This paper is already published at the Euromicro DSD 2016 conference

Dynamic hardware management of the H264/AVC encoder control structure using a framework for system scenarios

Yahya H. Yassin, Per Gunnar Kjeldsberg, Andrew Perkis
Department of Electronics and Telecommunications
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway, Email: yassin@ntnu.no

Francky Catthoor
IMEC
Kapeldreef 75
3000 Leuven, Belgium

***Abstract*—Many modern applications exhibit dynamic behavior, which can be exploited for reduced energy consumption. We employ a two-phase combined design-time/run-time methodology that identifies different run-time situations and clusters similar behaviors into system scenarios. This methodology is integrated with our framework for system scenario based designs, which dynamically tune the hardware to match the application behavior. We achieve significant energy reductions for an extracted control structure of a video codec widely used in hand-held devices today. We encode a video stream consisting of different frame sizes based on measured available wireless bandwidth. Energy consumption is measured with a modified microcontroller board from Atmel with two alternative voltage and frequency settings. While maintaining the perceptual video quality and frame rate, our method results in up to 44% energy reduction for our encoded streams, even after including an average worst-case tuning overhead of 8.5%. In reality this overhead is typically negligible since the worst-case assumes very frequent tuning, while in realistic situations it will be performed at a much lower rate. This makes the expected gain over 50%.**

I. Introduction

In embedded systems today, energy efficiency and power consumption is a major design constraint. Hand held devices are becoming more complex, because applications embedded in these devices require advanced functionality, in addition to increased parallelism. Many high end embedded systems today are more dynamic in nature, and conventional embedded system design techniques are not able to meet the requirements of today's demanding software functionality. State-of-the-art design methodologies try to cope with this challenge by identifying typical use cases and dealing with them separately, in order to reduce the increased complexity of the system [1]. Conventional techniques focus mainly on static analysis where no dynamism is present. When dynamism is introduced, then only the worst case behavior is used to perform the mapping. A gap exists between the high-level specification and the detailed implementation of embedded software design. A well balanced combined design-time/run-time two-phase methodology exist, and is known as the system scenario based design [2], [3]. This methodology consists of 5 design steps, and are described in Section II-A. We use this methodology in our framework for system scenarios (FSS) [4].

Reducing the power consumption does not necessarily result in energy efficiency. As an alternative, state-of-the-art techniques, such as race-to-halt (RTH) [5], are often used to speed up the execution of an application to achieve longer idle times. The static power consumption consumed with idle times is then reduced significantly due to the utilization of extremely low power states. RTH techniques reduces the total energy consumed by the system, and is considered state of the art for CPU bound workloads when the arrival time of the next set of tasks to be processed is unknown. According to Awan et al. [5], Dynamic Voltage and Frequency Scaling (DVFS) techniques are still more suitable for memory bound workloads and real time systems with deadlines and long idle times.

In general, multimedia applications consume significant amount of energy due to the stringent Quality of Service (QoS) requirements, as described by Hillestad [6]. Two-way interactive multimedia applications in particular have stringent requirements with respect to the quality of service delivered by the underlying transport network. Hillestad [6] generalizes two important requirements for IP-based streaming media applications. The first requirement is the real-time delivery to prevent buffer-underflow events, and the second requirement is smooth rate changes to prevent oscillations in perceived quality (e.g., quality as measured subjectively by end users). For instance, if the frame size of a streaming video changes with less than 10 seconds intervals, or if the next frame size is significantly larger than the previous frame, then the perceptual video quality may affect the end-user's experience.

Interactive multimedia applications normally require the use of advanced video codecs, such as the widely used H264/AVC video codec [7]. Implementing the H264/AVC video codec in hand-held devices is a challenge by itself because of the stringent requirements described by Hillestad [6] above, and because custom high performance digital signal processors (DSPs) are needed. These high perfor-

mance DSPs are in general more energy consuming than standard low-power of-the-shelf processors. On the other hand, general low power of-the-shelf processors struggle to process large video resolutions with acceptable processing times. Many researchers have therefore suggested multi-core solutions, where the H264/AVC codec is parallelized in order to concurrently process multiple frames or groups of pictures at the same time [8], [9].

Our goal is not only to improve the energy efficiency of the H264/AVC encoder, but generally improve the energy efficiency for data-dependent applications with similar behavior as the H264/AVC encoder. Therefore, we focus on the control structure of the H264/AVC encoder, as obtained from the Mediabench II website [10], on which we apply our framework for system scenarios (FSS) [4]. Our FSS framework is extended from only using Dynamic Frequency Scaling (DFS), to handling DVFS and RTH. The system scenarios approach is combined with RTH and DVFS on a SAM4L microcontroller board from Atmel [11], which contains an ARM Cortex M4 low power processor. In this paper we reduce the energy consumption by combining state-of-the-art techniques, i.e., DVFS and RTH, with the system scenario based design methodology [3] in our FSS framework. The experimentation is also substantially extended from our previous work [4].

