Lecture 14:

Processing Video at Cloud Scale

Visual Computing Systems
Stanford CS348V, Winter 2018
Big video data

Facebook 2016: 100 million hours of video watched per day

Youtube 2015: 300 hours uploaded per minute [Youtube]

FB Live Video

Snapchat Video

Netflix

Twitch
Today’s topic

- Challenges and key ideas of datacenter-scale video processing

- Three examples:
  - Video upload at Facebook
  - Massively parallel video encoding
  - A data-parallel framework for video processing
Facebook Streaming Video Engine (SVE)

- Designed for non-streaming video upload (not Facebook Live)
  - Facebook video posts
  - FB Messenger video shares
  - Instagram Stories
  - 360 videos

- Goals/requirements:
  - Low latency: minimize latency of start of upload to sharable state
    - Particularly for FB Messenger uploads
  - Flexible (support variety of applications, with different processing pipelines after upload)
  - Robust to faults and overload
Basic video sharing pipeline

1. Record

Client 1

2. Video upload

Datacenter

3. Process

(validation, reencoding, video analysis, thumbnail extract)

4. Store

5. Share Event

6. Stream to viewer

Client 2
Video upload and processing times *

File size of video

<table>
<thead>
<tr>
<th>Range</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>min, 1M</td>
<td>red</td>
</tr>
<tr>
<td>[1, 3M)</td>
<td>green</td>
</tr>
<tr>
<td>[3, 10M)</td>
<td>blue</td>
</tr>
<tr>
<td>[10, 30M)</td>
<td>orange</td>
</tr>
<tr>
<td>[30, 100M)</td>
<td>yellow</td>
</tr>
<tr>
<td>[100, 300M)</td>
<td>cyan</td>
</tr>
<tr>
<td>[300M, 1G)</td>
<td>red</td>
</tr>
<tr>
<td>[1G, max)</td>
<td>black</td>
</tr>
</tbody>
</table>

![Upload time CDF](image1)

![Encode time CDF](image2)

* Serialized times (SVE system will parallelize encoding across segments as discussed in a few slides)
Pipelining upload and processing

- Client application partitions video into segments prior to upload
- Client application optionally downsamples video (skipped if video recorded at low enough resolution, internet connection is fast, or device does not support HW accelerated encode)
- Upload and processing of video is pipelined (upload and processing is mostly parallelized)
- Processing itself can be parallelized across segments
This dimension exposes more parallelism, but increases complexity. At this point, the re-encoding tasks, which are the most computationally intensive, are operating at the maximum parallelism: segments of individual tracks. Thumbnail generation, which is also moderately time consuming, is grouped as a subset of a task group. However, some tasks, such as speech recognition, are operating at the segment level. Our classiﬁcation is more complex than one big script. This provides SVE with easy syntax for programmers to add most tasks. Most tasks specifying the notify task to be latency sensitive.

Figure 7: Simpliﬁed DAG for processing videos. The initial video is split into tracks for video, audio, and metadata. The video track contains the video track, while the audio track contains the audio track. Tasks can operate on either one track individually or all tracks together. Specifying a task to operate on an individual track enables SVE to extract some parallelism and is simple for programmers. For example, speech extraction and facial recognition only require the video track. The naturally processed time, which gives us most of the best of both worlds of parallelism and simplicity.

Figure 8: Pseudo-code for generating the simplified DAG.

```python
pipeline = create_pipeline(video)

video_track = pipeline.create_video_track()
if video.should_encode_hd
    hd_video = video_track.add(hd_encoding)
    .add(count_segments)

sd_video = video_track.add(
    {sd_encoding, thumbnail_generation},
).add(count_segments)

audio_track = pipeline.create_audio_track()

sd_audio = audio_track.add(sd_encoding)

meta_track = 
    pipeline.create_metadata_track()
    .add(analysis)

pipeline.sync_point(
    {hd_video, sd_video, sd_audio},
    combine_tracks,
).add(notify, 'latency_sensitive')
    .add(video_classification)
```

DAG node = “task”
Each task is executed serially on one video segment
Overall DAG execution can be parallelized
(assuming tasks and segments)

Facebook Video Posts: ~153 tasks
Messanger shares: 18 tasks
Instagram stories: 22 tasks
Coarse-grained parallel video encoding

- Parallelized across segments (Iframe inserted at start of segment)
- Concatenate independently encoded bitstreams

Smaller segments = more potential parallelism, worse video compression
Latency-sensitive applications: 10 second segments
Non-latency sensitive, long videos: 2 minute segments (maximize compression)
Overload control

- When FB cannot keep up with upload rate...

