Encoding, Fast and Slow:
Low-Latency Video Processing Using Thousands of Tiny Threads

(To appear in NSDI’17)

Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthik Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, Keith Winstein

https://github.com/excamera
Outline

- Vision & Goals
- Background on Video Processing
- Fine-grained Parallel Video Encoding
- \( \mu \): Supercomputing as a Service
- Evaluation
- Conclusion & Future Work
Google Docs for Video

* not really a Google product (yet)
"Make this movie black and white."
"Apply some awesome filter to my video."
"Pixelate this face everywhere in this video."
"Remake Star Wars Episode I without Jar Jar."
Can we achieve interactive speeds with very granular video processing in a distributed system?
The dilemmas

- Low-latency video processing requires fine-grained parallelism, but **the finer-grain the parallelism, the worse the compression efficiency**.

- Even if we have a way to avoid this penalty, we still need **thousands of threads, running in parallel**, with instant startup.
ExCamera

• In this project, we tried to directly address these two dilemmas.

• We made two contributions:

• A video encoder intended for massive fine-grained parallelism (the first dilemma).

• A framework that orchestrates thousands of threads running in parallel on AWS Lambda (the second dilemma).

• We call the whole system ExCamera.
Fine-grained Cloud Computing

• Parallel make

• "Laptop Extension"
Outline

• Vision & Goals

• Background on Video Processing

• Fine-grained Parallel Video Encoding

• μ: Supercomputing as a Service

• Evaluation

• Conclusion & Future Work
What's a video?

• A series of still images, displayed in order, at a specific rate.

• Movies are usually shown at 24 frames per second.
4K Video

4096×2160 pixels/frame

~14 MB/frame

24 fps

2.5 Gbps
Video Codec

- A piece of software or hardware that compresses and decompresses digital video.
How to compress?

• Compress each frame individually.

• JPEG Compression of a 4K frame:
  ~1 MB/frame → 192 Mbps
How to compress?

• Exploit the temporal redundancy in adjacent frames.

• Store the first frame on its entirety: a key frame.

• For other frames, we could just store the “diff” with the previous frame: interframes.
Encoder & Decoder

\[
\text{encode}(\text{img}[1..n]) \rightarrow [kf] + \text{if}[2..n]
\]

\[
\text{decode}([kf] + \text{if}[2..n]) \rightarrow \text{img}'[1..n]
\]
4K Video Encoded with VP8

Key Frame Size: ~1 MB

Inter Frame Size: ~25 KB
Traditional Parallel Video Encoding

\[\text{encode}(i[1..200]) \rightarrow [kf_1] + if[2..200] \]
\[\vdots\]
\[\text{encode}(i[1..10]) \rightarrow [kf_1], if[2..10]\]
\[\text{encode}(i[11..20]) \rightarrow [kf_{11}], if[12..20]\]
\[\text{encode}(i[21..30]) \rightarrow [kf_{21}], if[22..30]\]
\[\vdots\]
\[\text{encode}(i[191..200]) \rightarrow [kf_{191}], if[192..200]\]

Finer-grained parallelism \(\rightarrow\) more frequent key frames
Outline

• Vision & Goals
• Background on Video Processing
• Fine-grained Parallel Video Encoding
• \( \mu \): Supercomputing as a Service
• Evaluation
• Conclusion & Future Work
14.8-minute 4K Video

Encoding with vpxenc (VP8)

Single-Threaded  ~7.5 hours
Multi-Threaded  ~2.5 hours
Decoder State

- Decoder needs to maintain some information that evolves with each decoded frame.

- Traditional video codecs do not expose this information.

\[
\text{encode}(\text{img}[1..n]) \rightarrow [\text{kf}] + \text{if}[2..n] \\
\text{decode}([\text{kf}] + \text{if}[2..n]) \rightarrow \text{img'}[1..n]
\]

- We call this the \textit{state} and we made it \textit{explicit}.
Decoder State

source state
Decoder State

frame

source state

target state
Decoder State

output

source state

frame

target state
Decoder State

key frame → interframe → interframe → interframe
What we built: a video codec in explicit state-passing style

- VP8 decoder with no inner state:

  $$\text{decode}(\text{state}, \text{frame}) \rightarrow (\text{state}', \text{image})$$

- VP8 encoder: resume from specified state

  $$\text{encode}(\text{state}, \text{image}) \rightarrow (\text{state}', \text{interframe})$$
Divide the work into tiny tasks (6 frames each).

[Parallel, the slow work] Make tiny independent chunks.

[Serial, the fast work] Stitch the chunks together and remove keyframes.
ExCamera: Encoding, Fast and Slow

• Divide the video into 4-second batches.

