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

Presenter: Wen-Fu Lee
Outline

• Vision & Goals
• mu: Supercomputing as a Service
• Fine-grained Parallel Video Encoding
• Evaluation
• Takeaways
Outline

• Vision & Goals
  • mu: Supercomputing as a Service
  • Fine-grained Parallel Video Encoding
• Evaluation
• Takeaways
What we currently have

- People can make changes to a word-processing document
- The changes are instantly visible to the others
What we would like to have

- People can interactively edit and transform a video
- The changes are instantly visible to the others
"Look everywhere for this face in this movie."
"Remake Star Wars Episode I without Jar Jar."
The Problem
Currently, running such pipelines on videos takes hours and hours, even for a short video.

The Question
Can we achieve interactive collaborative video editing by using massive parallelism?
The challenges

• Low-latency video processing would need **thousands of threads**, running in parallel, with **instant startup**.

• However, **the finer-grained the parallelism, the worse the video compression efficiency**.
ExCamera

• Two contributions
  • Framework to run 5,000-way parallel jobs with IPC* on a commercial “cloud function” service.
  
  • Purely functional video codec for massive fine-grained parallelism.

*Inter-process communication (IPC)
Outline

• Vision & Goals
• mu: Supercomputing as a Service
• Fine-grained Parallel Video Encoding
• Evaluation
• Takeaways
## Where to find thousands of threads?

<table>
<thead>
<tr>
<th>Virtual machine</th>
<th>Amazon: EC2</th>
<th>Microsoft: Azure</th>
<th>Google: GCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Service Providers</td>
<td>Think about it as</td>
<td>Base layer</td>
<td>Unit = VM</td>
</tr>
<tr>
<td>Pros &amp; cons</td>
<td>[+] Thousands of threads</td>
<td>[+] Arbitrary Linux executables</td>
<td>[-] Minute-scale startup time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- OS has to boot up, ...</td>
</tr>
<tr>
<td></td>
<td>[+] High minimum cost</td>
<td>[-]</td>
<td>- 60 mins EC2, 10 mins GCE</td>
</tr>
</tbody>
</table>

| Running 3,600 threads for 1 sec | > $20 |
## Where to find thousands of threads?

<table>
<thead>
<tr>
<th>Cloud Service Providers</th>
<th>Virtual machine</th>
<th>Cloud function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon: EC2</td>
<td>Thousand of threads</td>
<td>AWS Lambda</td>
</tr>
<tr>
<td>Microsoft: Azure</td>
<td>Arbitrary Linux executables</td>
<td>Google Cloud Functions</td>
</tr>
<tr>
<td>Google: GCE</td>
<td>Minute-scale startup time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- OS has to boot up, ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>High minimum cost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- 60 mins EC2, 10 mins GCE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pros &amp; cons</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[+] Thousands of threads</td>
<td>[+] Thousands of threads</td>
</tr>
<tr>
<td></td>
<td>[+] Arbitrary Linux executables</td>
<td>[+] Arbitrary Linux executables</td>
</tr>
<tr>
<td></td>
<td>[+] Minute-scale startup time</td>
<td>[+] Sub-second startup</td>
</tr>
<tr>
<td></td>
<td>- OS has to boot up, ...</td>
<td>[+] Sub-second billing</td>
</tr>
<tr>
<td></td>
<td>[+] High minimum cost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- 60 mins EC2, 10 mins GCE</td>
<td></td>
</tr>
</tbody>
</table>

| Running 3,600 threads for 1 sec | > $20                                   | 10 cents                                     |
mu, supercomputing as a service

- **mu**, a library for designing and deploying general-purpose parallel computations on **AWS Lambda**.

- The system starts up thousands of threads in seconds and manages inter-thread communication.
**mu** software framework

- Coordinator
  - Long-lived server
  - Dependency-aware scheduling

- Rendezvous
  - Long-lived server
  - Inter-thread communication

- Workers
  - Short-lived Lambda function invocation
Outline

• Vision & Goals
• mu: Supercomputing as a Service
• Fine-grained Parallel Video Encoding
• Evaluation
• Takeaways
Now we have the threads, but...

• With the existing encoders, the finer-grained the parallelism, the worse the compression efficiency.
Video Codec

- A piece of software or hardware that compresses and decompresses a digital video.
Encoder

```
encode([[image_1], [image_2], ..., [image_n]]) → keyframe + interframe[2:n]
```

In a 4K video @15Mbps, a key frame is \(~1\) MB, but an interframe is \(~25\) KB.
decode(keyframe + interframe[2:n]) → [:image:image:image:]
Traditional parallel video encoding is limited

\[ \text{encode}(i[1:200]) \rightarrow \text{keyframe}_1 + \text{interframe}[2:200] \]
Traditional parallel video encoding is limited

\[
\text{encode}(i[1:200]) \rightarrow \text{keyframe}_1 + \text{interframe}[2:200]
\]

Serial ↓

Parallel ↓

[thread 01] \text{encode}(i[1:10]) \rightarrow \text{kf}_1 + \text{if}[2:10]
[thread 02] \text{encode}(i[11:20]) \rightarrow \text{kf}_1 + \text{if}[12:20] + 1MB
[thread 03] \text{encode}(i[21:30]) \rightarrow \text{kf}_2 + \text{if}[22:30] + 1MB

