COMPLEXITY METRIC DRIVEN ENERGY OPTIMIZATION FRAMEWORK FOR IMPLEMENTING MPEG-21 SCALABLE VIDEO DECODERS

Gouri Landge, Mihaela van der Schaar and Venkatesh Akella

Department of Electrical and Computer Engineering
University of California, Davis, CA-95616

ABSTRACT

We propose a systematic framework to optimize the energy consumption of wavelet-based scalable video decoders using generic computational complexity metrics derived from the frequency of execution of program basic blocks [1]. The complexity metrics are *independent* of the hardware architecture/resources of the decoders and capture both the time varying video content characteristics and the corresponding encoding parameters. We show how the *generic* complexity metrics can be translated into a platform-specific metric such as the execution time, which in turn can guide the selection of optimal voltage and frequency selection to optimize the energy. Preliminary results show 67% to 83% improvements in energy consumption on key functions of the 3D Wavelet based Scalable Codec (SVC), which is a likely candidate for the emerging MPEG-21 video standard [2]. Most of the processing in the proposed framework is done off-line, and hence it has limited impact in terms of delay or additional resource requirements for the decoder.

1. INTRODUCTION

Energy optimization of multimedia applications especially on battery operated devices such as laptops and PDAs is extremely important. Growing demand for higher data rates and enhanced functionality that results in more complex algorithms exacerbates this problem even further. In recent years, many techniques have been proposed to optimize the energy of multimedia applications on programmable processors [4][5][6][7]. In [4], a technique to dynamically adjust the pipeline depth of a processor is described to save power in response to varying computational workload. In [5], a method for energy optimization using runtime profiling of the input bit-stream and per frame frequency-voltage scaling is described. In [6], dynamic voltage scaling by monitoring the input data rate using FIFOs is proposed and in [7], an overview of techniques for dynamically tuning processor resources is presented. However, existing approaches suffer from two drawbacks. First, they are *ad-hoc,* i.e. they are created for a specific algorithm or implementation on a particular platform. There is currently no general framework to drive energy optimization that can be universally used across different video standards and various hardware/software implementations. The second drawback of existing work such as [4][6][7] is that they rely on *on-line* metrics, i.e. the decoder *learns* the data characteristics and drives the power optimization framework. The problem with on-line models is that they are based on small time-scales and hence, they have limited knowledge of long term variations in computational behavior and resource requirements. This results in modest benefits and unnecessary computational overhead at the decoder, to construct the metrics. Furthermore, these approaches cannot take advantage of specific decoding algorithm, making the metrics less accurate. As a consequence, the benefits of these metrics are limited. Existing approaches to video complexity metrics such as counting the number of basic operations such as adds, subtracts etc. (see for example [3]) are inadequate, because it is not possible to estimate the execution time accurately on a particular platform from these metrics. The actual execution time depends on the instruction set, micro-architecture details such as techniques to exploit instruction-level parallelism, pipeline depth, memory structure and bandwidth and the compiler optimization such as software pipelining and loop-unrolling etc.

We propose a new framework that is based on off-line metrics constructed by the encoder *explicitly* exploiting the knowledge of the algorithm. Given that the metrics are constructed *off-line* they can be complex and accurate and useful. Also, the proposed framework is general i.e. it is independent of the particular deployed video standard or the implementation platform. We achieve this by constructing *generic complexity metrics* (GCM) at the encoder that are based on an abstract representation of a generic decoder and translating these metrics into *real complexity metrics* (RCM) at the specific receiver. Using this framework, which decouples the GCM computation from the actual RCM, we obtain significant savings in energy of up to 83% on the investigated functions.

The paper is organized as follows. The methodology to create and transmit GCM is described in Section 2. Section 3 illustrates the methodology for estimating the GCMs using an example. The mechanism for translating GCM to RCM and frequency-voltage scaling to achieve energy savings is described in Section 4. Results are described in Section 5 and conclusions and future work is discussed in Section 6.

2. GENERIC COMPUTATIONAL COMPLEXITY METRICS FOR VIDEO BIT STREAMS

The computational complexity metric should satisfy the following requirements:

- The receiver (decoder) should be able to quickly and accurately estimate the execution time (RCM) of a given function based on the GCM.
- The metric should be generic to be able to satisfy a broad range of decoders with even broader range of implementations. Obviously, it is not practical to generate a metric for each possible receiver.

