a task is found, the corresponding frequency is chosen as the frequency for that task. This process is repeated until all tasks are assigned frequencies. After every task, the total energy (Etotal) is checked for Energy constraint (Ebudget). This process is followed by assignment of tasks to the processors. First, the tasks are sorted based on their rank and then assigned to the processors for which the earliest finish time of that particular task is the minimum. The final energy consumption is calculated based on the processor to which the task is assigned.

**4.1 FREQUENCY ASSIGNMENT**

The task scheduling problem is constrained by time and Energy, which are affected by the V/F level used for the task. Therefore, frequency allocation is the first step in the process.

Algorithm 1 first allocates a low and high-frequency level. The middle frequency out of the L levels is calculated from high and low and is the temporarily assigned frequency of the task. In line 7, the Energy consumed, and time taken by the middle frequency are calculated using equations (7) and (1), respectively. Lines 8-14 check for RT and energy constraints and update the high and low levels accordingly. For the timing constraints, the critical path time is considered along with the current task’s execution time. The critical path is the path that takes the longest time to completion. In Fig. 2, the critical path is highlighted in blue and is found by finding the path with the highest communication cost. Since the communication cost contributes significantly to the increased start time of a task, it needs to be considered in the critical path time. In the sample DAG shown in Fig. 2, the critical path is 1 🡪3 🡪 7 🡪 10.

*Fig. 4.1. Representation of the critical path*

**Frequency Assignment Algorithm**

|  |  |
| --- | --- |
| **Inputs:** f and v, G=(N,L,M,Di), U, CPT  **Outputs:** Frequencies assigned to tasks | |
| 1: | i represent tasks from 1 to N. l represent frequencies from 1 to L. |
| 2: | low, mid, and high are all frequencies |
| 3: | ci,l = 0, lowß 1, high ß L, Emin = ∞, Etotal = 0, tcp = 0; |
| 4: | for i in 1 to N |
| 5: | while (low <= high) |
| 6: | mid = (low+high)/2 |
| 7: | calculate Ei, mid, ti, mid using eqn. (7) and (1) respectively |
| 8: | if Etotal + Ei, mid <= Ebudget and ti, mid + tcp < Di |
| 9: | if Ei, mid <= Emin |
| 10: | l\* = lmid |
| 11: | Emin = Ei, mid |
| 12: | high = mid |
| 13: | else |
| 14: | low = mid +1 |
| 15: | if Emin + Etotal ≤ Ebudget then |
| 16: | Etotal = Emin, Emin =∞, ci,l∗ = 1 |
| 17: | tcp = tcp + ti,l∗ (i ∈ CPT) |
| 18: | else |
| 19: | Stop |
| 20: | end |
| 21: | end |

If these constraints are satisfied, we check for the minimum energy out of all the levels and update the high or low level accordingly. In the first iteration, the condition is satisfied by default as Emin is set to INT\_MAX, and the high value is changed to mid. This means that there is no need to consider levels that give high energy consumption. If the constraint checks fail, then the low value is set to mid, as any level lesser than mid will not be able to satisfy the constraints. This means that the middle frequency changes every iteration and will converge at the minimum energy consumption level, like binary search. The loop continues until a low value is greater than the high, which indicates that the optimal minimum level is found.

Finally, total energy consumption is considered with this new frequency level for the task and tested if it is within the energy budget. If this is satisfied, the frequency level is finalized for the current task, and the next task is tested. If the energy exceeds the given energy, then the frequency assignment fails.

**4.2 TASK ASSIGNMENT**

Once the frequency has been assigned to each task, the process of allocating the task to the processor begins.

1) Rank tasks: The first step is to calculate the rank of each task, starting from the exit task and then prioritizing the tasks based on the rank values. The rank of task ni can be calculated using the equation,

Where Average ExecTime(ni) is the average worst-case execution time of task ni and is given by the sum of the task execution times in all the processors divided by the number of processors; for example, in our motivational example from Table 1, we can calculate the average execution time of task 1 as (15+17+10)/3 = 14.

Comm Time (ni,j) is the communication time between tasks ni and nj, where nj is the successor of task ni.

Rank(nj) is the rank of successor of task ni.

After calculating the rank of each task, they are ordered according to the descending order of rank.

