PERFORMANCE SENSITIVE POWER AWARE MULTIPROCESSOR SCHEDULING IN REAL-TIME SYSTEMS

Ayaz Ali Khan, Muhammad Zakarya
COMSATS Institute of Information Technology, Islamabad, Pakistan
ayazak12345@gmail.com

Abdul Wali Khan University (AWKU), Mardan, Pakistan
mohd.zakarya@awkum.edu.pk

ABSTRACT

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. Conventionally, real-time systems are paying attention on periodic task models, in which tasks are released at regular 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 counter part the field of scheduling hybrid tasks i.e. periodic and aperiodic tasks on multiprocessor systems, still remains relatively unexplored. Similarly, higher power consumption issues arise as a challenge associated with such systems. Power aware scheduling is the cutting edge technique for reducing power constraints of multiprocessor systems. These systems generally remain under utilized thus becomes an ideal candidate for power aware scheduling. Recently a lot of work has been done on minimizing the energy requirement of processors. As a drawback of reducing power consumption of such systems, its response time is increased unreasonably, hence degrades the overall performance of the systems. In the prior work, higher importance is given to energy reduction while minimizing the response time of hybrid tasks is unnoticed. In this work we consider the importance of response time while reducing the power expenditure and utilization of the system. We propose a solution that reduces the power utilization and use of a multiprocessor system while the response time of tasks is kept with in a bound time. We assign aperiodic tasks to the processor that is under utilized for two reasons.

1. The underutilized processor has enough room to complete the aperiodic task with smallest possible time window and
2. There is a potential for running the system with low possible speed in addition to meeting deadline constraints.

The above solutions are obtained through mathematical foundation and experimental result supports our theoretical framework.

Keywords: (PEDS) Priority Exchange Dynamic Server, (RTS) Real Time System, (RM) Rate Monotonic, (DFS) Dynamic Frequency Scaling, (DM) Deadline Monotonic, (EDLS) Earliest Deadline Late Server, (EDF) Earliest Deadline First, (PDA) Personal Digital Assistant, (DVS) Dynamic Voltage Scaling, (WCET) Worst Case Execution Time

I. INTRODUCTION & CONCEPTS:

Multiprocessor can be defined as a computer system having more than one processor, each one sharing system main memory & peripherals, to concurrently process programs. Hence the scheduling algorithms optimal on uniprocessor machines are not subject to be optimal on multiprocessor machines. So for multiprocessors are concerned, we use different scheduling algorithms. According to the law of Gordon Moore which states that the number of transistors in microprocessors would double every year [1]. But the advances in microprocessors are not face with battery power, and the battery capacity is tripled since 1990's [2, 3]. Therefore there is a need for applying

energy efficient scheduling techniques that has become a major design consideration in realtime computing environment. There are many applications where aperiodic tasks are executed, as an outcome of exterior events. The time in which these proceedings take place are not controlled in reality by an application designer. For example, when keyboard buttons are pressed.

When there are periodic and aperiodic tasks in the given system, our goal is to reduce the response time of aperiodic tasks in such a manner that all the periodic tasks are still optimal & schedulable. The most cut down technique is to give out available time slots, that are left unused by the periodic tasks, to aperiodic tasks. The background scheduling is used to schedule aperiodic tasks which is quite simple; however using background scheduling the desired response is never obtained. To reduce the response time of aperiodic tasks, we use the concept of aperiodic server. The aperiodic server or total bandwidth server services aperiodic requests as soon as possible [4]. The aperiodic server is consisting of a period and fixed execution time called server capacity and is scheduled with the same algorithm that is used for the periodic tasks.

There are two types of scheduling techniques for multiprocessors including

- Partitioned scheduling and
- Global scheduling.

In partitioned scheduling, each task is assigned to a specific processor and then it is executed on that processor without migration. These processors are then scheduled independently and separately. This reduces the multiprocessor scheduling into a set of uniprocessor scheduling. With this scheduling we can use an optimal uniprocessor scheduling algorithm for multiprocessor systems. In partitioned scheduling the run time overhead is low as compared to global scheduling, because in partitioning technique the task does not migrate to other processors. But there is a shortcoming in schedulability bounds as the deadline to be missed if the total processor utilization exceeds ($\beta m + 1$) ($\beta+1$), where $\beta = 1/\alpha$ and α is a maximum utilization of individual tasks [5]. Let $\alpha = 1$ and $m \rightarrow \infty$ then the processor utilization is bound by 50%. We use partitioning method in our work because the overhead is low so the context switching and energy consumption would be reduced. The alternative is the global scheduling, in which all the tasks are stored in a single priority queue. The scheduler selects the task having the high priority for execution. In global scheduling the tasks are not fasten to a particular processor and it can be executed on any processor. The optimal uniprocessor scheduling algorithm (EDF, RM etc) gives low utilization on multiprocessors. The global scheduling is best in the worst case schedulability. All the task sets are schedulable, if the processor utilization is less than or equal to 100%. But the number of context switching and migration is a problem. The worst case processor utilization is less than 50%.

In multiprocessors the main issue is heating and energy. Our goal is to minimize the energy consumption so that the cooling cost will be reduced. We are scheduling periodic and aperiodic tasks such that the load is balanced among different processors and the energy consumption is reduced. Runtime power reduction mechanisms can reduce the energy expenditure. For energy reduction we can use the DVS in latest processors. It means that power is a linear function of frequency i.e. f and a quadratic function of the voltage i.e. V given by ($p \propto fV^2$). The voltage adjustment at an instant of time is called DVS, which is an effective way for power saving in current systems [2].

In addition to saving energy, another advantage of having reduced power consumption is lower cooling cost of the multiprocessing environment (web farms, clusters etc).

