Middle-East Journal of Scientific Research 24 (Techniques and Algorithms in Emerging Technologies): 236-241, 2016
ISSN 1990-9233;

DOI: 10.5829/idosi.mejsr.2016.24.TAET23532

Power-Aware Algorithms over Dual Processors for Real Time Periodic Applications

[1]B.M. Beena and [2]Dr. C.S.R. Prashanth

[1]Senior Assistant Professor, Department of CSE, NHCE, Bangalore,
Visvesvaraya Technological University, Belgaum,
[2] HOD and Dean of Academics, Department of CSE, NHCE, Bangalore,
Visvesvaraya Technological University, Belgaum,

Abstract: Consistently, dynamic voltage scaling (DVS) techniques have concentrated on diminishing the processor energy utilization instead of the whole framework power utilization. A two-fold processor framework comprises of two processors where the application tasks are executed utilizing Dynamic Voltage Scaling (DVS) to conserve energy. In our system, we utilize Earliest-Deadline-First (EDF) planning approaches on both the CPUs. We add to a couple of element calculations in light of these standards and assess their work execution provisionally. Our simulation results show compelling energy savings in contrast to existing Reliability-Aware energy management (RAPM) procedures for most execution situations. In Energy and Performance cognizant sharing scheduling algorithms over the two-fold processor for online applications that dynamically cut down the processor speed while still meeting timing constraints. The application tasks are run on both processors utilizing DVS to conserve energy.

Key words: Power Management · Real-Time Systems · Dynamic Voltage Scaling · Standby-Sparing · EDF Scheduling.

INTRODUCTION

In recent years, processor performance has increased at the expense of drastically increased power consumption. On the other hand, such increased power consumption decreases the lifetime of battery operated systems. Increased power consumption generates more heat, which causes heat dissipation to be a problem because expensive packaging and cooling technology are required which decreases the reliability [1]. Energy management is a frequent design concern for real time embedded systems.

In multiprocessor real time systems, Power management [2,3] adjusts the change in processor speed that has task execution time, which affects the scheduling of tasks on processors. This change may cause a violation of timing requirements. Here we present an energy-aware dual processor technique for Periodic Real-Time [1,4] applications that dynamically adjusts the speed while still meeting timing requirements. Dual processor technique combines hardware redundancy with Dynamic Voltage Scaling (DVS) [1,4] to save energy while preserving the system's original reliability [5] and offer opportunities to tolerate permanent processor faults. A Dual processor [6] system consists of a two processor. Where the application tasks are executed on both processor using Dynamic Voltage Scaling to save energy. Once the completion of a task is a success, the corresponding task is cancelled and excessive energy consumption is avoided. We employ Earliest-Deadline-First (EDF) [1,6] scheduling [7] policies on the both CPUs. Due to its dual-CPU structure, it can withstand the permanent fault of a single CPU. Also, since the backups execute with voltage scaling, the system's original reliability [5] (in terms of resilience with respect to transient faults) is preserved.

Dynamic Voltage Scaling (DVS) [5,4] technique reduces energy consumption, involving simultaneous scaling of the CPU supply voltage and frequency. The real-time execution semantics mandate that the timing constraints (the feasibility requirement) be met at run-time even when task response times may increase as a consequence of the energy management techniques. In addition to feasibility and energy management, reliability [6] and fault tolerance are other important objectives for real-time embedded systems.

Corresponding Author: B.M. Beena, Senior Assistant Professor, Department of CSE, NHCE, Bangalore,
Visvesvaraya Technological University, Belgaum.

Proposed Work: Bringing the system back to a previous safe state and repeating the execution is a common approach to deal with the transient faults. Recent study suggests that the transient fault rates increase significantly in systems where the supply voltage is cut down to conserve energy.

The main purpose of Dual Processor Technique is to minimize energy consumption, load sharing and to preserve reliability with respect to various faults in Real-Time embedded systems. Categorically, the technique employs two processors. The application tasks are executed on the both processor using DVS. The backup tasks also use DVS with little upper frequency.

Preamble: The Information Technology in general greatly contributes to global warming. Designers are considering environmental resource constraints along with more traditional IT business goals towards minimizing energy consumption, which has become an important objective for organizations [8]. Big data centres like Face book, Amazon and Google have got the most attention. In the past, the attention towards energy conservation was very less, eventually suffering the consequences.

