

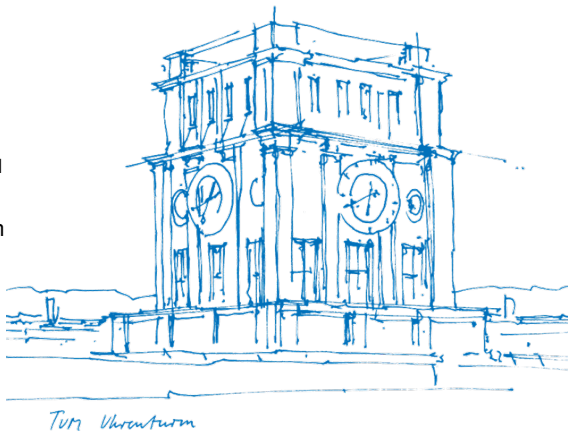
ria-jit: Dynamische Binärübersetzung RISC-V \rightarrow x86-64

Projektvorstellung

Noah Dormann¹, Simon Kammermeier¹,
Johannes Pfannschmidt¹, Florian Schmidt¹

¹Fakultät für Informatik, Technische Universität München
(TUM)

29. Januar 2021



1 Motivation

2 Technischer Überblick

3 Demo

4 Kontakt

Warum Großpraktikum?

Motivation

Warum Großpraktikum?

Arbeitsaufwand bedeutend höher als beim ERA-P.

Motivation

Warum Großpraktikum?

Arbeitsaufwand bedeutend höher als beim ERA-P. **Aber:**

Lessons learned

- tiefgreifendes Einarbeiten in ein komplexes Thema
- langfristige Arbeit an einem großen Softwareprojekt
- viel Erfahrung mit Git (~ 1000 commits)
- Aufrechterhalten einer großen Codebase (~ 19.000 LOC)
- Benchmarking und Testing mit professionellen Tools/Hardware

Motivation

Warum Großpraktikum?

Zudem:

Soft Skills

- Teambuilding und kollaborative Aufgabenverteilung
- Projektmanagement und langfristige Zeitplanung
- Wissenschaftliches Schreiben

Motivation

Warum Großpraktikum?

Sehr angenehm:

Vorteile im Großpraktikum

- **bedeutend freiere** Aufgabenstellung
- persönliche **Betreuung** und Feedback am Lehrstuhl
- *Bragging rights*

Dynamische Binärübersetzung

RISC-V \rightarrow x86-64

Technischer Überblick

Dynamische Binärübersetzung RISC-V → x86-64

x86-64

- 64-Bit Version der x86-ISA

RISC-V

- Neu entwickelte offene RISC-ISA
- 32- und 64-Bit-Varianten

Binärübersetzung

- Keine weit verbreiteten RISC-V-Prozessoren
- Für RISC-V kompilierte Programme können nicht getestet werden
- Binärübersetzung ermöglicht es RISC-V-Programme auf x86-64 auszuführen

Dynamische Binärübersetzung

- Problem: Wie in der Datei Befehle von Daten unterscheiden?
- Problem: Wie Sprungziele indirekter Sprünge bestimmen?
- Lösung: Befehle erst übersetzen, wenn benötigt.
- Direkte Übersetzung in x86-64-Instruktionen

Technischer Überblick

Dynamische Binärübersetzung RISC-V → x86-64

Vorgehen

- Blockweises Übersetzen
- Instruktionen nicht mehrfach übersetzen, bereits übersetzte Blöcke cachen
- Verwalten der Laufzeitumgebung (Register, Stack, System calls)
- Architekturunterschiede überwinden (unterschiedliche Registeranzahl, separates Zero-Register, unterschiedliche Randfälle, 3 Operanden Adressform)

Optimierungen

- Sprungziele oft vorher bekannt \implies direkt Blöcke aneinanderhängen
- Bessere Instruktionsabbildung

`./translator`

Repository: <https://github.com/ria-jit/ria-jit>

- Noah Dormann
n.dormann@tum.de
- Simon Kammermeier
simon.kammermeier@tum.de
- Johannes Pfannschmidt
johannes.pfannschmidt@tum.de
- Florian Schmidt
fs.schmidt@tum.de