In recent processors the relationship between frequency f and power p gives foundation to Dynamic Voltage Scaling

$$E = Pt \tag{1}$$

Where E is energy consumed, t is time taken and P is power consumed. The average power dissipation in processor is:

$$P_{avg} = P_{capt} + P_l + P_{stdby} + P_{sc} \quad (2)$$

Where P_{capt}, P_l, P_{stdby}, and P_{sc} is capacitance, leakage, standby and short circuit power. The P_l, P_{stdby} and P_{sc} are important but they are least important as compared to P_{capt}. So we will not consider P_l, P_{stdby}, and P_{sc}. So the P_{capt} is equal to:

$$P_{capt} = \sim CV_d^2 f \quad (3)$$

Where $\sim$ is the transition activity dependant parameter, C, V_d and f is switched capacitance, supply voltage, and clock frequency. Equation (3) shows that the supply voltage V_d is quadratic as compared to clock frequency f; furthermore it also shows that lowering the supply voltage would be the most efficient way to reduce the power consumption. But when V_d is reduced then the circuit delay t_d would be increased:

$$t_d = \frac{mV_d}{(V_d - V_{tv})^2} \quad (4)$$

Where t_{delay} is threshold voltage and m is a constant which will depend on gate size and capacitance. As from equation (4) the f and t_d are inversely proportional, so it would mean that the energy expenditure would be reduced in CMOS devices at the expense of performance delay. The frequency f is:

$$f = \frac{(V_d - V_{tv})^2}{kV_d} \quad (5)$$

Equation (5) shows that the clock frequency is directly proportional to supply voltage. If we would consider $P = P_{capt}$, then equation (3) can be written as:

$$P = \sim CV^2 f \quad (6)$$

Equation (6) shows that when the clock speed f and voltage is changed then it would effect power consumption linearly and quadratically, respectively.

We use partition-based system where the workload is partitioned among processors. We derive results for a single processor and then extended it to multiprocessors. We would schedule mixed tasks (periodic and aperiodic tasks) with EDF scheduling algorithm, and those tasks, which have the earliest deadline, would have the highest priority. Our processor would have the discrete speed and voltage levels where $v_1 < v_2 < ..... < v_n$ and $f_1 < f_2 < ..... f_{\max}$. We would consider the overhead of scheduling algorithm and voltage transition insignificant. The power spending by a processor with the speed f is given by $g(f)$ and the energy consumption during the interval $[t_1, t_2]$ is $\int_{t_1}^{t_2} g(f(t))dt$ [3].

II. RELATED WORK & EXISTING TECHNIQUES:

In this section we discuss some existing mechanisms and techniques.

A. Real time systems are those systems that would give us results in a given time period. When a computer, that controls a device, sensor would give the data at regular time period and the computer responds by sending signals to an actuator,

there must be a time bound in which the computer must respond to it. The ability of the system to respond with in a given time period depends on its capacity. If the system is unable to fulfill the demands, then we say that the system has inadequate resources. The system with limitless resources can fulfill the demands within the given time period. When the system is unable to respond within a given time period, it has different consequences i.e. there may be no upshot at all, or it may be minor or it may be catastrophic. The real time system is very simple like microcontroller or it may be complex like flight control system. Real time system examples are process control system, flight control system, intelligent highway systems, robotics, and high speed media communication system [6, 7].

B. Scheduling may be defined as the act of assigning resources to tasks. The scheduling can be applied to the tasks by defining the start time, so that no more than one task can request for a resource at that specific time, this is called static scheduling [xP90, xu93, foh94]. Other type of scheduling would be to assign priorities to tasks and then execute the task with the highest priority when there are more requests than the number of resources then the scheduling decision would be made. The priority scheduling can meet all deadlines where the time table schedule cannot. This situation occurs when the tasks that wants to execute is not known is advance or design, and the tasks execute when event occur.

C. Global Scheduling Algorithms, Such algorithms store the tasks in one queue that is mutual and common amongst all processing units. All tasks in the queue are maintained and are allocated with their specific priority. Suppose there are more than one task, then the task with the highest priority is selected from the queue and is executed on the specific processor using preemption and migration technique [21, 28]. Each processing unit preserves a status table that designates which tasks have previously committed to run. Additionally, each processing unit has a table that indicates how many processors have spare computational power. The time axis is split into slots, and these slots are of some fixed duration, and each processing unit on a regular basis sends information to its counterparts about the next slot that is free.

D. When the processor is overloaded it checks its surplus information and selects the processor that is most appropriate to complete the task with in its time period. However, it is possible that the surplus information is out of date and the selected processor can not execute the task. This problem can be solved by sending the task to the selected processor at the same time the originating processor ask from lightly loaded processor that how quickly they can execute the task. Then these responds are sent to the selected processing unit. When the selected processing unit is unable to execute the task with in its time period, then it can review the responds that which other processing unit is able to execute the task within its time period and then transfer the task to that processor.

E. Partitioning Scheduling Algorithms, Such algorithms divide the tasks in such a way that many task sets are created and then each task set is executed and schedulable on a particular processor. The tasks cannot migrate from one processor to the other, so the multiprocessor scheduling dilemma is altered into many uniprocessor scheduling dilemmas [21, 28]. In partitioning method several task sets are created and each task set is contained in different queues associated with each processor. As the multiprocessor scheduling problem is changed into many uniprocessor

scheduling problem so we can use an optimal uniprocessor scheduling algorithm to schedule the tasks. Global scheduling strategies have many shortcomings over partitioning scheduling strategies. For example, Partitioning typically has a little scheduling overhead as put side by side to global scheduling, as tasks do not require to migrate across processing units. In addition, partitioning scheduling strategies divide a multiprocessing unit scheduling problem to a set of uniprocessor scheduling problem and then some optimal uniprocessor scheduling algorithms can be used. On the other hand, partitioning scheduling strategy has two disadvantages over global scheduling. First, to find the optimal handing over of jobs to the processing units is a bin-packing problem that is an NP-complete problem. Therefore, jobs are frequently partitioned using non-optimal heuristics. Secondly [13], there exists tasks that are schedulable if they exist non-partitioned. In spite all this partitioning strategies are extensively used. Additionally the hybrid of partitioning / global scheduling algorithm can be used. For example, at any instant of time the task is allocated to a single processing unit and is allowed to migrate as well.

