Intel-Unnati AI&ML Journal
Number 3 (2025), pp. 1–5 Code:IUIT-2025

Machine Learning

Performance Benchmarking of DL Streamer Pipelines on Intel Iris Xe Graphics and CPU

Rayid.M.Afzal, Ebin Soyan, Aswin.S

Saintgits College of Engineering, Kottayam, Kerala

Abstract: This paper benchmarks and analyzes the performance of Intel® DL Streamer pipelines on a resource-constrained system powered by an Intel i5-1235U CPU and integrated Iris Xe GPU. Using standard object detection and classification models on GStreamer-based pipelines, we evaluate single and multi-stream performance, identify system bottlenecks, and document observed crashes and their causes. The aim is to assess DL Streamer's scalability on non-dedicated hardware and inform optimization strategies for lightweight deployments.

Keywords: DL Streamer, Intel Iris Xe, CPU benchmarking, media pipeline, GStreamer, AI pipeline performance, resource constraints, crash analysis, pipeline optimization, hardware acceleration

1 Introduction

DL Streamer is Intel's open-source pipeline framework built on GStreamer, allowing accelerated media analytics. This work attempts to deploy and stress-test a DL Streamer pipeline involving video decoding, object detection, and classification stages.

Our focus is on:

- Establishing maximum achievable throughput per stream on CPU vs GPU.
- Measuring how the system responds as stream count increases.
- Identifying crash points and bottlenecks.

This project was executed by Rayid M. Afzal, Ebin Soyan, and Aswin S., under the mentorship of Mr. Arun Sebastian, as part of the Intel-Unnati AI&ML initiative at Saintgits College of Engineering.

2 System Setup

- **CPU:** Intel i5-1235U @ 1.30GHz
- **GPU:** Intel® Iris® Xe Graphics (integrated)
- **RAM:** 8 GB
- **Operating System:** Ubuntu 24.04 LTS
- **Toolkits:** Installed via APT (`dlstreamer-gst` subset)

3 Pipeline Configuration

The models used in the pipeline were:

- **Object Detection:** person-detection-retail-0013 (FP16)
- **Classification:** person-attributes-recognition-crossroad-0230 (FP16)

The following GStreamer-based DL Streamer pipeline was used:

```
gst-launch-1.0 filesrc location=video.mp4 ! decodebin ! \
  gvadetect model=... model-proc=... device=CPU ! queue ! \
  gvaclassify model=... model-proc=... device=CPU object-class=person
      reclassify-interval=10 ! queue ! \
  gvawatermark ! videoconvert ! fpsdisplaysink text-overlay=true sync=
      false signal-fps-measurements=true
```

Listing : Single-stream Benchmark Pipeline

Scripted Multi-stream Launcher (CPU)

To benchmark performance under stress:

```
#!/bin/bash

NUM_STREAMS=8  # Adjust this for how many parallel streams you want
VIDEO_ORIGINAL="$HOME/Downloads/testinputnew.mp4"
VIDEO_DIR="$HOME/video_copies_cpu"
LOG_DIR="$HOME/scriptfinal/log"

# Model Paths
DETECTION_MODEL="$HOME/intel/person-detection-retail-0013/FP16/person-
    detection-retail-0013.xml"
CLASSIFICATION_MODEL="$HOME/intel/person-attributes-recognition-
    crossroad-0230/FP16/person-attributes-recognition-crossroad-0230.
    xml"
DETECTION_PROC="/opt/intel/dlstreamer/samples/gstreamer/model_proc/
    intel/person-detection-retail-0013.json"
CLASSIFICATION_PROC="/opt/intel/dlstreamer/samples/gstreamer/model_proc
    /intel/person-attributes-recognition-crossroad-0230.json"
```

```
13
14 # Create folders
15 mkdir -p "$VIDEO_DIR"
16 mkdir -p "$LOG_DIR"
17
18 # Clone videos
19 for ((i=1; i<=NUM_STREAMS; i++)); do
20   cp "$VIDEO_ORIGINAL" "$VIDEO_DIR/video_$i.mp4"
21 done
22
23 # Launch each stream in its own terminal
24 for ((i=1; i<=NUM_STREAMS; i++)); do
25   gnome-terminal --title="CPU␣Stream␣$i" -- bash -c "
26 ␣␣␣␣echo␣'>>>␣Running␣Stream␣$i'␣|␣tee␣\"$LOG_DIR/cpu_stream_$i.log\"
27 ␣␣␣␣gst-launch-1.0␣filesrc␣location=$VIDEO_DIR/video_$i.mp4␣!␣decodebin
       ␣!␣\
28 ␣␣␣␣␣␣gvadetect␣model=$DETECTION_MODEL␣model-proc=$DETECTION_PROC␣
       device=CPU␣!␣queue␣!␣\
29 ␣␣␣␣␣␣gvaclassify␣model=$CLASSIFICATION_MODEL␣model-proc=
       $CLASSIFICATION_PROC␣device=CPU␣object-class=person␣reclassify-
       interval=10␣!␣queue␣!␣\
30 ␣␣␣␣␣␣gvawatermark␣!␣videoconvert␣!␣fpsdisplaysink␣text-overlay=true␣
       sync=false␣video-sink=xvimagesink␣signal-fps-measurements=true␣\
31 ␣␣␣␣␣␣2>&1␣|␣tee␣-a␣\"$LOG_DIR/cpu_stream_$i.log\"
32 ␣␣␣␣notify-send␣'Stream␣$i␣Completed'
33 ␣␣␣␣exit"
34 done
```

