POWER OPTIMIZATION IN WIRELESS NETWORKS USING DYNAMIC VOLTAGE SCALING ALGORITHMS

Gavaskar Vincent[1], Sasipraba T. [2]

[1]Research Scholar, Sathyabama University, Chennai, India
[2]Professor & Dean, Sathyabama University, Chennai, India
Email: [1]gavaskarvins@gmail.com

ABSTRACT

Execution of tasks to be completed by specified deadline is paramount important for any scheduling algorithms. Most DVS algorithm doesn't consider real-time constraints and are based on solely average computational throughput. Since they use a simple feedback mechanism such as detecting the amount of idle time on the processor over a period of time and adjust the computational load, they cannot provide any timeliness guarantees and task may miss their execution deadlines. To alleviate the difficulty, we propose a simple mechanism for providing voltage scaling while maintaining real-time schedulability. In this mechanism we select the lowest possible operating frequency that will allow the RM or EDF scheduler to meet all the deadlines for a given task set. This frequency is set statistically and will not be changed unless the task set is changed. designing of SimDVS, a unified simulation environment for evaluating dynamic voltage scaling (DVS) algorithms. At present it evaluates two static RT-DVS algorithms and three dynamic RT-DVS algorithms. All RT-DVS algorithm's task execution module differs with each other. The SimDVS has been designed such a way that any new DVS algorithm can be evaluated easily by adding new task execution module to SimDVS. It also supports to add new machines specifications to in it.

Key words: SimDVS; RT-DVS; EDF scheduler

I. INTRODUCTION

The fundamental tradeoff between performance and battery life remains critically important. DVS tries to address the tradeoff between performance and battery life by taking into account two important characteristics of most current computer systems: (1) the peak computing rate needed is much higher than the average throughput that must be sustained; and (2) the processors are based on CMOS logic. The first characteristic effectively means that high performance is needed only for a small fraction of the time, while for the rest of the time, a low-performance, low-power processor would suffice. We can achieve the low performance by simply lowering the operating frequency of the processor when the full speed is not needed. DVS goes beyond this and scales the operating voltage of the processor along with the frequency. This is possible because static CMOS logic used in the vast majority of microprocessors today, has a voltage-dependent maximum operating frequency, so when used at a reduced frequency, the processor can operate at a lower supply voltage.

A. Real-time issues

For time-critical applications, however, the scaling of processor frequency could be detrimental. Particularly in real-time embedded systems like portable medical devices and cellular phones, where tasks must be completed by some specified deadlines, most algorithms for DVS known to date cannot be applied. These DVS algorithms do not consider real-time constraints and are based on solely average computational throughput. Typically, they use a simple feedback mechanism, such as detecting the amount of idle time on the processor over a period of time and then adjust the frequency and voltage to just handle the computational load. This is very simple and follows the load characteristics closely, but cannot provide any timeliness guarantees and tasks may miss their execution deadlines.

As an example, in an embedded camcorder controller, suppose there is a program that must react to a change in a sensor reading within a 5 ms deadline and that it requires up to 3 ms of computation time with the processor running at the maximum operating frequency. With a DVS algorithm that reacts only to average throughput, if the total load on the system is low, the processor would be set to operate at a low frequency, say half of the maximum and the task now requiring 6 ms of processor time cannot meet its 5 ms deadline. In general, none of the average

throughput-based DVS algorithms found in literature can provide real-time deadline guarantees.

In order to realize the reduced energy-consumption benefits of DVS in a real-time embedded system, we need new DVS algorithms that are tightly-coupled with the actual real-time scheduler of the operating system. In the classic model of a real-time system, there is a set of tasks that need to be executed periodically. Each task Ti, has an associated period Pi, and a worst-case computation time Ci. The task is released periodically once every Pi time units and it can begin execution. The task needs to complete its execution by its deadline, typically defined as the end of the period i.e., by the next release of the task. As long as each task Ti uses no more than Ci cycles in each invocation, a real-time scheduler can guarantee that the tasks will always receive enough processor cycles to complete each invocation in time. Of course, to provide such guarantees, there are some conditions placed on allowed task sets often expressed in the form of schedulability tests (P. Pillai and K.G. Shin 2001). A real-time scheduler guarantees that tasks will meet their deadlines given that:

*C*1.the task set is schedulable (passes schedulability test), and

*C*2.no task exceeds its specified worst-case computation bound.

