

Echtzeitsysteme

Zeitliche Analyse von Echtzeitanwendungen

Peter Ulbrich

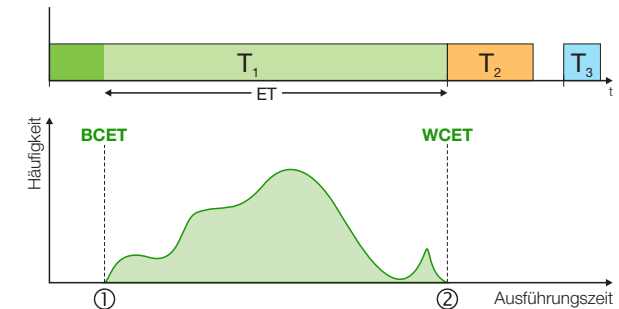
Lehrstuhl für Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://www4.cs.fau.de>

04. November 2015



Die maximalen Ausführungszeit



- Alle sprechen von der **maximalen Ausführungszeit**
 - **Worst-Case Execution Time (WCET)** e_i (vgl. III-2/26)
 - Unabdingbares Maß für **zulässigen Ablaufplan** (vgl. III-2/31)
- Tatsächliche Ausführungszeit bewegt sich zwischen:
 - 1 bestmöglicher Ausführungszeit (**Best-Case Execution Time, BCET**)
 - 2 schlechtest möglicher Ausführungszeit (besagter **WCET**)



Gliederung

- 1 Problemstellung
- 2 Messbasierte WCET-Analyse
- 3 Statische WCET-Analyse
 - Problemstellung
 - Timing Schema
 - Implicit Path Enumeration Technique
- 4 Hardware-Analyse
 - Die Maschinenprogrammebene
 - Cache-Analyse
 - Werkzeugunterstützung
- 5 Zusammenfassung



Bestimmung der WCET – eine Herausforderung

Wovon hängt die maximale Ausführungszeit ab?

Beispiel: Bubblesort

```
void bubbleSort(int a[], int size) {  
    int i, j;  
    for(i = size - 1; i > 0; --i) {  
        for(j = 0; j < i; ++j) {  
            if(a[j] > a[j+1]) {  
                swap(&a[j], &a[j+1]);  
            }  
        }  
    }  
    return;  
}
```

Programmiersprachenebene:

- Anzahl der Schleifendurchläufe hängt von der Größe des Feldes $a[]$ ab
- Anzahl der Vertauschungen (swap) hängt von dessen Inhalt
- ⚠ **Exakte Vorhersage ist kaum möglich**
 - Größe und Inhalt von $a[]$ kann zur Laufzeit variieren
 - Welches ist der **längste Pfad**?

Maschinenprogrammebene:

- Ausführungsdauer der **Elementaroperationen** (ADD, LOAD, ...)
- ⚠ **Prozessorabhängig** und für moderne Prozessoren sehr schwierig
 - Cache ~> Liegt die Instruktion/das Datum im schnellen Cache?
 - Pipeline ~> Wie ist der Zustand der Pipeline an einer Instruktion?
 - Out-of-Order-Execution, Branch-Prediction, Hyper-Threading, ...



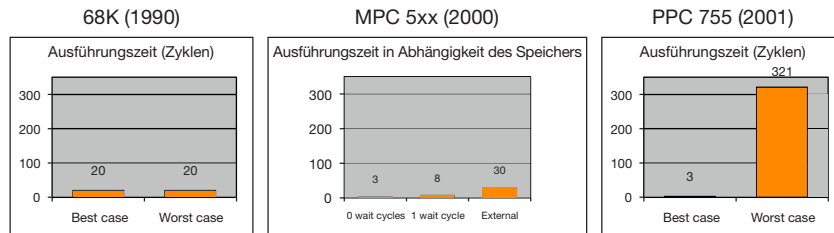
Ausführungszeit von Elementaroperationen

Die Crux mit der Hardware

Ausführungszeit von Elementaroperationen ist **essentiell**

Die Berechnung ist alles andere als einfach, ein Beispiel:

```
1 /* x = a + b */
2 LOAD r2, _a
3 LOAD r1, _b
4 ADD r3, r2, r1
```



