Dynamic Repartitioning of Real-Time Schedule on a Multicore Processor for Energy Efficiency*

Euiseong Seo[1], Yongbon Koo[2], and Joonwon Lee[1]

[1] Dept. of CS, Korea Advanced Institute of Science and Technology
[2] Electronics and Telecommunications Research Institute
ses@calab.kaist.ac.kr ybkoo@etri.re.kr joon@kaist.ac.kr

Abstract. Multicore processors promise higher throughput at lower power consumption than single core processors. Thus in the near future they will be widely used in hard real-time systems as the performance requirements are increasing. Though DVS may reduce power consumption for hard real time applications on single core processors, it introduces a new implication for multicore systems since all the cores in a chip should run at the same performance. Blind adoption of existing DVS algorithms may result in waste of energy since a core which requires low performance should run at the same high frequency with other cores. Based on the existing partitioning algorithms for the multiprocessor hard real-time scheduling, this article presents dynamic task repartitioning algorithm that balances task loads among cores to avoid the phenomena dynamically during execution. Simulation results show that in general cases our scheme makes additional energy saving more than 10% than that without our scheme even when the schedules are generated by WFD partitioning algorithm which is known as the best energy efficient partitioning algorithm.

1 Introduction

The use of real-time systems are getting wider and the target applications are getting more complex. Thus more powerful processors are demanded for real-time systems. To improve processor performance the processor vendors have competed to raise the operating frequencies of their processors. However the power consumption of a processor is increased proportionally to the cubic of its operating frequency f. Hence the power consumption of processors have been increased dramatically. However The concern for energy efficiency has been increased as the use of mobile equipments grows.

Multicore architecture[1] which integrates a few processor cores in a single chip draws attention because it provides more throughput without increasing f.

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Inofrmation Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment) (IITA-2005-C1090-0502-0031)

Similar to symmetric multiprocessors, multicore systems achieve linear speedup because each core has independent processing element and cache. Thus the throughput increase of a multicore processor is expected to be linear with respect to the increase of power consumption.

DVS(dynamic voltage scaling) is an another feature that changes V_{dd} and f of a processor during its operation. Many of multicore processors are to be expected to employ DVS. Current multicore technology offers a DVS feature that allows only the same frequency for all cores because individual voltage regulator for each core costs too much in both design and production. This limitation is expected to remain for foreseeable future[2].

Then the energy efficient scheduling of real-time tasks in this environment emerges. This problem is similar to that of the energy efficient real-time scheduling on multiprocessors of having DVS function. It is known as a NP-hard problem that scheduling hard real-time tasks for multiprocessors[3]. Thus many heuristic algorithms are introduced to solve this. Among many heuristics, partitioning scheduling [4–6] is one of the representative schemes which statically distributes tasks onto each processor. By partitioning scheduling the problem is transformed into single processor real-time scheduling problems which can be solved by using existing real-time schedulers like EDF(earliest deadline first)[5] or RM(rate-monotonic)[6].

Adding energy efficiency to the real time task schedule for a multiprocessor system has been a challenging problem. Most approaches depend on the idea[4, 7] of using existing single processor DVS algorithms[8–10] in each processor that has its own task set which is given by a certain partitioning algorithm. Applying those approaches to a multicore processor underperforms due to the aforementioned limitation of using DVS in multicore processor. Hence this paper suggests a dynamic repartitioning algorithm to reduce the difference of demanded performance among cores.

The rest of this paper is organized as follows. Section 2 reviews existing related work especially on the DVS algorithm for the real-time schedule on a processor. Section 3 describes the dynamic repartitioning problem and a heuristic algorithm for it. Section 4 presents the simulation results of the suggested algorithm. And section 5 summarizes our conclusions.

2 Related Work

EDF is an optimal algorithm in scheduling periodic real-time tasks on a processor. The utilization of a task is defined as its WCET(worst case execution time) divided by its period and the utilization of a task set is the sum of the utilizations of tasks in it. By using EDF it is guaranteed that the task set which have the processor utilization under 1.0 is always feasible to schedule. The decrease of the processor performance will increase WCET of each task. Thus lowering processor performance will increase the utilization of the task set too. Due to the property of EDF we can lower processor performance till the utilization of the task set becomes 1 while the dead-lines are still kept. Based on this concept Pillai et.

