Journal of Engineering Sciences and Information Technology
Volume (6), Issue (4): 30 Jun 2022
P: 133 - 150

ISSN: 2522- 3321

مجلة العلوم الهندسية وتكنولوجيا المعلومات
المجلد (6)، العدد (4): 30 يونيو 2022م
ص: 133 - 150

Development of Energy- Efficient Load Balancing Scheduling Algorithm in Real- time Systems

Youssef Mohammad Ntefeh

Mohammad Hijazieh

Faculty of Mechanical & Electrical Engineering || Tishreen University || Syria

Abstract: Real- time systems are considered as one of the most important topics that have attracted the attention of researchers in various scientific and technical fields due to their widespread in many fields of communication, informatics, operating systems and automatic control systems. The basic idea in real- time systems is the execution of a set of tasks that are assigned to CPUs, within a time constraint associated with each of them, and this time constraint is called a deadline.

Task scheduling in real- time systems is the main factor in determining the success or failure of the system. Therefore, many studies have imposed suggestions and hypotheses for improving scheduling, according to the underlined platform, whether it is single- processor, multi- processor, or multi- core processor.

Researchers have recently focused on studying and providing solutions to scheduling problems, taking into account energy consumption since energy consumption today plays an important role in determining the efficiency and reliability of the system. Energy consumption can be saved either statically by turning the processor into sleep mode or dynamically by changing the frequency of the processor cores and thus reducing power consumption. However, a slight increase in frequency leads to executing more tasks in less time and thus can lead to an increase in the efficiency of the real- time system by reducing tasks which may miss its deadline at the time of execution.

In this research, a new method has been proposed to control the processor frequency in an optimal manner that makes a balanced trade- off between power consumption and execution of tasks in real- time systems.

Keywords: Multi- core Processor, Power Consumption, Aperiodic Tasks, Periodic Tasks, Processor frequency.

تطوير خوارزمية جدولة مهام موازنة للحمل موفرة للطاقة في أنظمة الزمن الحقيقي

يوسف محمد نتيفه

محمد حجازية

كلية الهندسة الميكانيكية والكهربائية || جامعة تشرين || سوريا

المستخلص: تعتبر أنظمة الزمن الحقيقي أحد أهم المواضيع التي تسترعي اهتمام الباحثين في مختلف المجالات العلمية والتقنية وذلك لانتشار أنظمة الزمن الحقيقي في مجالات الاتصالات والمعلوماتية وأنظمة التشغيل ونظم التحكم. المبدأ الأساسي لأنظمة الزمن الحقيقي هو تنفيذ مجموعة من المهام المسندة إلى معالجات ضمن قيود زمنية خاصة بكل منها وتدعى الحد النهائي.

جدولة المهام في أنظمة الزمن الحقيقي هي العامل الأساسي في تحديد نجاح أو فشل النظام، لذلك وضع عدد من الباحثين مقترحات ونظريات لتحسين الجدولة وفقا لمنصة العمل إن كانت أحادية المعالج أو متعددة النوى أو متعددة المعالجات.

تركز الأبحاث الحالية على دراسة توفير حلول لمشاكل المعالجة آخذة بالحسبان استهلاك الطاقة حيث أنها تلعب دورا حاسما اليوم في تحديد مدى فعالية النظام وموثوقيته. يمكن تقليل استهلاك الطاقة الساكنة بوضع المعالج في حالة سكون ويمكن تقليل استهلاك

الطاقة الديناميكية بتعديل تردد نوى المعالجة وبذلك تقليل جهد العمل وتخفيض استهلاك الطاقة. في بعض الحالات تؤدي زيادة تردد العمل بقيمة بسيطة إلى إنجاز عدد أكبر من المهام في زمن أقل مما سيؤدي إلى زيادة فعالية النظام بتقليل عدد المهام التي قد تتجاوز الحد النهائي أثناء التنفيذ.

في هذه الدراسة يقدم الباحث تقنية جديدة للتحكم بتردد المعالج بطريقة أمثليه توزان ما بين استهلاك الطاقة وتنفيذ المهام في أنظمة الزمن الحقيقي.

الكلمات المفتاحية: معالج متعدد النوى، استهلاك الطاقة، مهام دورية، مهام لا دورية، تردد المعالج.

1- Introduction.

Real- time systems is distinguished from other systems by its commitment to strict time requirements, meaning that the system performs the required tasks according to time restrictions associated with each task, so that these restrictions should not be missed during execution, and therefore the execution output is not considered feasible if the task is executed outside these restrictions. Such systems are expected to be fast responding and guaranteed and a system is said to be in real- time if the correctness of the overall response of a process depend not only on its logical correctness, but also on the time frame in which it is executed. Real- time systems are categorized by their ability to miss deadline. Accordingly, there are three types of real- time systems, Hard Real- Time System (HRTS): The real- time system is hard when the execution of the task after deadline leads to a complete failure of the system even if it is executed correctly but is delayed, such as brake system of a car. Firm Real- Time System (FRTS): It is the system in which missing few deadlines does not cause a complete failure, but rather a degradation in the performance of the system, but increasing the number of tasks missing their deadline will inevitably lead to the failure of the system, such as the system of video calls or tracking by satellite. Soft Real- Time System (SRTS): In this type of real- time system, the performance of the system decreases if the tasks do not finish their work within their own deadline, the system won't fail, but rather leads to a delay in the response, such as the task of bank transfers via the internet.

1-1 Research problem:

The research addresses two problems in the real- time systems. The first is the waste of processing power in case of light load due to the lack of optimal exploit of energy in operating the system, due to processor remains standby without performing any task. The second problem is the decrease in system performance at the expense of a simple increase in processor power consumption.

1-2 Research hypothesis:

1. Light load in real- time systems leads to waste of energy consumption.
2. System overload in real- time systems will cause some tasks to miss their deadline.
3. Reducing processor frequency leads to reducing power consumption.
4. Processor Overclocking within boundaries won't cause hardware damage.

