MULTI-THREADED ARCHITECTURES AND BENCHMARK TESTS FOR REAL-TIME MULTI-VIEW VIDEO DECODING

C. Goktug Gurler¹, Anil Aksay², Gozde Bozdagi Akar², A. Murat Tekalp¹

¹ College of Engineering, Koç University, Istanbul, Turkey
² Electrical and Electronics Engineering Dept, Middle East Technical University, Ankara, Turkey
{cgurler, mtekalp}@ku.edu.tr, {anil, bozdagi}@eee.metu.edu.tr

ABSTRACT

3D video based on multi-view representations is becoming widely popular. Real-time encoding/decoding of such video is an important concern as the number and resolution of views increase. We present systematic methods for design and optimization of real-time multi-view video encoding/decoding algorithms using multi-core processors and provide benchmark results. The proposed multi-core decoding architectures are fully compliant with the current JVT-MVC international standard, and enable multi-threaded processing with negligible loss of encoding efficiency. Benchmark results show that multi-core processors and multi-threading decoding is necessary for real-time multi-view video decoding and display.

Index Terms— Multi-view video, MVC, Multi-core processors, multi-thread video encoding/decoding

1. INTRODUCTION

The current trend in designing more powerful general-purpose processors is based on multi-core architectures [1]. However, increasing the number of cores does not automatically yield performance gain if the software is not designed to efficiently and effectively divide the workload among multiple cores. Therefore, software developers should consider multi-threaded architectures to take advantage of the state of the art processors. One area where such consideration must be taken is multi-view video (MVV) encoding/decoding.

Real-time MVV encoding/decoding is a significant and timely subject, since 3D video is getting highly popular with the recent advances in 3D display technologies. There are several ongoing research projects on capture, representation, and transmission of 3D Video, such as the European projects ATTEST, 3DTV, 3Dphone, 3D4you, 3DPresence, Mobile3DTV [2], and the Japanese project FTV.

Up to date, video software developers have aimed speeding up encoders by choosing suboptimal algorithmic methods such as fast motion estimation and fast mode selection. While this approach has proved sufficient to achieve real-time performance in case of standard monoscopic video, it may not be possible to achieve real-time video encoding and decoding in the case of 3D MVV, when the number of views is larger than 2, without taking advantage of multi-core processors.

In order to encode MVV efficiently, the Joint Video Team (JVT) is working on the extension of H.264/AVC for Multi-View Video Coding (MVC). This work will be finalized in 2009 [3]. MVC mostly benefits from inter-view predictions, where an average gain of 1.5 dB is achieved compared to independent coding of each view (simulcast coding). However, this coding gain comes with additional computational complexity. In [4], a simplified prediction scheme is proposed, with negligible loss in coding efficiency. This prediction structure has been utilized in the multi-core architectures discussed below.

When multi-core systems are used, MVC decoding requires special care in order to implement an efficient multi-threading architecture. In [5], a new coding structure is proposed to allow parallel encoder/decoder operation for different views without significant change in coding efficiency and it is adopted in the upcoming MVC standard.

The aim of this paper is to propose a simpler method for real-time decoding of multi-view video using multi-core processors, and provide benchmark results. These benchmark results are intended to determine the number of cores required for real-time decoding of MVV for a given spatio-temporal resolution, encoding options, and number of views. The rest of this paper is organized as follows: In Section 2, we overview MVC encoding schemes that enable multi-threaded decoding. In Section 3, we describe the multi-threaded decoding architectures. In Section 4, we provide the benchmark result for MVV encoding and decoding. Finally, we draw conclusions in Section 5.

2. OVERVIEW OF MULTI-VIEW VIDEO ENCODING

MVC extension of H.264/AVC is based on High Profile, which has higher coding efficiency and is more complex to encode/decode. It features such tools as Hierarchical B-
pictures [9], CABAC and disparity compensation between the frames of different cameras.

MVC syntax allows encoder to have several different prediction schemes among cameras. In one extreme, each frame can be predicted from all other views, and in another extreme, it can be predicted only from the frames of the same view (simulcast). Depending on the prediction structure, coding efficiency varies.

