

Workload-Aware Dual-Speed Dynamic Voltage Scaling

Dinesh Rajan, Russell Zuck and Christian Poellabauer

Department of Computer Science and Engineering, University of Notre Dame
{dpandiar, rzuck, cpoellab}@cse.nd.edu

Abstract

Dynamic voltage scaling (DVS) is a frequently used technique in mobile and embedded systems, aimed at reducing the energy consumption of mobile processors. In systems with a discrete number of frequency levels, existing dual-speed DVS approaches compute an optimal theoretical CPU speed and approximate it by choosing the two neighboring discrete speed levels. By comparing experimentally the energy savings attained with different frequency combinations on a mobile platform, this work shows that choosing the two neighboring frequency levels does not necessarily yield the highest energy savings. As a result of the above observation, this work introduces an online approach to dual-speed DVS that a) formulates a model for speed selection based on the workload characteristics of the current task set, b) computes a frequency pair that yields the best possible energy savings for a given taskset and workload.

1 Introduction

In recent years, the use of mobile devices has become highly integrated into the daily lives of their users. However, a fundamental deficiency of these devices is their short battery lives, which demands the use of efficient energy and power management techniques such as power-aware scheduling policies [4], low-power modes for system resources [3]. One common energy management technique is that of *Dynamic Voltage Scaling (DVS)* which takes advantage of the fact that the processor power consumption is proportional to the product of frequency and the square of voltage. Substantial power savings can thus be realized by modulating the voltage and frequency combination of the processor [1]. Essentially, DVS algorithms attempt to maintain CPU utilization close to 100%, while taking into account the timing constraints of the processes being run on the device at any given time.

This paper investigates the use of DVS on a mobile platform based on the XScale PXA255 processor, which supports a limited number of discrete frequency levels. With such processors that support only a limited number of operating frequency levels, there can be a significant difference between the desired and the actual operating frequency which would

translate directly to a loss in potential energy savings. Dual-speed approaches to DVS address this limitation by calculating an optimal theoretical frequency setting and choosing two discrete speed settings (e.g., the two neighboring frequency levels) to approximate the theoretical result. Through our experiments, we compare the energy savings of different frequency combinations under different workloads, showing that the selection of the speed pair depends on the current workload characteristics. Thus a strategy considering only the two neighboring frequency levels does not always translate to the best possible approach yielding the highest energy savings. This paper introduces an *online* approach to dual-speed DVS that bases speed selection on the workload characteristics of the current task set and frequently recalculates the frequency pair selection to ensure maximum energy savings. To summarize, the main contributions of this work are 1) the insight that the selection of the optimal frequency pair depends on the type of workload the task set represents (CPU- versus I/O-intensive) and 2) the design of an online dual-speed DVS algorithm that recomputes the optimal frequency pair whenever the task set or workload changes.

2 Related Work

In [2, 5, 9], various DVS approaches that considered the effect of external memory accesses in computing a single optimal frequency setting were implemented. Our work extends on these efforts by employing a dual-speed approach to achieve the optimum frequency level. We propose an approach that exploits existing DVS techniques by dynamically computing two voltage-frequency combinations and a *ratio* of the run-times between these two states (determining how long the processor has to operate in each state). The key novelty of our work however is that we employ *dynamic feedback* from the current utilization and resource usage of tasks such as cache miss rates, to dynamically adjust the run-time ratio and, if required, the frequency-voltage settings (as opposed to previous static or compile-time solutions [7, 8]).

3 Dual-Speed Dynamic Voltage Scaling

With the dual-speed approach to DVS, *two* CPU speeds are computed, and the CPU is switched between these two

discrete speeds such that the optimal theoretical CPU speed is closely approximated. This will be the basis for our *workload-aware* DVS approach introduced in this paper.