Quelle: Christian Ferdinand [1]

Laufzeitbedarf ist hochgradig **Hardware-** und **kontextspezifisch**



Gliederung

- 1 Problemstellung
- 2 Messbasierte WCET-Analyse
- 3 Statische WCET-Analyse
 - Problemstellung
 - Timing Schema
 - Implicit Path Enumeration Technique
- 4 Hardware-Analyse
 - Die Maschinenprogrammebene
 - Cache-Analyse
 - Werkzeugunterstützung
- 5 Zusammenfassung



Messbasierte WCET-Analyse [3]

Idee: Prozessor selbst ist das präziseste Hardware-Modell
→ Dynamische Ausführung und Beobachtung der Ausführungszeit

Messbasierte WCET-Analyse:

- Intuitiv und gängige Praxis in der Industrie
- Weiche/feste Echtzeitsysteme erfordern keine sichere WCET
- Einfach umzusetzen, verfügbar und anpassbar
 - Verschafft leicht Orientierung über die tatsächliche Laufzeit
 - Geringer Aufwand zur Instrumentierung (Plattformwechsel)
 - Eingeschränkte Verfügbarkeit statischer Analysewerkzeuge (HW-Plattform)
- Sinnvolle Ergänzung zur statischen WCET-Analyse (III-3/11 ff)
 - Validierung statisch bestimmter Werte
 - Ausgangspunkt für die Verbesserung der statischen Analyse

Das Richtige zu messen ist das Problem!



Problem: Längster Pfad

Beispiel: Bubblesort

```
void bubbleSort(int a[], int size) {
    int i, j;
    for(i = size - 1; i > 0; --i) {
        for(j = 0; j < i; ++j) {
            if(a[j] > a[j+1]) {
                swap(&a[j], &a[j+1]);
            }
        }
    }
    return;
}
```

Aufruf: bubbleSort(a, size)

- Durchläufe, Vergleiche und Vertauschungen (engl. **Swap**)
- a = {1, 2}, size = 2
→ D = 1, V = 1, S = 0;
- a = {1, 3, 2}, size = 3
→ D = 3, V = 3, S = 1;
- a = {3, 2, 1}, size = 3
→ D = 3, V = 3, S = 3;



Für den **allgemeinen Fall nicht berechenbar** → Halteproblem

- Wie viele Schleifendurchläufe werden benötigt?



In Echtzeitsystemen ist dieses Problem häufig lösbar

- Kanonische Schleifenkonstrukte beschränkter Größe → max(size)
- Pfadanalyse → Nur maximale Pfadlänge von belang



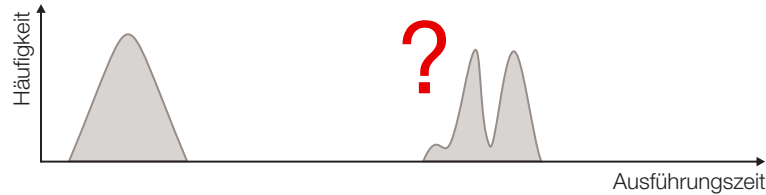
Herausforderungen der Messung

Messungen umfassen stets das **Gesamtsystem**

→ Hardware, Betriebssystem, Anwendung(en), ...

⚠ **Fluch** und **Segen**

■ Mögliches Ergebnis einer Messung:

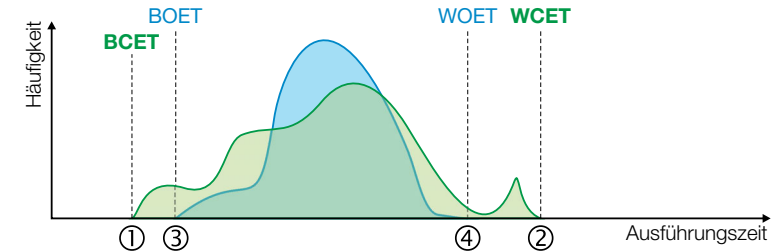


⚠ **Probleme** und **Anomalien**

- **Nebenläufige Ereignisse** unterbinden → Verdrängung
- **Gewählte Testdaten** führen nicht unbedingt zum **längsten Pfad**
- **Seltene** Ausführungsszenarien → Ausnahmefall
- **Abschnittsweise WCET-Messung** ↗ Globalen WCET
- Wiederherstellung des **Hardwarezustandes** schwierig/unmöglich



