Optimising OpenVX applications for implementation on GPUs

Chris Longstaff
Senior Director of Product & Technology Marketing, PowerVR
3rd May, 2017
Introduction

About Imagination, PowerVR & this presentation

- About Imagination
  - Imagination is a leading supplier of SoC IP
    - CPU - MIPS
    - Communications - ENSIGMA
    - Multimedia – PowerVR

- About PowerVR
  - PowerVR IP cores have shipped in over 3 billion products
  - Key markets are mobile, automotive, consumer and IoT (including security)
  - PowerVR GPUs support OpenGL ES, Vulkan, OpenCL OpenVX

- About this Presentation
  - The need for optimising OpenVX
  - Measuring OpenVX performance
  - Typical use case
  - Improvements
  - Summary
The need to optimise OpenVX applications

But hold on … shouldn’t that be automatic with OpenVX?

- OpenVX gives developers portability across platforms
- The implementation of OpenVX does allow for automatic optimisation of graphs – but this depends on the driver
- OpenVX targets many different forms of hardware (CPU, GPU, DSP, Dedicated H/W) and each may have primitives that are more or less efficient on that hardware
- Understanding the limitations of the specific platform, enables the application designer to better optimise for the target hardware, but most rules apply across platforms
Measuring performance

In order to optimise for a platform, we need to understand the “bottlenecks”

- Optimising an application for OpenVX is a combination of efficient driver & understanding the platform & API
- PowerVR supplies tools for assessing the workload on the GPU, these help us understand how we can ensure full SoC utilisation
- By analysing these workloads, improvements can be made to the application
Realworld use case

Sample application “lane departure”

- The sample used here is a simplified lane departure warning
- We will show the effects that the driver optimisation can have on the performance of the application
- We will show how the PowerVR tools can assist in the performance profiling and improvement of applications
OpenVX Graph for simple lane departure

(Simplified - Showing only the main blocks)

- Input Image
  - Extract Channel (R) (OpenVX)
  - Extract Channel (G) (OpenVX)
  - Extract Channel (B) (OpenVX)
  - Add
  - Subtract
  - Sobel (OpenVX)
  - Phase (OpenVX)
  - Canny Edge (OpenVX)
  - Hough Lines (User Node)
Measurement results

Driver optimisation turned off
Measurement results

Driver optimisation turned off

GPU frequently idle

All Operations sequential
OpenVX Graph for simple lane departure

With OpenVX Graph optimisations … specific to target hardware

Input Image

Convert

Sobel+Phase

Hough Lines (User Node)

Canny Edge (OpenVX)

Extract Channel (R) (OpenVX)

Sobel (OpenVX)

Phase (OpenVX)

Add

Extract Channel (G) (OpenVX)

Extract Channel (B) (OpenVX)

Canny Edge (OpenVX)

Subtract

Hough Lines (User Node)
GPU utilisation after driver optimisation

OpenVX graph optimised
GPU utilisation after driver optimisation

OpenVX graph optimised

GPU improved utilisation

Task completion in ~50% less time

Operations combined
OpenVX is a standards framework, which allows for platform portability.

PowerVR GPU driver will automatically optimise the OpenVX graph implementation to maximise GPU efficiency.

PowerVR provide comprehensive tools to ensure that you can both measure & optimise your implementation for GPU hardware.

Summary

PowerVR offers optimal OpenVX implementation for GPU
Thank you for your attention