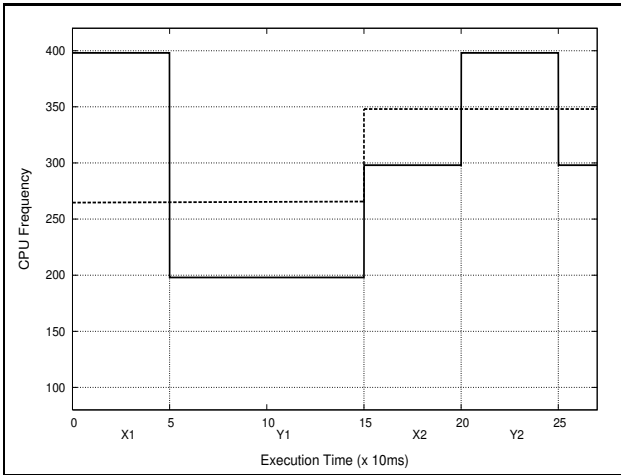


Figure 1. A dual-speed approach to Dynamic Voltage Scaling

To illustrate the performance benefits of a dual-speed approach, let us consider an example using Figure 1. Assume that the frequency computed to provide optimum energy savings in this example is 260MHz. The dotted line in the figure represents this optimum frequency level that changes dynamically (to 350MHz at time=15ms) with the utilization of the system. With a single speed approach, the processor would then need to be run at the higher frequency of 298MHz since there is no available supported frequency at 260MHz. Such an execution at a frequency higher than the optimum computed frequency often leads to a significant reduction in energy savings. An improvement to this approach would be to run between the two adjacent frequency levels (198MHz, 298MHz) so as to achieve a frequency closer to the optimum frequency of 260MHz.

While the adjacent frequency levels seem to be a logical choice for a dual-speed DVS approach, the remainder of the paper shows that different combinations of two discrete (f, V) -levels can be used to achieve the same average speed, however with different energy savings. Furthermore, the optimal combination of P-states (i.e., (f, V) -levels) can change during run-time based on changes in utilization as well as the type of workload experienced (i.e., memory or I/O-intensive versus processing-intensive tasks). In our example above, different energy savings could be achieved by executing between either of the frequency combinations of (99, 298)MHz, (99, 398)MHz, (198, 298)MHz or (198, 398)MHz. Our approach would dynamically choose one such combination, depending on the current utilization and workload, that would yield optimum energy savings. The dark lines in the figure represent the frequency combinations chosen by our approach.

4 Experimental Study

All our experiments and analysis were made on a Intel Sit-sang evaluation board based on the PXA255 XScale processor. The platform supports four discrete frequencies in the range of 99MHz to 398MHz. The operating system used on the platform is a Linux 2.4.19 kernel specially ported for this architecture.

For each supported clock frequency f_n , a *scaling factor* k_n can be obtained by executing a sample processing-intensive code at both the default frequency (maximum) f_{max} and f_n and dividing their measured run-times C_n and C_{max} . So we have: $k_n = C_n/C_{max}$. This is repeated for each available clock frequency for a given processor. The goal of frequency scaling is to get as close to 100% utilization as possible, i.e.,

$$U_{100\%} = \sum \frac{C'_i * k'}{T_i} \quad (1)$$

where k' is the yet unknown scaling factor, T_i is the period of task i and C'_i represents the actual measured service time for the task in its most recent invocation. To guarantee that best-effort tasks are not starved, we can replace $U_{100\%}$ with a relaxed value of U_x with $x < 100\%$. The value of k' can then be determined with: $k' = \frac{U_x}{\sum \frac{C'_i}{T_i}}$.

The resulting k' is compared to the experimentally computed values for scaling factor (k_n), and the scaling factor k_n closest to k' ($k_n \leq k'$) is selected. The clock frequency is then adjusted to the frequency f_n ($f_n \geq f'$). Note that the focus of our work is on aggressive energy conservation, i.e., instead of worst-case execution times, the proposed approach predicts actual execution times, where the accuracy of such predictions determines the amount of energy savings and the number of deadlines achieved.

While all types of resource access can affect the selection of frequency-voltage pairs, we primarily focus on memory accesses in this paper. The Memory Access Rate (MAR) is used in our experiments as a suitable representation of the system workload (memory-intensive versus processing-intensive). The MAR is defined as the ratio of the total number of cache misses to the number of instructions executed [5]:

$$MAR = \frac{\text{Data Cache Misses}}{\text{Number of Instructions Executed}} \quad (2)$$

In our work, we use a register to track the number of executed instructions while another register is used to keep count of cache misses. Based on the dynamic measurements of MAR, we use the following model for the prediction of future task run times [5]:

$$t_{run_time} = C_{CPU}(f_{CPU}) + (MAR * C_{bus}(f_{bus})) \quad (3)$$

This model considers a program's execution time by breaking it down to two components: a) the time spent executing instructions and b) the time spent stalled on memory accesses.

The factors $C_{CPU}(f_{CPU})$ and $C_{bus}(f_{bus})$ are constant values that are dependent on the CPU clock and bus frequency respectively. These values were computed using a test program capable of generating specified cache miss rates.

Figure 2 compares the energy consumption and runtime of the Sitsang platform for the four supported frequency levels.

