Executing Graphs with OpenCL

Erik Tomusk

IWOCL ‘21

26-29 April 2021
Enabling AI & HPC to be Open, Safe & Accessible to All

**Products**

- **Acoran**
  - Integrates all the industry standard technologies needed to support a very wide range of AI and HPC

- **ComputeAorta**
  - The heart of Codeplay’s compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

- **ComputeCpp**
  - C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

**Company**

Leaders in enabling high-performance software solutions for new AI processing systems

- Enabling the toughest processors with tools and middleware based on open standards
- Established 2002 in Scotland with ~80 employees

**Markets**

- High Performance Compute (HPC)
- Automotive ADAS, IoT, Cloud Compute
- Smartphones & Tablets
- Medical & Industrial

**Technologies:** Artificial Intelligence, Vision Processing, Machine Learning, Big Data Compute

**Partners**

- Broadcom
- CEVA
- Imagination
- Intel
- KNC
- NSI-TEXE
- Argonne
- Renesas
- Synopsys

And many more!
About me
About me

• Software engineer
About me

• Software engineer
• ComputeAorta team — OpenCL runtime
About me

- Software engineer
- ComputeAorta team — OpenCL runtime
  - www.youtube.com/watch?v=enyywRWJ7PA
About me

• Software engineer
• ComputeAorta team — OpenCL runtime
  • www.youtube.com/watch?v=enyvwRWJ7PA

Thanks:
  Romain Biessy
  Mehdi Goli
  Victor Lomüller
  Andrew Richards
  ComputeAorta team
  … and many others at Codeplay
Motivation
Motivation
Motivation
Motivation
Motivation
Motivation
Motivation

• Machine learning requires huge amounts of compute resources
Motivation

• Machine learning requires huge amounts of compute resources
• Machine learning problems are shaped like graphs
Motivation

• Machine learning requires huge amounts of compute resources
• Machine learning problems are shaped like graphs
• Large accelerators are being designed to execute graphs
Motivation

• Machine learning requires huge amounts of compute resources
• Machine learning problems are shaped like graphs
• Large accelerators are being designed to execute graphs
• Where does OpenCL fit in?
Motivation
Motivation

Accelerator Device
Motivation
Motivation
Motivation
Motivation

ML Model

ML Compiler

Accelerator Device
Motivation

ML Model
ML Compiler
ML Library

Accelerator Device
Motivation
Motivation

- ML Model
- ML Compiler
- ML Library
- Device Compiler
- Device Driver

Accelerator Device
Motivation

- ML Model
- ML Compiler
- ML Library
- Device Compiler
- Device Driver
- Accelerator Device
- CPU
Motivation

- ML Model
- ML Compiler
- ML Library
- Device Compiler
- Device Driver

Accelerator Device

CPU
Motivation
Motivation

Python Interface

Device Driver

Accelerator Device

CPU
Motivation

Python Interface
TensorFlow

Device Driver
Accelerator Device

CPU
Motivation

- Python Interface
- TensorFlow
- SYCL
- Device Driver
- Accelerator Device
- CPU
Motivation

- Python Interface
- TensorFlow
- SYCL
- OpenCL
- Device Driver

Accelerator Device

CPU
Motivation

• Can the OpenCL layer “see” the execution graph
• ... and explain it to the device driver?
Motivation

• Can the OpenCL layer “see” the execution graph
  • … and explain it to the device driver?
• Execution graph comes down from SYCL
Motivation

- Can the OpenCL layer “see” the execution graph
  - … and explain it to the device driver?
- Execution graph comes down from SYCL
- The OpenCL implementation and the device driver are tightly integrated
Motivation

Good

- Python Interface
- TensorFlow
- SYCL
- OpenCL
- Device Driver
- Accelerator Device
- CPU

- Accelerator Device
- Device Driver
- OpenCL
- SYCL
- TensorFlow
- Python Interface

Good
Motivation
Graphs in OpenCL
Graphs in OpenCL

• What does the OpenCL API “see”? 

© 2021 Codeplay Software Ltd.
Graphs in OpenCL

- What does the OpenCL API “see”?
- Kernels (functions) execute code
Graphs in OpenCL

- What does the OpenCL API “see”?
- Kernels (functions) execute code
Graphs in OpenCL

- What does the OpenCL API “see”? 
- Kernels (functions) execute code 
- Buffers store data
Graphs in OpenCL

• What does the OpenCL API “see”?
• Kernels (functions) execute code
• Buffers store data
  • Input, output, or inout
Graphs in OpenCL

• What does the OpenCL API “see”?  
• Kernels (functions) execute code  
• Buffers store data  
  • Input, output, or inout
Graphs in OpenCL

- What does the OpenCL API “see”?
- Kernels (functions) execute code
- Buffers store data
  - Input, output, or inout
- Kernels are enqueued for execution
Graphs in OpenCL

- What does the OpenCL API “see”?
- Kernels (functions) execute code
- Buffers store data
  - Input, output, or inout
- Kernels are enqueued for execution
- Some buffers written previously
- Some buffers are new
Graphs in OpenCL

