New lms frontend and staged computation graphs[1]

Ruben Fiszel

December 6, 2016

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

[1]Thanks to my beloved parents, my awesome supervisor Nada Amin, Prof. Martin Odersky, the lms master and author Tiark Rompf, and the delite folks Kevin James Brown and David Koeplinger.

Abstract

In this report, we explore staging, in particular the LMS framework and the development of its new frontend whose aim is ease the writing of staged dsl through, among others, shadowing of types. We also explore the usage of this new frontend for a particular case study: Staged computation graphs.

Contents

Introduction

If programming can be considered as an art, then programming languages are the brush of the artists. The tools are chosen according to individual preferences, and of course the constraints of the desired final work. One of those major constraint is efficiency. Efficiency is less crucial nowadays than it was back when computing power was expensive and scarce, but it is still a very desirable goal. Efficiency can be achieved by writing explicitely very precisely each step of the program. To enable this, some programming languages are constrained to stay quite close from the model of the hardware. Those programming languages are called "low-level", because close from the machine and thus distant from the heights of abstraction. But as program grow more complex, the need for abstraction does too.

To fill the gap between abstraction and machine language, programming languages with complex compilers were born. Ironically, the first programmng language, lambda-calculus, was the epitomy of abstraction and thought of before the first very computer to execute it. Compilers are heavy machinery that applies all sort of optimization before translating source code to machine language. The more abstract the language, the more gap there is to fill, and the more compilers have opportunity to optimize.

Nevertheless, growing complexity in compiler is not the panacea. In a wide range of programs, compilers are limited by the lack of domain specific knowledge of the program. Indeed, constraining a program to one specific domain enables a wide range of optimization possible only in this domain. Furthermore, code generation by "macro" is a rather poor way to enable the further abstraction brought by code that generate code. One solution to both issue raised is to write specific DSL in a staged environment. It is this path that is explored with LMS, a scala library for runtime code generation. In this work, we will explore a new lms implementation in order to enable a new frontend, more convenient for the end-user and with multiple advantages enabled by extended typeclass use. We also cover a case study of lms applied to computation graphs.

LMS

Lightweight Modular Staging (LMS) is a library written in Scala that enable staged meta-programming. Meta-Programming is the art of writing computer programs that can handle some programs as data. Hence they are able to reason about programs, which they are themselves, hence the name. In staged meta-programming, the goal is to generate new programs.

Staging

Why staging

Exp tree

Deep reuse of embedding language order

Frontend/Backend ?

lms

library author

delite

user

The new frontend

Lift

Typeclass

Typeclass overloading

Primitives types and collections

Computation Graph

Graph

Cycle check

Arithmetic

Benchmark

DerivableGraph

Backpropagation

MatrixGraph

Dimensions check

Conclusion

blablabla