World Applied Sciences Journal 22 (10): 1431-1439, 2013
ISSN 1818-4952
© IDOSI Publications, 2013
DOI: 10.5829/idosi.wasj.2013.22.10.602

Energy Efficient Workload Balancing Algorithm for Real-Time Tasks over Multi-Core

Muhammad Zakarya, Nadia Dilawar, Muazzam Ali Khattak and Maqssod Hayat

Department of Computer Science, Abdul Wali Khan University, Mardan, Pakistan

Abstract: A real-time system must respond fast enough that it can serve the task in a particular time interval. The key constraints in real-time systems are to meet timing bounds and for these systems it is necessary to complete all of their tasks in time. Due to increasing complexity of real-time applications, powerful processors are needed to accommodate computational starving applications. Multi-core processor is a solution to such applications. However, multi-core processor is still in its immaturity stage and there is need to address the multi-core partitioning issues with perspective that all cores are equally utilized. More powerful processors are required to execute such applications. Single core processor are not enough capable to meet the increasing complexity of real-time applications. Multiple single core processors in any system require more power consumption which is not tolerable. Multi-core processor provides the solution to complex and computational starving real-time applications. Multi-core processors can provide higher computational power at lower power consumption. Distribution of tasks deals with partitioning the given workload on all processing cores in such a way that all tasks must meet their deadlines. Advantage of multi-core processors can only be fully realized if all cores have equal workload. No workload partitioning technique has been proposed so far that ensures the distribution of workload on all cores equally. Recently, multi-core systems have presented a research challenge to real-time system designers form scheduling perspectives and a lot of attention is being devoted to this research area, however, to the best of our knowledge, no solution addresses real-time systems issues associated with scheduling. The aim of this research is to answer the scheduling problem for multi-core processors and an efficient workload partitioning technique that can fully utilize all the processing cores in multi-core system.

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

INTRODUCTION

A real-time system can be defined as a system that must perform the specific task within given or specific time. Real-time systems have their applications in computing, communication and information systems, such as air-bag in car, the bag inflates neither too soon nor too late in order to be aid and not to be potentially harmful. In real time systems processes are referred to as tasks and these have certain temporal qualities and restrictions [1]. All tasks will have a deadline, an execution time and a release time. In addition there are other temporal attributes that may be assigned to a task. The three mentioned are the basic one. The release time, or ready time is when the task is made ready for execution. The deadline is when a given task must be done executing and the execution time is how long time it takes to run a given task [2]. Real-time systems can be divided in to two categories, hard real-time systems and soft real-time systems. In hard real-time systems the deadline must be assured, for example a rocket fuel injection procedure must be completed in time. Soft real-time system provides priority to real-time tasks over the other tasks. In the case of real time system's failure results are devastating. Multi-core processor is a processor with two or more than two independent processing cores or units, sharing main memory. Multi-core processors supply high performance at low power consumption than the single core processor. Higher throughput is achieved by increasing the clock speed and adding the additional cores on the same chip. A single core processor can read and execute a single instruction at a time but multi-core processor can read and execute more than one instruction at the same time. These multiple cores in a single processor can be homogeneous

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

or heterogeneous. Homogeneous multi-core systems consist of the same cores but heterogeneous multi-core systems don't have the same cores. Applications that require more computational power are now focused on the multi-core architecture because it increases the throughput without increasing the operating frequency [2]. In multi-core systems, workload is distributed among all available processing cores but it is still a challenging task that how to spread the workload uniformly among all processing cores. Dividing the workload into subsets in a way that all these subsets are schedulable on each core is recognized as the partitioning problem.

Demand for more power and energy have increased the energy cost and the energy requirements as well. The production of energy is not sufficient for the usage as there is a big difference between both ratios. Alternatively energy distribution centers solve this crisis problem using load shading mechanism that have an impact on industry and economy. This is not only our daily life electronic equipment's that we use for our protection against the warm season, but there are other fields that have forced the industry, academia and researchers to think for solutions to current energy crisis. Our IT industry is one of the huge consumers. Information and Communication Technology industry is well thought-out to be a chief consumer of energy and in turn energetic contributor to the Green House Gas (GHG) emissions. Different aspects of IT industry are subject to the huge power usage. Computer Networks, Huge processing power & requirements, control of heat generated by processing speed and other major issues that are specific to IT industry are playing a major role in power consumption.

