Middle-East Journal of Scientific Research 15 (1): 94-105, 2013
ISSN 1990-9233
© IDOSI Publications, 2013
DOI: 10.5829/idosi.mejsr.2013.15.1.603

Power Aware Scheduling Algorithm for Real-Time Tasks over Multi-Processors

[1]*Muhammad Zakarya*, [1]*Uzma and* [2]*Ayaz Ali Khan*

[1]Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan
[2]Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan

Abstract: Multi-processors are used for high performance scientific computing, where each processor is assigned the same or different workload. Real-time systems are those systems that must complete the given operation in the given time or deadline. A number of scheduling algorithms are proposed in the literature that are optimal on uni-processor systems and are modified for scheduling over multi-processors. In this paper we propose a new scheduling algorithm SC-WITH-DVFS for multi-processors having four properties. The first one is that energy efficiency is obtained by dynamically scaling the frequency of each processor. The 2nd one is to schedule the most important tasks first by reducing the drawback of RM and DM and 3rd one to extend the concept of [1], for multi-processors the 4th one is to use TL_PLANE and task splitting techniques for load balancing to further enhance the power consumption. Proposed algorithm is verified for high efficiency for multi-processor real-time systems using mathematical modeling and simulation results.

Key words: Multiprocessor · Real-time Systems · RM · DM · DVFS · Scheduler

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. In recent times it is realized there is a need for energy reduction in processors. And a lot of work has been done on minimizing the energy reduction. When we reduce the energy consumption then the response time is increased. And it will degrade the performance of real time systems. 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 [1-4]. 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:

$$P_{avg} = P_{capt} + P_l + P_{stdby} + P_{SC} \tag{2}$$

Corresponding Author: Muhammad Zakarya, Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan.

where P_{capt}, P_l, P_{stdby} and P_{SC} is capacitance, leakage, standby and short circuit power. The P_l and P_{stdby} 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:

$$P_{capt} = \alpha C V_d^2 f \tag{3}$$

where α is the transition activity dependent 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:

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

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:

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

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:

$$P = \alpha C V^2 f \tag{6}$$

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

Multi-Processor Scheduling: The scheduling problem has predominantly studied on a uniprocessor system, which contains a single processing unit and all the jobs must be executed in it. In multiprocessor system the tasks must be executed on more than one processor. One of the optimal scheduling algorithms for multiprocessor system is Pfair scheduling. On the other hand, to assign the tasks optimally to the processing unit is an NP-hard problem [5-8]. As a result we must use the heuristics. The main purpose of heuristics is to find out that all tasks should be feasibly schedulable but is not responsible for the pledge allocation of tasks. Up to now, several heuristic multiprocessing unit scheduling algorithms are suggested and offered [9, 8, 10, 11, 12, 3, 13, 5, 6, 14, 2, 15, 16, 17, 18, 19]. When the feasibility of the task is checked then the communication overhead must also be considered. For example, assume that a task cannot begin before getting the output of another task. When the tasks are executing on the same processor then the communication overhead is zero. On the other hand when the tasks are executing on the separate processing units, then the communication overhead is greater and should be notified when checking the allotment for feasibility. The following postulations may be made, when someone is going to design a multiprocessing unit scheduling algorithm:

The Preemption of Job Is Allowable or Not: When the task is executing on a processor then it could be preempted by some higher priority task and its execution would be resumed later on. We suppose that there is no penalty linked with such preemption.

The Migration of Job Is Allowed or Not: The task that is executing on any processor may be preempted by another processor and can resume its execution on that processor. And there is no penalty allied with such migration.

The Job Parallelism Is Prohibited: That is, when the task is executed it must execute on one processing unit at any particular instant of time.

RM and DM are important scheduling algorithms for real-time systems. RM and DM both are fixed priority algorithms and give equal importance to each task, that 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 [20]. 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 [15] the authors have proposed a new algorithm that can schedule the most important tasks first. In [1] 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.

Proposed Solution: Scheduling of jobs optimally on multi-processors [21-24] is a hot research issue in academia, research laboratories and industry. There is a lot of work done for scheduling real-time tasks to achieve energy efficiency in multi-processor environment. In our algorithm SC-WITH-DVFS we have scheduled most important tasks first that prevents the starvation problem. We have also introduced the task splitting concept to utilize each processor with equal workload. The [15] 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. [15] 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. Assign weighted percentage (depends on task importance) to these three components task period, task deadline and priority component (PC).

For example:

Task deadline = 30%
Task period = 20%

Task Priority (TP) is the addition of weighted percentage of task period and task deadline i.e.

Task Priority = 50%

Task period component (TPC) is calculated by.