5. Particle swarm optimization algorithm could be used to find an optimal processor frequency to balance real- time system load.

1-3 Research aims

The main objective of this research is to propose an optimal method to achieve a balance between energy consumption and system throughput in real- time systems. The proposed method adjusts the frequency of the processor cores so that it is possible to increase or decrease the frequency according to the type, number and nature of the executed tasks. The proposed method will improve the performance of real- time systems in such a manner that allows a larger number of tasks to be executed, even if it is at the expense of increasing the frequency of the processor cores and thus increasing power consumption. On the other hand, if there are a few tasks to be executed, then the method attends to reduce the frequency of the processor cores and thus reduce power consumption, so in total this allows the real- time system to operate for a longer period of time, especially when the system is running on platforms with mobile power sources.

1-4 Research significance:

- Extending the processor lifespan of by reducing high frequency rates through minimizing system overload.
- Reducing power consumption in portable devices.
- Improving real- time system performance when working on devices with constant power source.
- Adapting the operating system to the instantaneous changes in the load.

1-5 Research tools:

- SIMSO [5] is used to simulate the scheduling of real time tasks under different algorithms and evaluate the performance and overheads of real- time system.
- Matlab is used to build the mathematical model for the scheduling algorithm, task sets and energy consumption equations.

1-6 Background and related works:

A lot of studies regarding balancing load in real- time operating systems considering energy efficiency and performance improvement have been carried out through the years in the real- time scheduling policies. A number of Dynamic Voltage and Frequency Scaling (DVFS) strategies are implemented and evaluated in terms of their performance and energy savings. Blocking Aware Based Partitioning BABP algorithm has been designed in [10] to partition task set among cores to reduce overall power consumption while avoiding a deadline miss compared to other partitioning strategies. A novel predictive genetic algorithm based energy saving technique for task allocation has been proposed in [17],

where each processor operates within fixed power boundaries, taking into account the tasks' inter-dependencies, while [20] indicates that the genetic algorithm suffers from the lack of generalization in scheduling to reach to an optimal solution thus combining it with other heuristics produce better results. A voltage islands technique has been proposed in [16], which assigns each core to a specific voltage group so that each group operates according to a specific energy level and assigns periodic tasks, according to cores utilization, to the most appropriate core to execute. Donga [8] proposed a new reprioritization method to solve the priority inversion problem and increase schedule ability through eliminating context switching and hence improve system performance. In [24], the authors present an approach to calculate the processor frequency which commensurate with the largest task within the task set then using the time surplus to migrate some tasks from one core to another in order to achieve load balancing. [12] Aim at the extending real- time system operating time when system powered by renewable energy source by developing a new scheduling algorithm based on DVFS technique and task duplication strategy. The idea of duplication is to schedule a task graph by mapping some tasks in the set redundantly to reduce the communication between tasks. In [13], the researchers propose an energy efficient real- time scheduling algorithm for reordering chunks in between deadlines and taking advantage of the utilization of the processor to extend deadlines and lower processor frequency hence saving energy. [1] Propose a scheduling technique commensurate with the required improvement. If the goal is to reduce response time, then aperiodic tasks will be executed at full speed while periodic tasks migrates to other cores if their deadline is earlier than aperiodic tasks. On the other hand, if the goal is to reduce energy consumption, then aperiodic tasks will be executed with lower speed by identifying appropriate processor speed without affecting the response time. In [18] scheduling scheme, namely, the energy- aware fault- tolerant dynamic scheduling scheme (EFDTS), is developed to optimize resource utilization and reduce energy consumption. A task classification method is proposed to partition the ready tasks into different classes and then assign them to the most suitable processor based on their classes to minimize the response time while considering energy consumption. Replication is used to minimize the task rejection ratio. Furthermore, a migration policy is developed that can simultaneously improve resource utilization and energy efficiency. A simulation tool (SIMSO) presented in [5] is designed for the understanding of real-time scheduling algorithms and evaluate real- time system performance and overheads. SIMSO used by many studies such as [3], [14], [4] and [7].

From all of these previous researches, it may be concluded that there is a need to have load balance energy efficient method in case of light load or system overload without hardware modification or neglecting any of the performance metrics. Our study proposes such a method to have optimum energy consumption as well as optimum throughput.

1-7 Outline:

This paper is organized as follows: Section 2 defines types and classifications of scheduling in real-time systems. Mathematical model of the LLREF scheduling algorithm is presented in section 3. Section 4 proposes the Energy- Efficient Load Balancing (EELB) method. Section 5 reviews the scenarios and results. Finally, conclusions and future work are discussed in section 6

2- Scheduling in real- time systems:

The scheduler is the most important component in real- time systems and usually in the form of a short- term task scheduler. The primary focus of this scheduler is on reducing the response time associated with each task rather than dealing with a deadline.

Scheduling takes place after arranging all of the tasks that are ready for execution and the tasks that were excluded from the processor during their execution without completing them as a result of the arrival of another higher priority task within the task queue. Scheduling must ensure the completion of the execution of each task within the deadline associated with it, taking into consideration task priorities. The purpose of scheduling is to utilize most of the processor time without failures in the real- time system [6]. In addition, scheduling should limit simultaneous access to system resources in order to reduce conflicts between tasks and obtain a more efficient system, and make the scheduling process optimal. Scheduling process is said to be non- optimal if the deadline is missed even for only one task. [14]

2-1. Classifications of task scheduling:

In the following section we review some general classifications of task scheduling. Scheduling in real- time systems is divided in terms of the platform processors' count into two parts: [15]

2.1.1 Uniprocessor scheduling:

The task queue is assigned to the single processor on the platform for scheduling.

