A Survey of Energy Efficient Scheduling Algorithms for Real-time Systems over Parallel Machines

Muhammad Zakarya[1], Nazar Abbas[2], Ayaz Ali Khan[3]

[1]Department of Computer Science, Abdul Wali Khan University, Mardan

[2,3]COMSATS Institute of Information Technology, Islamabad

Abstract:

A real-time system is defined as any information processing system which has to respond to externally generated input stimuli within a finite and specified period, in other words it is defined as the ability of the system to guarantee a response after a fixed time has elapsed. Computational real-time systems are increasingly used to control tasks in numerous application fields like aircraft, automotive and manufacturing process etc. Systems such as chemical and nuclear plant control, flight control systems, space missions, digital control, military systems, telecommunications, and multimedia systems all make use of real-time technologies. The most important attribute of real-time systems is that the correctness of such systems depends not only on the computed results but also on the time at which results are produced. In other words, real-time systems have timing requirements that must be met. Since the timing constraints are the most important characteristic of real-time systems, they are classified as hard or soft according to the usefulness of the computed results produced after the timing deadlines Parallel systems including HPC, multiprocessors, multicores, clusters, grids, clouds and distributed systems are very promising from performance perspective, however higher power consumption issues arises as a challenge associated with such systems.

Keywords: *Scheduling, Operating Systems, Parallel Systems, RTS, Energy Efficiency*

1. Introduction:

Real-time systems are paying attention on periodic task models, in which tasks are released at habitual time periods. On the other hand with maturity of multiprocessor structural design, today most real-time systems function in dynamic environment where human activities (aperiodic tasks) are predictable. Aperiodic tasks are to be completed as soon as possible; consequently the priority assigned to such aperiodic tasks ought to be higher than those of periodic tasks. Multiprocessor are very promising from performance perspective, however higher power consumption issues arises as a challenge associated with such systems. Since these systems generally remain under-utilized and the systems operate at maximum speed throughout and thus becomes an ideal candidate for power aware scheduling. When we reduce the energy consumption then the response time is increased. And it will degrade the performance of real time systems. In this paper we have summarized a number of scheduling algorithms for real-time applications that might result in less power consumption while system performance in maintained to the best level. In the following section we have to the point review of some general terms in this aspect. Real-time Systems are divided into two major classes.

- Hard real-time systems &
- Soft real-time systems

In hard real-time systems it is a must to carry out the deadline requirements. Hard real-time tasks cannot let pass any deadline, otherwise, disagreeable or deadly results will be produced in the system. Special purpose systems are considered hard. This is crucial in situations where system failure may result in damage or loss of life [6]. As compared to hard real-time systems, missing a deadline only degrades the overall system predictability in soft real-time systems. Although missing deadlines is not pleasing in a real-time environment, soft real-time tasks could miss some deadlines and the system could still work acceptably. In soft real-time systems however, lateness have no severe effect. There is some space for delayed operation and the failure has no such adverse effect on the entire system but decrease in a process or system quality. Such a system is called tolerant of incorrect completion times because the undesirable results can however be tolerated by these systems [16]. In maximum real-world environments such as space or airborne platform management, factory process control, aerospace and defense systems, nuclear systems, robotics, stock exchange, multimedia computing, medical systems and embedded intelligent devices, the real-time system is widely applied [17, 20]. A real-time system is bounded by the boundaries of time and load, and its goal is to properly schedule the tasks and make sure as more tasks as possible to be accomplished before their deadlines [19].

2. Real-Time Scheduling Theory

Predictability is the main constraint in a real-time system which distinguishes it from traditional computing system. This predictability to the real-time system is provided by scheduling algorithms and RTOS, which manage and schedule resources and tasks to reach the deadlines of the real-time system so that they may not be missed and to timely respond to the real-world environment. A scheduling algorithm executes a task at specific time. Two main types of scheduling algorithms exist.

i. Preemptive scheduling algorithms
ii. Non-preemptive scheduling algorithms.

In preemptive scheduling algorithm, the processor / resource is preempted from a low priority task when a higher priority task requests the same resource and the higher priority task gets starting. So such a method is used for tasks priorities assignment. In non-preemptive scheduling algorithms, the currently executing task is not preempted until and unless it completes its execution.