• What does the OpenCL API “see”?  
• Kernels (functions) execute code  
• Buffers store data  
  • Input, output, or inout  
• Kernels are enqueued for execution  
• Some buffers written previously  
• Some buffers are new  
• Some buffers are reused, others aren’t
Graphs in OpenCL

- What does the OpenCL API “see”?
- Kernels (functions) execute code
- Buffers store data
  - Input, output, or inout
- Kernels are enqueued for execution
- Some buffers written previously
- Some buffers are new
- Some buffers are reused, others aren’t
Graphs in OpenCL

• What does the OpenCL API “see”?
• Kernels (functions) execute code
• Buffers store data
  • Input, output, or inout
• Kernels are enqueued for execution
• Some buffers written previously
• Some buffers are new
• Some buffers are reused, others aren’t
• Buffer usage implies dependencies between kernels
Graphs in OpenCL
Graphs in OpenCL

- Commands are enqueued onto command-queues
Graphs in OpenCL

- Commands are enqueued onto command-queues
- Command-queues enforce happens-before dependencies
Graphs in OpenCL

- Commands are enqueued onto command-queues
- Command-queues enforce happens-before dependencies
- *In-order* queues infer dependencies from enqueue order
Graphs in OpenCL

- Commands are enqueued onto command-queues
- Command-queues enforce happens-before dependencies
- **In-order** queues infer dependencies from enqueue order
- **Out-of-order** queues require explicit event wait lists from OpenCL user
Graphs in OpenCL

- Commands are enqueued onto command-queues
- Command-queues enforce happens-before dependencies

For an in-order command queue, the kernel instances appear to launch and then execute in the same order; where we use the term appear to emphasize that when there are no dependencies between commands and hence differences in the order that commands execute cannot be observed in a program, an implementation can reorder commands even in an in-order command queue.

— The OpenCL™ Specification v3.0.6 §3.2.2 (emphasis added)
Graphs in OpenCL

• Commands are enqueued onto command-queues
• Command-queues enforce happens-before dependencies
• *In-order* queues infer dependencies from enqueue order
• *Out-of-order* queues require explicit event wait lists from OpenCL user
Graphs in OpenCL

• Commands are enqueued onto command-queues
• Command-queues enforce happens-before dependencies
• **In-order** queues infer dependencies from enqueue order
• **Out-of-order** queues require explicit event wait lists from OpenCL user
• Focus on in-order queues for the rest of the talk
Empirical Study
Empirical Study

• Train and use a simplified handwriting detection neural network
Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
  - Simplified neural network so graphs fit on slides
Empirical Study

• Train and use a simplified handwriting detection neural network
  • TensorFlow 1.9.0
  • Standard MNIST example
  • Simplified neural network so graphs fit on slides
  • Pre-Keras Python library
Empirical Study

• Train and use a simplified handwriting detection neural network
  • TensorFlow 1.9.0
  • Standard MNIST example
  • Simplified neural network so graphs fit on slides
  • Pre-Keras Python library
• TensorFlow using SYCL backend [1]

Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
  - Simplified neural network so graphs fit on slides
  - Pre-Keras Python library
- TensorFlow using SYCL backend [1]
  - Codeplay’s ComputeCpp SYCL implementation

Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
  - Simplified neural network so graphs fit on slides
  - Pre-Keras Python library
- TensorFlow using SYCL backend [1]
  - Codeplay’s ComputeCpp SYCL implementation
- SYCL using OpenCL backend

Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
  - Simplified neural network so graphs fit on slides
  - Pre-Keras Python library
- TensorFlow using SYCL backend [1]
  - Codeplay’s ComputeCpp SYCL implementation
- SYCL using OpenCL backend
  - Codeplay’s ComputeAorta OpenCL

Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
  - Simplified neural network so graphs fit on slides
  - Pre-Keras Python library
- TensorFlow using SYCL backend [1]
  - Codeplay’s ComputeCpp SYCL implementation
- SYCL using OpenCL backend
  - Codeplay’s ComputeAorta OpenCL
  - Instrumented to record OpenCL API calls

Empirical Study

- Train and use a simplified handwriting detection neural network
  - TensorFlow 1.9.0
  - Standard MNIST example
  - Simplified neural network so graphs fit on slides
  - Pre-Keras Python library
- TensorFlow using SYCL backend [1]
  - Codeplay’s ComputeCpp SYCL implementation
- SYCL using OpenCL backend
  - Codeplay’s ComputeAorta OpenCL
  - Instrumented to record OpenCL API calls
- Trace training and inference stages separately

Empirical Study

• Train and use a simplified handwriting detection neural network
  • TensorFlow 1.9.0
  • Standard MNIST example
  • Simplified neural network so graphs fit on slides
  • Pre-Keras Python library
• TensorFlow using SYCL backend [1]
  • Codeplay’s ComputeCpp SYCL implementation
• SYCL using OpenCL backend
  • Codeplay’s ComputeAorta OpenCL
  • Instrumented to record OpenCL API calls
• Trace training and inference stages separately
• Use hacky scripts to convert traces into graphs