2.1.2 Multiprocessor scheduling:

The task queue is assigned to be executed on all the processors on the platform. Scheduling on a multiprocessor platform is divided into two parts:

a. Partitioned scheduling:

The task frame is divided into a set of sub- frames, each frame is will be scheduled on a single processor in the platform, as if the multiprocessor platform had been replaced by a group of single-processor platforms.

b. Global scheduling:

The task queue is distributed to all the platform processors at once in order to be executed so that a task is selected and mapped to the unoccupied processor in the platform. Figure (1) shows the diffrence between global and partitioned scheduling.

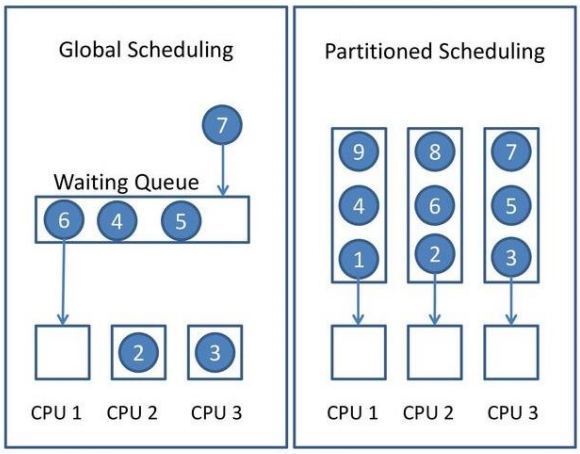

Figure (1) Global and partitioned scheduling

On the other hand, there is a second classification of scheduling in terms of being done before starting or during the system operation, as follows:

2.1.3 Static scheduling:

In this type of scheduling tasks arrival times should be well known by the system designer in order to be scheduled optimally, so this scheduling is mostly applied to periodic tasks.

The tasks in this category are scheduled during compilation time, as shown in figure(2), so the resulting overheads during the system operation in terms of switching between tasks, cache emptying and resources obtaining are very few compared to the dynamic category. Low overheads are considered as one of the features of this category. However, the inability of static scheduling to schedule sporadic and aperiodic tasks, because it is hard to predict the arrival time of such tasks, constitute the most important disadvantages of this category.

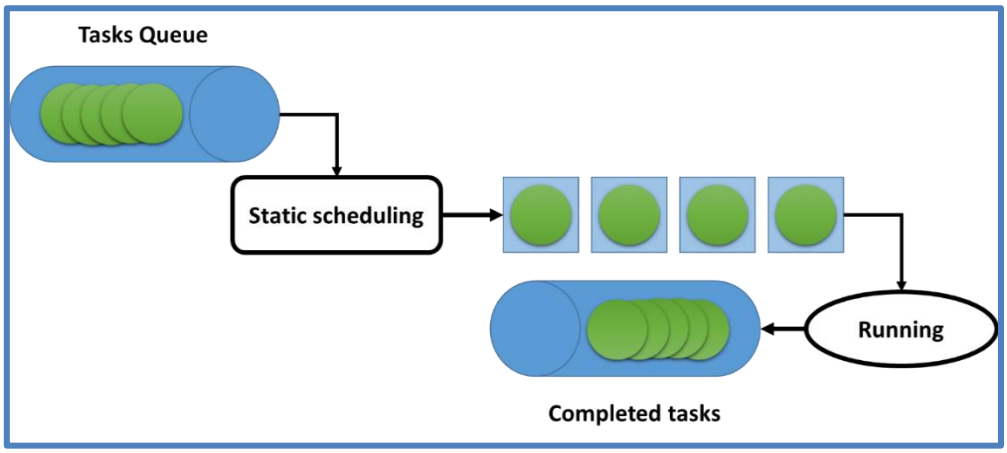

Figure (2) Static Scheduling

2.1.4 Dynamic scheduling:

Dynamic scheduling is the mechanism in which a task is scheduled by the scheduler at the runtime.

The tasks in this category have different priorities. The priority is assigned to tasks during execution according to the rules determined by the chosen scheduling algorithm. Dynamic scheduling is faster and more efficient because the compilation time is small compared to the previous category (static) therefore the system starts to run earlier. The ability to schedule sporadic tasks and aperiodic tasks is one of the most important features of this category. However, there are many burdens facing dynamic scheduling during operation in terms of context switching between tasks, cache emptying and resource obtaining which can cause a greater delay due to the fact that the demand for these resources is done dynamically during the system operation. This overhead considered the main disadvantages of this category.

Based on the characteristics of each of the previous two categories, we can say that choosing any of them when building a real- time system depends on the nature of the tasks in the system. Scheduling is also divided in terms of tasks migration strategies into three types:

2.1.5 Partitioning migration scheduling:

When selecting a task to be executed on one of the processor cores on the platform, each succession or repetition of the task, which is called "Job", at the beginning of each period will also be executed on the same processor core.

2.1.6 Restricted migration scheduling:

When selecting a task to be executed on one of the processor cores on the platform, the rest “jobs” of the current task can be executed either on the same processor core or on any other cores in the processor. However, when the “job” begins execution, it will definitely complete its execution exclusively on the current processor core.

2.1.7 Full migration scheduling:

When the “job” of a task begins executing on one of the processor cores, the job can complete its execution on any other core if necessary as a result of job exclusion from execution for a reason related to priorities. Full migration scheduling applies to all types of tasks. [2][23][4]

In terms of the nature of events, scheduling is classified into two categories:

2.1.8 Event driven scheduling:

The scheduler is called upon the arrival of a specific event in order to consider the status of the existing tasks and determine the tasks that will be taken for execution from among the set of tasks in the queue according to the preferences adopted by studied algorithm. This event can be the release of a new task or termination of a specific task or any other event related to the scheduling algorithm type. Figure (3) shows the difference between event driven and non- event driven scheduling in terms of response time.

2.1.9 Quantum driven scheduling:

The scheduler is invoked to make a scheduling decision when a certain period of time has passed. The process is done periodically, and the length of the period depends on the nature of the tasks in the queue, as well as on the nature of the processors on the platform.

