Parallelism In Video Streaming

Cameron Baharloo

ABSTRACT

Parallelism techniques are used in different parts of video streaming process to optimize performance and increase scalability, so a large number of client devices can play video, delivered over the Internet, with high quality and with no interruption. This paper provides an overview of these parallelism techniques at different phases of streaming video, from transcoding and uploading by service providers to their servers, to delivery over the network and playback on a client's device.

.

1. INTRODUCTION

Video streamed over the Internet has become a very common medium for consumers to watch a movie, TV show or any other type of video. It is becoming as common as the more traditional mediums such as DVD, or broadcast TV. Video streaming service providers such as Netflix, YouTube, Vudu, and Hulu are playing a major role in providing video content over the Internet to consumers, and by having a vast library of movies, TV shows and user generated video, they have become a major source of Internet traffic. In fact, at peak times during the day, Netflix traffic sometimes accounts for 30% of total Internet traffic in the U.S. In recent years, at the same time that video streaming has become more mainstream, the quality of videos has improved as well, such that all video streaming services provide HD quality video and some are just recently deploying Ultra High Definition (UHD) Video such as 4K UHD. Increased bandwidth by Internet Service Providers as well as increased computing power of client devices such as PC's and mobile phones are major factors that have allowed the ability to stream HD video over the Internet and playback of that HD quality video on end-user devices. But increase in bandwidth and computing power alone do not explain the increase in performance of video streaming in recent years. Other factors such as use of more innovative video compression technologies and more efficient delivery mechanisms play a role as well. One major factor to this increased performance is utilizing parallelism at different phases of video streaming. From the starting point of the service provider encoding a video for storage on their servers to the end point of video playback on a client device, parallelism techniques are used to increase performance and thereby prove a better video streaming user experience for consumers as well as lower costs for the service providers. This paper discusses parallelism at each step of the video streaming process with the main focus being parallelism techniques used in H.264 video compression which is used in encoding the video by the service provider and decoding that video on end-user device. Video compression is the main focus because that is where most of the parallelism in video streaming takes place. Also, parallelism used in other steps, such as parallel file upload, are not specific to video and have more general applications.

2. Overview of Video Streaming Process

A video goes through several steps before it is played back on the user's device.

1) *Transcoding:* Process of converting the original raw digital video file into a format suitable for online streaming. This process involves encoding the raw digital video into a compressed format such as H.264 so that its size is significantly reduced and is suitable for storage and transmission over the network.

2) *File Transfer:* The encoded and compressed video file is transferred into data servers or cloud storage. The transfer and storage of data should be efficient and scalable to address the performance requirements of video streaming. To achieve this, the transcoded video file is segmented into multiple chunks and uploaded in parallel.

3) *Content Delivery:* Upon client requesting a video, content delivery is initiated. Video streaming service providers use Content Delivery Networks (CDN) to efficiently deliver the video to client devices. CDN's are optimized to cache data, place video files at multiple global locations for global services, and many other delivery optimizations that are not done on the traditional model of delivering content through standard network servers. CDN's also support video transport protocols such as RTP, RTSP or HTTP Live Streaming (HLS).

4) *Video Decode and Playback:* When the video reaches the client device, it decodes the encoded video frames and plays it back on display. For example, for H.264 video, the H.264 decoder on the client device is responsible for this playback.

Figure 1 shows the end-to-end process of video streaming. In the following sections parallelism in each of the above steps is discussed, with more focus on steps 1 and 4, parallelism used encoding and decoding H.264 video.

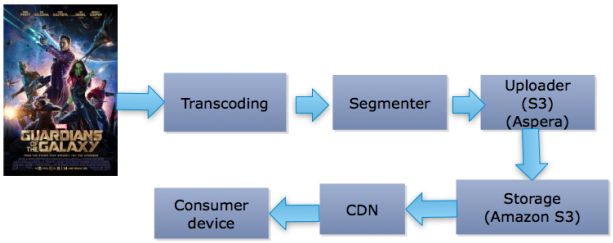

Figure 1. Video streaming steps

3. TRANSCODE PARALLELISM

To get raw video file into storage, it is encoded into a compressed format. The most common compression format currently in use is H.264 and it supports High Definition quality video. A 2 hour raw video source could be more than 500 GB and therefore it could take hours to encode. This long encoding time is not scalable for video streaming providers, especially if a content such as a TV show episode needs to be available online soon after the episode has aired. Video compression has many inherent parallelism properties that if utilized, provides significant speedup as opposed to sequential encoding. These parallelism techniques will be discussed in length in section 6 where decoding H.264 is discussed. In this section, the technologies specific to transcoding that enable scale and utilize parallelism techniques are discussed.