Aussagekraft messbasierter WCET-Analyse



■ Dynamische WCET-Analyse liefert **Messwerte**:

- 3 Bestmögliche beobachtete Ausführungszeit (Best Observed Execution Time, **BOET**)
- 4 Schlechtest mögliche beobachtete Ausführungszeit (Worst Observed Execution Time, **WOET**)



Messbasierte Ansätze unterschätzen die WCET meistens

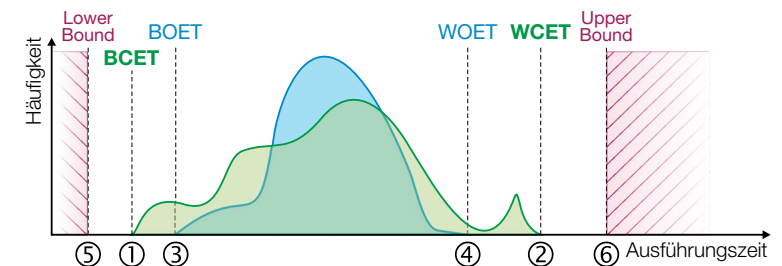


Gliederung

- 1 Problemstellung
- 2 Messbasierte WCET-Analyse
- 3 Statische WCET-Analyse
 - Problemstellung
 - Timing Schema
 - Implicit Path Enumeration Technique
- 4 Hardware-Analyse
 - Die Maschinenprogrammebene
 - Cache-Analyse
 - Werkzeugunterstützung
- 5 Zusammenfassung



Überblick: Statische WCET-Analyse



■ Statische WCET-Analyse liefert **Schranken**:

- 5 Geschätzte untere Schranke (**Lower Bound**)
- 6 Geschätzte obere Schranke (**Upper Bound**)



Die Analyse ist **vollständig** (engl. *sound*) falls $\text{Upper Bound} \geq \text{WCET}$



Berechnung der WCET?

Mit der Anzahl f_i der Ausführungen einer Kante E_i bestimmt man die WCET e durch Summation der Ausführungszeiten des längsten Pfades:

$$e = \max_P \sum_{E_i \in P} f_i e_i$$

Problem: Erfordert die explizite Aufzählung aller Pfade

→ Das ist algorithmisch nicht handhabbar

Lösung: Vereinfachung der konkreten Pfadsemantik

→ Abstraktion und Abbildung auf ein Flussproblem

- Flussprobleme sind mathematisch gut untersucht
- Im folgenden zwei Lösungswege: Timing Schema und IPET



Problem: Längster Pfad (Forts.)

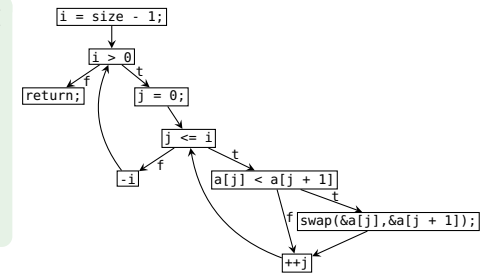
Die möglichen Wege lassen sich durch Kontrollflussgraphen beschreiben



Kontrollflussgraph (engl. *control flow graph*)

- Gerichteter Graph aus Grundblöcken (engl. *basic blocks*)
- Grundblöcke sind sequentielle „Code-Schnipsel“
 - hier wird gearbeitet → Grundblöcke verbrauchen Rechenzeit
- Kanten im Kontrollflussgraphen → Sprünge zwischen Grundblöcken

```
void bubbleSort(int a[], int size) {
    int i, j;
    for(i = size - 1; i > 0; -i) {
        for(j = 0; j < i; ++j) {
            if(a[j] > a[j+1]) {
                swap(&a[j], &a[j+1]);
            }
        }
        return;
    }
}
```