TPC = (weightage percentage * task period)

Now if task period is 25 then

TPC = 20% * 25 = 5

Task deadline component (TDC) is calculated by.

TDC = (weightage percentage * task deadline)

Now if task deadline is 40 then

TDC = 30% * 40 = 12

Priority component (PC) is calculated by

PC = (weightage percentage for priority - task priority)

Let if the task priority is 15 as assign by the user then its priority component is:

PC = 50 – 15 = 35

SC is calculated by

SC = TPC + TDC + PC

For example:

SC = 5 + 12 + 35
= 52

The task with highest scheduling component is scheduled first.

In our proposed algorithm SC-WITH-DVFS we use CPU burst for the task utilization that means a task completes its execution when its CPU burst is executed. We use tl_plane for load balancing on processor. In our proposed algorithm we 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.

Let we have a task set (T1, T2, T3 . . . . Tn) and each task is of style (φI, Ci, Pi, Di). All tasks must be executed according to its scheduling component increasing order. Priority based scheduling is used to assign the resource to the tasks. Tasks having higher scheduling components are assigned high priority while small scheduling component is considered as low priority. Ci/pi denote the utilization of the tasks and the total processors utilization is given by

$$U = \sum_{i=1}^{n} Ci/Pi$$

Algorithm 1: Set frequency for tasks

```
1. Sort all tasks in an increasing order of scheduling component of each task in active (t) set i.e. SC1 ≤ SC2 ≤ SC3----------≤ SCn
2. U = 0; // global variable
3. P = m:
4. For i = 1 to |Ti∈active (t) |do
5. U = U + ui;
6. T = U;
7. For i = 1 to |Ti∈ active (t) |do
8. If Ui > T / P then
9. Ti.speed = ui;
10. P = P - 1;
11. T = T - ui;
12. Else
13. Ti.speed = C / M;
```

In our proposed algorithm we have extended the work proposed in [1] and [15] and have implemented a new scheduling algorithm i.e. SC-WITH-DVFS for multi-processor real-time systems. The importance of our work is to reduce the drawback of RM and DM and also to minimize the power consumption of processors by scaling down the processor voltage and frequency, according to its active tasks. We have also focused to balance the load on processors by using the concept of tl_plane (to maximum utilize the processors to <= tl_plane) and task splitting techniques to equal load on each processor. In [1] DVFS is used to dynamically scale the Frequency of processors according to active tasks set in the ready queue. In [15, 25, 26, 27, 28, 29] the authors have scheduled the most important tasks first. We find scheduling components for each task. Our proposed algorithm has the following properties.

- SC-WITH-DVFS is used to saves energy when we dynamically scale the frequency for each processor according to the current active tasks. First we find the frequency for each active task by using algorithm 1 and then we scale the frequency of processor according to the current executed task.
- SC-WITH-DVFS reduces the drawback of RM and DM by finding the scheduling component of each task according to [1]. The task having highest scheduling component is considered the most important task and therefore is scheduled first,
- SC-WITH-DVFS is used to extend [1] for multiprocessors.
- Using tl_plane and task splitting techniques are used to achieve load balancing on processors. tl_plane is a restriction for the utilization of the processors. Along with this restriction the task splitting technique is used in our proposed algorithm SC-WITH-DVFS, to balance the load on each processor.

We set the frequency for the current active tasks, by using algorithm1. In our algorithm SC_WITH_DVFS we sort the scheduling component of all active tasks in descending order so that the most important task is schedule first. U contains the total utilization of all active tasks. M is the number of processors. We find the average utilization C/M, if task utilization is greater than its average utilization then we assign the utilization of the task as its frequency and the number of processor is reduced by 1, otherwise we assign the average utilization as its frequency to this task. So algorithm 1 is used to set the frequency for each task. Now by using algorithm 2 we initialize the tl_plane and find the execution cycle for each active task. Next we find the processor id for each active task and also set the frequency for the processor according to its current active task in the ready queue. We find the execution cycle for each task and also set processor frequency according to the current active task frequency. In step 6 N tasks are assigned to N processors and then processor utilization is subtracted from the tl_plane and assigns value to the rcycle_z (remaining cycle of a processor z).

Using step 13 the remaining tasks are dividing into N processors. The method of division is that next task is assigned to minimum utilized processor as mentioned in step 15. Now if li of a task is less than rcycle_min (remaining cycle on minimum processor) then the new task will start it's execution from the existing utilization ti_start on current processor as mentioned in step 2 and current processor utilization is increased by using step 23. But in case if li of task is greater than rcycle_min, then li of that task is splitted using step 26. TS is a part of that task which is migrated to another minimum utilized processor, while the splitted task frequency and its id is not changed, only the processor id is changed. To achieve load balancing the TS part of the task is further splitted using step 40 and 41.

