Toward the Placement of Power Management Points in Real Time Applications

Nevine AbouGhazaleh, Daniel Mossé, Bruce Childers and Rami Melhem
Department of Computer Science, University of Pittsburgh,
Pittsburgh, PA 15260
{nevine, mosse, childers, melhem}@cs.pitt.edu

Abstract

Dynamically changing CPU voltage and frequency at specific *power management points* (PMPs) has been shown to greatly save the processor energy. However, dynamic speed adjustment is not without overheads. In this paper we study the effect of different overheads on both time and energy. We also propose a theoretical solution for choosing the granularity of inserting PMPs in a program. We validate our theoretical results and show that the accuracy of the theoretical model is between zero and five management points of simulation results.

1 Introduction

In the last decade, there has been considerable research on low-power system design. On-going research has greatly influenced embedded real-time systems' design due to the number of applications running on power-limited systems that have tight temporal constraints. Recently, dynamic voltage scaling (DVS), which involves dynamically adjusting CPU voltage and frequency, has become a major research area. Reducing a processor's supply voltage typically leads to considerable power savings, but also introduces delays in response time and additional energy consumption for speed adjustments. Thus, there is a need to optimize energy consumption while considering both the savings achieved and the overhead of changing processor speed and supply voltage.

In Mosse et al. [1] we introduced what we now call *power management points* (PMPs), which are pieces of code that manage information about the execution of program segments to make decisions about and to change CPU speed. The desired speed is computed according to a speed setting algorithm (for examples, see [1, 12, 8, 3]). Although PMPs can be inserted by the compiler in a program or executed by the operating system, at specific times (e.g., context switch times), this paper focuses on compiler-inserted PMPs.

Dynamic speed setting schemes used in periodic real time systems take advantage of unused time to slow down the CPU speed of future tasks or task segments. This can be done when the system load is light or when there is time left from previous program segments. It is shown in [1] that statistical slack management produces savings in CPU energy consumption of up to 90% compared to no power management and up to 60% compared to a static speed setting scheme.

Compiler insertion of PMPs is particularly useful for programs with frequent procedure calls or loops with a relatively large number of iterations. To select the granularity of the program segment that is assigned a single speed, Hsu et al. [11] uses global program analyses to detect regions of sufficiently large granularity, and then selects a single region with the highest predicted benefit, where they place a speed setting instruction.

Our contributions in this paper are twofold: (1) modeling how to incorporate the effect of overhead in speed adjustment schemes, and (2) providing a theoretical solution for deciding the optimal number of equally spaced PMPs that achieve the minimum energy consumption. We compare our results from the theoretical solution with simulated results of our previous speed setting schemes from [1]. The theoretical results show a decision accuracy within five PMPs of the simulation results.

We describe our model in the next section. The effect of different overheads are considered and shown for dynamic speed setting schemes in Section 3. Section 4 presents a theoretical solution for selecting the best number of PMP and compares these results with simulation results for each scheme. Concluding remarks are in Section 5.

2 Model

Our techniques are targeted to embedded systems where applications execute for a specified allocated time, d, decided by a scheduler. In this initial work, we consider the sequential form of program execution, where a program can be divided into n segments of equal length, to determine an optimal number of PMPs. Such a model is applicable to loops that have large compile-time constant trip counts. In these loops, power management points can be placed every so many loop iterations to adjust processor speed. We are currently extending our work to more general programs.

In this paper, we insert a PMP before each program segment. A segment is characterized by its worst case execution time, wc_i, and average execution time avg_i. The actual execution time of segment i, ac_i, is only known at run time, but is limited by the wc_i such that $0 \leq ac_i \leq wc_i$. These times describe the execution behavior of segment i when the processor is running at its maximum speed. The quantity $\alpha = avg_i/wc_i$ is an indication of the expected slack in the execution time of the segment.

