APIs for Accelerating Vision and Inferencing: Options and Trade-offs

Neil Trevett, Khronos President
May 2018
The Search for Vision Performance and Portability leads to a Layered Acceleration Ecosystem.

- Libraries, Frameworks and run-times
  - AR Libraries
  - OpenXR
  - ARCore

- Single Source C++ Languages
  - CUDA
  - SYCL

- Explicit Kernel APIs
  - Vulkan
  - OpenCL

- Code Intermediate Representations
  - PTX
  - HSAIL

- Diverse Hardware (GPUs, DSPs, FPGAs)
  - GPU
  - FPGA
  - DSP
  - Dedicated Hardware
## Mobile Augmented Reality Libraries

<table>
<thead>
<tr>
<th>Feature</th>
<th>iOS</th>
<th>Android and iOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encapsulated Vision-based Functionality</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Also leveraging motion sensors</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Pose Tracking</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Plane Detection and Tracking</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Image Recognition and Tracking</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ambient light level and temperature</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Link to Neural Net-based Object Detection</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Access to Point Cloud</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Face tracking (iPhone X)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Light probe (using a tracked face)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multi-user cloud-based anchors</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>OS Availability</td>
<td>iOS</td>
<td>Android and iOS</td>
</tr>
</tbody>
</table>

AR Libraries typically use ARKit/ARCore if available or implement own tracking if not.

V1.0 is VR focused. Subsequent versions will extend to cross-platform AR.
Neural Network Workflow

Neural Net Training Frameworks

Training = Desktop / Cloud

Desktop and Cloud Hardware

Applications Using Embedded Vision and Inferencing

WinML

Framework Specific Formats

Compilation and Optimization

Vision and Neural Net Inferencing Runtimes

Diverse Inferencing Acceleration Hardware

Training = Desktop / Cloud

Deployment on Embedded Devices

Desktop and Cloud Hardware

cuDNN

MIOpen

MKL-DNN

TPU
NNVM - Open Compiler for AI Inferencing

1. Import Trained Network Description
2. Graph-level Optimizations
3. Decompose into primitive instructions
4. Emit programs executable by run-times

SPIR-V IR for parallel accelerators Backend in development
LLVM IR for CPUs

Paul G. Allen School of Computer Science & Engineering, University of Washington
SPIR-V Ecosystem

- Third party kernel and shader languages
  - glslang
  - DXC

- Third party kernel and shader languages
  - GLSL
  - HLSL

- SPIR-V (Dis)Assembler
- SPIRV-Cross
- SPIRV-Validator
- SPIRV-opt | SPIRV-remap

- Optimization Tools
- LLVM
- LLVM to SPIR-V Bi-directional Translators

- IHV Driver Runtimes
- Vulkan
- OpenGL

- OpenCL C
  - Front-end

- OpenCL C++
  - Front-end

- SYCL
  - Front-end

- SPIR-V Validator
  - SPIRV- (Dis)Assembler

- Khronos defined cross-API IR
  - Native graphics and parallel compute
  - Stable specification to complement LLVM
  - Open Source Project

- Khronos cooperating with Clang/LLVM Community

- Khronos-hosted Open Source Projects
  - https://github.com/KhronosGroup/SPIRV-Tools
Microsoft Windows
Windows Machine Learning (WinML)

Google Android
Neural Network API (NNAPI)
https://developer.android.com/ndk/guides/neuralnetworks/

Apple MacOS and iOS
CoreML
https://developer.apple.com/documentation/coreml
NNEF Ecosystem

NNEF open source projects hosted on Khronos NNEF GitHub repository
Apache 2.0 license
https://github.com/KhronosGroup/NNEF-Tools

NNEF 1.0 Provisional Released for industry feedback before finalization

Comparing Neural Network Exchange Industry Initiatives

<table>
<thead>
<tr>
<th>NNEF</th>
<th>ONNX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined Specification</td>
<td>Open Source Project</td>
</tr>
<tr>
<td>Stability for hardware deployment</td>
<td>Software stack flexibility</td>
</tr>
<tr>
<td>Multi-company Governance</td>
<td>Initiated by Facebook</td>
</tr>
<tr>
<td>Flat and Compound Ops</td>
<td>Flat Ops Only</td>
</tr>
</tbody>
</table>
Network Structure File
Distilled, platform independent network description
Human readable, syntactical elements from Python

Standardized Operations
Rigorously defined semantics
Linear, convolution, pooling, normalization, activation, unary/binary
Supports fully connected, convolutional, recurrent architectures

Two Levels of Expressiveness
FLAT: Basic transfer of computation graphs with standardized operations
Simple to parse and translate to vendor specific formats
COMPOSITIONAL: Define custom compound operations
Higher-level graph descriptions
More complex to parse but offers more optimization hints

Network Data File
Binary format contains parameter tensors
Supports float and quantized (integer) data
Flexible bit widths and quantization algorithms
Quantization algorithms expressed as extensible compound operations
Quantization info provided as hints for execution

Can associate multiple Data Files with one Network Structure File e.g. the same data in multiple formats

Split Structure and Data files
Easy independent access to network structure or individual parameter data
Set of files can use a container such as tar or zip with optional compression and encryption
Convolution Neural Network topologies can be represented as OpenVX graphs
  - Can also combine traditional vision and neural network operations

OpenVX Neural Network Extension
  - Defines OpenVX nodes to represent many common NN layer types
  - Layer types include convolution, activation, pooling, fully-connected, soft-max
  - Defines multi-dimensional tensors objects to connect layers

Kernel Import Extension
  - Enables loading of external program representations into OpenVX graphs
OpenVX / OpenCL Interop Extension

OpenVX user-kernels can access command queue and cl_mem objects to asynchronously schedule OpenCL kernel execution.

