Power Cognizant Algorithms Using Slack Reclamation Method

Beena B M ,
Senior Assistant Professor,
Department of Computer Science and Engineering,
New Horizon College of Engineering, Bangalore
beena.nh@gmail.com

Dr Prashanth C.S.R,
Professor and HOD,
Department of Computer Science and Engineering,
New Horizon College of Engineering, Bangalore
drprashanth@gmail.com

***Abstract*--In this paper, we present few energy-cognizant algorithms using slack reclamation method. In the present design, the Earliest-Deadline-First (EDF) scheduling policies have been employed on the both CPUs for the dynamic tasks and FIFO scheduling for static tasks. When a new task arrives and the primary processor is active then that task will be scheduled in secondary if that processor is inactive else based on priority context switching will happen. The proposed System shows significant energy saving compared to existing SSPT algorithms.**

***Keywords*: Dynamic Voltage Scaling, Standby-Sparing, EDF Scheduling, Slack, faults, Power Management.**

I. INTRODUCTION

Excessive power consumption in a microprocessor has several negative impacts: a decrease in system reliability, an increase in both the initial cost and the operation expenses of computers, and a reduction in the battery life of power-critical portable and handheld systems. With the ever increasing power density in modern computing systems energy is the first class system resource. There are a number of techniques to reduce power consumption, but this project focuses on dynamic voltage scaling (DVS) [1]. Dynamic voltage scaling is a technique to reduce the power consumption of a microprocessor by lowering the CPU. Reliability and fault tolerance are the other important concerns for the Real time embedded systems. There are several reasons for this new interest in DVS. First, portable and handheld systems (e.g. PDAs, cellular phones) have limited battery usage, so power saving will increase battery life. Second, a reduction in power consumption leads to increased reliability in microprocessors.

Higher power devices dissipate more heat. Excessive heat can permanently damage a chip, so lowering a processor's power reduces the likelihood of failure. Third, reducing power can reduce cost. Processor designers build their cooling systems for peak thermal characteristics that are far more stringent than the average thermal behavior, though the peak value is almost never reached. Power management [2] techniques can be used to indirectly regulate the thermal characteristics such that the peak thermal value is much closer to the average value, thus allowing designers to employ less costly cooling systems at the CPU level.

In Energy-Aware Standby-Sparing Technique for Periodic Real-Time Application, the application tasks are executed on the primary processor using DVS [3]. The spare does not use DVS and is reserved for executing backup tasks, if needed. Upon the successful completion of a primary task, the corresponding backup task is cancelled and excessive energy consumption is avoided. However, if the primary task fails, the backup runs to completion at the maximum frequency. Hence, the solution statically constructs the schedule on the spare CPU to delay the backups as much as possible to minimize the overlap between the primary and backup copies of the same task.

II. APPLICATION MODEL

In the present study, a parallel application is considered consisting of n real-time tasks with precedence constraints, represented by a directed acyclic graph (DAG) G = (V,E) [4] as shown in fig.1. The set of vertices (nodes) V = $\{T_1,...,T_n\}$ represents the set of tasks. The set of edges E = $\{E_1,...,E_m\}$ represents a partial order corresponding to the precedence constraints among tasks, with the interpretation that whenever the edge $(T_i,T_j) \in E$, the task T_j cannot start to execute until T_i has been completed. Hence, the ready time of a task is defined as the time instant at which all its predecessors have been completed. The worst-case execution time (WCET) of task T_i under the maximum normalized processing frequency $f_{max} = 1$ is denoted by c_i $(1 \leq i \leq n)$. When T_i is executed at a lowerfrequency f_i, its execution time is assumed to be ci/fi in the worst case. All tasks in the application need to complete by the deadline D, which is also the period (or frame) of the application. An example application with six tasks is shown in Figure 1, where the WCET of each task is labeled accordingly and the deadline is assumed to be D = 313.

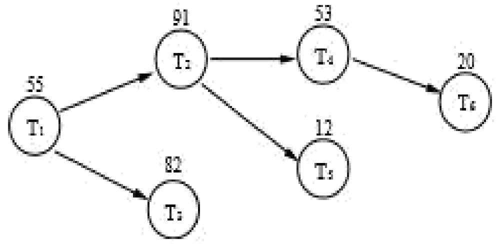

Figure1: DAG for the example application.