Given the parameters above, we can compute the *static slack* in the system, which is the amount of free time in the system with respect to the computational requirements of an application. The optimal static speed, S_{static}, for all segments can be computed as $\sum_{i=1}^{n} \frac{wc_i}{d} = load$. It has been proven that this speed is optimal, while meeting all deadlines [12]. Henceforth, we assume that all segments are slowed down to S_{static}, making the CPU busy at all times (albeit at a reduced speed), if $ac_i = wc_i, \forall i$. This is equivalent to having 100% load (or load = 1).

Furthermore, extra slack is generated whenever a program segment finishes its execution before the estimated worst-case time for this segment. This is called *reclaimed slack*.

For $CMOS$ technology, dynamic power consumption is directly proportional to the frequency and to the square of the supply voltage: $P = aCSV^2$, where a is the activity factor in the processor, C is the effective switched capacitance, S is the operating frequency (speed), and V is the supply voltage. In our work, we use a model similar to Transmeta's TM 5400 processor [4]. Our model has a 16-step frequency scale that ranges from 200MHz at 1.1V to 700 MHz at 1.65V. Each step is $\approx$ 33MHz. We consider the different overheads of each frequency and voltage change below.

2.1 Sources of Overhead

When computing and changing CPU frequency and voltage, several sources of overhead may be encountered. The principal sources of overhead are computing the new speed and setting the speed through a voltage transition in the processor's DC-DC regulator (resulting in a processor frequency change) and the clock generator (PLL). We denote changing both voltage and frequency by the term *speed change*. Speed changing takes time and consume energy. Below we discuss the time overhead; the energy consumed can be derived from the equation for power above, and knowing that $E = Pt$.

Computing the new speed: For each adjustment scheme considered, the time overhead, F, is approximately constant in terms of the number of cycles needed for execution. This includes the overhead of calling library functions and performing the operations that compute the new speed. Since this may be executed at different frequencies, the time overhead, O_1, is:

$$O_1(S_i) = \frac{F}{S_i} \tag{1}$$

where S_i is the CPU speed executing segment i (and the PMP code at the end of segment i). From experiments with SimpleScalar 3.0 [10] (a micro architectural simulator), where we implemented speed setting and inserted PMPs in applications like an MPEG decoder, we observed that the overhead of computing the new speed varied between 280 and 320 cycles. In the experiments below we fix the overhead of computing the new speed to 300 cycles.

Setting the new speed: To change voltage, a DC-DC switching regulator is employed. This regulator cannot make an instantaneous transition from one voltage to another [2]. This transition takes time and energy. When setting a new speed, the CPU clock and the voltage fed to the CPU need to be changed, incurring a wide range of delays. For example, the *Strong Arm SA-1100* is capable of on-the-fly clock frequency changes in the range of 59MHz to 206MHz where each speed and voltage change incurs a latency of up to 150 μsec [6], while the *lpARM* processor [7] (a low-power implementation of the *ARM*8 architecture) takes 25 μs for a full swing from 10 MHz to 100 MHz. Another example is the Transmeta TM5400, which is specifically designed for DVS [4]. Some systems can continue operation while speed and voltage change [7, 2], but the frequency continues to vary during the transition period. We take a conservative approach and assume that the processor can not execute application code during this period.

Moreover, when looking at changing speed from the energy perspective, the lpARM processor incurs at most 4μJ, which is equivalent to 712 full-load cycles for the transition between 5 - 80 MHz[5]. In our simulation we assume a constant number of overhead cycles, G, for each speed step transition. This overhead is assumed to be 320 cycles for every 33MHz step (from [5], 712 cycles for 5-80 MHz transition $\simeq$ 320 cycles for 33MHz transition). The time overhead for speed changes, O_2, depends on the speed that the CPU is executing the PMP and can be computed as follows:

$$O_2(S_{i-1}, S_i) = G\frac{d(S_{i-1}, S_i)}{S_{i-1}} \tag{2}$$

where $d(S_i, S_j)$ is a function that returns the number of speed steps needed to make a transition between S_i and S_j. In the Transmeta model, this function returns how many multiples of 33MHz is the difference between S_i and S_j. The energy overhead is assumed to follow the same power function presented in section 2 multiplied by the time taken to accomplish the speed transition.