- The overhead in terms of generation and distribution of the meta-data representing the GCM and the interpretation of the GCM to RCM at the receiver should be minimal in terms of memory requirements, time and power.

We exploit two characteristics of video processing to meet the challenges listed above. First, the decoder processes data that has been generated by the encoder. Hence the encoder has a priori knowledge of the sequence of functions processed by any decoder. Second, the computational kernel of most video processing algorithms such as interpolation, wavelet transform lifting steps, bit plane coding/decoding etc. is a *loop***.** The execution time of a loop-based program can be estimated (accurately) by determining the number of iterations each loop has to perform, given the execution time for one iteration of the loop. This is formalized using the notion of *a basic block* (BB) [1]. A BB is a section of code that is executed from start to finish with one entry and one exit. A program loop can be modeled as a BB corresponding to the body of the loop. The BB execution time is *deterministic* and known at compile time because video processing is a real-time application, which means the loops cannot have any non-deterministic operations like interrupts or input/output operations within the loop body. The total execution time of a program to process a given sequence of data can be obtained by knowing the clock frequency of the underlying processor and the number of iterations of each *BB*. As a result, the abstract machine for our complexity metrics is a loop execution engine, capable of executing a particular sequence of *BB*s: BB_1, BB2… BB_n. This sequence will be called the *basic block signature* of a given encoder, which given the complementary relationship between the encoder and decoder is recognized by any decoder. The basic block signature consists of a subset of the total number of BBs in a program that satisfy some criteria. In this paper, we choose the basic blocks that contribute at least 1 % of the overall run-time.

We define GCM as a set of number of iterations of the *BB*s in the *Basic Block Signature* for distinct Adaptation Units (AU) in the next group of frames (GOF), i.e.

GCM = $\{GC_i^j\}_{i=1..Q, j=1..K}$ where Q is the total number of significant *BB*s, i is the *BB id*, K is the total number of AUs in a GOF and j is the AU *id*. The GCM is transmitted as meta-data along with the video stream.

3. GENERIC COMPLEXITY METRICS COMPUTATION

In this section, we will illustrate how GCMs can be determined by considering a state-of-the-art wavelet video codec, based on the 3D ESCOT [9] implementation. The block diagram of such a coder is depicted in Figure 1. (For more information on SVC codecs and specific 3D ESCOT implementation, the interested reader is referred to [8][9][2]). In this SVC implementation, a number of temporal subbands are jointly entropy coded. We refer to this group of frames as a Frame Block.

For our illustration, we determine the GCM for the BB corresponding to the significance map sub-function (see DE_ZC in [9]) of the Entropy Decoding module. This function was selected because it represents a significant percentage (e.g. for certain sequences up to 60% [8] [11]) of the total computational complexity at medium and high bit rates.

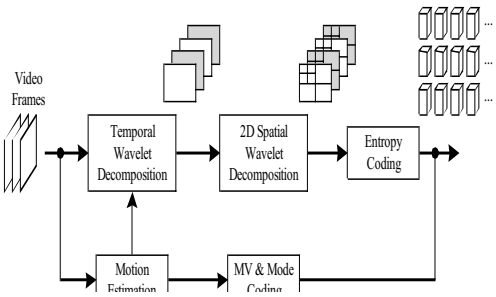

Figure 1. Block Diagram of MCTF Based Wavelet Video Encoder.