The application will be executed on a shared memory multiprocessor system with 2 identical processors with negligible communication overhead. In multiprocessor settings, dependent tasks are typically scheduled by the list scheduling technique. Therefore, when tasks have the same predecessors and become ready simultaneously, we use the Early DeadLine first (EDF) [2] heuristic to determine their execution orders. In this paper, we assume that an application's canonical executions under list scheduling with EDF can finish no later than its deadline.

111. TASK MODEL

We consider a standby-sparing system that consists of a primary CPU and a spare CPU. The main copy of each job J_{ij} runs on the primary CPU, which is assumed to have the DVS capability, while the backup copy, denoted by B_{ij} , runs on the spare CPU without any voltage scaling (i.e. at the maximum frequency). And if any high priority task arrives that will be scheduled in idle processor. The frequency f of the primary CPU is adjustable up to a maximum frequency f_{max}. We normalize all frequency values with respect to f_{max}, weconsider a set of periodic real-time tasks = $\{T_1... T_m\}$. 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 T_i has worst-case execution time c_i under the maximum CPU frequency, and the period p_i. The relative deadline of task T_i is assumed to be equal to its period. A job J_{ij} may take up to c_i/f time units when executed at frequency f. That is backup B_{ij} takes at most c_i time units since it is executed without voltage scaling. The task periods are generated randomly.[1]

1V. POWER MODEL

The processors in the system are assumed to have DVS capability, which is a common feature in modern processors. Each processor has the capability of operating in three different power modes. The tasks are executed in the active state of the processor. When the processor is not executing tasks, it can be in idle or sleep states. We now describe the power characteristics of each of these states.

A. Active

We model the power consumption in the active mode. The power consumption of the system consists of static and dynamic power components. The static power P_s is dominated by the leakage current of the system. The dynamic power P_d includes a frequency- independent power component P_{ind} driven by the modules such as memory and I/O subsystem in the active state, and a frequency-dependent power component which depends on the supply voltage and frequency of the system.

$$P_{active} = P_s + P_{ind} + C_e V_{dd}^2 f$$

Above, C_e denotes the effective switching capacitance. The processor supply voltage V_{dd} has a linear relation- ship with frequency f. Therefore, Equation 4.1 can be re-written as,

$$P_{active} = P_s + P_{ind} + C_e f^3$$

When the voltage/frequency scaling is applied through the DVS technique, the processor frequency can be adjusted within a range between a minimum CPU frequency f_{min} and a maximum CPU frequency f_{max}.We assume that the processors can only operate at L different discrete frequency levels $\{f_1, f2...f_L\}$, where $f_1 = f_{min}$ and $f_L = f_{max} = 1.0$ are the minimum and maximum frequency, respectively. The time overhead for frequency changes is assumed to be incorporated into the tasks' WCETs.

All frequency values in the project are normalized with respect to f_{max}. The existence of P_s and P_{ind} implies the existence of a threshold frequency, called critical speed or energy - efficient frequency, below which DVS ceases to be effective.

B. Idle

The processor can switch to idle power state when it is not executing any task. In this state, the processor consumes low dynamic power, P_0. The overhead for transitioning to idle state and back is not significant. Hence, the power consumption in idle state is given by:

$$P_{idle} = P_s + P_0$$

C. Sleep

Sleep state is the lowest power state for the processor. In this state, power components other than the static power P_s become negligible. We would like to put the processor to sleep state whenever the system is idle. Since putting a processor to sleep state involves significant time and energy overhead. The concept of break- even time (Δ_{crit}) is used. If the processor is put to sleep state for at least as long as Δ_{crit} time, the energy savings in the sleep state can amortize the transition overhead. In DPM scheme, when the idle interval is expected to exceed Δ_{crit}, the processor is transitioned to sleep state for energy savings. Therefore, it is beneficial to have longer idle intervals to take advantage of the DPM technique. If the idle interval is expected to be relatively short, the processor switches to idle state instead.

V. EXISTING SYSTEM

The application tasks are executed on the primary processor using DVS [3]. The spare does not use DVS and is reserved for executing backup tasks, if needed. Upon the successful completion of a primary task, the corresponding backup task is cancelled and excessive energy consumption is avoided. However, if the primary task fails, the backup runs to completion at the maximum frequency. Hence, the solution statically constructs the schedule on the spare CPU to delay the backups as much as possible to minimize the overlap between the primary and backup copies of the same task.As shown in Figure 2 the technique uses two processors, with the potential of tolerating the permanent fault of a single CPU. On the primary processor, the task is run using DVS techniques whenever there is dynamic slack, thus reducing power consumption. The scheduling algorithm on the primary processor used is EDF [1].

