Statische Code-Optimierung

mit LLVM und Polly

Seminar „Effiziente Programmierung“

11. Januar 2018

Kim Wittenburg

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Agenda

1. Einleitung
2. Verschiedene Optimierungen
3. Herausforderungen
4. LLVM und Clang
5. Das Polly-Projekt

Einleitung

- Was ist statische Optimierung?
- Algorithmische Analyse und Modifikation von Code zur Übersetzungszeit
- Ziel: Effizienterer Code
- Analyse deutlich schwieriger als Modifikation des Codes
- Style-Checker zählen wir hier nicht dazu

Constant Folding

Loop Unrolling

Tail Call Elimination

Symbolic Execution

Dead Instruction Elimination

Code Sinking

Dead Code Elimination

Constant Propagation

Function Inlining

Memory Optimization

Model Checking

Loop Rotation

Call Graph Analysis

Data Flow Optimization

Control Flow Analysis

Alias Analysis

Stack Height Reduction

Arithmetic Simplification

Dependency Analysis

Dead Store

Constant Folding

Loop Unrolling

Tail Call Elimination

Symbolic Execution

Dead Instruction Elimination

Dead Code Elimination

Code Sinking

Constant Propagation

Function Inlining

Memory Optimization

Model Checking

Loop Rotation

Call Graph Analysis

Data Flow Optimization

Control Flow Analysis

Alias Analysis

Stack Height Reduction

Arithmetic Simplification

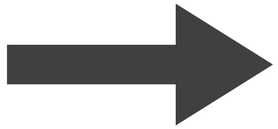
Dependency Analysis

Dead Store

Constant Folding

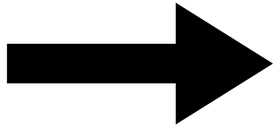
und Constant Propagation

```
int a = 2 * 50 + 17;
```



```
int a = 117;
```

```
int a = 30;
int b = 9 - (a / 5);
int c;

c = b * 4;
if (c > 10) {
    c = c - 10;
}
return c * (60 / a);
```

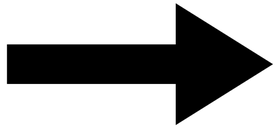


```
int a = 30;
int b = 3;
int c;

c = 12;
if (true) {
    c = 2;
}
return c * 2;
```

Dead Code Elimination

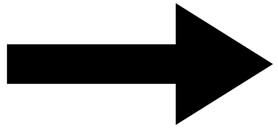
und Dead Store

```
int a = 30;
int b = 3;
int c;

c = 12;
if (true) {
    c = 2;
}
return c * 2;
```



```
return 2 * 2;
```

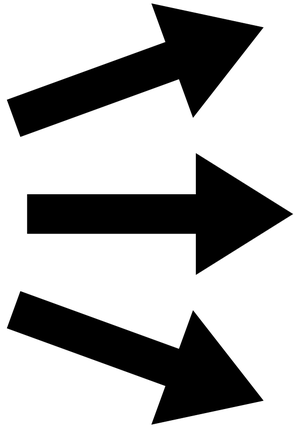
```
int foo() {
    int a = 24;
    int b = 25;
    int c;
    c = a * 4;
    return c;
    b = 24;
    bar();
    return 0;
}
```



```
int foo() {
    int a = 24;
    int c;
    c = a * 4;
    return c;
}
```

Alias Analysis

```
struct {
    int foo;
} *p, *q;
...

p->foo = 1;
q->foo = 2;
int i = p->foo + 2;
```



```
i = 3;
```

```
i = ?;
```

```
i = 4;
```

Alias Analysis ist im Allgemeinen unentscheidbar

Approximation durch Analyse des Programmflusses

Herausforderungen

```
// Sinus Function
float mySin(float x) {
    static double y = 0;
    y = y + x;
    return y;
}
```

```
float calculate(float x,
                unsigned C) {
    float sum = 0;
    for(int i = 0; i < C; ++i)
        sum += mySin(x);
    return sum;
}
```


```
float calculate(float x,
                unsigned C) {
    return C ? mySin(x) * C
             : 0.0f;
}
```