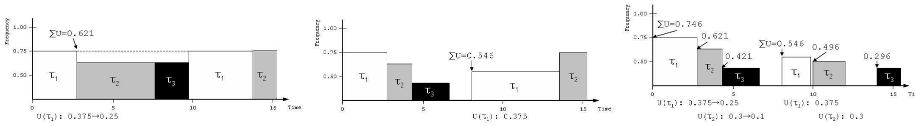
al.[10] suggested three heuristics, *static, cycle-conserving* and *look-ahead*, which adjust processor performance for the real-time schedules that were generated by EDF or RM algorithms.

Table 1. Example task set

Task	Period	WCET	Load	1st Exec. Time	2nd Exec. TIme
τ_1	8 ms	3 ms	0.375	2 ms	1 ms
τ_2	10 ms	3 ms	0.300	1 ms	1 ms
τ_3	14 ms	1 ms	0.071	1 ms	1 ms

static adjust performance to make the utilization of the task set to 1 and stay in the decided performance level all the time. For example the utilization of the task set described in Table 1 is 0.746 and let the maximum frequency of the processor is 1.0. The lowered frequency of 0.746 make the utilization of the task set 1.

In general the actual execution time of a task shows much difference from the WCET of the task. The early completion of a task makes room for more energy saving. For example let there be a task set described in Table 1, there occur considerable amount of idle time with *static* algorithm. To exploit these idle periods from the early completions of the tasks, when a task is completed *Cycle-conserving* algorithm updates the utilization of the task as the actual execution time divided by its period. Thus after the updates the performance will be lowered according to the updated utilization and the updated value will be used till the next release of that task. After the next release of the task, the utilization of the task should be restored to its initial value to keep the deadline. This algorithm improves energy efficiency much in case that the actual execution times tend to be much different from the WCETs. Figure 1 shows using *cycle-conserving* algorithm for the example task set in 1.



(a) After end of τ_1 (b) After end of τ_2 and τ_3 (c) Actual execution flow

Fig. 1. Cycle-conserving scheduling of example task set[10]

Because *cycle-conserving* algorithm dynamically utilizes the idle period from the early completion of the tasks, it saves more energy than *static* algorithm. However as depicted in Figure 1 (c) the tasks are executed at high performance at the starting and the performance had decreased as time flowed and this made idle time again. If most of the tasks are finished in much less than WCET, it is more effective to execute a task with lowered performance at the starting time and to execute the task with raised performance after certain threshold to keep the dead-line. *Look-ahead* algorithm is based on this idea.

In addition to this research, many DVS algorithms on periodic hard real-time systems are suggested. Aydin et. al.[8] suggested GDRA(generic dynamic reclaiming algorithm) and AGR(aggressive speed adjustment) algorithms to exploit the situation that the actual execution times are less than WCETs. The basic concept of GDRA is similar to *cycle-conserving* and AGR tries performance adjusting based on the execution history of tasks. The algorithm suggested by Gruian[9] starts to execute a task at low performance and based on the probability distribution of the execution time, it gradually raises the performance as the execution of the task goes on.

For the DVS scheduling on multiprocessor environment the basic approach is that adopting the existing single processor DVS algorithms onto the results from the partitioning algorithms. Aydin et. al.[4] also evaluated several partitioning heuristics for the energy efficiency and the result showed that Worst-Fit-Decreasing is the best energy efficient partitioning algorithm.

Yang et. al.[2] suggested an energy efficient static scheduling algorithm of hard real-time tasks on the multicore processors including DVS which is the same assumption in this paper. However it is based on the assumption that all the tasks have same periods and the execution time will be always same.

3 DVS Scheduling on a Multicore Processor

3.1 Dynamic Repartitioning Problem

The aim of this work is devising an algorithm that reduces the waste of energy due to the difference of performance demand among cores for executing periodic real-time tasks on a multicore processor in which cores have same V_{dd}. The real-time scheduling is assumed to be done by the partitioning approach. And our algorithm will work on the resulting schedule dynamically during the execution.