III. EXISTING PROBLEM:

In recent times it is realized that there is a need for energy reduction in processors, a lot of work has been done on minimizing the system energy consumption. As a drawback of reducing energy consumption of the system, its response time is increased which degrades the overall performance of the systems. In prior work, higher importance is given to energy reduction and reducing response time of aperiodic tasks remains unnoticed.
We consider the importance of response time while reducing the power consumption of the multiprocessor system. With mathematical foundation we proposed a solution that reduces the power consumption of a multiprocessor system while the response time of tasks is kept within bound.

IV. PROPOSED SOLUTION:

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. Traditionally, the focus of real-time systems are periodic task model where the release time of tasks are known, however with the advancement of multiprocessor design, the real-time systems are also using aperiodic tasks where the release time are not known in advance. The aperiodic tasks should be completed as quickly as possible; therefore the priority of aperiodic tasks must be greater than periodic tasks. In contrast to its counter part i.e. uniprocessor systems, the field of scheduling mixed tasks (periodic and aperiodic) on multiprocessor system still remains unexplored. Similarly, higher power consumption issues arise as a challenge associated with such systems. 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. In prior work, higher importance is given to energy reduction and reducing response time of hybrid tasks is unnoticed.

We consider the importance of response time also while the energy reduction is achieved. In our work we propose a solution that reduces the power consumption of a multiprocessor system while the response time of tasks is kept within bound.
Mixed workload scheduling (Periodic and aperiodic Tasks):

A. PERIODIC TASKS

Periodic tasks are those types of tasks that would appear after a fixed interval of time. A Periodic task set $T = \{T_1, T_2, ..., T_n\}$ that arrive at time t = 0, where every task T_i has Two parameters (p_i, c_i), where p_i is the time period and c_i is WCET of the task.

- All tasks are independent and preemptable.
- The task T_i has relative deadline D_i is equal to p_i.
- The released instance $R_{i,j}$ of task T_i is called the $j-th$ job of task T_i. The $R_{i,j}$ release Time would be $p_i \times (j-1)$.
- The task T_i WCET would be known in advance.

B. APERIODIC TASKS

Aperiodic tasks are those tasks that appear at any time and we doesn't know that when these tasks would reappear. The Aperiodic jobs $\{\pm_m / m = 1, 2..... \}$ have two parameters (r, e), where r is the release time of job and not known in advance, e is the WCET of $\pm_m$. We considering 'r' as the arrival time and 'e' as the execution time, the aperiodic load would be $\mu = e/r$. In our work we would change the load up to the maximum level. The aperiodic tasks would be run on a special type of server called Total Bandwidth Server. The TBS [4] has the capacity $u_s = c_s / p_s$ where c_s is the execution budget and p_s is period of the server. The $m-th$ aperiodic job $\pm_m$, having the execution time e_m and arrival time r_m, is given the deadline:

$$d_m = \max(r_m, d_{m-1}) + \frac{e_m}{u_s} \qquad (7)$$

Where e_m is WCET. So Equation (7) shows that when we have higher u_s then d_m would be earlier.

SCHEDULING MIXED WORKLOAD WITH M PROCESSORS:

According to EDF, a task set is schedulable iff

$$u_{tot} = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1 \qquad (8)$$

Where, u_{tot} is total system utilization, c_i is execution time and p_i is the time period. With periodic tasks we have also aperiodic tasks so we must also consider it with periodic tasks, such that the utilization of periodic and aperiodic tasks must be less than or equal to one.

$$u_p + u_s \leq 1 \qquad (9)$$

In Equation (3) we have $0 \le u_p, u_s < 1$ and $0 < u_p + u_s \le 1$, at frequency $f = f_m$ ($f = 1$ in this case), as f =1 so we say that this approach is not a DVS approach because system is running at full speed.

Equation 9 shows that the system is running at full speed and gives us the lowest system utilization. So all the tasks running at their WCET, so the system utilization is far less than 1. It means that the processor is doing nothing for most of the time. It also means that the energy is wasted during the idle time intervals, as the processor is running at their full speed. The energy consumption can be reduced by lowering the speed of the processor, but lowering the speed would take the task longer to complete and the response time of the tasks would be increased. According to [39], the frequency component is added to Equation (9) as:

$$u_p + u_s \le \frac{f_i}{f_m} \qquad (10)$$

Where f_m is maximum speed of the processor and f_i is the suitable speed so that the task set is feasible schedulable. We represent the initial speed of the processor f_i by f_{static} , and we denote f_i / f_m by $\sim_b$.

RESPONSE TIME CONSTRAINT:

As Equation (10) gives us the lowest frequency, so that the mixed tasks are feasibly schedulable but the execution time are scaled by a factor of $1/\sim_b$. When we decrease the frequency then the voltage consumption will be reduced, and according to Equation $p_{cmos} = v^2 f$, when we reduce the speed then the power consumption would be reduced, but the response time of the task would be increased and the system performance would be degrade. When we run an application then the energy requirement at time t would be $E = p.t$. So energy consumption would be $E \propto v^2$.