When the DVS is applied in a real-time system it must ensure that both of these conditions hold. We have collected algorithms to integrate DVS mechanisms into the two most-studied real-time schedulers, Rate Monotonic (RM) and Earliest-Deadline-First (EDF) schedulers.

RM is a static priority scheduler and assigns task priority according to period it always selects first the task with the shortest period that is ready to run (released for execution). EDF is a dynamic priority scheduler that sorts tasks by deadlines and always gives the highest priority to the released task with the most. In the classical treatments of these schedulers, both assume that the task deadline equals the period (i.e., the task must complete before its next invocation) that scheduling and other overheads are negligible and that the tasks are independent (no task will block waiting for another task). In our design of DVS to real-time systems, we maintain the same assumptions since our primary goal is to reduce energy consumption rather than to derive general scheduling mechanisms. In the rest of this section, we present our algorithms that perform DVS in time-constrained systems without compromising deadline guarantees of real-time schedulers.

B. Static voltage scaling

We first propose a very simple mechanism for providing voltage scaling while maintaining real-time schedulability. In this mechanism we select the lowest possible operating frequency that will allow the RM or EDF scheduler to meet all the deadlines for a given task set. This frequency is set statically and will not be changed unless the task set is changed.

To select the appropriate frequency, we first observe that scaling the operating frequency by a factor x ($0 < x < 1$) effectively results in the worst-case computation time needed by a task to be scaled by a factor 1/x while the desired period (and deadline) remains unaffected. We can take the well-known schedulability tests for EDF and RM schedulers from the real-time systems literature and by using the scaled values for worst-case computation needs of the tasks can test for schedulability at a particular frequency. The necessary and sufficient schedulability test for a task set under ideal EDF scheduling requires that the sum of the worst-case utilizations (computation time divided by period) be less than one, i.e.,

$$C1/P1 + C2/P2 + C3/P3 + \ldots + Cn/Pn \leq 1 \qquad (1)$$

Using the scaled computation time values, we obtain the EDF schedulability test with frequency scaling factor ?:

$$C1/P1 + C2/P2 + C3/P3 + \ldots + Cn/Pn \leq \alpha \qquad (2)$$

Similarly, we start with the sufficient (but not necessary) condition for schedulability under RM scheduling and obtain the test for a scaled frequency. The operating frequency selected is the lowest one for which the modified schedulability test succeeds. The voltage, of course, is changed to match the operating frequency. Assume that the operating frequencies and the corresponding voltage settings available on the particular hardware platform are specified in a table provided to the software.

EDP_test (α):

($C1/P1 + C2/P2 + C3/P3 + \ldots + Cn/Pn \leq \alpha$)
return true;
else return false;

RM-test (α):

If ($\bar{V}\, Ti \varepsilon \{ T1, T2, T3, \ldots Tn \mid P1 \leq P2 \leq P3 \leq \ldots \leq Pn \}$
$\lceil \pi/p1 \rceil * C1 + \ldots + \lceil \pi/\pi \rceil (* Ci \leq \alpha * \pi)$
return true;
else return false;

select frequency (x):
use lowest freq.
$fi \varepsilon \{ f_1, f_2, f_3, f_4, f_5 \ldots f_m/f_1 < f_2 < f_3 < f_4 \ldots < f_m \}$
such that RM-test (f_i/f_m) or EDF_test (f_1 / f_m) } is true.

Fig. 1. Static voltage scaling algorithm for EDF and RM schedulers

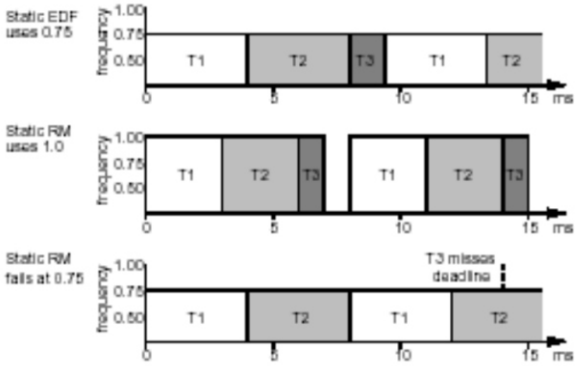

Fig. 2. Static voltage scaling

Table 1: Example Task Set

Task (Ti)	Computing Time (Ci)	Period (Pi)
1	3 ms	8 ms
2	3 ms	10 ms
3	1 ms	14 ms

summarizes the static voltage scaling for EDF and RM scheduling, where there are m operating frequencies $f1, f2, f3, \ldots fm$ such that $f1 < f2 < f3 < \ldots < fm$.

