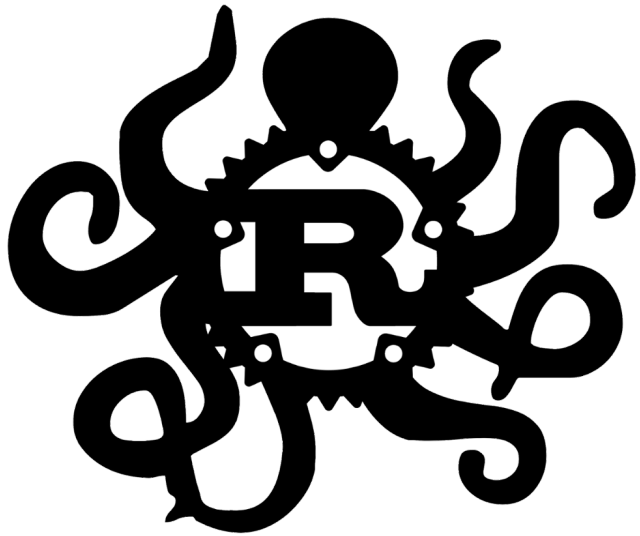
Karlsruher Institut für Technologie

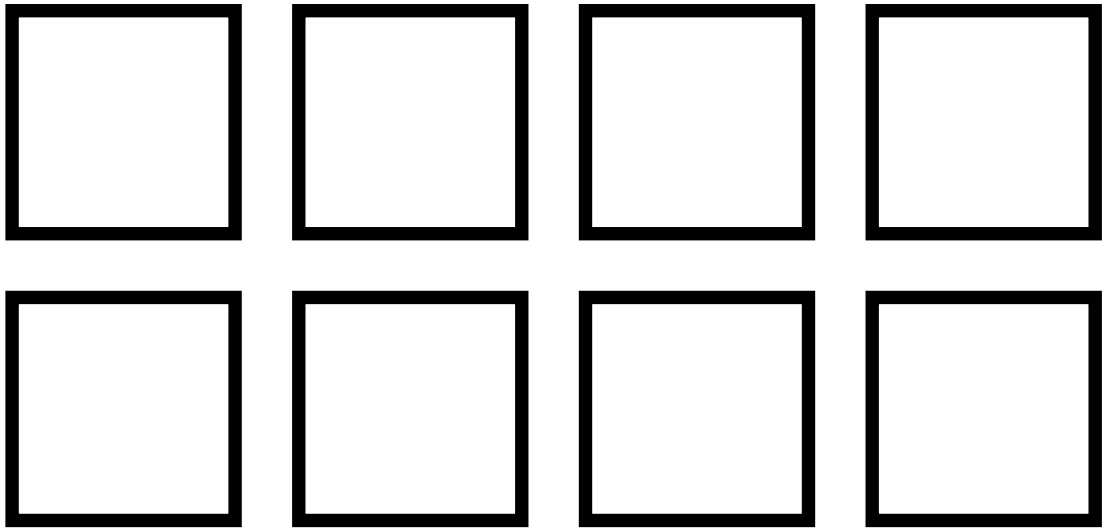
Invasives Rust

Hermann Heinz Erich Krumrey

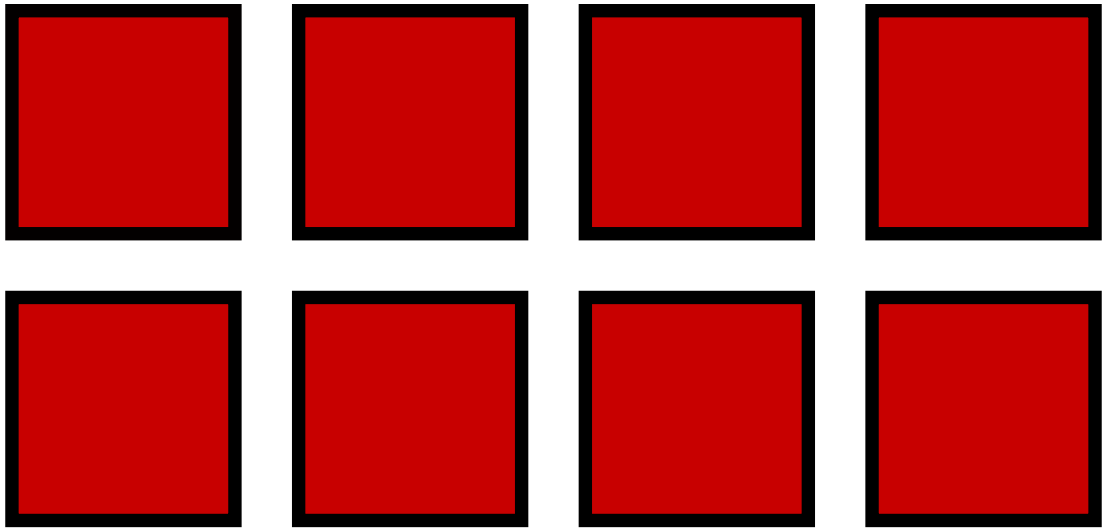
Lehrstuhl Programmierparadigmen, IPD Snelting