The GCM is determined by the distribution of the significant (non-zero) symbols at the various bitplane levels. The distribution of significant symbols across each bit plane *bp* for a spatio-temporal band at temporal level **n** and spatial level **m** is denoted as $D_s^{n,m}(bp)$. At encoding time, $D_s^{n,m}(bp)$ is computed for the maximum number of bitplanes. However, at the decoder side, only a subset of the precomputed $D_s^{n,m}(bp)$ values is used corresponding to the transmitted (decoded) bitplanes $BP^{n,m}$. The GCM for the DE_ZC sub-function can be then determined as:

$$GCM_{DE\_ZC} = \sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{fb=1}^{FB}[(BP^{n,m,fb}\cdot L^m - \sum_{bp=1}^{BP^{n,m,fb}} D_s^{n,m,fb}(bp))*P],$$

where

- N = total number of Temporal Levels,
- M = total number of Spatial Levels,
- FB = the number of Frame Blocks at n^{th} temporal level in the temporal decomposition tree,
- $BP^{n,m,fb}$ = the decoded bitplanes at n^{th} temporal level and m^{th} spatial band for a frame block fb,
- L^m = the band size at m^{th} spatial level,
- $D_s^{n,m,fb}(bp)$ = the distribution of significant symbols at nt^{h} temporal level and m^{th} spatial band for a frame block fb,
- P = Frame Block Size.

In the previous equation, to derive the number of zero symbols, we subtract $\sum_{bp=1}^{BP^{n,m,fb}} D_s^{n,m,fb}(bp)$ that quantifies the number of significant symbols from the term $BP^{n,m,fb}\cdot L^m$ representing the maximum number of symbols that can be processed.

The overhead to compute $D_s^{n,m,fb}(bp)$ for every spatio-temporal band is negligible as compared to the total encoding complexity. Moreover, the similarity between consecutive temporal bands at the same temporal level can be exploited to further reduce the overhead significantly. For instance, only one band at each temporal level is used for estimating the GCM. We can further reduce the overhead for the GCM computation by considering only the lowest (most significant) temporal bands in the estimation process. In this way, tradeoffs can be made between the accuracy of the GCM estimation and the computation overhead. Note that the accuracy decrease is mainly visible at high bit rates, since at lower rates the highest temporal bands exhibit only few significant symbols.

Figure 2 shows the actual and estimated GCM of the DE_ZC function for different complexity estimations, i.e. based on a different number of temporal bands. Model 1 is computed by selecting one band at each temporal level. Model 2 is determined using only two most important temporal bands (i.e. lowest temporal level). Model 3 is computed based solely on the total number of transmitted bitplanes $BP^{n,m,fb}$. This last model gives an upper bound of the complexity associated with this BB. Here, due to space limitations, only the results for two sequences are presented. Our extensive study has shown similar results for other sequences. The data clearly shows that even for low complexity estimations, only a small error margin to the actual GCM is observed.

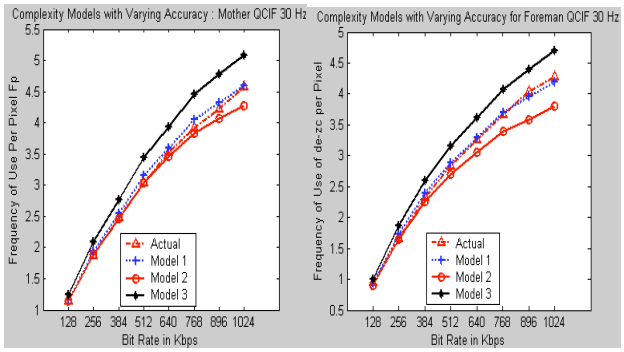

Figure 2 : GCM Estimation with Varying Accuracy

4. METHODOLOGY

Our framework consists of the following steps. Some of them are performed at the encoder and some of them are performed at the decoder.

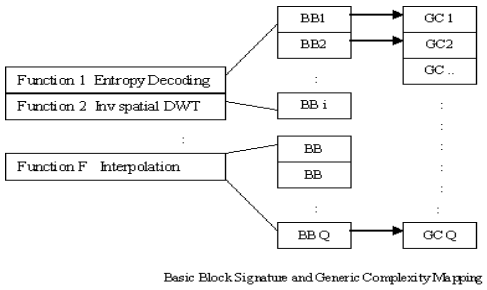

Figure 3 – Basic Block Signature and Generic Complexity Mapping

At the Encoder:

STEP 1:

Compute the GCMs using the techniques described in Section 2 and Section 3. This involves creating a structure shown in Figure 3. The encoder assumes that a generic decoder will consists of a set of functions each of which may have a set of basic blocks. The BB signature is a sequence of significant basic blocks chosen by a particular criteria such as those that contribute at least 1% to the computation. This is done *once* when the new encoder is deployed.