Problem: Nebeneffekte verhindern viele Optimierungen

Herausforderungen

- Nebeneffekte
- Funktionsaufrufe
- Pointer
- Reflexion
- Dynamische Sprachfeatures

LLVM und Clang

LLVM und Clang

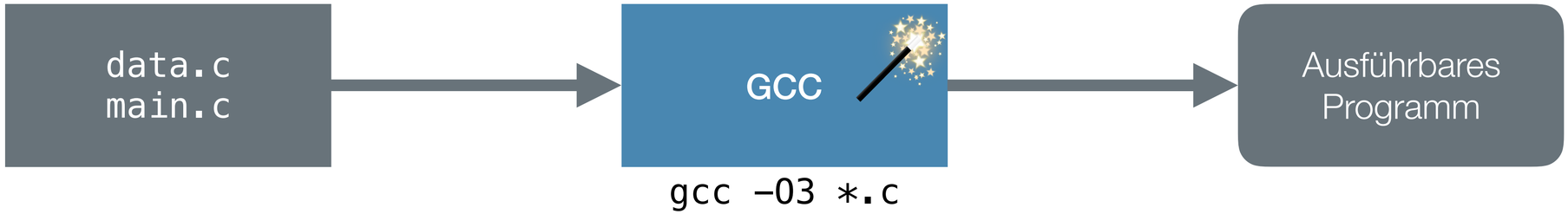


Abbildung 1: Übersetzungsphasen von GCC und LLVM (Nachzeichnung)

basiert auf [1] (Figure 11.1, Figure 11.2)