The tasks are running at lower frequency $\sim_b$, so the execution times of the tasks are increased. The deadline of TBS [40] would become:

$$d_m(\sim_b) = \max(r_m, d_{m-1}) + \frac{e_m}{u_s . \sim_b} \qquad (11)$$

The deadline for TBS is delayed as

$$d_m(\sim_b) - d_m = \max(r_m, d_{m-1}) + \frac{e_m}{u_s . \sim_b} - \max(r_m, d_{m-1}) + \frac{e_m}{u_s} = \frac{e_m}{u_s}(\frac{1}{\sim_b} - 1) \qquad (12)$$

It is clear from Equation (6) that the deadline is increased, so the response time of aperiodic tasks would also be increased. The aperiodic tasks are very less in real time applications as compared to periodic tasks such as java based videophone, which runs the garbage collector (aperiodic task) almost every 600ms for every 3.732ms [40]. In those systems where aperiodic tasks came less frequently as compared to periodic tasks, and those aperiodic tasks need quick response then one solution is Equation (6) in which we run the task at full speed. But there is a disadvantage as the curve of energy and voltage is convex in nature [41]. When we increase the voltage then the power consumption would be increased quadratically. Equation (6) shows that we decrease the scheduling priority and the response time of aperiodic tasks is increased. We consider that when the response time of aperiodic task is smaller than p_s (worst case), then there is no need of frequency scaling. To avoid the performance degradation of aperiodic tasks, we restrict this deadline delay. As it is showed earlier that the TBS has to execute

aperiodic jobs for c_i intervals during any interval of length p_s In our work, when we apply DVS then this delay must be less than or equal to p_s i.e. $d_m(\sim_b) - d_m \le p_s$. As we have a range of speed levels ($f_1 < f_2 < .....f_m$), authors in [42] gives a technique to find suitable frequency f_k for aperiodic job $\pm_m$.

$$\sim_b = (1 + \frac{p_s.u_s}{e_m})^{-1} \qquad (13)$$

In case ($d_m(\sim_b) - d_m \le p_s$), our algorithm completes aperiodic load before p_s. Equation (13) is for single processor, however, for multiple processors we need to find the $\sim_b$ on all m processors and allocate the aperiodic task $\pm_m$ to the processor having lower $\sim_b$. We would use the partitioning scheduling method for the periodic tasks and global scheduling method for aperiodic tasks.

The above formulation is valid for a single processor, while the intended purpose of this work is to accommodate aperiodic tasks on multiprocessor system and make sure aperiodic jobs completed as soon as possible, and no periodic task sever miss the deadline. We are using global scheduling mechanism for the aperiodic jobs.

Initially, periodic tasks are assigned to all available processors i.e., periodic load is uniformly distributed among processors. Also, every processor has a TBS for aperiodic tasks and it's very likely that TBS has different capacity on all processors and the larger capacity it has, the better it would be, because aperiodic job will complete much early as compare to low capacity TBS.

With our approach, let processor 1 has TBS and the required speed for completing aperiodic job is obtained with $\sim_{1b}$, where subscript 1 points to processor 1.

$$\sim_{1b} = (1 + \frac{p_s.u_s}{e_m})^{-1}$$

The required speed for processor 2 is obtained with:

$$\sim_{2b} = (1 + \frac{p_s.u_s}{e_m})^{-1}$$

Similarly for m-th processor it would be:

$$\sim_{mb} = (1 + \frac{p_s.u_s}{e_m})^{-1}$$

We get speed for all the processors. Once this step is done, we encounter the solutions:

RUN APERIODIC TASK ON SLOWEST POSSIBLE SPEED:

We determine the lowest possible speed for aperiodic task on all the processors $\sim_l = \min(\sim_{1b}, \sim_{2b}, ...\sim_{mb})$ so that it gets completed with the bounded time where $\sim_l$ is the lowest speed of all processors. This solution result in reducing energy consumption of the over all system, as it runs on lowest speed. However, the response time of aperiodic task gets large due to the lowest system speed

RUN APERIODIC TASK WITH HIGHEST POSSIBLE SPEED:

This solution gives the maximum available speed for aperiodic task on all the processors. In other words: $\sim_h = \max(\sim_{1b}, \sim_{2b}, ...\sim_{mb})$. We find the speed of

aperiodic task on all the processors $1,2,3,\ldots\ldots m$ and then the processor with the highest possible speed $\sim_h$ is selected. This means that the TBS on that particular processor has largest capacity and can only respect the time constraint when executed on the $\sim_h$ (highest speed) i.e. the processor I needs $\sim_h$ (highest) speed to completed aperioid task within period. As discussed earlier, running a processor at maximum speed mean consuming maximum system power, which can not be compromised unnecessarily. In this work, since, we are maintaining a queue of aperiodic tasks, a particular aperiodic task will be consider for execution at individual processors and the one which can execute the task with highest speed is assigned the task. We opt for the first option (Run aperiodic task on slowest possible speed) because it will result in lower energy consumption and the aperiodic task will be completed with assigned time window, which is the main contribution of the work. The proposed technique will make sure to complete the aperiodic task within the time window and reduce the total power consumption of the multiprocessor system. In other words, based on ps of all processors, our technique will find the smallest speed $\sim_l$ such that aperiodic task gets completed within a permissible time.

V. IMPLEMENTATION, SIMULATION & RESULTS:

First we produce periodic tasks and calculate their utilization and average values. Similarly the task set generated is then divided and mapped onto multiprocessors. Utilization of all tasks are shown and aperiodic tasks are assigned to processors that are under utilized i.e lesser periodic load is assigned. As a final outcome, response time of aperiodic task, to frequency of the system and corresponding power consumption is drawn at the end.