Literature Review: In [10] an algorithm is proposed in order to decrease the power consumption while maintaining the constraint of real time system and using technique of dynamic voltage scaling (DVS). Experimental results show that using a fair scheduling policy, the proposed algorithm provides, on average, energy savings ranging from 34% to 74%.

In the initial state, both cores are switched off, so consuming no power. When a task is ready to run, one core is switched on and starts working at its minimum speed. Then, if another task is ready, the scheduler estimates if that core satisfies the WCET of both of the tasks,(i.e., the running one and the incoming one), at the current speed. If this is not possible, the second core is also switched with the minimum speed. From this point, if we cannot guarantee the WCET of additional incoming tasks we increase the frequency up to the maximum one. As tasks finalize this process is reversed, reducing the frequency and shutting down the cores on an individual basis. The planned algorithm increases and decreases the voltage and frequency of both processors at the same time. While it is possible to use different voltages and frequencies in each processor (i.e. per-core DVS), this option is more complex and expensive, since it requires more voltage regulators and complicates the power delivery network.

Workload Balancing on Multi-Core: Load balancing is computer networking methodology to distribute work load across multiple computers or cores network links, central processing units, disk drive and other devices to achieve the optimal resource utilization, maximize throughput, minimize response time and avoid over load. In simple terms, load balancing is a method to spread tasks out over multiple resources. By processing tasks and guiding sessions on different servers, load balancing helps a network avoid annoying downtime and delivers optimal performance to users. There are virtual load balancing solutions that work in a manner similar to virtual applications or server environments. There are also physical load balancing hardware solutions that can be integrated with a network. The method used depends entirely upon the team implementing the solution and their particular needs. Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time and avoid overload. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. The load balancing service is usually provided by dedicated software or hardware, such as a multilayer switch or a domain name server.

Task Splitting: Task splitting is a process of dividing a task into two or more subtasks. We can perform task splitting when task do not fit on a processor or whenever we needed. In the task splitting process subtasks of any task cannot be executed in parallel on different cores or processors. Suppose we want to split a task τ into two subtasks τ' and τ''. First portion of τ is τ' and τ'' is the second portion of τ. The subtasks τ' and τ'' will be assign to different cores or different processors and they cannot be executed in parallel. Furthermore, both τ' and τ" must complete their execution within task's relative deadline [28]. In general when we split any task, we assume that after splitting a task the total size of the task or object will not change from the original size of the un-spilt task [28]. Whenever real-time tasks are considered for splitting, the penalty of splitting a task is not zero [31, 28].

Cycle-Conserving: Cycle-conserving algorithm utilizes the difference between actual execution time of a task and the worst case execution time (WCET) of a task. Generally actual execution time of a task is different from the worst case execution time [9]. Utilization of a task is updated in cycle-conserving algorithm as the actual execution time of a task divided by its period. Utilization of a task is updated at every release of a task and on completion of a task. Utilization of a task should be restored to its original value when the task starts its execution, to complete the execution within deadline. This algorithm is helpful to increase the energy efficiency when the actual execution times have difference from the WCETs [9].

