Future Generation Computer Systems 56 (2016) 247–261

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Dynamic core allocation for energy efficient video decoding in homogeneous and heterogeneous multicore architectures

Rajesh Kumar Pal [*], Ierum Shanaya, Kolin Paul, Sanjiva Prasad

*Indian Institute of Technology Delhi, India*

## HIGHLIGHTS

- Present dynamic core allocation for video decoding on homogeneous multicores.
- Present an energy-efficient video decoding method for heterogeneous multicores.
- Show energy savings with dynamic core allocation.
- Analyze factors influencing frame decoding time.

ARTICLE INFO

*Article history:*
Received 15 January 2015
Received in revised form
23 August 2015
Accepted 16 September 2015
Available online 28 September 2015

*Keywords:*
Core allocation
H.264 video decoding
Embedded system
Heterogeneous multicores

ABSTRACT

This paper describes two dynamic core allocation techniques for video decoding on homogeneous and heterogeneous embedded multicore platforms with the objective of reducing energy consumption while guaranteeing performance. While decoding a frame, the scheme measures "slack" and "overshoot" over the budgeted decode time and amortizes across the neighboring frames to achieve overall performance, compensating for the overshoot with the slack time. It allocates, on a per-frame basis, an appropriate number and types of cores for decoding to guarantee performance, while saving energy by using clock gating to switch off unused cores. Using the Sniper simulator to evaluate the implementation of the scheme on a modern embedded processor, we get an energy saving of 6%–61% while strictly adhering to the required performance of 75 fps on homogeneous multicore architectures. We receive an energy saving of 2%–46% while meeting the performance of 25 fps on heterogeneous multicore architectures. Thus, we show that substantial energy savings can be achieved in video decoding by employing dynamic core allocation, compared with the default strategy of allocating as many cores as available.

## 1. Introduction

Contemporary video decoders for real-time, large and detailed digital movies/videos on embedded platforms require high CPU performance. H.264 [1] is one of the best video codecs in terms of compression and quality. Its compression efficiency is at least twice that of the earlier codecs such as MPEG-2, and MPEG-1 [2]. The decoding process of H.264 produces video with perceptibly high quality. However these advanced features come at a cost of increased computational requirements: the video encoders/decoders exploit advanced instruction sets (MMX/SSE/SSE2), instruction-level parallelism, and parallelism provided by modern processors (multi/manycores). Multi-threaded implementations of H.264 codec take advantage of multiple cores provided by embedded processors such as ARM Cortex A15 and Intel Silvermont.

While playing video on devices such as mobiles and tablets, users expect high video quality *as well as* long battery life. These are conflicting requirements: high quality of video means better resolution and higher frame rates, which need more computation and thus more energy. The general approach in the video decoders is to utilize as many cores as available on the multicore platform. For example, in *libavcodec*, the leading audio/video codec library, there is a flag *threads* in *AVCodecContext*; when set to *auto* this lets the decoder detect the number of available cores and spawn as many threads. Employing all available cores definitely helps to meet the performance, albeit at the cost of higher energy consumption. We show in this work that with intelligent core allocation at the frame level, performance can be guaranteed with significantly lower energy consumption. The proposed core allocation methodology can be employed easily on all embedded multicore platforms to enhance battery life while providing the desired performance.

* Corresponding author.
*E-mail addresses:* rkpal@cse.iitd.ac.in (R.K. Pal), ierum@cse.iitd.ac.in (I. Shanaya), kolin@cse.iitd.ac.in (K. Paul), sanjiva@cse.iitd.ac.in (S. Prasad).

http://dx.doi.org/10.1016/j.future.2015.09.018
0167-739X/© 2015 Elsevier B.V. All rights reserved.

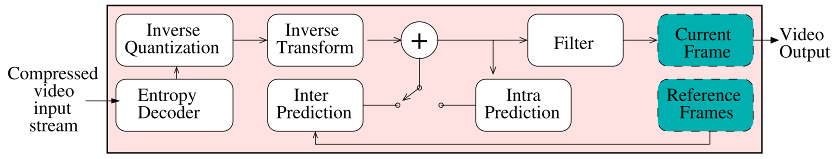

**Fig. 1.** H.264 decoder.

Soft real-time applications such as video encoders/decoders and speech/image recognition have soft deadline constraints. These applications can gracefully accommodate occasional deadline misses. For example, if the video decoder can decode and render the required number of frames (more than 24 fps) within the deadline of 1 s in spite of few frames locally missing their individual deadline, an overall perceptibly acceptable quality of video is achieved. From the performance perspective, these applications may miss their local deadlines occasionally but must meet the global deadline. Most frames are decoded before their local deadlines. Thus one can amortize the decode times across frames and compensate for the occasional local deadline misses while meeting the global deadline. The technique explored in this paper is based on this observation. Isovic et al. [3] used an alternative strategy of frame skipping to meet the global deadline.

