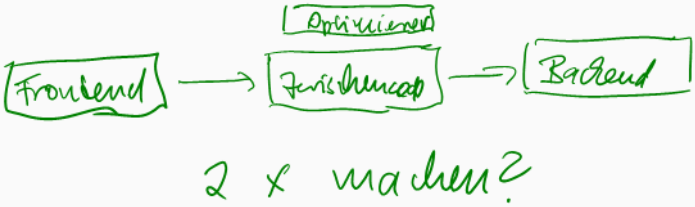
LLVM als IR

BC George (HSBI)

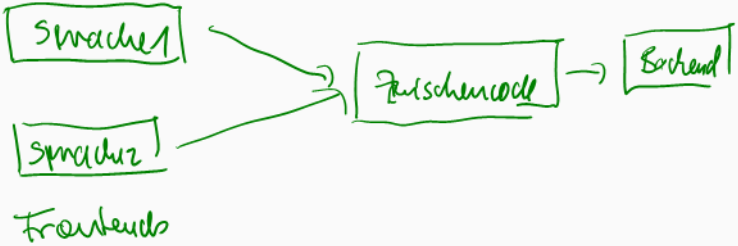
Motivation

Motivation

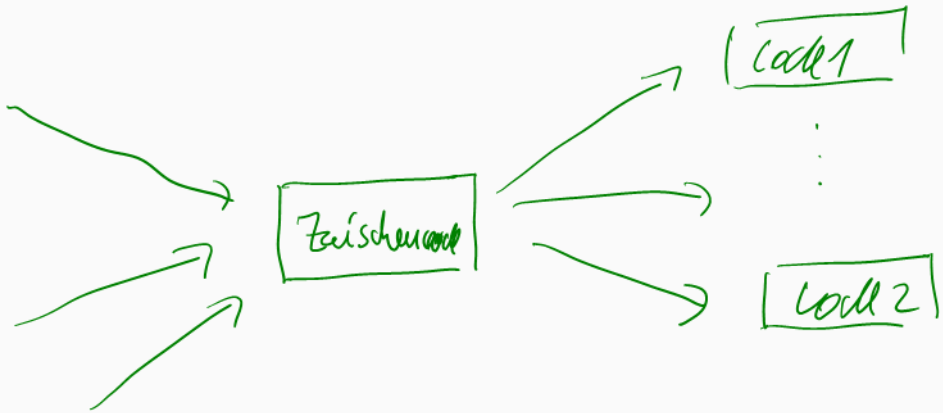


Es ist ein neuer Prozessor entwickelt worden mit einem neuen Befehlssatz, und es sollen für zwei Programmiersprachen Compiler entwickelt werden, die diesen Befehlssatz als Ziel haben.

Was tun?



Thema für heute: *Ein* Zwischencodeformat für verschiedene Programmiersprachen und Prozessoren



LLVM - Ein Überblick

Was ist LLVM?

- **ursprünglich**: **L**ow **L**evel **V**irtual **M**achine
- Open-Source-Framework
- zur modularen Entwicklung von Compilern u. ä.
- für Frontends für beliebige Programmiersprachen
- für Backends für beliebige Befehlssatzarchitekturen

“Macht aus dem Zwischencode LLVM IR automatisch Maschinencode oder eine VM.”

Kernstücke des LLVM:

- ein virtueller Befehlssatz
- ein virtuelle Maschine
- LLVM IR: eine streng typisierte Zwischensprache
- ein flexibel konfigurierbarer Optimierer
- ein Codegenerator für zahlreiche Architekturen
- LMIR: mit Dialekten des IR arbeiten

Was kann man damit entwickeln?

- Debugger
- JIT-Systeme (virtuelle Maschine)
- AOT-Compiler
- virtuelle Maschinen
- Optimierer
- Systeme zur statischen Analyse
- etc.

mit entkoppelten Komponenten, die über APIs kommunizieren (Modularität)

Wie arbeitet man damit?

- (mit Generatoren) ein Frontend entwickeln, das Programme über einen AST in LLVM IR übersetzt
- mit LLVM den Zwischencode optimieren
- mit LLVM Maschinencode oder VM-Code generieren

Was bringt uns das?

n Sprachen für m Architekturen übersetzen:

- n Frontends entwickeln
- Optimierungen spezifizieren
- m Codegeneratoren spezifizieren

statt $n \times m$ Compiler zu schreiben.

Wer setzt LLVM ein?

Adobe AMD Apple ARM Google

IBM Intel Mozilla Nvidia Qualcomm

Samsung ...