In Section II-A we give a brief overview of the system scenarios based design methodology presented by Gheorghita et al. [3], and in Section II-B we highlight the importance of targeting data-variables. Related work is presented in Section III, and a motivational example is given in Section IV. Our experimental setup is explained in Section V, how we identify our system scenarios is discussed in Section VI, and the implementation of our prediction and switching mechanism is presented in Section VII. We present and discuss the energy consumption in Section VIII, and conclude our work in Section IX.

II. Definitions

A. The system scenario based design methodology

A two-phase design-time and run-time system scenario design methodology is presented by Gheorghita et al. [3]. It exploits the dynamic variation in an application, where system knobs, such as frequency, voltage, and low power modes, are tuned dynamically at run-time in order to gain significant reductions in energy consumption.

At design-time, different run-time situations (RTSs) are identified, usually through profiling. RTSs with similar costs, e.g., execution time, energy consumption, or memory requirement, are clustered into scenarios. These system scenarios are different from typical use case scenarios because they take detailed knowledge of the actual resource requirements into account, thus giving rise to more optimized designs [3].

A run-time scenario prediction mechanism is then determined at design-time, based on the output from the identification step. At run-time, the prediction mechanism determines whether or not to switch from one system scenario to another based on the actual system parameter values.

The exploitation step at design-time applies further optimizations to each scenario, similarly to what would have been done to the static implementation without the scenario approach. The system knob settings of each scenario are decided here. At run-time, the exploitation step is simply the execution of the scenario.

In order to change the system from one set of run-time platform configuration parameters (system knobs) to another, a scenario switching mechanism must be implemented. The switching mechanism to be used at run-time is determined at design-time. This mechanism will switch scenario based on the output of the prediction step at run-time.

An optional calibration configuration can be designed (at design-time) and used (at run-time) in order to fine-tune the decision making.

B. Control variables vs data variables

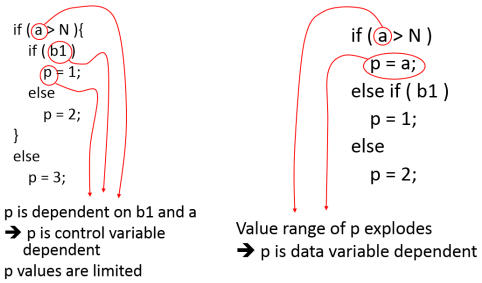

Fig. 1. Control vs data variable dependency

In order to better understand the challenge of data variable dependencies, Figure 1 illustrates a simplified code snippet illustrating the difference between a control variable and data variable dependency.

The code snippet on the left in Figure 1 illustrates a control variable dependency. The variable p in this case have a limited value range, and it will be relatively easy to monitor the application behavior if p is later used, for example, as a loop parameter.

The code snippet on the right shows a data variable dependent situation, where the value range of p equals the value range of a. If a is an input parameter, and p is used as a loop parameter in succeeding computations, then the loop size varies with the input parameter value range. Assuming the input parameter is a 16-bit integer value, then the value range of a will be equal to 2^{16}. This value range would simply produce too much run-time overhead for the scenario prediction and switching mechanisms.

III. Related work

Many researchers have optimized energy efficiency in embedded systems by proposing improved dynamic scheduling algorithms, such as Leakage Control EDF

(LC-EDF) and Leakage Control Dual Priority (LC-DP) for reducing leakage in hard real-time systems [12], 3-competitive offline and constant competitive ratio online algorithms for power saving while considering shutdown in combination with DVFS [13], and a combination of DVFS and shutdown to minimize the overall energy consumption based on the latest arrival time of jobs [14]. Most of these approaches produce significant analysis overhead. Awan et al. [5], propose a more efficient solution by combining an online and offline algorithm that selects the best feasible sleep mode based on the available time slack for a given workload with significantly less run-time overhead.

A method for automatic discovery of system scenarios that incorporate correlations between different parts of applications is proposed by Gheorghita et al. [15], [16]. When many-valued parameters are considered, the overhead for scenario prediction is significant, however. Hammari et al. [17] propose a method based on the projection of scenarios onto an RTS parameter space, where many-valued parameters are handled with the use of polyhedral techniques. These techniques are needed to fully automate the scenario analysis, which is not required for partly automated systems where the scenario information is inserted by the designer.

In general, the prediction mechanism is highly dependent on how the identification step is performed, as indicated by previous work [15], [16], [17]. The authors in previous work conclude that the number of variables used in the prediction should be low. This is also the same for other identification approaches, e.g., the method of using loop-trip count profiling [18].