Motivation

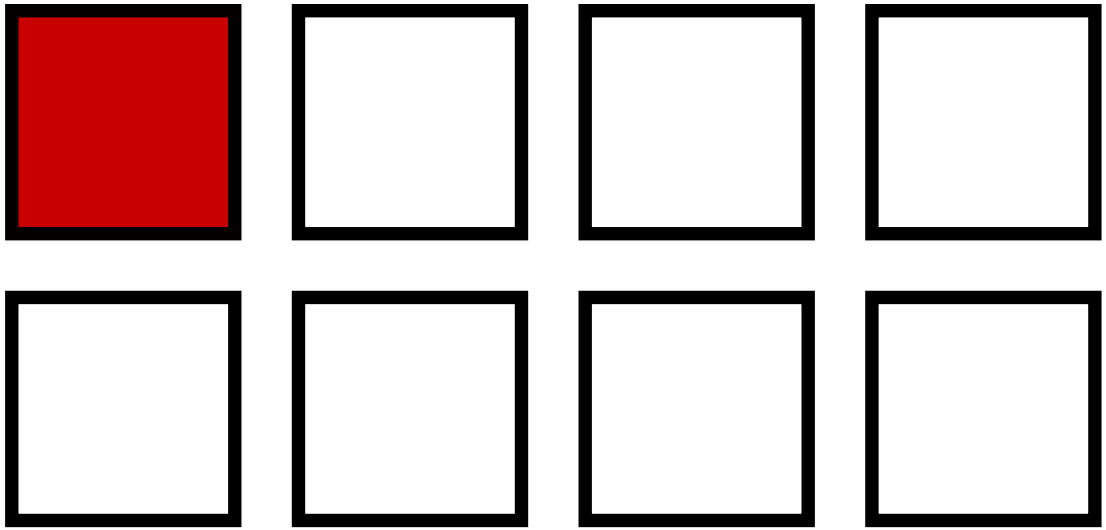


Motivation



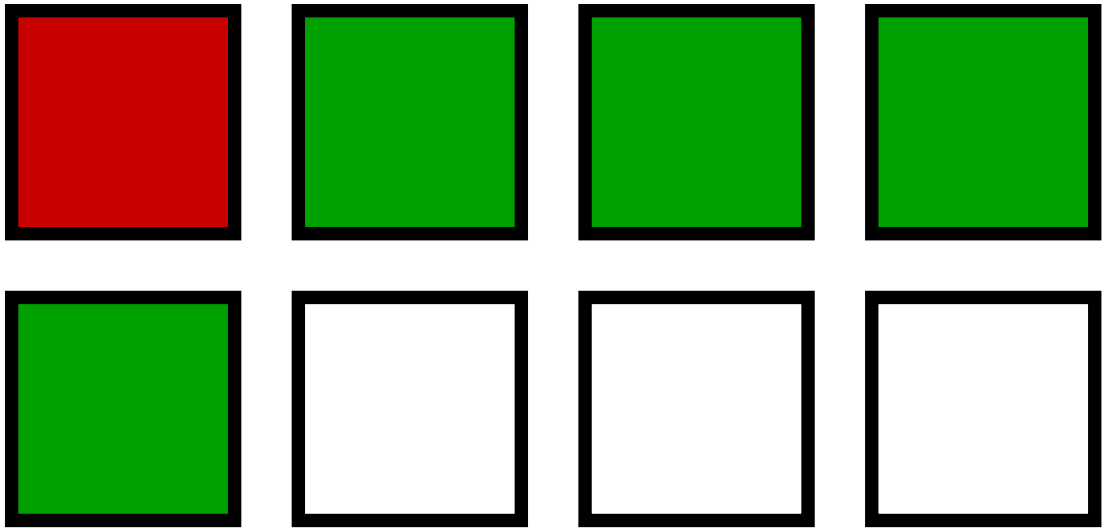
1. Programm 1 beginnt Ausführung auf 8 Recheneinheiten

Motivation



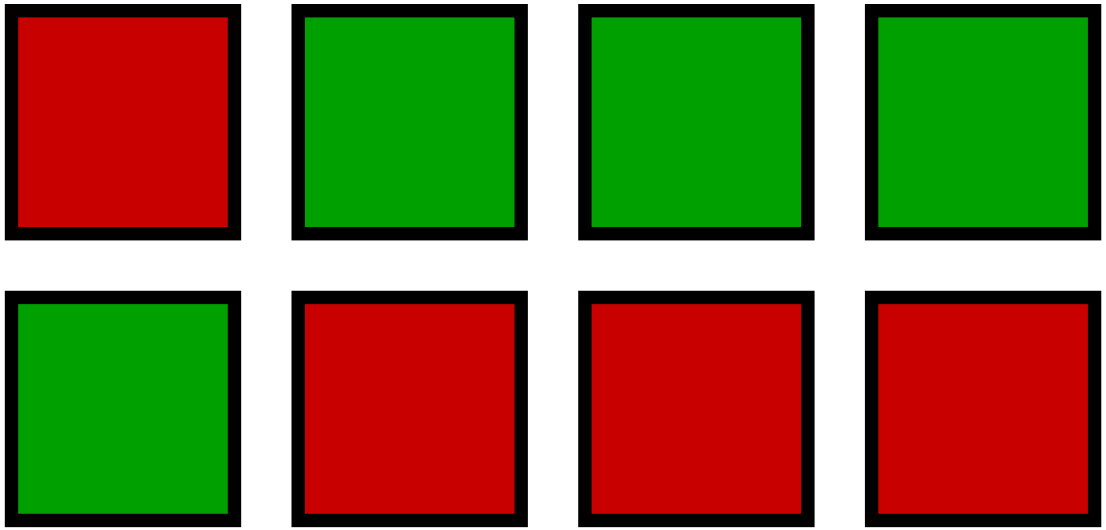
1. Programm 1 beginnt Ausführung auf 8 Recheneinheiten