LLVM und Clang

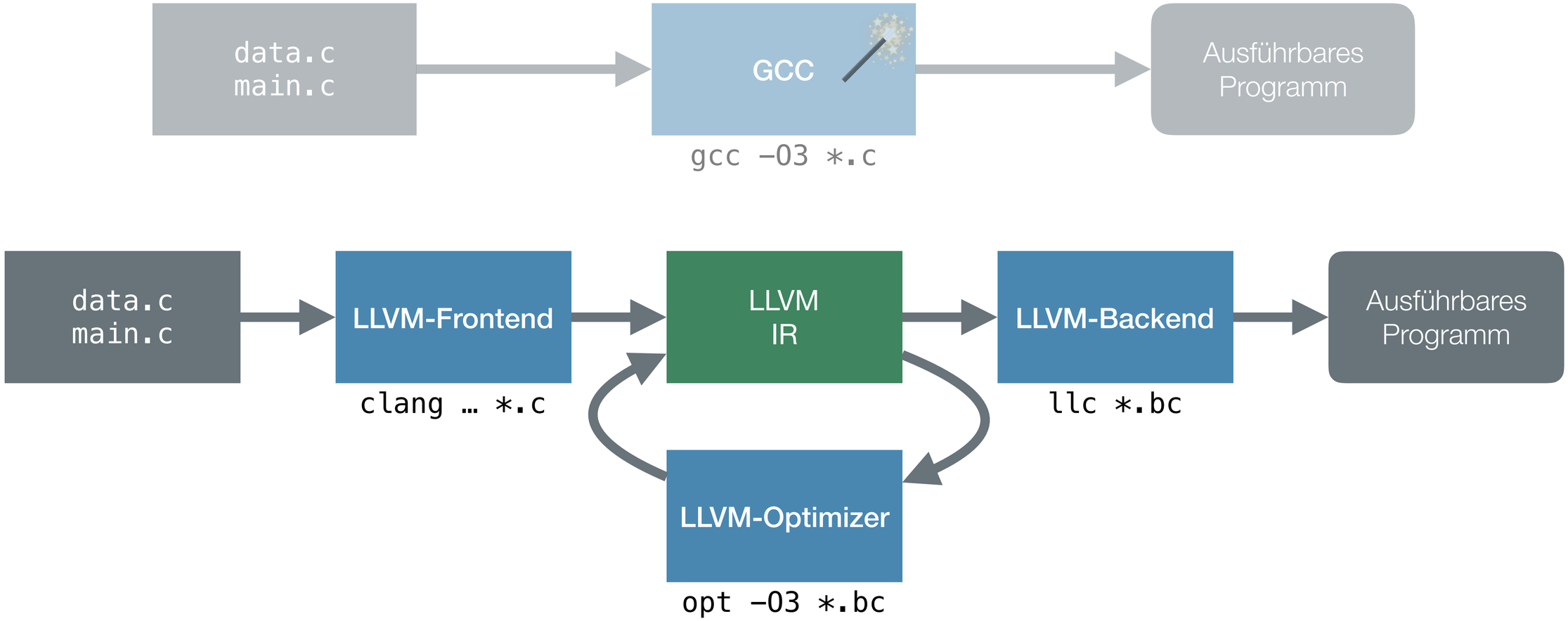


Abbildung 1: Übersetzungsphasen von GCC und LLVM (Nachzeichnung)

basiert auf [1] (Figure 11.1, Figure 11.2)

LLVM Pass Pipeline

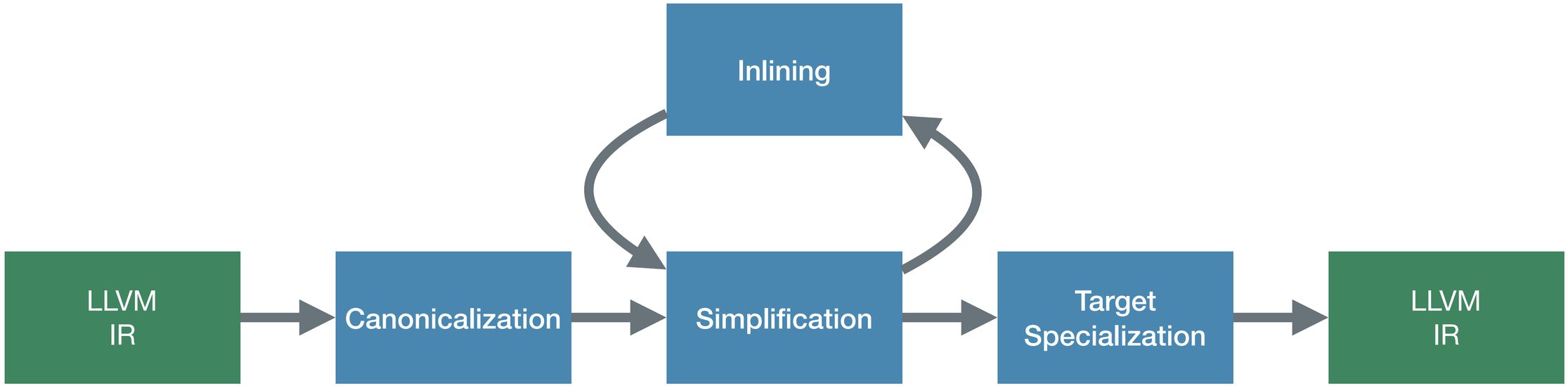


Abbildung 2: LLVM Pass Pipeline (Nachzeichnung)

basiert auf [2] (S. 25)

Demo

Callgraph-Analyse mit LLVM

Polly

Polly

- Eine Reihe von LLVM-Passes zur Optimierung von Schleifen
- Optimierung auf Laufzeit und Datenlokalität
- Polly-API für eigene Optimierungen verfügbar
- Repräsentation von Schleifen als ganzzahlige Polyeder