Previous work mainly focuses on control structures in applications with limited or no data variable dependency. When data dependent RTS-parameters dominates the application behavior, the value range increases significantly. Increased value range in the RTS parameter space usually increases the run-time prediction overhead.

Damavandpeyma et al. [19] use DVFS as a system knob, and propose a compact scenario aware data flow graph (SADF). Their approach assumes that the system scenarios have been clustered as part of the identification step, where the clustered result does not contain any many-valued parameters, and the number of control- and/or data-variables used in their analysis is limited. Their approach works well for data dependent variables with limited dependency variation.

The data variable challenge increases with the combination of multiple data-dependent RTS parameters and nested loops.

We combine the FSS framework from our previous work [4] with DVFS and RTH to achieve significant energy reductions when data variable dependency is present. The data variable dependency is handled by using inline functions that combine multiple RTS parameters, and classify different ranges as scenarios. Low overhead and globally placed detection equations are defined that can detect the required cycle count of the application. The output of these detection equations are input to our prediction mechanism. The prediction mechanism then compares its input to different ranges in a lookup table, before deciding which scenario to choose. Our approach is scalable with increasing numbers of system knobs and an increased number of RTS-parameters within an application. In the following sections we will show how we were able to achieve significant energy reductions with our FSS framework implemented on a SAM4L microcontroller board.

IV. Motivational example

Hand-held video communication via the internet has become more common in recent years through the use of programs such as Skype, Viber, and Hangouts. A major challenge with hand-held devices is that the power consumption increases significantly during a video call. The phone's camera is used to encode video and stream it over the internet continuously. At the same time, video from the caller is received and decoded continuously. Considering that video is streamed with a rate of 24-30 frames per second, the amount of simultaneous encoding and decoding is significant. One of the commonly known challenges is that the user experience of a video call can vary based on the internet connection speed, the processing limitations of the hand-held device, and battery level. A popular and advanced codec such as H264/AVC is suitable for hand-held devices [7], and is considered as a state-of-the-art codec. The H264/AVC codec is far more advanced than its predecessors, and introduce more dynamism, especially with varying input frame sizes. This dynamicity can be exploited using the system scenarios design methodology presented by Gheorghita et al. [3].

There exist proposals for video resizers today, such as the "Apparatus and method for resizing an image" [20], which rescales a camera's video frame size. A simple control logic could easily be added to change the frame size of a picture, such that the dynamic behavior of the internet connectivity could be exploited and used to decide the scale factor of the video before it is encoded and transmitted.

Figure 2 shows the results of measurements we have performed of available internet bandwidth for a 500 second period on three different wireless networks. The upper graph shows the wireless local area network (WIFI) at NTNU, while the middle and lower graphs shows measurements over mobile Long-term Evolution (LTE) [21] and wideband code-division multiple-access (WCDMA) [22] networks, respectively. The measurements are performed using the bandwidth measurement tool IPERF [23]. IPERF is used to measure available bandwidth between two IP-addresses. Our measurements are performed with 10 seconds intervals between a Sony Xperia Z3 android-based smartphone and a computer running Ubuntu. Figure 2 clearly shows significant dynamicity in the available bandwidth, which we can exploit by using our

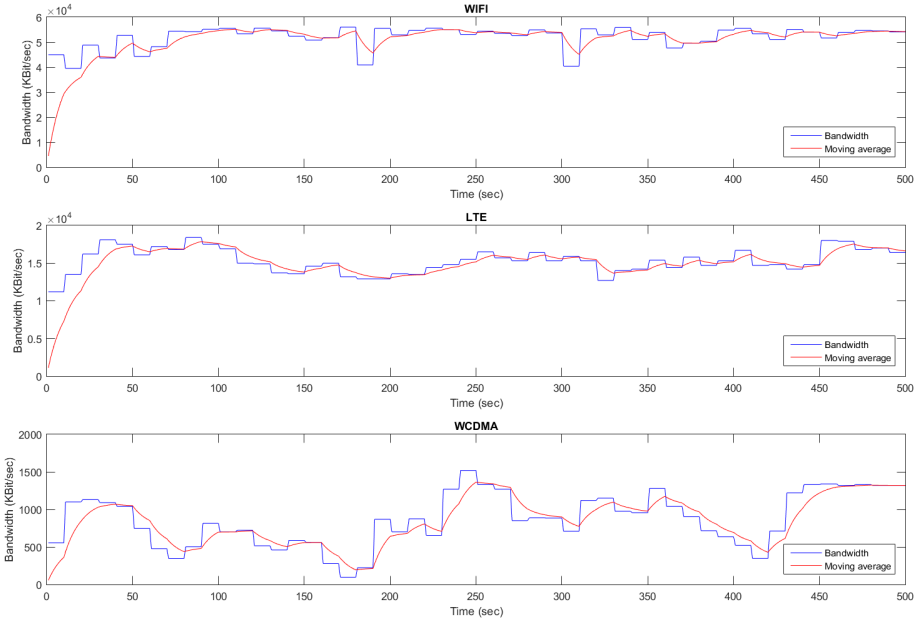

Fig. 2. Available bandwidth measurement and moving average over WIFI, LTE and WCDMA

FSS-framework [4]. In addition we calculate the moving average over the available bandwidth, as seen in Figure 2.

