nGraph/PlaidML Demo for SysML 2019

Robert Earhart [1]

Abstract

We propose to demonstrate the nGraph/PlaidML compiler toolchain, by performing inference and training of a simple style transfer network on commodity hardware. TensorFlow integration and custom operation definition will be included.

1 Technology

The PlaidML compiler generates specialized machine learning programs for a variety of hardware targets, such as commodity CPUs and GPUs. It is straightforward to install and configure on most well-known operating systems, often by installing a single Python package and running a setup command to select the desired hardware backend.

A specialized machine learning program generated by the compiler consists of a set of kernels shaped for optimized execution given the available hardware resources, and an execution schedule that orders kernels to take advantage of temporal memory locality and to fit within device memory constraints.

It's straightforward to write novel machine learning operations and compile them by using *Tile*, a language for specifying polyhedral tensor operations. Additionally, most common machine learning operations are provided via *nGraph*, a machine-learning operation library. *nGraph* supports a number of frontend interfaces, including TensorFlow and ONNX; it can also be coded to directly, either in C++ or in Python.

Both PlaidML and nGraph are licensed under the Apache License 2.0.

2 Novelty

The demonstration shows that:

- It's relatively easy to set up the nGraph/PlaidML toolchain on commodity hardware
- The toolchain has good performance out-of-the-box
- The toolchain supports training and inference
- The toolchain supports writing novel kernels
- The toolchain includes tools to help better understand performance

Taken together, we believe the demonstration shows the utility of the nGraph/PlaidML toolchain for both production and research use cases.

3 Live Action

The audience will be shown a MacBook Pro. The machine will be running the stock operating system configuration, with the additions of iTerm2, Anaconda, and the Anaconda packages & Python wheels implementing the demo.

The demo will start by opening an iTerm2 terminal window, which the demonstrator will use to:

- Create an Anaconda environment with PlaidML, nGraph, the nGraph/TensorFlow bridge, TensorFlow, and the demo itself:

  ```
  conda env create -n demo
  conda activate -n demo
  ```

- Configure PlaidML:

  ```
  plaidml-setup
  ```

- Launch the demo Jupyter notebook:

  ```
  ngraph-plaidml-demo-sysml-2019
  ```

The first cell of the notebook will show the device configuration being used by PlaidML — i.e. what needs to be specified about a device in order for PlaidML to generate efficient kernels for it.

The next cell of the notebook will compare the latency of style transfer using native TensorFlow (as installed from

[1]Intel Corporation, Santa Clara, California, USA. Correspondence to: Robert Earhart <robert.earhart@intel.com>.

Proceedings of the 2^{nd} SysML Conference, Palo Alto, CA, USA, 2019. Copyright 2019 by the author(s).

Anaconda) and TensorFlow with the PlaidML backend (via nGraph and the nGraph/TensorFlow bridge). One UI button will be used to start the camera feed, and a second will take a snapshot (stopping the feed) and apply two models to it, trained from Vincent Van Gogh's *Starry Night* and Georges Seurat's *A Sunday on La Grande Jatte*. On snapshot, for each backend, a row of three images will be displayed: a source image captured by the camera, followed by the image after transformation by each model, with the transfer latency displayed below each output image.

The next cell of the notebook will demonstrate the use of PlaidML's eventing system for analyzing system performance. Again, one button will be used to start the camera feed, and a second will be used to take a snapshot (stopping the feed). On snapshot, the notebook will apply a style transfer using PlaidML, gathering a trace log; when complete, the cell will display a scatter plot of the style transfer kernels. When a kernel is selected, the generated source code of the kernel will be displayed below the scatter plot.

The next cell of the notebook will demonstrate training a style transfer model, using Leonardo da Vinci's *Mona Lisa* as the source image, displaying the time per epoch.

The final cell of the notebook will demonstrate writing a custom Tile operation. The cell code will contain a Tile function implementing a grouped convolution followed by a ReLU, along with sizes for the input tensors. When run, the cell will differentiate the function (displaying the autodiff-generated Tile code), compile both forward and backward passes, and display the generated source code of the resulting kernels.

4 Interactive Section

A MacBook Pro will be running style transfer inferences live from its builtin camera. A Jupyter notebook will display the live view next to the inference output. Above the outputs, the notebook will display controls allowing users to select the style model being used and the backend implementation; below the outputs, the notebook will display a rolling graph of inference times, with markers to show changes in the style model and in the selected backend.

Additionally, cards will be provided with a link (textual and QR) to a GitHub repository with the demo sources and links to pre-packaged Python wheels. The audience will be encouraged to experiment with the demo on their own equipment — all demo sources will be released under the Apache License 2.0.

5 Demo Equipment

The demonstrator will bring:

- Two machines: each, a 2018 13 inch MacBook Pro with touchbar, with an Intel Iris Plus Graphics 655. One will be configured to run the live action demo; the other will be reserved for audience interaction
- A monitor, for displaying the screen of the MacBook running the Live Action portion of the demo
- Take-home cards with links to the demo sources

6 Special Needs

The demo has no special needs.