Transcoding is a computationally expensive operation and service providers are increasingly choosing to perform this operation in the cloud and distributed over their networks. This architecture provides great flexibility in that video streaming service providers do not even need to build their own infrastructure to transcode video content. They can use 3rd party vendors specialized in transcoding and file storage. These vendors perform video encoding on special High Performance Computing clusters in parallel. A raw video file is chunked into segments and each computing node of the cluster processes one or more segments in parallel. Parallelism used in encoding is at a lower granularity than parallelism performed for decoding where the goal is playback of the content and not just efficient and fast storage. Therefore, each computing node in clusters processes a segment of a few minutes of video whereas decoding parallelism is performed on one or more video frames which is at most a few seconds long.

The transcoding clusters usually have special hardware where the video compression algorithm is done in hardware rather than in software. The advantage of using hardware encoders is that it is faster than software encoding, but the disadvantage is that they provide less flexibility in terms of upgrades and updates to video compression process. But even for those architectures that use software, they use high level of parallelism using multiple processors and combination of CPU and GPU units.

4. FILE TRANSFER PARALLELISM

After a video is encoded, it needs to be transferred to storage servers. As mentioned in previous section, the video is processed in few minute chunks in parallel. Once each chunk is processed it is uploaded to storage servers, basically making the file transfer parallel. Cloud storage and distributed file systems are used for storing video content. And similar to transcoding, video streaming service providers use 3rd party vendors that have specialized cloud storage services. This eliminates the need for video streaming providers to build their own storage infrastructure that support complex parallel upload and file access. For example, Netflix uses Amazon Simple Storage Service (Amazon S3) for transfer and storing their video content. These storage solutions support parallel uploads and can reconstruct the original file from the transferred parts. In addition, cloud storage architectures have better scalability and parallelism support compared to traditional file system architectures. One common storage architecture used in cloud storage is Object Storage Architecture. It allows better utilization of parallelism by allowing compute nodes to access storage devices in parallel. It separates data and its metadata and distributes system metadata enabling shared file access without a central bottleneck.

With parallel file transfer support, each segment of video uploaded by transcoder is itself transferred in multiple chunks, meaning that for example a 5 minute encoded segment is transferred using multipart upload of 30 second chunks. The parallel upload, together with the previous step of parallel transcode, significantly speeds up the process of making video content available for delivery. Also using storage architectures, such as Object Storage, that better utilize parallelism, a better scalability is achieved in video file download as well.

Figure 2 shows a diagram of how a video is chunked into segments and encoded and uploaded in parallel to cloud storage services such as Amazon S3.

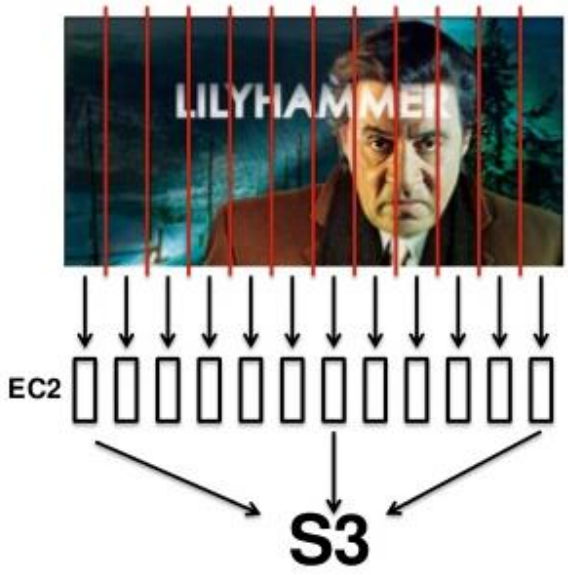

Figure 2. Parallel File Transfer

5. CONTENT DELIVERY PARALLELISM

Parallelism has limited use in content delivery because of the nature of video streaming. While for encoding and file storage the order of video segments during encoding and file transfer are not important as long as they are placed in correct order at the end, for video playback, the correct order is necessary during playback. A video frame cannot be played before all the frames before it is played. Therefore, parallel delivery and download has a smaller role.

One part of content delivery that parallelism has a role is in media streaming protocol. Traditional protocols used for video streaming, such as RTP and RTSP, deliver video over the UDP protocol. More recently HTTP Live Streaming (HLS) has become more common because Apple's iOS devices use it. HLS allows delivery over widely available CDN's because, unlike UDP streams, streams can traverse any firewall or proxy server.