Priority-Driven Scheduling:

For understanding the importance of priority based scheduling, we consider an example presented in [24]. Suppose that a system has only two periodic tasks τ_1 and τ_2 having periods 50 and 100 respectively. Both are initiated at critical instant. Assume that both τ_1 and τ_2 have worst case execution times of $C_1 = 25$ ms and $C_2 = 40$ ms respectively. Processor utilization of τ_1 is, $U_1 = 25/50 = 0.5 = 50$ % and τ_2 *has*, $U_2 = 40/100 = 0.4 = 40$ % and the total requested CPU utilization is $U = U_1 + U_2 = 50 + 40 = 90$ %. So total of 90 % CPU is used. Here the author [24] considers two cases to execute τ_1 and τ_2.

Case1: Assigning priorities by using a static priority scheduling algorithm where priority (τ_1) > priority (τ_2).

Case 2: Assigning priorities by not using any scheduling algorithm where priority (τ_2) > priority (τ_1). Both of the cases are illustrated by the following figures. In Fig 2, both tasks meet their respective deadlines. In Fig 3, however, task1 misses a deadline, even though 10 % CPU idle time is available.

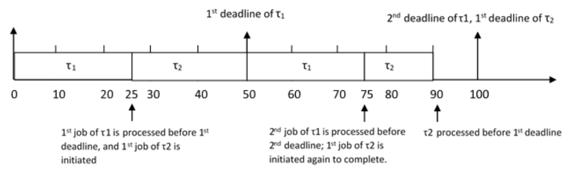

1st deadline of τ1
2nd deadline of τ1, 1st deadline of τ2
τ1
τ2
τ1
τ2
0 10 20 25 30 40 50 60 70 75 80 90 100
1st job of τ1 is processed before 1st deadline, and 1st job of τ2 is initiated
2nd job of τ1 is processed before 2nd deadline; 1st job of τ2 is initiated again to complete.
τ2 processed before 1st deadline

Fig 2: ***Priority (τ_1) > Priority (τ_2)***

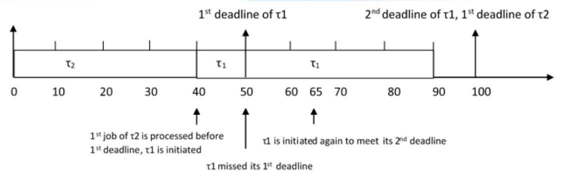

1st deadline of τ1
2nd deadline of τ1, 1st deadline of τ2
τ2
τ1
τ1
0 10 20 30 40 50 60 65 70 80 90 100
1st job of τ2 is processed before 1st deadline, τ1 is initiated
τ1 is initiated again to meet its 2nd deadline
τ1 missed its 1st deadline

Fig 3: ***Priority (τ_2) > Priority (τ_1)***

In fact many more new tasks can be added into the system by using a static priority scheduling algorithm because the total processor utilization is only 90 % and remaining 10 % CPU time is available for executing other tasks. This example shows that how much properly a priority-driven scheduling algorithm can schedule tasks.

In scheduling phenomenon, a *priority* is typically a positive integer representing the hurry or importance assigned to an activity. By convention, the hurry is in inverse order to the numeric value of the priority, as priority 1 is the highest level of priority. We shall assume here for simplicity purpose that a task has a single, fixed priority. On the given condition's we can consider the following two simple scheduling disciplines:

Preemptive priority based execution:

When the processor is at rest, the ready task with the peak priority is chosen for execution; but still at any stage of time, the execution of a specific task can be preempted if a task of superior priority becomes into a ready state. As a result, at all times the processor is either at rest or executing the ready state task with the utmost priority. Scheduling is a priority motivated/based technique. Therefore scheduling of tasks in a hard real time systems are classified into two wide-spectrum categories

i. Fixed priority based scheduling
ii. Dynamic priority based scheduling

For a given set of jobs, the general scheduling problem asks for sorted form according to which the jobs are to be executed in such a way that different constraints are fulfilled. Generally, a job is characterized by its processor execution time, ready state time, task deadline (validity duration), and resource requirements. The execution of a job may or may not be interrupted i.e. (preemptive scheduling or non-preemptive scheduling). On behalf of the set of jobs, there is a preference relation which constrains the organized form of execution. In particular, the execution of a job cannot set in motion until the execution of all its predecessors (according to the priority relation) is completed. On the system in which the jobs or tasks are to be executed is characterized by the amounts of resources, available the following basics goals should be considered strictly in dealing with scheduling problems in a real-time system:

i. Meeting the timing constraints (processor execution time, ready state time) of the real-time system
ii. Preventing concurrent access to shared resources and multiple shared devices
iii. Attaining a high grade of utilization while satisfying the timing constraints of the real-time system; however this is not a most important driver.
iv. Dropping the cost & charge of context switches caused by preemption in scheduling algorithm
v. Plummeting the communication cost in real-time distributed systems; we should find the most advantageous way to decompose the real-time applications into smaller parts or portions, in order to have the smallest amount of communication cost between mutual portions (where each & every portion or part of a real-time application is assigned to a single computer).

Additionally, the following attributes and objectives are much loved in advanced real-time systems:

a. Studying a combination of hard real-time, firm real-time, and soft real-time activities which implies the livelihood of applying dynamic scheduling policies that respect the optimality criteria.
b. Task scheduling for a real-time system whose behavior and attitude is dynamically adaptive, reconfigurable, and consequently reflexive and intelligent as well.
c. Considering reliability and dependability of different tasks, security, and safety.

3. Scheduling Algorithms

a. Earliest Deadline First (EDF) Scheduling

As the name describes, the earliest-deadline-first scheduling algorithm uses the deadline of a task as its priority. The task comprises the earliest deadline has the highest priority, while the task with the latest deadline has the lowest priority and vice versa. One characteristics of this scheduling algorithm is that the schedulable bound is 100% for all task sets. Secondly, because priorities are assigned in a dynamic pattern, therefore the periods of tasks can be changed at any time. One of the most important problems with the EDF scheduling algorithm is that there is no way to assurance which tasks will fail in a transient overloaded situation.

One of the most famous used algorithms belonging to task scheduling algorithms family is the EDF scheduling algorithm, according to which priorities assigned to tasks are inversely proportional to the absolute deadlines of the active jobs on the system. The feasibility analysis of periodic task sets under EDF scheduling algorithm was first presented in 1973 by Liu and Layland, who presented in their respective paper on the subject, that, under the same simplified assumpt[illegible] for Rate Monotonic (RM) scheduling al $\sum_{i=1}^{n} U_i \leq 1.$ of n periodic tasks is schedulable by t duling algorithm, if and only if

In 1974, Dertouzos showed in his research that EDF scheduling algorithm is optimal and feasible amongst all preemptive scheduling algorithms, in the sense that, if there exists a feasible schedule for a task set by any scheduling algorithm, then the schedule produced by EDF scheduling algorithm is also feasible. Later, Mok presented another optimal algorithm, called Least Laxity First (LLF), a detail is available in [22], which assigns the processor to the active task with the smallest laxity and carelessness. Although both LLF scheduling algorithm and EDF scheduling algorithm are optimal algorithms, but still LLF scheduling algorithm has a larger overhead due to the higher number of context switches caused by laxity and carelessness changes at run time. For this reason, most of the work done in the real-time research society concentrated on EDF scheduling algorithm to relax some unsophisticated suppositions and assumptions and extend the feasibility analysis to more general cases.

The Earliest Deadline First (EDF) scheduling algorithm is the most extensively and widely used scheduling algorithm for real-time systems. For a set of preemptive tasks (are they periodic, a-periodic, or sporadic), EDF scheduling algorithm will find a schedule if a schedule is possible. The application of EDF scheduling algorithm for non-preemptive tasks is not as widely studied. EDF scheduling algorithm is optimal for sporadic non-preemptive tasks, but EDF scheduling algorithm may not find an optimal schedule for periodic and a-periodic non-preemptive tasks; it has been shown that scheduling periodic and a-periodic non-preemptive tasks is NP-hard.