EE Algorithm: Workload balancing algorithm is used for energy efficiency in multi core systems. When all tasks are completed their execution before than the particular deadline, there is a Chance for extra strike that raises the power savings. The system quickness for processing requests can be adjusted on run-time to decrease the cores. This run-time power scaling is called Dynamic Voltage Scaling (DVS). In multi core system DVS is responsible for power supply to each core which is an efficient technique for decreasing cores power consumption. This algorithm is divided into two parts first it will calculate the worst case execution time and then the second part is calculate the actual time.

```
Start task assignment algorithm:
    Beg ← 1
    Q task = tasks in ascending order of periods
    $U\ avg = \frac{\sum_{i=1}^{n} U}{m}$
    For I ← 1 to m – 1
    Do
    UC I ← 0
    For j ← Beg to n
    Do
    UC i ← UC i + U j
    If (UC i = = U avg)
    Do
    Exit
    End if
    If (UC i > U avg)
    Do
    Diff ← UC i – U avg
    C split ← S× P j
    V2 ← {(C split, P j), (C j – C split, p j)}
    Q task[j] ← (C j – C split, p j)
    Q task [j+1] ← (C split, p j)
    End if
    End for
    Core i < τ Beg to τ j
    Beg ← j + 1
    n ← n + 1
    End for
    Core m < τ Beg to τ n
    End
```

Fig. WCET & CCi

```
Upon task-completion (Qi)
Qi ← cci/pi
Task assignment (Qi)
```

```
When task-release (Qi)
Qi ← w i/pi
Task assignment (Qi)
```

Fig. Task Splitting Algorithm

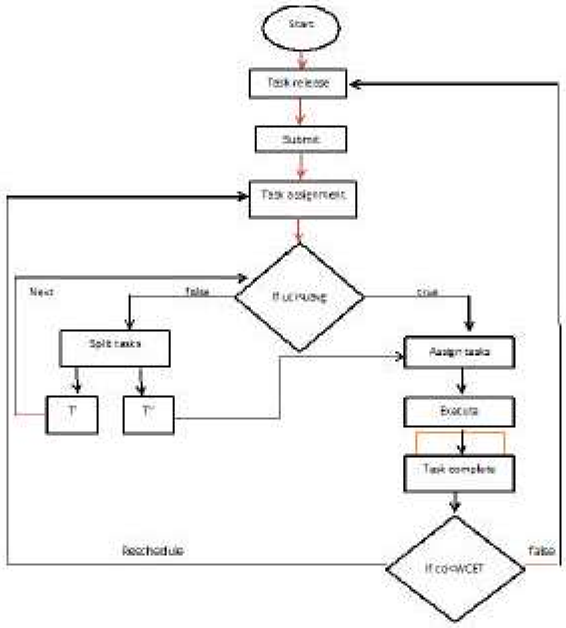


Fig. EE Algorithm Flow Chart

A lot of work on energy efficiency is related to the tasks scheduling algorithms. The concept behind the scheduling mechanism is to assign tasks and processes to processors based on their execution speed and other properties. There are two types of scheduling techniques. 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. 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 [10] the authors proposed DVS. They claim that 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 HPC systems. In recent processors the relationship between frequency f and power p gives foundation to Dynamic Voltage Scaling explained in equation.

$$E = Pt$$

where E is energy consumed, t is time taken and P is power consumed. We can achieve the low performance by simply reducing the operating frequency of the processor when the peak speed is not required. As a result DVS scales the operating voltage of the processor along with the frequency. DVS is a standard for managing the power consumption of a system.

Example 1:

Task Set T = { (2, 10) (1, 15) (4, 15) (5, 25) (3, 20) }
The given 5 are distributed among 3 cores. So
The cost of n = 5 and m = 3. 'Beg' is an integer variable with initial value 1.
Q task = { (2, 10) (4,15) (1,15) (3,20) (5,25) } tasks are pressed into array in ascending order according to the period of tasks. So we will calculate the sum of each core utilization and also calculate average utilization and then divide the tasks with number of cores.
Ui = 0.200000, 0.266666, 0.666666, 0.150000, 0.250000
U = 0.200000 + 0.266666 + 0.666666 + 0.150000 + 0.250000 = 0.883333
Uavg = (0.200000 + 0.266666 + 0.666666 + 0.150000 + 0.250000) / 3
Uavg = 0.294444
Ui = 0.100000, 0.200000, 0.666666, 0.100000, 0.160000
U = 0.100000 + 0.200000 + 0.666666 + 0.100000 + 0.160000 = 0.626667
Uavg = (0.100000 + 0.200000 + 0.666666 + 0.100000 + 0.160000) / 3
Uavg = 0.208889
Initially i = 1, J=1 and utilization of core UC1 = 0
J = 1
UC1 = 0 + U1
= 0 + 0.200000 = 0.200000 suppose task 1 is completed in 1 clock cycle instead of 2 clock cycle so the new task 1 is (1,10) so
Uc1 = 0 + U1
= 0 + 0.100000 = 0.100000
J = 2
UC1 = 0.100000 + U2
= 0.100000 + 0.266666 = 0.366666 suppose task 2 is completed in 3 clock cycle instead of 4 clock cycle so new task 2 is (3,15) so
Uc1 = 0.100000 + 0.200000 = 0.300000
So UC1 > U avg
Diff = 0.300000 – 0.208889 = 0.091111
Csplit = 0.091111 × 15 = 1.366667
V2 = { (1.366667, 15) (3 – 0.366667, 15) }
Qtask [2] = (1.366667, 15)
Qtask [2 + 1] = (1.633333, 15)
Qtask = { (1,10) (1.366667,15) (1.633333,15) (1,15)(3,20) (5,25) } after splitting τ2 new values pushed into the array Q task.
Core1 = τ1 to τ2
τ1 and τ2 is allocated to core 1.
Now Beg = 3, n = 6
Increase 1 in the cost of i and now i=2, J=3 and utilization of core 2 is
Primarily zero (UC2 = 0).
J = 3
UC2 = 0 + U3
= 0 + 0.091111 = 0.091111
J = 4
UC2 = 0.091111 + U4
= 0.091111 + 0.066666 = 0.157777 task 4 remain the same so task 4 is (1,15)
J = 5
UC2 = 0.157777 + U5
= 0.157777 + 0.150000 = 0.307777 suppose task 5 is

completed in 2 clock cycles instead of 3 clock cycles so the new task 5 is (2,20)

UC2 = 0.157777 + 0.100000 = 0.257777

As UC2 > U avg

Diff = 0.257777 – 0.208889 = 0.048889

Csplit = 0.0488889 × 20 = 0.977776

V2 = { (0.97776, 20) (2 – 0.977776, 20) }

Qtask [5] = (0.977776, 20)

Qtask [5 + 1] = (1.022222, 20)

Q task = { (1,10) (1.366667,15) (1.633333,15 (1,15) (0.977776,20) (1.022222,20) (5,25) } after

Splitting τ5 new values are pushed into the array Q task.

Core2 = τ3, τ4, τ5

τ3, τ4 and τ5 are allocated to core 2.

Suppose task 7 completed in 3 clock cycle instead of 4 clock cycle so the task 7 is (4,25)

Q task = { (1,10) (1.366667,15) (1.633333,15 (1,15) (0.977776,20) (1.022222,20) (4,25) }

Now Beg = 6, n = 7

Exterior for loop has been route m-1 times as shown in algorithm 1. It will be finished and all other tasks are assigned to the last processing core.

Core = τ6 and τ7

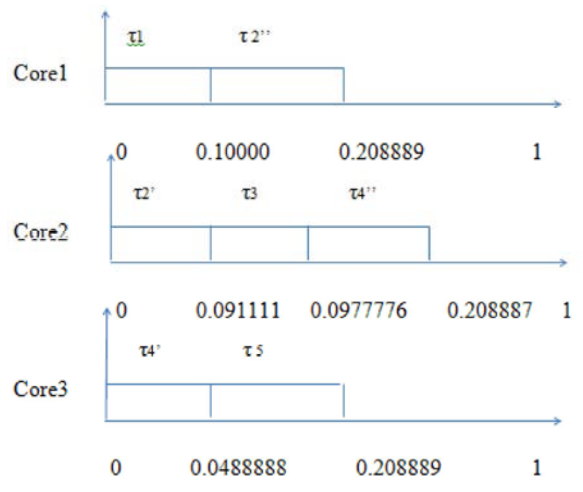
Final Results: At the end all tasks are allocated to each processing core equally according to the Uavg.

