Processor Frequency Setting for Energy Minimization of Streaming Multimedia Application

Andrea Acquaviva
DEIS - University of Bologna
V.le Risorgimento 2
Bologna, Italy
aacquaviva@deis.unibo.it

Luca Benini
DEIS - University of Bologna
V.le Risorgimento 2
Bologna, Italy
lbenini@deis.unibo.it

Bruno Riccò
DEIS - University of Bologna
V.le Risorgimento 2
Bologna, Italy
bricco@deis.unibo.it

ABSTRACT

In this paper, we describe a software-controlled approach for adaptively minimizing energy in embedded systems for real-time multimedia processing. Energy is optimized by clock speed setting: the software controller dynamically adjusts processor clock speed to the frame rate requirements of the incoming multimedia stream. The speed-setting policy is based on a system model that correlates clock speed with best-case, average-case and worst-case sustainable frame rate, accounting for data-dependency in multimedia streams. Experiments on an MP3 decoding application show that computational energy can be drastically reduced with respect to fixed-frequency operation.

1. INTRODUCTION

One of the most critical constraints on portable embedded systems is power consumption, that is directly linked to battery size, weight and lifetime, and impacts system cost and reliability. In the design of embedded systems, a microprocessor-based architecture is often a forced choice because of its flexibility and fast time-to-market. In these architectures, the CPU must handle a large fraction of the computational load imposed by applications and it is a major contributor to the power budget. In general, CPU energy consumption depends on the type of workload imposed by applications. We focus on ultra-portable embedded devices targeted to streaming multimedia applications, such as audio and video decoding.

An algorithm can dynamically reconfigure the system to provide the required services and performance levels in a power-efficient way. Modern hardware components provide a large freedom in dynamically adjusting important power-correlated parameters such as clock frequency and supply voltage allowing quick adaptation also at run time. Hence, it is possible to reduce power consumption by adjusting system speed and supply voltage at the minimum level necessary to match real-time constraints. Algorithmic power optimization is not a new concept. In [5], Chandrakasan et al. explore dynamic voltage setting in DSP with a variable workload. Differently from custom DSP, general purpose SOCs are not targeted to a particular application, then an adaptation is necessary even with a fixed workload, in order to reconfigure the hardware resources in a power-efficient way. The work by Chandrakasan et al. demonstrate the effectiveness of decreasing together speed and voltage with respect to simply shut down the system in idle periods.

Voltage regulation is seen as a generalization of shutdown where the voltage levels are quantized in more than two values (*on* and *off*); from this point of view, variable voltage allows better adaptation to different workloads. Recently, Sinha in [10] has investigated the idea of applications that are aware of their power requirements and help the operating system in taking decisions about resources to be allocated, clock frequency and supply voltage. The energy model by which the actual values of voltage and frequency are derived, however, assumes a linear relationship between clock frequency of the processor core and execution time of a certain task. This model is not suitable in the context of real time processing, and in general does not take into account real-life systems bottlenecks like memory latency.

In [6], the authors take a dual approach: here the algorithm adapts its requirements to resource availability. The same work also shows that a large amount of power is spent by the processor when in idle state because of limited network bandwidth or real-time synchronization requirements. The assumption at the basis of the variable voltage power management techniques is that power consumption scales down with k^3, where k is the voltage and speed scaling factor [4]. In addition, being the execution time inversely proportional to k, energy depends on the square of the supply voltage and not on the clock frequency, so is not useful to change only processor speed in order to save energy, unless supply voltage is scaled down as well.

On the contrary, recent results [1][3][8][9] on real systems have demonstrated that running at less than the maximum frequency can be advantageous. In the following section we provide a theoretical explanation for this fact, which motivates the adaptive algorithmic power optimization strategy presented later in the paper. In such a context, this work analyzes the power-performance trade-off of multime-

dia applications running on embedded processors, to obtain a characterization and a methodology to determine processor speed which allows energy saving while satisfying real-time constraints.