C. Cycle-Conserving RT-DVS

Although real-time tasks are specified with worst-case computation requirements they generally use much less than the worst case on most invocations. To take best advantage of this a DVS mechanism could reduce the operating frequency and voltage when tasks use less than their worst-case time allotment and increase frequency to meet the worst-case needs. When a task is released for its next invocation we cannot know how much computation it will actually require, so we must make the conservative assumption that it will need its specified worst-case processor time. When the task completes, we can compare the actual processor cycles used to the worst-case specification. Any unused cycles that were allotted to the task would normally be wasted idling the processor. Instead of idling for extra processor cycles, we can devise DVS algorithms that avoid wasting cycles (hence "cycle conserving") by reducing the operating frequency. These algorithms are tightly-coupled with the operating system's task management services, since they may need to reduce frequency on each task completion and increase frequency on each task release. The main challenge in designing such algorithms is to ensure that deadline guarantees are not violated when the operating frequencies are reduced.

For EDF scheduling, as mentioned earlier, we have a very simple schedulability test: as long as the sum of the worst-case task utilizations is less than x, the task set is schedulable when operating at the maximum frequency scaled by factor x. If a task completes earlier than its worst-case computation time we can reclaim the excess time by recomputing utilization using the actual computing time consumed by the task. This reduced value is used until the task is released again for its next invocation. We illustrate this in Figure 2 using the same task set and available frequencies as before, but using actual execution times from Table 1. Here, each invocation of the tasks may use less than the specified worst-case times but the actual value cannot be known to the system until after the task completes execution. Therefore, at each scheduling point (task release or completion) the utilization is recomputed using the actual time for completed tasks and the specified worst case for the others and the frequency is set appropriately.

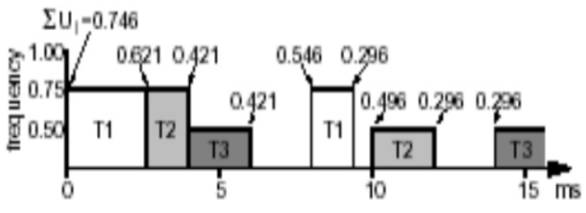

Fig. 3. Example of cycle-conserving EDF

Table 2. Two Invocations of Task Set

Task	Invocation 1	Invocation 2
1	2 ms	1 ms
2	1 ms	1 ms
3	1 ms	1 ms

Select_frequency ():
use lowest freq.
fi c { $f_1, f_2, f_3, f_4, f_5 \ldots f_m/f_1 < f_2 < f_3 < f_4 \ldots < f_m$ }
such that $U_1 + U_2 + U_3 + \ldots U_n \leq f_i/f_m$)
upon task_release (T_i):
set U_i to C_i/P_i:
select_frequency ():
upon task_compltion (T_i):
set U_i to cc_i/P_i:
select_frequency ();

Fig. 4. Cycle-conserving DVS for EDF schedulers

The algorithm (figure 4) itself is simple and works as follows. Suppose a task *Ti* completes its current invocation after using cci cycles which are usually much smaller than its worst-case computation time *Ci*. Since task *Ti* uses no more than cci cycles in its current invocation we treat the task as if its worst-case computation bound were *cci*. With the reduced utilization specified for this task, we can now potentially find a smaller scaling factor ? (i.e., lower operating frequency) for which the task set remains schedulable. Trivially, given that the task set prior to this change was schedulable, the EDF schedulability test will continue to hold and *Ti* (which has completed execution) will not violate its lowered maximum computing bound for the remainder of time until its deadline. Therefore, the task set continues to meet both conditions *C*1 and *C*2 imposed by the real-time scheduler to guarantee timely execution and as a result, deadline guarantees provided by EDF scheduling will continue to hold at least until *Ti* is released for its next invocation. At this point, we must restore its computation bound to *Ci* to ensure that it will not violate the temporarily-lowered bound and compromise the deadline guarantees. At this time, it may be necessary to increase the operating frequency. At first glance, this algorithm does not appear to significantly reduce frequencies, voltages and energy expenditure. However, since multiple tasks may be simultaneously in the reduced-utilization state, the total savings can be significant.

