```
const size_t N = 1024;
int32_t values[N];

TASK(τ_sort){
  ...
  for(i = 0; i < N-1; i++){
    for(j = 0; j < N-1-i; j++){
      if(compare(values[j], values[j+1]))
        swap(&values[j], &values[j+1]);

    }
  }
}
```

```
const size_t N = 1024;
int32_t values[N];

TASK(τ_sort){
  ...
  for(i = 0; i < N-1; i++){
    for(j = 0; j < N-1-i; j++){
      if(compare(values[j], values[j+1]))
        swap(&values[j], &values[j+1]);

    }
  }
}
```

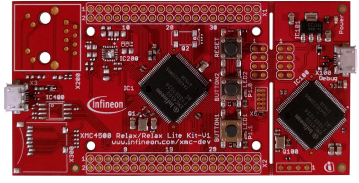
- Verschachtelte Schleifen

```
const size_t N = 1024;
int32_t values[N];

TASK(τ_sort){
  ...
  for(i = 0; i < N-1; i++){
    for(j = 0; j < N-1-i; j++){
      if(compare(values[j], values[j+1]))
        swap(&values[j], &values[j+1]);

    }
  }
}
```

- Verschachtelte Schleifen
- Eingabeabhängige Pfade

```
const size_t N = 1024;
int32_t values[N];

TASK(τ_sort){
  ...
  for(i = 0; i < N-1; i++){
    for(j = 0; j < N-1-i; j++){
      if(compare(values[j], values[j+1]))
        swap(&values[j], &values[j+1]);

    }
  }
}
```

- Verschachtelte Schleifen
- Eingabeabhängige Pfade
- Hardware-Modellierung



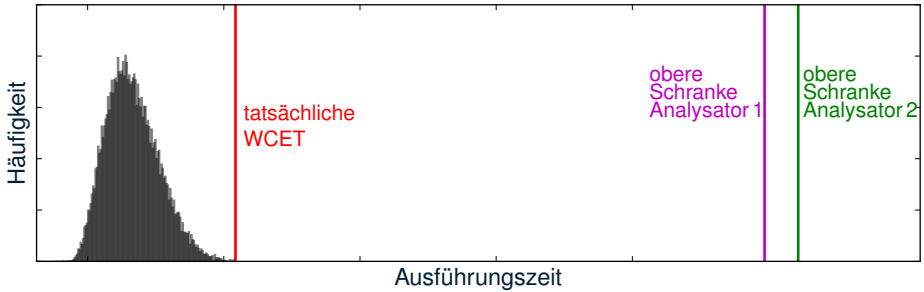
Fehlendes Vergleichsmaß

Fehlendes Vergleichsmaß

Fehlendes Vergleichsmaß

- Bestimmung Genauigkeit der Analyse: **tatsächlicher Worst-Case** essenziell

Fehlendes Vergleichsmaß



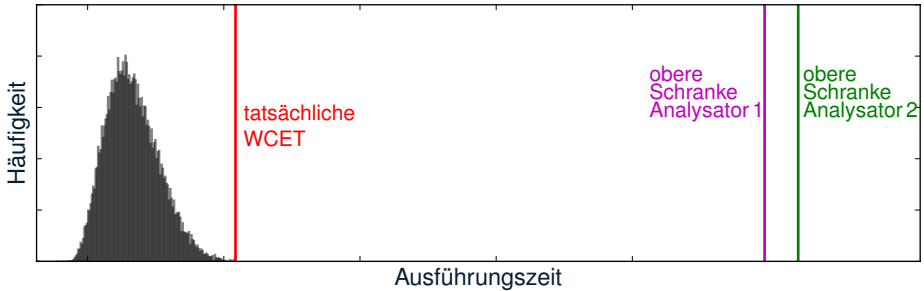
- Bestimmung Genauigkeit der Analyse: **tatsächlicher Worst-Case** essenziell
- **Existierende Benchmarks**: automatische Ermittlung des Worst-Cases nicht möglich

