Emerging space of video-driven applications
CMU Panoptic studio
Capture of 480 640x480 video streams
147 MPixel capture at 24 fps (3.5 GPixel/sec)

[Image Credit: Joo et al. 2015]
3D human pose reconstruction

40-second sequence (captured human social interactions)

Reconstruction time:
hand-coded solution by MS grad student — 7 hours on a 4-GPU machine [Cao 2016]
Example: human pose detection

- Attempt #1: do it in Python
- Attempt #2: import multiprocessing
- Attempt #3: rewrite it in C++
- Attempt #4: use multiple GPUs (6 month checkpoint)
- Attempt #5: scale across a cluster
Video application challenges

- Video is huge!
  - Requires compression, decode (efficient pixel delivery)
  - Varying access patterns
  - 3-25x blowup in size for explode to frames
  - 1000x vs. raw pixels
- Execution requires heterogeneous hardware
- Stateful/stenciled computation patterns
No existing system well suited for the job

- Distributed data-analytics frameworks [MapReduce, Spark]
- Distributed machine learning frameworks [TensorFlow, MxNet]
- Array/raster databases [SciDB, RasDaMan, GIS databases]
Goal: design a system that makes it easy to rapidly author video analysis pipelines that scale.
Scanner: design goals / principles

**Design principle 1: keep it simple**
- Enable non-expert programmers
- Focus on video applications

**Design principle 2: be efficient**
- “Near-HW-peak single-node perf” then scale out
- Utilize heterogenous hardware

**Non goal:**
- Do not be a new kernel description language
- Do not be a general purpose dataflow/streaming engine
Setup

I have a list of videos in a directory...

- myvideos/cam000.mp4
- myvideos/cam001.mp4
- myvideos/cam002.mp4
- ...
- myvideos/cam479.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

- Human Pose estimation
- Object detector
- Face detection network
- Depth/normal estimator

NVIDIA cuDNN

[Cao16] [Redmon16] [Bansal17] [Hu17]
Represent videos as relations (tables)

myvideos/cam000.mp4
myvideos/cam001.mp4
myvideos/cam002.mp4
...
myvideos/cam479.mp4

Ingest into Scanner...

Scanner dataset: capture_session
Computations: DAGs of image processing operations

Pipeline kernels map to heterogeneous resources: CPUs, GPUs, ASICs
Scanner maps pipelines onto a stream of video frames from tables

Pipeline:
Input: cam_000  
Output: pose_000

Input:
<table>
<thead>
<tr>
<th>id</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: pose_000

<table>
<thead>
<tr>
<th>id</th>
<th>2dpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sampling/joining input frames

Stride, gather, range, join, ...

Stride by 2:

Join two tables:
Distributing tasks across a cluster

Cluster

<table>
<thead>
<tr>
<th>id</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node 0: multi-core CPU + GPU

- I/O
- GPU HW Decoder
- Resize Instance 0
- DNN Eval Instance 0
- Estimate Pose Instance 0

Node 1: multi-core CPU + 2 GPUs

- I/O
- GPU HW Decoder
- Resize Instance 1
- DNN Eval Instance 1
- Estimate Pose Instance 1

- I/O
- GPU HW Decoder
- Resize Instance 2
- DNN Eval Instance 2
- Estimate Pose Instance 2

Pipeline:
Input: cam_000 [0, 1, 2, 3, ...]
Output: pose_000

012
3
Demo
Video challenges
Storing video collections efficiently

Logical relation

<table>
<thead>
<tr>
<th>frame_id</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stored Representation

H.264 encoded bytestream

<table>
<thead>
<tr>
<th>iframe #</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>864b</td>
</tr>
<tr>
<td>54</td>
<td>5478b</td>
</tr>
<tr>
<td>240</td>
<td>8194b</td>
</tr>
</tbody>
</table>
Background: h264 Encoded Video

Stored Representation

cam_000:frames  H.264 encoded bytestream
Background: h264 Encoded Video

Stored Representation

cam_000:frames

<table>
<thead>
<tr>
<th></th>
<th>H.264 encoded bytestream</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOP</td>
<td>GOP</td>
</tr>
<tr>
<td>G..</td>
<td>GOP</td>
</tr>
<tr>
<td>GOP</td>
<td>GOP</td>
</tr>
</tbody>
</table>

GOP
Background: h264 Encoded Video

Stored Representation

cam_000:frames

H.264 encoded bytestream

GOP  GOP  G..  GOP  GOP  GOP

GOP

I  P  P  B  P
Background: h264 Encoded Video

Stored Representation

H.264 encoded bytestream

GOP | GOP | G.. | GOP | GOP | GOP

I   P   P   B   P

cam_000::frames

iframe # | offset
0        | 864b
54       | 5478b
240      | 8194b
....     |...

...
Maintain keyframe index to assist parallel decode
Kernels need more than one frame

- Goal #1: Temporal extent
- Goal #2: Sampling
- Goal #3: Batching
- Goal #4: Modularity
Two-level hierarchy on stream elements

Pipeline batch size: 4
Consecutive elements: 1024
Two-level hierarchy on stream elements

Pipeline batch size: 4
Consecutive elements: 1025
Two-level hierarchy on stream elements

Stride by 2:

<table>
<thead>
<tr>
<th>id</th>
<th>frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Warmup size = 1

Pipeline batch size: 4
Consecutive elements: 1024
Stenciling patterns

Stencil

Stride & stencil

Stencil & stride
Stenciling patterns
Solution: change kernel interface
Evaluation
Scanner has low overhead

Single Node:
Throughput Relative to Hand-Tuned Pipelines

Throughput

Scanner
Baseline

CPU Histogram
GPU Optical Flow
CPU DNN Eval
GPU Gather
Scaling to dense GPU nodes

Throughput (relative to expert-tuned)

Scanner (4-GPU)
Multi-node scaling

- **Histogram**
  - Throughput (fps) vs. Num nodes
  - Green: Scanner (CPU only)
  - Yellow: Scanner (GPU only)

- **Optical Flow**
  - Throughput (fps) vs. Num nodes
  - Green: Scanner (CPU only)
  - Yellow: Scanner (GPU only)

- **DNN Eval**
  - Throughput (fps) vs. Num nodes
  - Green: Scanner (CPU only)
  - Yellow: Scanner (GPU only)
3D pose reconstruction

Processing 40 seconds of video from CMU Panoptic studio

Grad student hand-tuned: 7 hrs (4 GPU node)
Scanner: 2.6 hrs (4 GPU node)
Scanner on cluster: 38 mins (4 node x 4 GPU/node cluster)

Approaching viability for marker less motion capture.
Shot segmentation (cinematography analysis)

608 feature length films (2.4 TB)
103M frames
Histogram-based shot segmentation of all films: 4.7 hrs (4 node cluster, 4 GPUs/node)
What’s next?

- Scale out with Google Cloud
- 3 PB Internet Archive dataset
- Youtube 8M face database
- Esper frontend for Magic Grant
Thank you!

https://github.com/scanner-research/scanner