Assume f_j is frequency set by static scaling
algorithm
select_frequency ():
set $s_m =$ max_cycles_until_next_deadline ();
use lowest freq.
$f_1 \varepsilon \{ f_1, f_2, f_2, f_3, f_4, f_5, \ldots f_m/f_1 < f_2 < f_3 < f_4 \ldots < f_m \}$
such that $(d_1 + \ldots + d_n)/s_m \leq f_1/f_m$

upon task-release (Ti):
set $c\_left_i = C_i$;
set $s_m =$ max_cycles_until_next_deadline();
set $s_j = s_m * f_j/f_m$;
allocate_cycles (s_j);
select_frequency ();

upon task_completion (T_i):
set c_$left_i = 0$;
set $d_i = 0$;
select_frequency ();
during task_execution (T_i):
decrement c_$left_i$ and d_i;

allocate cycles (k): /* tasks sorted by period */
for $i = 1$ to
n, $T_1 \varepsilon \{ T_1, T_2, T_3, \ldots T_n \mid P_1 \leq P_2 \leq P_3 \leq \ldots \leq P_n \}$
if (c_$left_i < k$)
set $d_i =$ c_ $left_i$;
set $k = k -$ c_ $left_i$;

else
set $d_i = k$;
set $k = 0$;

Fig. 5. Cycle-conserving DVS for RM schedulers

We could use the same schedulability test-based approach to designing a cycle-conserving DVS algorithm for RM scheduling, but as the RM schedulability test is significantly more complex ($O(n2)$) where, n is the number of tasks to be scheduled), we will take a different approach here. We observe that even assuming tasks always require their

worst-case computation times, the statically-scaled RM mechanism discussed earlier can maintain real-time deadline guarantees. We assert that as long as equal or better progress for all tasks is made here than in the worst case under the statically-scaled RM algorithm, deadlines can be met here as well regardless of the actual operating frequencies. We will also try to avoid getting ahead of the worst-case execution pattern this way, any reduction in the execution cycles used by the tasks can be applied to reducing operating frequency and voltage.

Although conceptually simple, the actual algorithm (Figure 5) for this is some what complex due to a number of counters that must be maintained. In this algorithm, we need to keep track of the worst-case remaining cycles of computation *c*_left*i*, for each task *Ti*. When task *Ti* is released *c*_left*i* is set to *Ci*. We then determine the progress that the static voltage scaling RM mechanism would make in the worst case by the earliest deadline for any task in the system. We obtain *sj* and *sm*, the number of cycles to this next deadline, assuming operation at the statically-scaled and the maximum frequencies, respectively. The *sj* cycles are allocated to the tasks according to RM priority order, with each task *Ti* receiving an allocation $di \leq c\_left_i$ corresponding to the number of cycles that it would execute under the statically-scaled RM scenario over this interval. As long as we execute at least *di* cycles for each task *Ti* (or if *Ti* completes) by the next task deadline, we are keeping pace with the worst-case scenario, so we set execution speed using the sum of the *d* values. As tasks execute, their *c*_left*i* and *d* values are decremented. When a task *Ti* completes, *c*_left*i* and *di* are both set to 0 and the frequency and voltage are changed. Because we use this pacing criteria to select the operating frequency this algorithm guarantees that at any task deadline, all tasks that would have completed execution in the worst-case statically-scaled RM schedule would also have completed here, hence meeting all deadlines.

These algorithms dynamically adjust frequency and voltage, reacting to the actual computational requirements of the real-time tasks. At most, they require 2 frequency / voltage switches per task per invocation (once each at release and completion), so any overheads for hardware voltage change can be accounted in the worst-case computation time allocations of the tasks. As we will see later, the algorithms can achieve significant energy savings without affecting real-time guarantees.

D. Look-Ahead RT-DVS

The final (and most aggressive) RT-DVS algorithm that we introduce in this project attempts to achieve even better energy savings using a look-ahead technique to determine future computation need and defer task execution. The cycle-conserving approaches discussed above assume the worst case initially and execute at a high frequency until some tasks complete and only then reduce operating frequency and voltage. In contrast, the look-ahead scheme tries to defer as much work as possible and sets the operating frequency to meet the minimum work that must be done now to ensure all future deadlines are met. Of course, this may require that we will be forced to run at high frequencies later in order to complete all of the deferred work in time. On the other hand, if tasks tend to use much less than their worst-case computing time allocations, the peak execution rates for deferred work may never be needed and this heuristic will allow the system to continue operating at a low frequency and voltage while completing all tasks by their deadlines.

