

Compiler für Eingebettete Systeme (CfES)

Sommersemester 2009

Dr. Heiko Falk

Technische Universität Dortmund

Lehrstuhl Informatik 12

Entwurfsautomatisierung für Eingebettete Systeme

Kapitel 3

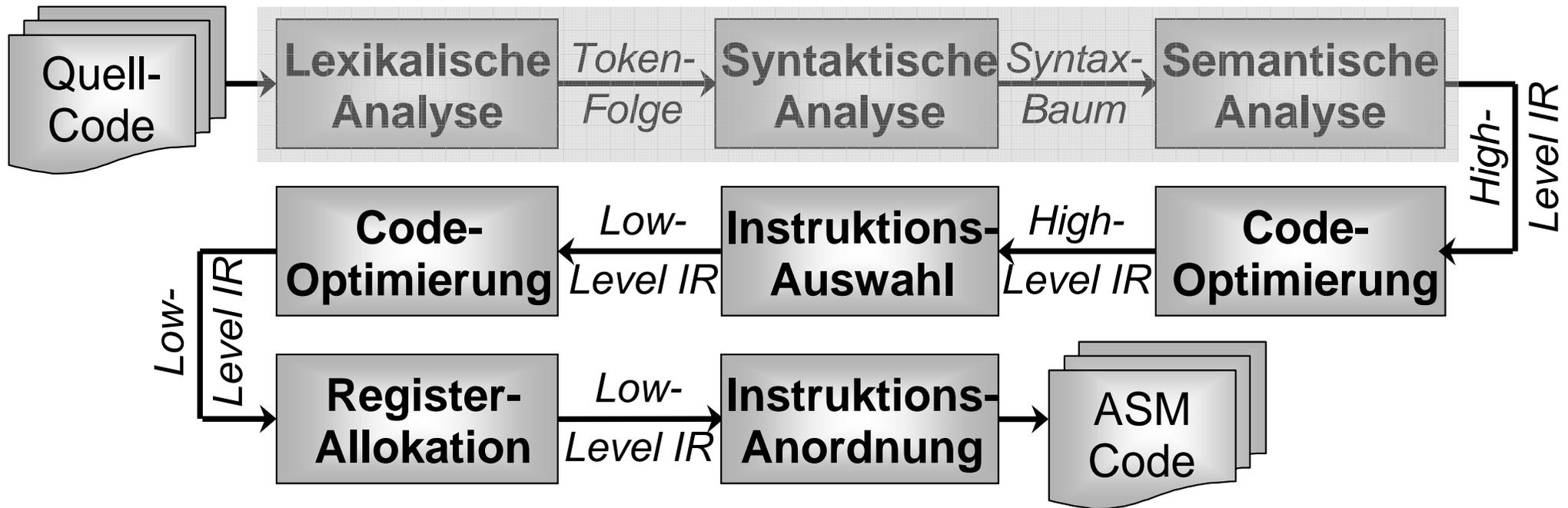
Prepass-Optimierungen

Gliederung der Vorlesung

- Kapitel 1: Compiler für Eingebettete Systeme
- Kapitel 2: Interner Aufbau von Compilern
- **Kapitel 3: Prepass-Optimierungen**
 - Motivation von Prepass-Optimierungen
 - Loop Nest Splitting
- Kapitel 3: Prepass-Optimierungen
- Kapitel 4: HIR Optimierungen und Transformationen
- Kapitel 5: Instruktionauswahl
- Kapitel 6: LIR Optimierungen und Transformationen
- Kapitel 7: Register-Allokation
- Kapitel 8: Compiler zur WCET_{EST}-Minimierung
- Kapitel 9: Ausblick