Core1 = { (2, 8) (1.366667, 15) } UC1 = 0.208889

Core2 = { (1.633333, 15) (1, 15) (0.977776, 20) } UC 2 = 0.208889 Core3 = { (1.022222, 20) (4, 25) } UC 3 = 0.208889.

Core 1 has 2 tasks, core 2 has 3 tasks and core 3 has 2 tasks but all cores are equal utilized

Example 1 is also verified through figure 4-1.



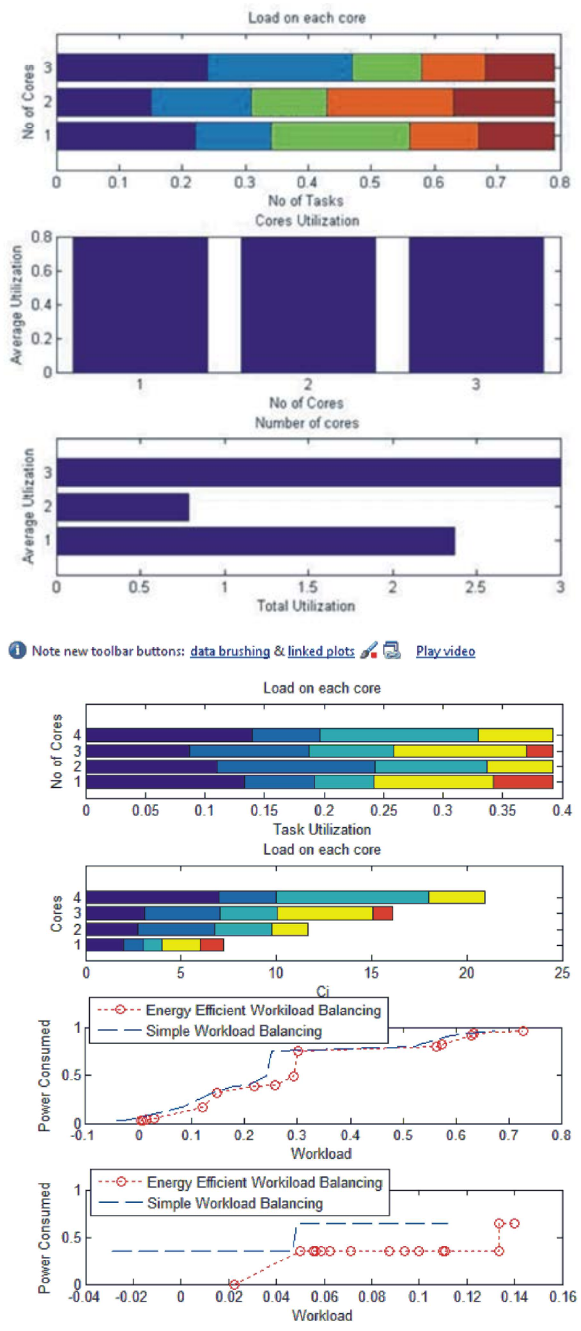


Fig. Giant chart

Example 2:

T = {(1,4)(3,10) (2,8)(1,4) (3,20)}

Q_{task} = {(1,4)(0.6,4) (0.4,4)(2,8)(0.5,10) (2.5,10)(3,20)}

$Core_1 \leftarrow$ {(1,4)(0.6,4)}

UC_1 = 0.40

$Core_2 \leftarrow$ {(0.4, 4) (2, 8) (0.5, 10)}

$UC_2 = 0.40$
$Core_3 \leftarrow \{(2.5, 10)\ (3, 20)\}$
$UC_3 = 0.40$

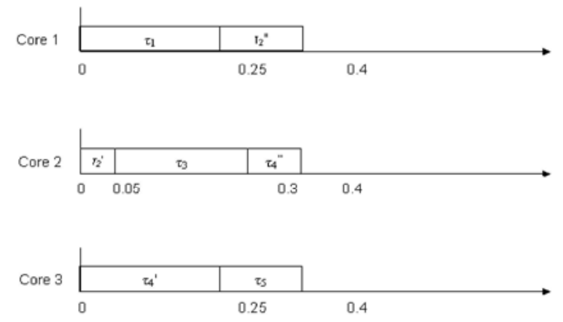
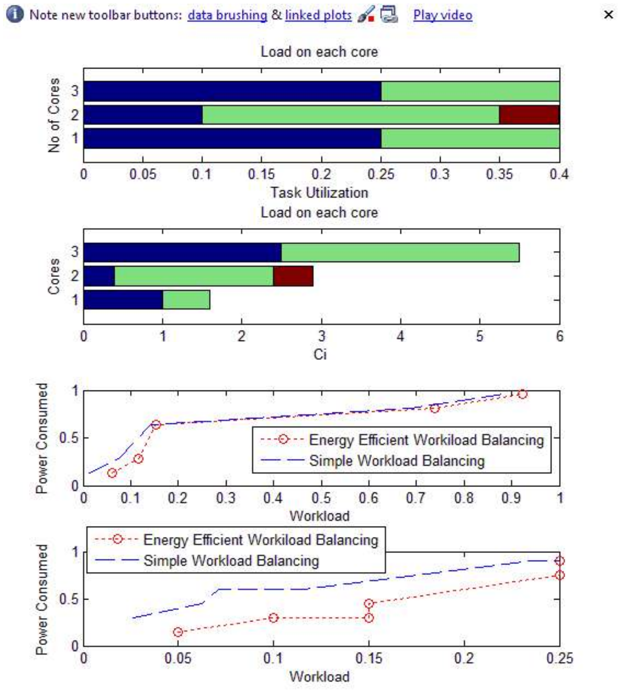


Fig. load on each core


Note new toolbar buttons: data brushing & linked plots Play video

Comparative Study: Seo *et al.* [9] presented a repartitioning algorithm with the aim to maintain L of all processing cores to have similar value every time by dynamically migrating tasks between cores. This can be done because the execution time of each task is less than the deadline. Dynamic utilization L_n can be defined as shown in following equation [9].

$$L_n = \sum_{\forall completed\,\tau_i} \frac{cc_i}{P_i} + \sum_{\forall incompleted\,\tau_i} \frac{W_i}{P_i}$$

Cores are denoted with n and cc_i is the last execution time of τ_i at that time. W_i is the worst case execution time of a task τ_i [9].

Basic idea of dynamic repartitioning of real-time scheduling algorithm is migrating tasks from one core to other core which have the low L. Cores are grouped into two groups: (i) donator and (ii) grantee group. Any one core cannot be from both groups. A core can only belong to one group. Initially partitioned tasks are assigned to donator group but then these tasks can be move able to grantee group. Grantee group have to execute those tasks which are shifted from the donator group and these tasks are not partitioned to grantee group.

By making two groups, tasks cannot be moved from any core to other core. Donator group can only move its tasks to the other core which belongs to grantee group. Donator group core can shift the task which has the lowest utilization in this group to the grantee group core with minimum utilization among that group. This process will be repeated until L of the destination exceeds L of the source. Whenever task is shifted from one core to other core, algorithm checks that L of the core does not exceed from 1. If it increases algorithm will restore the task to its source core where it initially partitioned.

In our proposed technique no group is made of cores or neither of tasks. Tasks are assigned to all processing core one by one. Core are considered similar and portion of splitted task is moved to the next core. In our proposed technique no core will offload its task completely and still it will maintain the core utilization equal to the average utilization.

Kato and Yamasaki presented a portioned scheduling scheme for the multiprocessors. This technique assigns the tasks to specific processor. A task can be split in two subtask and these two portioned are assign to different processor if any processor don't have the sufficient capacity. To achieve higher schedulability less pre-emption are required in this scheme [31].

This algorithm can successfully schedule a task set with system utilization much higher than 50 percent though the least upper bound is 50 percent.

In real time-time scheduling with task splitting on Mp, processor one is filled with tasks 100 percent and remaining processor are filled according to some specific value. Tasks are split able but these subtasks can be executed in any order if they are not corresponding to each other. But these tasks cannot be executed in parallel in any case. In our proposed technique splitted subtasks cannot execute in any order or in parallel. τ_1' will always has the highest priority then other tasks that are assigned to that core. In our case utilization of each core is equal to the average utilization but in real-time scheduling with task splitting processor, core one is filled up to 100 percent capacity.