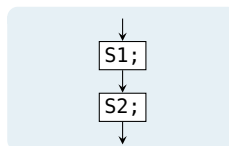


Lösungsweg₁: Timing Schema

Eine einfache Form der Sammelsemantik

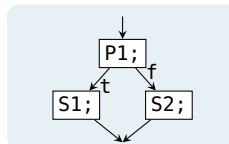
Sequenzen → Hintereinanderausführung

`S1();`
`S2();` Summation der WCETs:
 $e_{seq} = e_{S1} + e_{S2}$



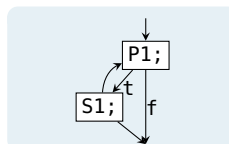
Verzweigung → bedingte Ausführung

`if(P1())`
`S1();`
`else S2();` Maximale Gesamtausführungszeit:
 $e_{cond} = e_{P1} + \max(e_{S1}, e_{S2})$



Schleifen → wiederholte Ausführung

`while(P1())`
`S1();` Schleifendurchläufe berücksichtigen:
 $e_{loop} = e_{P1} + n(e_{P1} + e_{S1})$



Timing Schema: Bubblesort

```
void bubbleSort(int a[], int size) {
    int i, j;
    for(i = size - 1; i > 0; -i) {
        for(j = 0; j < i; ++j) {
            if(a[j] > a[j+1]) {
                swap(&a[j], &a[j+1]);
            }
        }
        return;
    }
}
```

Schleife L_1 : $P_1 = i > 0$

- Rumpf: L_2 ; $--i$;
- Durchläufe: $size - 1$
- $e_{L_1} = e_{P_1} + (size - 1)(e_{P_1} + e_{L_2} + e_{--i})$

Schleife L_2 : $P_2 = j < i$

- Rumpf: C_1 ; $++j$;
- Durchläufe: $size - 1$
- $e_{L_2} = e_{P_2} + (size - 1)(e_{P_2} + e_{C_1} + e_{++j})$

- Verzweigung C_1 : $P_3 = a[j] > a[j + 1]$

- $S_1 = \text{swap}(\&a[j], \&a[j + 1])$

→ $e_{C_1} = e_{P_3} + e_{S_1}$

- Funktionsaufruf $S_1 = \text{swap}(\&a[j], \&a[j + 1])$

- Analog zum hier vorgestellten Verfahren



Timing Schema: Eigenschaften, Vor- und Nachteile

Eigenschaften

- Traversierung des abstrakten Syntaxbaums (AST) **bottom-up**
 - An den Blättern beginnend, bis zur Wurzel
 - Ausgangspunkt sind also explizite Pfade
- **Aggregation** der maximale Ausführungszeit nach festen Regeln
 - Für Sequenzen, Verzweigungen und Schleifen

Vorteile

- + Einfaches Verfahren mit geringem Berechnungsaufwand
- + Skaliert gut mit der Programmgröße

Nachteile

- Informationsverlust durch Aggregation
 - Korrelationen (z. B. sich ausschließende Zweige) nicht-lokaler Codeteile lassen sich nicht berücksichtigen
 - Schwierige Integration mit einer separaten Hardware-Analyse
- Nichtrealisierbare Pfade (infeasible paths) nicht ausschließbar
 - Unnötige Überapproximation



Lösungsansatz₂: Implicit Path Enumeration Technique



Explizite Pfadanalyse ohne Vereinfachung nicht handhabbar



Lösungsansatz: Nutzung impliziter Pfade

→ **Implicit Path Enumeration Technique (IPET)** [2]

- **Vorgehen:** Transformation des Kontrollflussgraphen in ein ganzzahliges, lineares Optimierungsproblem (ILP)
 - 1 Bestimmung des **Zeitanalysegraphen** aus dem Kontrollflussgraphen
 - 2 Abbildung auf ein **lineare Optimierungsproblem**
 - 3 Annotation von **Flussrestriktionen**
 - Nebenbedingungen im Optimierungsproblem
 - 4 Lösung des Optimierungsproblems (z.B. mit gurobi¹)



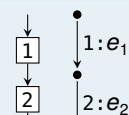
Globale Vereinfachung des Graphen statt lokaler Aggregation



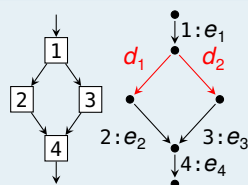
Der Zeitanalysegraph (engl. *timing analysis graph*)