- Delay latency-insensitive tasks
- Redirect uploads to new datacenter region
- Delay processing of new uploads
Massively parallel video encoding
Challenge

- Video encoding is inherently sequential (current frame represented in terms of data from previous frames)
- Coarse-grained parallelism is possible, at the cost of reduced compression (additional keyframes)
  - 4K video: keyframe ~ 11MB, interframe ~ 10s of KB
- Growing interest in “Serverless computing”

Today’s AWS Lambda Pricing
$0.000002 per request + $0.00001667 per GB/sec

<table>
<thead>
<tr>
<th>Memory (MB)</th>
<th>Free tier seconds per month</th>
<th>Price per 100ms ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>3,200,000</td>
<td>0.0000000208</td>
</tr>
<tr>
<td>192</td>
<td>2,133,333</td>
<td>0.0000000313</td>
</tr>
<tr>
<td>256</td>
<td>1,600,000</td>
<td>0.0000000417</td>
</tr>
</tbody>
</table>

Question: Is it possible to parallelize video encode onto thousands of cores?
Expressing a video encoder in a continuation passing style

// prob model: tables representing encoding of values in video stream
// reference_images contains three prior images
state := (prob_model, reference_images[3]);

// just a full image
keyframe := image pixels for entire frame

// prediction_modes and motion vectors define how to predict current
// frame given decoder state
// residue is correction to this prediction
interframe := (prediction_modes, motion_vectors, residue)

// decoding a frame generates one image of pixels, and
// an updated decoder state
decode(state, compressed_frame) -> (new_state, image)

// generate an interframe approximating image given the current
// decoder state. This operation requires expensive motion estimation.
encode-given-state(state, image, quality_param) -> interframe

// use prediction info from interframe to estimate prediction for
// image given new state, store new residue in new_interface
// Note: rebase is cheap because it does not perform motion estimation
rebase(new_state, image, interframe) -> new_interframe
Massively parallel video encoding using “rebase”

// N = frames per thread
// x = threads to use
ParallelEncode[N,x]

In parallel, each thread encodes N consecutive frames of video
(generates 1 keyframe + N-1 interframes)

In parallel, each thread decodes its N frames of video
Thread t passes its final decoder state to thread t+1.

In parallel, each thread executes encode-given-state() on its first frame
using the decoder state it receives, replacing what was a keyframe with an
interframe that is dependent on decoding the prior thread’s output.
(requires the original image)

In sequence, each thread rebases its remaining frames based on state after
decoding first frame. When complete, thread t sends final decoder state
to thread t+1
Computation schedule for ParallelEncode[16,6]

Using 16 threads to encode $16 \times 6 = 96$ frames of video
Leverage wide parallelism + preserve reasonable video quality

ExCamera results are on 3600 cores

<table>
<thead>
<tr>
<th>System</th>
<th>Bitrate at 20 dB SSIM</th>
<th>Encode time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExCamera[6,16]</td>
<td>27.4 Mbps</td>
<td>2.6 minutes</td>
</tr>
<tr>
<td>ExCamera[6,1]</td>
<td>43.1 Mbps</td>
<td>0.5 minutes</td>
</tr>
<tr>
<td>vpxenc multi-threaded</td>
<td>27.2 Mbps</td>
<td>149 minutes</td>
</tr>
<tr>
<td>vpxenc single-threaded</td>
<td>22.0 Mbps</td>
<td>453 minutes</td>
</tr>
</tbody>
</table>

Stanford CS348V, Winter 2018
Scanner
Motivating questions

If I wanted to grab a few terabytes of video, store it in a database, and perform pixel-level analyses on frames from the collection using a cluster of high-compute-density nodes, what system should I use?

If large-scale, frame-level processing was widely available, what are new types of questions a data analyst might ask about increasingly abundant video datasets?

What are the software tools needed to conduct visual data mining on large video collections?
"KrishnaCam" egocentric video dataset

72 hours of recording over nine months: (Sep 2014 – May 2015)

Google Glass
Ensemble of face detectors for KrishnaCam
Large-scale autonomous vehicle datasets

[Image Credit: Kundu et al. 2016]
American TV news dataset

- 3 years of CNN, FOX, MSNBC
- 72,000 hours of video, 12 billion frames
Example questions

- Where are the commercials?
  - Largely to exclude from analysis, but may wish to analyze commercials independently?