2) Compute EFT: After ordering the tasks based on their rank, take the task with the highest rank and calculate the earliest start time for that task ni at a particular processor uy using the equation,

Where EST(ni​,uy​) is the Earliest Start Time of task ni on processing unit uy, nentry is the entry task, avail[y] is the earliest time when processor uy is ready for task execution, AFT[nk] represents the actual finish time (AFT) of predecessor task of task ni and ​ represents the communication time between tasks nk and ni when they are running on two different processors, ux and uy. If both tasks are assigned to the same processor, communication time will be 0. The fundamental difference between EFT and AFT is that EFT is the value before task assignment, whereas AFT is the value after task assignment.

Once EST is computed, the earliest finish time needs to be calculated for that task ni, running in that particular processor uy at a frequency fl determined using the FA algorithm. The formula for calculating EFT is given by,

**Task Assignment Algorithm**

|  |  |
| --- | --- |
|  | **Inputs:** fj and vj for each task, G=(N,C,W), U  **Outputs:** Processor assignment, SL(G), E\_total (G) |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17: | start  for each task nj ∈ N:  Calculate Rank(nj) using equation (13)  sort the tasks in descending order of Rank and store in a priority list P  while (P ≠ ∅) do  Select the highest ranked task nj from P  fj ← frequency assigned to task nj during the FA process  For each uy ∈ U:  Compute EST (nj, uy, fj) // Eq. (14)  Compute EFT (nj, uy, fj) // Eq. (15)  Assign task nj to the processor uy ∈ U such that EFT (nj, uy, fj) is minimized  Calculate E(nj) // Eq. (7)  Remove task nj from P  end  Calculate E\_total (G)  Calculate SL(G) = AFT (nexit)  end |

3) Processor assignment: All processors go through the same procedure, and their EFTs are determined. After comparing each EFT, the task is assigned to the processor with the lowest EFT.

Once the processor assignment is done, calculate the energy associated with that task execution using equation (7).

4) Steps two and three are repeated after choosing the subsequent task from the priority list. This process continues for every other task until every task is mapped to the processor.

5) The final step is to calculate total energy consumption which is the sum of energy consumptions for individual tasks. The Schedule length of the application is given as the Actual Finish Time of the exit task.

**CHAPTER 5**

**SIMULATION RESULTS**

**5.1 VIVADO SIMULATION OUTPUT**

The performance of the proposed FA and TA algorithms has been evaluated in the AMD Vivado simulator using Verilog. A set of voltage and frequency pairs has been provided for the task’s operation, from which the optimal V/F pair is selected using the FA algorithm. The other inputs include Ebudget, Execution cycles for each task, Deadlines, Critical path tasks, Communication cost, execution time of each task, etc. After providing all the necessary inputs, the algorithm has been simulated, and the task scheduling has been done. The output of the simulation has been shown in Fig. 5.1,

A green line on a black background

AI-generated content may be incorrect.

*Fig. 5.1. Waveform showing the assigned frequency and total energy (Vivado simulator)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Initially, the clock signal was provided at 100MHz, and the M, f, and v represent the Execution cycles for each task and available frequency and voltage values, respectively. Since creating a 2d array in Verilog is complex, we have converted it into vector form and used it accordingly. The D represents the deadline for each task, which must be strictly followed. The communication and execution time values are provided from the task graph shown in our motivation example. Once the reset signal is low, the frequency assignment starts and is shown as assigned\_frequencies. The final result is stored in assigned\_frequencies. The corresponding energy consumption is calculated using equation (4) during each frequency assignment. Once the frequency has been assigned to all the tasks, the FA\_Done signal becomes 1, indicating that the FA algorithm is complete. Then, the task assignment algorithm starts its execution, and the tasks are allocated to different processors based on the EFTs of the processor for a particular task. The TA\_done signal becomes 1 when the TA process is over. The processor allocation and total energy consumptions are shown in Fig. 5.1.

*Fig. 5.2. processor and frequency assignment output*

**5.2 COMPLEXITY**

The proposed algorithm has two parts: Frequency Assignment and Task Assignment. The frequency assignment uses a binary search over the available frequency levels for each processing element, giving a time complexity of O (|N| \* log |L|), where ∣N∣ is the number of processing elements and ∣L∣ is the number of frequency levels. The task assignment uses a rank-based method that evaluates each task against all processing elements, resulting in O(|N|\*|U|) complexity, where ∣U∣ is the number of tasks. So, the total complexity of the proposed method is O ((|N| \* |L|) + (|N|\*|U|)). In comparison, the greedy algorithm uses a linear scan for frequency selection and the same task assignment strategy, giving a slightly higher complexity of O ((|N| \* |L|) + (|N|\*|U|)). ISAECC and GDES are more computationally expensive, with ISAECC taking O (|N|2 \* |U| \* |L|) and GDES taking O (|N|2 \* |U|). This shows that the proposed approach achieves better scalability while maintaining efficiency.