In order to improve the system's performance it is obvious to reduce power consumption. Many adequate technologies with new competence are being designed as magical cures. Still, saving power is a much more complex architectural problem. The energy consumption is not only determined by the efficiency of the physical resources, but the resource management system deployed in the infrastructure and efficiency of applications running in the system.

Power management techniques are employed to conserve power across servers. In static Power management [7,3], we measure minimal processor speed at compile time to ensure that the tasks execute just-in-time by applying static scheduling [6] algorithms at compile time to optimize power consumption making use of static slack to delay the execution of tasks. In dynamic Power management, the processor supply voltage and speed adjusted together at run time. The tasks are then run with minimal voltage supply and speed to conserve energy using run time behaviour to reduce power when the system is idle or on light work load. Dynamic Power management [4] can be applied using Dynamic Voltage Scaling (DVS) [3,6] which can be implemented at the CPU-level or system-level to save energy. In DVS, the voltage supplied to a component is either increased or decreased depending upon the circumstances. Current processors [1] that use DVS typically have an operating voltage range from full to half of the maximum Vdd. Due to technology scaling; microprocessor performance has increased tremendously albeit at the cost of higher power consumption. Energy efficient operation has therefore become a very pressing issue, particularly in mobile applications which are battery operated.

Existing Systems: The present system works on a single processor. RAPM aims at reducing power consumption of tasks by using the available Slack [9] given by DVFS and maintain system's original reliability [5] by scheduling [3] a recovery job whose execution is scaled down. If a transient fault is detected during the execution of job J, then the recovery job will be dispatched at maximum frequency, f(max). The advantage of RAPM over DVFS is that it, RAPM maintains the system's original reliability [5].

The RAPM techniques run both the Primary and its corresponding backup task on the same processor. This technique eliminates the transient faults. It has two major drawbacks,

- Cannot offer performance greater than 50%
- Does not provide redundancy to tolerate permanent failures

In this system, pre-emptive scheduling [6] is used. A standby processor is added to withstand the permanent failure of single CPU. The scheduling [6] algorithm used are EDF (Earliest Deadline First) and EDL [10] (Earliest Deadline Late). On the Primary processor, the task is run using DVS [1,4] techniques whenever there is dynamic Slack [6], thus reducing power consumption. The scheduling algorithm on the Primary processor [5] used is (EDF) [11]. On the secondary processor, the backup task [6] is run at maximum frequency. This helps in retaining the system's original reliability [5]. EDL is used to schedule the backup tasks. Using this algorithm we delay the backup task as much as possible. Many backup tasks are cancelled even before they start executing. This happens when the corresponding Primary task completes executing successfully on the Primary processor before the backup [4] task starts on the secondary processor.

If the Primary task runs to completion successfully, the backup task is cancelled. Otherwise, the backup task runs to completion. The joint use of (EDF) [1] and EDL on Primary and secondary processors respectively, minimizes the overlap between the two copies at run time. It also helps us reduce energy costs due to backup [4] task executions. EDLschedule is computed offline at the pre-processing phase and the idle intervals are recorded. Intervals reserved in EDL schedule for the backup task are not considered during computation of available Slack for slow down.

Whenever a job arrives, its EDL is calculated and idle intervals are recorded. If the Primary processor is idle and a job arrives, it starts executing by obtaining the slowdown rate for execution. If the Primary processor is busy, the priority of the arrived job is considered and pre-empted accordingly. A job with an earlier deadline is given a higher priority. Once the task on the Primary processor runs to completion, a test is performed to check if the task isexecuted successfully. If yes, the backup task is cancelled. Else, the backup task runs to completion.

Proposed Sspt: Whenever a real time task arrives on the system, it is put into the real time task queue. The queue that holds the non real time tasks is called stored in the non real time queue. On the arrival of a task, the operating system checks if the task is a real time task or a non real time task and it places the task in the respective queue. In the proposed system, this step is skipped and the users will be prompted to enter the number of real time tasks. The real time tasks are taken as an input from the users and put into its respective queue. The system design consists of two processors. The two processors i.e. processors P1 and P2 operate on tasks from the real time queue. On the processors P1 and P2, the real time tasks are scheduled using the (EDF) scheduling [3] algorithm. On the processors P1 and P2, the real time tasks which execute at a scaled down frequency.