Motivation von Prepass-Optimierungen

➔ **Rückblick: Struktur eines Optimierenden Compilers mit 2 IRs:**

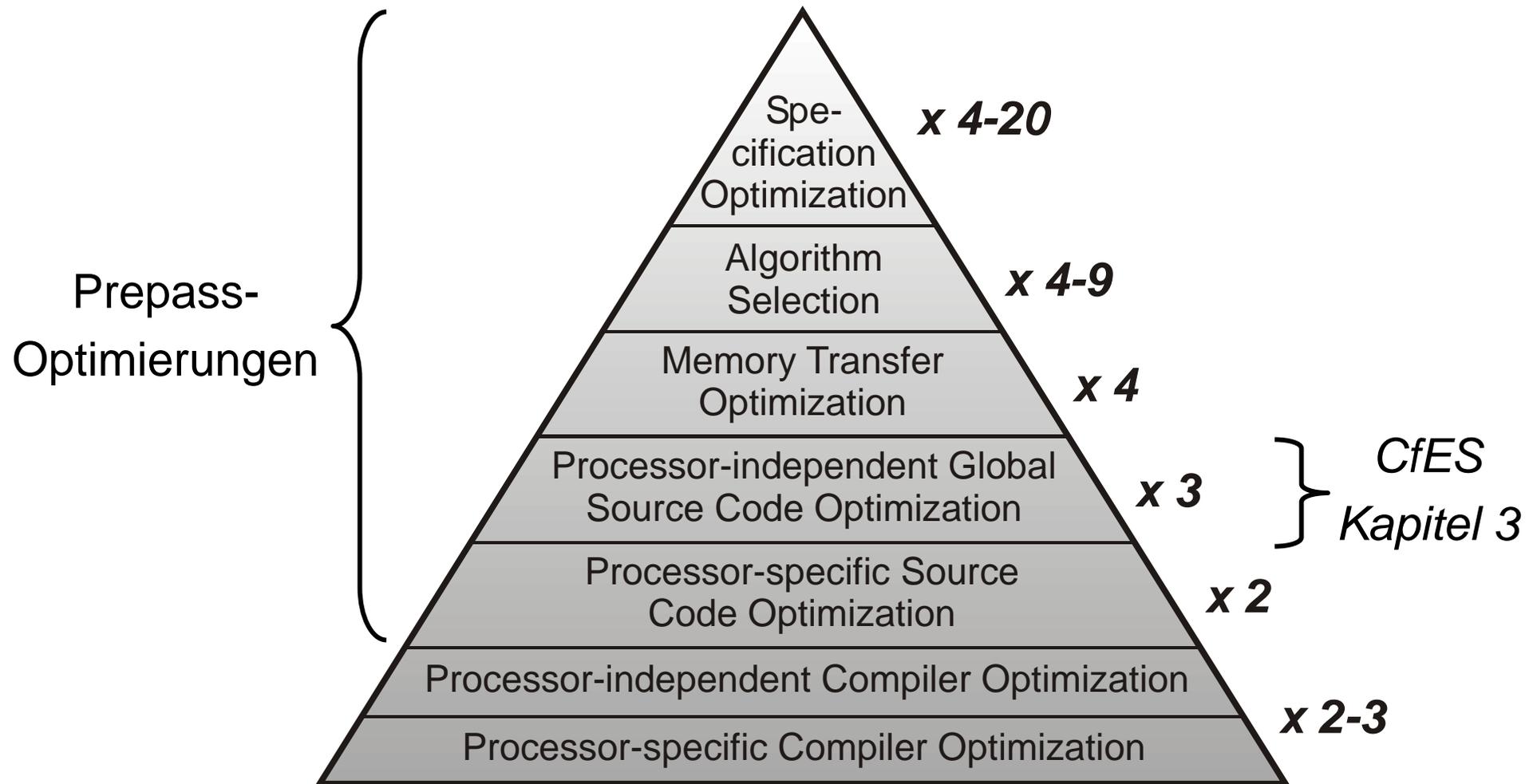


Frage: Darf nur der Compiler Code optimieren?