Externe LLVM-Projekte

Für folgende Sprachen gibt es Compiler oder Anbindungen an LLVM (neben Clang):

Crack Go Haskell Java Julia Kotlin
Lua Numba Python Ruby Rust Swift ...

Für weitere Projekte siehe Projects built with LLVM

Unterstützte Prozessorarchitekturen

x86 AMD64 PowerPC PowerPC 64Bit Thumb

SPARC Alpha CellSPU PIC16 MIPS

MSP430 System z XMOS Xcore ...

Einige Komponenten von LLVM

Einige Komponenten (Projekte) von LLVM

- Der LLVM-Kern incl. Optimierer
- MLIR für IR-Dialekte
- Der Compiler Clang
- Die Compiler-Runtime-Bibliothek

Der LLVM-Kern

LLVM Core: Optimierer und Codegenerator für viele CPU- und auch GPU-Architekturen

- Optimierer arbeitet unabhängig von der Zielarchitektur (nur auf der LLVM IR)
- sehr gut dokumentiert
- verwendete Optimierungspässe fein konfigurierbar
- Optimierer auch einzeln als Tool `opt` aufrufbar
- wird für eigene Sprachen als Optimierer und Codegenerator eingesetzt

Wozu ein Optimierer?

- zur Reduzierung der Codegröße
- zur Generierung von möglichst schnellem Code
- Zur Generierung von Code, der möglichst wenig Energie verbraucht

Allgegenwärtig in LLVM: Der Optimierer

Der Optimierer in LLVM

- Teil von LLVM Core
- kann zur Compilezeit, Linkzeit und Laufzeit eingesetzt werden
- nutzt auch Leerlaufzeit des Prozessors
- läuft in einzelnen unabhängig konfigurierbaren Pässen über den Code

Einige Optimierungen in LLVM

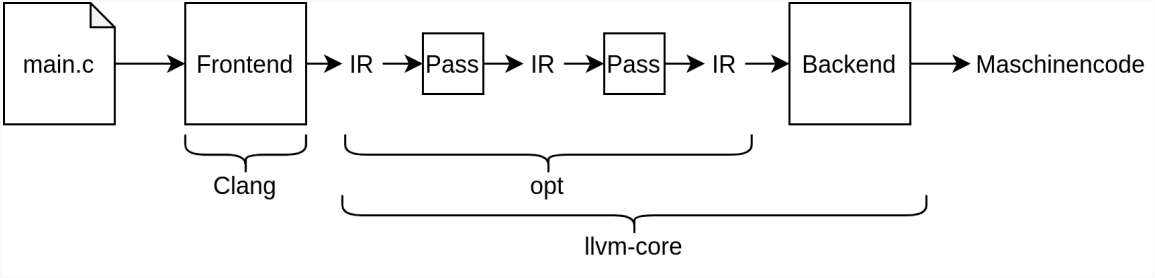
- Dead Code Elimination
- Aggressive Dead Code Elimination
- Dead Argument Elimination
- Dead Type Elimination
- Dead Instruction Elimination
- Dead Store Elimination
- Dead Global Elimination

MLIR

- Framework zur Definition eigener Zwischensprachendialekte
- zur high-level Darstellung spezieller Eigenschaften der zu übersetzenden Sprache
- erleichtert die Umsetzung des AST in Zwischencode
- z. B. für domänenspezifische Sprachen (DSLs)
- z. B. für bestimmte Hardware
- mehrere Abstraktionen gleichzeitig benutzbar

Der Compiler Clang

Clang: schneller C/C++/Objective-C - Compiler auf Basis von LLVM mit aussagekräftigen Fehlermeldungen und Warnungen



Die Sanitizer in der Compiler-Runtime-Bibliothek

Sanitizer: Methoden zur Instrumentierung (Code der in das kompilierte Programm eingebettet wird) zur Erleichterung der Lokalisierung und Analyse verschiedenster Fehlerquellen, z. B.:

- Speicherfehler und Speicherlecks (z. B. use-after-free)
- Race Conditions
- undefiniertes Verhalten (Overflows, Benutzung von Null-Pointern)
- Benutzung von nicht-initialisierten Variablen

Wrap-Up

Wrap-Up

LLVM ist eine (fast) komplette Infrastruktur zur Entwicklung von Compilern und compilerähnlichen Programmen.

Die wichtigsten Bestandteile:

- der Zwischencode LLVM IR
- der LLVM Optimierer
- der Codegenerator mit Sanitizern

LICENSE