In next section we have shown the performance and results of our algorithm.

Algorithm 2: SC-WITH-DVFS

```
1: for i = 0 to |active tasks|
TPCi = Wpi * Tpi
TDCi = Wpi * Tdi
Pci = Tpci + TDCi;
Sci = TPCi + TDCi + Pci
2: For i =0 to |active Ti| do
C = C + Ci;
TL-plane = ⌊C / m⌋
3: Set frequency for tasks.
4: Z = 1;
5: For i = 0 to Ti do
Li = Ci;
6: If Li <= TL_plane
7: If Z ≤ M then
8: task_exe=t + Li;
// to time for Ti to be executed
9: Ti.proc_id = Z;
//task must be execute on z processor
10: Z.task_id = Ti;
11: Z.speed = Ti.speed
12: Proc_z_util = proc_z_util + Li;
Rcycle_z = tl_plane - proc_z_util;
13: Z = Z + 1
14: Else
15: compare processors & find min execute
cycle processor
16: If Li < Rcycle_min
17: task_exe = Li;
18: task_proc_id = min;
19: Ti_start = proc_i_util;
// task start its execution from Ti_start
20: z_task_id = Ti;
21: z_speed = Ti_speed;
22: proc_min_util = proc_min_util + Li;
23: Rcycle_min = TL_plane - proc_min_util;
24: If Li > Rcycle_min
25: TSi = Li - Rcycle_min;
26: repeat step 15;
27: task_exe = Rcycle_min;
28: Repeate step 17 to 20;
29: proc_min_util =
 proc_min_util + Rcycle_min;
30: repeat step 22;
31: If TSi ≠ 0;
```

Algorithm 2: SC-WITH-DVFS

```
32: Repeat step 15.
33: if TSi < Rcycle_min
34: Task_exe = spi;
35: repeate step 17 to 20;
36: proc_min_util = proc_min_util + TSi;
37: repeat step 22;
38: else
39: if TSi ≥ Rcycle_min
40: TS2 = TSi - Rcycle_min;
41: task_exe = Rcycle_min;
42: repeate step 17 to 20;
43: proc_min_util =
 proc_min_util + Rcycle_min;
44: repeat step 22;
45: if TS2 ≠ 0
46: repeate step 15;
47: task_exe = sp2i;
48: repeate step 17 to 20;
49: proc_min_util = proc_min_util + TS2;
50: repeate step 22;
51: If Li <= TL_plane
52: l2 = li - tl_plane;
53: repeat step 15;
54: task_exe = tl_plane;
55: repeat step 17 to 20;
56: proc_min_util = proc_min_util + tl_plane;
57: repet step 22;
58: if l2 ≠ 0
59: repeat step 15;
60: task_exe = l2;
61: repeat step 17 to 20;
62: proc_min_util = proc_min_util + l2;
63: repeat step 22;
```