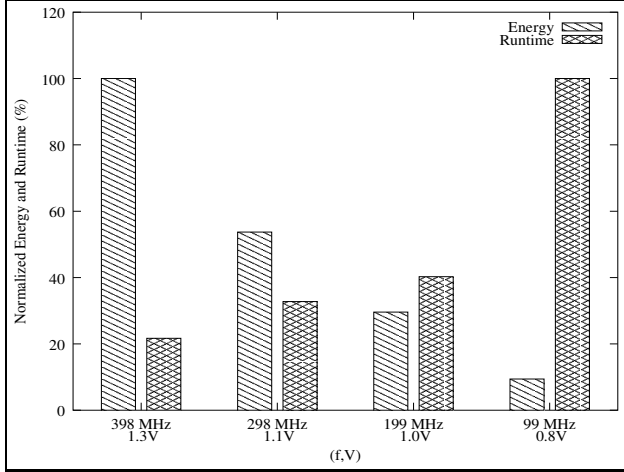


Figure 2. The energy consumption in percentage normalized to the default setting of (398MHz,1.3V) and the application runtime in percentage normalized to the slowest setting of (99MHz,0.8V).

From the utilization-based approach described above, the scaling factor k' can range typically from 1.0 to 100.0 and is interpreted as the degree to which the system is under-utilized (i.e. a larger k' represents a more under-utilized system). Figures 3-5 depict the normalized energy consumptions of the XScale processor (normalized to the default speed of (398MHz,1.3V) for various scaling factors. Figure 3 represents the energy consumption when there are no memory accesses being made, while Figures 4 and 5 represent the energy consumed under two different rates of memory accesses. Each of the regions depicted in the figures represent the range between two supported operating frequencies. So, region 1 represents the range between the two lowest frequencies 99 & 199MHz. Regions 2 and 3 map the ranges above the second lowest frequency, that is between 199 & 298 MHz and 298 & 398MHz respectively. The energy consumption presented here is measured for all possible combinations that can be utilized to achieve the desired computed frequency level that lies within a particular region, by executing between these combinations. The graphs are plotted against the proportion of run-times that are normalized to the run-time at the lower frequency. For example, consider Figure 3. Region 1, as discussed earlier, represents the energy consumption for all those frequency combinations that can be used to achieve a frequency level between 99MHz and 198MHz. The region

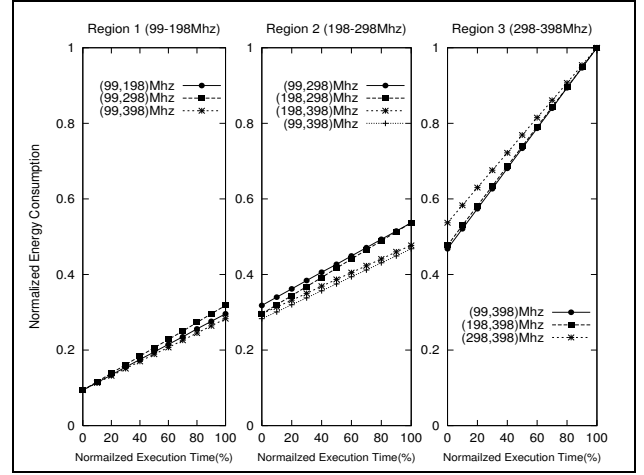


Figure 3. Comparison of different E-states for processing-intensive tasks.

1 has three plots that represent the possible frequency combinations that can be used to achieve any desired frequency between 99 and 198MHz. A value of 10% on the x-axis corresponds to running the task at the higher speed of a particular combination for 10% of its total execution time and 90% of the remaining time at the lower speed in that combination.

It can be observed from these graphs (Figures 3-5) that for a given region, running between its two end-point frequencies does not always achieve the maximum possible energy savings. As an example consider, Figure 3, where for region 2 we see that execution between 99 & 398MHz offers more energy savings when compared to running between the two end frequencies of the region - 198 & 298MHz. Typically, many DVS approaches would select to run at the next highest frequency level (298MHz) when their computed optimum frequency falls within this region. On the other hand, more fine-grained approaches would switch executions between 198 & 298 MHz to achieve the optimum frequency. As can be inferred from the graphs, both approaches would not be able to achieve maximum possible energy savings.

It is also to be observed that the energy consumed by each of the possible frequency combinations varies significantly with the rate of memory accesses. For the same example considering region 2, we see that running between 198 & 398MHz for tasks that access memory (frequently/infrequently) would result in minimum energy consumption thereby achieving higher energy savings.

The reason for these changes are that memory accesses of memory-bound applications introduce significant latencies into the application run-times, and where these additional penalties differ for each of the possible frequency combinations.