On the other hand, non-preemptive EDF techniques have produced near to most favorable schedules for periodic and a-periodic tasks, principally when the system is lightly loaded. When the system is overloaded, however, it has been shown that EDF approach leads to dramatically and vividly poor performance (low success rates).

b. Rate Monotonic Scheduling

Rate monotonic is a fixed priority scheduling algorithm for the real time systems. Fixed priority means that the priority of a task is not change during its execution means that same priority is assigned to all jobs of a task. Priority is assigned according to the period of a task. Period of a task mean's the time after which a next job of a task is released. So in rate monotonic the priority based on its period. So in the fixed priority scheduling algorithm the period of a task is compared and assign high priority to the task have small period. Alternatively the rate of tasks is inverse to the period, so jobs with higher rate have higher priority.

c. Deadline Monotonic Scheduling

Deadline monotonic is a fixed priority scheduling algorithm for the real time systems. Fixed priority means that the priority of a task is not change during its execution means that same priority is assigned to all jobs of a task. Priority is assigned according to the relative deadline of a task, shorter the deadline higher the priority. Deadline monotonic is equal to rate monotonic when Di=Pi.

d. Maximum Urgency First Scheduling

MUF is a mixed priority scheduling. MUF is the combination of mixed and dynamic priority scheduling algorithm. MUF define urgency for each task. Urgency is a combination of two fixed priorities (criticality and user priority) and a dynamic priority. MUF assign priorities in two phases.
First phase: Consist of three steps.

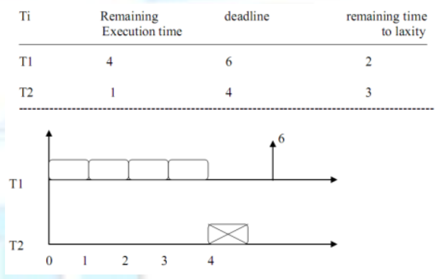
1) Sort the tasks according to its periods then define critical set.
2) Task in critical set is high priority and other task is low priority.
3) Optimal unique priority is assign to each task.

Second phase: Consist of three steps:

1) If there is only one critical task it executes it.
2) If there is more than one critical task execute the one with highest dynamic priority. The task with least laxity is considered to be the highest priority.
3) If there is more than one task with the same laxity select the one with the highest user priority.

e. MMUF Scheduling

Due to MUF scheduling mechanism critical task fail. MUF select the task with maximum laxity the remaining execution time of T1 is greater than T2 laxity so T2 miss its deadline, as shown in example below.



Ti	Remaining Execution time	deadline	remaining time to laxity
T1	4	6	2
T2	1	4	3

MMUF is the new version. The aim of MMUF is to solve the problem of MUF. MMUF use a unique importance parameter. MMUF either use EDF or MLLF to define the dynamic priority. MMUF is used instead of LLF because it reduced the context switching.
MMUF consist of two phases.
Fixed priorities:

1. Order the tasks according to its importance.
2. Define the critical task for the first N tasks so the total CPU load factor does not exceeds from 100%

Dynamic priority:

1. If there is it least one critical task in the ready queue

(a) Select the one with earliest deadline

(b) If more than one task has the same critical task then select the one with highest impotence.

2. If there is no critical task in a ready queue.

(a) Select the one with earliest deadline

(b) If more than one task has the same critical task then select the one with highest impotence.

4. Related Work:

Those Computers or Systems having the capability to complete the execution of a task before a deadline are called real-time systems. Real time systems not only convey correct results but when i.e. in time these results are delivered. In other words real-time systems are defined as, "those systems in which the correctness of the system depends not only on the logical result of calculation, but also at the time at which the outcome are created". If the timing constraints of the system are not met, system malfunction is said to have occurred. Hence, it is necessary & indispensable that the timing constraints of the systems are assured to be met. To guarantee timing behavior requires that the system be *predictable*. Predictability means that when a task is initiated or putted in ready state it must be possible to conclude its completion time with assurance aspect. A real-time system may be any information processing system which has to respond to externally generated input stimuli within a limited and particular period. To define in another way it is the ability of the system to promise a reply after a fixed time has beyond. Computing real-time systems are increasingly used to manage tasks in various application fields like jet, automotive and manufacturing process etc. These systems have to deal with a foremost limitation, the computation outcomes must be provided inside a delay which allows the system to keep the process under its direct control. Real-time systems are in general battery operated and battery is mandatory to be replenished on a regular basis, to keep the system operational. Since real-time systems commonly remain under-utilized and it is recommended to take up the system speed, subject to the workload so that the battery life is unlimited.