A. SIMULATIONS STUDY

PERIODIC TASKS:

Execution Time	Time period	Utilization
20.0000	204.0000	0.0980
30.0000	214.0000	0.1402
14.0000	214.0000	0.0654
17.0000	218.0000	0.0780
27.0000	231.0000	0.1169
18.0000	257.0000	0.0700
29.0000	259.0000	0.1120
27.0000	261.0000	0.1034
25.0000	261.0000	0.0958
21.0000	288.0000	0.0729
4.0000	288.0000	0.0139
30.0000	290.0000	0.1034
5.0000	310.0000	0.0161
13.0000	323.0000	0.0402
1.0000	332.0000	0.0030
3.0000	348.0000	0.0086
27.0000	352.0000	0.0767
24.0000	365.0000	0.0658
30.0000	383.0000	0.0783
2.0000	385.0000	0.0052
14.0000	400.0000	0.0350

29.0000	406.0000	0.0714
9.0000	409.0000	0.0220
10.0000	410.0000	0.0244
22.0000	414.0000	0.0531
19.0000	420.0000	0.0452
2.0000	421.0000	0.0048
18.0000	429.0000	0.0420
9.0000	440.0000	0.0205
19.0000	448.0000	0.0424
29.0000	449.0000	0.0646
4.0000	454.0000	0.0088
27.0000	486.0000	0.0556
6.0000	486.0000	0.0123

Total Utilization = 1.8661
Average = 0.6220
Tasks Executed on Processor_1 =

Execution Time	Time period
20	204
30	214
14	214
17	218
27	231
18	257
29	259
27	261

Tasks Executed on Processor_2 =

Execution Time	Time period
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
25	261
21	288
4	288
30	290
5	310
13	323
1	332
3	348
27	352
24	365
30	383
2	385
14	400
29	406
9	409
10	410
22	414

19	420
2	421

Tasks Executed on Processor_3 =

Execution Time	Time period
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
18	429
9	440
19	448
29	449
4	454
27	486
6	486

Utilization of Processor_1 = 0.7840
Utilization of Processor_2= 0.8360
Utilization of Processor_3= 0.2461
The less utilized Processor is Processor_3 and it can execute Aperiodic load up to 0.7539

APERIODIC TASKS:

Execution Time	Time period	Utilization
1.0000	670.0000	0.0015
1.0000	666.0000	0.0015
1.0000	655.0000	0.0015
2.0000	982.0000	0.0020
2.0000	952.0000	0.0021
2.0000	619.0000	0.0032
3.0000	883.0000	0.0034

4.0000	749.0000	0.0053
5.0000	869.0000	0.0058
5.0000	825.0000	0.0061
4.0000	616.0000	0.0065
6.0000	802.0000	0.0075
8.0000	986.0000	0.0081
8.0000	923.0000	0.0087
5.0000	527.0000	0.0095
8.0000	786.0000	0.0102
7.0000	651.0000	0.0108
10.0000	909.0000	0.0110
9.0000	665.0000	0.0135
11.0000	783.0000	0.0140
10.0000	681.0000	0.0147
14.0000	951.0000	0.0147
13.0000	872.0000	0.0149
11.0000	716.0000	0.0154
11.0000	667.0000	0.0165
14.0000	822.0000	0.0170
12.0000	680.0000	0.0176
12.0000	675.0000	0.0178
11.0000	604.0000	0.0182
10.0000	546.0000	0.0183
19.0000	974.0000	0.0195
13.0000	616.0000	0.0211
19.0000	891.0000	0.0213
20.0000	933.0000	0.0214
17.0000	788.0000	0.0216
12.0000	549.0000	0.0219
12.0000	520.0000	0.0231
13.0000	557.0000	0.0233
24.0000	989.0000	0.0243
18.0000	722.0000	0.0249
21.0000	839.0000	0.0250
25.0000	998.0000	0.0251
13.0000	518.0000	0.0251
22.0000	830.0000	0.0265
25.0000	934.0000	0.0268
24.0000	839.0000	0.0286
20.0000	698.0000	0.0287
17.0000	591.0000	0.0288

Aperiodic load =

Columns 1 through 8

0.0015 0.0030 0.0045 0.0066 0.0087 0.0119 0.0153 0.0206

Columns 9 through 16

0.0264 0.0324 0.0389 0.0464 0.0545 0.0632 0.0727 0.0829

Columns 17 through 24

0.0936 0.1046 0.1182 0.1322 0.1469 0.1616 0.1765 0.1919

Columns 25 through 32

0.2084 0.2254 0.2430 0.2608 0.2790 0.2974 0.3169 0.3380

Columns 33 through 40

0.3593 0.3807 0.4023 0.4242 0.4472 0.4706 0.4948 0.5198

Columns 41 through 48

0.5448 0.5698 0.5949 0.6215 0.6482 0.6768 0.7055 0.7342

Response_time =

Columns 1 through 8
0.0001 0.0003 0.0009 0.0009 0.0020 0.0158 0.0195 0.0284
Columns 9 through 16
0.0327 0.0386 0.0525 0.0593 0.0593 0.0617 0.0791 0.0867
Columns 17 through 24
0.0994 0.1023 0.1144 0.1221 0.1337 0.1350 0.1393 0.1496
Columns 25 through 32
0.1617 0.1679 0.1795 0.1913 0.2058 0.2224 0.2229 0.2369
Columns 33 through 40
0.2405 0.2425 0.2500 0.2666 0.2843 0.3005 0.3005 0.3106
Columns 41 through 48
0.3163 0.3163 0.3345 0.3409 0.3433 0.3493 0.3607 0.3761

Response_time_frequency =

Columns 1 through 8
0.0005 0.0013 0.0027 0.0029 0.0044 0.0188 0.0233 0.0331
Columns 9 through 16
0.0385 0.0458 0.0612 0.0698 0.0715 0.0759 0.0954 0.1053
Columns 17 through 24
0.1206 0.1262 0.1413 0.1520 0.1669 0.1717 0.1797 0.1938
Columns 25 through 32
0.2099 0.2203 0.2364 0.2528 0.2720 0.2937 0.2993 0.3187
Columns 33 through 40
0.3279 0.3357 0.3491 0.3718 0.3958 0.4186 0.4251 0.4420
Columns 41 through 48
0.4547 0.4615 0.4867 0.5003 0.5101 0.5237 0.5428 0.5662