2. MULTIMEDIA SYSTEM MODEL

In multimedia stream processing, data come into the system from the environment through a wireless network or a wired link from a host computer. The main processing unit is a general-purpose microprocessor, integrated in a SOC architecture. The CPU processes a block of data and sends the results to the output interface when finished. To improve efficiency and parallelism, current SOCs contain input-output hardware units that can buffer data and manage standard communication channels (e.g. serial port, parallel port) in parallel with the CPU. High-bandwidth input-output devices often use DMA for enhanced performance. Our target system is based on the StrongARM1100 core [2]. StrongARM1100 implements several power management capabilities. For example, we can adjust the frequency in a range of discrete values or we can stop the clock for some components also at run-time. Frequency can be programmed via software by writing a control word in a processor register. In StrongARM1100 twelve frequency levels are available by programming a PLL.

Multimedia stream processing algorithms take as an input a stream of encoded with a rate established by the input channel bandwidth. The input stream is generally structured in frames. Each frame is processed by the CPU which extract the encoded information and forward it to the output channel. The processing speed of the CPU and the synchronization framework with the I/O drivers must guarantee the output data rate required to mach real time constraints.

3. VARIABLE FREQUENCY AND ENERGY OPTIMIZATION

In this section we discuss how energy reduction can be obtained by setting clock frequency even in the absence of an associated voltage regulation. This is in contrast with the common assumption that speed-setting is effective only accompanied by an adequate voltage-setting policy. Of course, if voltage is scaled with frequency, more power can be saved, but the point here is that this is not a forced choice. To demonstrate our claim, we start by looking at the usual expression of dynamic power consumption:

$$P = V_{DD}^2 \cdot C_{eff} \cdot f \tag{1}$$

(where V_{DD}^2 is the supply voltage, C_{eff} is the average switched capacitance, and f is the CPU clock frequency). Because multimedia streams have a frame-based structure, it is useful to consider the frame processing time T_{frame}. The energy consumption in time T_{frame} can be immediately obtained as:

$$E_{frame} = V_{DD}^2 \cdot C_{eff} \cdot f \cdot T_{frame} \tag{2}$$

where T_{frame} is the frame processing time. $T_{frame} = N_{frame} \cdot t$, where N_{frame} and t are the number of clock cycle necessary to elaborate a frame, and the cycle time respectively. We have then:

$$E_{frame} = C_{eff} \cdot V^2 \cdot N_{frame} \tag{3}$$

because $f = 1/t$. E_{frame} depends on N_{frame}, i.e., on the workload. Now, because of the CPU must interface with external hardware which is in general slower (ex: off-chip memories), there are times in which the CPU is idle, so let us now express N_{frame} as:

$$N_{frame} = N_{useful} + N_{idle} \tag{4}$$

We conservatively assume that N_{useful} (the number of cycles spent in execution useful operations) is fixed for a given algorithm, while N_{idle} (the number of cycles wasted with the CPU being idle) can be seen as a function $N_{idle}(f)$ where f is one of the available processor frequencies. N_{idle} is a non-decreasing function of f.

N_{idle} identifies CPU idleness finely dispersed among useful operations (mainly during memory wait cycles on cache misses). This term varies with $f$since memory access time is fixed, adjusting the frequency involves variation in number of wait states in a bus cycle. This happens when the CPU is not the speed limiting element.

Thus:

$$E_{frame} = V_{DD} \cdot C_{eff} \cdot (N_{useful} + N_{idle}(f)) \tag{5}$$

Now it must be considered that real time algorithm have the constraint of provide a minimum amount of data output in a given time T_{max} depending on the output bandwidth required. For example in audio MPEG decoding, this bandwidth depends on the sample rate of the decoded sound. The output bandwidth is directly proportional to the frame elaboration speed, that can be expressed as $1/T_{frame}$. To keep the real time constraints, this speed must be greater than $1/T_{max}$.