Thus from the above data, we make the conclusion that an optimal DVS algorithm needs to consider the load of a system in selecting a frequency-voltage setting for optimum energy

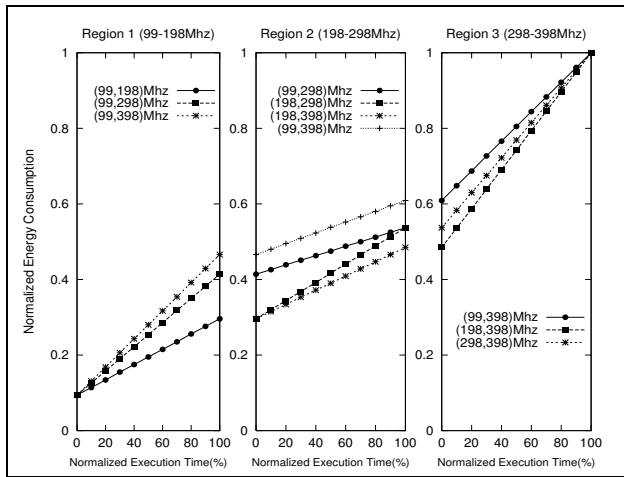


Figure 4. Comparison of different E-states tasks with infrequent memory accesses.

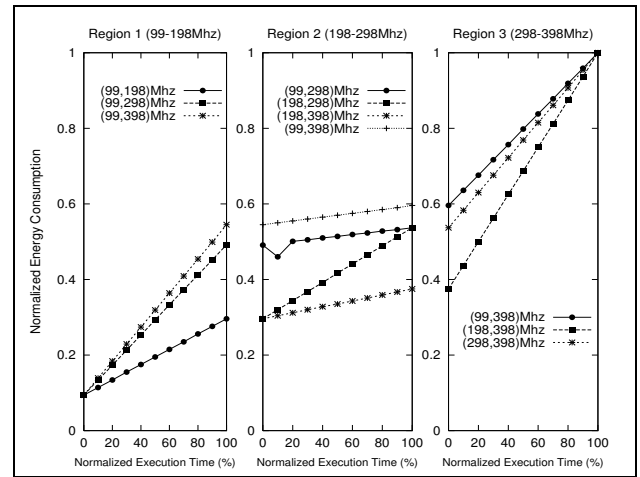


Figure 5. Comparison of different E-states for tasks with frequent memory accesses.

savings. In the next section we introduce an online feedback-based DVS approach to achieve this goal.

5 Implementation

As discussed above, a Dynamic Voltage Scaling approach should always incorporate parameters representing the current workload of the system in its computations determining an optimum voltage-frequency level for execution. For our approach implementing a dual-speed mechanism, we employ feedback from the special purpose registers providing information about the current memory access rates (MAR) to derive the current workload of the system. The implemented algorithm computes a pair of frequency-voltage combinations and a ratio of run times between them whenever there is a change in the utilization and/or the memory access rate. The utilization for a given task set is calculated as described in Equation 1, with execution times predicted as given in Equation 3. The algorithm is executed every time the CPU scheduler is run. A detailed explanation of the implemented mechanism can be found in [6].

6 Summary and Conclusion

In this work, the drawbacks of single-speed approaches to DVS and those that do not consider the current workload of the task set were discussed and analyzed. Based on our analysis, we proposed and implemented a dynamic workload-aware voltage scaling algorithm that employed a dual-speed approach. The dual-speed scheme presented always yielded the maximum energy savings by selecting the optimum frequency-voltage combination based on the current workload characteristics derived from the MAR value.

Our future work will extend the efforts presented in this paper to include other device components capable of energy

management such as the I/O devices of storage, network cards which typically offer low-power idle/sleep modes.

References

- [1] T. Burd and R. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proc of the 28th Annual Hawaii International Conference on System Sciences.*, January 1995.
- [2] K. Choi, R. Soma, and M. Pedram. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In *Proc. of the International Symposium on Low Power Electronics and Design (ISLPED)*, August 2004.
- [3] D. Helmbold, D. Long, and B. Sherrod. A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proc. of the Intl. Conference on Mobile Computing and Networking*, 1996.
- [4] P. Mejia-Alvarez, E. Levner, and D. Mosse. Power-Optimized Scheduling Server for Real-Time Tasks. In *Proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, September 2002.
- [5] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005.
- [6] D. Rajan, R. Zuck, and C. Poellabauer. A Dual Speed Approach to Workload-Aware Voltage Scaling. *Technical Report TR-2006-05*, University of Notre Dame, Available at http://www.cse.nd.edu/research/tech_reports.
- [7] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-Aware Static Timing Analysis. In *Proc. of the 24th IEEE International Real-Time Systems Symposium (RTSS)*, 2003.
- [8] F. Xie, M. Martonosi, and S. Malik. Compile-time Dynamic Voltage Scaling Settings: Opportunities and Limits. In *Proc. of the ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI'03)*, June 2003.
- [9] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Hardware-assisted Asynchronous Dynamic Voltage Scaling. *SIGPLAN Not.*, 40(7):203–212, 2005.