Fehlendes Vergleichsmaß

- Bestimmung Genauigkeit der Analyse: **tatsächlicher Worst-Case** essenziell
- **Existierende Benchmarks**: automatische Ermittlung des Worst-Cases nicht möglich

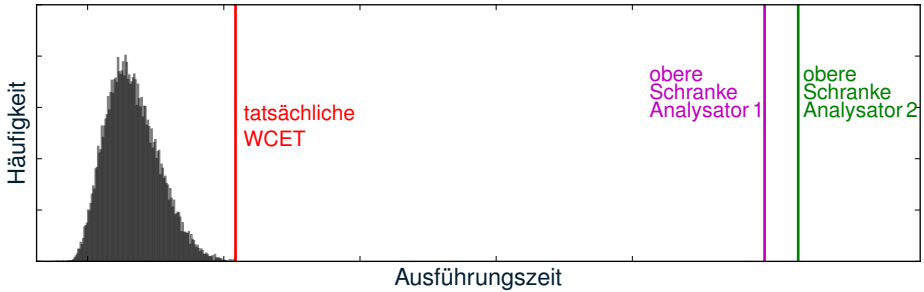
☞ **Evaluation & Validierung**: Wissen über Worst-Case notwendig

Fehlendes Vergleichsmaß



Wie lässt sich der tatsächliche Worst-Case ermitteln?

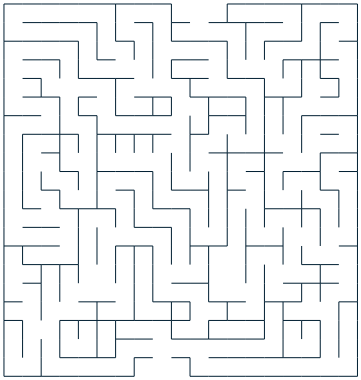
Fehlendes Vergleichsmaß

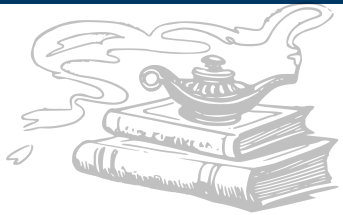


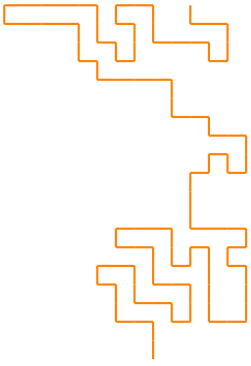
Wie lässt sich der tatsächliche Worst-Case ermitteln?

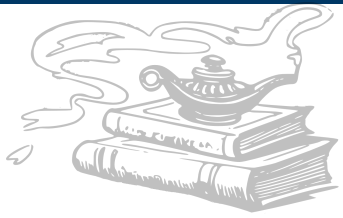
☞ *GenE*: **generiert Benchmarks**, deren Programmfakten bekannt sind