- **OpenVX data objects**
- **cl_mem buffers**

**Application**

- Copy or export cl_mem buffers into OpenVX data objects
- Map or copy OpenVX data objects into cl_mem buffers

- Fully asynchronous host-device operations during data exchange

**Enables custom OpenCL acceleration to be used within OpenVX User Kernels**
OpenCL Ecosystem Roadmap

**2011**
- OpenCL 1.2
- OpenCL C Kernel Language

**2015**
- SYCL 1.2
  - C++11 Single source programming

**2017**
- SYCL 1.2.1
  - C++11 Single source programming

*Work with industry to bring Heterogeneous compute to standard ISO C++*

**OpenCL ‘Next’**
Flexible and efficient deployment of parallel computation across diverse processor architectures

*More deployment options: Enabling dispatch of OpenCL C kernels from Vulkan runtimes*
SYCL Ecosystem

- Single-source heterogeneous programming using STANDARD C++
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL
- Fast and powerful path for bring C++ apps and libraries to OpenCL
  - C++ Kernel Fusion - better performance on complex software than hand-coding
  - SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL DNN, SYCL GTX, VisionCpp
- Close cooperation with ISO C++
  - C++17 Parallel STL hosted by Khronos
  - C++20 Parallel STL with Ranges
- Implementations
  - triSYCL, ComputeCpp, ComputeCpp SDK ...
- More information at http://sycl.tech
Pervasive Vulkan 1.0

Major GPU Companies supporting Vulkan for Desktop and Mobile Platforms

Platforms

- Desktop
- Mobile (Android 7.0+)
- Media Players
- Consoles
- Virtual Reality
- Cloud Services
- Embedded

http://vulkan.gpuinfo.org/
Clspv OpenCL C to Vulkan Compiler

- Experimental collaboration between Google, Codeplay, and Adobe
  - Successfully tested on over 200K lines of Adobe OpenCL C production code
- Compiles OpenCL C to Vulkan’s SPIR-V execution environment
  - Proof-of-concept that OpenCL kernels can be brought seamlessly to Vulkan
  - Significant parts OpenCL C 1.2 so far - shaped by submitted workloads

Prototype open source project
https://github.com/google/clspv

Increasing deployment options for OpenCL kernel developers e.g.
Vulkan is a supported API on Android

Possible future project - if interest?
OpenCL Next - Deployment Flexibility

- Vendors can support ANY combination of features to suit their hardware/market
  - If all exposed features are conformant - the implementation is conformant
- Khronos will define Feature Sets equivalent to current profiles
  - Existing profiles and device types not going away! No changes to existing applications
- Opportunity to coalesce industry support around market-focused feature sets
  - Khronos aiming to provide Feature Set infrastructure for the industry to leverage

OpenCL 2.2 Functionality

- Khronos-defined
  - OpenCL 2.2 Full Profile Feature Set
  - OpenCL 1.2 Full Profile Feature Set
- Industry-defined
  - E.g. ‘Inferencing’ Feature Set

= queryable, optional feature
Safety Critical APIs - Khronos Experience

Need for new-generation APIs for safety certifiable vision, graphics and compute e.g. ISO 26262 and DO-178B/C

General lack of industry consensus on how APIs should be designed to streamline safety certification

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL SC 1.0 - 2005
Fixed function graphics subset

OpenGL SC 2.0 - April 2016
Shader programmable pipeline subset

OpenGL ES 2.0 - 2007
Shader programmable pipeline

OpenVX SC 1.1 - May 2017
Restricted "deployment" implementation only executes pre-compiled binary format

Copyright © 2018 The Khronos Group
Khronos Safety Critical Advisory Forum

Industry outreach and cooperation

AESIN
Automotive ADAS & AV + security
https://aesin.org.uk

MISRA C++
C++ WG23 Programming Vulnerabilities
ISO C Safe and Secure SG
ISO C++ Vulnerabilities Safety Critical SG

Generating guidelines for designing safety critical APIs to ease system certification.
Open to Khronos member AND industry experts
https://www.khronos.org/advisors/kscf

We are inviting safety critical experts to join KSCAF!
No cost or work commitment

Khronos SC Activities

OpenCL SC TSG
Working on OpenCL SC
Gathering requirements

SYCL SC
Guidelines to augment Industry First Safe and Secure Parallel and Heterogeneous C++
Safe AI for Automotive

OpenVX SC 1.1 - May 2017
Restricted “deployment”
implementation only executes pre-compiled binary format
Need for Camera Control API?

- Advanced control of sensor and camera subsystem - with cross-platform portability
  - Generate sophisticated image stream for advanced imaging & vision apps
- No platform API currently fulfills all developer requirements
  - Portable access to growing sensor diversity: e.g. depth sensors and sensor arrays
  - Cross sensor synch: e.g. synch of camera and MEMS sensors
  - Advanced, high-frequency per-frame burst control of camera/sensor: e.g. ROI
  - Multiple input, output re-circulating streams with RAW, Bayer or YUV Processing

OpenKCAM standard is currently on ice - do we need to restart?
Key Takeaways and What’s Next?

• Vision tools and API ecosystem becoming increasingly sophisticated
  • Layering libraries, languages and run-times
  • Machine learning stacks continue to grow in flexibility

• Exchange formats and compiler technologies essential to flexible deployment
  • Open standards evolving alongside proprietary solutions
  • For developers that value cross-platform deployment

• Safety-critical APIs becoming increasingly essential
  • Many vision applications need system certification

• Still no cross-vendor camera APIs?
  • Is the time yet right for this to be a target for standardization?

• Please join if your company interested in Khronos open standards
  • Neil Trevett | ntrevett@nvidia.com | @neilt3d