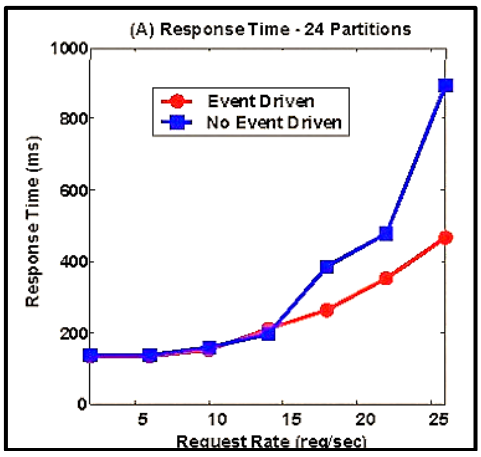

Figure (3) Event Driven Scheduling

3- LLREF scheduling algorithm.

LLREF scheduling algorithm belongs to the dynamic type, i.e. it dynamically prioritizes tasks during system execution. LLREF is also event- driven, meaning tasks rearranged when a specific event occurs.

In terms of the nature of priorities assignment, LLREF scheduling algorithm is dynamic at the "job" level, this means it is based on full migration strategy where the "job" of the current task can resume its execution either on the same core on which it was started before being excluded due to priorities, or on any other processor core on the platform. The events in this algorithm are divided into two parts, the main events and the secondary events. The main events in this algorithm are:

- Task release event: It is the event of releasing a specific task and joining to the task queue which is served by the processor cores on the platform.
- End of task execution event: It is the event of terminating the current task by one of the processor cores, removing it from the task queue, and allowing a new task to enter the queue.

The secondary events in this algorithm are:

- Ceiling event (C): This event is triggered when there is no additional time left for execution, which means the execution of the current task cannot be delayed so that it does not violate its time constraint (miss the deadline).
- Bottom event (B): This event is triggered when the current task finishes the amount of work it has to do during the TL- Plane (Time and Local Execution time Plane) which requires calling the scheduler exceptionally to schedule the tasks again and add a new task to the queue. [22]

3.1 Mathematical models of tasks in real- time systems:

The mathematical model of the tasks studied in this research can be formulated as follows:

Assuming that we have a queue of tasks (Q) containing a set of periodic (P), sporadic (S), and aperiodic (A) tasks to be scheduled on a multi- core processor, then the model of periodic tasks can be described as follows:

3.1.1 Periodic task model:

- Task arrival time (t_p): It is the time at which an acknowledgment of a new event is issued on the real- time system, where tasks are expected to arrive according to equal time periods. Periodic task arrival time can be calculated by formula:

$$\forall P \in Q \Rightarrow t_P(n+1) = t_P(n) + \tau \ (1)$$

- Execution time (E_p): It is the processing time that occupied by the task being executed on the processor.
- Deadline (D_p): It is the time during which a task must finish executing before reaching it.

Noting that the periodic tasks in this research have an implicit deadline as:

$$\forall P \in Q \Rightarrow \tau_P = D_P \ (2)$$

3.1.2 Sporadic task model:

- Task arrival time (t_s): It is the time at which an acknowledgment of a new event is issued in the real-time system. In sporadic task model it is stipulated that no consecutive events for the same task may occur at the same time. Sporadic task arrival time can be calculated by formula:

$$\forall S \in Q \Rightarrow t(n)_S \neq t(n+1)_S \ (3)$$

- Execution time (E_s): It is the processing time that occupied by the task being executed on the processor.
- Deadline (D_s): It is the time during which a task must finish executing before reaching it.
- Number of sporadic task arrivals (N_s): Indicating the number of times the sporadic task was released during the execution period[11].

3.1.3 Aperiodic task model:

- Task arrival time (t_A): It is the time at which an acknowledgment of a new event is issued on the real- time system, and unlike sporadic tasks, consecutive events for the same task can arrive with a time difference greater than or equal to zero. Aperiodic task arrival time can be calculated by formula:

$$\forall A \in Q \Rightarrow t(n)_A \leq t(n+1)_A \ (4)$$

- Execution time (E_A): It is the processing time that occupied by the task being executed on the processor.

- Deadline (D_A): It is the time during which a task must finish executing before reaching it.
- Random task arrival number (N_A): Indicating the number of times the random task was released during the execution period[9].

3.2 Multi- core processor model:

The processor studied in this paper is a multi- core processor that contains two, four or eight cores, according to the studied scenario. These cores are identical, and each core has its own L1 cache memory, and the cores share the second level of cache L2 cache memory.

3.4 Mathematical models of LLREF algorithm:

First, the algorithm calculates the laxity time for each task, which is the difference between the deadline and the task actual execution time as defined by the formula:

$$\forall T: \{ P, S, A \} \in Q \Rightarrow L_T = D_T - E_T \quad (5)$$

Where: (L_T): Laxity time.

(D_T): Deadline.

(E_T): Execution time.

Then the algorithm sorts the tasks in ascending order according to the amount of laxity time (L_T). The ordered tasks at the front of queue are mapped to the processor cores so that each task is assigned to one core. Task execution continues until one of the following three events is triggered:

- An incoming task has released.
- Bottom event.
- Ceiling event.

When a new task released scheduler is then called to calculate the laxity time and reorder the tasks. As for the event (B), it occurs when a task finishes execution, and therefore the scheduler is called in order to make the scheduling decision again. The event (C) occurs when the amount of laxity time is zero, which means:

$$L_T = D_T - E_T = 0 \quad (6)$$

The scheduler then called in order to make the scheduling decision and put such tasks into execution; otherwise they will miss their time constraint (deadline).

The flowchart shown in Figure (4) describes how the LLREF algorithm works.

3.5 Power consumption model:

The total power consumed in the processor is defined as the sum of dynamic and static power as follows:

$$P\_dissTotal = P\_Dynamic + P\_Static \quad (6)$$

$$P\_Dynamic = C.V\text{^}2.F \quad (7)$$

V: processor voltage.