*TABLE 5.1. Complexities of different algorithms*

|  |  |
| --- | --- |
| Algorithm | Complexity |
| Proposed Algorithm | O ((|N| \* log |L|) + (|N|\*|U|)) |
| Greedy Approach | O ((|N| \* |L|) + (|N|\*|U|)) |
| ISAECC | O (|N|2 \* |U| \* |L|) |
| GDES | O (|N|2 \* |U|) |

**5.3 ENERGY**

**5.3.1 Motivational Application**

The frequency and task assignment algorithm has been implemented for the motivating example application, demonstrating its practical applicability and effectiveness. For this case, the frequency assigned to each processing element, the processor to which each task is mapped, the resulting schedule length, and the overall energy consumption have been computed. These results are summarized in Table 5.2, providing a clear view of how the proposed algorithm optimizes both performance and energy efficiency. Additionally, the Gantt chart shown in figure 5.3 illustrates the actual task scheduling timeline based on the proposed approach. It shows how different tasks are assigned to processors and also shows the actual start time and finish time of the tasks in their respective processors. The Gantt chart also emphasizes that the application execution was completed well within the deadline and does not violate it while trying to reduce energy consumption.

*TABLE 5.2. Task assignment using the proposed algorithm for the motivating example*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ni | uy | fy,l (normalized) | AFT (ni) | Energy (ni) |
| n1 | u3 | 0.6 | 16 | 2.82 |
| n2 | u3 | 0.6 | 46 | 5.36 |
| n3 | u1 | 0.6 | 47 | 3.38 |
| n4 | u2 | 0.6 | 39 | 2.54 |
| n5 | u2 | 0.6 | 62 | 3.95 |
| n6 | u3 | 0.6 | 62 | 2.82 |
| n7 | u1 | 0.44 | 65 | 2.04 |
| n8 | u1 | 0.44 | 91 | 1.53 |
| n9 | u2 | 0.6 | 83 | 3.66 |
| n10 | u2 | 0.73 | 111 | 2.56 |
| E\_total (G) = 30.66, SL(G) = 111 | | | | |

A diagram of a graph

AI-generated content may be incorrect.

*Fig.5.3. Scheduling Gantt chart for the motivational application using the proposed algorithm*

**i. Energy Consumption for a fixed deadline**

The energy consumption of the proposed algorithm has been evaluated for the motivating example DAG graph and compared against existing algorithms—Greedy, ISAECC, and GDES—under a fixed deadline constraint. This comparison highlights the energy efficiency achieved through the proposed frequency and task assignment strategy. As shown in Fig. 5.4, the proposed algorithm consumes the least energy (30.66 units), outperforming GDES (31.58), ISAECC (33.98), and especially the Greedy approach (40.88). These findings show how effectively the proposed method works to meet timing constraints while consuming less energy, which makes it a good fit for embedded systems with limited energy.

*Fig 5.4. Comparison of energy consumption for fixed deadline*

**ii. Energy Consumption for variable deadlines**

We have further compared the energy consumption of the proposed algorithm against Greedy, ISAECC, and GDES for a range of deadline values, and the results are illustrated in Fig. 5.5. The Energy vs. Deadline graph shows that when the deadline is tight, all algorithms perform similarly in terms of energy consumption. However, as the deadline becomes more relaxed, the proposed algorithm significantly outperforms the others by operating at a lower frequency and distributing tasks more efficiently. This leads to significant energy savings, demonstrating the algorithm’s adaptability and effectiveness in dynamic scheduling scenarios where flexibility in deadline constraints is available. Table 5.4. shows energy savings over other algorithms. The table clearly shows that the proposed algorithm is superior to greedy approach and significantly higher over ISAECC and GDES algorithm in terms of energy efficiency.