- Ein **Zeitanalysegraph (T-Graph)** ist ein gerichteter Graph mit einer Menge von Knoten $\mathcal{V} = \{V_i\}$ und Kanten $\mathcal{E} = \{E_i\}$
 - Mit genau einer **Quelle** und einer **Senke**
 - Jede Kante ist Bestandteil eines Pfades P von der Senke zur Kante
 - Jeder Kante wird ihre WCET e_i zugeordnet
- ⚠ Verzweigungen benötigen **Dummy-Kanten** d_i

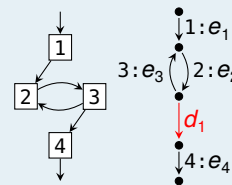
Sequenz



Verzweigung



Schleife



Graphentheorie annotiert Kosten klassischerweise **an Kanten**



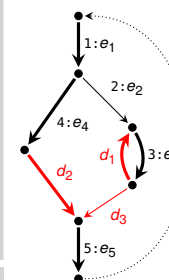
Zirkulationen



Abbildung $f : \mathcal{E} \mapsto \mathbb{R}$ heißt **Zirkulation**, falls sie den Fluss erhält

- Kanten wird die **Zahl der Ausführungen** f_i als Fluss zugeordnet
- **Flusserhaltung:** Jeder Knoten wird gleich oft betreten und verlassen
 - Erfordert die Einführung einer Rückkehrkante E_e mit $f_e = 1$

- Ausschluss ungültiger Abarbeitungen durch **Flussrestriktionen**
 - Formulierung als **Nebenbedingungen** des Optimierungsproblems
 - Beschränkung der maximalen Anzahl von Schleifendurchläufen

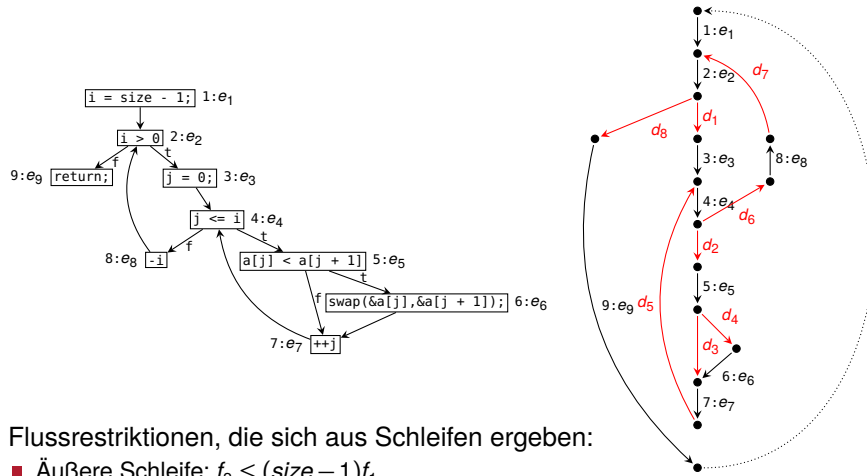


Beispiel

- $f_1 = f_2 + f_4$ wird durch die Zirkulation garantiert
- gültige Zirkulation: $\{E_1, E_4, d_2, E_5, E_e\} \cup \{E_3, d_1\}$
 - aber **keine gültige Abarbeitung**
- Flussrestriktion $f_3 \leq 5f_2$ löst dieses Problem
 - wird E_2 nicht abgearbeitet, so gilt $f_3 \leq 5 \cdot 0 = 0$
 - hier: Beschränkung auf 5 Schleifendurchläufe
 - Nebenbedingung des Optimierungsproblems



Beispiel: Bubblesort



- Flussrestriktionen, die sich aus Schleifen ergeben:
 - Äußere Schleife: $f_2 \leq (size - 1)f_1$
 - Innere Schleife: $f_4 \leq (size - 1)f_3$
- Flussrestriktionen, die sich aus Verzweigungen ergeben:
 - Bedingte Vertauschung: $f_{d_3} + f_6 = f_7$