F: processor frequency.

C: proportionality constant.

$$P\_Static=V.I\_Leak \quad (8)$$

I_Leak: leakage current.[21] [19]

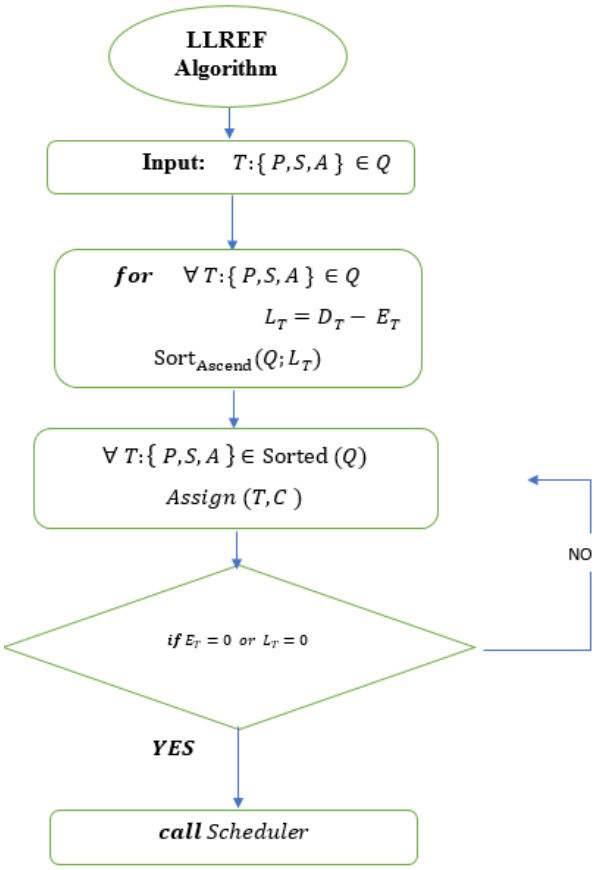

Figure (4) LLREF Algorithm

4- The proposed Energy- Efficient Load Balancing (EELB) method:

The main idea of proposed Energy- Efficient Load Balancing method is to adjust the frequency of the processor cores in the multi- core processor so that it is possible to increase or decrease the frequency according to the type, number and nature of the executed tasks. Accordingly, the proposed method will improve the performance of real- time systems by either running it normally with less power consumption in case of a light to medium load or allowing a greater number of tasks to be executed in case of system overload. The proposed method calculates optimal frequency value via implementation of PSO (Particle Swarm Optimization) algorithm. The PSO objective function is set to be the frequency value in case of light to medium load or the number of failed tasks in case of system overload.

5- Scenarios and results:

In the following section, we will review a set of proposed scenarios for applying the enhancement approach to a set of periodic, sporadic, and aperiodic tasks using a multi- core processor, taking into account the performance measurement criteria including context switching overheads, scheduling overheads, number of failed tasks and power consumption.

Table (1) Parameters of light load scheduling scenario

Number of cores	[2, 4, 8]
Number of tasks	20
Periodic tasks percentage	70 %
Sporadic tasks percentage	20 %
Aperiodic tasks percentage	10 %
L1 cache size	[32, 64, 128] KB respectively
L2 cache size	[512, 1024, 2048] KB respectively
Line size	16 Byte
Simulation time	5000 msec

5.1 Light load scenario:

In this scenario, the performance of the EELB method was compared with the default method in terms of power consumption during light based on the data shown in the table (1).

Figure (5) shows the light load power consumption scenario results when working on two, four and eight cores.

5.2 Context switching scenario:

In this scenario, the performance of the proposed method was compared with the default method in terms of context switching overhead based on the data shown in the table (2).

Figure (6) shows context switching overheads scenario results when working on two, four and eight cores.

Table (2) Parameters of scheduling overheads scenario

Number of cores	[2, 4, 8]
Number of tasks	**100**
Periodic tasks percentage	**70 %**
Sporadic tasks percentage	**20 %**
Aperiodic tasks percentage	**10 %**
L1 cache size	**[32, 64, 128] KB respectively**
L2 cache size	**[512, 1024, 2048] KB respectively**
Line size	**16 Byte**
Simulation time	**5000 msec**