*Fig 5.5. Comparison of energy consumption at different deadlines*

*TABLE 5.3. Energy consumption at different deadlines*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Deadline | Proposed Algorithm | Greedy Approach | GDES Algorithm | ISAECC Algorithm |
| 101 | 32.42 | 41.28 | 31.58 | 35.94 |
| 107 | 29.99 | 40.19 | 31.093 | 35.01 |
| 112 | 29.33 | 39.7 | 31.077 | 33.98 |
| 123 | 28.99 | 38.5 | 30.87 | 33.28 |
| 128 | 27.92 | 37.2 | 30.87 | 32.39 |
| 181 | 26.18 | 34.77 | 30.87 | 28.66 |
| 192 | 26.01 | 34.45 | 30.87 | 28.66 |
| 200 | 26.01 | 34.45 | 30.87 | 28.66 |
| 215 | 26.01 | 34.45 | 30.87 | 28.66 |

*TABLE 5.4. Reduction in energy consumption over other algorithms (%)*

|  |  |  |  |
| --- | --- | --- | --- |
| Deadline | Proposed Algorithm | Greedy Approach | GDES Algorithm |
| 101 | 27.3 | - | 10.8 |
| 107 | 34.0 | 3.6 | 16.7 |
| 112 | 35.3 | 5.9 | 15.8 |
| 123 | 32.8 | 6.4 | 14.7 |
| 128 | 33.2 | 10.5 | 16.0 |
| 181 | 32.8 | 17.9 | 9.4 |
| 192 | 32.4 | 18.6 | 10.2 |

**C. Comparison of processor assignment, AFT and energies of each task:**

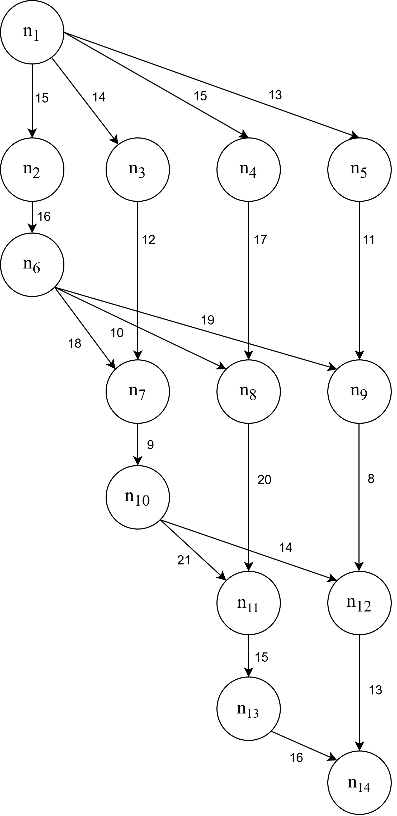
In this study, the processor assignment, schedule length, and individual task energy consumption have been analyzed and compared with those of the ISAECC and GDES algorithms. This detailed comparison shows how the proposed algorithm manages workload distribution and energy efficiency at a finer granularity. The results of this comparison are presented in Tables 5.5 and 5.6, which highlight the improvements achieved in both task-level energy savings and overall scheduling efficiency. It shows that for a similar schedule length, the energy consumption of our algorithm is less as compared to other energy efficient scheduling algorithms. These tables clearly demonstrate that the proposed algorithm not only balances task allocation effectively across processors but also achieves lower energy consumption per task compared to the baseline algorithms.

*TABLE 5.5 GDES vs. Proposed algorithm using motivational example from Fig. 3.2*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ni | uy | |  | fy,l(normalized) | |  | AFT (ni) | |  | Energy (ni) | |
| GDES | prop |  | GDES | prop |  | GDES | prop |  | GDES | prop |
| n1 | u3 | u3 |  | 1 | 0.6 |  | 10 | 16 |  | 2.96 | 2.82 |
| n2 | u3 | u3 |  | 1 | 0.6 |  | 29 | 46 |  | 5.64 | 5.36 |
| n3 | u1 | u1 |  | 1 | 0.6 |  | 34 | 47 |  | 3.56 | 3.38 |
| n4 | u2 | u2 |  | 0.86 | 0.6 |  | 30 | 39 |  | 2.66 | 2.54 |
| n5 | u3 | u2 |  | 1 | 0.6 |  | 40 | 62 |  | 3.26 | 3.95 |
| n6 | u2 | u3 |  | 0.86 | 0.6 |  | 48 | 62 |  | 4.85 | 2.82 |
| n7 | u1 | u1 |  | 0.26 | 0.44 |  | 68 | 65 |  | 1.84 | 2.04 |
| n8 | u1 | u1 |  | 0.44 | 0.44 |  | 74 | 91 |  | 1.31 | 1.53 |
| n9 | u2 | u2 |  | 0.6 | 0.6 |  | 85 | 83 |  | 3.51 | 3.66 |
| n10 | u2 | u2 |  | 0.4 | 0.73 |  | 101 | 111 |  | 2.24 | 2.56 |