Ganzzahliges Lineares Optimierungsproblem

■ Zielfunktion: Maximierung des gewichteten Flusses

$$WCETe = \max_{(f_1, \dots, f_e)} \sum_{E_i \in \mathcal{E}} f_i e_i$$

→ der Vektor $(f_1, \dots, f_e)$ maximiert die Ausführungszeit

■ Nebenbedingungen: Garantieren tatsächlich mögliche Ausführungen

- Flussserhaltung für jeden Knoten des T-Graphen

$$\sum_{E_j^+ = V_i} f_j = \sum_{E_k^- = V_i} f_k$$

- Flussrestriktionen für alle Schleifen des T-Graphen, z.B.

$$f_2 \leq (size - 1)f_1$$

- Rückkehrkante kann nur einmal durchlaufen werden: $f_{E_e} = 1$



IPET: Eigenschaften, Vor- und Nachteile

- Betrachtet implizit alle Pfade des Kontrollflussgraphen
 - Erzeugung des Zeitanalysegraphen
 - Überführung in ganzzahliges lineares Optimierungsproblem
- Vorteile
 - + Möglichkeit komplexer Flussrestriktionen
 - z. B. sich ausschließende Äste aufeinanderfolgender Verzweigungen
 - + Nebenbedingungen für das ILP sind leicht aufzustellen
 - + Viele Werkzeuge zur Lösung von ILPs verfügbar
- Nachteile
 - Lösen eines ILP ist im Allgemeinen NP-hart
 - Flussrestriktionen sind kein Allheilmittel
 - Beschreibung der Ausführungsreihenfolge ist problematisch



Gliederung

- 1 Problemstellung
- 2 Messbasierte WCET-Analyse
- 3 Statische WCET-Analyse
 - Problemstellung
 - Timing Schema
 - Implicit Path Enumeration Technique
- 4 Hardware-Analyse
 - Die Maschinenprogrammebene
 - Cache-Analyse
 - Werkzeugunterstützung
- 5 Zusammenfassung



WCET eines Code-Snippets?

Werte der Grundblöcke sind Eingabe für die Flussanalyse

 **Grundproblem:** Ausführungszyklen von Instruktionen zählen

```
_getopt:
  link    a6, #0      ; 16 Zyklen
  moveml  #0x3020, sp@- ; 32 Zyklen
  movel   a6@(8), a2    ; 16 Zyklen
  movel   a6@(12), d3   ; 16 Zyklen
```

■ Ergebnis: $e_{\text{getopt}} = 80$ Zyklen

■ Annahmen:

- Obere Schranke für jede Instruktion
- Obere Schranke der Sequenz durch Summation

Quelle: Peter Puschner [2]

 **Äußerst pessimistisch und zum Teil falsch**

■ **Falsch** für Prozessoren mit **Laufzeitanomalien**

- WCET der Sequenz > Summe der WCETs aller Instruktionen

■ **Pessimistisch** für **moderne Prozessoren**

- Pipeline, Cache, Branch Prediction, Prefetching, ... haben großen Anteil an der verfügbaren Rechenleistung heutiger Prozessoren
- Blanke Summation einzelner WCETs ignoriert diese Maßnahmen



Hardware-Analyse



Hardware-Analyse teilt sich in verschiedene Phasen

- Aufteilung ist nicht dogmenhaft festgeschrieben

■ **Integration** von Pfad- und Cache-Analyse

1 Pipeline-Analyse

- Wie lange dauert die Ausführung der Instruktionssequenz?

2 Cache- und Pfad-Analyse sowie WCET-Berechnung

- Cache-Analyse wird direkt in das Optimierungsproblem integriert

■ **Separate** Pfad- und Cache-Analyse

1 Cache-Analyse

- Kategorisiert Speicherzugriffe mit Hilfe einer Datenflussanalyse

2 Pipeline-Analyse

- Ergebnisse der Cache-Analyse werden anschließend berücksichtigt

3 Pfad-Analyse und WCET-Berechnung



Beispiel: Cache-Analyse [4, Kapitel 22]



Cache: ein kleiner, schneller Zwischenspeicher

- Zugriffszeiten variieren je nach Zustand des Caches enorm:

Treffer (engl. *hit*), Daten/Instruktion sind im Cache $\sim e_h$

Fehl Schlag (engl. *miss*), Daten/Instruktion sind nicht im Cache $\sim e_m$



Hits sind schneller als **Misses**: $e_m \gg e_h$

→ Strafe liegt schnell bei > 100 Taktzyklen

■ **Eigenschaften** von Caches mit Einfluss auf deren Analyse

Typ ■ Cache für **Instruktionen**

■ Cache für **Daten**

■ **kombinierter** Cache für **Instruktionen und Daten**

Auslegung ■ **direkt abgebildet** (engl. *direct mapped*)

■ **vollasoziativ** (engl. *fully associative*)

■ **satz- oder mengenassoziativ** (engl. *set associative*)

Seitenersetzungsstrategie

■ engl. *(pseudo) least recently used*, (Pseudo-)LRU

■ engl. *(pseudo) first in first out*, (Pseudo-)FIFO



Ergebnisse der Cache-Analyse

■ Wissen ob eine Instruktion / ein Datum im Cache ist, oder nicht:

must, die Instruktion ist **garantiert im Cache**

→ man kann immer die schnellere Ausführungszeit e_h annehmen

- wird für die Vorhersage von Treffern verwendet

may, die Instruktion ist **vielleicht im Cache**

→ ist dies nicht der Fall, muss man die Ausführungszeit e_m annehmen

- wird für die Vorhersage von Fehlschlägen verwendet

persistent, die Instruktion **verbleibt im Cache**

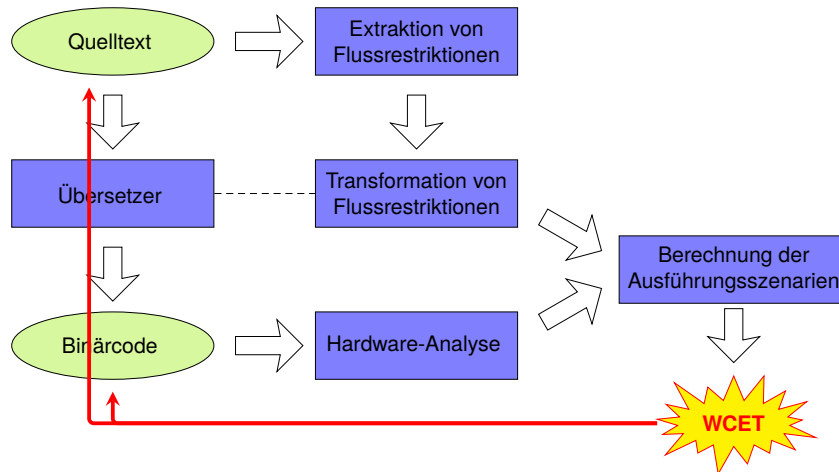
→ erster Zugriff ist ein Fehlschlag, alle weiteren sind Treffer

→ erster Zugriff: e_m , weitere Zugriffe: e_h

- ist besonders für Schleifen interessant, die den Cache „füllen“



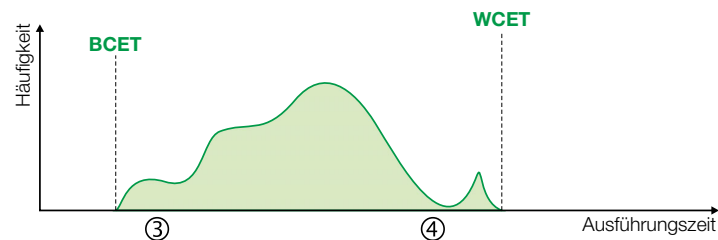
Werkzeugkette für die WCET-Analyse [3]



Gliederung

- 1 Problemstellung
- 2 Messbasierte WCET-Analyse
- 3 Statische WCET-Analyse
 - Problemstellung
 - Timing Schema
 - Implicit Path Enumeration Technique
- 4 Hardware-Analyse
 - Die Maschinenprogrammebene
 - Cache-Analyse
 - Werkzeugunterstützung
- 5 Zusammenfassung