In our prediction schemes, there are frames (time instants) in which the prediction is done within the same view only, and frames where prediction is done with frames within the same time instant only. We present different frame types and predictions between them in Figure 1.

---

**I frames:** Frames that can be decoded by themselves.

**Anchor frames:** Frames that are predicted only from frames that are from the same time instant. These frames do not use frames from the same camera.

**B frames (non-anchor frames):** Frames that can be decoded using only frames from its own camera. It uses both previous and future frames.

Figure 1: Encoding Schemes

3. MULTI-THREADED ARCHITECTURES FOR VIDEO DECODING

Common 3D players have three fundamental tasks that are summarized in Figure 2. The first task is to decode the video stream and to form the frames. These frames belong to a particular view and a particular time instant. The second operation, named as interdigitizing, is to merge frames that belong to same time instant and obtain a single image for that instant. The output image contains pixels from all views in a scaled spatial resolution and cannot be viewed correctly without a 3D display. The pattern of interdigitizing frames depends on the types of the display and the number of views. The final step is to display the interdigitized images on screen in a timely fashion.

Figure 2: Block diagram of a typical 3D player

One can distribute these three tasks among separate threads and assume that the system is taking advantage of the multiple cores. However, the decoding process requires most of the CPU cycles. Therefore, if only a single thread is responsible for decoding, it will be the bottleneck. The most efficient way of implementing multi-threading for a 3D player requires handling decoding in a multi-threaded manner. The key aspect is to identify the independently decodable code blocks and forward them to a separate thread for processing. This can be done by observing the inter-view dependencies of the encoded streams. The rest of this section is dedicated to explain how to identify independently decodable code blocks for simulcast and MVC streams.

---

Figure 3: The Multi-Threaded Decoding Schemes

3.1. Multi-Threaded Decoding for Simulcast Encoding

In the case of simulcast encoding, each view is encoded as a separate H.264/AVC stream and already independently decodable. Therefore, decoding each video stream by a separate thread is the trivial solution for simulcast case.
The number of cores present in the system can be used to determine the number of threads to decrease the threading overhead. Figure 3-A shows the distribution of frames among two threads to run with a dual core processor. In our case, one of the threads decodes streams 0 to 3 while the second one decodes 4 to 7. In the quad-core case, each thread decodes two of the eight streams. The cost of such simple decoding mechanism is increase in the bitrate, since independent encoding of multiple video streams has lower encoding efficiency.

3.2. Multi-Threaded Decoding for MVC Encoding

The inter-view dependencies decrease the overall bitrate of the content but introduce decoding complexity. In MVC, we have single backward compatible independent stream that is intended for standard monocular displays with standard H264/AVC decoders. Rest of the streams depend on the I-Frame of this independent stream. Figure 4 exemplifies the decoding of stream 6. In this example, the stream recorded by camera 4 is the independent stream. Therefore, before decoding stream 6, I-Frames of stream 4 and the anchor frames of stream 5 must be decoded. Only then, the frames of stream 6 can be decoded.

3.2.1. Dual-Core

The inter-view dependencies for the dual-core scheme is shown in Figure 3. Similar to the simulcast case, one of the threads decodes streams 0 to 3 while the other decodes streams 4 to 7. The difference is that even though the first thread is responsible only for streams 0 to 3 it needs to decode I-Frames of stream 4 due to the dependency relations. Therefore, I-frames of stream 4 are decoded twice by each thread. This creates a small processing redundancy when compared with the single threaded decoding. However, this is a negligible cost to create two independent threads that can decode each half of the content simultaneously in the case of dual-core processors.

3.2.2. Quad-Core

In dual-core case, we split the content into half by the independent view. For creating two more threads with minimum decoding redundancy, each half is further split into two by altering inter-view dependencies. The flexible structure of the MVC allows us to define dependencies for streams that are not adjacent to each other. The drawback of defining such dependencies is the increase in the overall bitrate since the similarities between two views start to diminish as their camera locations deviate from each other. In Section 5.1, we present the effects of non-adjacent dependencies on the bitrate. Figure 3-B shows the newly introduced dependencies and the distribution of four independently decodable frames among threads.