2. Programm 1 sendet Ergebnisse über das Netzwerk

Motivation



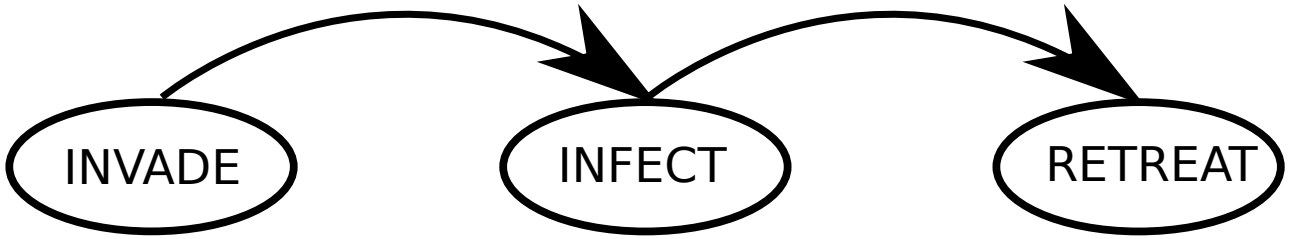
1. Programm 1 beginnt Ausführung auf 8 Recheneinheiten
2. Programm 1 sendet Ergebnisse über das Netzwerk
3. Programm 2 beginnt Ausführung auf 4 Recheneinheiten