The actual algorithm for look-ahead RT-DVS with EDF scheduling is shown in Figure 6. As in the cycle-conserving RT-DVS algorithm for RM, we keep track of the worst-case remaining computation *c*_left*i* for the current invocation of task *Ti*. This is set to *Ci* on task release, decremented as the task executes and set to 0 on completion. The major step in this algorithm is the deferral function. Here, we look at the interval until the next task deadline, try to push as much work as we can beyond the deadline and compute the minimum number of cycles *x*, that we must execute during this interval in order to meet all future deadlines. The operating frequency is set just fast enough to execute '*s*' cycles over this interval.

Select_frequency (x);

use lowest freq.

$fi \in \{ f_1, f_2, f_3, f_4, f_4, f_5, \ldots f_m / f_1 < f_2 < f_3 < f_4 \ldots < f_m \}$

such that $x \leq f_i / f_m$

upon task_release (T_i);

set $c\text{–}left_i = C_i$;

defer ();

```
upon task_completion (Ti):
set c – lefti = 0;
defer ();

during task_execution (Ti):
decrement c_lefti;

defer ():
set U = C1/P1 + C2/P2 + C3/P3 + … + Cn/Pn;
set s = 0;
for i = 1 to
  n, Ti ε { T1, T2, T3, T4, … Tn | D1 > D2 > D3 ≥ … ≥ Dn }
/* Note: reverse EDF order of tasks */
    set U = U – Ci/Pi;
    set x = max (0, c_lefti = (1 – U) (Di – Dn));
    set U = U + (c_lefti – x)/(Di – Dn);
    set s = x + x;
    select_frequency (s/ (Dn – current_time));
```

Fig. 6. Look ahead DVS for EDF schedulers

To calculate, we look at the tasks in reverse EDF order (i.e., latest deadline first). Assuming worst-case utilization by tasks with earlier deadlines (Effectively reserving time for their future invocations) we calculate the minimum number of cycles *x*, that the task must execute before the closest deadline Dn in order for it to complete by its own deadline. A cumulative utilization *U* is adjusted to reflect the actual utilization of the task for the time after *Dn*. This calculation is repeated for all of the tasks using assumed worst-case utilization values for earlier-deadline tasks and the computed values for the later-deadline ones. The '*s*' is simply the sum of the *x* values calculated for all of the tasks and therefore reflects the total number of cycles that must execute by Dn in order for all tasks to meet their deadlines. Although this algorithm very aggressively reduces processor frequency and voltage it ensures that there are sufficient cycles available for each task to meet its deadline after reserving worst-case requirements for higher-priority (earlier deadline) tasks.

E. Summary of RT-DVS algorithms

All of the RT-DVS algorithms we presented thus far should be fairly easy to incorporate into a real-time operating system and do not require significant processing costs. The dynamic schemes all require computation (assuming the scheduler provides an EDF sorted task list) and should not require significant processing over the scheduler. The most significant overheads may come from the hardware voltage switching times. However in all of our algorithms, no more than two switches can occur per task per invocation period, so these overheads can easily be accounted and added to the worst-case task computation times.

For above examples we assume that the 0.5, 0.75 and 1.0 frequency settings need 3, 4, and 5 volts respectively and that idle cycles consume no energy. More general evaluation of our algorithms will be done in the next section.

II. SIMULATION AND RESULT ANALYSIS

A. Varying machine specifications

All of the previous simulations used three set of available frequency and voltage scaling settings. We now investigate the effects of varying the simulated machine specifications. The following tables summarize the hardware voltage and frequency settings, where each column consists of the relative frequency and the corresponding Processor voltage.

Table 3: Machine 1 Specification

Frequencies (normalized) f_i	0.5	0.75	1.0
Voltages(Volts) V_i	3	4	5

Table 4: Machine 2 Specification

Frequencies (normalized) f_i	0.375	0.5	0.625	0.75	0.875	1.0
Voltages (Volts) V_i	2.5	3	3.5	4	4.5	5

Table 5: Machine 3 Specification

Frequencies (normalized) *fi*	0.5	0.75	0.83	1.0
Voltages (Volts) *Vi*	3	4	4.5	5

Table 5: Machine 4 Specification

Frequencies (norm) f_i	0.36	0.55	0.64	0.73	0.82	0.91	1.0
Voltages (Volts) V_i	1.4	1.5	1.6	1.7	1.8	1.9	2.0