Polly

```
for (i = 0; i <= n; i++)
  for (j = 0; j <= i; j++)
    S(i,j);
```

$$= \{S(i,j) \mid 0 \leq i \leq n \wedge 0 \leq j \leq i\}$$

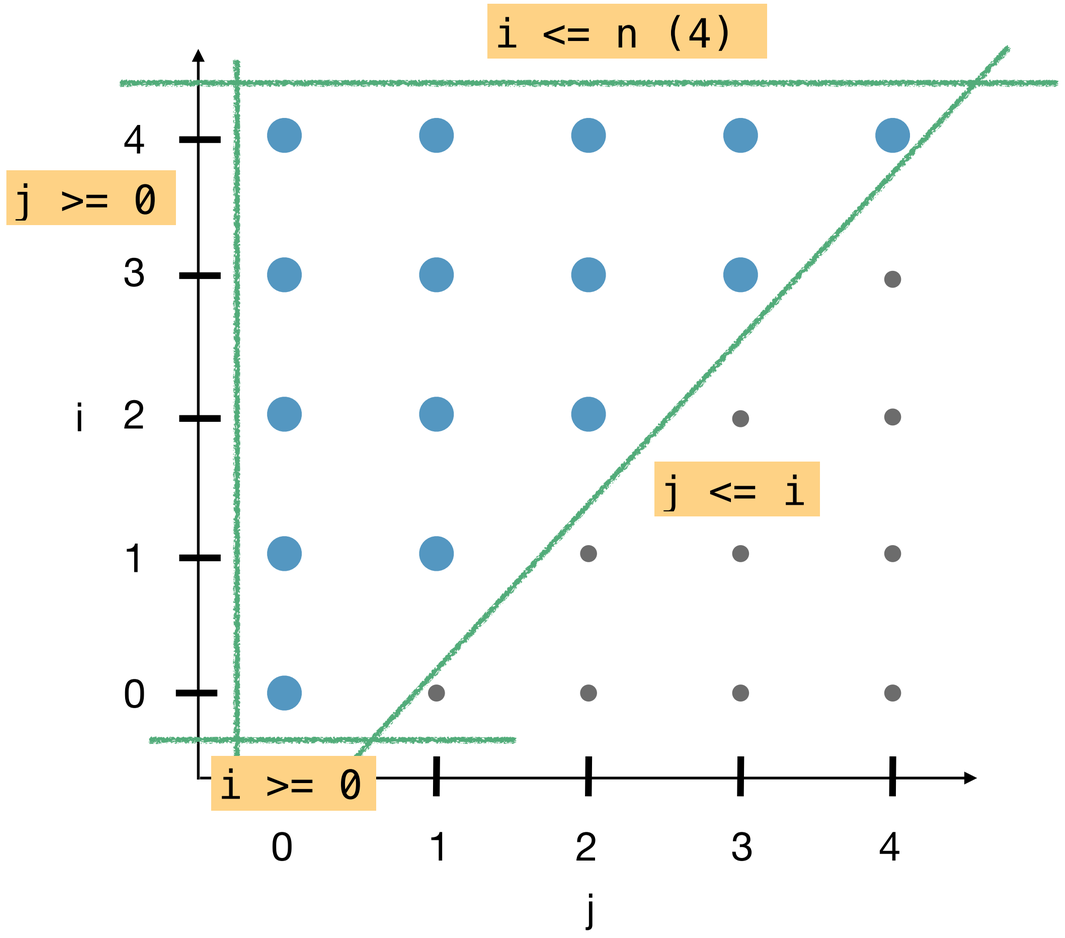


Abbildung 3: Iterationsraum (Nachzeichnung)

basiert auf [2] (S. 2)

Benutzung von Polly

Einbinden von Polly in LLVM

```
opt -O3 -polly ...
```

Beim `clang`-Frontend können `opt`-Optionen mit `-mllvm` Übergeben werden:
`clang -O3 -mllvm -polly ...`

Parallelisierung mit Polly

```
opt -O3 -polly -polly-parallel ...
```

Debugging mit Polly

```
opt -O3 -polly -polly-show ...
opt -O3 -polly -polly-view-all ...
```

Polly-Optimierung erzwingen

```
opt -O3 -polly -polly-process-unprofitable ...
```