We study the impact of varying this overhead on the selection of the optimal number of PMPs in Section 4.

3 Speed Adjustment Schemes

We use two schemes from [1] as examples to demonstrate how to include the aforementioned overhead in speed adjustments at each PMP. Deadlines are only violated in cases where the processor needs to run at almost the maximum speed to meet the application's deadline and there is not enough slack to accommodate the time overhead for a single speed computation. We regard this as insignificant for the purpose of this study.

3.1 Proportional Dynamic Power Management

In this scheme, reclaimed slack is uniformly distributed to all remaining segments proportional to their worst-case execution times. The time overheads for both computing and setting the new speed are subtracted from the time remaining to the deadline. Our main concern here is to compute the exact time left for the application to execute before the deadline. The processor's speed for segment i, S_i, is computed as follows: the execution times for the remaining tasks are stretched out based on the remaining time to the deadline $(d - t_{i-1})$, while taking into consideration the overhead of switching the speed of the current task $(O_{cur}(S_{i-1}, S_i) = O_1(S_{i-1}) + O_2(S_{i-1}, S_i))$ and the overhead of potentially needing to switch the speed of the next task to the static optimal speed $(O_{next}(S_i, S_{static}) = O_1(S_i) + O_2(S_i, S_{static}))$, where S_{static} is the optimal static speed [12]. The total overhead, O_{total}, is:

$$O_{total}(S_{i-1}, S_i) = O_{cur}(S_{i-1}, S_i) + O_{next}(S_i, S_{static}) \tag{3}$$

With overhead, this scheme computes a new speed as:

$$S_i = \frac{\sum_{j=i}^{n} wc_j}{d - t_{i-1} - O_{total}(S_{i-1}, S_i)} \tag{4}$$

where t_{i-1} is time at the beginning of the i^{th} PMP (the end of segment $i-1$).

Given that the voltage setting overhead is dependent on the new frequency, we note that S_i appears on both sides of the formula. It is solved iteratively and we have observed that, on average, it converges in about two iterations. Because

the algorithm starts with the speed for the previous segment, there is typically minimal change in speed, which leads to fast convergence.

3.2 Dynamic Greedy Power Management

The Dynamic Greedy scheme distributes the reclaimed slack only to the segment immediately after a PMP. The way to compute the speed of the next segment is similar to (4): the time for the next task is spread over the remaining time to the deadline, minus the overhead of computing and switching speeds from (3).

$$S_i = \frac{wc_i}{d - t_{i-1} - \frac{\sum_{j=i+1}^{n} wc_j}{S_{static}} - O_{total}(S_{i-1}, S_i)} \quad (5)$$

3.3 Results

We simulated our two power management schemes with and without overhead for a hypothetical program divided into different number of equal length segments. The actual execution times ac_i of each segment is drawn randomly from a normal distribution. Each data point presented is an average of 500 runs. We use the energy function from Section 2, and the energy consumption is normalized to a case where no power management is empolyed.

In Figure 1 we show the energy consumption based on the number of PMPs for the Proportional and the Dynamic Greedy schemes. The optimal number of PMPs varies according to many factors, like α or the speed adjustment scheme. We show results for $\alpha = 0.6$ and 0.8 and a variable number of PMPs (from 5 to 30). From the figure, the optimal number of PMPs, based on simulation, is between 10 and 20 for α=0.6 and between between 5 and 15 for $\alpha = 0.8$. Other values of α behaved consistently with these results.