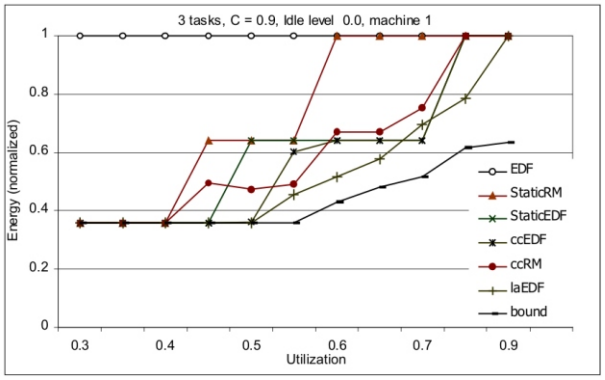

Fig. 7. (a) Normalized energy consumption with machine 1

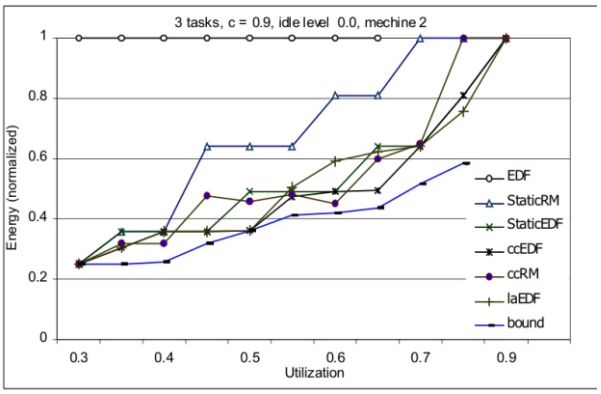

Fig. 7. (b) Normalized energy consumption with machine 2

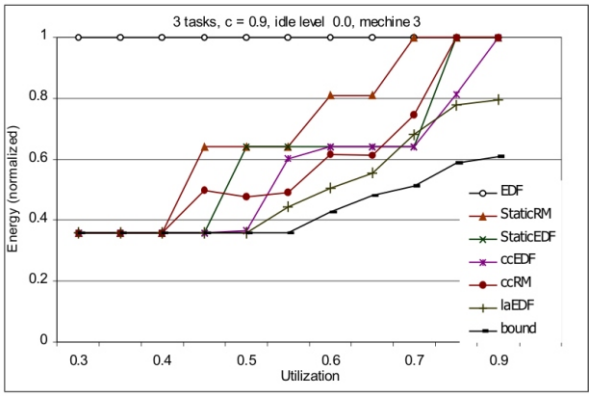

Fig. 7. (c) Normalized energy consumption with machine 3

Figure 7 shows the simulation results for machines 1, 2, 3 and 4. The Machine 1 and 4 used in all of the previous simulations, have frequency settings that can be expected on a standard PC motherboard, although the corresponding voltage levels were arbitrarily selected.

Machine 3 differs from Machine 1 in that it has the additional frequency setting, 0.83. With this small change, we expect only slight differences in the simulation results with these specifications. The most significant change is seen with cycle-conserving EDF (and Statically-scaled EDF, since the two are identical here). With the extra operating point in the region near the cross-over point between ccEDF and ccRM, the ccEDF algorithm benefits by shifting the cross-over point closer to full utilization.

As it has many more settings to select from the plotted curves tend to be smoother. Also, since the relative voltage range is smaller with this specification

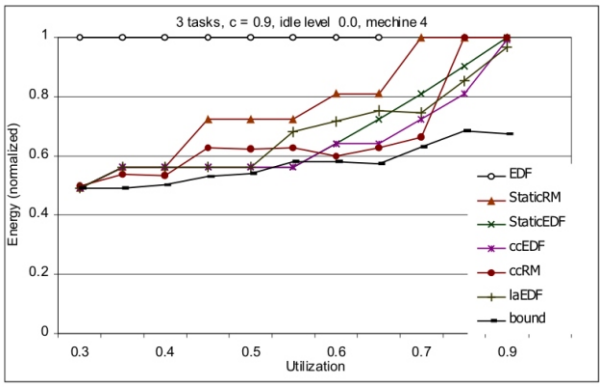

Fig. 7. (d) Normalized energy consumption with machine 4

the maximum savings is not as good as with the other two machine specifications. More significant is the fact that the cycle-conserving EDF outperforms the look-ahead EDF algorithm. The ccEDF and staticEDF benefit from the large number of settings, since this allows them to more closely match the task set and reduce energy expenditure.

With fewer settings, the frequency selected would be somewhat higher, so less processing is deferred, lessening the likelihood of needing higher voltage and frequency settings later thus improving performance. In a sense, the error due to a limited number of frequency steps is detrimental in the ccEDF scheme, but beneficial with laEDF. These results, therefore, indicate that the energy savings from the various RT-DVS algorithms depend greatly on the available voltage and frequency settings of the platform.