Suche nach Pfad durch Labyrinth

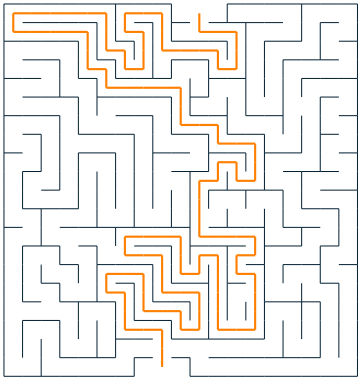


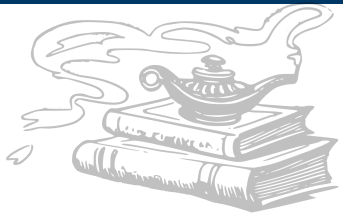


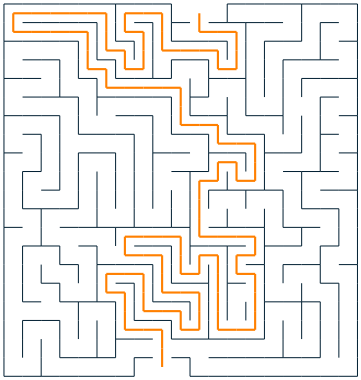


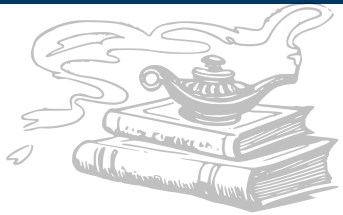


Suche nach Pfad durch Labyrinth



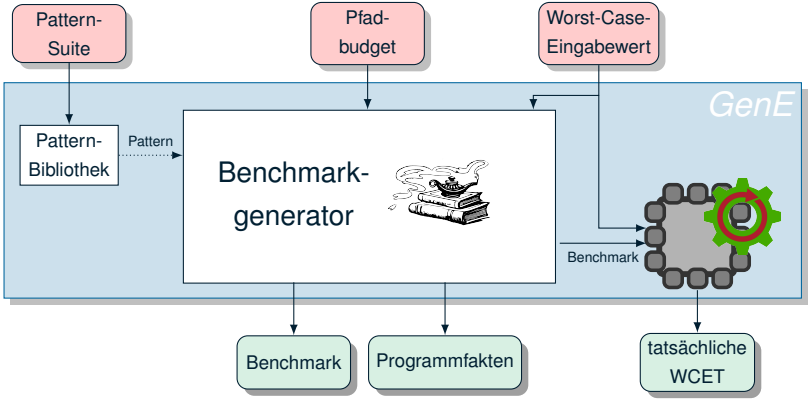




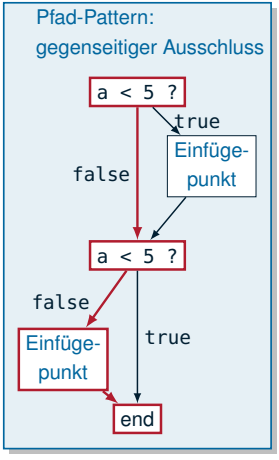


☞ Generativer Ansatz: **Worst-Case-Pfad bekannt**

Übersicht: *GenE*



Pattern

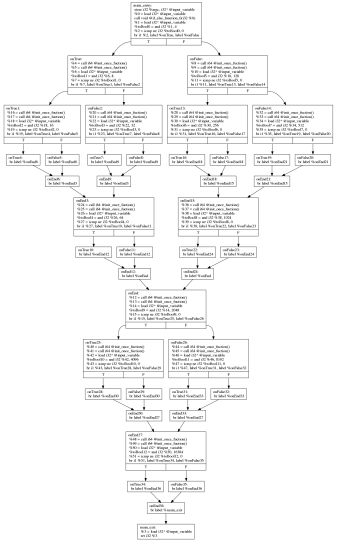


Pattern

- **Enthalten Programmfakten**
- **Kombinierbar**: ineinander verwoben
- Pfad-Pattern, Schleifen, arithmetische Operationen, Variablen, Funktionsaufrufe, ...
- **Realistisch** & **herausfordernd**
  - Industrie-Anwendungen
  - Literatur
- **Pattern-Suites** verhindern monolithische Benchmarks