- Can we break the video into shots?
Geena Davis Inclusion Quotient (GD-IQ)

Project between Google and The Geena Davis Institute on Gender in Media

Men are seen and heard nearly twice as often as women

36% Female on-screen time
35% Female speaking time

Women are seen on-screen more than men only in one film genre: horror

<table>
<thead>
<tr>
<th>Genre</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horror</td>
<td>53%</td>
</tr>
<tr>
<td>Romance</td>
<td>45%</td>
</tr>
<tr>
<td>Comedy</td>
<td>40%</td>
</tr>
<tr>
<td>Sci-Fi</td>
<td>36%</td>
</tr>
<tr>
<td>Drama</td>
<td>34%</td>
</tr>
<tr>
<td>Action</td>
<td>29%</td>
</tr>
<tr>
<td>Biography</td>
<td>30%</td>
</tr>
<tr>
<td>Crime</td>
<td>23%</td>
</tr>
</tbody>
</table>

American TV news analysis (first 100 hrs)

- Fareed Zakaria GPS: 17% female
- CNN Newsroom: 43%
- Situation Room: 32%
- CNN Newsroom with Poppy Harlow: 40%
- The Lead with Jake Tapper: 32%
- America News Headquarters: 35%
- The Five: 40%
- The Real Story With Gretchen Carlson: 45%
- Shepard Smith Reporting: 27%
- On the Record With Brit Hume: 28%
Cinematography analysis
Collaboration with Alex Hall, Maneesh Agrawala (Stanford)

What is the average length of shot in a movie?

Does the director favor close ups or wide shots? How much camera motion is used?

What are the main color palettes in the film?

How do these traits vary across films or time?

“Star Wars Episode IV: a New Hope”
Segmented into shot boundaries based on image histograms
Sensing human social interactions

CMU Panoptic Studio
480 video cameras (640 x 480 @ 24fps)
147 MPixel video sensor
(3.5 GPixel/sec)

[Joo 2015]
Data from the CMU Panoptic Studio
Capturing human social interactions

40-second sequence

3D pose reconstruction time: hand-coded solution — 7 hours on 4-Titan Xp’s [Cao 2016]

[Courtesy Yaser Sheikh, Tomas Simon, Hanbyul Joo] [Joo et al. 2015]
Facebook Surround 360 VR video

14 2K x 2K cameras

2048 x 2048 PointGrey Camera @ 30 FPS

14 cameras

8K x 8K stereo panorama output = 12.5 secs per frame on 32-core CPU
**Workload**

- 100’s to 1000’s of videos, different video lengths

- Sparse frame sampling: evaluate a kernel on every (Nth) frame, or on a specific list of frames (every frame with people)
  - DNN inference (detection, facial landmarks, segmentation, etc.)

- Operations that require a window of multiple frames in sequence
  - Optical flow, hyperlapse sliding windows, etc…

- Operations with temporal dependencies (frame-to-frame state)
  - Object tracking

- Operations that combine multiple video streams
Scanner: design goals / principles

Design principle 1: keep it simple
- Enable non-expert programmers (computer vision researchers, visual data analysts) to rapidly develop and deploy video analysis applications at cloud scale

Design principle 2: be near speed-of-light efficient
- “Near-HW-peak single-node perf”, then scale out
- Utilize heterogenous hardware: ASICs for video encode/decode, run kernels on multi-core CPUs, GPUs, future DNN accelerators
What do I mean by scale? axes of scaling

Scale to arbitrarily large video datasets:
- A modest number of long videos (hundreds of feature films)
- Many short videos (1M YouTube videos)

Scale to large numbers of compute resources:
- Efficiently utilize a single large parallel machine (grad student workstation)
- Efficiently utilize many machines (100’s GPUs or 1000’s CPUs in the cloud)

Scale to real-time processing:
- Processing (and joining results of) multiple video streams

Current non-goal of Scanner: scaling to very large single images (satellite images, maps, etc.)
Basic Scanner workflow

I have a list of videos in a filesystem...
myvideos/vid00.mp4
myvideos/vid01.mp4
myvideos/vid02.mp4
...
myvideos/vid99.mp4

And I have a library of parallel pixel processing kernels for CPUs and GPUs:

- Image crop/rescale (Halide)
- Depth from disparity (Halide)
- Optical flow (OpenCV)
- Eigen (C)
- Video Tracker (CUDA)
- Caffe DNN Eval
- NVIDIA cuDNN
- Human Pose estimation [Cao16]
- Object detector [Redmon16]
- Face detection network [Hu17]
- Depth/normal estimator [Bansal17]
Represent videos as relations (tables)

myvideos/vid00.mp4
myvideos/vid01.mp4
myvideos/vid02.mp4
...
myvideos/vid99.mp4

Ingest into Scanner...