B. *Varying computation time*

In this set of experiments, we vary the distribution of the actual computation required by the tasks during each invocation to see how well the RT-DVS mechanisms take advantage of task sets that do not consume their worst-case computation times. In the preceding simulations, we assumed that the tasks always require their worst-case computation allocation. Figure 8a-c shows simulation results for tasks that require a constant 90%, 70%, and 50% of their worst-case execution cycles for each invocation. We observe that the statically-scaled mechanisms are not affected, since they scale voltage and frequency based solely on the worst-case computation times specified for the tasks. The results for the cycle-conserving RM algorithm do not show significant change, indicating that it does not do a very good job of adapting to tasks that use less than their specified worst-case computation times. On the other hand, both the cycle-conserving and look-ahead EDF schemes show great reductions in relative energy consumption as the actual computation performed decreases.

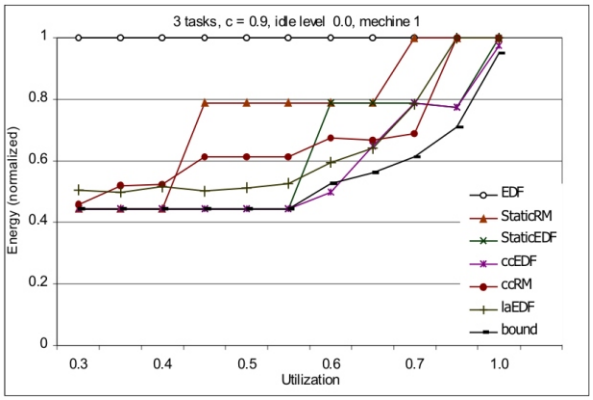

Fig. 8. (a) Normalized Energy consumption with computation set to fixed fraction of worst-case allocation $c = 0.9$

Figure 9 shows the simulation results using tasks with a uniform distribution between 0 and their worst-case computation. Despite the randomness introduced, the results appear identical to setting computation to a constant one half of the specified value for each invocation of a task. This makes sense, since the average execution with the uniform distribution is 0.5 times the worst-case for each task. From this, it seems that the actual distribution of computation per invocation is not the critical factor for energy conservation performance.

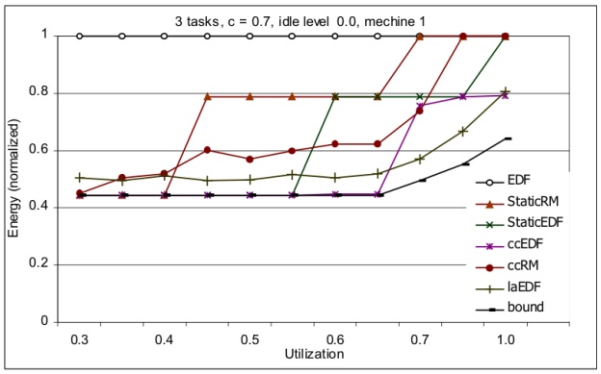

Fig. 8. (b) Normalized Energy consumption with computation set to fixed fraction of worst-case allocation $c = 0.7$

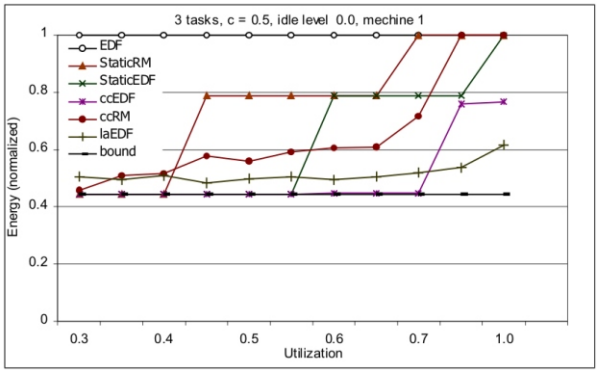

Fig. 8. (c) Normalized Energy consumption with computation set to fixed fraction of worst-case allocation $c = 0.5$

For the dynamic mechanisms the average utilization that determines relative energy consumption while in the static scaling methods the worst-case utilization is the determining factor. The exception is the ccRM algorithm which albeit dynamic, has results that primarily reflect the worst-case utilization of the task set.