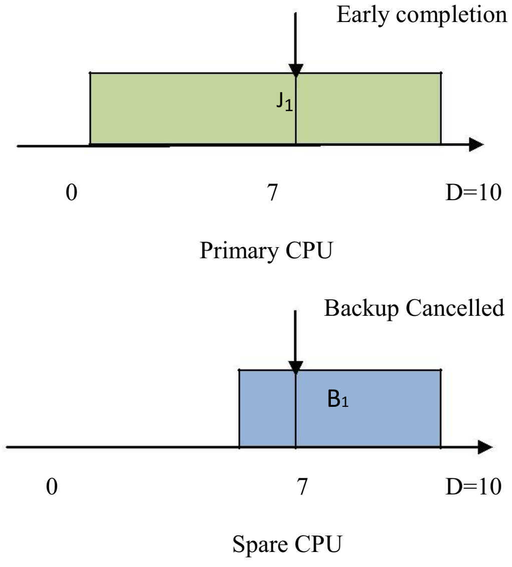

Fig. 2: Standby-Sparing System for a Single Task.

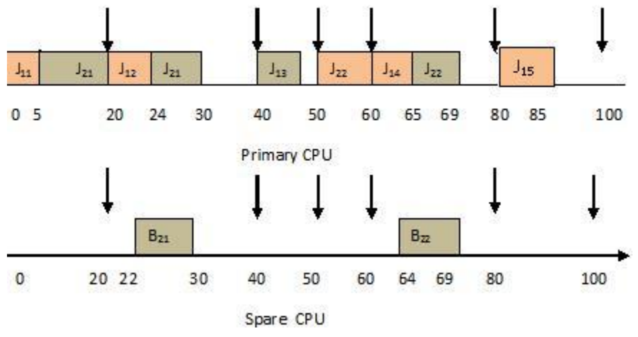

Fig.3: Fault-free execution in SSPT

On the secondary processor, the backup task 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 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 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 task executions. EDL schedule is computed offline at the pre-processing phase and the idle intervals are recorded. Intervals reserved in EDL schedule [6] 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 is executed successfully.

There are two variants of SSPT (Standby Sparing for periodic tasks) used:

A. Aggressive SSPT

A job can utilize the entire slack and slow down as much as possible. If the job completes before the deadline, the slack can be used for upcoming tasks. [1]

B. Conservative SSPT

A task is not allowed to run below its average case utilization, this approach has a balanced slack distribution among all jobs. In this system RAPM, aggressive SSPT, conservative SSPT are compared. Conservative SSPT has said to have shown superior performance with energy gains up-to 15% [1].

C. Disadvantages of existing system

- Most of the time, spare CPU is idle. Wasting energy.
- When a new task with high priority comes it goes for context switching, which needs energy .Same time spare CPU may be idle consuming energy.
- Spare CPU may remain idle for very long time.
- Does context switching in spare CPU also

V1. PROPOSED SOLUTION

Whenever a task arrives on the system, they are put into the task queue. In the proposed system the same set of tasks will be considered for both static and dynamic scheduling.

User will be prompted to enter the number of tasks required and it gives an option either to input task related values or the system will generate randomly. The system will sort all the tasks based on its arrival time and deadline and will be giving id to the tasks. System also gives a chance to test the execution after getting an error in static scheduler.

On the arrival of a task, the operating system will schedule its backup if idle period is available and start scheduling according to the proposed algorithms.

The system design consists of three processors. The first two processors of the three processors, i.e. processors P1 and P2 operate on real time tasks. And the third processor i.e. P3 operates non real time task. On the processor P1, the real time tasks are scheduled using the EDF scheduling algorithm. On the processor P2, the real time tasks which execute at a scaled down frequency on the processor P1 are scheduled using the EDL scheduling algorithm. On third processor the non-real time tasks are scheduled using the FCFS.

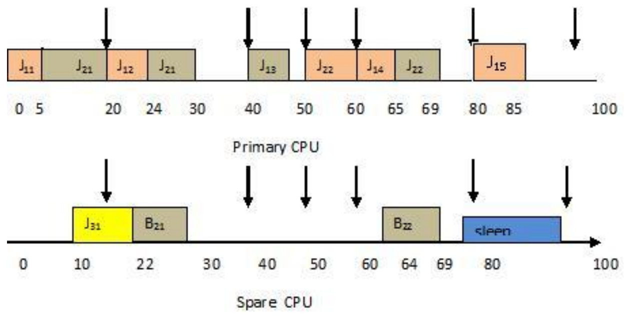

Fig.4: Scheduling of high priority in backup

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

- $P_d = C_{ef} \times V_{dd}^2 \times f$

  Where P_d is the dynamic power dissipation,

  C_{ef} is the effective switching capacitance,

  V_{dd} is the supply voltage,

  f is the frequency

- $E_i = C_{ef} \times V_{dd}^2 \times c_i$

  E_i is the energy consumed by task i

  c_i is the number of cycles taken to execute the task.