The periodic task set $\mathcal{T}$ which is executed in the assumed environment is defined as Equation 1. The Period of task τ_i is represented as P_i and W_i means the WCET of τ_i. u_i which is the task utilization of τ_i is defined as W_i/P_i.

$$\mathcal{T} = \{\tau_1(P_1, W_1),\ \ldots,\ \tau_n(P_n, W_n)\} \tag{1}$$

The dead-lines of each task are assumed to be same as their periods. Tasks have no dependency among them. The preemption and the migration of the tasks among cores are possible. The cost for the preemption and migration is assumed to be free because it can be considered in the actual implementation stage.

A multicore processor is defined as Equation 2. Processor S have m cores C in it.

$$\mathcal{S} = \{C_1, \ldots, C_m\} \tag{2}$$

Each core has a dedicated partitioned task set $\mathcal{T}_m$. The utilization sum U_m of $\mathcal{T}_m$ belonged to a certain core m does not exceed 1.0. To be simple if cores are operated at a relative performance p and the power consumption in a core is $g(p)$, at a certain point the power consumption in the processor is $mg(max(U_1, ..., U_m))$. In real the cores in the idle period from unexpected early completion consume only leakage power at the corresponding V_{dd}.

Performance of all cores are assumed to be decided by *cycle-conserving* algorithm. Among the performance demand of *cycle-conserving* algorithm for each core, the maximum is chosen for the performance of the whole cores. The additional power consumption compared to the multiprocessor systems occurs when the performance demands of cores are different from those of each other. And by using *cycle-conserving* the difference is dynamically changing according to the actual execution time of the tasks. Thus we define dynamic utilization L_n of core n as Equation 3. cc_i means the last execution time of τ_i at that time. Thus the power consumption of a certain point is $mg(max(L_1, ..., L_m))$. Energy consumption is the product of time and power. As a result our aim can be defined as maintaining L of all cores to have similar value all the time by dynamically migrating tasks between cores because the execution times of each tasks are meaningless as far as the dead-lines are kept.

$$L_n = \sum_{\forall completed\ \tau_i} \frac{cc_i}{P_i} + \sum_{\forall incompleted\ \tau_i} \frac{W_i}{P_i} \tag{3}$$

3.2 A Heuristic Approach

In this section we suggest dynamic repartitioning heuristic algorithm as a solution to the problem in section 3.1. Algorithm 1 is the pseudo code of dynamic repartitioning algorithm. Basic idea of the suggested algorithm is migrating tasks from the cores which have high L to the cores having low L until all the cores have similar L. This repartitioning occurs at the completion and the release of the tasks.

Cores can be categorized into donator or grantee group. The two groups are exclusive. In other words if a core is in donator group then it can not be in grantee group. A core in donator group have tasks which is initially partitioned to that core but migrated to some cores in grantee group. A core in grantee group have tasks to run at that time which is not initially partitioned to that core. By separating these two groups, the situations that a core give its task to others and get tasks from others can be easily prevented. This makes the algorithm work more effectively and in understandable manner.

As shown in line 28 and 35, L of a core is updated when tasks are completed and released. If a task is released, algorithm checks that updated L is not over 1.0. When L is over 1.0 there should be migrated tasks from other cores in the core. Thus all the migrated tasks are returned to their original cores. After this procedure repartitioning function will be called. Repartitioning function which is described from line 12 to line 26 is actually doing migration job. It decides the source core as the core with the highest L. If the source core in grantee group then the source core should return the task with minimum utilization in the source core to the initially partitioned core of the task. If the source core is not grantee group then the source core will migrate the task which have the lowest utilization in the source core to the destination core which is the minimum utilization core among the cores not in the donator group. This procedure will be repeated until L of the destination exceeds L of the source. When it happens it can be thought that all cores have less differences of L than the lowest task utilization in the core which has the minimum utilization at that time.