Polly helfen

```
__builtin_assume(bool);
```

Polly in der LLVM Pass Pipeline

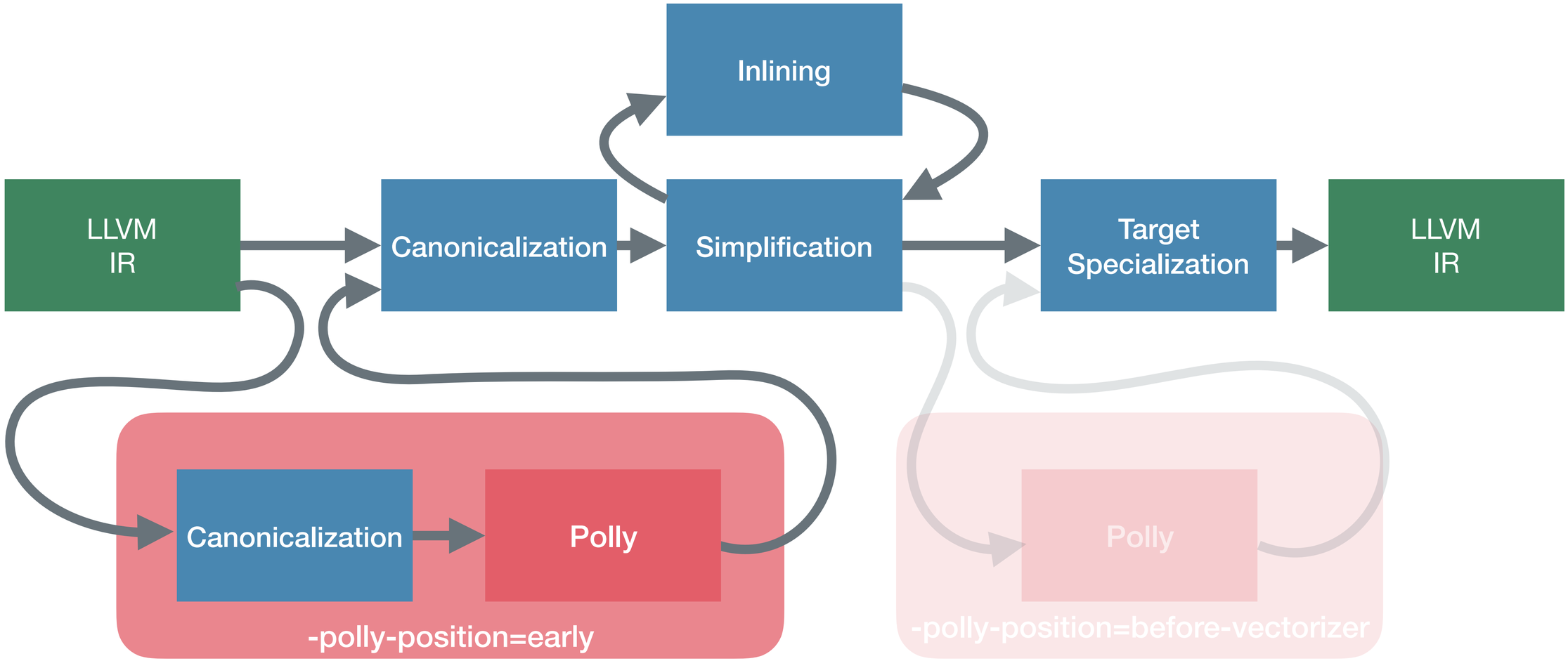


Abbildung 4: Polly in der LLVM Pass Pipeline (Nachzeichnung)

basiert auf [2] (S. 26-27)