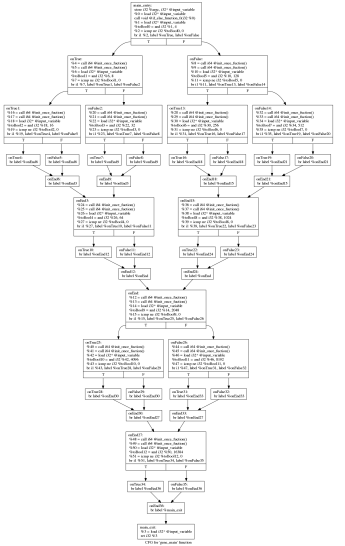
```
store volatile i32 %arith8619, i32* @cons0
%10 = load i32* @gene_glob_input_variable
ret i32 %10

br434.then.wcp: ; preds = %br433.then.wcp
call void @iofunc71()
%arith7873 = or i32 %4, 1239184893
%arith7874 = sdiv i32 %arith7873, #779274285
store volatile i32 %arith7874, i32* @cons0
store volatile i32 %arith7041, i32* @cons0
%arith7875 = sdiv i32 %arith7873, 477723546
%arith7876 = add i32 %arith7875, 212494280
call void @iofunc82()
%11 = load i32* @gene_glob_input_variable
%12 = and i32 %11, 4096
%13 = icmp ne i32 %12, 0
br i1 %13, label %br489.then.wcp, label %br489.else

br434.else: ; preds = %br433.then.wcp
%arith8206 = mul i32 %arith8205, 2039652656
%14 = load i32* @gene_glob_input_variable
%15 = and i32 %14, 64
%16 = icmp ne i32 %15, 0
br i1 %16, label %br491.then, label %br491.else

br434.end.wcp: ; preds = %br491.end, %br490.end.wcp
%arith8222 = sdiv i32 %arith7038, 2108165700
call void @iofunc87()
%arith8416 = sdiv i32 %arith8222, 1909311980
%17 = load i32* @gene_glob_input_variable
%18 = and i32 %17, 16
%19 = icmp ne i32 %18, 0
br i1 %19, label %br500.then.wcp, label %br500.else

br489.then.wcp: ; preds = %br434.then.wcp
%arith8151 = or i32 %arith7876, 110525940
store volatile i32 %arith8151, i32* @cons0
%arith8153 = xor i32 %arith8152, 938560787
store volatile i32 %arith8160, i32* @cons0
%arith8163 = sdiv i32 %arith8162, 940335150
br label %br489.end.wcp
```

- Vermeidung von **monolithischen Benchmarks**
- Beachtung der Analysephase: **Pfad-** & **Hardware-Analyse**
- Benchmarks mit **spezifischen Eigenschaften**
  1. Arithmetische Operationen, Zuweisungen, Verzweigungen:
     ☞ `GenE --suite value_analysis`

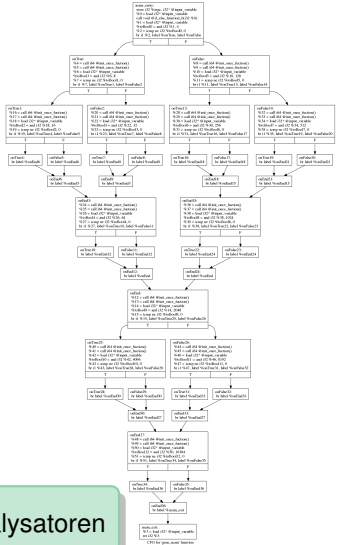


- Vermeidung von **monolithischen Benchmarks**
- Beachtung der Analysephase: **Pfad-** & **Hardware-Analyse**
- Benchmarks mit **spezifischen Eigenschaften**
  1. Arithmetische Operationen, Zuweisungen, Verzweigungen:
     ☞ `GenE --suite value_analysis`
  2. Kein Pessimismus durch Schleifen & Pfade:
     ☞ `GenE --suite single_path`

     ...
- Benchmarks **maßgeschneidert für Analyseszenario**



- Vermeidung von **monolithischen Benchmarks**
- Beachtung der Analysephase: **Pfad-** & **Hardware-Analyse**
- Benchmarks mit **spezifischen Eigenschaften**
  1. Arithmetische Operationen, Zuweisungen, Verzweigungen:
     ☞ `GenE --suite value_analysis`
  2. Kein Pessimismus durch Schleifen & Pfade:
     ☞ `GenE --suite single_path`
     ...
- Benchmarks **maßgeschneidert für Analyseszenario**