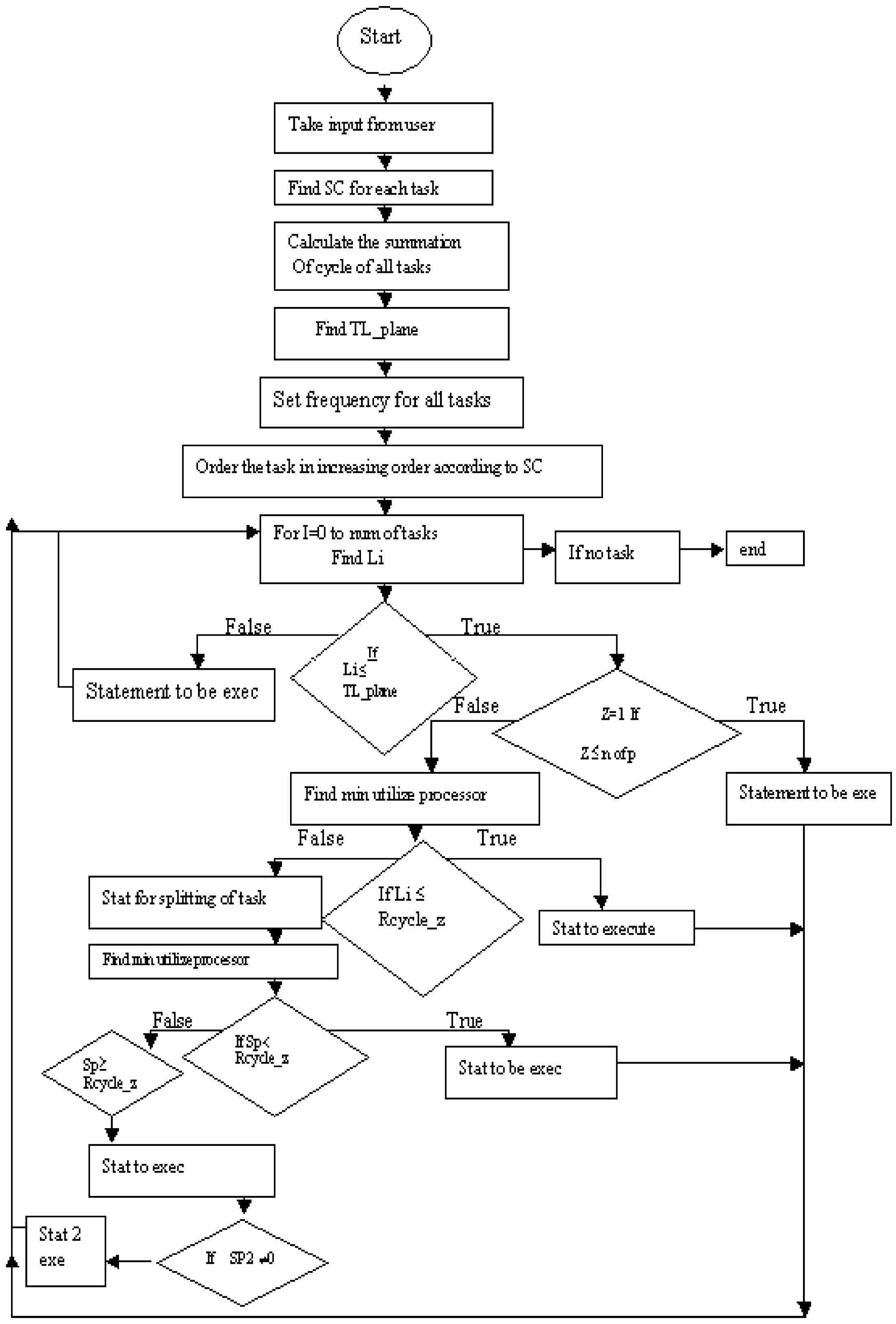

Fig. 3.1: Flow Chart

RESULTS

Example 1

Table 6.1: tasks set containing 10 tasks with his properties

Inputs:

	Pi	Di	Ci	Priority
T1	25	40	7	6
T2	50	60	12	1
T3	25	30	8	10
T4	40	15	3	3
T5	10	15	7	8
T6	25	15	7	5
T7	15	25	5	4
T8	30	95	6	2
T9	75	85	4	9
T10	20	50	13	20

Table 6.2: tasks set containing 10 tasks with his properties

Outputs:

	TPC	TDC	PC	SC
T1	5	12	44	61
T2	10	18	49	77
T3	5	9	40	54
T4	8	5	47	60
T5	2	5	42	49
T6	5	5	45	55
T7	3	8	46	57
T8	6	29	48	83
T9	15	26	41	82
T10	4	15	30	49

Sort task in increasing order based on its scheduling components.

T5 ≤ T10 ≤ T3 ≤ T6 ≤ T7 ≤ T4 ≤ T1 ≤ T2 ≤ T9 ≤ T8

Number of tasks = 10

Number of processors = 3

$$\text{TL\_plane} = \sum_{i=1}^{10} \text{Ci/P} = 24$$

Gantt Chart: A

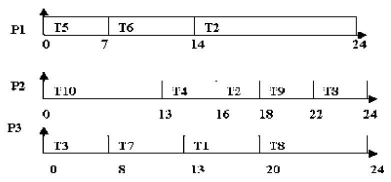

Fig. 6.1: Analysis load on three processors

Now we reduce the number of processor

Number of tasks = 10

Number of processors = 2

$$\text{TL\_plane} = \sum_{i=1}^{10} \text{Ci/P} = 37$$

Gantt Chart: B

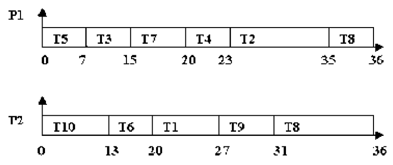

Fig. 6.2: Analysis load on two processors

From Graph A and B it is clear that when number of processor is increase the utilization of processor is decrease.