Table 1: Comparative Study

	Proposed scheme	Partitioned fixed-priority preemptive scheduling	Real-time scheduling with task splitting on MP	Simple power-aware scheduling for multi-core systems
Workload on cores	100 percent Equal	88 percent	N/A	Never equal
Task migration	m-1	N/A	N/A	Until cores are balanced
Task splitting	Yes	Yes	Yes	No
Number of splitted tasks	m-1	m-1	N/A	No
Extra load on any core	No	N/A	N/A	Yes
Energy expenditure due to unbalanced workload	No	N/A	N/A	Yes
Splitted objects per core	At least one and at most two	At most one	N/A	N/A
Utilization bound per core	100% with EDF	60% with partitioning DM scheduling	N/A	N/A
Number of splitted portions	Two	N/A	Two	N/A
Execution order of splitted portions	Not in parallel and τ_1' before than τ_2''	N/A	Not in parallel but τ_1' and τ_2'' execute in any order	N/A

This biggest difference between our proposed scheme and fixed-priority preemptive scheduling is that we can have at least one and at most two splitted objects on same core but different objects. However in partition fixed-priority preemptive scheduling core can have at most one splitted object remaining objects moved on the last processing core [28]. In partitioned fixed-priority preemptive scheme, if a task on processor is need to be split then the highest priority task τ_h on the processor is selected for splitting. This is referred as the Highest Priority Task Splitting (HPTS) [28]. In our proposed scheme tasks are arranged in ascending order and these tasks are assigned to cores one by one. We only split that task is the reason of increasing the core's utilization more the average core utilization. That task has the lowest priority on that core. But after splitting portion one will get the highest priority o the next core. Rajkumar *et al.* [28] have also proved that 60% utilization bound per core can be achievable by partitioned deadline-monotonic scheduling. It is also possible to achieve utilization bound about 65% with light weight tasks.

Bautista *et al.* [32] proposed a simple power-aware real time scheduling for multi-core systems. This algorithm moves the complete task to those cores which are less load or they don't have any workload. This algorithm reduces the energy consumption while fulfilling the constraints of soft real-time application. Table 5-1 shows the difference between proposed technique and simple power-aware scheduling. As compare to our proposed technique simple power aware technique is never able to maintain the workload equal on cores because it tries to move the complete task to less loaded core so it is depend on the size of the task. It is online algorithm therefore whenever new task come, scheduler search for the less loaded core and when it find the less loaded core it assign the task to that core. Simple power-aware scheme do not split the task so it move the complete task to other core.

CONCLUSION

Due to increasing complexity of real-time applications, powerful processors are needed to accommodate computational starving applications. Multi-core processor is a solution to such applications. However, multi-core processor is still in its immaturity stage and there is need to address the multi-core partitioning issues with perspective that all cores are equally utilized.

In this literature we proposed a workload partitioning algorithm for multi-core systems. We have achieved workload partitioning with 100 percent precision. This algorithm is capable to distribute workload among 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. In proposed technique all the processing cores are considered to be homogeneous, so if the workload is distributed evenly, all cores will complete their work at the same time. We can state that this proposed scheme is fair scheme to distribute the workload on multi-core system and it is the finest algorithm to fully utilization of the available processing cores.

The efficiency of proposed algorithm is highlighted through the simulation and synthetic data is tested on the simulation. We found that proposed algorithm is enough efficient to distribute all kind of data set equally among all processing cores. Through this simulation we can also find out that provided data set to the simulation is schedulable with our technique or not after partitioning the data set on all the processing cores.

Proposed workload partitioning algorithm can be used for reducing the energy consumption in multi-core systems. This proposed algorithm can easily be used with cycle-conserving technique which can update the utilization of core dynamically on release and completion of a task. These tasks are capable to complete their execution earlier than it's provided worse case execution time. So we are able to propose repartitioning algorithm for multi-core systems to reduce the energy consumption.