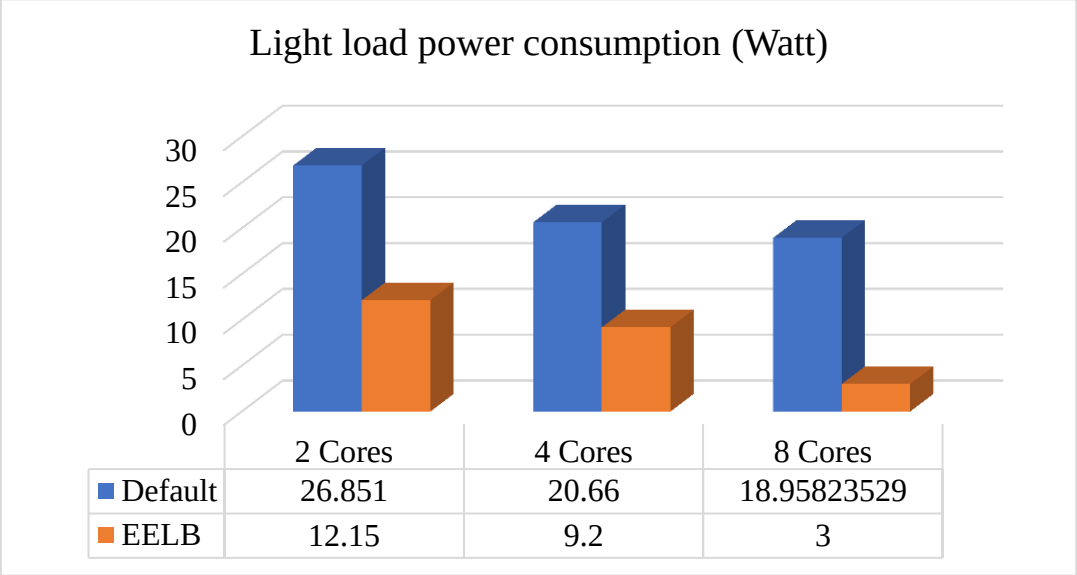

Figure (5) Light load power consumption scenario

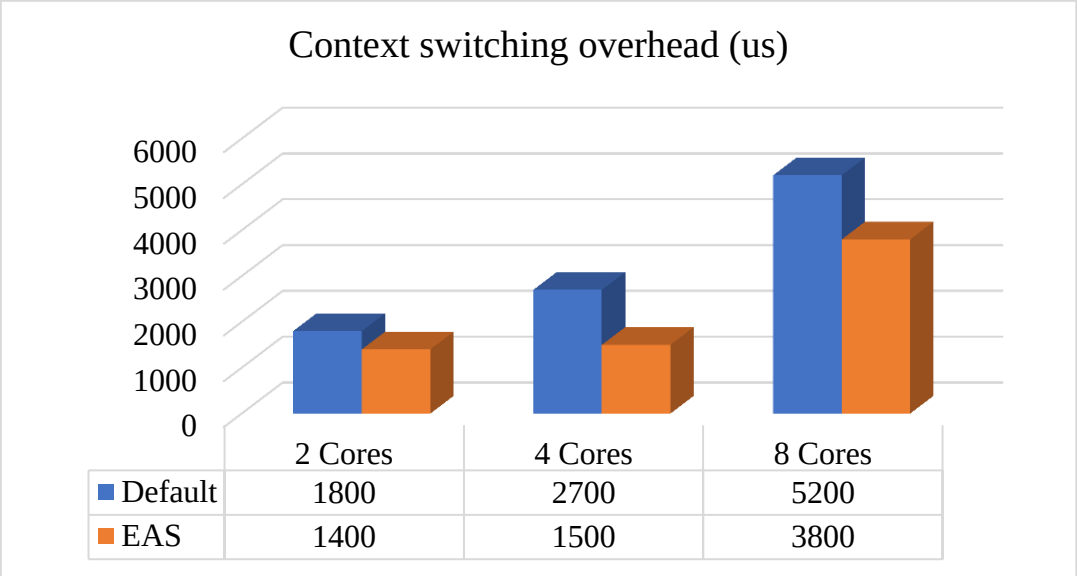

Figure (6) Context Switching Overheads Scenario

5.3 Scheduling overheads scenario:

In this scenario, the performance of the EELB method was compared with the Default method in terms of scheduling overheads based on the data shown in the table(2).

Figure (7) shows scheduling overheads scenario results when working on two, four and eight cores.

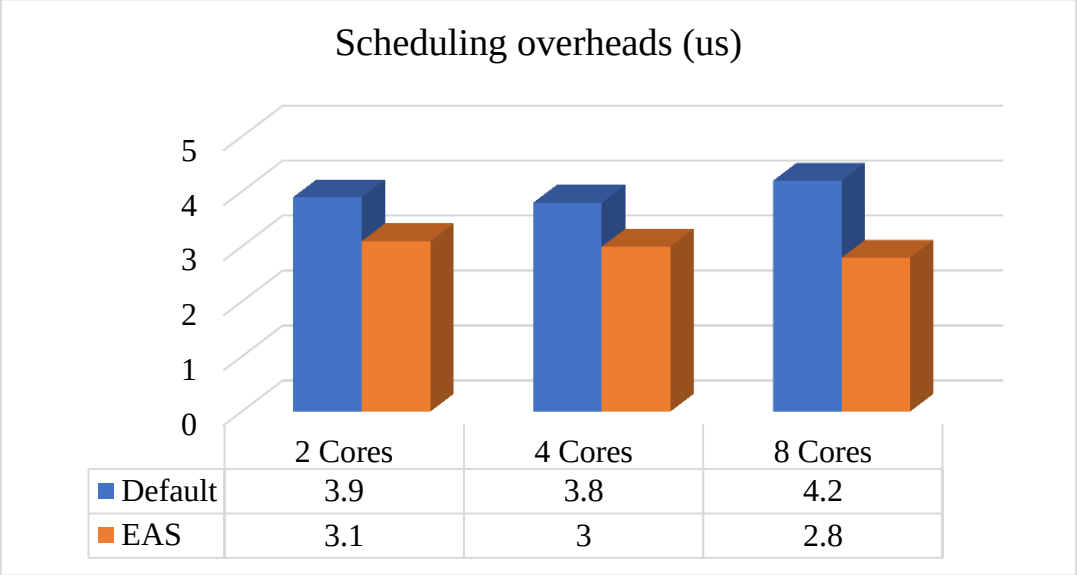

Figure (7) Scheduling overheads scenario

5.4 Number of failed task scenario:

In this scenario, the performance of the EELB method was compared with the default method in terms of number of failed tasks based on the data shown in the table (2).

Figure (8) shows the number of failed task's scenario results when working on two, four and eight cores.