The target of power optimization is to reduce E_{frame} acting on N_{idle}, under the constraint:

$$\frac{f}{N_{useful} + N_{idle}} \geq \frac{1}{T_{max}} \tag{6}$$

Speed-setting pursues the target of power minimization by decreasing f so that in (6) N_{idle} decreases while N_{useful} and T_{max} are fixed, because they affect the output data bandwidth. The lower bound of f, namely f_{opt} is fixed by the requirement that all the useful work be executed in T_{max} (just-in-time computation).

From the expression of T_{frame}, it can be observed that increasing f by a certain factor

Speed-setting effectiveness depends on the workload characteristics and the system's architecture (both hardware and software). It reduces the costs of memory latency in terms of CPU wait states, hence, in execution dominated by memory access (high miss rate), and where memory latency is higher, this technique is more effective [3]. In addition, from a system energy perspective, since the CPU clock often feeds other on-chip components, additional system power can be saved by reducing useless work on these as well (even if in some cases they implement power down and gated clock strategies).

On the other hand, since the frequency cannot be adjusted continuously, it is hard to completely eliminate CPU idleness. As an additional concern when evaluating the effectiveness of a speed-setting policy, the delay and the energy spent to set the processor speed must be considered. However, in the context of the proposed algorithm, this penalty is not noticeable because the appropriate frequency value is choosen at the beginning of the stream.

4. IMPACT OF CLOCK FREQUENCY ON PERFORMANCE

As previously mentioned, the effectiveness of a speed setting policy depends on the hardware characteristics and on the workload. Therefore a characterization of the system performance as a function of clock frequency is needed in order to choose the speed which guarantees the level of performance required.

Multimedia processing is carried out on a frame by frame basis, every iteration yielding a variable amount of output data. Since real-time applications must produce a fixed amount of output data in a given period of time, the frame processing rate $FR(f)$ is an appropriate metric to indicate the level of performance supported by the system at a given clock frequency f. The main challenge in modeling the frame rate as a function of clock frequency is that it depends on the characteristics of the multimedia stream as well. In general, we have $FR(f, s, d)$, where s is a parameter representing characteristics of the entire stream, namely the sample rate sr and the bit rate br, and d represents characteristics of a single frame in the stream (e.g., frame size). For MP3 audio, the achievable FR at a given clock frequency is a strong function of the stream's bit rate and sample rate.

For a fixed $br = br^*$, and $sr = sr^*$, frame-by-frame variations are quite small, but non-negligible. To build our performance model, we analyze several streams at br^* bit rate and sr^* sample rate, and we monitor the worst-case frame processing time, as well as the best-case frame processing time at various frequencies. Then, we define three curves $FR_B(f)$, $FR_A(f)$, $FR_W(f)$, representing best case frame rate (i.e., the frame rate that could be achieved if all frames in the stream could be processed at max speed), the average case frame rate, and the worst-case frame rate, respectively. By construction, $FR_B(f) \geq FR_A(f) \geq FR_W(f)$. The three curves are normalized with respect to $FR_A(f_{MAX})$, the average frame rate achieved when the processor is run at maximum speed. All three curves are monotonically increasing in frequency. The normalized $FR_A(f)$ has maximum value 1.

The same process is repeated for several different values of br and sr, including all corner cases (i.e., maximum and minimum br and sr in a range of allowed values). The normalized curves are plotted on the same $FR \times f$ plane. We then obtain three normalized curves: the *overall best* $FR^o_B(f)$, the *overall average* $FR^o_A(f)$ and the *overall worst* $FR^o_W(f)$. The first one is obtained by selecting the largest FR value among all FR_B curves for each frequency point. The second one is obtained by averaging all values of FR_A curves at every frequency. The third one is obtained by selecting the smallest FR value among all FR_W curves for each frequency point. The curves $FR^o_B(f)$, $FR^o_B(f)$ and $FR^o_W(f)$ are the performance model for the system.