Response_static_time =

Columns 1 through 8
0.0001 0.0004 0.0011 0.0013 0.0027 0.0168 0.0212 0.0309
Columns 9 through 16
0.0364 0.0440 0.0597 0.0688 0.0714 0.0770 0.0980 0.1096
Columns 17 through 24
0.1269 0.1351 0.1531 0.1674 0.1863 0.1959 0.2094 0.2295
Columns 25 through 32
0.2523 0.2701 0.2942 0.3192 0.3477 0.3791 0.3955 0.4265
Columns 33 through 40
0.4483 0.4698 0.4981 0.5362 0.5762 0.6157 0.6405 0.6767
Columns 41 through 48
0.7098 0.7387 0.7866 0.8242 0.8595 0.9001 0.9475 1.0000

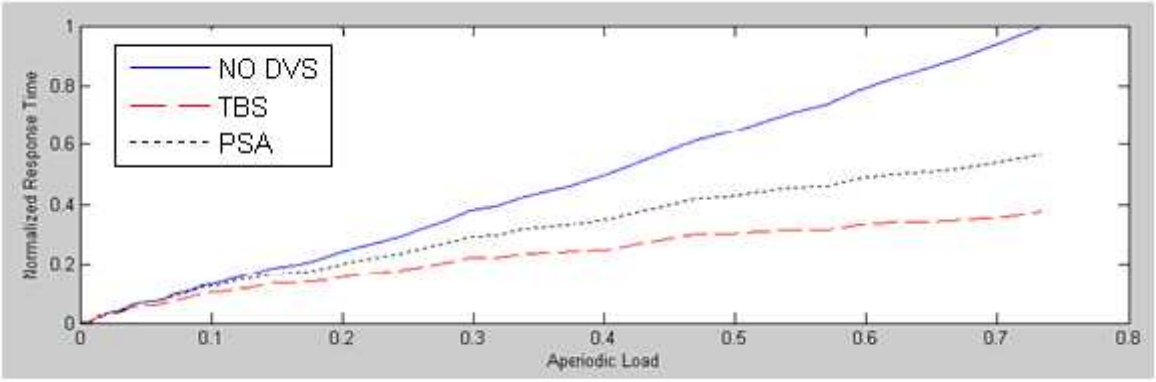
Maximum_Energy_Consuumed =

Columns 1 through 14
1 1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 15 through 28
1 1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 29 through 42
1 1 1 1 1 1 1 1 1 1 1 1 1 1
Columns 43 through 48
1 1 1 1 1 1

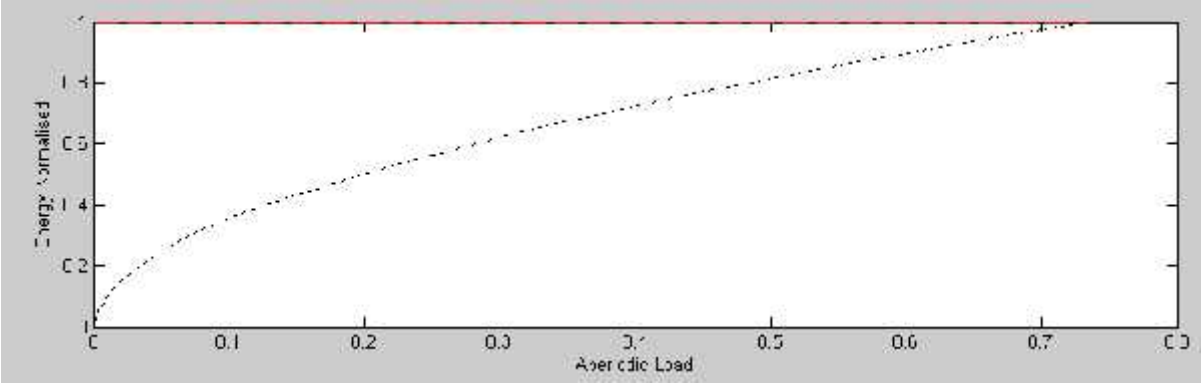
Energy_Required =

Columns 1 through 8
0.0212 0.0424 0.0636 0.0675 0.0887 0.1099 0.1311 0.1522
Columns 9 through 16
0.1734 0.1946 0.2158 0.2370 0.2582 0.2794 0.3006 0.3218
Columns 17 through 24
0.3430 0.3642 0.3854 0.4066 0.4278 0.4490 0.4702 0.4913
Columns 25 through 32
0.5125 0.5337 0.5549 0.5761 0.5973 0.6185 0.6397 0.6609
Columns 33 through 40
0.6821 0.7033 0.7245 0.7457 0.7669 0.7881 0.8093 0.8304
Columns 41 through 48

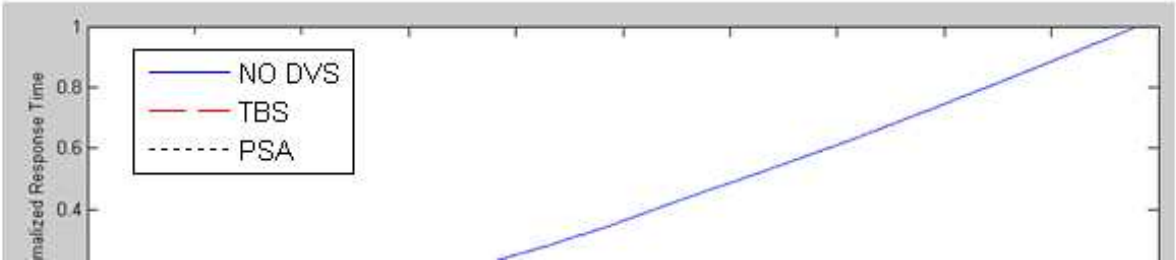
0.8516 0.8728 0.8940 0.9152 0.9364 0.9576 0.9788 1.0000