We focus on the H264/AVC encoder control structure obtained from the Mediabench II website [10], where we adapt the platform's voltage and frequency according to the input frame size, assuming that a video resizer circuit rescales the input frames based on the moving average measurement of the available bandwidth in Figure 2.

V. Experimental setup

The energy consumption of the H264/AVC encoder control structure is measured on a SAM4L board from Atmel [11], and we measure live current consumption and core voltage using a NI myDAQ measurement device [24]. Time stamped data is logged with a locally developed Labview program. The single-core FSS framework is used to implement the H264/AVC encoder control structure with system scenarios.

Our version of the SAM4L board contains the ATSAM4LC4CA microcontroller. The microcontroller contains an ARM Cortex M4 processor, and supports two different run modes. RUN0 operates with a core voltage of 1.8V, while RUN1 operates at 1.2V. The microcontroller also supports various low power states, i.e., sleep modes, where the processor clock is turned off, and multiple oscillators with different frequency ranges. For the scope of our research, we have selected the internal oscillators RCFAST (12 MHz) and RC80M (80 MHz) as the main clock sources for our DVFS settings. Since the ARM Cortex M4 processor only supports frequencies up to 48 MHz, the RC80M oscillator is clock divided to 40 MHz when it is selected as the main clock source. Hence, our DVFS settings are PS0 (1.8 V and 40 MHz) and PS1 (1.2 V and 12 MHz).

Based on the SAM4L board datasheet, the DVFS is triggered by a write to configuration registers after a write to protective lock registers. In other words, a switch is triggered after a few clock cycles. We have measured the switching time between our two clock settings, which take approximately 15 clock cycles when both oscillators are on continuously, and approximately 50 clock cycles if the currently unused oscillator is powered down. The added power consumption by having both oscillators enabled is negligible because unused oscillators that are idling does not contribute to any measurable power consumption. We have measured the total static power, of which the oscillators only contribute a minor part, to be less than 16% of the total power consumption in PS0, and less than 29% in PS1. In our scope, we trade a slightly increased static power for a significantly reduced switching time. The static power consumption on the SAM4L board is approximated by measuring the total power consumption for two different frequencies in both power modes, and estimating the effective capacitance under the assumption that the activity factor (α) is equal to 0.5 in Equation 1.

$$P_{\text{Total}} = P_{\text{Static}} + P_{\text{Dynamic}} = P_{\text{Static}} + \alpha C V^2 f \qquad (1)$$

When performing dynamic voltage scaling on the SAM4L board, the main clock frequency supported in the new voltage level has to be configured first. According to the datasheet, the 1.2 V (PS1) configuration does not support frequencies larger than 12 MHz. The frequency setting must therefore be switched from 40 MHz to 12 MHz before the voltage switch from 1.8 V (PS0) to 1.2

V (PS1). When changing configuration from PS1 to PS0 the frequency must be scaled after the voltage switch to maintain stability.

In addition, the voltage output from the voltage regulator has to be stable after a voltage switch before the CPU can continue processing. After a switch from PS1 to PS0 we have to wait for a status flag which signals that the voltage level has stabilized. The CPU is therefore stalled while waiting for this flag to be set in the SAM4L board. However, when a voltage switch is triggered from PS0 to PS1 then waiting for this flag is not necessary before continuing the CPU execution. As described by Atmel Support, when switching from a higher voltage level to a lower level, the voltage regulator is stable and will regulate fast enough to continue execution in succeeding cycles. The measured current consumption will gradually decrease after this switch until the charge on the decoupling capacitor connected to the core voltage has been consumed down to its PS1 level. The power delivered from the regulator on the other hand is equal to the PS1 level immediately after the voltage switch.

Our SAM4L board came initially with a 22 μF decoupling capacitor connected between the core voltage input and ground. The large decoupling capacitor resulted in extremely large switching times. This was confirmed by Atmel Support, and they recommended a change to 4.7 μF to guarantee that all modules would function properly. Since we are only interested in the CPU, we measured the DVFS switching times with lower decoupling capacitors. Our experiments showed that the CPU functioned correctly using a 1 μF decoupling capacitor, giving a DVFS switching time comparable to state-of-the-art processors. Table I shows the switching times and energy overhead when switching from PS0 to PS1 and vice versa. The energy overhead is measured by measuring the time between the last instruction before and the next instruction after the DVFS switch, before multiplying this time with the power consumption for the run mode before the switch. An oscilloscope was used for the timing measurement.