We consider a Dual processor system that consists of Two CPU. The each separate job Jij and Jkl runs on both the CPUs, which is assumed to have the DVS [6] capability, while the backup of jobs, denoted by Bij and Bkl, runs on the same CPU with voltage scaling (i.e. at the little maximum frequency than used in DVS). The frequency fi and fk of the CPUs is adjustable up to a maximum frequency of fimax and fkmax. We normalize all frequency values with respect to fimax and fkmax. We consider a set of Periodic Real-Time [7,10] tasks ¥ = {T1... Tm}. The real-time task sets are specified using a pair of numbers for each task, indicating its period and worst-case computation time. Each periodic task Ti has worst-case execution time ci under the maximum CPU frequency and the period pi.

The workload executes on a set of two cores. Each core can operate at one of the K different frequency settings ranging from a minimum frequency, fmin to fmax. We denote by F the set of available frequency settings. Without loss of generality, we normalize the frequency levels with respect to fmax (i.e., fmax = 1.0). At frequency f, a core may require up to ci/f time units to complete a job of task Ti.

The relative deadline of task Ti is assumed to be equal to its period. A job Jij and Jkl may take up to ci/f time units and ck/f times units respectively when executed at frequency fi and fk. Note that the backup [6] Bij and Bkl takes at more than ci/f time units and ck/f times units respectively since it is executed with voltage scaling of more than frequency used in DVS.

OUR WORK

START
/*Tasks arrives to the system */
Task Jk is released at time t:
Save_Task(list) /*list contains all the tasks released at that time*/
/*Start execution of Tasks when it arrives*/
If the primary processor is idle **then** Dispatch (Jk,t)
wi?? *ci /*Task executes at normalized frequency and Remaining execution time of*
task at fmax/*
Execute (Jj/Jk)/ if Ji was previously running else run Jk */*
Save_Backup(list)/ Save Jk task for backup execution if it fails in primary */*
***Else** /* Task Jj is running */*
If** (priority (Jk) > priority (Jj) **then
*wi?? ci /*Task executes at normalized frequency and Remaining execution time of*
task at fmax/*
*Execute (Jk) /*Jj priority should be rearranged using EDF */*
Save_Backup(list)/ Save Jk task for backup execution if it fails in primary */*
EDF (Jk,Jj)
Else
*wi?? ci /*Task executes at normalized frequency and Remaining execution time of*
task at fmax/*
Execute (Jj)
/ Check for completion of Task execution*/*
***If** (Jj completes)*
Remove_Backup(Jj)
Else
wi?? ci / Task executes at fmax frequency*/*
Execute_Backup(Jj)
/ After successfully completion of Task */*
If (Task list! =0)
*Dispatch (Jl, t1)/*Execution of task starts at time t1 */*
*wi?? ci /*Task executes at normalized frequency and Remaining execution time of*
task at fmax/*
*Execute (Jl) /*Task priority should be rearranged using EDF */*

Save_Backup(list)/ Save Jl task for backup execution if it fails in primary */*
Else *Wait ()/*Wait for next task arrival */*
END

It is assumed that the processor power consumption is dominated by the dynamic power dissipation Pd, which is given by Pd=Cef* Vdd2 * f (where Cef is the effective switching capacitance, Vdd is the supply voltage and f is the processor clock frequency.).Processor speed, represented by f, is almost linearly related to the supply voltage: f=k*(Vdd2-Vt)2/Vdd, where k is constant and Vt is the threshold voltage. The energy consumed by a specific task ëi can be given as Ei=Pd * AT', where AT is the actual time taken by a task at CPU. When we decrease processor speed, we also reduce the supply voltage. Thus, the processor power consumption is reduced cubically with f and the task energy consumption is reduced quadratically with the high expense of linearly decreasing speed and increase in execution time of the task. So it is referred to speed adjustment as both changing the processor supply voltage and frequency.

The following are the formulae and values used for calculating the energy:

- Pd = Cef x Vdd 2 x f

In this work, we focus on two efficient heuristics for Slack [6] distribution.

At run-time, whenever a job is released, it is distributed between two processor queue and it is dispatched immediately if the any processor is idle. Otherwise, a job is dispatched only if it has a higher priority than the currently executing job according to (EDF) [1] policy.

In run-time both processors will be executing assigned task. The processors speed will be changing according to the worst case execution time of each processor.

In case of pre-emption, the current job is pre-empted and we update the minimum additional time required to complete the job in the worst-case (under maximum frequency).

Every time a job is dispatched or resumed after pre-emption at time t, it can use the idle intervals between t and its deadline for slowdown. If no such idle interval exists, the job runs at set frequency on the both processor. At completion time of a Primary [4] job, we initiate the corresponding acceptance test. If no error is detected, it will execute the next waiting task immediately.