✓ Bestimmung **Stärken & Schwächen** von Analysatoren

Pfadbudget

- Anzahl von **Instruktionen entlang Worst-Case-Pfad** (z.B. 10 000)

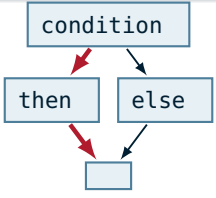
- Anzahl von **Instruktionen entlang Worst-Case-Pfad** (z.B. 10 000)

Problem: variierende Instruktionslaufzeiten

1. Caches
2. Pipelining

...

☞ *Notwendigkeit:* Modellierung **Hardwareverhalten**



- Anzahl von **Instruktionen entlang Worst-Case-Pfad** (z.B. 10 000)

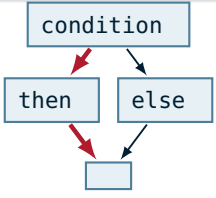
Problem: variierende Instruktionslaufzeiten

1. Caches
2. Pipelining

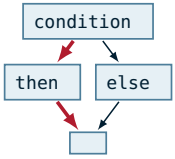
...

☞ *Notwendigkeit:* Modellierung **Hardwareverhalten**

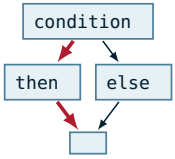
- Ziele
  1. Keine Mikroarchitekturmodellierung während Benchmark-Generierung
  2. Worst-Case-Eingabewert führt zu Worst-Case-Pfad



- *Anforderung an GenE*:
  Worst-Case(nicht-Worst-Case-Pfade) $\leq$ *Best-Case(Worst-Case-Pfad)*
- Laufzeit von Instruktion:
  $t_{\texttt{instr}} = t_{\text{icache}} + t_{\text{dcache}} + t_{\text{pipeline}} + t_{\text{calc}}$



- *Anforderung an GenE*:
  Worst-Case(nicht-Worst-Case-Pfade) ≤ Best-Case(Worst-Case-Pfad)
- Laufzeit von Instruktion:
  $t_{\texttt{instr}} = t_{\text{icache}} + t_{\text{dcache}} + t_{\text{pipeline}} + t_{\text{calc}}$



Lösung: maximale Laufzeitunterschiede

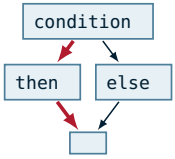
- Finde **Maxima der Worst-Case-Laufzeiten** aller Instruktionen
- Beispiel (ARM Cortex-M4)

```
add r4, r5
```
$t_{\texttt{add},\text{min}} = 1$

```
ldr r0, [r0, 42]
```
$t_{\texttt{ldr},\text{max}} = 7$

  - 7 x `add` kompensieren 1 x `ldr` im Worst-Case
  - ☞ Übergewichtungsfaktor: $\mathscr{F} = 7$

- *Anforderung an GenE*:
  Worst-Case(nicht-Worst-Case-Pfade) $\leq$ *Best-Case(Worst-Case-Pfad)*
- Laufzeit von Instruktion:
  $t_{\texttt{instr}} = t_{\text{icache}} + t_{\text{dcache}} + t_{\text{pipeline}} + t_{\text{calc}}$



Lösung: maximale Laufzeitunterschiede

- Finde **Maxima der Worst-Case-Laufzeiten** aller Instruktionen
- Beispiel (ARM Cortex-M4)

```
add r4, r5
```
$t_{\texttt{add},\text{min}} = 1$

```
ldr r0, [r0, 42]
```
$t_{\texttt{ldr},\text{max}} = 7$

  - 7 x `add` kompensieren 1 x `ldr` im Worst-Case
  - ☞ Übergewichtungsfaktor: $\mathscr{F} = 7$