Motivation von Prepass-Optimierungen

- **Optimierungen außerhalb des Compilers heißen**
 - *Postpass-Optimierung*, wenn sie *nach* dem Compiler ablaufen
 - *Prepass-Optimierung*, wenn sie *vor* dem Compiler ablaufen
- **Vorteile von Prepass-Optimierungen:** 
 - Quellcode-Transformation leichter verständlich & zugänglich
 - Erlaubt manuelles „Ausprobieren“ einer Optimierungstechnik vor einer aufwändigen Implementierung.
 - Wegen Quellcode-Niveau unabhängig vom aktuellen Compiler; prinzipiell für jeden Compiler der Quellsprache anwendbar.
 - Wegen Quellcode-Niveau unabhängig von einer festen Zielarchitektur; prinzipiell für beliebige Architekturen anwendbar.

Abstraktionsebenen v. Optimierungen



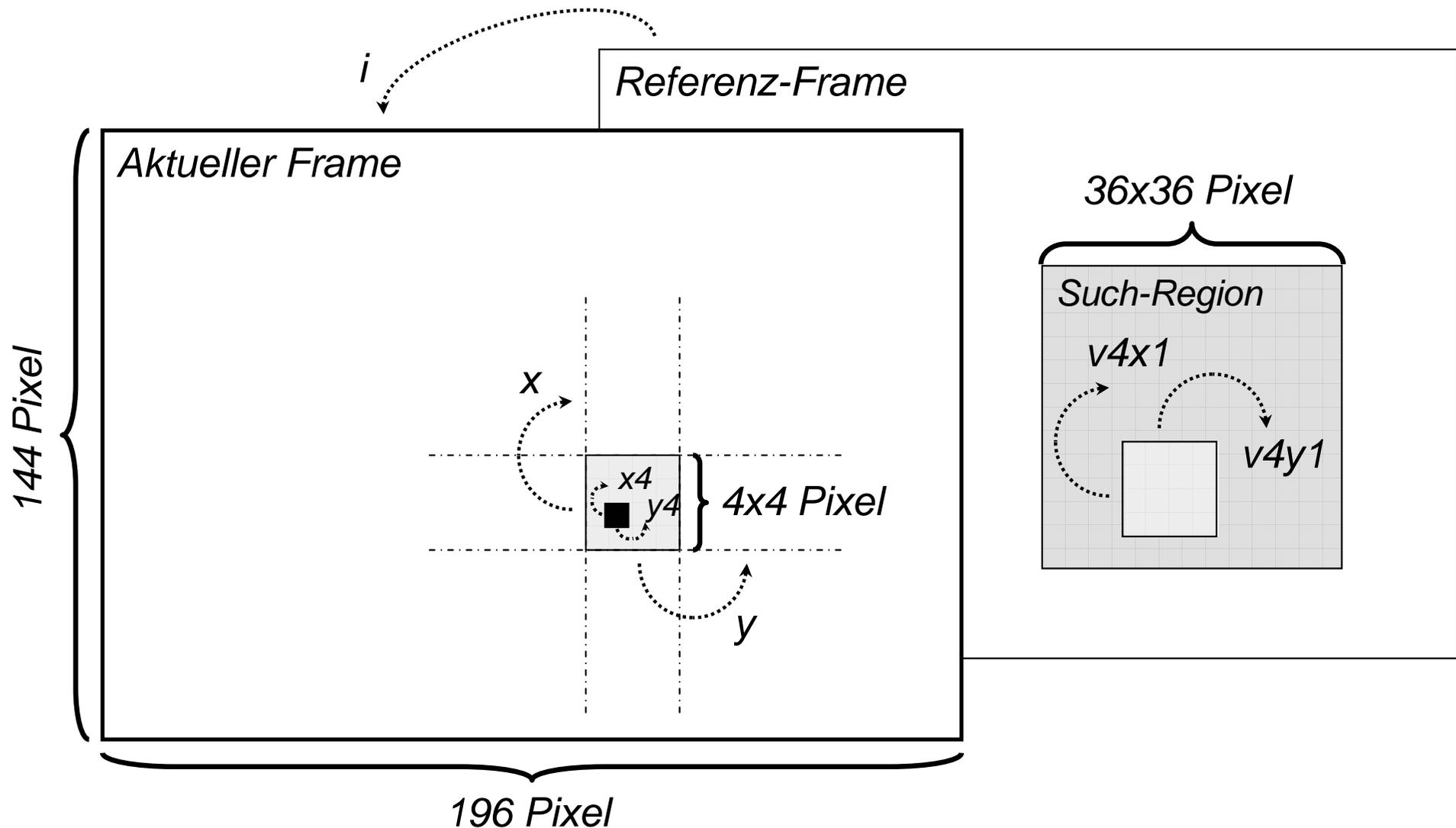
Gliederung der Vorlesung

- Kapitel 1: Compiler für Eingebettete Systeme
- Kapitel 2: Interner Aufbau von Compilern
- **Kapitel 3: Prepass-Optimierungen**
 - Motivation von Prepass-Optimierungen
 - Loop Nest Splitting
- Kapitel 3: Prepass-Optimierungen
- Kapitel 4: HIR Optimierungen und Transformationen
- Kapitel 5: Instruktionauswahl
- Kapitel 6: LIR Optimierungen und Transformationen
- Kapitel 7: Register-Allokation
- Kapitel 8: Compiler zur WCET_{EST}-Minimierung
- Kapitel 9: Ausblick

Einsatzgebiet Loop Nest Splitting

- **Eingebettete Multimedia-Anwendungen:**
 - *Datenfluß-dominiert*, d.h. erhalten große Datenmengen als Eingabe, berechnen große Datenmengen als Ausgabe (im Gegensatz zu kontrollfluß-dominierten Steuerungs- und Regelungsanwendungen).
 - Hauptteil der Laufzeit in (tief) *verschachtelten Schleifen* verbracht.
 - *Einfache Schleifenstrukturen* mit bekannten bzw. durch Compiler analysierbaren Unter- und Obergrenzen.