STEP 2:

Each BB is associated with GCi that denotes the number of iterations that the BB is supposed to make. This is derived from the complexity metric (shown in Section 3) and it is computed once for every adaptation unit.

STEP 3:

The GCM values are transmitted as meta-data together with the corresponding bitstream.

At the Decoder:

In the case of energy adaptation, the RCM is the actual decoding time for a video bit stream on a specific hardware/software platform. The GCM generated by the encoder has to be translated into an RCM at the decoder.

STEP 1

Create a table called BBET (basic block execution time) with the BB starting instruction address A and the execution time per invocation of the *BB* in terms of the number of processor clock cycles T for all the BBs in the basic block signature of the encoder. T is known at compile time as most modern compilers perform basic block analysis for code optimization and instruction scheduling. It can also be obtained by profiling using tools such as Vtune from Intel for x86-based platforms. T incorporates platform-specific (and implementation-specific) details such as the hardware/software partitioning, special resources such as zero-overhead loops, co-processors and optimizations such as loop unrolling or software pipelining.

STEP 2

Using the received GCM meta-data,
FOR each *BBi* : *i* from 1 to *Q*
FOR each adaptation unit j: 1 to K

(a) Look up BBET to get the corresponding basic block address and basic block execution time. (A_i, T_i)

(b) Number of iteration for the next set of frame blocks $= \{GC_i^j\}$

(c) The RCM in terms of number of clock cycles for a frame block *j*, for the basic block *i*, will then be given by $RC_i^j = GC_i^j * T_i$

STEP 3- Voltage and Frequency Reconfiguration

Using the RCM, the optimal voltage and frequency of operation of the processor for a given set of frames is computed as follows:
FOR each adaptation unit j: 1 to K

(i) Clock Frequency for Frame Block j : $F_j = RC_i^j / (T_{total} / K)$

(ii) Voltage for Frame Block j : $V_j = (F_j * V_{\max}) / F_{\max}$

(iii) *if* $(V_j < V_{floor}) \rightarrow V_j = V_{floor}$

where V_{floor} is the minimum operating voltage, $V_{\max}$ is the maximum operating voltage, $F_{\max}$ is the maximum operating frequency of the processor, T_{conf} is frequency reconfiguration overhead (the time required for the phase-locked loops to stabilize, during this time the processor is usually stalled or idle), the total decoding time available is T_{total}.

STEP 1 is executed once during the installation of the decoder. STEP 2 is executed once for each adaptation unit in the worst-

case. STEP 3 is also executed once for each adaptation unit in the worst case. In the next section we will quantify the energy savings and the overhead for a set of typical video sequences.

5. RESULTS

The amount of energy saved by reducing the voltage and frequency dynamically based on our framework is calculated as shown below. We make the following assumptions. The Frame Block of four frames is considered as the Adaptation Unit (AU). This is because in the referred implementation of the codec, entropy coding is done across a block of four frames and because of the high correlation among the neighboring frames computational complexity for each of the 4 frames in a block is very similar. We further assume that, the time spent in de_zc sub-function for a block of four frames is 32 ms, the maximum frequency of operation is $F_{\max}$ = 2.6 GHz, The maximum voltage at which the processor operates is $V_{\max}$ = 3.3 V, Voltage floor is 1.0 V and the frequency reconfiguration overhead is 30 us based on Intel Speed Step technology [10] Maximum energy spent without voltage frequency scaling:

$$E_{\max} = C_{eff} * (V_{\max})^2 * F_{\max} * T_{total}$$

Note here that, although less time would be actually spent in the de_zc sub-function, the processor is still operated at $V_{\max}$ and $F_{\max}$, hence no energy savings could be achieved.

Energy spent with optimal voltage/frequency scaling:

$$E_{opt} = \sum_{i=1}^{2^N} (C_{eff} * V_i^2 * F_i * T_i)$$

Worst case energy overhead for voltage/frequency scaling:

$$E_{ovrhd} = (C_{eff} * V_{\max}^2 * F_{\max} * 2^N * 30\mu s)J$$

Energy Savings =

$$(([E_{\max} - (E_{opt} + E_{ovrhd})] / E_{\max}) * 100)\%$$

Table 1 lists the energy savings obtained for various sequences with different characteristics of motion and texture. We find that the savings in energy range from 67% to 83% for the DE_ZC subfunction.