	16KHz	24KHz
16KBit/s	65.36	69.67
32KBit/s	63.25	67.95
64KBit/s	60.61	66.56

Table 1: Frame Rate look-up table: the values represent $FR^o_A(f_{max})$ in $frame/sec$ at different sample rate (row) and bit rate (column).

The frequency setting algorithm exploits the knowledge of $FR^o_B(f)$, $FR^o_B(f)$ and $FR^o_W(f)$ curves, as well as the knowledge of the $FR_A(f_{MAX})$ for each allowed combination of br and sr. It can be summarized as follows: when stream decoding begins, the algorithms extracts the br and sr information from the stream header, and looks up the corresponding value of $FR_A(f_{MAX})$ in the table shown in figure 1. Furthermore, given the sr value, it is possible to determine what is the average frame rate FR that the system must support to guarantee real-time playback of decompressed audio by the following equation:

$$FR_{req} = \frac{sr}{N_{samples}} \tag{7}$$

where $N_{samples}$ is the number of samples per frame [1].

Given a FR requirement, the frequencies f_{min}, f_{av}, f_{max} are computed by intersecting the curves $FR^o_B(f)$, $FR^o_A(f)$, $FR^o_W(f)$ with the horizontal line $FR_{req} = FR/FR_A(f_{MAX})$, and finding the abscissas of the intersections, as shown in Figure 2. Frequencies f_{min}, f_{av}, f_{max} define a range of allowed frequencies for speed setting. Running the processor at f_{av} should be enough to provide real-time playback, but some buffering is required to accommodate frame decoding rate jitter. Alternatively, it is possible to run the processor at f_{max}. At this frequency, real-time performance is guaranteed on a frame-by-frame basis, with minimal jitter compensation buffering. However, the processor consumes more energy than what is needed most of the time. Finally, f_{min} frequency can be used for short periods of time if we find out that frames are consistently processed faster than the average rate. Clearly, it is also possible to run the processor faster than f_{max} (if f_{max} is smaller than the maximum processor frequency). This is clearly sub-optimal from the energy viewpoint, but it can be a forced choice in systems where processor time is not fully dedicated to MP3 decoding. In this case, the performance model described above makes it possible to quantify the fraction of processor time that is made available for other tasks by clocking the processor at $f > f_{max}$.

It is important to stress that the $FR(f)$ curves are not linear in general. This is because the memory system and interfaces do not speed up like the processor with increasing clock frequency. Increasing f leads to a decrease of the ratio N_{useful}/N_{idle}, therefore the frame rate does not increase linearly with f. The slower the speed of the external hardware

[1] This number is fixed to 576 for MPEG1, and to 576 for MPEG1 phase2

(e.g., memory access time), with respect to the processor, the flatter the performance curve, and the greater can be the effectiveness of the speed-setting policy. Other than the hardware characteristics, the shape of the curve depends on the ratio between the computation time spent inside the CPU and that spent outside the CPU. Considering the non ideality of the external memory, this can be expressed also as the ratio between the external accesses and the total memory accesses, which in turns is equal to the cache miss rate. It must be observed that the minimum frame rate also takes into account the decrease in bandwidth caused by the synchronization of the processor with the I/O controllers and the external peripherals as well.

5. EXPERIMENTAL RESULTS

The results presented in this work are obtained for the system architecture of the HP SmartBadgeIII prototype handheld device [11], based on the StrongARM SA-1100 embedded core. The embedded application is MPEG-layerIII audio decoding. Performance and power are obtained using the simulation tools described in [11], which has been validated against hardware measurements. The first plot, in Figure 1, shows how energy per frame changes with clock frequency for two MP3 streams, with different br (same sr). The plot is obtained by running MP3 decode at the maximum frame rate achieved at a given clock frequency. This can be slower or faster than the one required for real time playback. The purpose of this plot is to show that energy per frame monotonically increases with frequency (contradicting the simplistic model where energy is constant with variable frequency), because the processor wastes energy waiting for slow memories during cache misses.