- Maximum frequency = 1 GHz
- Threshold frequency = 0.25 * maximum frequency

Algorithm to avoid context Switching

```
Task J_k is released at time t:

Update_alpha(list) /*list contains all the tasks
released at that time*/

w_i c_i /*Remaining execution time of task k at f_max */

If the primary processor is idle then Dispatch(J_k , t)

else/* Task J_j is running */

if(priority (J_k ) >priority (J_j) then

/* case to avoid context switching*/

If secondary processor is idle thensecondary.
Dispatch(J_k , t).

Cancel all the backups scheduled to start during this
time period and cancel B_k .

/* to cancel one executing task */

Else if both processor has same taskJ_j running. Cancel
the task which needs more energy.

then dispatch(J_k ,t) in that processor.

/*context switch*/

$w_k \leftarrow w_k - (\Gamma_k \times f_k)$

/* Γ_k : execution time of J_k in current dispatch */

Dispatch(J_k ,t)

End
```

Algorithm to avoid running Backup

```
J_j completes at time t:

Run the acceptance test if no error is detected then Cancel

the backup B_j on secondary processor. Releases the time

allocated to idle time available.

End if
```

Algorithm for sleep

```
$J_j$ will be released only after $_p$.
/*$\Delta_p$. time for next arrival*/
/* $_{crit}$ is the predefine time duration for which processor
can go for sleep without loss of energy*/
If $_p \geq \Delta_{crit}$ then
    if secondary-processor is idle during this time then
    processor can go for sleep for $_p$ units of time and wake up
    when next process arrives.
End if
```

Algorithm to drop from Processor *bj_j*

```
completesin secondary.
if $j_j$ is present in primary list then remove from the list.
If $j_j$ is the next task to run in primary then drop that.
If $j_j$ is the active task to in primary then cancel that
```

Algorithm to Schedule Static Tasks

```
If $T_k$ arrived first
If Primary Processor idle then Dispatch ($J_k$ ,t)
Else next task will be Tk.
Secondary. Dispatch($J_j$,t)
End if
```

```
Function Dispatch($J_j$, t)
    Slack ← $EDL_\psi$ (t,deadline ($J_j$))
    $f_i$ ← Set_Speed($w_i$, slack)
    Dispatch $J_j$ on the primary processor at frequency $f_i$
End function
Function Update_alpha (list)
    Schedules the backups ofall the tasks in the list.
    Drops backup if it wouldn't meet deadline.
    $EDL_\psi$ ($t_1$,$t_2$) gives the total idle time available between $t_1$ and $t_2$ .
End function
```

V11. EXPERIMENTAL EVALUATION

In this section, we present experimental results to demonstrate the performance of energy cognizant algorithms by stack reclamation method. Our evaluation involves comparison of the three systems one without any power management methods , second one the existing system and the third one is the proposed system.