*TABLE 5.6 ISAECC vs. Proposed algorithm using motivational example from Fig. 3.2*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ni | uy | |  | fy,l (normalized) | |  | AFT (ni) | |  | Energy (ni) | |
| ISAECC | prop |  | ISAECC | prop |  | ISAECC | prop |  | ISAECC | prop |
| n1 | u3 | u3 |  | 1 | 0.6 |  | 10 | 16 |  | 3.78 | 2.82 |
| n2 | u1 | u3 |  | 0.6 | 0.6 |  | 65 | 46 |  | 4.10 | 5.36 |
| n3 | u1 | u1 |  | 0.6 | 0.6 |  | 42 | 47 |  | 3.51 | 3.38 |
| n4 | u2 | u2 |  | 1 | 0.6 |  | 28 | 39 |  | 3.41 | 2.54 |
| n5 | u3 | u2 |  | 0.6 | 0.6 |  | 28 | 62 |  | 3.22 | 3.95 |
| n6 | u3 | u3 |  | 1 | 0.6 |  | 38 | 62 |  | 3.78 | 2.82 |
| n7 | u1 | u1 |  | 1 | 0.44 |  | 73 | 65 |  | 3.03 | 2.04 |
| n8 | u1 | u1 |  | 1 | 0.44 |  | 79 | 91 |  | 2.27 | 1.53 |
| n9 | u2 | u2 |  | 0.6 | 0.6 |  | 102 | 83 |  | 3.81 | 3.66 |
| n10 | u2 | u2 |  | 1 | 0.73 |  | 111 | 111 |  | 3.03 | 2.56 |

**5.3.1 Gaussian Elimination Application**

Gaussian elimination is primarily used to solve systems of linear equations. It transforms a system of equations into an equivalent but simpler system, making it easier to find the solutions. This method also finds applications in finding matrix inverses, calculating determinants, and determining linear independence of vectors.

The proposed algorithm is compared with the other algorithms using the Gaussian Elimination application. Fig. 5.6 shows the DAG of the GE application using the matrix size m = 5. The number of tasks is calculated based on the matrix size and is given by the formula |N| = (m2+m-2)/2. This application allows us to test the algorithms for task parallelism. Fig. and the table y show the scheduling gantt chart and the task assignment for the GE application using the proposed algorithm, respectively.

*Fig. 5.6. DAG of the Gaussian Elimination application for m = 5*

*TABLE 5.7. Task assignment using the proposed algorithm for the motivating example*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ni | uy | fy,l (normalized) | AFT (ni) | Energy (ni) |
| n1 | u1 | 0.6 | 24 | 4.22 |
| n2 | u2 | 0.33 | 66 | 2.11 |
| n3 | u1 | 0.4 | 51 | 2.69 |
| n4 | u3 | 0.33 | 84 | 3.52 |
| n5 | u1 | 0.4 | 99 | 4.65 |
| n6 | u2 | 0.26 | 107 | 2.48 |
| n7 | u2 | 0.6 | 148 | 7.33 |
| n8 | u1 | 0.33 | 163 | 2.81 |
| n9 | u3 | 0.4 | 182 | 6.61 |
| n10 | u2 | 0.4 | 163 | 2.47 |
| n11 | u1 | 0.4 | 237 | 5.14 |
| n12 | u3 | 0.33 | 215 | 2.57 |
| n13 | u3 | 0.4 | 287 | 3.43 |
| n14 | u3 | 0.33 | 316 | 3.34 |
| E\_total (G) = 53.37 , SL(G) = 316 | | | | |

A screenshot of a computer

AI-generated content may be incorrect.

*Fig. 5.7. Scheduling Gantt chart for the Gaussian elimination application using the proposed algorithm*

**i. Energy Consumption for a fixed deadline**

The energy consumption of the proposed algorithm has been evaluated for the Gaussian Elimination DAG graph and compared against existing algorithms—Greedy, ISAECC, and GDES—under a fixed deadline constraint. This comparison highlights the energy efficiency achieved through the proposed frequency and task assignment strategy. As shown in Fig. 5.8, the proposed algorithm consumes the least energy (51.42 units), outperforming GDES (52.96), ISAECC (55.32), and the Greedy approach (53.92). These findings show how effectively the proposed method works to meet timing constraints while consuming less energy, which makes it a good fit for embedded systems with limited energy.

