Summer 2019 Internship Report

Gus Smith
November 18, 2019

sampl

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

tvm.ai

Disclaimer: I have no formal background in program analysis!

I have no formal background in program analysis. I have practical experience with data flow analyses of programs from my Master's, but that's it.

Summer goal: Analyze deep learning workloads for "accelerability"

Realization: Most common workloads expressed in Relay are simply straight-line code. These can be analyzed statically.

My solution:

1. Build simple analysis framework in Relay
2. Write analyses in the framework that gather useful information
3. Meet with architects, iterate on analyses

The things I can't talk about here are things like what workloads they cared about, what features they were looking at in terms of "accelerability".

(my definitions of) Static and Dynamic Analysis

- In my terminology, **static analysis ≈ dataflow analysis,** because these are straight-line programs
- **Dynamic analysis:** instrumenting code
- What other types of analysis are there out there?

Instrumenting code with counters and other data-gathering widgets

Static Analysis Framework

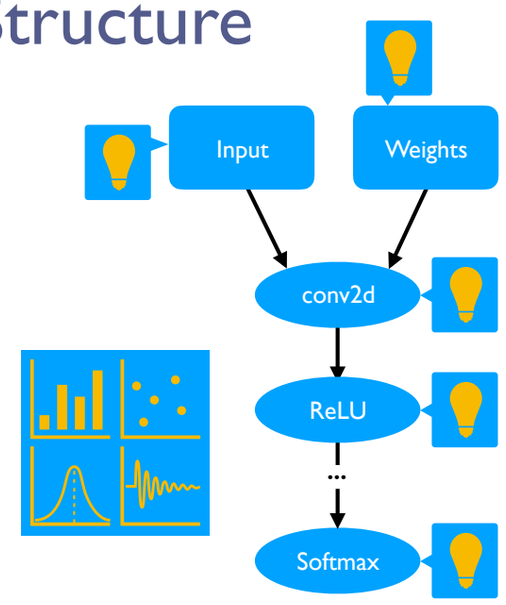
The analysis framework helps the user **design passes** which extract static information about a program, and helps the user **organize and export** that information.

There’s not much to this framework. It’s an incredibly thin layer over the existing pass framework, which simply makes it easier to attach data to program nodes, and format the output.

Analysis Structure

The analysis framework imposes a general structure for analyses

- Two parts: **initial analysis phase** (visiting every program node and extracting data) and **summary phase** (visiting the extracted data and producing summary results)
- Essentially **map** and **reduce** phases



Background: ExprVisitor

The analysis framework is built off of Relay's ExprVisitor.

```
1  import tvm
2  from tvm import relay
3
4
5  class MyPass(relay.ExprVisitor):
6      def visit_call(self, call):
7          super().visit_call(call)
8          print(call.op)
9
10
11 MyPass().visit(relay.const(1) - (relay.var('x') * relay.var('y')))
```

```
/Users/gus/.pyenv/versions/3.7.4/bin/python tmp.py
v0.0.4
multiply
v0.0.4
subtract
```

ExprVisitor allows you to visit the nodes of a Relay program.

A class which subclasses ExprVisitor can override ExprVisitor's visitor methods, to choose which types of nodes it visits. For example, overriding visit_call here will lead to us visiting the calls in this Relay program.

AnalysisPass Class

- A thin wrapper over the ExprVisitor class
- Helper functions like _add_detail make it easy to attach analysis data to a node
- Passes can depend on data generated by previous passes, but dependencies are implicit right now

The AnalysisPass class is the main thing added in the analysis framework

Demo

https://github.com/gussmith23/tvm/blob/analysis-framework-demo/demo.ipynb

Dynamic Analysis Experiments

We can instrument the program with counters in two ways:

- Using references—easy to implement, but not good Relay style
- "Functionally", where every value becomes a tuple of (value, counter)—better Relay style, but harder to implement

Note: Relay programs may not be dynamic enough to warrant dynamic analysis

Improvements/Future Directions

- Separate analysis description from analysis results
- Explicit dependencies between passes
- Make the framework more useful for gathering actual compiler passes
- Build dynamic analysis tools
  - Add passes to Relay which rewrite programs, passing around counters and other data-gathering things

Currently, an instance of an analysis is only usable once, because it stores the result data internally. I'd rather an instance of an analysis be more like a concrete instantiation of a parametrized analysis, which can be run over multiple workloads.

Maybe using the pass manager?

Links

https://github.com/microsoft/Analysis-Framework-for-TVM

https://github.com/gussmith23/tvm/blob/analysis-framework-demo/demo.ipynb