3.2.3. Multi-Threaded Decoding for MVC Encoding

The inter-view dependencies for the quad-core scheme is shown in Figure 3. The difference is that even though the first thread is responsible only for streams 0 to 3 it needs to decode I-Frames of stream 4 due to the dependency relations. Therefore, I-frames of stream 4 are decoded twice by each thread. This creates a small processing redundancy when compared with the single threaded decoding. However, this is a negligible cost to create two independent threads that can decode each half of the content simultaneously in the case of dual-core processors.

Figure 4: The frames required for decoding stream 6

4. RESULTS

4.1. Encoding Tests and Results

The results are provided for multi-view video sequences “Adile” (Computer generated animation by Momentum [6]) and “Ice” (Converted to 3D from 2D scene using [7]) [Source: BBC documentation “Planet Earth”). Resolution of Adile sequence is 640x480 and resolution of Ice sequence is 640x384. GoP size is selected as 16 frames. Multi-view codec is MVC Reference Software JMVC 3.0.2 [8]. We encoded the first 241 frames of both sequences with fixed Quantization Parameters (QP) {26, 30, 34} to generate rate-distortion curves. PSNR values are averaged over all frames.

4.1.1. Encoding Efficiency

As shown in Figures 5 and 6, MVC Encoding for Quad-Core does not introduce a penalty in encoding efficiency.
Overhead is less than 10% in bitrate. In addition, it is shown that using MVC simplified prediction scheme provides more than 3dB PSNR gain when compared with simulcast encoding.

4.2. Decoding Tests and Results

4.2.1. Decoding Test Setup

We have personal computers with single, dual and quad core processors at identical clock frequencies (2.4 Ghz) in our test-bed. In order to achieve real-time decoding statistics, the proposed architectures are implemented. We had modified ffmpeg library [10] for adding the MVC features as described in [11]. Adile and Ice sequences are used as the content for testing the maximum frame rate of systems. Both contents are captured from eight different positions therefore, each displayed frame requires eight decoding operations in addition to interdigitizing.

<table>
<thead>
<tr>
<th>Decoding Parameters</th>
<th>FPS/Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cores</td>
<td>Content Type</td>
</tr>
<tr>
<td>4</td>
<td>Simulcast</td>
</tr>
<tr>
<td></td>
<td>QuadCore</td>
</tr>
<tr>
<td></td>
<td>DualCore</td>
</tr>
<tr>
<td></td>
<td>SingleCore</td>
</tr>
<tr>
<td>2</td>
<td>Simulcast</td>
</tr>
<tr>
<td></td>
<td>QuadCore</td>
</tr>
<tr>
<td></td>
<td>DualCore</td>
</tr>
<tr>
<td></td>
<td>SingleCore</td>
</tr>
<tr>
<td>1</td>
<td>Simulcast</td>
</tr>
<tr>
<td></td>
<td>QuadCore</td>
</tr>
<tr>
<td></td>
<td>DualCore</td>
</tr>
<tr>
<td></td>
<td>SingleCore</td>
</tr>
</tbody>
</table>

Table 1: Achieved maximum frame rates

4.2.2. Decoding Test Results

We summarize the achieved results in Table 1 and make the following observations:

- Increasing the number of the cores over a certain number starts yielding diminishing returns because some operations in the software has fixed costs. They start to dominate as the other tasks are handled more efficiently.
- The simulcast decoding is faster than the MVC decoding because it has no processing redundancy. Decoding an I-Frame is nearly four times more costly than other frames. However, the RD performance of simulcast encoding is considerably lower than that of MVC.
- We can see that the threading overhead in our implementation is quite low since the difference between dual core encoding scheme and quad core encoding scheme yields very close results on a dual core processor.

5. CONCLUSIONS

We performed benchmark tests using standard resolution video with eight-views. Even with standard resolution video, we were not able to achieve commonly used 30 fps display rate for 8-view video using a single core processor. It is natural to think that future systems will require higher spatial resolutions and more views to increase the quality of 3D perception. Therefore, we believe that it is vital to develop efficient multi-threaded decoding algorithms that most effectively utilize multi-core possessors.

ACKNOWLEDGEMENTS

A. Murat Tekalp has been supported by Turkish Academy of Sciences (TUBA).

REFERENCES