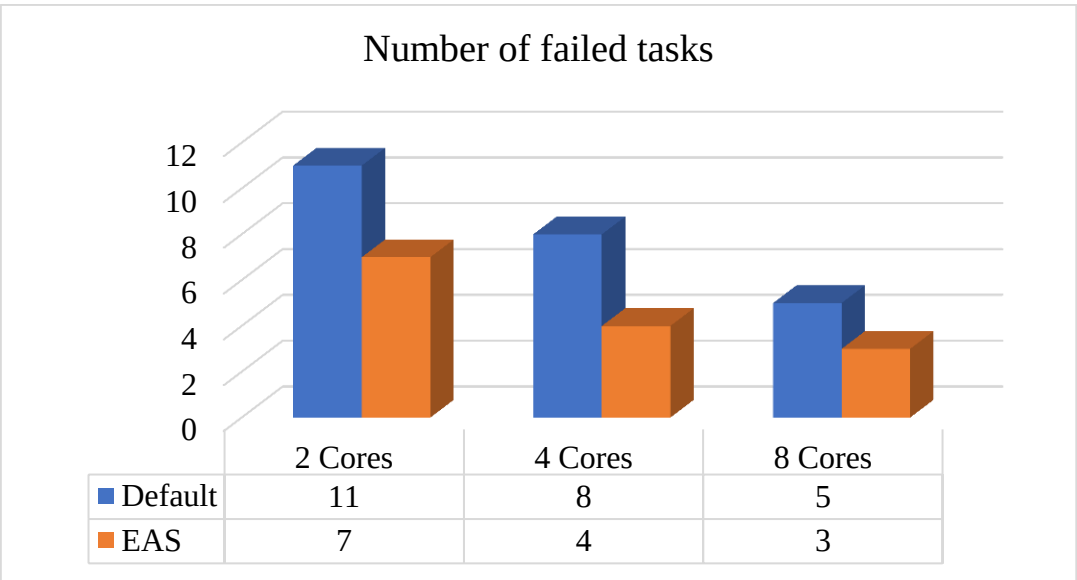

Figure (8) Number of failed tasks scenario

5.5 Power consumption scenario

In this scenario, the performance of the EELB method was compared with the Default method in terms of power consumption based on the data shown in the table (2).

Figure (9) shows power consumption scenario results when working on two, four and eight cores.

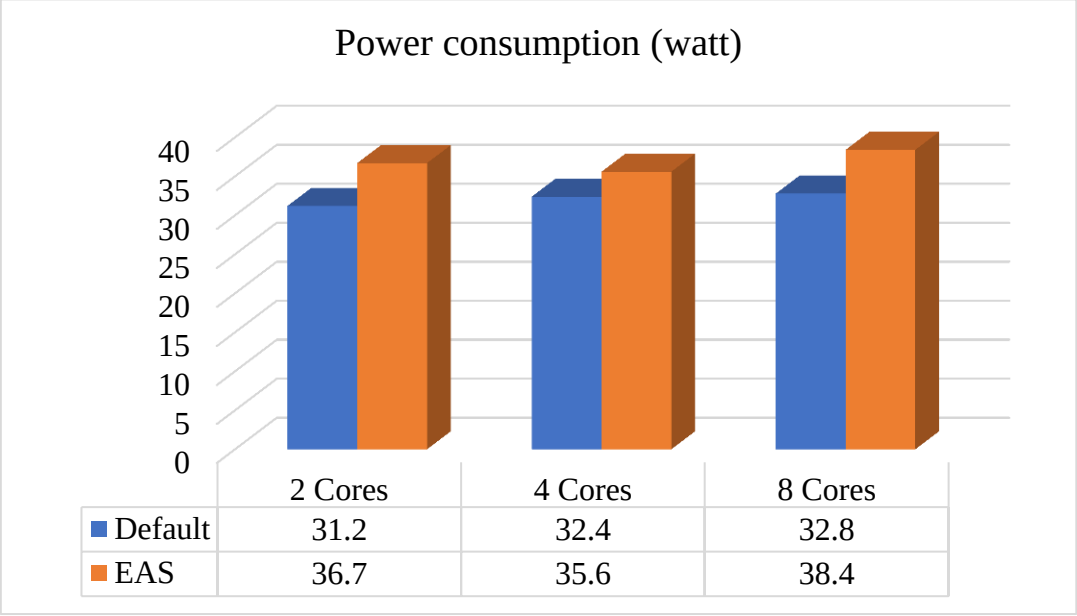

Figure (9) Power consumption scenario

5.6 Discussion.

After reviewing the results obtained in this research, we can point a number of conclusions, which we can summarize in the following:

1- The proposed EELB method reduced the impact of context (30- 40) %. This leads to improved resource management and reducing its consumption as much as possible. A slight increase in the processor frequency led to the speed of task execution and thus reducing the need to interrupt the execution of tasks and its context switching.

2- The proposed EELB method reduced scheduling overheads by (12- 23) % due to fewer context switches and thus reduces the need for making scheduling decisions.

3- The proposed EELB method contributed to reducing the number of tasks that may miss their deadlines by (30- 45) %, and the largest percentage was when working on two cores, which highlights the role of the proposed method in improving the performance of real- time systems when working with fewer resources.

4- The proposed EELB method led to an increase in energy consumption by (8- 14) %, and this increase is considered slight compared to the other benefits achieved at the level of the previous three parameters.

6- Conclusion:

This work discussed real- time systems in terms of their types, classification and importance, then the task scheduling in real- time systems was explained and their types and characteristics were reviewed. The LLREF scheduling algorithm selected known to be the most energy efficient algorithm according to [22] due to its simplicity and fast decision making. EELB method proposed in this paper aims to improve

the LLREF scheduling algorithm in a way that balances between system load and energy consumption. It depends on using PSO to find the optimal frequency value to achieve EELB goals the simulation results illustrate that our algorithms made significant improvement when compared to default executing state. We have shown via proof and simulation that power consumption reduced when load is light and system performance increased when the system is overloaded.

6.1 Recommendations and future work:

The future work of this research would be, to develop the proposed method (EELB) and test it in different working conditions, among which are:

1- Using another optimization algorithm.
2- The possibility to apply additional scenarios containing a different number of tasks and processor cores.
3- Generate different utilization rates for the processor.
4- Testing the proposed method on processors with physical hardware that requires greater energy consumption and studying the effect of the proposed method on improving energy consumption when working on these platforms.

References.