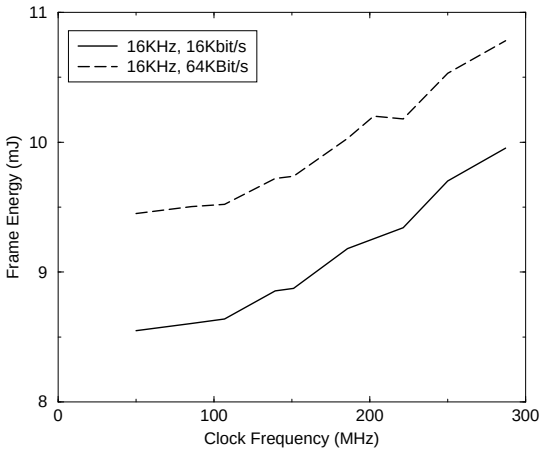

Figure 1: Energy consumption per frame

The actual energy penalty for excessive clock speed is even larger than what is shown in Figure 1, because active decoding must be stopped when the output buffer is full, to avoid frame loss. Even if we stop decoding by forcing the processor in idle state, power consumption in idle state is non-null (50mW). Hence, idle power is consumed when the processor is idle waiting for the output frame buffer to empty.

Figure 2 shows the overall FR vs. clock frequency curves, obtained with the procedure described in the previous section. All three curves are normalized on the y axis to the $FR_A(f_{MAX})$ value. This is all the information needed by the speed setting algorithm. A frame rate specification, set on the y axis, implies three frequency values, shown on the x axis. Remember that the tree curves do not depend on sr and br, hence they are a characteristic of the MP3 decode algorithm, and can be used with any MP3 stream, with the only caveat that the $FR_A(f_{MAX})$ must be available, and it must be pre-stored in a lookup table for every possible sr and br.

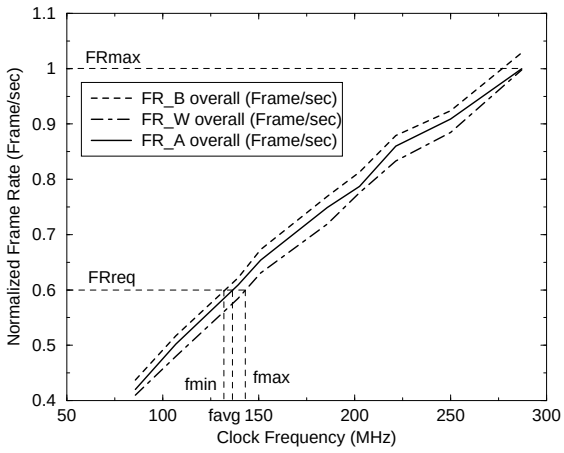

Figure 2: Frequency setting

Finally, Figure 3 shows the energy penalty paid when running the processor at a clock frequency larger than f_{min}. The solid line shows the energy overhead in the assumption that when the processor is idle it consumes negligible power. The dashed curve shows the actual power penalty, that accounts for processor idle power as well. Notice how the two curves diverge at higher frequencies, as the percentage of idle time becomes larger. Energy-per-frame savings of more than 40% are obtained with respect to the trivial policy that clocks the processors always at maximum speed. As an example, consider an audio stream with sr = 16KHz and br = 16KBit/sec. At the maximum frequency, and hence without optimization, the energy per frame results $10.989mJ$, as shown in figure 1. In order to apply our algorithm, first the frame rate required FR_{req} must be obtained directly by the knowledge of sr. In this case, from equation 7 it follows: $FR_{req} = 27.78frame/sec$. By looking in the look-up table with the appropriate value of sr and br the corresponding value of $FR_A(f_{max})$ = 65,36 frame/sec can be obtained, with f_{max} = 287MHz. This value is used to scale the normalized frame rate curve shown in figure 2. At that point, we find f_{min} = 85.7MHz, $f_{max} = 106.7MHz$ as abscissas corresponding to the ordinate FR_{req} in the plot of the scaled frame rate curve obtained above. In this case f_{avg} is either equal to f_{max} or f_{min} because there is not an allowed processor value between 85.7 and 106.7MHz. Looking at the energy plot of figure 3 it can be found that, if we conservatively choose f_{max}, the energy per frame results $8.64mJ$. Hence we obtain 21% of energy reduction. It must be noted that in this case f_{min} is lower than the minimum frequency indicated in the plot of figure 3 because the former refers to the best case frame rate indicated by the FR_B curve in figure 2.