We noticed that for higher number of PMPs inserted in a program, the average number of step transitions needed at each PMP for the greedy scheme exceeds those needed for the proportional scheme. This adds a greater energy burden on the greedy's total energy consumption more than the proportional's consumption. As a result, although in general, the greedy energy consumption is less than the proportional's at load = 1, in figure 1

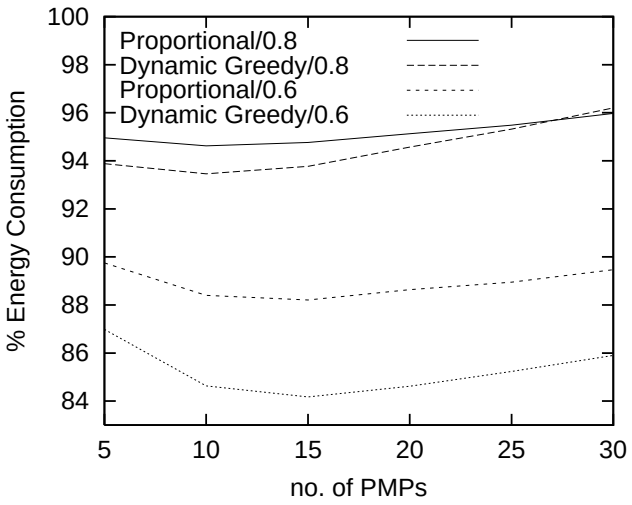

Figure 1: Total Energy Consumption for different schemes versus the number of PMPs at $\alpha = 0.6$ and $\alpha = 0.8$.

there is an overlapping of their energy curves for $\alpha = 0.8$. There is a smaller difference in energy as the number of PMPs increases for $\alpha = 0.6$.

The curve shapes are due to two opposing forces. First, the more management points, the better the power management and thus the lower energy consumption. However, the amount of energy consumed increases with an increasing number of PMPs, due to the overhead. The combination of these two factors is illustrated in Figure 1.

We also see that Greedy has lower energy consumption than Proportional. This is because Greedy is more aggressive at slowing down the CPU speed, counting on future reclaiming for slowing down future tasks.

Next we present a theoretical solution for selecting the number of PMPs to insert in a program.

4 Optimal Number of PMPs

The minimum energy consumption for any scheme depends on the optimal number of equally-spaced PMPs due to the effect of the energy overhead on the total energy consumed.

In this section, we develop a theoretical framework for deciding on the number of PMPs, given that: (1) each segment has perfect execution behavior (i.e, $ac_i = avg_i$ for all $1 \leq i \leq n$), (2) there is a constant time overhead, $h \approx O_1 + O_2$, for the insertion of any PMP, and (3) the speed range is

continuous.

The total energy consumed for n segments is computed based on the formula given in Section 2 for power, and knowing that the $E = Pt$, where t is the time taken. Part of each segment's energy is consumed in the actual execution of the segment, avg_i, while the other part is consumed in the overhead induced by changing the speed. The total energy, E_n, is the summation of the segments' energies as shown below.

$$E_n = aC\sum_{i=1}^{n} S_i V_i^2 (avg_i + \frac{h}{S_i}) = \gamma \sum_{i=1}^{n} S_i^3 (avg_i + \frac{h}{S_i}) \quad (6)$$

where the speed S_i is proportional to voltage, and γ is a multiplicative factor. The energy overhead is reflected in this equation by the term h/S_i, where it represents the average time taken for each speed change at each PMP.

In our analytical solution, we compute the speed S_i using the following formulas. We use these speed values to evaluate the energy consumption of the actual execution of the segment. The formulas are derived from the corresponding ones presented in Section 3 using our earlier assumptions.

Proportional

$$\frac{S_{Static}}{S_i} = \frac{n}{n-i+1} \prod_{k=1}^{i-1} \left[1 - \frac{\alpha}{n-k+1}\right] \quad (7)$$

where n is the number of placed PMPs.

Dynamic Greedy

$$\frac{S_{Static}}{S_i} = \frac{1-(1-\alpha)^i}{\alpha} \quad (8)$$

Due to the space limitation, the derivation of these formulas is not included in this paper.

4.1 Performance of Analytical Model

The optimal number of PMPs varies based on several parameters in the program execution behavior, such as variability of the execution α, and the amount of overhead for changing the speed. Figures 2 and 3 show the effect of varying the number of overhead cycles, h, on the total energy consumed using the analytic model. The results shown are for $\alpha = 0.6$, although other values of α have similar behavior. The optimal number of PMPs in the Proportional scheme lies in the range of 5-15, while for Dynamic Greedy it is from 10 to 30. As predicted, this optimal number decreases with increased overhead. However, this does not apply when $\alpha = 1$, because as α reaches 1 the desired optimal speed reaches S_{static}, with no CPU time to reclaim. Henceforth, we exclude the $\alpha = 1$ case from our experiments.