• Each batch is divided further to 16 chunks, $\frac{1}{4}$ second each.
1. [Parallel] Download 6-frame chunk of raw video

- **thread 1**: 1, 5, 6
- **thread 2**: 7, 11, 12
- **thread 3**: 13, 17, 18
- **thread 4**: 19, 23, 24
2. [Parallel] \texttt{vpxenc} → keyframe, interframe[5]

Google's VP8 encoder

\texttt{encode(img[1..n])} \rightarrow [kf] + if[2..n]
3. [Parallel] decode $\rightarrow$ state $\rightarrow$ next thread

Our explicit-state style decoder

decode(state, frame) $\rightarrow$ (state', image)
4. [Parallel] *last thread’s state* $\rightarrow$ *encode*

Our explicit-state style encoder

$\text{encode}(\text{state, image}) \rightarrow (\text{state}', \text{ interframe})$
4. [Parallel] last thread’s state $\rightarrow$ encode

Adapt a frame to a different source state

$\text{rebase}(\text{state, image, interframe}) \rightarrow \text{interframe'}$
5. [Serial] \textit{last thread’s state} $\xrightarrow{\text{rebase}} \text{state} \xrightarrow{\text{next thread}}$

Adapt a frame to a different source state
\textbf{rebase}(\textit{state, image, interframe}) $\rightarrow$ \textit{interframe}'}
5. [Serial] *last thread’s state* $\rightarrow$ rebase $\rightarrow$ state $\rightarrow$ *next thread*

Adapt a frame to a different source state

\[\text{rebase} (\text{state}, \text{image}, \text{interframe}) \rightarrow \text{interframe}'\]
6. [Parallel] Upload finished video
Thousands of tiny threads
Outline

• Vision & Goals
• Background on Video Processing
• Fine-grained Parallel Video Encoding
• μ: Supercomputing as a Service
• Evaluation
• Conclusion & Future Work
Execution Environment

• We need an execution environment, where we can run thousands of threads in parallel, each for a few minutes.

• Cloud functions!
AWS Lambda is an underutilized supercomputer

- Intended for event handlers and Web microservices
- Supports JavaScript (node.js), Java, Python
- But in practice, you can upload a zip file containing a binary executable and simply exec it from the Python handler
Lambda can launch 3,600 threads in 3 seconds
Hardware

- 1.5 GiB RAM, 2.8 GHz CPU, very little disk space
- 5-minute execution limit
Costs

- **Price:** 2.5 milli-cents per second (billed in 100 ms increments)

- 3,600 threads for one minute → $5.40

- Unique features:
  - Sub-second billing,
  - Thousands of threads,
  - Fast startup.
μ, supercomputing as a service

- We built μ, a library for designing and deploying massively parallel computations on AWS Lambda.

- ExCamera is just an example, possibilities are endless.
Outline

- Vision & Goals
- Background on Video Processing
- Fine-grained Parallel Video Encoding
- μ: Supercomputing as a Service
- Evaluation
- Conclusion & Future Work
Quality vs. Bitrate

SSIM (dB) vs. average bitrate (Mbit/s)

- vpx (single-threaded)
- vpx (multithreaded)
Quality vs. Bitrate

![Graph showing the relationship between SSIM (dB) and average bitrate (Mbit/s) for different VPX configurations: single-threaded, multithreaded, and with 6-frame chunks.](image-url)
Quality vs. Bitrate

![Graph showing the relationship between SSIM (dB) and average bitrate (Mbit/s) for different encoding methods: vpx (single-threaded), vpx (multithreaded), vpx (6-frame chunks), and ExCamera. The graph demonstrates how SSIM changes with varying bitrates for each method.](attachment:image.png)
14.8-minute 4K Video

Encoding with vpxenc (VP8)

Single-Threaded ~7.5 hours

Multi-Threaded ~2.5 hours

ExCamera 2.1 minutes
Outline

• Vision & Goals
• Background on Video Processing
• Fine-grained Parallel Video Encoding
• μ: Supercomputing as a Service
• Evaluation
• Conclusion & Future Work
Vision: Real-time Collaborative Video Processing

• Editing and sharing: Google Docs for Video.
  • "Make this movie black and white."
  • "Apply some awesome filter to my video."
  • "Pixelate this face everywhere in this video."
  • "Remake Star Wars Episode I without Jar Jar."
Takeaways

• Low-latency video processing

• Two major contributions:
  • A video encoder intended for massive fine-grained parallelism.
  • A framework that orchestrates thousands of threads running in parallel on AWS Lambda.

• 64× faster than existing encoder, for less than $10.

• The future is granular & massively parallel
  • Parallel make
  • "Laptop Extension"

excamera@cs.stanford.edu
How time breaks down

threads
1
4
7
10
13
16

start
25 s
50 s
75 s
100 s

download vpxenc decode wait encode-given-state wait rebase upload

25 s
50 s
75 s
100 s