In case of an error, we can continue running the backup as scheduled with the little high frequency than usual.

The Dispatch procedure invokes a Set Speed procedure that takes into account the remaining execution time requirement of task Ti under maximum frequency (namely, wi) and available Slack [9] (denoted by the variable slack [9] in the algorithm) to determine the frequency assignment for the job.

The Set Speed procedure can use different heuristics to determine the exact amount of slack [10] to allocate to the job at dispatch time.

START
*/*Tasks arrives to the system */*
Task Jk and Jl are released at time t1 and t2:
*Save_Task_1 (list) /*list contains all the tasks which are going to execute on*
Processor 1/*
*Save_Task_2 (list) /*list contains all the tasks which are going to execute on*
Processor 2/*
*/*Start execution of Tasks at Processor 1 when it arrives*/*
If *the processor 1 is idle* ***then*** *Dispatch (Jk,t)*
*Wi*ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
*Execute (Jj/Jk)/*if Ji was previously running else run Jk*/*
Else */* Task Jj is running */*
If *(priority (Jk) > priority (Jj)* ***then***
Wi ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
*Execute (Jk) /*Jj priority should be rearranged using EDF */*
EDF (Jk,Jj)
Else
Wi ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
Execute (Jj)
*/*Start execution of Tasks at Processor 2 when it arrives*/*
If *the processor 2 is idle* ***then*** *Dispatch (Jk,t)*
Wi ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
Execute (Jj/Jk)/ if Ji was previously running else run Jk */*
Else */* Task Jj is running */*
If *(priority (Jk) > priority (Jj)* ***then***
Wi ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
*Execute (Jk) /*Jj priority should be rearranged using*

*EDF */*
EDF (Jk,Jj)
Else
Wi ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
Execute (Jj)
/ Checking for completion of Task execution in both Processor1 and Processor2*/*
***If** (Jj completes)*
Execute_NextTask ()
Else
*Wi*ci /* Task executes at fmax frequency*/*
Re-execute (Jj)
/ After successfully completion of Task execute next scheduled task in both Processor 1 and Processor parallel*/*
***If** (Task_List ! =0)*
*Dispatch (Jm, t2)/*Execution of task starts at time t1 */*
Wi ci /*Task executes at normalized frequency and Remaining execution time of task at fmax*/*
*Execute (Jm) /*Task priority should be rearranged using EDF */*
***Else** Wait ()/*Wait for next task arrival */*
END

Experimental Evaluation: We have developed a simulator to evaluate the potential energy savings from voltage scaling in a real-time scheduled system. The simulator takes as input a task set, specified with the period and computation requirements of each task, as well as several system parameters and provides the energy consumption of the system for each of the algorithms we have developed. The simulation assumes that a constant amount of energy is required for each cycle of operation at a given voltage.

The proposed system was implemented and tested extensively on all the possible scenarios.

Graph 1: Comparison of energy consumption in Existing Method and Proposed Method

The below snapshots shows the energy consumed by the Existing and Proposed methods for 10, 4, 9, 11 Tasks. And it also gives the percentage difference between energy consumption Existing and Proposed methods. For the existing method the Primary [4] processor runs all the tasks and if tasks fail then only spare processor will act. Hence the load on Primary [4] processor will be high which leads to degradation of the performance of the system. But in the proposed method the tasks will be shared with two processors which lead to good performance of the system.

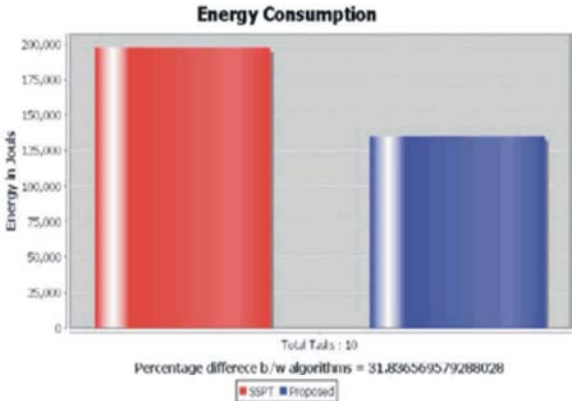

Fig. 7.5: (a) Energy consumed by Existing and Proposed method for 10 tasks. The comparison of proposed and existing algorithm is 38.84%

Units: x-axis corresponds to number of tasks and y-axis corresponds to energy consumption in Joule

Graph 2: Energy consumption in Existing Method and Proposed Method for different Task list sets. The energy consumption for each task depends on the arrival time of the task, execution time utilized by the task. If the tasks fail to execute at first attempt the deadline of the task also be accountable for the calculating or energy consumption value.

