Introduction to LLVM Compiler Infrastructure

Róbert Baručák, Jakub Šulek

Fakulta informačních technologií
VUT v Brně

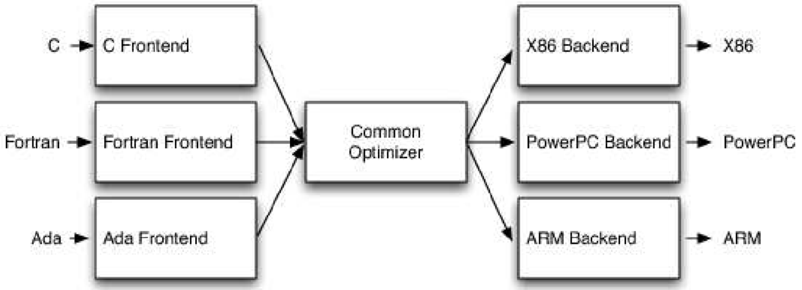
3. 12. 2012

LLVM

Collection of modular and reusable
compiler and toolchain technologies

- Modern, SSA-based compilation strategy
- Capable of static and dynamic compilation
- Arbitrary programming languages

Architecture



[1]

- Multiple source languages
- LLVM IR
- Multiple target architectures

Frontends and Optimizer

- Ada, C, C++, D, Fortran, Objective-C etc.
- Scalar, interprocedural, profile-driven optimizations
- Some simple loop optimizations

Backends

- Instruction Selection
- Scheduling and Formation
- Register Allocation
- Prolog/Epilog Code Insertion
- Late Machine Code Optimizations
- Code Emission

LLVM Intermediate Representation

LLVM IR is low-level RISC-like virtual instruction [3], [2]

- Key operations of ordinary processor
- Avoiding machine specific constraints

Three different forms of LLVM code representation:

- An in-memory compiler IR
- An on-disk bitcode representation (suitable for JIT)
- A human readable assembly language representation

LLVM Three Address Code

- Most LLVM operations in three address form
- LLVM IR has fairly standard set of arithmetic and logical instructions (`add, sub, mul, div ...`)
- Polymorphic instructions
- Exceptions `PHINode, call`

```
%X = div int 4, 9 ; Signed integer division
%Y = div uint 12, 4 ; Unsigned integer division
%cond = seteq int %X, 8 ; Produces a boolean value
br bool %cond, label %True, label %False
True:
...
```

SSA Form of LLVM IR

- SSA as a primary form of code representation
- Simplification of dataflow optimizations
- Need to control flow merges – definition of ϕ function

```
<result> = phi <type> [<val0>, <label0>], ... ,
[<valN>, <labelN>]
```

SSA Form of LLVM IR

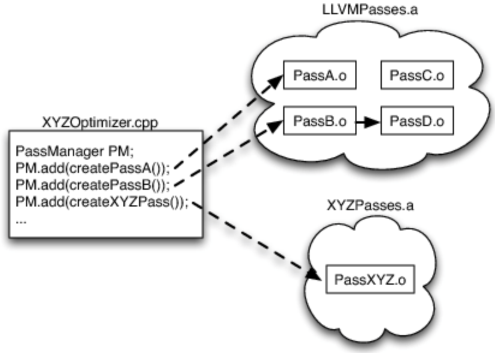
//PHINode - The PHINode class is used to represent the magical mystical PHI node, that can not exist in nature, but can be synthesized in a computer scientist's overactive imagination.

PHINode.h

LLVM Passess

Three main types of passess [4]

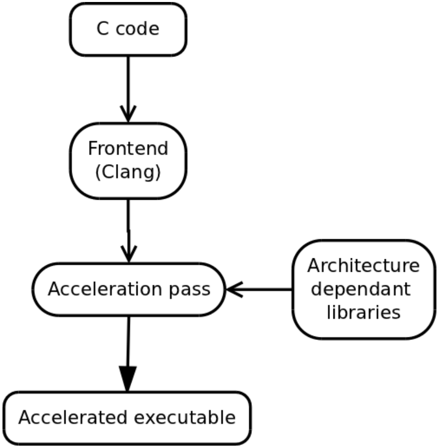
- Analysis
- Transform
- Utility



[1]

Applications of LLVM framework

RAVAC(Robust Automatic Vector Accelerating Compiler)



Applications of LLVM framework

- Clang – "LLVM native"C/C++/Objective-C compiler
- Polly – auto-parallelism and vectorization using a polyhedral model
- Dragonegg – integrates LLVM with GCC 4.5 parsers

Thank you

Amy Brown and Greg Wilson.
The Architecture of Open Source Applications.
2011.

Chris Lattner.
The llvm target-independent code generator [online].
http://llvm.org/docs/CodeGenerator.html, 2012-04-19 [cit. 2012-05-01].

Chris Lattner and Vikram Adve.
Llvm language reference manual [online].
http://llvm.org/docs/LangRef.html, 2012-04-19 [cit. 2012-05-04].

Reid Spencer and Gordon Henriksen.
Llvm's analysis and transform passes [online].
http://llvm.org/docs/Passes.html, 2012-04-19 [cit. 2012-05-04].