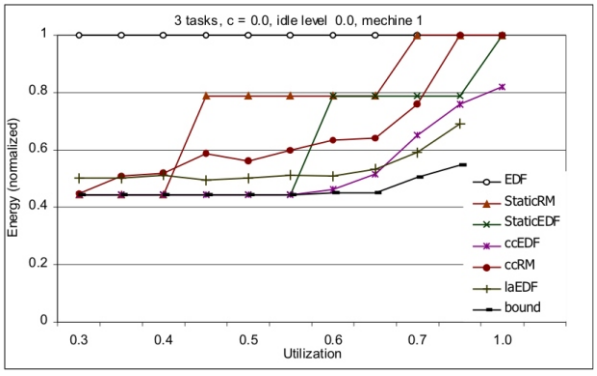

Fig. 9. Normalized Energy consumption with uniform distribution for computation

III. CONCLUSION & FUTUREWORK

We also discussed in details about designing of SimDVS, a unified simulation environment for evaluating dynamic voltage scaling (DVS) algorithms. At present it evaluates two static RT-DVS algorithms and three dynamic RT-DVS algorithms. All RT-DVS algorithm's task execution module differs with each other. The SimDVS has been designed such a way that any new DVS algorithm can be evaluated easily by adding new task execution module to SimDVS. It also supports to add new machines specifications to in it.

In the future, we would like to expand this work for aperiodic task set with varying deadlines. These algorithms will be implemented in portable devices to measure the important parameters which are all assumed to negligible during simulation, like switching overheads and computations overheads needed for voltage scaling. We will investigate DVS with probabilistic or statistical deadline guarantees. We will also explore integration with other energy conserving mechanisms, including application energy adaptation and energy-adaptive communication (both real-time and best-effort). Additionally, although developed for portable devices, RT-DVS is applicable widely in general real-time systems. The energy savings works well for extending battery life in portable applications, but can also reduce the heat generated by the real-time embedded Controllers in various factory or home automation products or even reduce cooling requirements and costs in large- scale multiprocessor supercomputers.

REFERENCES

[1] M. Fleischmann, (2008) Crusoe Power Management: Reducing the Operating Power with LongRun. In Proc. of Hot Chips 12 Symposium.

[2] Advanced Micro Devices, Inc., (2010) AMD PowerNow Technology.

[3] Intel, Inc., (2008) the Intel(R) XScale (TM) Microarchitecture Technical Summary.

[4] F. Yao, A. Demers and S. Shenke, (1995) A Scheduling Model for Reduced CPU Energy. In Proc. of 36th Annual Symposium on Foundations of Computer Science, pages 374-382.

[5] I. Hong, G. Qu, M. Potkonjak and M.B. Srivastava, (1998) Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor. In Proc. Of Real-Time Systems Symposium, pages 178-187.

[6] T. Ishihara and H. Yasuura, (1998) Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In Proc. of International Symposium On Low Power Electronics and Design, pages 197-202.

[7] Shin, K. Choi and T. Sakurai, (2010) Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In Proc. of International Conference on Computer-Aided Design, pages 365-368.

[8] H. Aydin, R. Melhem, D. Mosse and P.M. Alvarez, (2001) Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In Proc. of Real-Time Systems Symposium.

[9] P. Pillai and K.G. Shin, (2011) Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In Proc. of 13th ACM Symposium on Operating Systems Principles (SOSP'01).

[10] D. Shin, W. Kim, J Jeon, J Kim and S.L. Min, (2002) SimDVS: An Integrated Simulation Environment for Performance Evaluation of DVS algorithms.

[11] G. Quan and X. Hu, (2011) Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In Proc. of Design Automation Conference, pages 828-833.

[12] D. Shin, J. Kim and S. Lee, (2009) Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. IEEE Design and Test of Computers, 18(23):20-30.

[13] F. Gruian (2001) Hard Real-Time Scheduling Using Stochastic Data and DVS Processors. In Proc. of International Symposium on Low Power Electronics and Design, pages 46-51.

[14] W. Kim, J. Kim and S.L. Min, (2010) A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. To appear in Proc. of Design, Automation and Test in Europe (DATE'Oœ).

[15] J. Lehoczky, L. Sha and Ding, (1989) The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In Proc. of Real-Time Systems Symposium, pages 166-171.

[16] T. Burd, T. Pering, A. Stratakos and R. Brodersen, (2000) A Dynamic Voltage Scaled Microprocessor System. In Proc. of International Solid-State Circuits Conference, pages 294-295.

[17] D. Shin, W. Kim, J Jeon, J Kim and S. L. Min, (2010) SimDVS: An Integrated Simulation Environment for Performance Evaluation of DVS algorithms.