Power Consumption:

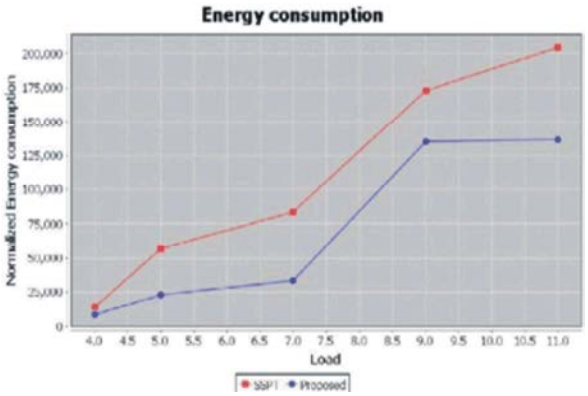

Fig. 7.6: Energy consumed by Existing and Proposed method for different Task list

CONCLUSION AND FUTURE ENHANCEMENT

Here explored a hardware redundancy technique for Periodic Real-Time [1,4] tasks based on Dual processor technique. The main contribution of this effort will be energy-efficient scheduling [3] algorithm for pre-emptive

Periodic Real-Time [1,4] tasks running on a Dual processor system. The framework uses the (EDF) [1] algorithm for scheduling [3] tasks on both processors. This allows executing the different tasks in different processors based on the deadline as the priority. An advantage of this framework is that often the re-execution of the tasks can be cancelled upon the early and successful completion of the tasks. Simulation results underline potential for energy savings compared to RAPM for most scenarios, while provisioning for permanent faults and still preserving the original reliability [5] in terms of tolerance to transient faults.

We have presented a first step towards runtime dynamic voltage scaling in Standby sparing Technique and we plan to improve further for real-time tasks. The proposed algorithm is particularly efficient for low-to-modest workload scenarios and we like to improve for heaviest work-load scenarios. Finally, we plan to extend our algorithm for real time, non-real time and multi processor systems.

REFERENCES

1. Energy-Aware cheduling (EAS) Project,By Amit Kucheria Posted January 27, 2015.
2. Energy-efficient task scheduling algorithms on heterogeneous computers with continuous and discrete speeds,Luna Mingyi Zhanga, Keqin Li, Dan Chia-Tien, Yanqing Zhang L.M. Zhang et al. / Sustainable Computing: Informatics and Systems, 3: 109-118.
3. Anton Beloglazov, Rajkumar Buyya, Young Choon Lee and Albert Zomaya, 2010. A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems.
4. Yifeng Guo and Dakai Zhu, 2013. Member, IEEE, Hakan Aydin, Member, IEEE and Laurence T. Yang Senior Member, IEEE "Energy-Efficient Scheduling of Primary/Backup Tasks in Multiprocessor Real-Time Systems (Extended Version)".
5. Mohammad A. Haque and Hakan Aydin Dakai Zhu, 2013. Energy-Aware Task Replication to Manage Reliability for Periodic Real-Time Applications on Multicore Platforms.
6. Mohammad A. Haque, HakanAydin and Dakai Zhu, 2011. Energy-Aware Standby-Sparing Technique for Periodic Real-Time Applications, Proc of IEEE, 2011
7. Jejurikar, R. and R. Gupta, 2004. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in Proceedings of the Low Power Electronics and Design, ISLPED.
8. Energy based Efficient Resource Scheduling: 2014. A Step Towards Green Computing, Sukhpal Singh and Inderveer Chana, International Journal of Energy, Information and Communications, l.5(2): 35-52.
9. Dakai Zhu and Hakan Aydin, 2007. Reliability-Aware Energy Management for Periodic Real- Time Tasks, Proc of IEEE,
10. Dakai Zhu, Rami Melhem and Bruce R. Childers, 2003. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack [10] Reclamation in Multiprocessor Real-Time Systems, Proc of IEEE.
11. Ramesh Mishra, NamrataRastogi, Dakai Zhu, Daniel Moss´e and Rami Melhem, 2003. Energy Aware Scheduling for Distributed Real-Time Systems, Proceedings of the International Parallel and Distributed Processingg Symposium.