Motivation



1. Programm 1 beginnt Ausführung auf 8 Recheneinheiten
2. Programm 1 sendet Ergebnisse über das Netzwerk
3. Programm 2 beginnt Ausführung auf 4 Recheneinheiten
4. Programm 1 führt wieder Berechnungen aus, jetzt auf 4 Recheneinheiten

Invasives Rechnen



- Ressourcenbewusstes Programmieren
- 3 Phasen:
  1. Invade - Ressourcen reservieren
  2. Infect - Ressourcen nutzen
  3. Retreat - Ressourcen freigeben
- OctoPOS und iRTSS bieten Software-Grundlage
- Angepasste Hardware um invasive Grundfunktionen zu unterstützen

SPARC LEON

- SPARC
  - Skalierbare Prozessorarchitektur
- SPARC-V8
  - Basierend auf SPARC
  - 32-bit Architektur
- LEON
  - SPARC-V8 basierte Prozessorfamilie
  - Konfigurierbare, Open-Source VHDL Designs
  - Geeignet für den Einsatz in angepassten ASICs, SOCS und FPGAs

Rust - Motivation

- Sichere Speicherzugriffe
- Effiziente und sicherere Parallelberechnung
- Konzeptionell an C-artige Sprachen angelehnt
- Höhere Abstraktionen, um den Einstieg zu erleichtern
- Speichersicherheit und Abstraktionen sollen nicht auf Kosten der Leistung erreicht werden

Rust - Ownership, Move-Semantik und Referenzen

- Ownership
  - Das zentrale Alleinstellungsmerkmal der Programmiersprache
  - Jeder Speicherbereich wird nur einer einzigen Variable zur Verfügung gestellt
  - Beim Verlassen des Geltungsbereichs wird der Speicherbereich freigegeben
- Move-Semantik
  - Ownership kann auf andere Variablen übertragen werden
  - Ursprüngliche Variable ist nach einem Move nicht mehr verwendbar
- Referenzen
  - Unendliche unveränderliche Referenzen
  - Nur eine veränderliche Referenz
  - Move einer Variable nicht möglich wenn Referenzen existieren

Rust - Ownership - Beispiel

Karlsruher Institut für Technologie

Owner: a

```
...
let a = String::from("Hello World!");



...
```

Rust - Ownership - Beispiel

Owner: a

```
...
let a = String::from("Hello World!");
println!(a);


...
```

```
Hello World!
```

Rust - Ownership - Beispiel

Owner: a

```
Hello World!
```

```
...
let a = String::from("Hello␣World!");
println!(a);
{ let b = String::from("Hallo␣Welt!"); }

...
```

Rust - Ownership - Beispiel

Karlsruher Institut für Technologie

Owner: a

```
...
let a = String::from("Hello␣World!");
println!(a);
{ let b = String::from("Hallo␣Welt!"); }
println!(b);
...
```

```
error[E0425]:
cannot find value ‘b‘ in this scope
```

Rust - Move-Semantik - Beispiel

Owner: a

```
...
let a = String::from("Hello World!");


...
```

Rust - Move-Semantik - Beispiel

Owner: b

```
...
let a = String::from("Hello␣World!");
let b = a;

...
```

Rust - Move-Semantik - Beispiel

Owner: b

```
...
let a = String::from("Hello␣World!");
let b = a;
println!("{}", a);
...
```

```
error[E0382]:
use of moved value: ‘a‘
```

Rust - Traits

- Wie Interfaces in anderen Sprachen
- Können Verhalten bezüglich Ownership und Move-Semantik beeinflussen
- Drop
  - Destruktor
  - drop()-Methode wird beim verlassen des Geltungsbereichs aufgerufen
- Copy
  - Erlaubt Copy-Semantik statt Move-Semantik
  - Jedes mal, wenn ein Move geschehen würde, stattdessen bitweise Kopie

Rust - SPARC

Karlsruher Institut für Technologie

Evtl rauslassen.