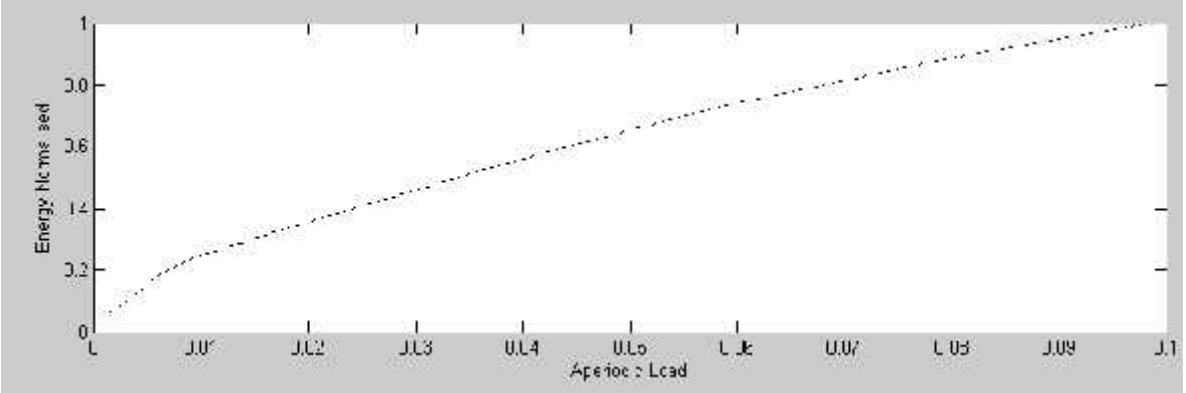
Response Time versus Aperiodic load (Case 1)



Energy Consumption Aperiodic load (Case 1)



Response Time versus Aperiodic load (Case 2)



Energy Consumption Aperiodic load (Case 2)

VI. CONCLUSION & FUTURE WORK:

Recently a lot of efforts are put for energy reduction of processors. As a drawback of reducing energy consumption of the system, its response time is increased, hence degrades the overall performance of the systems. In prior work, higher importance is given to energy reduction and reducing response time of hybrid tasks is unnoticed. We consider the importance of response time also while the energy reduction is achieved. In our work we propose a solution that reduces the power consumption of a multiprocessor system while the response time of tasks is kept within bound. In future the algorithm can be extended to accomplish and schedule large number of jobs over a huge network environment like Grid and Cloud Computing.

REFERENCES:

[1] A.P Chandrakasan, S.Sheng, and R.W. Brodersen. Low Power CMOS Digital Design, IEEE journal of Solid State Circuits, 1992, pp .472-484.

[2] T. D. Burd., T.A. Pering, A. J. Stratakos, and R. W. Rodersen, Adynamic Voltage Scaled Microprocessor system. IEEE Journal of Solid State Circuits, Vol. 35, No. 11,pp. 1571-1580,2000.

[3] D. Shin and J. Kim:, Dynamic voltage scaling of mixed task sets in priority-driven systems. IEEE Transaction on CAD of Integrated Circuits and Systems 25(3): 438-453 (2006).

[4] M. Spuri and G. Buttazzo,Scheduling Aperiodic Tasks in Dynamic Priority Systems, Journal of Real-Time Systems, 10(2):179-210, 1996.

[5] P. Pillai, and K. G. Shin, Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, In Proc. of ACM Symp. On Operating Systems Principles, pages 89-102, 2001.

[6] M. Joseph, *"Real-time Systems: Specification, Verification and Analysis,"* Prentice Hall, 1996.

[7] P. A. Laplante, *"Real-time Systems Design and Analysis, An Engineer Handbook,"* IEEE Computer Society, IEEE Press, 1993.

[8] C. M. Krishna and K. G. Shin, *"Real-Time Systems,"* MIT Press and McGraw-Hill Company, 1997.

[9] G. C. Buttazzo, *"Hard Real-Time Computing Systems: predictable scheduling algorithms and applications,"* Springer company, 2005.

[10] K. Frazer, *"Real-time Operating System Scheduling Algorithms,"* , 1997.

[11] W. A. Halang and A. D. Stoyenko, *"Real Time Computing,"* NATO ASI Series, Series F: Computer and Systems Sciences, Volume 127, Springer-Verlag company, 1994.

[12] J. A. Stankovic and K. Ramamritham, , *"Tutorial Hard Real-Time Systems,"* IEEE Computer Society Press, 1988.

[13] M. Garey, D. Johnson, *"Complexity Results for Multiprocessor Scheduling under Resource Constraints,"* SICOMP, Volume 4, Number 4, pp. 397-411, 1975.

[14] A. K. Mok, *"Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment,"* Ph.D. thesis. Department of Electronic Engineering and Computer Sciences, Mass. Inst. Technol., Cambridge MA, May, 1983.

[15] J. Y.-T. Leung and J. Whitehead, *"On the complexity of fixed priority scheduling of periodic real-time tasks,"* Performance Evaluation, Volume 2, pp. 237-250, 1982.

[16] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel, *"Proportionate progress: A notion of fairness in resource allocation,"* Algorithmica , Volume 15, Number 6, pp. 600-625, June, 1996.

[17] C. A. Phillips, C. Stein, E. Torng, and J. Wein, *"Optimal time-critical scheduling via resource augmentation,"* In Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 140-149, El Paso, Texas, 4-6 May, 1997.

