HKG18-417

OpenCL Support by NNVM & TVM

Jammy Zhou - Linaro
Agenda

- OpenCL Overview
- OpenCL in NNVM & TVM
- Current Status
OpenCL Introduction

- Open Computing Language
  - Open standard maintained by Khronos with vendor specific extensions
  - Cross platform, heterogeneous, parallel programming
  - CPU, GPU, DSP, FPGA, and other accelerators

- OpenCL versions:
  - OpenCL 1.0: Dec. 08, 2008
  - OpenCL 1.1: June 14, 2010
  - OpenCL 1.2: Nov. 15, 2011
  - OpenCL 2.0: Nov. 18, 2013
  - OpenCL 2.1: Nov. 16, 2015
  - OpenCL 2.2: May 16, 2017
  - OpenCL-V

- Future convergence with Vulkan

- Supported devices:
  - Desktop GPUs
  - Desktop CPUs/GPUs/APUs
  - Mobile GPUs
  - FPGAs
  - DSPs
  - ARM
  - Qualcomm
  - Xilinx
  - Texas Instruments
  - VeriSilicon

- Adobe PDF converter created the image.
OpenCL Programming Model

● A typical OpenCL application consists of host program and device program
  ○ Host APIs
    ■ Platform/device management, context management
    ■ Memory management, command queue and scheduling
    ■ Program object, kernel object, kernel execution, etc
  ○ Device kernels
    ■ OpenCL C (C99 subset with extensions for parallelism)
    ■ OpenCL C++ (static C++14 with templates, lambdas and etc, introduced in OpenCL 2.2)
    ■ On-device dispatch support was introduced in OpenCL 2.0

● OpenCL Compiler
  ○ Convert the device kernel to device executable code
  ○ Support online and offline compiling
  ○ SPIR (Standard Portable Intermediate Representation)
    ■ SPIR 1.2 and SPIR 2.0 were based on LLVM
    ■ SPIR-V cross-API standard introduced in OpenCL 2.1 and Vulkan, isolated from LLVM
OpenCL Execution Model

- HW Implementations for Parallel Computing
  - SIMD (Single Instruction Multiple Data)
    - NEON for Arm CPU, SSE/AVX for X86 CPU, Arm Mali GPUs, etc
  - VLIW (Very Long Instruction Word, a.k.a, MIMD)
    - Arm Mali GPUs
  - SIMT (Single Instruction Multiple Threads)
    - Warps for Nvidia and Imagination GPUs, wavefronts for AMD GCN GPUs

- OpenCL Data Parallel Abstraction
  - NDRRange (a.k.a, Index Space)
  - Work Group
  - Work Item
    - Minimum kernel execution unit

Host API: clEnqueueNDRangeKernel

Kernel Functions:
- get_work_dim
- get_global_size
- get_global_id
- get_global_offset
- get_num_groups
- get_group_id
- get_local_id
- get_local_size
OpenCL Memory Model

- **Memory Types**
  - Global/Constant Memory - Accessible to all work items
  - Local Memory - Accessible to work items in the same work group
  - Private Memory - Accessible to only one work item

- **SVM (Shared Virtual Memory, introduced in OpenCL 2.0)**
  - Extends the global memory region into the host memory region
  - Makes buffer sharing between host and device easier
  - Three SVM types
    - Coarse-Grained Buffer SVM
    - Fine-Grained Buffer SVM (optional)
    - Fine-Grained System SVM (optional)

- **Platform/device specific configurations**
NNVM & TVM Introduction

- Fragmentation in the AI ecosystem: multiple frameworks with different front-end interfaces and backend implementations
  - Difficult to switch between frameworks by application and algorithm developers
  - Different backends to maintain by framework developers for various accelerators
  - Multiple frameworks to support by chip vendors with duplicated efforts

- NNVM & TVM to address above challenges
  - **NNVM (Neural Network Virtual Machine)** - Computation Graph IR stack
    - Computation graphs are standardized and optimized for models from different frameworks
    - NNVM compiler leverages TVM backend implementations
  - **TVM (Tensor Virtual Machine)** - Tensor IR stack
    - Inspired by HalideIR, and is DLPack compatible
    - Multiple language support: C++, Python, Java and Javascript
    - Minimum runtime for deployment on ‘small’ embedded devices
  - OpenCL to support inference acceleration by mobile GPUs on Arm platforms
NNVM & TVM Compilation Stack

DL Frameworks
- mxnet
- Caffe
- Theano
-CNTK

NNVM Frontends
- Pre-trained Models
  - mxnet
  - K
  - ML

NNVM Symbol & Param Dict

NNVM Compiler
- Computation Graph Optimizations
- Tensor Operators & Property Registry
- Compiler Toolchain

TVM
- TOPI
  - Operators
  - Schedule Primitives & Optimizations
- Codegen

TVM Graph Runtime
- The final execution graph in JSON format
- TVM Modules for both host & device
- The updated param dict of str to NDArray

Local or RPC
- X86 CPU
- Arm CPU
- AMDGPU
- NVPTX

CUDA Optimization
- ROCM Optimization
- MALI Optimization
- RASP Optimization

CUDA
- LLVM
- SPIRV
- OpenCL
- Metal
TVM PackedFunc

- Type-erased function as unified function type in TVM
- Support restricted argument types for simplicity
  - int, float, string, PackedFunc, Module, TVM Node, and etc
  - Can register extension types for augmentation
- Used by both the compilation stack and the deployment stack

```cpp
class PackedFunc {
public:
    using FType = std::function<void (TVMArgs args, TVMRetValue* rv)>;
    PackedFunc() {} explicit PackedFunc(FType body) : body_(body) {}
    template<typename... Args>
    inline TVMRetValue operator()(Args&& ...args) const;
    inline void CallPacked(TVMArgs args, TVMRetValue* rv) const;
    inline FType body() const;
    bool operator==(std::nullptr_t null) const { return body_ == nullptr;
    } bool operator!=(std::nullptr_t null) const {
        return body_ != nullptr;
    }
private:
    FType body_;};
```