Empirical Study
Empirical Study

Inputs (Pixels) | Neuron Layer 1 | Neuron Layer 2 | Outputs (Digits)
---|---|---|---
0 | 0 | 0 | 0
1 | 1 | 1 | 1
2 | 2 | 2 | 2
... | ... | ... | ... 
27 | 127 | 31 | 9
Empirical Study
MNIST Inference Execution Graph
MNIST Training Execution Graph
Graphs in OpenCL

Python Interface
TensorFlow
SYCL
OpenCL
Device Driver
Accelerator Device
CPU
Graphs in OpenCL

Python Interface
TensorFlow
SYCL
OpenCL
Device Driver
Accelerator Device
CPU
Graphs in OpenCL

Python Interface

TensorFlow

SYCL

OpenCL

Device Driver

Accelerator Device

CPU
Execution graph is visible to OpenCL
Graphs in OpenCL
Graphs in OpenCL

• Execution graph is visible to OpenCL
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
- Simplified MNIST model has one layer
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
- Simplified MNIST model has one layer
  - Deep neural networks can have hundreds of layers
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
- Simplified MNIST model has one layer
  - Deep neural networks can have hundreds of layers
- This particular API stack hides some producer/consumer information from OpenCL
Graphs in OpenCL

• Execution graph is visible to OpenCL
  • OpenCL implementation can work with the device driver to execute the graph
• Diagram showed one iteration of training a simplified MNIST example
  • This model takes a few thousand iterations to train
  • Vulkan-style command buffers could reduce the number of times the graph must be constructed
• Simplified MNIST model has one layer
  • Deep neural networks can have hundreds of layers
• This particular API stack hides some producer/consumer information from OpenCL
  • Could be improved with:
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
- Simplified MNIST model has one layer
  - Deep neural networks can have hundreds of layers
- This particular API stack hides some producer/consumer information from OpenCL
  - Could be improved with:
    - Explicitly declaring read-only and write-only buffers
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
- Simplified MNIST model has one layer
  - Deep neural networks can have hundreds of layers
- This particular API stack hides some producer/consumer information from OpenCL
  - Could be improved with:
    - Explicitly declaring read-only and write-only buffers
    - Using the __constant address space
Graphs in OpenCL

• Execution graph is visible to OpenCL
  • OpenCL implementation can work with the device driver to execute the graph
• Diagram showed one iteration of training a simplified MNIST example
  • This model takes a few thousand iterations to train
  • Vulkan-style command buffers could reduce the number of times the graph must be constructed
• Simplified MNIST model has one layer
  • Deep neural networks can have hundreds of layers
• This particular API stack hides some producer/consumer information from OpenCL
  • Could be improved with:
    • Explicitly declaring read-only and write-only buffers
    • Using the __constant address space
    • Using out-of-order command-queues
Graphs in OpenCL

- Execution graph is visible to OpenCL
  - OpenCL implementation can work with the device driver to execute the graph
- Diagram showed one iteration of training a simplified MNIST example
  - This model takes a few thousand iterations to train
  - Vulkan-style command buffers could reduce the number of times the graph must be constructed
- Simplified MNIST model has one layer
  - Deep neural networks can have hundreds of layers
- This particular API stack hides some producer/consumer information from OpenCL
  - Could be improved with:
    - Explicitly declaring read-only and write-only buffers
    - Using the __constant address space
    - Using out-of-order command-queues
    - Using device-specific OpenCL extensions
Conclusion
Conclusion

• OpenCL can be used to represent graphs to graph accelerators
Conclusion

• OpenCL can be used to represent graphs to graph accelerators
• OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
Conclusion

- OpenCL can be used to represent graphs to graph accelerators
- OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
  - Many ways to get it wrong
Conclusion

- OpenCL can be used to represent graphs to graph accelerators
- OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
  - Many ways to get it wrong
  - In the API stack
Conclusion

- OpenCL can be used to represent graphs to graph accelerators
- OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
  - Many ways to get it wrong
  - In the API stack
  - In the OpenCL implementation
Conclusion

• OpenCL can be used to represent graphs to graph accelerators
• OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
  • Many ways to get it wrong
  • In the API stack
  • In the OpenCL implementation
• Most users don’t write OpenCL code, but use an API with an OpenCL backend
Conclusion

• OpenCL can be used to represent graphs to graph accelerators
• OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
  • Many ways to get it wrong
  • In the API stack
  • In the OpenCL implementation
• Most users don’t write OpenCL code, but use an API with an OpenCL backend
  • API developer’s responsibility to ensure API communicates user’s graph to OpenCL
Conclusion

• OpenCL can be used to represent graphs to graph accelerators
• OpenCL by itself does not guarantee that an accelerator’s device driver will see a graph
  • Many ways to get it wrong
  • In the API stack
  • In the OpenCL implementation
• Most users don’t write OpenCL code, but use an API with an OpenCL backend
  • API developer’s responsibility to ensure API communicates user’s graph to OpenCL
  • OpenCL implementor’s responsibility to communicate graph to device driver
We're Hiring!
codeplay.com/careers/

erik@codeplay.com

@codeplaysoft  info@codeplay.com  codeplay.com