[18] M. Moir and S. Ramamurthy, *"Pfair scheduling of fixed and migrating tasks on multiple resources,"* In Proceedings of the Real-Time Systems Symposium, IEEE Computer Society Press, Phoenix, AZ, December, 1999.

[19] J. Anderson and A. Srinivasan, *"Early release fair scheduling,"* In Proceedings of the EuroMicro Conference on Real-Time Systems, IEEE Computer Society Press, pp. 35-43, Stockholm, Sweden, June 2000.

[20] H. Aydin, P. Mejia-Alvarez, R. Melhem, and D. Mosse, *"Optimal reward-based scheduling of periodic real-time tasks,"* In Proceedings of the Real-Time Systems Symposium, IEEE Computer Society Press, Phoenix, AZ, December, 1999.

[21] S. Funk, J. Goossens, and S. Baruah, *"On-line Scheduling on Uniform Multiprocessors,"* , 22nd IEEE Real-Time Systems Symposium (RTSS'01), pp. 183-192, London, England, December, 2001.

[22] J. M. Bans, A. Arenas, and J. Labarta, *"Efficient Scheme to Allocate Soft-Aperiodic Tasks in Multiprocessor Hard Real-Time Systems,"* PDPTA 2002, pp. 809-815.

[23] Z. Xiangbin and T. Shiliang, *"An improved dynamic scheduling algorithm for multiprocessor real-time systems,"* PDCAT'2003. In Proceedings of the Fourth International Conference on Publication, pp. 710- 714, 27-29 August, 2003.

[24] S. Lauzac and R. Melhem, *"An Improved Rate-Monotonic Admission Control and Its Applications,"* IEEE Transactions on Computers, Volume 52, Number 3, pp. 337-350, March, 2003.

[25] P. Holman and J. H. Anderson, *"Using Supertasks to Improve Processor Utilization in Multiprocessor Real-Time Systems,"* 15th Euromicro Conference on Real-Time Systems (ECRTS'03), Porto, Portugal, 2-4 July, 2003.

[26] D. A. El-Kebbe, *"Real-Time Hybrid Task Scheduling Upon Multiprocessor Production Stages,"* International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, 22-26 April, 2003.

[27] B. Andersson and J. Jonsson, *"The Utilization Bounds of Partitioned and Pfair Static-Priority Scheduling on Multiprocessors are 50 percent,"* 15th Euromicro Conference on Real-Time Systems (ECRTS'03), Porto, Portugal, July 02-04, 2003.

[28] C. M. Krishna and K. G. Shin, *"Real-Time Systems,"* MIT Press and McGraw-Hill Company, 1997.

[29] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, *"A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms,"* Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Edited by J. Y. Leung, Published by CRC Press, Boca Raton, FL, USA, 2004.

[30] B. Sprunt, J. Lehoczky, and L. Sha, *"Exploiting Unused Periodic Time For Aperiodic Service Using the Extended Priority Exchange Algorithm,"* In Proceedings of the 9th Real-Time Systems Symposium, pp. 251-258. IEEE, Huntsville, AL, December, 1988.

[31] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *"Deadline Scheduling for Real-Time Systems, EDF and related algorithms,"* Kluwer Academia Publishers, 1998.

[32] B. Sprunt, *"Aperiodic Task Scheduling for Real-Time Systems,"* Ph.D. Thesis, Department of Electrical and Computer Engineering Carnegie Mellon University, August, 1990.

[33] M. Spuri and G. C. Buttazzo, *"Efficient Aperiodic Service under Earliest Deadline Scheduling,"* In Proceedings IEEE Real-Time Systems Symposium, pp. 2-11, San Juan, Puerto Rice, 7-9 December, 1994.

[34] M. Spuri and G. Buttazzo, *"Scheduling Aperiodic Tasks in Dynamic Priority Systems,"* The Journal of Real-Time Systems.

[34] Osman S. Unsal and Israel Koren. System-level power-aware design techniques in real-time systems. Proceedings of the IEEE, 91(7):1055{1069, 2003.

[35] Wu chun Feng, Michael S. Warren, and Eric Weigle. The bladed beowulf: A coste _ective alternative to traditional beowulf. In IEEE International Conference on Cluster Computing (CLUSTER 2002), 23-26, Chicago, IL, USA, 2002.

[36] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez. Reducing power in high-performance microprocessors. In DAC '98: Proceedings of the 35th annual conference on Design automation, pages 732{737, New York, NY, USA, 1998. ACM Press.

[37] K. Nowka, G. Carpenter, and B. Brock. The design and application of the powerpc 405LP energy-e_cient system on chip. IBM Journal of Research and Development, 47(5/6), September/November 2003.

[38] Kanishka Lahiri, Sujit Dey, Debashis Panigrahi, and Anand Raghunathan. Battery-driven system design: A new frontier in low power design. In ASPDAC '02: Proceedings of the 2002 conference on Asia South Paci_c design automation/ VLSI Design, page 261, Washington, DC, USA, 2002. IEEE Computer Society.

[39] P. Pillai, and K. G. Shin, Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, In Proc. of ACM Symp. On Operating Systems Principles, pages 89-102, 2001.

[40] D. Shin and J. Kim:, Dynamic voltage scaling of mixed task sets in priority-driven systems. IEEE Transaction on CAD of Integrated Circuits and Systems 25(3): 438-453 (2006)

[41] T. Ishihara and H. Yasuura, Voltage Scheduling Problem for Dynamically Variable Voltage Processors, In Proceedings of International Symposium On Low Power Electronics and Design, 1998, pp. 197-202.

[42] Nasro Min-Allah, Asad-Raza Kazmi2, Ishtiaq Al3, Xing Jian-Sheng, Wang Yong-Ji "Minimizing Response Time Implication in DVS Scheduling for Low Power Embedded Systems"