☞ Wähle $\mathscr{F}$ **ausreichend groß**: $\mathscr{F} = 25$

Beispiel: Benchmarkgenerierung

Worst-Case-Eingabewert: $\cdots 01$

Übergewichtungsfaktor: $\mathscr{F} = 25$

Beispiel: Benchmarkgenerierung

Worst-Case-Eingabewert: $\cdot\,\cdot\,|\,0\,|\,1\,|$

Budget: 1010

Übergewichtungsfaktor: $\mathscr{F} = 25$

Beispiel: Benchmarkgenerierung

Worst-Case-Eingabewert: 

Übergewichtungsfaktor: $\mathscr{F} = 25$

Budget: 1010

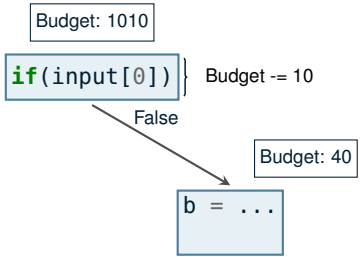
```
if(input[0])
```

Budget -= 10

Beispiel: Benchmarkgenerierung

Worst-Case-Eingabewert: | · · | 0 | 1 |

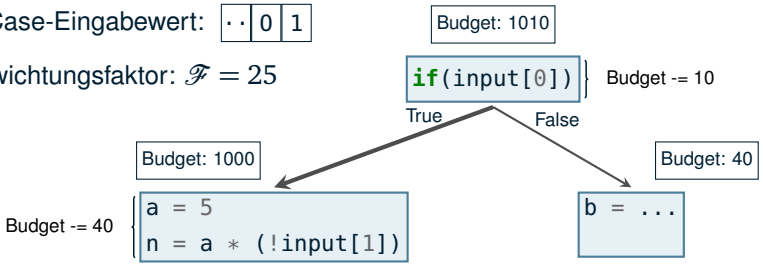
Übergewichtungsfaktor: $\mathscr{F} = 25$



Beispiel: Benchmarkgenerierung

Worst-Case-Eingabewert: `· · 0 1`

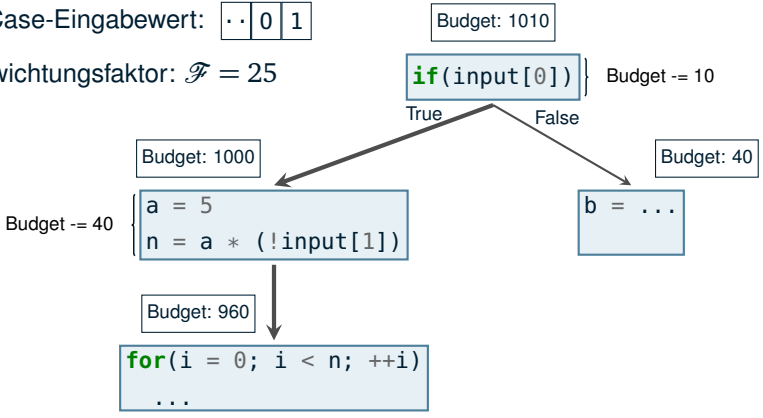
Übergewichtungsfaktor: $\mathscr{F} = 25$



Beispiel: Benchmarkgenerierung

Worst-Case-Eingabewert: `··` | `0` | `1`

Übergewichtungsfaktor: $\mathscr{F} = 25$



Evaluation: Stärken & Schwächen

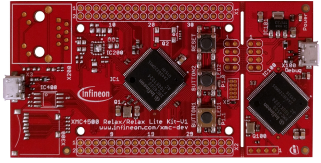
	constant loop	input-dependent loop	down-sampling loop	triangular loop
aiT	✓	✓	✓	✗
Platin	✓	✓	✗	✓

- Modulare Struktur von Pattern-Suites ⇝ verhindert monolithische Benchmarks
- Benchmarks mit **einem herausfordernden Pattern**
  - Variable, arithmetische Operationen, Zuweisungen
  - Herausforderndes Schleifen-Pattern