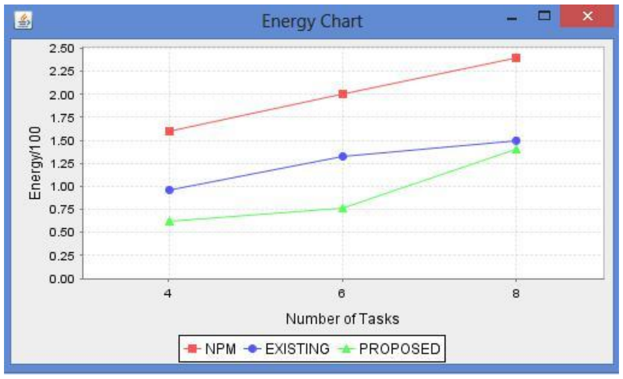

Figure 5: Energy consumed based on different load

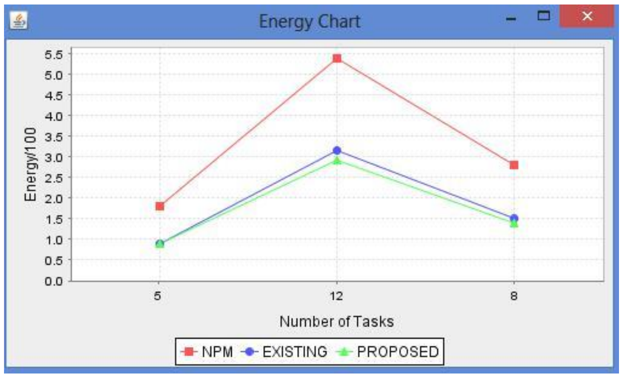

Figure 6: Energy consumed based on different load

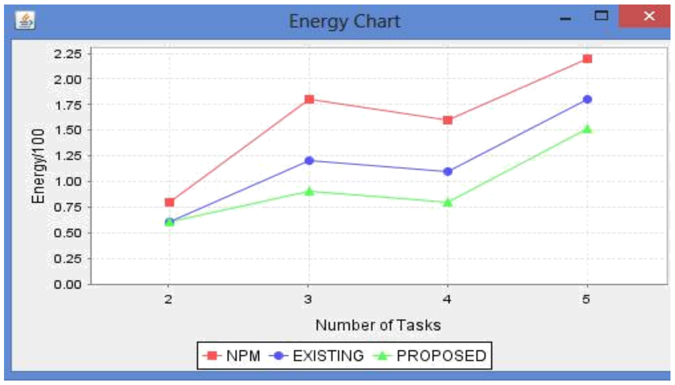

Figure 7 : Energy consumed based on different load

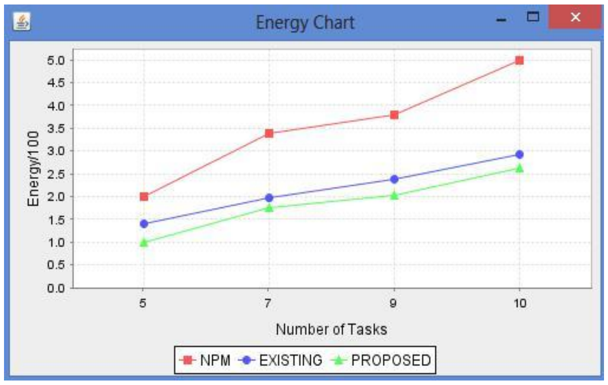

Figure 8: Energy consumed based on different load

V111. CONCLUSION AND FUTURE ENHANCEMENT

Due to the following reasons, the proposed system is better than the existing systems:

- Better scheduling
- Handles non real time tasks
- Very Less context switching
- More energy conservation in specific cases
- Provides more hardware reliability

The proposed system conserves approximately up to10% power when compared to the existing system, i.e. the standby sparing technique.

The proposed system finds its applications in,

- Operating System design
- Servers
- Battery operated devices

Proposed system can work with threshold energy so that there is no constrain on time, enhancement for the proposed system can be done by adding a constraint on time also.

REFERENCES:

[1]. Mohammad A. Haque, Hakan Aydin and Dakai Zhu, “Energy-Aware Standby-Sparing Technique for Periodic Real-Time Applications”, Proc. of IEEE 29th International Conference on Computer Deisgn, 2011, pp.190-197.

[2]. Keqin Li,”Energy Efficient Scheduling of parallel tasks on multiprocessor computers”, Proc. of IEEE International Symposium on Parallel and Distributed Processing, 2011, pp.804-813

[3]. Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya, “A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems “, IEEE Transactions on Computers, vol. 58, No.10, 2009, pp. 1382-1397.

[4].Yifeng Guo, Dakai Zhu, Hakan Aydin, and Laurence T. Yang **“**Energy-Efficient Scheduling of Primary/Backup Tasks in Multiprocessor Real-Time Systems (Extended Version)”, Proc. of IEEE International Symposium on Parallel and Distributed Processing, 2013, pp.896-902.

[5] Mohammad A. Haque, Hakan Aydin Dakai Zhu.”Energy Management of Standby-Sparing Systems for Fixed-Priority Real-Time Workloads”, Handbook of Energy-Aware and Green Computing, CRC Press, 2012, pp. 763-769

[6]. Mohammad A. Haque, Hakan Aydin Dakai Zhu, “Energy-Aware Task Replication to Manage Reliability for Periodic Real-Time Applications on Multicore Platforms”, IEEE/ACM International Conference on Cyber, Physical and Social Computing (CPSCom) 2013, pp. 117-124.