Real-time systems are more often than not, battery operated and battery is compulsory to be replenished repeatedly to keep the system ready. Since real-time system normally remain under-utilized and it is recommended to assume the system speed, subject to the workload so that the battery life is comprehensive. The two major techniques of minimizing the processor power expenditure are:

- Shutdown and
- Slowdown

Though promising for broad-spectrum purpose system, shutdown techniques are not suitable for real-time systems, to shutting down and then reactivating the shut downed device may result in deadline let pass. Slowdown through DVS, DVFS is known to be efficient for power minimization. Most suitable algorithms have been suggested for fixed priority scheduling over a fixed number of voltage levels. When tasks complete former than the own deadline, there is a chance for additional (dynamic) slowdown that increases the power savings. DVS referred as a power saving technique, which is achieved by reducing power dissipation of the core by lowering the supply voltage and operating frequency. DVS is a best standard for administration of the power utilization of a system. It is based on the fact that the dynamic (switching) power *P* of CMOS circuits is strongly dependent on the core voltage *V* and the clock frequency *f* according to equation

$$p = v^2 f$$

Since

$$E = p \times t$$

It is shown that the execution time is inversely proportional to the frequency and thus, the total energy E for the computation is directly proportional to the square of the voltage:

$$E = v^2$$

Prior works on energy aware scheduling have mainly focused attention on preemptive scheduling in real time systems, however there is a shortcoming of using preemptive scheduling, which is the number of context switching is higher as compared to non-preemptive scheduling. Context switch is large both in terms of time as well as in terms of energy. Consequently, this number becomes irrationally very much large, when DVS technique is applied, as the task preemption further increases. It is shown in [3] that a context switch takes a time 25 to 150 micro-seconds and energy (up to 4 µJ) overhead. This impact is due to changing of the voltage and allowing it to become steady. In Particular, marketable processors have different frequency overheads such as Intel's Strong-Arm takes up to 150 microseconds [6] while XScale takes around 30 micro seconds [4].

A complete voltage transition can be performed in less than 300 micro-seconds in Transmeta Cursoe 5300 processor [5].

To keep in mind the above negative characteristic associated with using DVS in preemptive scheduling, there is a need to stay away from preventable context switching and still implements to investigate DVS into non-preemptive scheduling. It is predictable that the planned amalgamation will more reduce the power consumption of the real-time system without making any compromise on the timing constraints of the real time system.

The [31] proposed a new technique that eliminates the drawback of both scheduling algorithm (RM and DM) by introducing a priority component (PC), which is defined by the user, based on task importance. [31] contains three components i.e. task period, task deadline and priority component (PC). It allocated a weightage percentage to the entire components (task period, task deadline), further priority of task is obtained by adding the weightage percentage of task period and task deadline. Then scheduling component (SC) is computed by adding these three components. All tasks are rearranged according to it's SC by increasing order. The one with highest SC is considered as most important task and is executed first.

In SC-WITH-DVFS [31] the authors use CPU burst for the task utilization that means a task completes its execution when its CPU burst is executed. They have used tl_plane for load balancing on processors. In [31] they find tl_plane for each processor and each processor is utilized to it's tl_plane. tl_plane is a restriction for the processors that processor must not be utilized after it's tl_plane. tl_plane is calculated by adding the CPU burst of all tasks and divides it by number of processors.
So if C is CPU burst of each task and M is the number of processor then tl_plane is calculated by

$$\text{tl\_plane} = \sum_{i=0}^{N} (C / M)$$

For the achievement of load balancing the task splitting technique is also used with tl_plane. The method for the task splitting is that when a CPU burst Ci of a task i is greater than the remaining tl_plane of a processor then it must be divided. The method according to which Ci must be divided is that if the CPU burst Ci of a task i is equal to the remaining tl_plane then it is executed on current processor and the remaining CPU burst of task is migrated to the minimum utilized processor i.e. a processor which have minimum cycles of a task, executed.
In [30] the authors have proposed the concept to dynamically scaling the frequency of each processor according to the current active tasks in the ready queue. In this algorithm there is no concept for the important tasks and unimportant tasks leading to starvation problems.