TABLE I
DVFS switching times between PS0 and PS1.

From	To	Cycles @ 12 MHz	Switching time (μS)	Overhead Energy (μJ)
PS1	PS0	197	16.40	0.0623
PS0	PS1	15	1.25	0.0281

VI. Scenario identification methodology

The H264/AVC encoder control structure is modeled under the assumption that all memory accesses take one clock cycle. This model is profiled at design-time in order to explore the dynamic range of loop iteration count variations for various input frame sizes. In addition, different voltage and frequency settings are profiled in order to find optimal DVFS configurations. Our profiled results showed that the optimal energy consumption was achieved if the maximum available frequency setting for each voltage level is selected. This observation resulted in two scenarios, PS0 (1.8V and 40 MHz) and PS1 (1.2V and 12 MHz).

Table II shows an evenly distributed selection of true 16:9 resolutions that we have selected, which our SAM4L board can process within a one second time frame. It also shows the corresponding scenario selections and calculated run times, based on our available power modes in the SAM4L board. In Table II, the Scenario* column represents the scenario distribution for a platform with more available voltage settings, and is discussed in Section VIII.

The largest frame size the platform can handle from our selection is measured to be 816 x 459. The required run time (in clock cycles) for each frame size is estimated by counting all loop iterations and multiplying with all instructions within each loop based on the program disassembly. These numbers are compared to the measured run time for each frame size. From the measured results, we observed that the cycle count could be approximated by multiplying the number of all loop iterations for any input frame size with a factor of 4.22. This factor corresponds to the additional assembly instructions needed to set up and execute all for loops. The number of all loop iterations is calculated as shown in Listing 1. The total number of iterations are combined together with other parameters for all the nested loops in the application, resulting in the loopcnt variable. This loopcnt variable is multiplied by the approximated factor of 4.22. The separation of the approximated factor allows for easy portability of the scenario prediction mechanism to other platforms, because other platforms might implement the loop handling instructions differently. This difference would result in a higher or lower factor. The largest frame processing time is therefore approximated to 34.7 million clock cycles in an unoptimized setting where no pipelines are utilized. We use this frame size processing time as our worst case scenario, and we exploit it in combination with DVFS and RTH for lower frame sizes. We benefit from DVFS if the processing time of a smaller frame size with a lower frequency setting is less than the processing time of our worst case frame size. Therefore, we have to take our available system knobs into consideration when we classify our system scenarios.

Our scenario selection will be dependent on the platform's running frequency. In PS1 the platform is processing a frame 70% slower than in PS0. Therefore, frame sizes that need more than 30% of the run time of our worst case frame size are processed with PS0 (Scenario 0), and smaller frame sizes are processed with the PS1 configuration (Scenario 1).

Our arguments above are generalized in Equation 2, where we set the Scenario 1 upper limit for the maximum frame processing time in PS1 to be equal to the frame processing time of our largest frame (816 x 459) in PS0. The Scenario 1 upper limit is calculated as shown in Equation 3. In Equation 2 and 3, $T_{\text{WC frame}}$ is the

TABLE II
SCENARIO ASSIGNMENT BASED ON RUN TIME

Input frame size	Run time (million cycles)	Scenario	Scenario*
816 x 459	34.70	0	0
752 x 423	29.54	0	0
688 x 387	24.79	0	1
624 x 351	20.46	0	2
560 x 315	16.54	0	2
496 x 279	13.04	0	3
432 x 243	9.96	1	4
368 x 207	7.29	1	4
288 x 162	4.54	1	5
224 x 126	2.80	1	5

execution time in seconds of the worst case frame size. T_{sc_1} is the maximum allowed execution time for Scenario 1 in seconds, LC_{sc_0} and LC_{sc_1} are the total number of loop iterations for a specific frame in PS0 or PS1, respectively. β is the approximated scale factor used to get the total number of clock cycles, and f_{sc_0} is the platform's main clock frequency when Scenario 0 is active. In other words, if the required frame processing time is above 30% of our worst case frame processing time, then the platform switches from Scenario 1 to 0.

$$T_{\text{WC frame}} = T_{\text{sc}_1} = \frac{\beta \cdot LC_{\text{sc}_0}}{f_{\text{sc}_0}} = \frac{\beta \cdot LC_{\text{sc}_1}}{0.3 \cdot f_{\text{sc}_0}} \quad (2)$$

$$LC_{\text{sc}_1} = 0.3 \cdot LC_{\text{sc}_0} = 0.3 \cdot 8,222,691 = 2,466,807 \quad (3)$$

VII. PREDICTION AND SWITCHING MECHANISM

The input frame sizes of the H264/AVC encoder is used directly in the calculations to determine succeeding loop sizes and memory accesses. The input data variable dependency is modeled as part of our application monitoring unit (AMU) in the FSS framework [4]. This dependency is shown in Listing 1, where inline application dependent equations are pre-calculated (lines 2-7), before a global lookup-table is updated with the combined RTS parameter (line 10) as part of the overall AMU. This RTS parameter is then read by the AMU_1 function, which detects the next scenario (line 13). The application dependent calculations in lines 2-6 correspond to succeeding (nested) for- and/or while-loops in the encoder control structure, which are combined together in line 7 to estimate the required frame processing time in terms of loop iterations.