- nostd
- JSON
- libcore
- Cargo

octorust

- Hilfsprogramm zum Kompilieren von invasiven Rust-Programmen
- Kompilierung von **.rs**-Dateien und Cargo-Projekten
- Unterstützung von SPARC-V8
- Automatische Verlinkung mit iRTSS/OctoPOS
- ca. 650 Zeilen Code (Python)

octolib

- Direkte C-Rust Bindings
- Rust-spezifische Änderungen
- ca. 750 Zeilen Code (Rust)

octolib - Rust Bindings

- Beispiel C Funktion zu Rust Funktion

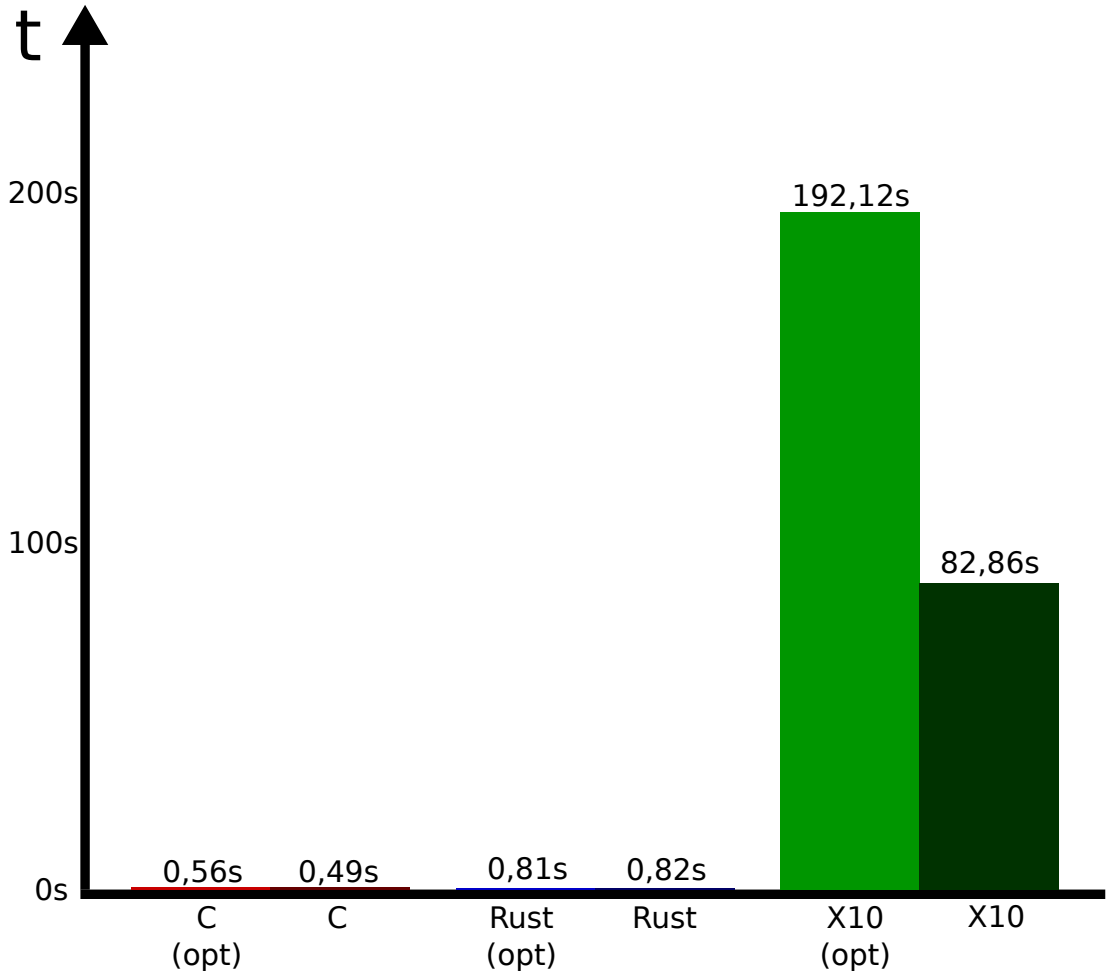
octolib - Rust Improvements

- AgentClaim
- Closures

Evaluation

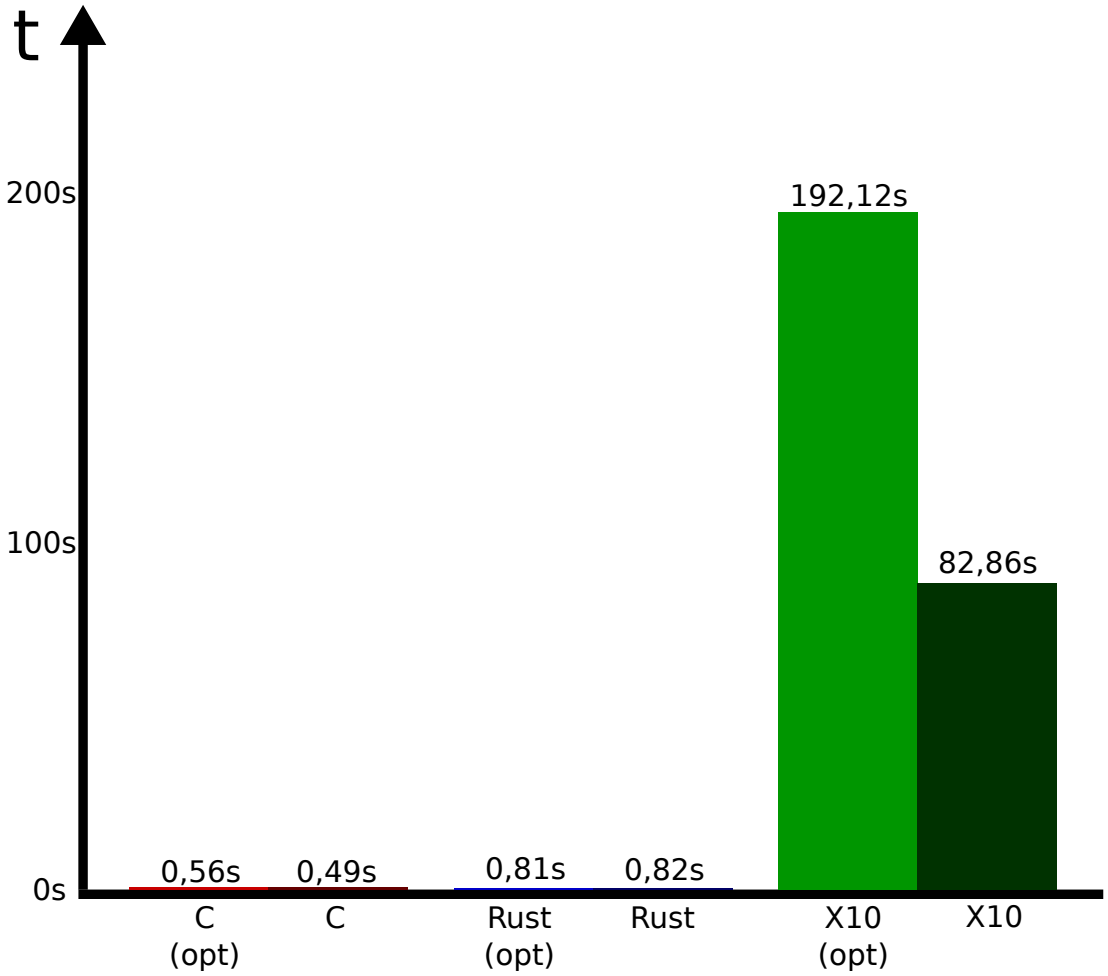
- temci
- 50-mal
- Intel i5...
- Benchmarks
  - Kompilierungsdauer
  - Anlaufzeit
  - Parallele Primzahlenberechnung
  - Allokation von Objekten

