Programming Languages and Compiler Construction
Department of Computer Science
Christian-Albrechts-University of Kiel

Master Thesis

An LLVM Backend for Accelerate

Timo von Holtz

April 15, 2014

Advised By:
Priv.-Doz. Dr. Frank Huch
Assoc. Prof. Dr. Manuel M T Chakravarty

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

April 15, 2014

Timo von Holtz

Todo list

Contents

List of Figures

List of Tables

1 Introduction

Write Introduction

2 Contributions

2.1 llvm-general-quote

When writing a companyiler using LLVM in Haskell there is a good tutorial on how to do it at `http://www.stephendiehl.com/llvm/`. It uses *llvm-general* to interface with LLVM. The general idea is to use a monadic generator to produce the AST on the fly. Let's look at an code fragment to get an idea how this works.

```
for :: Type                                   -- type of the index
  → Operand                                   -- starting index
  → (Operand → CodeGen Operand)               -- loop test to keep going
  → (Operand → CodeGen Operand)               -- increment the index
  → (Operand → CodeGen ())                    -- body of the loop
  → CodeGen ()
for ti start test incr body = do
  loop ← newBlock "for.top"
  exit ← newBlock "for.exit"

    -- entry test
  c    ← test start
  top  ← cbr c loop exit

    -- Main loop
  setBlock loop
  c_i ← freshName
  let i = local c_i

  body i

  i'   ← incr i
  c'   ← test i'
  bot  ← cbr c' loop exit
  _    ← phi loop c_i ti [(i', bot), (start, top)]

  setBlock exit
```

As you can tell this is much boilerplate code. We have to define the basic blocks manually and add the instructions one by one. This has some obvious drawbacks, as the code can get unreadable pretty quickly.

A solution is to use quasiquotation[5] instead. The idea behind quasiquotation is, that you can define a DSL with arbitrary syntax, which you can then directly transform into

Haskell data structures. This is done at compile-time, so you get the same type safety as writing the AST by hand.

I implemented *llvm-general-quote*, a quasiquotation library for LLVM. Using my library, the code using a loop looks like this:

```
[llg|
define i64 @foo(i64 %start, i64 %end) {
  entry:
    br label %for

  for:
    for i64 %i in %start to %end with i64 [0,%entry] as %x {
        %y = add i64 %i, %x
        ret i64 %y
    }
}
|]
```

This will expand to the following LLVM IR:

```
define i64 @foo(i64 %start, i64 %end) {
entry:
  br label %for

for:                                   ; preds = %for.body, %entry
  %i = phi i64 [ %i.new, %for.body ], [ %start, %entry ]
  %x = phi i64 [ %y, %for.body ], [ 0, %entry ]
  %for.cond = icmp ule i64 %i, %end
  %i.new = add nuw nsw i64 %i, 1
  br i1 %for.cond, label %for.body, label %for.end

for.end:                               ; preds = %for
  ret i64 %x

for.body:                              ; preds = %for
  %y = add i64 %i, %x
  br label %for
}
```

This is clearly more readable. Furthermore, one can see much more clearly what the produced code will be.

Another advantage of quasiquotation is antiquotation. This means you can still reference arbitrary Haskell variables from within the quotation. Using this the following are equivalent:

- `[llinstr| add i64 %x, 1 |]`
- `let y = 1 in [llinstr| add i64 %x, $opr:(y) |]`

The design of *llvm-general-quote* is inspired by *language-c-quote*, which is also used in the cuda implementation of Accelerate. I use "Happy" and *Alex*.

Bibliography

[1] Stephen Diehl. *Implementing a JIT Compiled Language with Haskell and LLVM.* Jan. 2014. URL: `http://www.stephendiehl.com/llvm/`.

[2] Chris Dornan and Simon Marlow. *Alex: A lexical analyser generator for Haskell.* URL: `http://www.haskell.org/alex/`.

[3] Andy Gill and Simon Marlow. "Happy: the parser generator for Haskell". In: *University of Glasgow* (1995).

[4] Timo von Holtz. *llvm-general-quote.* URL: `https://github.com/tvh/language-llvm-quote`.

[5] Geoffrey Mainland. "Why it's nice to be quoted: quasiquoting for haskell". In: *Proceedings of the ACM SIGPLAN workshop on Haskell workshop.* ACM. 2007, pp. 73–82.

[6] Geoffrey Mainland and Manuel MT Chakravarty. *language-c-quote.* URL: `http://hackage.haskell.org/package/language-c-quote`.

[7] Benjamin Scarlet. *llvm-general.* 2013. URL: `http://hackage.haskell.org/package/llvm-general`.