Example 2

Table 6.3: tasks set containing 20 tasks with his properties

Inputs:

Ti	Pi	Di	Ci	Priority
T1	25	40	7	6
T2	50	60	12	1
T3	25	30	8	10
T4	40	15	3	3
T5	10	15	7	8
T6	25	15	7	5
T7	15	25	5	4
T8	30	95	6	7
T9	75	85	4	9
T10	20	50	13	20
T11	85	60	9	15
T12	10	45	6	14
T13	80	65	8	25
T14	35	70	4	7
T15	10	45	5	12
T16	25	35	12	14
T17	45	75	3	13
T18	65	90	10	3
T19	95	100	14	13
T20	25	20	15	21

Table 6.4: tasks set containing 20 tasks with his properties

Outputs:

Ti	TPC	TDC	PC	SC
T1	5	12	44	61
T2	10	18	49	77
T3	5	9	40	54
T4	8	5	47	60
T5	2	5	42	49
T6	5	5	45	55
T7	3	8	46	57
T8	6	29	48	83
T9	15	26	41	82
T10	4	15	30	49
T11	17	18	35	70
T12	2	14	36	52
T13	16	20	25	61
T14	7	21	43	71
T15	2	14	38	53
T16	5	11	36	52
T17	9	7	37	53
T18	13	27	47	87
T19	19	30	37	86
T20	5	6	29	40

$T20 \le T5 \le T10 \le T12 \le T16 \le T15 \le T17 \le T3 \le T6 \le T7 \le T4 \le T1 \le T13 \le T11 \le T14 \le T2 \le T9 \le T8 \le T19 \le T18$

Number of tasks = 20

Number of processors = 3

$$TL\_plane = \sum_{i=1}^{20} Ci/P = 53$$

Gantt Chart: A

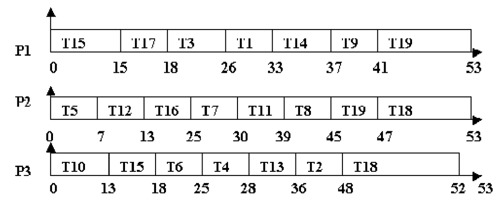

Fig. 6.3: Analysis load on three processors

Now we reduce the number of processor

Number of tasks = 10

Number of processors = 2

$$TL\_plane = \sum_{i=1}^{10} Ci/P = 79$$

Gantt Chart: B

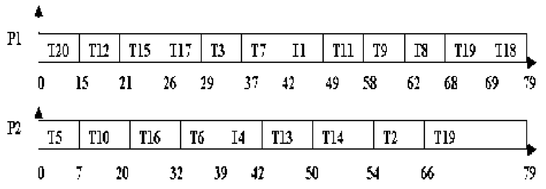

Fig. 6.4: Analysis load on two processors

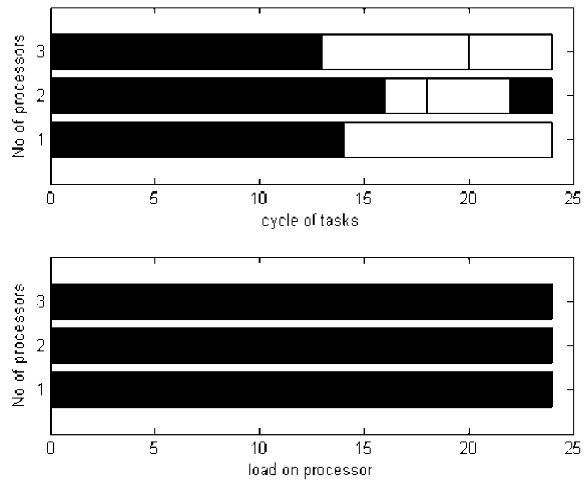

Fig. 6.5: Simulation Example 1

Our simulations were created in parallel computing toolbox of MATLAB. Different task sets were scheduled and the algorithm was verified.

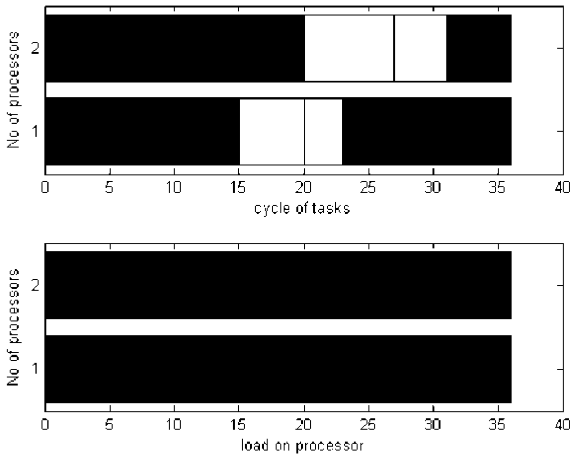

Fig. 6.6: Simulation Example 2

The first 3 tasks are assigned to the 3 processors and the next tasks are assigned to the processor having minimum No of cycle and so on. The last task T3 of cycle 12 is greater than the remaining cycle on a P1. So task 3 is splitted i.e. 10 cycle of T3 is executed on processor 1 and the remaining 2 cycle of T3 is executed on the processor having minimum cycles that is P2. The last task T4 of cycle 6 is greater than the remaining cycle on P3, therefore task 4 is splitted i.e. 4 cycle of T4 is executed on processor 3 and the remaining 2 cycle of T4 is executed on the processor having minimum cycles that is P2. And thus all the processor are equally utilized.