*Fig 5.8. Comparison of energy consumption*

**ii. Energy Consumption for variable deadlines**

The same experiment as done for the motivating applications is conducted for the GE application. The deadline for the application is varied and energy consumption is noted. The results are shown on Fig.5.9. We can infer that for a looser deadline scenario the proposed algorithm consumes less energy compared to the other algorithms. The percentage reduction in energy consumption of the proposed algorithm compared to the other algorithms is shown in Table 5.8. The maximum reduction is seen at higher deadlines at about 22% when compared to ISAECC.

|  |  |  |  |
| --- | --- | --- | --- |
| Deadline | Greedy | GDES | ISAECC |
| 229 | - | - | - |
| 265 | - | - | 0.1 |
| 301 | - | 0.1 | 0.8 |
| 337 | 3.0 | 4.4 | 11.0 |
| 373 | 8.5 | 5.8 | 18.5 |
| 409 | 8.2 | 7.7 | **20.4** |

*TABLE 5.8. Percentage reduction in energy consumption of proposed vs other algorithms for varying deadlines.*

*Fig 5.9 Comparison of energy consumption at different deadlines*

**iii. Comparison of processor assignment, AFT and energies of each task:**

This experiment compares the task assignment, frequency assignment, AFT and Energy consumption for each task of the different algorithms for the GE application. This detailed comparison shows how the proposed algorithm manages workload distribution and energy efficiency at a finer granularity. The results of this comparison are presented in Tables 10 and 11, which highlight the improvements achieved in both task-level energy savings and overall scheduling efficiency. It shows that for a similar schedule length, the energy consumption of our algorithm is less as compared to other energy efficient scheduling algorithms.

*TABLE 5.9. ISAECC vs. Proposed algorithm using Gaussian Elimination application*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ni | uy | |  | fy,l (normalized) | |  | AFT (ni) | |  | Energy (ni) | |
| ISAECC | prop |  | ISAECC | prop |  | ISAECC | prop |  | ISAECC | prop |
| n1 | u1 | u1 |  | 0.6 | 0.6 |  | 25 | 24 |  | 4.4 | 4.22 |
| n2 | u2 | u2 |  | 0.86 | 0.33 |  | 50 | 66 |  | 3.19 | 2.11 |
| n3 | u1 | u1 |  | 0.6 | 0.4 |  | 43 | 51 |  | 3.23 | 2.69 |
| n4 | u2 | u3 |  | 0.44 | 0.33 |  | 96 | 84 |  | 3.12 | 3.52 |
| n5 | u3 | u1 |  | 0.33 | 0.4 |  | 192 | 99 |  | 4.22 | 4.65 |
| n6 | u2 | u2 |  | 0.6 | 0.26 |  | 69 | 107 |  | 3.23 | 2.48 |
| n7 | u3 | u2 |  | 0.33 | 0.6 |  | 138 | 148 |  | 3.99 | 7.33 |
| n8 | u2 | u1 |  | 0.6 | 0.33 |  | 171 | 163 |  | 3.22 | 2.81 |
| n9 | u1 | u3 |  | 0.6 | 0.4 |  | 228 | 182 |  | 4.4 | 6.61 |
| n10 | u2 | u2 |  | 1 | 0.4 |  | 153 | 163 |  | 2.27 | 2.47 |
| n11 | u3 | u1 |  | 0.6 | 0.4 |  | 227 | 237 |  | 6.15 | 5.14 |
| n12 | u1 | u3 |  | 1 | 0.33 |  | 242 | 215 |  | 5.30 | 2.57 |
| n13 | u3 | u3 |  | 0.44 | 0.4 |  | 258 | 287 |  | 3.64 | 3.43 |
| n14 | u3 | u3 |  | 0.86 | 0.33 |  | 269 | 316 |  | 3.54 | 3.34 |