HTTP Live Streaming supports Adaptive Bitrate Streaming which is dynamically adjusting the bitrates based on the current available bandwidth. Video is therefore stored with multiple bitrates and segments at different bitrates are provided for the same time frame of the video. If network bandwidth drops, the client device switches to the lower bitrate (lower quality) but it avoids interrupting the video stream and buffering. Parallelism can be utilized in HLS protocol where video segments from different bitrates are requested and sent in parallel. In addition, the receiver requests multiple segments in parallel by maintaining a limited number of HTTP sessions. It can then adapt media bitrates while receiving previously requested segments, and basically performing Adaptive Bitrate switches for next video segments while processing and playing back current video segment. This parallelism technique significantly reduces potential buffering and increases video playback quality.

6. VIDEO ENCODE/DECODE PARALLELISM

At the two ends of video streaming process, are video encoding, done at the service end, and the reverse step of video decoding, done at the client device end. Video coder-decoder (codec) is responsible for both tasks using hardware or software based codec implementation of different video formats. H.264 is the currently the most common video encoding format in video streaming. Tasks performed by video codecs are great candidates for utilizing parallelism and in fact all implementations of codecs support parallel processing that can be utilized at different configurations based on hardware capabilities. For instance a higher degree of parallelism, and therefore more complex parallelism techniques, are used on encoding clusters compared to the degree of parallelism used for the same video on a desktop computer or mobile device. In the following sections, an overview of H.264 is given followed by different parallelism techniques used in H.264 codecs.

6.1 H.264 Overview

H.264 or MPEG-4 Part 10 Advanced Video Coding (MPEG-4 AVC) is currently the most widely used video compression format for streaming video. It is a block-oriented, motion-compensation based video compression standard. Motion compensation is a coding algorithm technique used to predict a video frame based on previous frames and therefore enabling compression of the video by eliminating the need to store full frame data of the predicted frame. H.264 was created by ITU-T Video Coding Expert Group (VCEG) and ISO/IEC Motion Picture Expert Group (MPEG). H.264 was developed to create better quality video at lower bit rates compared to older standards such as MPEG-2. In fact it has half the bit rate of MPEG-2 video. This is achieved by having a more efficient compression algorithm and an increased compression rate. As a result, the capacity and network bandwidth needed to store and transmit HD quality video is reduced which reduces cost of storage and transmission by the service and network providers. At the heart of H.264 compression is using motion-compensation. It takes advantage of the fact that neighboring frames of a movie have slight differences in terms of motion, either camera or an object has moved slightly. Using information from one frame, the next frame can be predicted. As a result, some full reference frames are stored in entirety. Using just the difference of a reference frame from next neighbor frames, which is usually small, the neighboring frames can be predicted and then reconstructed at decoding time. By storing partial data for these predicted frames, compression is achieved. A video in H.264 has hierarchical structure and is structured so that a sequence of frames, usually starting with a reference I frame followed by multiple P and B frames, form a Group Of Pictures (GOP) unit. Each GOP is also divided into frames and each frame is divided into independent segments called slice. Each slice is divided into small square blocks of 16x16 pixels called Macroblocks. Macroblocks are divided into 4x4 blocks which are composed of pixels. Figure 3 shows H.264 frame hierarchy and structure.

The major tasks of H.264 encoder are Prediction, Transform and Encode. The decoder performs the opposite, Decode, Inverse Transform and Reconstruct.

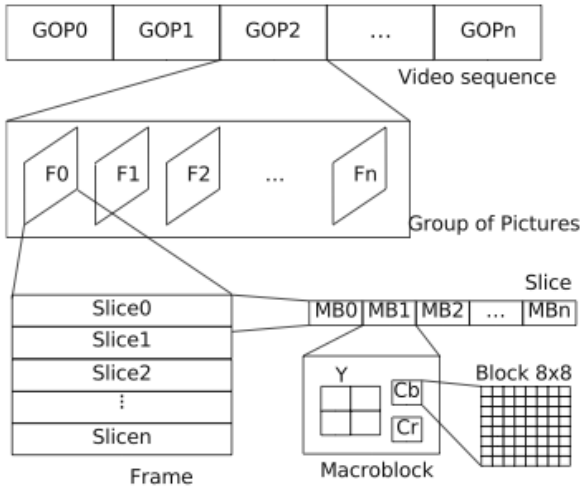

Figure 3. H.264 Frame Structure

6.1.1 Prediction

H.264 encoding is done by processing a 16x16 block of pixels, referred to as Macroblock, and predicting the Macroblock either

from encoded data of neighboring blocks within the same frame (Intra-frame prediction), or from already encoded data from similar blocks of previous frames (Inter-frame prediction). Intra-predictions use spatial redundancy inside a frame to reduce the amout of data necessary to encode adjacent blocks. Given that at most times, adjacent blocks within a frame have very little difference, using Intra-prediction enables greatly reducing the size of a single frame. Inter-prediction uses motion compensation as discussed above, and basically takes advantage of temporal redundancy between frames as the frames in a sequence have great chance to be similar.