The AMU_1 function is a light weight inline function which compares the global RTS lookup-table values with global scenario lookup table values in order to determine which scenario the platform will adopt to. This approach is a generalization that takes into account multiple scenario parameters and RTS parameters. The generalization with the use of lookup tables would simplify an extension to multiple independent RTS parameters and/or scenarios, while in our particular case it would have been enough to compare the RTS parameter with the threshold between Scenario 0 and 1 directly. The values in the global scenario lookup table is found from our identification step, from Equation 3. If the value of the global RTS lookup table is greater than the value in the global scenario lookup table, then the platform changes its configurations to PS0 if it is in PS1, and vice versa. The platform adaption manager (PAM) function is then called by the AMU and performs the platform re-configuration using the DVFS system knob as described in Section V.

```
1  // Functions for monitoring RTS parameters
2  long loopcnt1 = (size_y + img_pad_size + img_pad_size)
       * (2 + (3*(size_x + img_pad_size + img_pad_size)
   ));
3  long loopcnt2 = ((size_x + 2 * img_pad_size) * 2) * (5
       + ((3*(maxy - 2 - 2)) + (3*(ypadded_size - maxy
   + 2))));
4  long loopcnt3 = ((je2 + 2)/2) * (2 + (((ie2 - 2)/2) +
   2));
5  long loopcnt4 = (ie2 + 2) * (((je2 - 2)/2) + 2);
6  long loopcnt5 = (size_y/4) * (1 + (6*(size_x/4)));
7  long loopcnt = loopcnt1 + loopcnt2 + loopcnt3 +
       loopcnt4 + loopcnt5;
8
9  // Update combined RTS parameter
10 (g_rts_data)[0] = loopcnt;
11
12 // Scenario detection/prediction
13 AMU_1(&g_scenario);
```

Listing 1. Functions combining data-dependent RTS parameters

VIII. EXPERIMENTAL RESULTS

In our experiments we simulate a realistic video sequence of 500 seconds, assuming it can be processed with 25 ARM Cortex M4 cores in parallel in order to meet the timing requirements for a frame rate of 25 frames per second (fps). The parallel cores are assumed able to process each frame individually with negligible synchronization overhead. This solution requires buffering and software pipelining of the frames to maintain forward frame dependency requirements. Other researchers have suggested similar approaches where their focus targets the parallelization of the H264/AVC encoder [8], [9].

Our scenario implementation is independent of this parallelization, and can be used on a faster processor for a single-core solution. After each frame is processed, we force the core into a low power retention mode, where the measured energy consumption is negligible. Before entering retention mode, a configurable timer is set to wake up the system just before the next frame arrives.

Our video sequence is measured using all three IPERF network measurements in Figure 2. To process a frame, we specify a threshold requirement for the available bandwidth to be equal to at least five times the required bandwidth for a frame rate of 25 fps with a particular frame size. This threshold allows other activity to continue on the network, and avoids using up all the available bandwidth. The limited number of available voltage levels in the SAM4L board would otherwise force the system to use the most power consuming voltage setting for lower thresholds instead of exploiting the dynamism in

the available bandwidth. Our simulated 500 seconds video sequence is created by combining different frame sizes from Table II in a sequence that fits with the available bandwidth of the measured networks in Figure 2, and our threshold requirements.

The application code is compiled and measured in a conventional way with compiler optimization parameter O3. When we optimize the application code with the O3 configuration, we assume it is a good enough optimization for the exploitation step in the system scenario design methodology presented by Gheorghita et. al. [3]. The energy consumption is measured in real time using our experimental setup, with and without the scenario mechanism. The overhead energy consumption of our scenario mechanism is measured by forcing the scenario mechanism to select PS0 each time a scenario switch takes place. This approach would not reduce the energy, but would trigger the scenario mechanisms in our AMU and PAM modules. The results are then compared to the energy consumption without using the scenario mechanism.

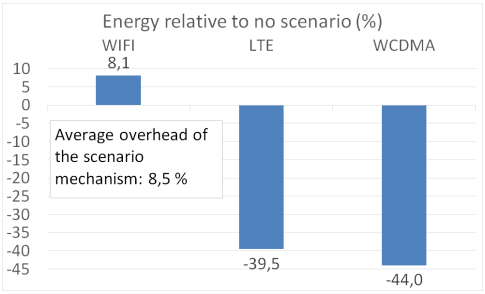

Fig. 3. Energy consumption relative to no system scenarios.