*TABLE 5.10. GDES vs. Proposed algorithm using Gaussian Elimination application*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ni | uy | |  | fy,l (normalized) | | |  | AFT (ni) | |  | Energy (ni) | |
| GDES | prop |  | GDES | prop |  | | GDES | prop |  | GDES | prop |
| n1 | u1 | u1 |  | 1 | 0.6 |  | | 15 | 24 |  | 4.453 | 4.22 |
| n2 | u1 | u2 |  | 0.6 | 0.33 |  | | 42 | 66 |  | 3.794 | 2.11 |
| n3 | u1 | u1 |  | 0.6 | 0.4 |  | | 53 | 51 |  | 3.092 | 2.69 |
| n4 | u2 | u3 |  | 0.33 | 0.33 |  | | 73 | 84 |  | 2.779 | 3.52 |
| n5 | u3 | u1 |  | 0.6 | 0.4 |  | | 61 | 99 |  | 4.637 | 4.65 |
| n6 | u1 | u2 |  | 1 | 0.26 |  | | 52 | 107 |  | 3.266 | 2.48 |
| n7 | u1 | u2 |  | 1 | 0.6 |  | | 74 | 148 |  | 6.234 | 7.33 |
| n8 | u1 | u1 |  | 0.4 | 0.33 |  | | 104 | 163 |  | 2.40 | 2.81 |
| n9 | u1 | u3 |  | 1 | 0.4 |  | | 89 | 182 |  | 4.453 | 6.61 |
| n10 | u1 | u2 |  | 1 | 0.4 |  | | 83 | 163 |  | 2.672 | 2.47 |
| n11 | u1 | u1 |  | 1 | 0.4 |  | | 125 | 237 |  | 6.234 | 5.14 |
| n12 | u3 | u3 |  | 0.33 | 0.33 |  | | 134 | 215 |  | 2.391 | 2.57 |
| n13 | u1 | u3 |  | 1 | 0.4 |  | | 147 | 287 |  | 6.531 | 3.43 |
| n14 | u3 | u3 |  | 0.33 | 0.33 |  | | 337 | 316 |  | 2.391 | 3.34 |

**5.4 HARDWARE IMPLEMENTATION**

**CHAPTER 6**

**SUMMARY AND CONCLUSION**

**6.1 Summary**

In this thesis, we have proposed an energy-efficient task assignment algorithm for heterogeneous embedded systems that allocates a frequency and processor to each task such that the application consumes less energy while still meeting its deadline. Several research papers related to energy-efficient task scheduling were reviewed, and the advantages and drawbacks of each algorithm were noted to arrive at a novel solution to this energy-efficiency problem. After a thorough literature review, we developed a low-complex yet energy-efficient frequency assignment and task allocation strategy. The Frequency Assignment (FA) algorithm uses binary search for frequency selection, reducing the computational complexity to O(N⋅logL), where N is the number of tasks and L represents the number of frequency levels. It also considers both energy and deadline constraints. The total energy consumption must not exceed the maximum allowable energy of the application denoted by Ebudget. Similarly, the schedule length of the application must not exceed the given deadline. For task allocation, we proposed a Task Assignment (TA) algorithm based on Rank and Earliest Finish Time (EFT), which ensures that tasks are assigned to the available processor cores optimally while minimizing the overall execution time. The final energy consumption is calculated based on the processor to which each task is allocated and the execution time of the task on that processor. The proposed algorithm has been implemented in AMD Vivado for a motivating application and the Gaussian Elimination task graph, and their corresponding energy consumption has been computed. In addition, we simulated other energy-efficient task assignment algorithms such as ISAECC, Greedy, and GDES, and compared their energy consumption with that of our proposed algorithm. The deadline was also varied, and the corresponding energy consumption was recorded. The results showed that the proposed algorithm consistently consumes less energy compared to the other three algorithms under a fixed deadline and schedule length. When the deadline was varied, the energy consumption across all algorithms was similar under strict deadlines, but as the deadline became more relaxed, our algorithm clearly outperformed the others in terms of energy savings. This leads to significant energy reductions, demonstrating the algorithm’s adaptability and effectiveness in dynamic scheduling scenarios where deadline flexibility is available.

**6.2 Conclusions and Future**

This study introduced a novel task-scheduling algorithm tailored for heterogeneous embedded platforms, focusing on optimizing both frequency selection and task allocation to enhance energy efficiency. By employing a binary search-based frequency assignment strategy and a rank and EFT based task assignment method, the proposed approach significantly reduces computational complexity and energy consumption compared to conventional algorithms. Simulation results using AMD Vivado validated the effectiveness of the method, showing improvements not only in energy savings but also in schedule length and execution efficiency. Furthermore, the FPGA-based implementation reinforces the practical viability of the approach for real-world embedded systems. Overall, the proposed strategy strikes a balance between low complexity and high energy efficiency, making it a promising solution for future energy-aware embedded applications.