Whenever a migration occur or a task is released the algorithm checks that L of the core with the incident does not exceed 1.0. If L of a certain core becomes to exceed 1 then the migrated tasks in it will be restored to the cores in which the tasks were scheduled initially. Thus suggested algorithm never breaks the dead-lines by the property of EDF algorithm.

4 Evaluation

The suggested algorithm is evaluated by simulations. Our simulator uses several partitioning algorithms like NFD(Next-Fit-Decreasing), FFD(First-Fit-Decreasing), BFD(Best-Fit-Decreasing) and WFD. For comparison all the simulations were done both with and without dynamic repartitioning algorithm.

There are many factors which affect the energy consumption. Based on the related researches[10, 7, 4, 8, 9] we extracted major factors described in Table 2 which were changed to simulate the different situations. α means the upper limit of the utilization which a task can get. The utilizations of tasks are randomly generated and they follow the uniform distribution.

Table 2. Parameters used in the evaluations

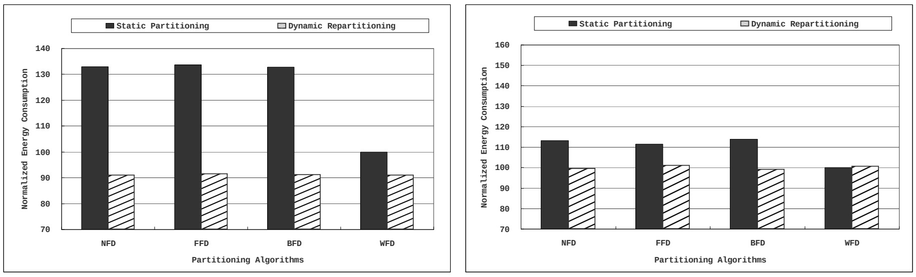
Parameters	Values
α	0.1, 0.3, 0.5
Number of Cores (m)	2, 4, 8, 16
Task Load ($\sum_{i=1}^{m} \frac{U_i}{m}$)	0.1, 0.5, 0.9
Execution time (cc)	Normal distribution with μ: {20, 50, 80}% of WCET and σ: 1/6

Figure 2 shows the difference of energy consumption according to the partitioning heuristics and the combination of each heuristic and the suggested algorithm. The results are normalized to that of WFD without dynamic repartitioning. With the result we found that WFD performs better than the other

Algorithm 1 Dynamic task repartitioning algorithm

```
$C_{max}$ and $C_{min}$ each returns the core with the highest and the lowest $L$
$\mathit{\Gamma}(C)$ returns a task $\tau_r$ such that:
    $\forall i$ where $\mathit{\Pi}(\tau_i) = \mathit{\Phi}(\tau_i) = C$, $(u_i <= u_r) \wedge (r \in i) \wedge (\tau_r$ is ready to run$)$
$\mathit{\Pi}(\tau)$ means the core where $\tau$ is partitioned initially
$\mathit{\Phi}(\tau)$ means the core where $\tau$ is currently located
$\mathbb{D}$ is the set of $C$ such that:
    $\exists \tau$ where $\mathit{\Phi}(\tau) = C, \mathit{\Pi}(\tau) \neq C$                    /* Donator */
$\mathbb{G}$ is the set of $C$ such that:
    $\exists \tau$ where $\mathit{\Pi}(\tau) = C, \mathit{\Phi}(\tau) \neq C$                    /* Grantee */

migrate($\tau$, $C$):
    $\mathit{\Phi}(\tau) \longleftarrow C$;

repartitioning():
    do
        $C_{src} \longleftarrow C_{max}$;
        if $C_{src} \in \mathbb{G}$
            $\tau_l \longleftarrow \mathit{\Gamma}(C_{src})$;
            $C_{dst} \longleftarrow \mathit{\Pi}(\tau)$;
            migrate($\tau_l$, $C_{dst}$);
        else
            $\tau_l \longleftarrow \mathit{\Gamma}(C_{src})$;
            $C_{dst} \longleftarrow C_{min}$ where $C_{min} \notin \mathbb{D}$;
            if $(L_{dst} + u_l) > 1$
                migrate($\tau$, $C_{dst}$);
            else
                break;
    while $(L_{C_{src}} > L_{C_{dst}})$;

upon task_release($\tau_i$):
    $L_i \longleftarrow W_i/P_i$;
    if $(L_{\mathit{\Phi}(\tau_i)} > 1)$
        for each task $\tau'$ in $\mathit{\Phi}(\tau_i)$
            if $\mathit{\Pi}(\tau') \neq \mathit{\Phi}(\tau_i)$
                migrate($\tau'$, $\mathit{\Pi}(\tau')$);
    repartitioning();

upon task_completion($\tau_i$):
    $L_i \longleftarrow cc_i/P_i$;                    /* $cc_i$ := actual used cycles */
    repartitioning();
```