Figure 3 shows the energy reductions relative to the energy consumed by a no scenario implementation utilizing a race to halt (RTH) setup in each of the measured networks of Figure 2. A retention mode is used when the system is halted, which consumes less than 4.4 μW . The average energy overhead of our scenario mechanism is measured to be 8.5% for the 500 second sequence. These measurements are taken using an extreme conservative approach, however, where the scenario mechanism is triggered between each frame even if the scenario for the next frame is the same as the previous one. We perform this conservative triggering approach in all our measurements. In reality, it is not necessary to trigger the scenario mechanisms if the next predicted scenario remains the same. The total amount of actual scenario switches are therefore significantly lower than per processed frame. In fact, our system scenario mechanism only switches from PS0 to PS1 two times, and from PS1 to PS0 one time during the 500 second sequence for the LTE network. The rest of the time, our conservative triggering approach forces each frame to trigger a DVFS switch to the same DVFS setting. If this is avoided, for example by a simple if-sentence that only triggers the switching mechanism if the next DVFS setting is different than the existing setting, the energy overhead of the scenario mechanism will be negligible for the network examples shown here.

In the best case situation, all frame sizes are suitable for the PS1 configurations. We obtain in this situation up to 44% energy reduction as compared to a RTH solution without the use of the scenario mechanism. These results include the average scenario mechanism overhead of 8.5% energy increase for the 500 second sequence with our conservative triggering approach. When the available bandwidth is higher than our threshold for our largest frame size, then the scenario mechanism is not needed. It simulates our worst case situation, where the system is running constantly in the PS0 platform configuration. In this situation the energy consumption does not increase more than the measured scenario overhead. Our results meets Hillestad's [6] requirement for perceptual video quality, as mentioned in Section I, while maintaining the frame rate.

Our SAM4L board configuration capability is limited to two voltage states. If more voltage states were available, which is the case for modern Intel processors for example, the energy consumption is expected to decrease further since each frame size could correspond to a separate DVFS setting. To demonstrate the added gain of more DVFS settings, we have extrapolated our available DVFS settings with six evenly spaced voltage and frequency levels ranging from 1.05 V to 1.8 V and from 6 MHz to 40 MHz, respectively. The corresponding scenario assignment for the different frame sizes are shown in the scenario* column in Table II. We simulated our video sequence by manually applying the new scenario selection based on our bandwidth measurements shown in Figure 2, and calculating the dynamic energy. A lower threshold value, equal to four times the required bandwidth, is selected to utilize better our extrapolated DVFS settings. The increased number of DVFS settings allowed the system to use a lower threshold for the bandwidth requirement to process a particular frame. Our simulated result based on extrapolated DVFS settings is shown in Figure 4 for the LTE network. From this figure, we observe that the the system switches to a higher frequency five times, and it scales down three times. Our existing solution with two DVFS settings would switch back and fourth between two settings 6 times in total. Taking Equation 1 and the energy overhead we measured from Table I into consideration, our extrapolated results indicates that the dynamic energy is now reduced 59% compared to not having scenarios.

IX. Conclusion

Our framework for the systems scenario based designs (FSS) is combined with race to halt (RTH) and dynamic voltage and frequency scaling (DVFS) in order to achieve significant energy reductions for an H264/AVC encoder control structure. We measure our results on a modified SAM4L board from Atmel, which enables online DVFS

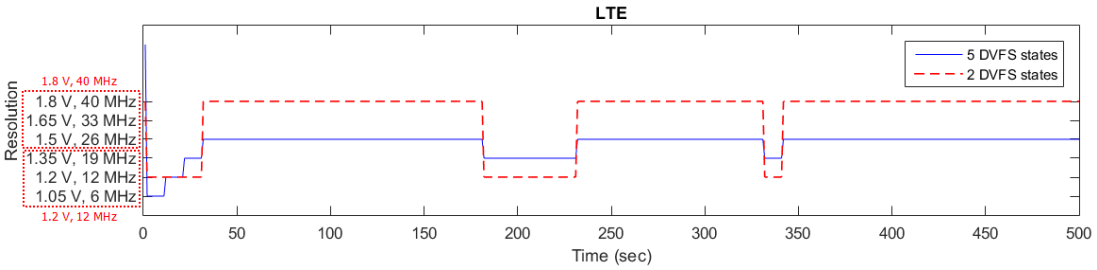

Fig. 4. DVFS state transitions during the 500 second video clip.

tuning with switching times comparable to state-of-the-art switching times.