5. Comparative Study

Multiprocessor environment is used for processor intensive real-time applications, where tasks are assigned to processor subject to some pre-defined criteria such as CPU load etc. Conventionally, real-time systems are paying attention on periodic task models, in which tasks are released at habitual time periods. On the other hand with maturity of multiprocessor structural design, today most real-time systems function in dynamic environment where human activities (aperiodic tasks) are predictable. Aperiodic tasks are to be completed as soon as possible; consequently the priority assigned to such aperiodic tasks ought to be higher than those of periodic tasks. In distinction to its counterpart i.e. uniprocessor systems, the field of scheduling miscellaneous tasks i.e. periodic and aperiodic tasks on multiprocessor systems, still remains unexplored. Similarly, higher power consumption issues arise as a challenge associated with such systems [27, 28, 29, 30].

RM and DM [46, 47] are important scheduling algorithms for real-time systems. RM and DM both are fixed priority algorithms and give equal importance to each task, which yields a major drawback of RM and DM. In case of RM, RM assign a higher priority to the task having short period due to which unimportance task having short period is scheduled first from the importance tasks having longer periods [23]. The same criteria is also implies to deadline monotonic in which the scheduling criteria based on its deadline, where a task having short deadline is schedule first from those important tasks having longer deadline. In [1] the authors have proposed a new algorithm that can schedule the most important tasks first. In [2] the authors have proposed the concept to dynamically scaling the frequency of each processor according to the current active tasks in the ready queue. In this algorithm there is no concept for the important tasks and unimportant tasks leading to starvation problems. In [34] if the TL plane is a real number, then all processors are equally utilized. If TL plane is FF number, then at least one processor is less utilized, in which case there a surety that at least one task has been splitted. If number of tasks is increased than number of processors, then processors are more utilized.

In [1] the authors have proposed a new load balancing algorithm for scheduling real-time tasks in a multi-core processor technology. They have also introduced task splitting concept to balance the load on each core to minimize its energy requirements. They have achieved workload partitioning with 100 percent precision. This algorithm is capable to distribute workload amongst all processing cores equally and keep the utilization of all processing cores on average. This means that no processing core is extra loaded or extra burdened. The existing problem in proposed technique is that all processing cores are considered to be homogeneous. Their argument that if the workload is distributed evenly, all cores will complete their work at the same time, is true but still a gap is there about heterogeneous technology. The authors have discussed cycle-conserving technique which can update the utilization of core dynamically on release and completion of a task and hence results in power management.

Criteria	Rate monotonic (RM)	Deadline monotonic (DM)	Next-fit Algorithm	Utilization balancing algorithm	Power efficient rate monotonic scheduling for multiprocessor system	new scheduling algorithm for real time system	energy-efficient scheduling algorithm for sporadic real-time tasks in multiprocessor systems	Real-time scheduling with task splitting on multiprocessors (MP):	SC-WITH-DVFS
Task importance	Same	Same	Same	Same	Same	Priority component is added	Same	Same	Priority component is added
Processor	Uniprocessor	MP	MP	MP	MP	Uniprocessor	MP	MP	MP
Scheduling criteria	P*i*	D*i*	P*i*	Util	P*i*	P*i* D*i* Task priority	P*i*	P*i*	P*i* D*i* Task priority TL-plane
Task scheduling	Tasks are arranged in ascending order based on P*i*	Tasks are arranged in ascending order based on D*i*	Tasks are arranged in ascending order based on P*i*	Tasks are arranged in ascending order based on Util	Tasks are arranged in ascending order based on P*i*	Tasks are arranged in ascending order based on scheduling component	Tasks are arranged in ascending order based on P*i*	Tasks are arranged in ascending order based on P*i*	Tasks are arranged in ascending order based on scheduling component
Time difference	P*i* does not change with time	D*i* does not change with time	P*i* does not change with time	Task Util does not change with time	P*i* does not change with time	P*i* , D*i*, task priority does not change with time	P*i* does not change with time	P*i* does not change with time	P*i*, D*i*, task priority does not change with time
Problem	Starvation	Starvation	Starvation, Load balancing	No task splitting Perfect balancing of utilization across different processor is difficult.	Starvation There is no task splitting. And all processor are not equally utilized	Starvation Is reduce for only uniprocessor	Starvation	Starvation, Load balancing	In some situations at most one processor is less utilized
Energy saving by	N/A	N/A	N/A	N/A	N/A	N/A	DVS	N/A	Load balancing
Task Splitting	No	No	Yes	No	No	No	No	Yes	Yes
Load Balancing	Yes	Yes	No	No	No	N/A	Yes	No	Yes