Video frames in H.264 are classified into I, P and B frames. I frame stands for Intra-frame and has no data dependency on other frames. These are the reference frames and their full data is stored, hence, no data dependency on other frames. P frame stands for Predicted frame. It is predicted from previously encoded I or P frame. B frame stands for Bi-directional predicted frame. It is predicted from two previous I or P frames.

6.1.2 Transform

A block of pixels is transformed using 4x4 or 8x8 integer transform, an approximate form of the Discrete Cosine Transform (DCT). DCT is a reversible and lossless operation, meaning the encoder and the decoder do the exact same thing, and that it does not directly help data compression, but just reorganize the coefficients of a block by frequency order, with the high frequencies at the beginning of the block, and the low frequencies at the end. The DCT transformation operations are basically matrix calculus and therefore lend themselves well for parallel processing. The same DCT is applied to all of the blocks of the picture.

6.1.3 Encode

The video coding process produces a number of values that must be encoded to form the compressed bit stream. These values include information to enable the decoder to re-create the prediction, information about structure of compressed data, information about the complete video sequence and quantized transform coefficients obtained at the Transform stage. In H.264, these values are converted into binary codes using Entropy Coding, a form of variable-length arithmetic coding algorithm. An entropy encoder compresses data by replacing each fixed-length input symbol by the corresponding variable-length prefix-free output code word. Entropy coding is a very linear process and cannot be parallelized.

6.2 H.264 Parallelism Methods

H.264 codec implementations use two types of parallelism:

1) *Task-Level Parallelism:* Functional decomposition of the algorithm where parts of the functions performed on a frame, or Group Of Pictures, is assigned to different processors.
2) *Data-Level Parallelism:* Each part of the video structure hierarchy, GOP, frame, slice, and so on, can be explored as units of processing for parallelism, where an operation is performed on multiple units at the same time. For instance multiple slices of a frame can be decoded in parallel.

6.3 Parallelism Among Decode Stages

Performing decode stages in parallel is the task-level parallelism. The idea is to decompose the encode/decode stages into independent functions that can be performed in parallel. In practice, implementations of task-level parallelism utilize a multi-stage pipelining strategy, very similar to pipelining used in processors for instruction processing. For example, a four stage pipeline of parsing, entropy decoding, macroblock decoding, and filtering is used in a pipelined encoding architecture. Each stage can be done in parallel and assigned to different processor. Note that filtering is a final decoding step used to prevent blocking artifacts resulted from transformations applied to blocks. Also, some stages of the pipeline such as macroblock decoding, can use data-level parallelism since independent blocks can be processed in parallel.

6.4 Data-Level Parallelism

In the following subsections, data-level parallelism techniques used at different granularity of data unit, GOP, frame, slice, and so on, are discussed.

6.4.1 *Group Of Pictures-Level Parallelism:* Multiple frames that form a GOP are assigned to a processor or computing node. This requires a lot of memory for the processor because it is using bigger chunks of data. It is therefore better suited for multicomputer encoders or transcoding clusters discussed earlier and not suited for a codec on a mobile device. Note that each processor can also perform further parallelization using multiple threads, for instance assigning a frame to each thread.

6.4.2 Frame-Level Parallelism

Each frame is assigned to a processor. Frames have data dependency as discussed earlier. P frames have dependency on previous reference I frame or P frames. B frames have data dependency on two previous I or P frames. Because of data dependencies in P and B frames, it is harder to encode multiple P and B frames at the same time.

If P frame encoding is complete, then dependent P and B frame processing can start. To optimally utilize parallelism, I and P frames get encoded with higher priority than B frames.

Due to data dependencies between frames, it is more complex to optimally parallelize at the frame level.

6.4.3 *Slice-Level Parallelism:* Slices within a frame are independent and therefore good candidates for parallelism. System can simultaneously decode all slices in any order. One disadvantage is the increasing bit-rate. Slices break the dependence between macroblocks and decrease compression efficiency because macroblocks from one slice cannot utilize a macroblock from an adjacent slice for prediction compression. Frame-level and GOP-level parallelism enable more efficient compression and increased bit-rate.

6.4.4 *Macroblock-Level Parallelism:* Parallelism at the macroblock level can be used using a spatial domain or temporal domain.