Polly in der LLVM Pass Pipeline

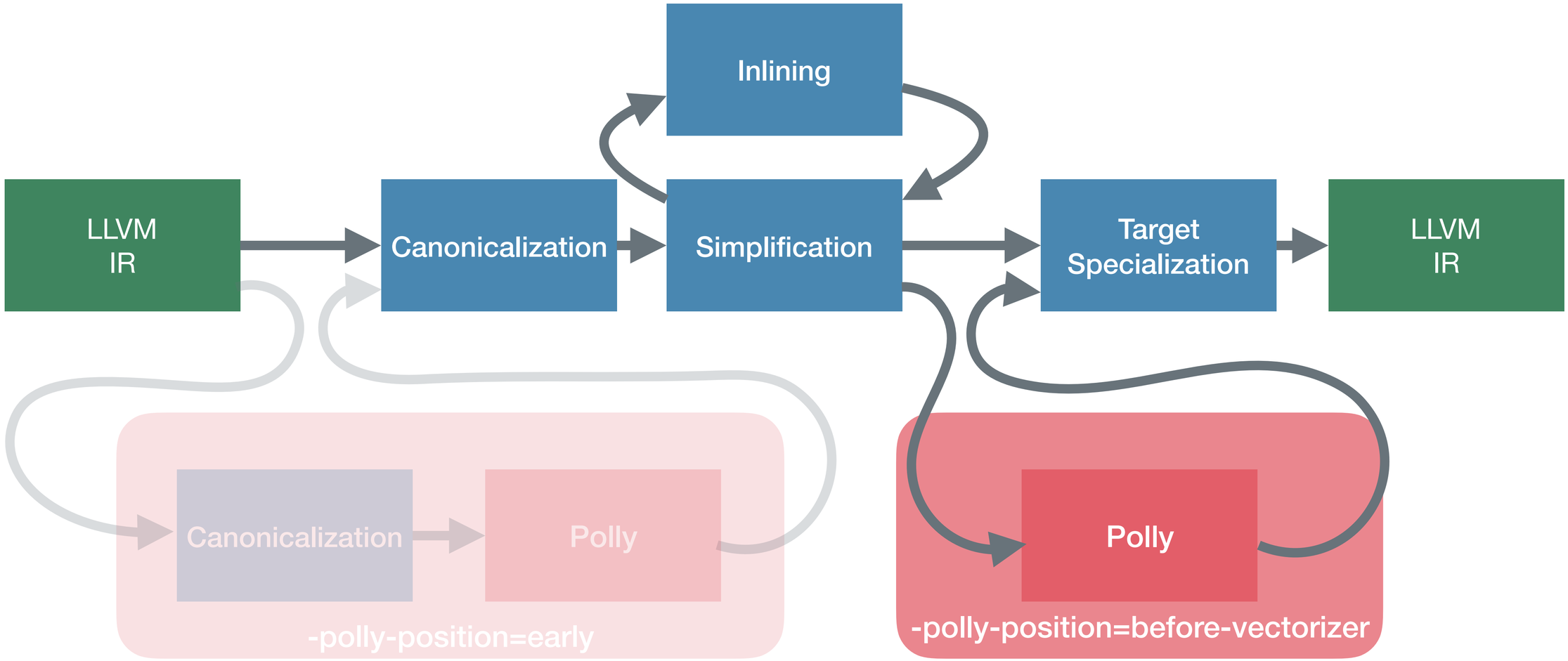


Abbildung 4: Polly in der LLVM Pass Pipeline (Nachzeichnung)

basiert auf [2] (S. 26-27)