The first 2 tasks are assigned to the 2 processors and the next task is assigned to the processor having minimum No of cycle and so on. The last task T5 of cycle 6 is greater than the remaining cycle on a P2. So task 5 is splitted i.e. 5 cycle of T5 are executed on processor P2 and the remaining 1 cycle of T5 is executed on the processor P1. The utilization on P1 and P2 is 36 i.e. equal loads on two processors.

The first 3 tasks are assigned to the 3 processors and the next task is assigned to the processor having minimum No of cycle and so on. The last task T7 of cycle 14 is greater than the remaining cycle on a P1. So task T7 is splitted, 12 cycles of T7 is executed on processor P1 and the remaining 2 cycle of T7 is executed on the processor having minimum cycle. this is P2.the last tasks T8 of cycle 10 is greater than remaining cycle on processor P2.so T8 is splitted 6 cycle of T8 is executed on P2 and reaming 4 cycles is executed on p3. The utilization on P1, P2 is 53 and P3 is 52.

The first 2 tasks are assigned to the 2 processors and the next task is assigned to the processor having minimum No of cycle and so on. The last task T9 of cycle 14 is

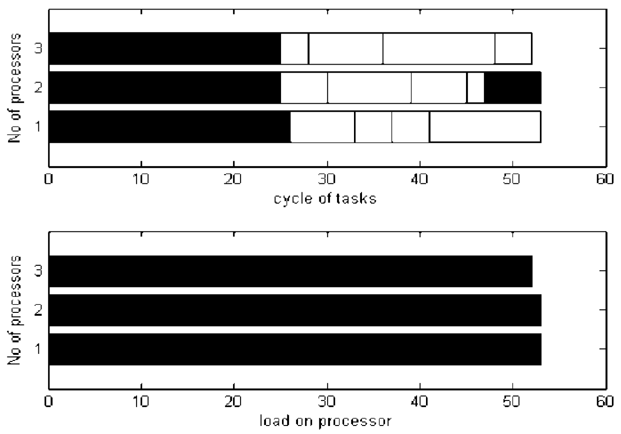

Fig. 6.7: Simulation Example 3

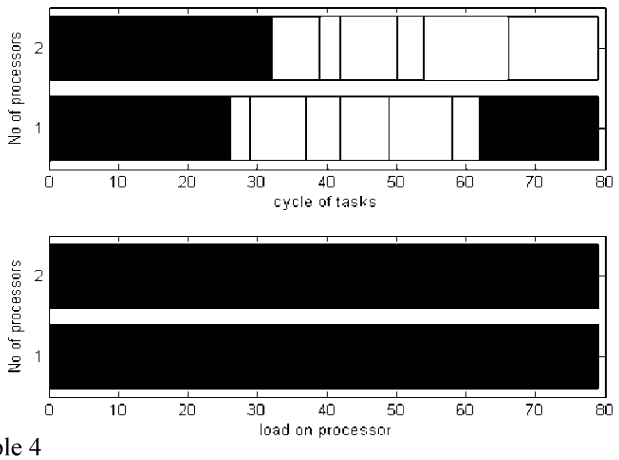

Fig. 6.8: Simulation Example 4

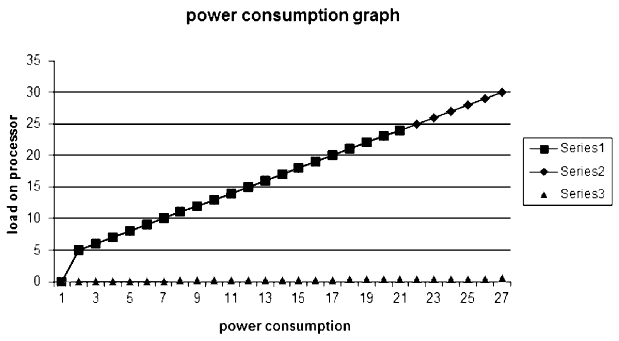

Fig. 6.9: Comparison of energy consumption

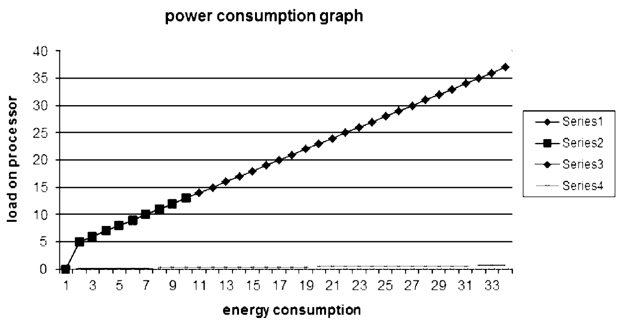

Fig. 6.9 comparison of energy consumption

Table Comparison: Proposed Algorithm with Exsisting Algorithms

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 multip rocessor systems	Real-time scheduling with task splitting on multiprocessors (MP):	Proposed Algorithm
Task importance	Equal	Equal	Equal	Equal	Equal	Priority component is added	Equal	Equal	Priority component is added
Processor	Uniprocessor	Multiprocessor	Multiprocessor	Multiprocessor	Multiprocessor	Uniprocessor	Multiprocessor	Multiprocessor	Multiprocessor
Scheduling criteria	Task period	Task deadline	Task period	Task utilization	Task period	Task periodTask deadlineTask priority	Task period	Task period	Task periodTask deadlineTask priorityTL-plane