Spatial domain is processing macroblocks of a single frame and it can be parallelized when all intra-frame dependencies are satisfied. This is basically intra-frame parallel processing of macroblocks.

Temporal domain is processing macroblocks of different frames in parallel. For temporal domain, both intra-frame, and inter-frame dependencies of a macroblock must be satisfied before it can be processed.

In practice, spatial and time domain parallelism for macroblocks are combined. This is referred to as 3D Wave Parallelism (Figure 4) and it takes advantage of the fact that a frame only depends on reference area of its previous frame. Once that reference area is decoded, decoding of the new frame can start and the processing will not have to wait for the entire reference frame's decoding to complete. This allows intra-frame parallelization in addition to inter-frame parallelization. While a reference frame is still being processed, parts of the next dependent frames can be processed as well.

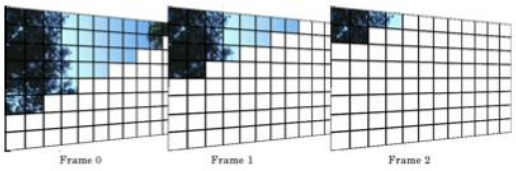

Figure 4. 3D-Wave Parallelisation

6.5 H.264 Parallelism Summary

In practice, both task-level and data-level parallelism is used to process a video. Also, data-level parallelism is used at different granularities at the same time, so all the parallelism techniques mentioned above are combined to speedup video encoding and decoding. For example, multiple frames (GOP) can be assigned to a computing node in a cluster or a processor core within a multi-core computer, and each processor can perform more fin-grained parallel processing at frame, slice or macroblock level.

6.6 H.264 Parallelism Implementations

There are H.264 hardware video encoders such as Intel QuickSync and some applications such as OSX Airplay Mirroring use this hardware accelerated video decoding. Hardware codecs are also utilized in transcoding clusters to encode raw digital video files very fast.

There are software implementations such as ffmpeg that can be configured to utilize parallelism and use both CPU and GPU for parallelism. ffmpeg H.264 codec is the most common implementation on client devices such as PC's, set-top boxes and mobile phones.

6.7 Next Generation Video Compression

The next generation encoding is High Efficiency Video Coding (HEVC or H.265), which is an evolution of H.264 and allows for even more degree of parallelism. It requires 50% less bit-rate for the same quality as H.264. It is therefore suited for Ultra HD (4K HD) quality video streaming. It is already being rolled out by services such as Netflix.

Competing encodings also exist. The main ones are VP8 and VP9 from Google that do not have patent and licensing issues that H.264 and H.265 have. They have similar architecture (block based using motion compensation) and the same parallelism techniques are used for those formats in current implementations such as ffmpeg.

7. CONCLUSION

With more and more people using online video steaming as the main source for watching a movie or TV shows, the performance requirement for video streams become more important. In addition, with the availability of higher quality formats, such as Ultra High Definition video, it becomes necessary to encode, store, transmit and decode a video very efficiently to provide high quality video at reasonable cost and at scale. Utilizing parallelism at different stages of video streaming can significantly contribute to achieving the performance requirement goals of streaming high quality video. Specifically, video codecs are great candidates for incorporating parallelism techniques in their encode/decode operations. H.264 and next generation H.265 implementations heavily use parallelism in their configurations.

8. REFERENCES

[1] Mauricio Alvarez Mesa, Chi Ching Chi, Thomas Schier and Ben Juurlink.. Evaluation of Parallelization Strategies for Emerging HEVC Standard. 2012 IEEE International Conference on Acoustics, Speech and Signal Processing.

[2] Cor Meenderinck, Arnaldo Azevedo, Ben Juurlink, Mauricio Alvarez Mesa, Alex Ramirez. Parallel Scalibility of Video Decoders. Journal of Signal Processing Systems, November 2009.

[3] Amazon Web Services, Netflix's Transcoding Transformation. Streaming Media East Conference 2013.

[4] Richard Gerber, Aart J.C Bik, Kevin B. Smith, Xinmin Tian. Optimizing Video Encoding Using Threads and Parallelism. Dr Dobb's Journal, June 11, 2010.

[5] Kiran Misra, Jie Zhao, and Andrew Segall. Entropy Slices for parallel entropy coding. Technical Report JCTVC-B111, July 2010.

[6] H.264 Advanced video coding for generic audiovisual services. ITU-T publication.

[7] Genhua Jin, Hyuk-Jae Lee. A Parallel and Pipelined Execution of H.264/AVC Intra Prediction. The Sixth IEEE International Conference on Computer and Information Technology, 2006.