Embedded processors for multimedia communication devices often adopt heterogeneous multi-core architecture in order to achieve good power efficiency for executing mixed control/data processing tasks. When playing a video, the processing resources are responsible for more than 60% of the power consumption [4,5]. This leads to a drastic decrease in mobile devices autonomy as lithium battery technologies are not evolving fast enough to absorb the ever-growing energy requirements of such mobile architectures [6]. Due to the limitation of the microprocessor fabrication technologies, it is expected that only 20% of the energy saving will be achieved in the next few years [7]. Thus, one should consider the optimization of overall system including the hardware and the software platforms to cope with the energy saving issue. To take full advantage of these multi-level energy saving opportunities, mobile systems designers should deal with the increasing system complexity and heterogeneity. Various approaches like dynamic task scheduling, heterogeneous architectures, hybrid parallel and hybrid pipeline schemes, frame level parallelism are used for obtaining the performance and energy efficiency in video decoding.

Most prior work [8–10] has used DVS/DVFS to trade-off between energy and performance. *To the best of our knowledge, ours is the first use of slack time to determine when and how many cores to switch on dynamically at runtime to save energy while meeting performance constraints*. When decoding a frame, we measure the slack and overshoot times over a budgeted decode time and use the slack time to compensate for the overshoot. We assign a suitable number of cores on a per-frame basis to guarantee performance in homogeneous multicore architectures. The unused cores are shut off using clock/power gating, thus saving energy. For heterogeneous multicore architectures, we assign suitable type of cores in required numbers on a per-frame basis to preserve energy while guaranteeing performance. We find that our schemes are profitable for embedded platforms as significant benefits can be achieved in terms of energy, without changing the hardware or software, merely by controlling the core allocation dynamically.

This paper makes four major contributions:

- We show that the default strategy of allocating as many cores as available on the platform leads to substantial energy wastage.
- We present a simple core allocation methodology for H.264 video decoding on homogeneous multicore platforms to meet the performance while conserving energy.
- We present a dynamic core allocation methodology for video decoding on heterogeneous multicore architectures that saves energy while meeting performance.
- We identify and analyze the factors that influence frame decoding time on multicore architectures.

In the next section we overview the default core allocation and then present the dynamic core allocation strategy that conserves energy while meeting the required performance. In Section 3 we describe our experimental methodology and details of test video sequences. Section 4 provides an insight into the factors influencing frame decoding time. Section 5 presents the results and its analysis. Section 6 discusses related work in this area. Section 7 concludes the paper with directions for extending this work.

## 2. Core Allocation for H.264 Decoder

In this section, we present an overview of the core allocation in H.264 video decoder and thereafter propose dynamic core allocation.

### 2.1. Default core allocation

The threaded implementation of H.264 video decoders is done in the following two fundamentally different ways.

- *Functional Decomposition*: As shown in Fig. 1 each frame gets decoded after passing through several functional stages: inverse transforms and quantization, intra prediction, motion compensation and deblocking filter. Each function is performed by a separate thread. This implementation has limited scalability on multicore architectures as to increase the number of threads we must partition a function into two or more threads. Due to interdependence and tight coupling between sub-functions of a function, division becomes difficult. Moreover unbalanced workload to the threads can cause thread waiting and synchronization delays. These limitations hamper utilization of large number of cores for such implementations of video decoders.
- *Data Domain Decomposition*: The hierarchy of data domain decomposition in H.264 is shown in Fig. 2. A H.264 video consists of many groups of pictures (GOP). Each GOP is made up of a number of frames. Each frame consists of slices. A slice is an independent and self-contained encoding unit. A slice is further divided into macroblocks which are 16 × 16 pixels. Motion estimation and entropy decoding is done at macroblock level. Depending on the required scalability, threads can be created at different levels of this hierarchy. As we move towards lower levels of the hierarchy, more threads can be created. One significant advantage of data domain decomposition over functional decomposition is that each thread processes the same operation on different data blocks, each having the same dimensions. The scalability, thread homogeneity and even distribution of workload amongst the threads make this type of implementation suitable for exploiting large number of cores. Considering these factors, we decided on a data domain decomposition with a multi-threaded implementation of the H.264 decoder.