Task scheduling	Tasks are arranged in ascending order based on period	Tasks are arranged in ascending order based on deadline.	Tasks are arranged in ascending order based on period	Tasks are arranged in ascending order based on utilization	Tasks are arranged in ascending order based on period	Tasks are arranged in ascending order based on scheduling component	Tasks are arranged in ascending order based on period	Tasks are arranged in ascending order based on period	Tasksarearranged in ascending order based on scheduling component
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 multip rocessor systems	Real-time scheduling with task splitting on multiprocessors (MP):	Proposed Algorithm
Time difference	Task period does not change with time	Task deadline does not change with time	Task period does not change with time	Task utilization does not change with time	Task period does not change with time	Task period, task deadline,task priority does not change with time	Task period does not change with time	Task period does not change with time	Task period, task deadline, task priority doesnot 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	StarvationIs reduce for only uniprocessor	Starvation	Starvation, Load balancing	N/A
Energy saving by	N/A	N/A	N/A	N/A	N/A	N/A	DVS	N/A	Load balancing

greater than the remaining cycle on a P2. So task T9 is splitted, 13 cycle of T19 is executed on processor P2 and the remaining 1 cycle of T9 is executed on the processor P1. The utilization on P1 and P2 is 79 i.e. equal loads on two processors.

Comparative Study: Our algorithm was executed several times. It runs in n^2 worst case analysis. In our algorithm 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.

CONCLUSION

In this paper we have proposed a new scheduling algorithm for multiprocessor real-time systems, which is used for the energy efficiency [30, 31, 32, 29, 33, 34] as will as to reduce the drawback of RM and DM by scheduling the most important task first. Our algorithm also focuses on load balancing on processors. 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. We consider the importance of response time of important real-time tasks 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 important 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. Zhang, D., F. Chen, H. Li, S. Jin and D. Guo, 2011. An energy-efficient scheduling algorithm for sporadic real-time tasks in multiprocessor systems, (IEEE international confererence on high performance computing and communication, in 2011).
2. Mohammadi, A. and S.G. Akl, 2005. Scheduling Algorithms for Real-Time Systems, Arezou Mohammadi and Selim G. Akl. (July 15, 2005).