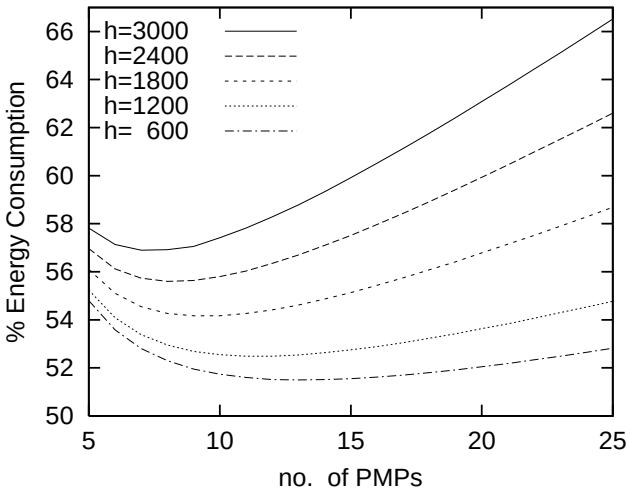

Figure 2: Total energy consumption for the Proportional scheme versus the number of PMPs, for different overheads, where $\alpha = 0.6$.

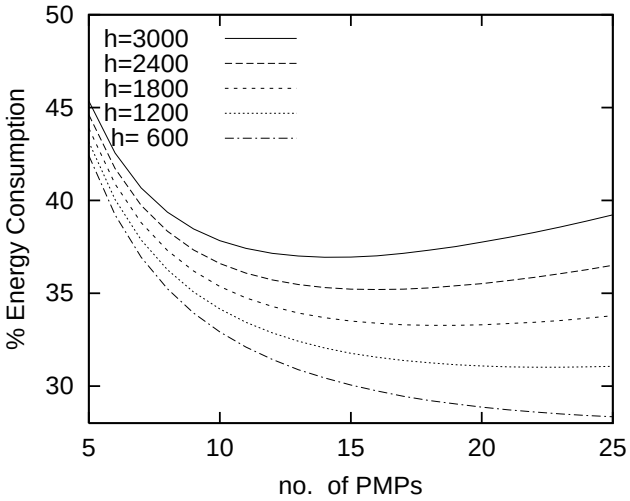

Figure 3: Total energy consumption for Dynamic Greedy scheme versus the number of PMPs, for different overheads, where $\alpha = 0.6$.

We ran experiments to validate our theoretical model by comparing the results with simulation results for the Proportional and the Dynamic

Greedy schemes.

Table 1 shows the number of PMPs determined by the theoretical model and the simulation for the Proportional scheme. The table shows the results for the same programs with different α and overhead values that might describe different DVS processors. The overhead values are presented as a pair of the theoretical overhead h and its corresponding simulated overhead F and G. For example, (1000/ 300,320) means h is 1000 cycles, while F and G are equal to 300 and 320 cycles, respectively. We use these figures for the two overheads (theoretical and simulated) because, from experiments with SimpleScalar, we observed that, in the Proportional scheme, the average number of transitions for the whole programs is 2.2 ($300 + 2.2 * 320 \approx 1000$). The table also shows variations in the theoretical of ± 2 PMPs from the simulation. There is a strong matching in the α's middle range, which are the most typical values of α. The variations come from the assumption that the speed is continuous in the theoretical method while it is discrete in the simulation. Moreover, the simulation is limited by a minimum speed.

	(1000/ 300,320)		(2000/ 600,640)		(3000/ 900,960)	
α	T	S	T	S	T	S
0.2	10	12	7	7	6	7
0.4	12	12	9	9	7	6
0.6	12	12	9	9	7	6
0.8	11	9	8	6	7	5

Table 1: Theoretical (**T**) versus Simulation (**S**) choice of optimal number of PMPs for the Proportional scheme.