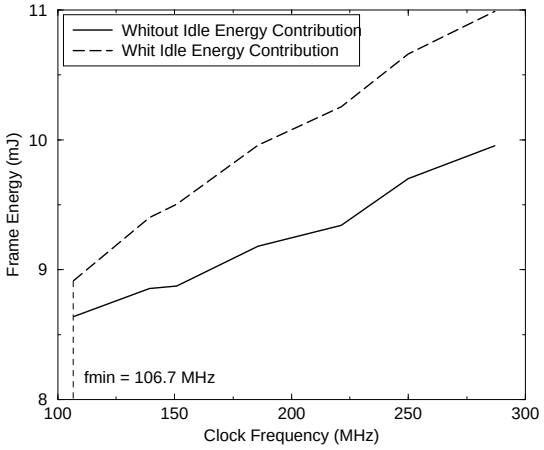

Figure 3: Energy Penalty for a 16KHz, 16Bit/sec audio stream

6. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced an approach for automatic run-time setting of the optimum processor frequency that minimizes energy for streaming MP3 audio decoding. The technique has been applied on an embedded portable appliance based on the StrongARM SA-1100 core, obtaining sizable energy-per-frame reduction. Adaptive speed setting is based on a performance vs. clock frequency model that is obtained by pre-characterization once for all for a given application. At run time, frame rate requirements are obtained by analyzing the MP3 steam header, and then a range of acceptable processor clock frequencies is automatically determined based on the performance model.

Future work in this area will focus on speed setting policies for embedded applications (such as MPEG2 video) where clock speed requirements change rapidly even within a single stream. Variable speed setting policies that take into account the impact of input and output buffering also warrants further investigation.

7. REFERENCES

[1] A. Acquaviva, L. Benini, B. Riccó, "An Adaptive Algorithm for Low-Power Streaming Multimedia Processing," *DATE*, in advance, March 2001.

[2] Advanced RISC Machines Ltd., Advanced RISC Machines Architectural Reference Manual, *Prentice Hall, New York*, July 1996.

[3] F. Bellosa, "Endurix: OS-Direct Throttling of Processor Activity for Dynamic Power Management," *Technical Report TR-I4-99-03, University of Erlangen*, June 1999.

[4] A. P. Chandrakasan, S. Sheng, R. W. Brodersen, "Low-Power CMOS Digital Design," *Journal of Solid State Circuits, IEEE*, Vol. 27 No. 4, pp. 473–484, April 1992.

[5] A. P. Chandrakasan, V. Gutnik, T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing," *ISLPED*, pp. 347–352, August 1996.

[6] M. Flinn, M. Satyanarayanan, "Energy-Aware Adaptation for Mobile Application" *ACM SOSP*, pp. 48–63, December 1996.

[7] T. Ishihara, H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processor," *ISLPED*, pp. 197–202, August 1998.

[8] T. L. Martin, D. P. Siewiorek, "The Impact of Battery Capacity and Memory Bandwidth on CPU Speed-Setting: A Case Study," *ISLPED*, pp. 200–205, August 1998.

[9] T. L. Martin, "Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing," *Ph.D. Thesis, Carnegie Mellon University*, August 1999.

[10] A. Sinha, A. P. Chandrakasan, "Energy Aware Software" *IEEE Int. Conference on VLSI Design*, pp. 50–55, January 2000.

[11] T. Simunic, L. Benini, G. De Micheli, " Cycle-Accurate Simulation of Energy Consumption in Embedded Systems," *IEEE Design Automation Conference*, pp. 867–872, June 1999.