Global Function Registry (name -> PackedFunc)

```
tvm::runtime::Registry::Get
```

```
tvm::runtime::Registry::ListNames
```

Foreign Function Interface

```
tvm.register_func
tvm.get_global_func
tvm.list_global_func_names
```

Python

C++

```
c_runtime_api
```

TVM_REGISTER_GLOBAL
TVM Runtime Device API

- **DeviceAPI** - abstracts the device specific interface for memory management
  - SetDevice
  - GetAttr
  - AllocDataSpace/FreeDataSpace
  - CopyDataFromTo
  - StreamSync
  - SetStream
  - AllocWorkspace/Freeworkspace

- **OpenCLWorkspace**
  - OpenCL implementation of DeviceAPI for above functions
TVM Module

● Module
  ○ Runtime container of packed functions generated by TVM
  ○ Driver glue are implemented in C++ and exposed to users such as Python via runtime API
  ○ Class functions defined in C++
    ■ GetFunction - Get packed function from current module by name
      ● Calls GetFunction of ModuleNode
    ■ Import - Import another module into this module
      ● OpenCL module is imported to the host module
    ■ LoadFromFile - Load a host function module from file

● ModuleNode - internal container of Module
  ○ OpenCLModuleNode - handle the execution of OpenCL kernels in one OpenCL program
    ■ Implement GetFunction, SaveToFile, SaveToBinary and GetSource functions
      ● OpenCLWrappedFunc is defined and returned to invoke the OpenCL kernel for GetFunction
    ■ Since clSetKernelArg is not thread-safe, a thread-local kernel table is maintained for thread-safe support
    ■ OpenCLModuleCreate
      ● Create/initialize the OpenCLWorkspace and OpenCLModuleNode, returns TVM Module
TVM TOPI (Tensor OPerator Inventory)

- **Python decorator mechanism**
  - @tvm.target.generic_func
    - Wrap a target generic function with dispatch func
    - Dispatch dictionary is maintained (key -> func)
  - @func.register
    - Register special function to the dispatch dictionary

- **Python context switch via ‘with’ keyword**
  - A context manager (class Target here) follows ‘with’
  - __enter__ and __exit__ methods for context switch

- **Schedule and operator for OpenCL**
  - OpenCL reuses the ROCm schedule, ROCm sets both “rocm” and “gpu” keys
  - CUDA target register many functions with both “cuda” and “gpu” keys
  - Mali target registered functions for conv2d, schedule_conv2d_nchw, schedule_dense, and schedule_depthwise_conv2d_nchw

- **NNVM leverage the operator attribute mechanism to use functions in TOPI**

```python
import tvm

# wrap function as target generic
@tvm.target.generic_func
def my_func(a):
    return a + 1

# register specialization of my_func under target cuda
@my_func.register("cuda")
def my_func_cuda(a):
    return a + 2

# displays 3, because my_func is called
print(my_func(2))

# displays 4, because my_func_cuda is called
with tvm.target.cuda():
    print(my_func(2))
```
OpenCL CodeGen

- LoweredFunc is the final IR representation before codegen
  - The schedules are translated to LoweredFuncs with the IR Pass
  - Functions for OpenCL device are separated from host

```
CodeGenOpenCL
- storage scope and sync
- fp16/fp64 extension
- thread index
- type handling
- vector support, ...
```

```
Array<LoweredFunc> ->CodeGenOpenCL
```

```
CodeGenOpenCL
```

```
CodeGenOpenCL->OpenCLModuleCreate
```

```
OpenCLModuleCreate
```

```
OpenCLModuleCreate->OpenCLModuleNode
```

```
OpenCLModuleNode
```

```
Module
```
TVM Deployment Stack

Execution graph in JSON format

OpenCL Context

Graph Runtime

JSON Reader

Storage Setup

Executor Setup

set_input

get_output

prediction **tabby, tabby cat**

Notes: For RPC deployment, OpenCL support should also be enabled in host side

Param dict of str to NDArray

set_input

DeviceAPI

GetFunction

OpenCLWorkspace

TVM Module (with OpenCL Module)
Examples with OpenCL

- Simple MAD operation
  - Local
  - RPC

- MXNet resnet18_v1 model for prediction
  - Local
  - RPC
Current Status

- **Optimization** for Mali T860 MP4 (OpenCL 1.2) with Mali specific schedule
  - Done by Lianmin Zheng from SJTU on Firefly RK3399 platform
  - Also verified on Rock960 96Boards with RK3399

* ACL v17.12 was used for comparison
* VGG16 requires more than 2GB memory
Next?

- Bubblegum96 (PowerVR G6230, OpenCL 1.2 EP)
  - Basic OpenCL verification done
  - PowerVR specific schedule to address issues - TODO
- More 96Boards to support
  - Hikey960 (Mali G71 MP8, OpenCL 2.0)
  - DB820c (Adreno 530, OpenCL 2.0)
- Others
  - Integrate ACL into TVM for Mali similarly as cuDNN
  - Validation of more frameworks and models with NNVM/TVM
  - Additional support and optimizations for OpenCL 2.0
  - Anything else?

TVM Roadmap: https://github.com/dmlc/tvm/projects/1
Thank You

#HKG18

HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org