Table 1 Energy Savings for DE_ZC sub-function

Sequence Description	**Energy Savings** (%)
Akiyo Frames 1-64	67.41
Foreman Frames 1-64	82.80
Foreman Frames 236-300	77.24
Mother Frames 1-64	80.13

Table 2 shows the overhead related to the complexity driven frequency scaling mechanism, for a GOF of 64 frames of a QCIF sequence encoded using 4 temporal decomposition levels, 3 spatial decomposition levels and 10 bit precision for the coefficient values ($BP^{n,m,fb}$). It can be clearly seen that this mechanism introduces very insignificant overhead in terms of computation power as well as memory requirements.

Table 2 – Overhead of the framework

Overhead Type	**Overhead Values**
GCM Computation at Encoder	~1.521 M cycles
Memory for GCM Transmission as meta data.	864 bits
Runtime for Steps 1, 2 & 3 at decoder	~500 cycles

6. CONCLUSIONS & FUTURE WORK

We introduce a framework for complexity metric driven energy optimization for MPEG-21 scalable video decoders on a general-purpose processor capable of adjusting the frequency and voltage dynamically. Even though we focused on energy optimization in this paper, the framework can also be used for other dynamic (run-time) optimization of processor resources such as allocation of critical resources such as buffers, data caches and optimal hardware/software partitioning. Also, the framework is not restricted to wavelet-based codecs. It can be used for other video standards as well. Future work will consist of refining the metrics for greater accuracy and automating some of the steps such as BBET construction and optimal selection of the basic blocks in the basic block signature.

7. REFERENCES

[1] Alfred Aho, Ravi Sethi and Jeffrey, Ullman, "COMPILERS: Principles, Techniques and Tools", Addison-Wesley, 1986. (Chapter 9, page 528)

[2] J.R. Ohm, M. van der Schaar, J. Woods, "Inter-frame wavelet coding – Motion Picture Representation for Universal Scalability", Image Communications, Special issue on Digital Cinema, to appear June 2004.

[3] M. Horowitz, A. Joch, F. Kossentini, A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis", IEEE Trans. Circuits and Systems for Video Tech., vol. 13, No. 7, July 2003

[4] S. Kim, C. H. Ziesler, M.C. Papaefthymiou, "Fine-grain real-time reconfigurable pipeline", IBM J. RES. & DEV. Vol. 47, No. 5/6 Sept/Nov 2003

[5] Daniel G Sachs, Sarita V Adve, Douglas L Jones, "Cross-layer Adaptive Video Coding To Reduce Energy on General Purpose Processors", ICIP 2003, pages 25-28

[6] Gutnik, V.; Chandrakasan, *A.P.* "Embedded power supply for low-power DSP" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol: 5 , Issue: 4 , Dec. 1997 Pages:425 - 435

[7] Albonesi, D. H,, Balasubormonian, R. et. al " Dynamically Tuning Processor Resources with Adaptive Processing", IEEE Computer, December 2003, pages 49-58

[8] J. Xu, R. Xiong, B. Feng, G. Sullivan, M. Lee, F. Wu, S. Li, "3D Sub-band Video Coding using Barbell Lifting", ISO/IEC JTC1/SC29/WG11, MPEG2004/M10569/S05, March 2004

[9] J. Xu, S. Li, Y. Zhang, Z. Xiong, "A Wavelet Video Coder Using Three Dimensional Embedded Sub-band Coding With Optimized Truncation (3-D ESCOT)" http://research.microsoft.com/china/papers/Wavelet_Codec_Using_3D_ESCOT.pdf

[10] Gochman, Ronen et. al "Intel Pentium M Processor: Microarchitecture and Performance, Intel Technology Journal, Vol 7, Issue 2, May 2003

[11] Gouri Landge, Mihaela van der Schaar, Venkatesh Akella, "Complexity Analysis of Scalable Motion-Compensated Wavelet Video Decoders", SPIE August 2004,Vol 5558-143