Listing : Full Multi-stream CPU Launcher Script

A similar version was created for GPU tests with `device=GPU`. Video clones were stored in separate folders for CPU and GPU tests.

4 Performance Observations

4.1 GPU Performance (Iris Xe)

- **Single Stream:** FPS ˜0.2
- **System Lag:** Severe freezing even for one stream.
- **2 Streams:** System crashed and became unresponsive.
- **Conclusion:** GPU offload not effective; fallback to CPU likely.

4.2 CPU Performance

- **Streams Tested:** 1 to 8
- **FPS Values:** Decreasing with stream count:
  - 1 Stream: 22.42 FPS
  - 2 Streams: 10.26 FPS

  - 3 Streams: 6.35 FPS
  - 4 Streams: 4.34 FPS
  - 5 Streams: 3.27 FPS
  - 6 Streams: 2.73 FPS
  - 7 Streams: 1.92 FPS
  - 8 Streams: 1.08 FPS
- **Swap Usage:** Swap increased to 8GB to prevent crashes.
- **Crash Point:** 9 streams triggered OOM killer.

5 Crashes and Bottlenecks

- **GPU bottleneck:** Iris Xe failed to accelerate inference. FPS remained below 1, suggesting fallback to CPU. Despite explicitly setting `device=GPU`, inference was likely not offloaded to the GPU due to VAAPI misconfiguration, model format issues, or driver-level limitations. Monitoring with `intel_gpu_top` showed minimal GPU utilization, reinforcing that inference load remained on the CPU.
- **CPU bottleneck:** RAM saturated at 8GB. Swap was heavily used beyond 6 streams, degrading performance.
- **Swap Memory Pressure:** The system's 8GB RAM was fully consumed beyond 6 streams, causing heavy reliance on swap memory. Swap usage reached 80% at 7 streams, and was expanded to 8GB to stabilize 8-stream performance. Even then, average FPS dropped below usable levels, confirming RAM capacity as a critical bottleneck in multi-stream scenarios.
- **OOM Killer Crash:** At 9 streams, swap memory was fully utilized, and the system crashed without properly freeing the pipeline. The absence of the usual "freeing pipeline" log message suggested the Linux OOM killer terminated the process. This abrupt crash highlights a scalability ceiling for the pipeline under current system constraints.

6 Benchmark Results

Device	Streams	Avg FPS	Notes
CPU	1	22.42	Stable
CPU	2	10.26	Minor degradation
CPU	3	6.35	Acceptable
CPU	4	4.34	Manageable
CPU	5	3.27	Lag noticeable
CPU	6	2.73	Degraded experience
CPU	7	1.92	Severe lag
CPU	8	1.08	Swap maxed
CPU	9	–	Crash (OOM)
GPU	1	0.2	Freezing, ineffective
GPU	2	–	Crash

Table 1: DL Streamer Performance Benchmarks

7 Conclusion

DL Streamer pipelines push integrated GPUs like the Iris Xe to their limits. Despite Intel's optimized models and GStreamer enhancements, performance on low-end hardware was bottlenecked heavily. CPU-only execution showed better scaling but also capped out due to thermal and RAM constraints.

Recommendations:

- Use quantized or smaller models for embedded setups.
- Optimize inference intervals to balance latency.
- Consider dedicated GPUs or VPUs for real-time multi-stream analytics.

Acknowledgments

Thanks to Intel® Unnati for toolkits and guidance. Special thanks to our project mentor Arun Sebastian sir and also grateful to Saintgits College of Engineering for support during the project.