\vdots

[thread 20] \text{encode}(i[191:200]) \rightarrow \text{kf}_{191} + \text{if}[192:200] + 1MB
Traditional parallel video encoding is limited

\[
\text{encode}(i[1:200]) \rightarrow \text{keyframe}_1 + \text{interframe}[2:200]
\]

parallel ↓

\[
\begin{align*}
\text{[thread 01]} & \quad \text{encode}(i[1:10]) \rightarrow \text{kf}_1 + \text{if}[2:10] \\
\text{[thread 02]} & \quad \text{encode}(i[11:20]) \rightarrow \text{kf}_1 + \text{if}[12:20] \\
\text{[thread 03]} & \quad \text{encode}(i[21:30]) \rightarrow \text{kf}_2 + \text{if}[22:30] \\
& \vdots \\
\text{[thread 20]} & \quad \text{encode}(i[191:200]) \rightarrow \text{kf}_{191} + \text{if}[192:200]
\end{align*}
\]

finer-grained parallelism \Rightarrow more key frames \Rightarrow worse compression efficiency
What we built: a video codec in an 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
  • $\text{encode}(\text{state}, \text{image}) \rightarrow \text{interframe}$

• Adapt a frame to a different source state
  • $\text{rebase}(\text{state}, \text{image}, \text{interframe}) \rightarrow \text{interframe}'$
ExCamera
Encoder’s Algorithm
1. [Parallel]
Download a tiny chunk of raw video
2. [Parallel] Google’s VP8 encoder

`encode(img[1:n]) → keyframe + interframe[2:n]`
3. **Parallel**

decode(state, frame)

Our explicit-state style decoder

\[
\text{decode}(\text{state}, \text{frame}) \rightarrow (\text{state}', \text{image})
\]
4. **Parallel**

\[
\text{encode(state, image)}
\]
5. [Serial] 
rebase(state, image, interframe)
5. **Serial**

`rebase(state, image, interframe)`

Adapt a frame to a different source state

`rebase(state, image, interframe) → interframe'`
5. **Serial**

rebase(state, image, interframe)

Adapt a frame to a different source state:

\[ \text{rebase(state, image, interframe)} \rightarrow \text{interframe'} \]
6. [Parallel]
Upload finished video
Time Distribution

[Diagram showing time distribution with slow and fast parts, threads, and various stages like download, vpxenc, decode, wait, encode-given-state, wait, rebase, upload]
Wide range of different configurations

ExCamera\[n, x\]

tnumber of frames in each chunk
Wide range of different configurations

ExCamera $[n, x]$  

number of chunks "rebased" together
Outline

• Vision & Goals
• mu: Supercomputing as a Service
• Fine-grained Parallel Video Encoding
• Evaluation
• Takeaways
How well does it compress?
How well does it compress?

14.8-minute 4K Video @20dB

- vpxenc Single-Threaded: 453 mins
- vpxenc Multi-Threaded: 149 mins
- ExCamera[6, 16]: 2.6 mins

Encoding Speed
Outline

• Vision & Goals
• mu: Supercomputing as a Service
• Fine-grained Parallel Video Encoding
• Evaluation
• Takeaways
# ExCamera vs. PyWren

<table>
<thead>
<tr>
<th></th>
<th>PyWren</th>
<th>ExCamera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>Using AWS Lambda</td>
<td>Inter-thread communication</td>
</tr>
<tr>
<td>Different</td>
<td>No</td>
<td>Coordinator &amp; rendezvous</td>
</tr>
<tr>
<td></td>
<td>Serverless</td>
<td></td>
</tr>
</tbody>
</table>
Takeaways

• Target: Low-latency video processing

• Two major contributions
  • Framework to run 5,000-way parallel jobs with IPC on AWS Lambda.
  • Purely functional video codec for massive fine-grained parallelism.

• 56× faster than existing encoder, for <$6.
  • Lots of speedup from fine-grained parallelism -> need to restructure the application to get maximum benefits out of it.
Reference

• http://pages.cs.wisc.edu/~shivaram/cs744-readings/excamera.pdf
• https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi
• https://doublehorn.com/comparing-the-big-3-aws/
• https://en.wikipedia.org/wiki/VP8
Thanks for your attention.
Q&A
Backup
Functions

- \text{state} := (\text{prob} \_\text{model}, \text{references}[3])
- \text{decode}(\text{state}, \text{compressed} \_\text{frame}) \rightarrow (\text{state}', \text{image})
  - \text{decode}(\text{any state}, \text{key} \_\text{frame}) \rightarrow (\text{state}', \text{image})
- \text{interframe} :=
  - (\text{prediction} \_\text{modes}, \text{motion} \_\text{vectors}, \text{residue})
- \text{encode-given-state}(\text{state}, \text{image}, \text{quality}) \rightarrow \text{interframe}
- \text{rebase}(\text{state}, \text{image}, \text{interframe}) \rightarrow \text{interframe}'
Cold start vs. Warm start
Demo: Massively parallel face recognition on AWS Lambda

• \(~6\text{ hours}\) of video taken on the first day of NSDI.
  • 1.4TB of uncompressed video uploaded to S3.

• Adapted \textit{OpenFace} to run on AWS Lambda.
  • OpenFace: face recognition with deep neural networks.

• Running 2,000 \textit{Lambdas}, looking for a face in the video.
The future is granular, interactive and massively parallel

- Parallel/distributed make
- Interactive Machine Learning
  - e.g. PyWren (Jonas et al.)
- Data Visualization
- Searching Large Datasets
- Optimization