During simulation, we noticed that, on average, Dynamic Greedy performs step transitions three times more than the Proportional scheme. Thus the choice of $h = 3000$ that corresponds to simulation overhead of $F = 300$ and $G = 320$. Table 2 shows that the optimal number of PMPs varies dramatically with α. For example, this variation at overhead (3000/ 300,320) ranges from 9 to 29 PMPs, corresponding to α's range 0.2-0.8. This higher number of PMPs is in concert with the higher number of speed changes that are made in Greedy.

Although not shown, we observed that the theoretical results are closer to the simulated results as the F and G overheads decrease. The difference between the simulated and theoretical results can be seen by comparing Figure 1 with Figures 2 and 3. The difference results because the analytical model does not take overheads into account when computing the new speeds, only when computing the energy. Further research is needed to obtained a more tight coupling between the values of the theoretical and the simulated overheads.

	(3000/ 300,320)		(6000/ 600,640)		(9000/ 900,960)	
α	T	S	T	S	T	S
0.2	29	25	20	15	16	11
0.4	22	19	14	12	11	9
0.6	14	12	10	9	8	6
0.8	9	9	7	6	5	4

Table 2: Theoretical (**T**) versus Simulation (**S**) choice of optimal number of PMPs for the Dynamic Greedy scheme.

5 Conclusion

For variable voltage systems, the overhead and selection of a speed setting scheme must be carefully considered. There may be cases where the energy consumption exerted by the overhead of selecting and setting a new speed overwhelms any energy savings of a speed setting algorithm. This implies that system energy can be jeopardized by employing such speed adjustments.

To minimize the overhead of speed adjustments, it is critical that for programs with a relatively small number of segments to know the optimal number of adjustment points for choosing the best speed adjustment scheme. However, for programs with a large number of segments, it is sufficient to identify the boundary of optimality, as the energy curve will become flatter beyond the optimal number of power management points.

We also saw that Greedy has smaller energy than Proportional, especially for workloads with higher variability in the actual execution time, because Greedy is a more aggressive scheme that implicitly takes advantage of future reclaimed time.

References

[1] Daniel Mossé, Hakan Aydin, Bruce Childers and Rami Melhem. Compiler-Assisted Dynamic power-Aware Scheduling for Real-Time Applications. *Work-*

shop on Compilers and Operating Systems for Low Power, COLP00. 2000.

[2] I. Hong, G. Qu, M. Potkonjak and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Dec 1998.

[3] C. M. Krishna and Y. H. Lee. Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D.C., May 2000.

[4] Transmeta Corporation, Crusoe Processor Specification, http://www.transmeta.com

[5] T. Burd, T. Pering, A. Stratakos, R. Brodersen. Dynamic Voltage Scaled Microprocessor System. *International Solid-State Circuits Conference, ISSCC 2000*, pp.294-295, 2000.

[6] R. Min, T. Furrer, and A. Chandrakasan. Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks. *IEEE VLSI Workshop*, 2000.

[7] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the lpARM Microprocessor System. *International Symposium on Low Power Electronics and Design 2000*, pp.96-101, 2000.

[8] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Design Automation Conference, DAC'99*, pp. 134-139.

[9] J. Pouwelse, K. Langendoen and H. Sips. Dynamic Voltage Scaling on a Low-Power Microprocessor. In *Mobile Computing Conference (MOBICOM)*, 2001.

[10] D. Burger, and T. Austin. The SimpleScalar Tool Set, Ver 2.0 . *University of Winsconsin-Madison CS Technical report no. 1342*, 1997.

[11] Chung-Hsing Hsu, Ulrich Kremer, and Michael Hsiac. Compiler-Directed
Dynamic Voltage/Frequency Scheduling for Energy Reduction in Microprocessors. *International Symposium on Low Power Electronics and Design*, August 2001.

[12] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and Aggressive Power-Aware Scheduling Techniques for Hard Real-Time Systems. *To appear in 2001 IEEE Real-Time Systems Symposium (RTSS'01)*, December 2001.