System scenarios are identified based on data-dependent control variables, varying with the input frame size. These scenarios are predicted by using inline estimation functions for the expected execution time. These functions are executed at run time, and are compared to a predefined set of ranges for the scenario prediction with minimum overhead. Our switching mechanism then re-configures the platforms voltage and frequency accordingly.

We use IPERF to measure the available bandwidth in WIFI, LTE, and WCDMA networks for 500 seconds. A video stream consisting of different frame sizes is then composed based on the measured network data and a threshold requirement. Compared to a no system scenario solution with RTH we achieve up to 44% energy reduction for the encoded streams, including a 8.5% average energy overhead with conservative tuning. The perceptual video quality and frame rate is maintained, and we show how we could improve the energy consumption a further factor 2 if a few more voltage levels are available by extrapolating our measured results. We expect similar results to be obtained by other applications with similar characteristics. The techniques for scenario identification and switching presented here enables efficient use of our framework for system scenarios, especially for applications where the dynamic behavior is determined by many-valued data variables.

References

[1] D. Wu *et al.*, "Scenario-based system design with colored petri nets: an application to train control systems," *Software & Systems Modeling*, pp. 1–23, 2016.

[2] Z. Ma, *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogeneous Platforms*. Springer, 2007.

[3] S. V. Gheorghita *et al.*, "System-scenario-based design of dynamic embedded systems." *ACM Trans. Design Autom. Electr. Syst.*, vol. 14, no. 1, 2009.

[4] Y. Yassin *et al.*, "System scenario framework evaluation on efm32 using the h264/avc encoder control structure," in *2015 European Conference on Circuit Theory and Design (ECCTD)*, Aug 2015, pp. 1–4.

[5] M. A. Awan *et al.*, "Race-to-halt energy saving strategies," *Jour. of Systems Architecture*, vol. 60, no. 10, pp. 796 – 815, 2014.

[6] O. Hillestad, "Evaluating and enhancing the performance of ip-based streaming media services and applications," Ph.D. dissertation, Norwegian university of science and technology, 2007, doctoral thesis.

[7] G. Rao *et al.*, "Real-time software implementation of h.264 baseline profile video encoder for mobile and handheld devices," in *2006 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP 2006 Proc.*, vol. 5, May 2006.

[8] Z. Zhao *et al.*, "A highly efficient parallel algorithm for h.264 video encoder," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 5, May 2006, pp. V–V.

[9] S. Sun *et al.*, "A highly efficient parallel algorithm for h.264 encoder based on macro-block region partition," in *High Performance Computing and Communications*, ser. Lecture Notes in Computer Science, R. Perrott, B. Chapman, J. Subhlok, R. de Mello, and L. Yang, Eds. Springer Berlin Heidelberg, 2007, vol. 4782, pp. 577–585.

[10] C. Lee *et al.*, "Mediabench: a tool for evaluating and synthesizing multimedia and communications systems," in *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, Dec 1997, pp. 330–335.

[11] Atmel, "Sam4l xplained pro user guide," 2014.

[12] Y.-H. Lee *et al.*, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Real-Time Systems, 2003. Proc. on 15th Euromicro Conference*, July 2003, pp. 105–112.

[13] S. Irani *et al.*, "Algorithms for power savings," *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007.

[14] L. Niu *et al.*, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '04. New York, NY, USA: ACM, 2004, pp. 140–148.

[15] S. V. Gheorghita *et al.*, "Automatic scenario detection for improved wcet estimation," in *Proceedings of the 42Nd Annual Design Automation Conference*, ser. DAC '05. New York, USA: ACM, June 2005, pp. 101–104.

[16] ——, "Scenario selection and prediction for dvs-aware scheduling of multimedia applications." *Signal Processing Systems*, vol. 50, no. 2, pp. 137–161, 2008.

[17] E. Hammari *et al.*, "Identifying data-dependent system scenarios in a dynamic embedded system," *The International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'12, Las Vegas, USA*, July 2012.

[18] M. Palkovic *et al.*, "Dealing with variable trip count loops in system level exploration." in *Proc. of the 4th Workshop on Optimizations for DSP and Embedded Systems*, March 2006, pp. 19–28.

[19] M. Damavandpeyma *et al.*, "Throughput-constrained dvfs for scenario-aware dataflow graphs," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, April 2013, pp. 175–184.

[20] P. Van Dyke *et al.*, "Apparatus and method for resizing an image," Jun. 8 2010, uS Patent 7,733,405.

[21] S. Sesia *et al.*, *Front Matter*. John Wiley & Sons, Ltd, 2009.

[22] E. Dahlman *et al.*, "Wcdma-the radio interface for future mobile multimedia communications," *Vehicular Technology, IEEE Transactions on*, vol. 47, no. 4, pp. 1105–1118, Nov 1998.

[23] "Iperf," networking with iperf. [Online].

[24] National Instruments, "Ni mydaq measurement board," 2015.