dmlc / **HalideIR**

Symbolic Expression and Statement Module for new DSLs

17 commits | **1** branch | **0** releases | **3** contributors

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

tqchen More relations updates — Latest commit `cb3c025` 18 days ago

src	[DOC] Fix doxygen	2 months ago
.gitignore	gitignore TAGS. (#11)	2 months ago
LICENSE	[NOTE] Add Release Note	4 months ago
Makefile	[NOTE] Add Release Note	4 months ago
README.md	More relations updates	18 days ago

README.md

HalideIR: Symbolic Arithmetic IR Module

HalideIR is a base module for building symbolic expression and arithmetic simplification for building new DSLs. It is isolated and refactored from part of Halide project (credit should go to the original authors). It is used in the TVM project.

Note that some portions of the TVM compiler outside of this folder are also adapted from Halide codebase, where we needed similar logic (e.g loop vectorization). These are commented where they occur.

Motivation

We build this component during the development of TVM project. Symbolic expression data structure and simplification libary is essential to for such compiler stack project.

Unfortunately there is no standalone module that fits our need at the moment. We find that the IR and simplification module of Halide project fits such purposes nicely. So we isolated and refactor the coresponding module into this repo.

The major goal is minimum and interpolatable with more front-end languages. HalideIR is used in TVM project. Here are the few major improvements.

- An isolated depenency free symbolic IR and simplification (no LLVM dependency).
- The project modularized into logical components.
- All IR structure are serializable and publically accessible from front-end language(e.g. python)
  - This supports ease of development and proptyping in python
- Think of re-usablity when adding new DSL structure.
  - A runtime dispatching mechanism is introduced to allow ease of add new IR node.
- Simplified variable defintiion rule: change from match by string to match by pointer
  - This ensures each variable to have single define location (like SSA)

Besides these changes, we also re-factored the code to resolve some of the issues in the codebase. Some of these changes addresses the pain point raised in Halide project. The detailed change are listed in the later part of the project.

Project Structure

Based on code from Halide(release_2017_05_03). The code is componetized into four logical components

- tvm: TVM container code for interpolation, basic data structures.
- base: base utilities and type.

- ir: The IR data structure
- arithmetic: Arithemetic simplification.

Code Style

We keep old files in old code style, but use Google C style in the newly added files.

List of Changes

- Replace implmentation of Float16.cpp with a simpler version that does not depend on LLVM
- IntrusivePtr
  - Remove IntrusivePtr, change everything to base on std::shared_ptr
  - See tvm/node.h
  - Add support for IR reflection via NodeRef.VisitAttrs
  - All the IR is constructable from python via TVM
    - This enables quick proptyping and debuging from python
- Call
  - Remove Parameter, BufferPtr from Variable, Call, Load
  - This simplifies dependency, we also find it is cleaner to simply use Variable for Parameter.
- AssertStmt
  - Add body field to AssertStmt, which represents the scope where the assert condition holds
  - This makes it easier to do Visitor pattern that take benefit of the scope assert information.
- AttrStmt
  - This is a new Stmt that can be used to annotate attribute of certain things (e.g. content type of buffer). This removes the need of string matching for properties and hacks around string matching in Let.
  - We use this extensively to add new hints in the environment scope.
- FunctionContent
  - Make FunctionBaseNode abstract, to replace FunctionContent

- Provide, Realize, Prefetch, ProducerConsumer, Call
  - Remove use of string name matching of function
  - When place where function is needed, use FunctionRef
  - FunctionRef is uniqued matched by internal pointer.
- Variable, Store, Allocate, Free, For, Load, Let, LetStmt
  - Remove use of string name matching of Variable
  - When place where variable is needed, use VarExpr
  - VarExpr is uniqued matched by internal pointer.
- Variable
  - Rename Variable.name -> Variable.name_hint, to avoid confusion.
  - Variable.name_hint is not used to uniquely identify the variable.
  - By default, each Variable should be only defined once
    - This is in analog to SSA, and can make ir pass simplier(remove need of scoping)
- Provide, Realize, Prefetch
  - Change Provide and Realize to associate with value_index.
  - Original Provide/Realize with multiple values can be represented by several Provide and Realize chained together
  - This allows outputs of a Function to have different shapes.
- Make every field the IR reflectable and accessible from python
  - std::vector<> - > Array<>(tvm/container.h)
  - struct Range -> Range(Range.h)
- Range
  - Remove constructor Range(min, extent) to Range::make_with_min_extent(min, extent)
  - The original constructor could make new user confuse since a typical way is range(begin, end) in both c++ and python
- Simplify Visitor
  - Remove use of string name mapping in the Scope
  - Remove Scope in Substitute to check name conflicts because we use variable pointer matching

    - Add Expr&/Stmt& to visit interface in the IRVisitor and IRMutator
- Add a more flexible RTTI and dynamic dispatch support, see src/tvm/node.h
    - IRFunctor allows plugin of new IR Node more easily, without chaning interface