videos = ['vid_00.mp4',..., 'vid_99.mp4']
db = scanner.Database();
video_tables = db.ingest_videos(videos)
videos = ['vid_00.mp4',..., 'vid_99.mp4']
db = scanner.Database();
video_tables = db.ingest_videos(videos)

frame = db.ops.FrameInput()
sparse_frames = frame.stride(10)
resized = db.ops.Resize(
    frame = sparse_frames,
    width = 496, height = 398)
detections = db.ops.DNN(
    frame = resized,
    model = 'face_dnn.prototxt',
    batch = 8)
frame_detections = detections.space(10)
faces = db.ops.Track(
    frame = frame,
    detections = frame_detections,
    warmup = 20)
output = db.ops.Output(columns=[faces])

* Similar to TensorFlow graph, or SVE processing graph
Scanner dataflow graph operations

Sequence sampling operations (stride, range, indexed gather)
Output sequence contains subset of elements of input sequence

**e.g.**
*Run on every 5th frame*
*Run on a list of frames known to contain faces*

Note: this is sampling, it is not data-dependent filtering
Scanner dataflow graph operations

Bounded stateful operations
Each output element depends on results of processing prior output element
Runtime guarantees effects from at least W prior "warmup" elements will be visible

e.g.
Object tracking
Minimizing temporal discontinuities while running in parallel
DAGs of data-parallel operations
map, stencil, stride, fold, bounded_fold, partition/flatten

(a) Map (with batching)

(b) Stencil

(c) Strided Sampling

(d) Spacing

(e) Dense Strided Stencil

(f) Sparse Strided Stencil

(g) Bounded State (Warmup=2)

not shown:
scan_sequential
partition/flatten
Scanner runtime
Parallel execution in Scanner

Work distribution supports mid-job node failure and addition
Google Cloud Storage / S3 bindings for storage

Node 0: multi-core CPU + GPU

Node 1: multi-core CPU + 2 GPUs

video.mp4
Per-element (sparse) dependency analysis

(a) Map (with batching)
(b) Stencil
(c) Strided Sampling
(d) Spacing
(e) Dense Strided Stencil
(f) Sparse Strided Stencil
(g) Bounded State (Warmup=2)
Accelerated parallel sparse frame decode using index of keyframe locations

Scanner maintains index of keyframe locations to enable work-efficient parallel decode
Scanner performance
Efficient video decode under sparse access

(1080p video)

Video Decode Throughput

<table>
<thead>
<tr>
<th>Throughput (relative to OpenCV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS</td>
</tr>
<tr>
<td>A: STRIDE-1</td>
</tr>
<tr>
<td>B: STRIDE-24</td>
</tr>
<tr>
<td>C: RANGE</td>
</tr>
<tr>
<td>D: KEYFRAME</td>
</tr>
<tr>
<td>E: GATHER</td>
</tr>
</tbody>
</table>

16-core CPU, Single-Instance

- A: STRIDE-1
- B: STRIDE-24
- C: RANGE
- D: KEYFRAME
- E: GATHER

16-core CPU, Multi-Instance

- A: STRIDE-1
- B: STRIDE-24
- C: RANGE
- D: KEYFRAME
- E: GATHER

1 GPU

- A: STRIDE-1
- B: STRIDE-24
- C: RANGE
- D: KEYFRAME
- E: GATHER

Stanford CS348V, Winter 2018
Scaling to bigger machines
Scalability to many cloud machines

Using many resources to process a 2.2 hour movie quickly
(reduce latency of analysis)

Large Dataset Scalability

Using many resources to process database of many videos
Facebook Surround 360 VR video generation

44-node computation DAG
8K x 8K stereo panorama output = 12.5 secs per frame on 32-core CPU

Joins multiple video streams
Bounded state operations (smoothing across parallel partitions)

15 minute sequence: (103 GB)

Single node Scanner (32-core CPU)
- 5 secs / frame

Multi-node on Google Compute Engine
- 64 x 32-core nodes (2048 cores)
- Approaching real-time
3D human pose reconstruction

Processing 40 seconds of video from CMU Panoptic studio

Grad student hand-tuned: 7 hrs (1 node x 4 Titan Xp GPUs)
Scanner: 2.6 hrs (1 node x 4 Titan Xp GPUs)
Scanner on GCE: 25 mins (25 nodes x 4 K80 GPUs = 100 K80s)

Performance makes extended capture sessions possible
Today’s summary

- Last time: algorithmic techniques for increasing efficiency of video processing

- Today: Cloud-scale infrastructure for processing large amounts of video at scale

- Good example of need for new infrastructure (to run at scale) AND new algorithms (to reduce the amount of work done)
  - The cost of per-frame processing remains way too high to mine large video collections in a brute force manner
  - Running three detectors: Faster R-CNN, Convolutional Pose Machines, TinyFaces on all 6M 720p frames on GCP ~ $3400
  - Currently resort to heavy use of subsampling frames