[1] Ali, A., & Zakarya, M. (12 2020). POWER AND PERFORMANCE BASED HYBRID MULTIPROCESSING FOR REAL- TIME SCHEDULING. doi:10.13140/RG.2.2.12494.28484

[2] Anjaria, K(18)., & Mishra, A. (2017). Thread scheduling using ant colony optimization: An intelligent scheduling approach towards minimal information leakage. Karbala International Journal of Modern Science, 3(4), 241–258. doi: 10.1016/j.kijoms.2017.08.003

[3] Ara, G., Cucinotta, T., & Mascitti, A. (2022). Simulating Execution Time and Power Consumption of Real- Time Tasks on Embedded Platforms.

[4] Bertout, A., Forget, J., & Olejnik, R. (10 2014). Minimizing a real- time task set through Task Clustering. doi:10.1145/2659787.2659820

[5] Chéramy, M., Hladik, P.- E., & Déplanche, A.- M. (2014, Julie). SimSo: A Simulation Tool to Evaluate Real- Time Multiprocessor Scheduling Algorithms. 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real- time Systems (WATERS), 6 p. Opgehaal van https://hal.archives- ouvertes.fr/hal- 01052651

[6] Christine Niyizamwiyitira, Lars Lundberg, A Utilization- Based Schedulability Test of Real- Time Systems Running on a Multiprocessor Virtual Machine, The Computer Journal, Volume 62, Issue 6, June 2019, Pages 884–904, https://doi.org/10.1093/comjnl/bxz005

[7] De Bock, Y., Broeckhove, J., & Hellinckx, P. (2015). Hierarchical Real- Time Multi- core Scheduling through Virtualization: A Survey. doi:10.1109/3PGCIC.2015.32

[8] Donga, M. J., & Holia, M. S. (2021). A Hybrid Multiprocessor Scheduler for Soft Real- Time System with Processor Affinity Concept. Journal of Physics: Conference Series, 2089(1), 012022. doi:10.1088/1742- 6596/2089/1/012022

[9] El Osta, R., Chetto, M., & El Ghor, H. (2021). An optimal energy aware aperiodic task server for autonomous IoT sensors. International Journal of Electrical and Computer Engineering Research, 1(2).

[10] El Sayed, M. A., Saad, E. S. M., Aly, R. F., & Habashy, S. M. (2021). Energy- Efficient Task Partitioning for Real- Time Scheduling on Multi- Core Platforms. Computers, 10(1), 10. https://doi.org/10.3390/computers10010010

[11] Funk, S., & Nanadur, V. (10 2009). LRE- TL: An Optimal Multiprocessor Scheduling Algorithm for Sporadic Task Sets.

[12] Ghonoodi, A. (2019). Green Energy- aware task scheduling using the DVFS technique in Cloud Computing. Journal of Advances in Computer Research, 10(1), 1- 10.

[13] Han, L., Canon, L.- C., Liu, J., Robert, Y., & Vivien, F. (2019). Improved Energy- Aware Strategies for Periodic Real- Time Tasks under Reliability Constraints. 2019 IEEE Real- Time Systems Symposium (RTSS), 17–29. doi:10.1109/RTSS46320.2019.00013

[14] Hladik, P.- E. (04 2018). A brute- force schedulability analysis for formal model under logical execution time assumption. 609–615. doi:10.1145/3167132.3167199

[15] Levin (5), G., Funk, S., Sadowski, C., Pye, I., & Brandt, S. (2010). DP- FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. 2010 22nd Euromicro Conference on Real- Time Systems, 3–13. doi:10.1109/ECRTS.2010.34

[16] Liu, Jun & Guo, Jinhua. (2015). Energy efficient scheduling of real- time tasks on multi- core processors with voltage islands. Future Generation Computer Systems. 56. 10.1016/j.future.2015.06.003.

[17] Mahmood, A., Khan, S., Albalooshi, F., & Awwad, N. (2017). Energy- Aware Real- Time Task Scheduling in Multiprocessor Systems Using a Hybrid Genetic Algorithm. Electronics, 6(2), 40. https://doi.org/10.3390/electronics6020040

[18] Marahatta, A., Wang, Y., Zhang, F., Sangaiah, A. K., Tyagi, S. K., & Liu, Z. (2019). Energy- Aware Fault-Tolerant Dynamic Task Scheduling Scheme for Virtualized Cloud Data Centers. Mob. Netw. Appl., 24(3), 1063–1077. doi:10.1007/s11036- 018- 1062- 7

[19] Peronaglio(15), F., Manacero, A., Lobato, R., & Spolon, R. (04 2017). Modeling Real- Time Schedulers for Use in Simulations Through a Graphical Interface. 10. doi:10.22360/springsim.2017.anss.020

[20] Pradhan, S., Sharma, S., Konar, D., & Sharma, K. (07 2015). A Comparative Study on Dynamic Scheduling of Real- Time Tasks in Multiprocessor System using Genetic Algorithms. International Journal of Computer Applications, 120, 975–8887. doi:10.5120/21340- 4346

[21] Priesmann, J., Nolting, L., Kockel, C. et al. Time series of useful energy consumption patterns for energy system modeling. Sci Data 8, 148 (2021). https://doi.org/10.1038/s41597- 021- 00907- w

[22] Prifti, V., Balla, B., Zane, R., Fejzaj, J., & Tafa, I. (2014). A Comparison of Global EDF and LLREF Scheduling Algorithms.

[23] Rihani(33), H. (2017). Many- Core Timing Analysis of Real- Time Systems (Université Grenoble Alpes). Opgehaal van https://tel.archives- ouvertes.fr/tel- 01875711

[24] Wu, X., Lin, Y., Han, J.- J., & Gaudiot, J.- L. (2010). Energy- Efficient Scheduling of Real- Time Periodic Tasks in Multi- core Systems. In C. Ding, Z. Shao, & R. Zheng (Reds), Network and Parallel Computing (bll 344–357). Berlin, Heidelberg: Springer Berlin Heidelberg.