✓ *GenE* **deckt Stärken & Schwächen auf**

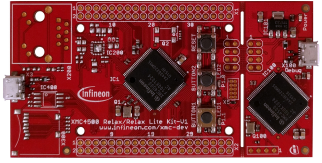
Evaluation der WCET-Analysatoren `Platin` & `aiT`

- WCET-Analysator: `aiT` von AbsInt
- Hardware: ARM Cortex-M4
- *GenE*: 10 000 generierte Benchmarks
- Optimum: 0% Überabschätzung



Evaluation der WCET-Analysatoren `Platin` & `aiT`

- WCET-Analysator: `aiT` von AbsInt
- Hardware: ARM Cortex-M4
- *GenE*: 10 000 generierte Benchmarks
- Optimum: 0% Überabschätzung



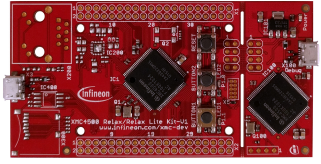
`Platin`

Instruktionscache

- **Deaktiviert: 96%**
- **Aktiviert: –**

Evaluation der WCET-Analysatoren `Platin` & `aiT`

- WCET-Analysator: `aiT` von AbsInt
- Hardware: ARM Cortex-M4
- *GenE*: 10 000 generierte Benchmarks
- Optimum: 0% Überabschätzung



`Platin`

Instruktionscache

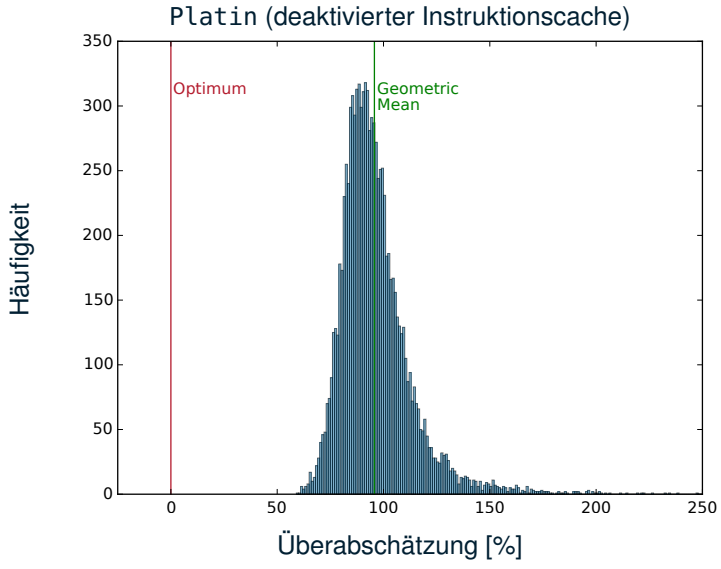
- **Deaktiviert: 96%**
- **Aktiviert: –**