TABLE COMPARISON: EXSISTING ALGORITHMS

6. Conclusion:

Power aware scheduling is the culling edge technique for reducing power constraints of multiprocessor systems. These systems generally remain under-utilized thus becomes an ideal candidate for power aware scheduling. In recent times it is realized there is a need for energy reduction in processors, a lot of work has been done on minimizing the energy reduction. As a drawback of reducing energy consumption of the system, its response time is increased, hence degrades the overall performance of the systems [36].

In multi-processor environments [37] i.e. HPC (including clusters, grids and clouds) the main issue is heating and energy conservation. Our goal is to study different techniques that helps in minimizing the energy consumption so that the cooling cost will be reduced. Scheduling periodic and aperiodic tasks such that the load is balanced amongst different processors and the energy consumption is reduced, is a major concern and an active research topic. Runtime power reduction mechanisms can also reduce the energy expenditure to some extent. Some approaches have arisen frustrating to diminish energy spending at HPC but still HPC providers are in want of mechanisms and techniques not only for sinking energy eating but also for accomplishing with the required QoS to guarantee the customer happiness. There still stay alive some gaps that must be sheltered to attain the energy performance stability that is essential in HPC and scientific computing environment [38].

References

[1] Zakarya, M., Dilawar, N., Khattak, M. A., & Hayat, M. (2013). Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core. *World Applied Sciences Journal*, *22*(10), 1431-1439.

[2] Zakarya, M., & Khan, A. A. (2012). Cloud QoS, High Availability & Service Security Issues with Solutions. IJCSNS, 12(7), 71

[3] Zakarya, Muhammad, Izaz Ur Rahman, and Imtiaz Ullah. "An Overview of File Server Group in Distributed Systems." Ijtech.org

[4] Zakarya, Muhammad, Ayaz Ali Khan, and Hameed Hussain. "Grid High Availability and Service Security Issues with Solutions." (2010): 978-1.

[5] Y. Shin and K. Choi,"power conscious fixed priority scheduling for hard real-time systems", in design automation conference, 1999, pp.134-139.

[6] KRISHNA, C. M., AND LEE, Y.-H. Voltage-clock-scaling techniques for low power in hard real-time systems. In Proceedings of the IEEE Real-Time Technology and

[7] Applications Symposium (Washington, D.C., May 2000), pp. 156–165.

[8] PERING, T., AND BRODERSEN, R. The simulation and evaluation of dynamic voltage scaling algorithms. In Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'98 (Monterey, CA, Aug. 1998), pp. 76–81.

[9] POUWELSE, J., LANGENDOEN, K., AND SIPS, H. Energy priority scheduling for variable voltage processors. In Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01 (Huntington Beach, CA, Aug. 2001).

[10] Alexandros Zerzelidis, A Framework for Flexible Scheduling in Real-Time Middleware, PhD Thesis, The University of York, Department of Computer Science, November 2007.

[11] www.embedded.com

[12] Hui Chen, Jiali Xia, A Real-Time Task Scheduling Algorithm Based on Dynamic Priority, In Proceedings of International Conferences on Embedded Software and Systems, 2009.

[13] A. Burns, D. Prasad, A. Bondavalli, F. Giandomenico, .Ramamritham, J. Stankovic, L. Strigini, The Meaning and Role of Value in Scheduling Flexible Real-Time Systems. Journal of Systems and Architecture, 46, pp. 305~325, 2000.

[14] http://www.wisegeek.com/what-is-real-time.htm

[15] Gaurar Arora , Automated Analysis and Prediction of Timing Parameters in Embedded Real-Time Systems using Measured Data, MS Thesis, University of Maryland, June 1997.