3. Lindh, F., T. Otnes and J. Wennerström, 0000. Scheduling Algorithms for Real-Time Systems.
4. Chen, J. and C. Fu Kuo, 2006. Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms (Council of Agriculture, Executive Yuan, Taiwan, since Dec. 2006.)
5. Singh, J., 0000. An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System.
6. Baruah, S. and J. Goossens, 0000. Deadline Monotonic Scheduling on Uniform Multiprocessors.
7. Buttazzo, G.C., 2005. Rate Monotonic vs. EDF: Judgment Day, (2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.)
8. Kato, S., A. Takeda and N. Yamasaki, 2008. Global Rate Monotonic Scheduling with Priority Promotion (published in the IPSJ Transactions on Advanced Computing System (ACS), 2(1): 64-74.
9. Manabe, Y. and S. Aoyagi, 1995. A Feasibility Decision Algorithm for Rate Monotonic Scheduling of Periodic Real-Time Tasks, (NTT Basic Research Laboratories 3-1 Morinosato-Wakamiya, Atsugi-shi, Kanagawa 243-01 Japan, 1995 IEEE).
10. Baskaran, S. and P. Thambidurai, 2012. Energy Efficient Scheduling for Real-time Embedded Systems with Precedence and Resource Constraints, (International Journal of Computer Science, Engineering and Applications (IJCSEA) 2(2).
11. Singh, J., B. Patra and S. Prased Singh, 2011. An Algorithm to Reduce the Time Complexity Of Earliest Deadline First Scheduling Algorithm in Real-Time System, (IJACSA) International Journal of Advanced Computer Science and Applications, 2(2).
12. Srinivasan, A. and S. Baruah, 2001. Deadline-based scheduling of periodic Task systems on Multiprocessor, (the University of North Carolina, department of computer science, CB3175, Chapel Hill, NC 27599- 3175, USA received 22 May 2001; received in revised form 10 January 2002 communicated by F.Y.L.chin).
13. Lehczky, J., L. Sha and Y. Ding, 1989. The rate monotonic scheduling algorithm, Exact Characterization And Average Case Behavior, (Department of Statistics, Department of Computer Science Carnegie Mellon University and 1989 IEEE).
14. Zhu, D., N. Abou Ghazaleh, D. Moss´e and R. Melhem, 2002. Power Aware Scheduling for AND/OR Graphs in Multi-Processor Real-Time Systems, (Conference on Parallel Processing, Aug. 2002).
15. Y.C. and R. Ramesh, 2010. A new scheduling algorithm for real time system, (International Journal of Computer and Electrical Engineering, 2(6): 1793-8163.
16. Zakarya, M., I.U. Rahman and A.A. Khan, 2012. Energy crisis, global warming & IT industry: Can the IT professionals make it better some day? A review. In Emerging Technologies (ICET), 2012 International Conference on, pp: 1-6. IEEE.
17. Zakarya, M. and I.U. Rahman, 2012. Towards Energy Efficient High Performance Computing Perceptions, Hurdles & Solutions, Technical Journal UET Taxila (Pakistan) 2012.
18. Khan, A.A. and M. Zakarya, 2010. Performance Sensitive Power Aware Multiprocessor Scheduling in Real-time Systems, Technical Journal UET Taxila (Pakistan).
19. Beitollahi, H., S. Ghassem Miremadi and G. Deconinck, 2007. Fault-Tolerant Earliest Deadline-First Scheduling Algorithm (©2007 IEEE).
20. Lakshmanan, K., R. Kumar and J.P. Lehoczky, 0000. Partitioned Fixed-Priority Preemptive Scheduling for Multi-Core Processors, Carnegie Mellon University Pittsburgh, PA 15213, USA
21. Anderson, B., 2003. static-priority scheduling on Multiprocessor, (Department of Computer Engineering Chalmers University of Technology.
22. Salmani, V., S. Taghavi Zargar and M. Naghibzadeh, 2005. A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks, (Word Academy of Science, Engineering and Technology.
23. Seo, E., J. Jeong, S. Park and J. Lee, 2008. Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors, IEEE Transactions on Parallel and Distributed Systems, 19(11).
24. Kato, S. and N. Yamasaki, 2007. "Real-Time Scheduling with Task Splitting on Multiprocessors", In Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp: 441-450, 21-24 Aug. 2007.
25. Zakarya, M., I. Rahman and A. Ali Khan, 2012. Energy Crisis, Global Warming and IT Industry: Can the IT Professionals make it better Some Day? A Review:" (©2012 IEEE).
26. Zakarya, M. and I. Rahman, 2011. Towards Energy Efficient High Performance Computing Perceptions, Hurdles & Solutions" (Technical Journal, University of Engineering and Technology Taxila, 2011).

27. Min-Allah, N., A. Raza Kazmi, I. Ali, X. Jian-Sheng, W. Yong, 0000. Minimizing Response Time Implication in DVS Scheduling for Low Power Embedded Systems.
28. Saliu, M.O. and K. Salah, 0000. Starvation Problem in CPU Scheduling For Multimedia Systems.
29. Wang, L., S.U. Khan, D. Chen, J. Kołodziej and R. Ranjan, 2013. Energy-aware parallel task scheduling in a cluster - Future Generation Computer Systems.
30. Zakarya, M., N. Dilawar, M.A. Khattak and M. Hayat, 2013. Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core. World Applied Sciences Journal, 22(10): 1431-1439.
31. Zakarya, M. and A.A. Khan, 2012. Cloud QoS, High Availability and Service Security Issues with Solutions. IJCSNS, 12(7): 71.
32. Valentini, Giorgio Luigi, *et al.*, 2011. "An overview of energy efficiency techniques in cluster computing systems."
33. Zakarya, Muhammad, Izaz Ur Rahman and Imtiaz Ullah, 0000. "An Overview of File Server Group in Distributed Systems." Ijtech.org.
34. Zakarya, Muhammad, Ayaz Ali Khan and Hameed Hussain, 2010. "Grid High Availability and Service Security Issues with Solutions, pp: 978-1.