REFERENCES

1. Diana Bautista, Julio Sahuquillo, Houcine Hassan, Salvador Petit and Jos´e Duato, 0000. A Simple Power-Aware Scheduling for Multicore Systems when Running Real-Time Applications, Department of Computer Engineering (DISCA)Universidad Polit´ecnica de Valencia, Spain.
2. Muhammad Zakarya and Izaz Ur Rahman, 0000. Towards Energy Efficient High Performance Computing Perceptions, Hurdles & Solutions, Technical Journal, University of Engineering and Technology Taxila,2011
3. Junyang Lu and Yao Guo, 0000. Energy-Aware Fixed-Priority Multi-core Scheduling for Real-Time Systems, 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.
4. Frederic Pinel, Jhonatan E. Pecero, Samee U. Khan and Pascal Bouvery, 2011. Energy-efficient scheduling on millicluster with performance constraints, 2011 IEEE/ACM International Conference on Green Computing and Communications.
5. Alexandra Fedorova, Mergo Seltzer and Micheal D. Smith, 0000. Cache-Fair Thread Scheduling for Multicore Processor.
6. Jian-ia Chen and Chin-Fu Kuo, 0000. Energy-Efficient Scheduling for Real-time Systems on Dynamic Voltage Scaling(DVS)platforms.
7. Xing Fu, Khairul Kabir and Xiaorui Wang, 0000. Cache-Aware Utikization Control for Energy efficiency in Multi-Core Real-Time Systems.
8. Wan Yeon Lee, 0000. Energy-effcient Scheduling of Periodic Real-timr Tasks on Lightly Loaded Multi-Core Processor.
9. Haisang Wu, E. and Douglas Jensen, 0000. The impact of Energy-Efficient Scheduler Overhead on the performance of Enbeded real-time Systems.
9. Amit Sinha and Ananth P. Chandrakasan, 0000. Energy Efficient Real-Time Scheduling.
10. Wei-Mei Chen, His-Yin Hung and Jhe-Ming Liang, 2011. Energy-efficient scheduling of periodic real-time tasks for reliable multi-core systems, 978-1-4244-8165-1/11/$26.00 2011 IEEE.
11. Santhi Baskran 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): April 2012.
12. Audsly, N., A. Burns, M. Richardson, K. Tindell and A.J. Wellings, 0000. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling.
13. Santh Baskaran and P. Thambiduraib, 2010. Energy Efficient real-Time Scheduling in Distributed Systems, IJCSI International Journal of computer Science Issue, 7(3): 4.
14. Jagbeer Singh, Bichitranda Patra and Satyendra Prasa Singh, 2011. An Algorithm to Reduce the Time complexity of Earlist Deadline First Scheduling Algorithm in Real-time System, IJCSI International Journal of Advance Computer Science Issue, 2(2): February 2011.
15. Jagbeer Singh, 0000. An Algorithm to Reduce the Time Complexity of Earlist Deadline First Scheduling Algorithm in Real-Time System.
16. Omid Amir Ghiasvand and Maziar Ahmad Sharbafi, 2011. Using Earlist Deadline First Algorithm for Coalition Formation in Dynamic Time-critical Envirnament, International Journal of Information and Education Technology, 1(2): June 2011.
17. Jorn B. Brandenburg, Jhon M. Calandrino and James H. Anderson, 0000. On the Scheduling of Real-Time Scheduling Algorithms on Multi-core platforms: A Case Study.
18. Claudio Scordino and Giuseppe Lipari, 0000. Using Resource Reservation for Power-Aware Scheduling.

19. Andreas Merkel, Frank bellosa, Memory-aware Scheduling for Energy Efficient on Multicore Processor.
20. Trevor Pering, Prof. Ropberi Brodersen, Energy Efficient Voltage Scheduling for Real-Time Operating System.
21. Nasro Min-Allah, Hameed Hussain, Samee Ullah Khan and Albert Y. Zomaya, 0000. Power efficient rate Monotonic Scheduling for Multi-core Systems.