[16] Sang H-Son, Lecture on Real-Time Database Systems and Data Services: Issues and Challenges, Department of Computer Science, University of Virginia Charlottesville.

[17] www.netrino.com/Embedded-Systems/How-To/RMA-Rate-Monotonic- lgorithm

[18] Min-Allah N, Young-Ji W, Jiang-Sheng X, Liu J, Revisiting Fixed Priority Techniques, In Proc of Embedded and Ubiquitous Computing, LNCS 4808, pp. 134~145, 2007.

[19] Enrico Bini, Giorgio C. Buttazzo, Giuseppe Lipari, Minimizing Energy in Real-Time Systems with Discrete Speed Management, ACM Transactions on Embedded Computing Systems 8(4), July 2009.

[20] J. L.W.V. Jensen, Sur les Functions Convexes et les Inegalites entreles Valeurs Moyennes, Acta Math, Vol, 30, pp. 175~193, 1906.

[21] T. Ishihara and H. Yasuura, Voltage Scheduling Problem for Dynamically Variable Voltage Processors, In International Symposium on Low Power Electronics and Design, pp. 197~202, 1998.

[22] K. .leffay, D.F. Stanat, C.U. Martel, On non-preemptive scheduling of periodic and sporadic tasks, in: Proc. of the Real-Time Systems Symposium, 1991, pp. 129-139.

[23] X. C. Yuanv M. C. Saksena, A. K. Agrawala, A decomposition approach to non-preemptive real-tune scheduling, Real-Time Syst., 6(1), 1994, pp. 7-35.

[24] V. Swaminathan,K. Chakarbarty, Pruning-based energy optimal deterministic I/O device scheduling for hard real time systems,ACM Trans embedded comput.syst.4(1),2005,pp,141-167

[25] N. Guan, D Qingxu, G. zonghua, X Wenyao, Y. Ge, schedulability analysis of non-preemptive and non-preemptive EDF on partial runtime reconfigurable FPGAs ACM trans des autom electron systs.,13(4),2008,pp1-43.

[26] K.sangwon, L joonwon, k. jinsoo, runtime feasibility check for non-preemptive real time periodic tasks.inf.,process,Lett,97(3),2006,pp.83-87.

[27] R. jeuikar. R. gupta, energy aware non-preemptive scheduling for hard real time systems in proc. of 17th of euromicro conference on real –time systems,2005,pp.21-30.

[28] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery, 20(1):46–61, January 1973.

[29] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In IFIP Congress, pages 807–813, 1974.

[30] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Trans. on Software Engineering, 15(12):1497–1506, December 1989.

[31] M. Zakarya, Uzma, AA Khan, Power Aware Scheduling Algorithm for Real-Time Tasks over Multi-Processors, Middle-East Journal of Scientific Research 15(1): 94-105, 2013

[32] Y. .C and R. Ramesh, A new scheduling algorithm for real time system, (international journal of computer and electrical engineering, vol.2, no.6, December, 2010 1793-8163).

[33] V. Salmani, S. Taghavi Zargar, and M. Naghibzadeh, A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks, (Word Academy of Science, Engineering and Technology 2005).

[34] M. Zakarya, I. Rahman and A. Ali Khan, Energy Crisis, Global Warming & IT Industry: Can the IT Professionals make it better Some Day? A Review:"(©2012 IEEE)

[35] N. Min-Allah1, A. Raza Kazmi, I. Ali, X. Jian-Sheng, W. Yong- Minimizing Response Time Implication in DVS Scheduling for Low Power Embedded Systems,

[36] J. Chen, C. -Fu Kuo, Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms *(*Council of Agriculture, Executive Yuan, Taiwan, since Dec. 2006.*)*

[37] E. Seo, J. Jeong, S. Park, and J. Lee, Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors, IEEE TRANSACTIONS ON PARALLEL

AND DISTRIBUTED SYSTEMS, VOL. 19, NO. 11, NOVEMBER 2008

[38] Khan, A. A., & Zakarya, M. PERFORMANCE SENSITIVE POWER AWARE MULTIPROCESSOR SCHEDULING IN REAL-TIME SYSTEMS, Technical Journal UET Taxila (Pakistan) 2010.