heuristics in multicore systems too. However with dynamic repartitioning the energy consumption of all heuristics become similar. By using dynamic repartitioning algorithm combined with NFD which shows the worst energy efficiency 42% of energy was saved.



(a) Task load = 0.5 (medium load)

(b) Task load = 0.1 (low load)

Fig. 2. Normalized energy consumption according to the partitioning algorithms ($m = 4$, $\alpha = 0.1$, μ of $cc = 20\%$ of WCET)

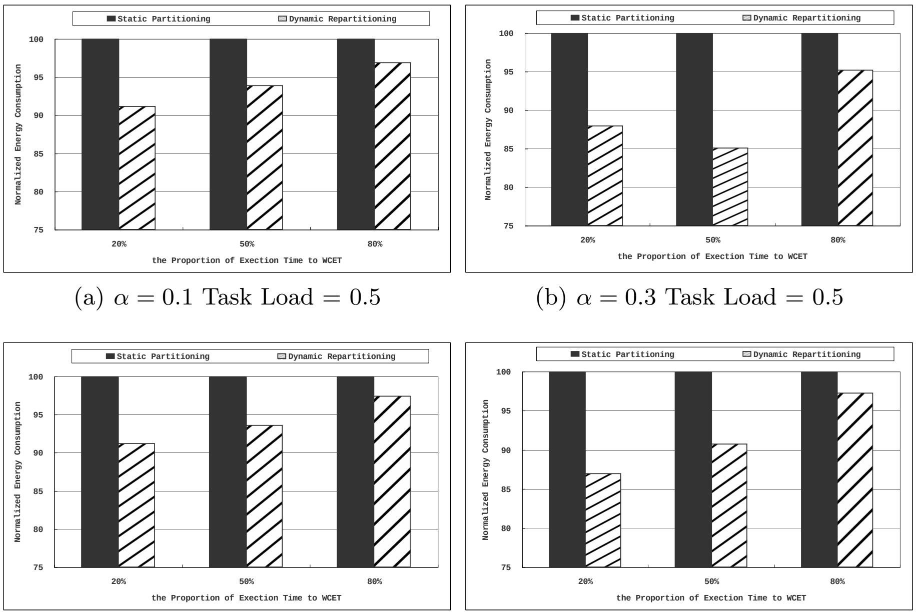
As shown in Figure 2 in the low load the dynamic repartitioning did not produce good results. Under low load all cores had low performance demands. Thus there were little differences among the performance demands of the cores. Due to that the benefits from dynamic repartitioning were little there.

In case that the task loads were over 0.5, as shown in Figure 3 (a),(b),(c) and (d) the dynamic repartitioning worked better as the difference between cc and $WCET$ grew. This is because that more differences causes more changes of L and thus the load balance among cores were more spoiled. We also found that bigger α made dynamic repartitioning more effective. Small α means that the length of WCET is relatively short compared to the period. Thus there is relatively low margin of cc variance. Moreover the number of tasks increased when the α decreased because the task load was fixed. The actual execution time of a task is randomly decided at every rounds. If the number of task grows, in macro view there is less change of L because more samples produce more stable average value and L is an aggregation value of the random values.

The differences of demanding performance among cores tend to be greater when the number of cores grows because many cores mean many different demanding performances at a certain time. Figure 4 shows the tendency. With this result we can tell that while the number of cores grows, the benefit from dynamic repartitioning also grows.