Überabschätzung `aiT`

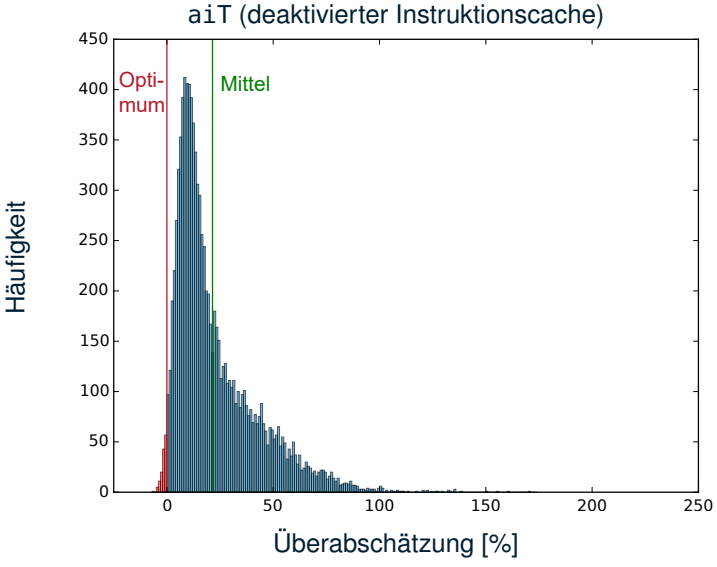
Instruktionscache

- **Deaktiviert: 23%**
- **Aktiviert: 36%**

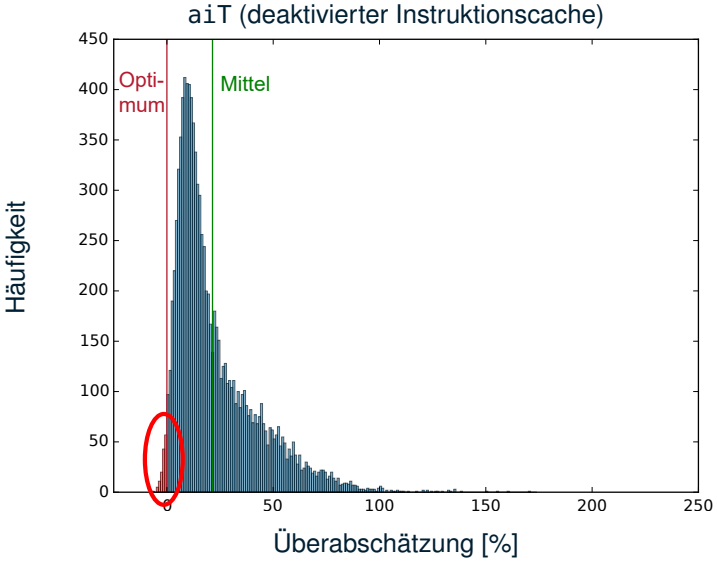
Evaluation: Platin



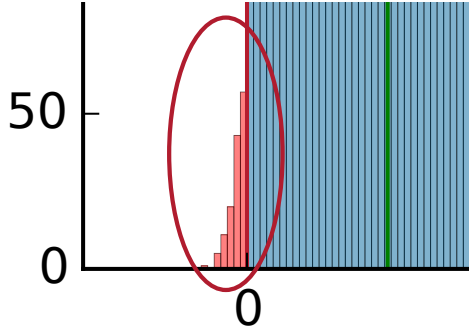
Evaluation: aiT



Evaluation: aiT

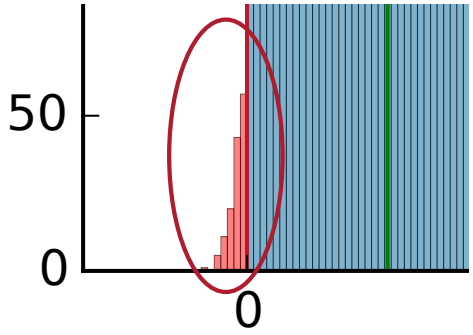


Evaluation: Bugs in WCET-Analysator `aiT`



- Beobachtung: Unterabschätzungen der tatsächlichen WCET
- *AbsInt*: Fehler in WCET-Analysator entdeckt durch Benchmarks von *GenE*
  1. Pipeline-Modell: spekulative Sprungvorhersage
  2. Speicher-Modell: fehlerhafte Zugriffsdauer

Evaluation: Bugs in WCET-Analysator `aiT`



- Beobachtung: Unterabschätzungen der tatsächlichen WCET
- *AbsInt*: Fehler in WCET-Analysator entdeckt durch Benchmarks von *GenE*
  1. Pipeline-Modell: spekulative Sprungvorhersage
  2. Speicher-Modell: fehlerhafte Zugriffsdauer

✓ *GenE*: Bestimmung Genauigkeit & **Unterabschätzungen aufgedeckt**