Polly – Leistung

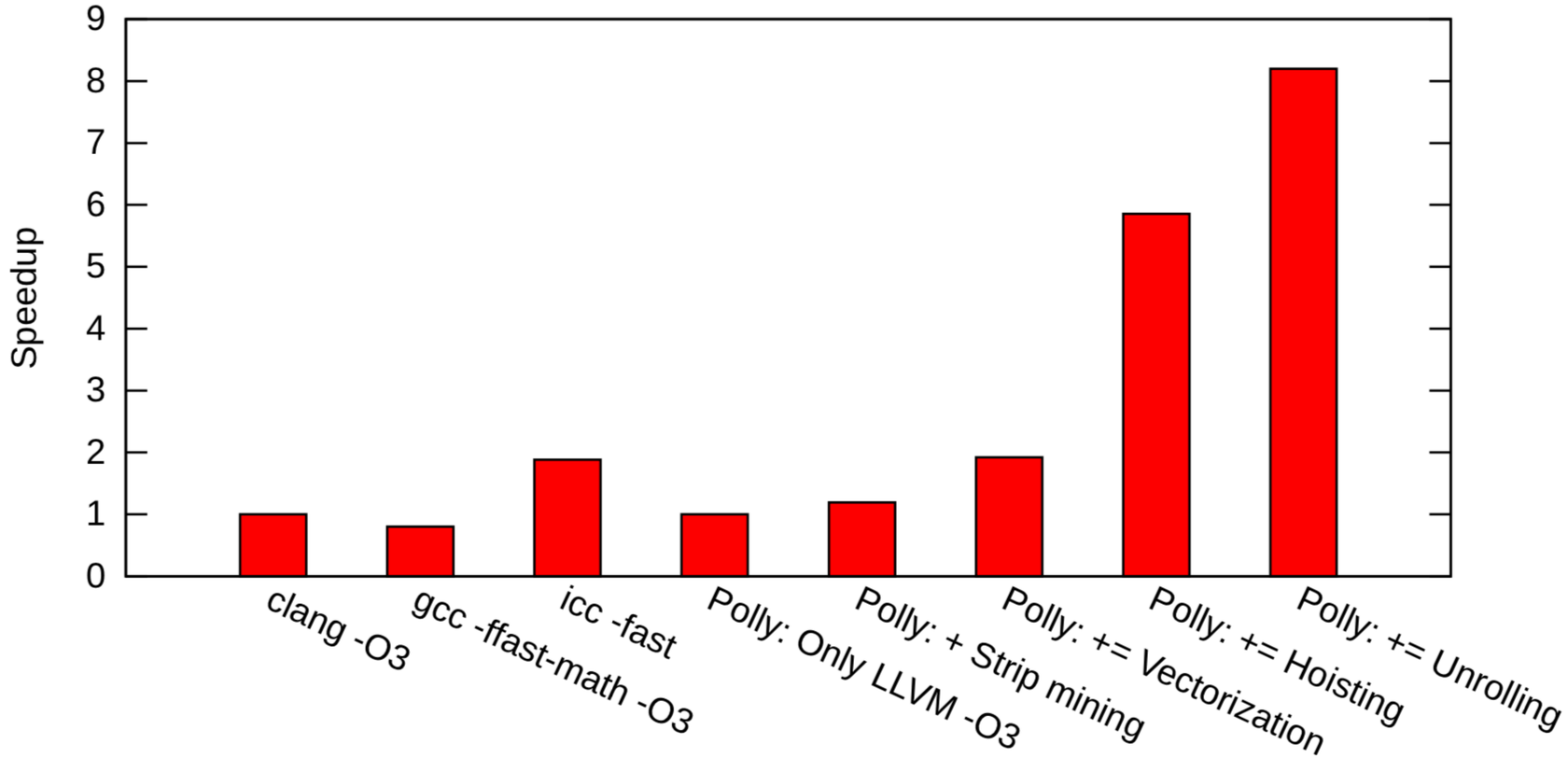


Abbildung 5: Polly-Leistungsdaten

10000 Gleitkomma-Matrixmultiplikationen (32×32). Quelle: [3] (Figure 23)

Vor- und Nachteile von Polly

- Geringer Overhead beim Kompilieren
- Bei kleinen Datensätzen kann Polly zu einem Slowdown führen
- Spezialisierte Bibliotheken für lineare Algebra (BLAS) sind noch deutlich schneller als Polly-optimierter Code.
- Automatisierte Parallelisierung mit Speedups von bis zu 100

Zusammenfassung

- Statische Optimierung: Analyse und Modifikation des Programmcode zur Übersetzungszeit.
- Vollautomatische Optimierung oft nur begrenzt möglich.
- LLVM/Clang ist ein modularer, optimierender Compiler mit vielen verschiedenen Optimierungen.
- Polly kann automatisch komplexe Schleifen optimieren und parallelisieren.

Vielen Dank

Quellen

1. Lattner, C. http://www.aosabook.org/en/llvm.html

2. Grosser, T. C.; Doerfert, J.; Benaissa, Z. *Analyzing and Optimizing your Loops with Polly*. (Präsentation, EuroLLVM 2016, Barcelona, 17. März 2016). Verfügbar unter https://www.youtube.com/watch?v=mXve_W4XU2g.

3. Grosser, T. C. (2011). *Enabling Polyhedral Optimizations in LLVM* (Diplomarbeit). Verfügbar unter https://polly.llvm.org/publications/grosser-diploma-thesis.pdf.

4. https://llvm.org/docs/Passes.html

5. https://clang.llvm.org/comparison.html

Grafiken und Codebeispiele

- Magic Wand: http://www.psdgraphics.com/psd-icons/magic-wand-icon/
- Constant Folding: https://en.wikipedia.org/wiki/Constant_folding
- Dead Code Elimination: https://en.wikipedia.org/wiki/Dead_code_elimination
- Alias Analysis: https://en.wikipedia.org/wiki/Alias_analysis

Bonusmaterial

Matrixmultiplikation mit Polly

```
void calculate(float *restrict A, float *restrict B, float *restrict C) {
    for(int i = 0; i < 32; i++)
        for (int j = 0; j < 32; j++)
            for (int k = 0; j < 32; k++)
                C[i][j] += A[k][i] * B[j][k];
}
```