Resümee



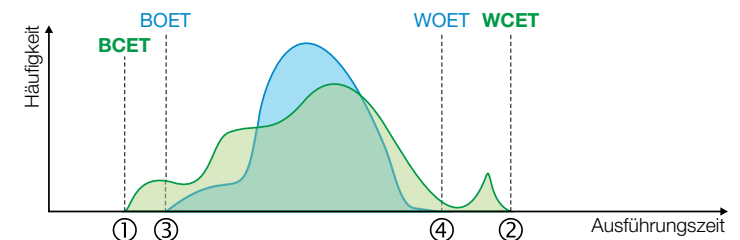
WCET-Bestimmung gliedert sich grob in zwei Teilprobleme

- **Programmiersprachenebene** (makroskopisch) $\leadsto$ finde die längsten Pfade durch ein Programm
- **Maschinenprogrammebene** (mikroskopisch) $\leadsto$ bestimme die WCET der Elementaroperationen



Tatsächliche Ausführungszeit: **BCET** / **WCET**

Resümee



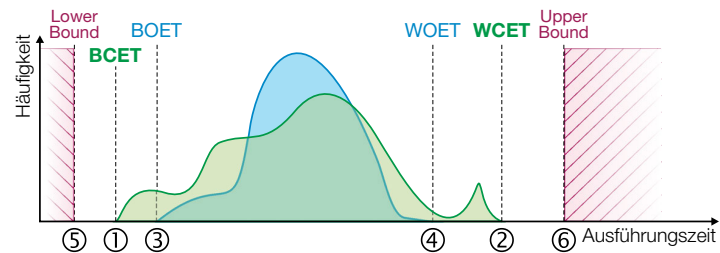
Dynamische Analyse $\leadsto$ Beobachtung der Ausführungszeit

- Messung bezieht beide Ebenen mit ein
- Vollständige Messung im Allgemeinen **nicht möglich** $\leadsto$ **Unterapproximation**



Gemessene Ausführungszeit: **BOET** / **WOET**

Resümee



Statische Analyse → schätzt die Ausführungszeit

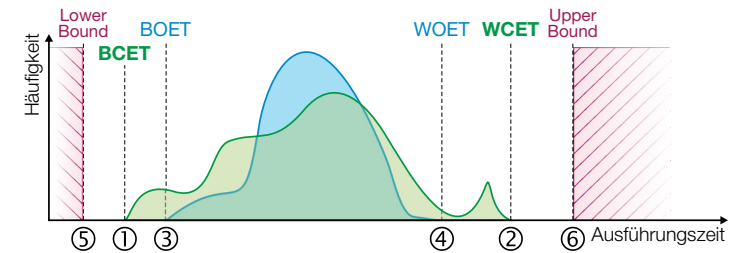
- Pfadanalyse (Programmiersprachenebene)
- Lösungswege: Abstraktion (Timing Schema vs. IPET)
- Gibt pessimistische Schranken an ~ **Überapproximation**



Geschätzte Ausführungszeitgrenzen: **Lower- / Upper Bound**



Resümee



Hardware-Analyse → Eingaben für die WCET-Berechnung

- Hauptaufgaben: **Cache-** und **Pipeline-Analyse**
- must-Approximation und may-Approximation



Werkzeugunterstützung kombiniert Ebenen und macht die WCET-Analyse handhabbar



Literaturverzeichnis

- [1] Ferdinand, C. ; Heckmann, R. ; Wolff, H.-J. ; Renz, C. ; Parshin, O. ; Wilhelm, R. :
Towards model-driven development of hard real-time systems.
In: *Model-Driven Development of Reliable Automotive Services*.
Springer, 2008, S. 145–160
- [2] Puschner, P. :
Zeitanalyse von Echtzeitprogrammen.
Treitlstr. 1-3/182-1, 1040 Vienna, Austria, Technische Universität Wien, Institut für
Technische Informatik, Diss., 1993
- [3] Puschner, P. ; Huber, B. :
Zeitanalyse von sicherheitskritischen Echtzeitsystemen.
<http://ti.tuwien.ac.at/rts/teaching/courses/wcet, 2012.> –
Lecture Notes
- [4] Wilhelm, R. :
Embedded Systems.
<http://react.cs.uni-sb.de/teaching/embedded-systems-10-11/lecture-notes.html>,
2010. –
Lecture Notes