5 Conclusion

This paper introduces the problem of using DVS on a multicore processor which has the limitation that all cores should run at the same performance level. As far



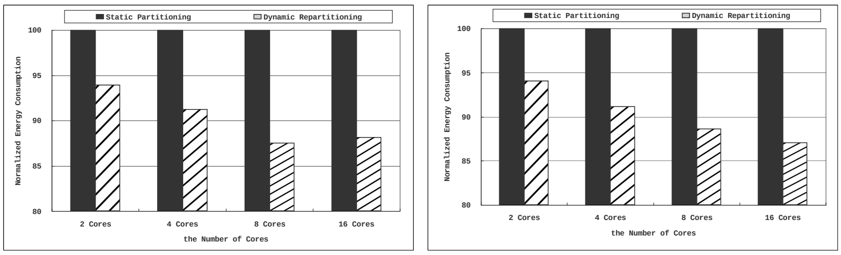
(a) $\alpha = 0.1$ Task Load $= 0.5$

(b) $\alpha = 0.3$ Task Load $= 0.5$

(c) $\alpha = 0.1$ Task Load $= 0.9$

(d) $\alpha = 0.3$ Task Load $= 0.9$

Fig. 3. Normalized energy consumption (m = 4, WFD partitioned)



(a) Task load = 0.9

(b) Task load = 0.5

Fig. 4. Normalized energy consumption corresponding to the number of cores ($\alpha = 0.1$, WFD partitioned, μ of cc = 20% of WCET)

as we know this problem is introduced in this paper for the first time. And as a solution to that problem we suggest dynamic repartitioning algorithm based on the partitioned schedule which have been used on the multiprocessor systems. To reduce the energy consumption from unbalanced load of the cores the suggested algorithm migrates tasks from high load cores to low load cores based on the dynamically updated load when a task is released and completed.

The simulation results show that in general cases more than 10% of additional energy is saved even with WFD partitioning, the best energy efficient partitioning algorithm. Moreover with other partitioning algorithm, up to 42% of additional energy saving was achieved. No matter which partitioning method is employed at the beginning, our scheme results in the same level of energy consumption which is always more efficient than the best energy efficient partitioning.

However the algorithm presented in this paper is blind to the purpose of a system. Considering the fact that most mobile embedded system is targeted for a specific purpose, each system is expected to have a specific set of tasks to run. In the further work, we will develop more intelligent algorithms which are tailored for the purpose of the target system.

References

1. Inc., A.M.D.: Multi-core processors - the next evolution in computing. White paper, Advanced Micro Devices, Inc. (2005)
2. Yang, C., Chen, J., Luo, T.: An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition. (2005) 468–473
3. Leung, J., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation **2**(4) (1982) 237–250
4. Aydin, H., Yang, Q.: Energy-aware partitioning for multiprocessor real-time systems. In: International Parallel and Distributed Processing Symposium. (2003) 113b
5. Lopez, J.M., Garcia, M., Diaz, J.L., Garcia, D.F.: Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In: Proceedings of 12th Euromicro Conference on Real-Time Systems. (2000) 25–33
6. Lopez, J.M., Garcia, M., Diaz, J.L., Garcia, D.F.: Minimum and maximum utilization bounds for multiprocessor RM scheduling. In: Proceedings of 13th Euromicro Conference on Real-Time Systems. (2001) 67–75
7. Anderson, J.H., Baruah, S.K.: Energy-aware implementation of hard-real-time systems upon multiprocessor platforms. In: Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems. (2003) 430–435
8. Aydin, H., Melhem, R., Mosse, D., Mejia-Avarez, P.: Power-aware scheduling for periodic real-time tasks. IEEE Transactions on Computers **53**(5) (2004) 584–600
9. Gruian, F.: Hard real-time scheduling for low-energy using stochastic data and DVS processors. In: Proceedings of the 2001 international symposium on Low power electronics and design. (2001) 46–51
10. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proceedings of the 18th ACM Symposium on Operating Systems. (2001) 89–102