Kompilierungsdauer



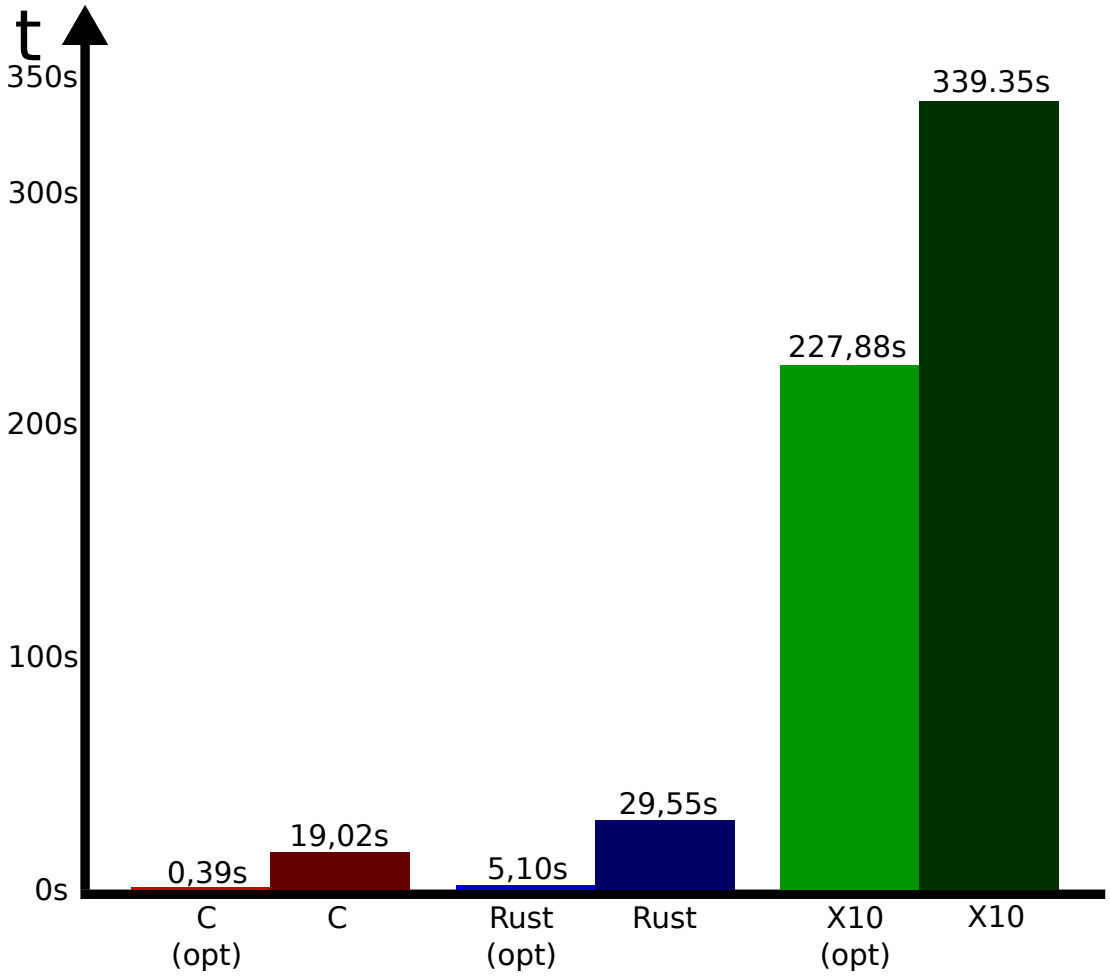
- X10 langsam

Anlaufzeit



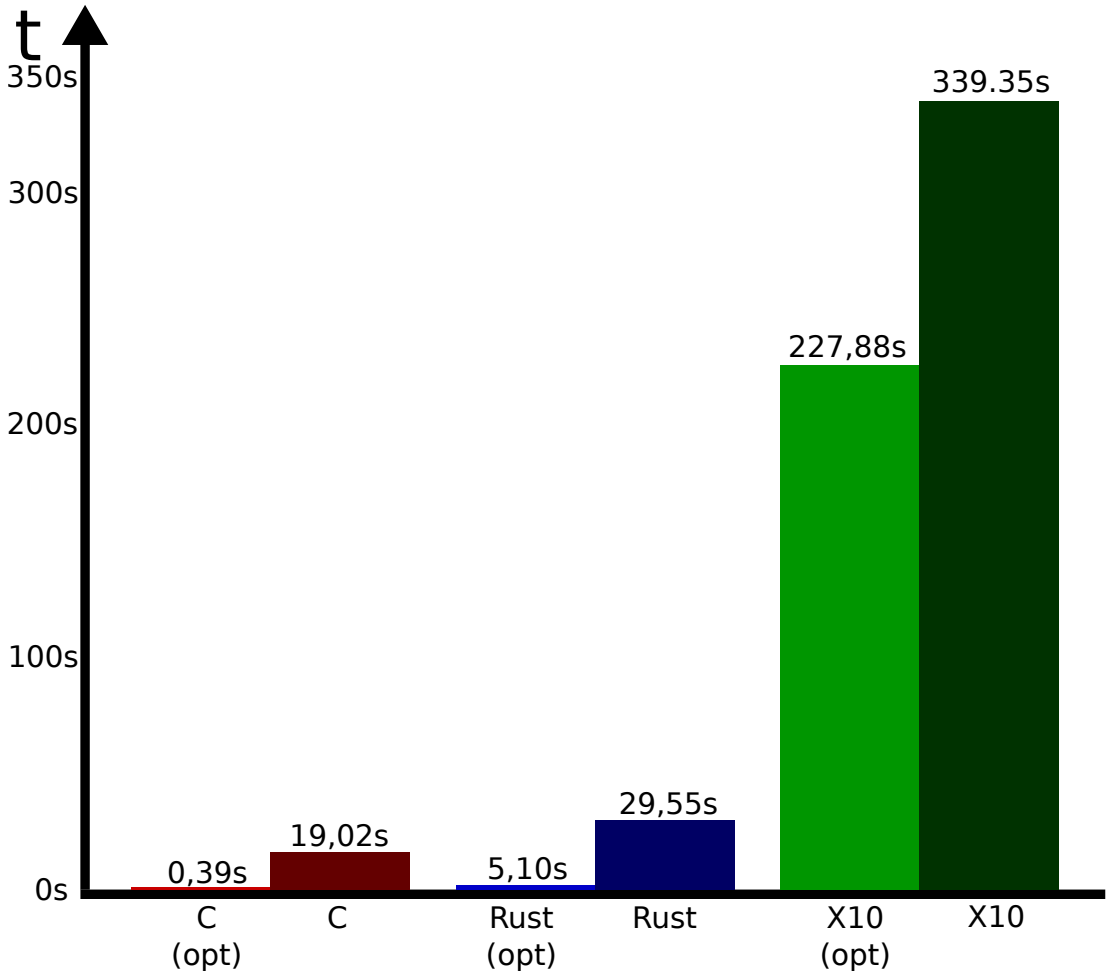
- X10 langsam

Parallele Primzahlenberechnung



- Rust langsam

Allokation von Objekten



- X10 langsam

Zusammenfassung und Fazit

Vielen Dank für ihre Aufmerksamkeit!

Karlsruher Institut für Technologie

- 4 Wichtigste Folien:
  - Invasives Rechnen
  - Ownership, Move-Semantik, Referenzen
  - Traits
  - octolib