For future work, we aim to extend this approach to handle more stringent timing constraints, where the deadline is tight, and energy optimization becomes more challenging. Additionally, we plan to handle changing workloads and explore the use of machine learning to make the scheduling more flexible and efficient in real-time systems.

**REFERENCES**

[1] Zhuo Tang, Ling Qi, Zhenzhen Cheng, Kenli Li, Samee U. Khan, and Keqin Li. 2016. An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. J. Grid Comput. 14, 1 (March 2016), 55–74.

[2] J. Niu, C. Liu, Y. Gao and M. Qiu, "Energy Efficient Task Assignment with Guaranteed Probability Satisfying Timing Constraints for Embedded Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 8, pp. 2043-2052, Aug. 2014.

[3] G. Xie, X. Xiao, H. Peng, R. Li and K. Li, "A Survey of Low-Energy Parallel Scheduling Algorithms," in IEEE Transactions on Sustainable Computing, vol. 7, no. 1, pp. 27-46, 1 Jan.-March 2022.

[4] Hu, Biao, et al. “Online Energy-Efficient Scheduling of DAG Tasks on Heterogeneous Embedded Platforms.” Journal of Systems Architecture, vol. 140, July 2023, p. 102894.

[5] Abhishek Mishra, Anil Kumar Tripathi, Energy efficient voltage scheduling for multi-core processors with software controlled dynamic voltage scaling, Applied Mathematical Modelling, Volume 38, Issue 14, 2014, Pages 3456-3466.

[6] Antolak, Ernest, and Andrzej Pulka. “Energy-Efficient Task Scheduling in Design of Multithread Time Predictable Real-Time Systems.” IEEE Access, vol. 9, 2021, pp. 121111–27.

[7] Huang, Kai & Jing, Ming & Jiang, Xiaowen & Chen, Siheng & Li, Xiaobo & Tao, Wei & Xiong, Dongliang & Liu, Zhili. (2020). Task-Level Aware Scheduling of Energy-Constrained Applications on Heterogeneous Multi-Core System. Electronics. 9. 2077.

[8] Ying Li, Jianwei Niu, Mohammed Atiquzzaman, Xiang Long, Energy-aware scheduling on heterogeneous multi-core systems with guaranteed probability, Journal of Parallel and Distributed Computing, Volume 103, 2017, Pages 64-76.

[9] Z. Quan, Z. -J. Wang, T. Ye and S. Guo, "Task Scheduling for Energy Consumption Constrained Parallel Applications on Heterogeneous Computing Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 5, pp. 1165-1182, 1 May 2020.

[10] G. Xie, G. Zeng, X. Xiao, R. Li and K. Li, "Energy-Efficient Scheduling Algorithms for Real-Time Parallel Applications on Heterogeneous Distributed Embedded Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 12, pp. 3426-3442, 1 Dec. 2017.

[11] Li, Xinmei, et al. “Approximation-Aware Task Deployment on Heterogeneous Multicore Platforms With DVFS.” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 7, July 2023, pp. 2108–21.

[12] G. Bernat, A. Colin, and S. M. Petters, “WCET analysis of probabilistic hard real-time systems,” in Proc. 23rd IEEE Real-Time Syst. Symp., 2002, pp. 279–288.

[13] Y. Xie, G. Zeng, Y. Chen, R. Kurachi, H. Takada, and R. Li, “Worst case response time analysis for messages in controller area network with gateway,” IEICE Trans. Inf. Syst., vol. 96, no. 7, pp. 1467–1477, Jul. 2013.

[14] K. M. Tarplee, R. Friese, A. A. Maciejewski, H. J. Siegel, and E. K. Chong, “Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques,” IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 6, pp. 1633–1646, Jun. 2016.

[15] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” ACM Comput. Surveys, vol. 43, no. 4, Oct. 2011, Art. no. 35.

[16] Hongzhi Xu, Binlian Zhang, Chen Pan, Keqin Li, Energy-efficient scheduling for parallel applications with reliability and time constraints on heterogeneous distributed systems, Journal of Systems Architecture, Volume 152, 2024.

[17] Hongzhi Xu, Renfa Li, Lining Zeng, Keqin Li, Chen Pan, Energy-efficient scheduling with reliability guarantee in embedded real-time systems, Sustainable Computing: Informatics and Systems, Volume 18, 2018, Pages 137-148.