 - Manipulation großer mehrdimensionaler Felder.
 - *Typisches Beispiel: Streaming-Anwendungen wie MPEG4.*

Beispiel: MPEG4 Motion Estimation



Quellcode MPEG4 Motion Estimation

```

for (i=0; i<20; i++)
  for (x=0; x<36; x++)
    for (y=0; y<49; y++)
      for (vx=0; vx<9; vx++)
        for (vy=0; vy<9; vy++)
          for (x4=0; x4<4; x4++)
            for (y4=0; y4<4; y4++) {
              if (4*x+x4<0 || 4*x+x4>35 ||
                  4*y+y4<0 || 4*y+y4>48)
                then_block_1; else else_block_1;
              if (4*x+vx+x4-4<0 || 4*x+vx+x4-4>35 ||
                  4*y+vy+y4-4<0 || 4*y+vy+y4-4>48)
                then_block_2; else else_block_2; }

```

Beobachtungen

- **Bei Übersetzung und Ausführung dieses Quellcodes:**
 - Insgesamt werden 91.445.760 If-Statements ausgeführt.
 - Wegen If-Statements sehr unregelmässiger Kontrollfluß.
 - Zusätzlicher arithmetischer Aufwand:
Multiplikationen, Additionen, Vergleiche, logisches Oder, ...
- ☞ *Performance dieses Codes durch Kontrollfluß eingeschränkt,
nicht durch Berechnung der Bewegungsvektoren!*

Loop Nest Splitting

- **Automatisierte Schleifen- & If-Statement-Analyse:**
 - x , y , $x4$ und $y4$ nehmen niemals Werte ein, so daß Bedingungen $4*x+x4<0$ und $4*y+y4<0$ jemals wahr wären.
 - ☞ *Bedingungen können durch konstanten Wahrheitswert '0' ersetzt werden.*
 - Für $x \geq 10$ oder $y \geq 14$ sind beide If-Statements zwingend erfüllt, so daß deren *then*-Zweige ausgeführt werden.
 - ☞ *Für mehr als 92% aller Ausführungen der innersten $y4$ -Schleifen sind beide If-Statements wahr.*

Quellcode nach Loop Nest Splitting

```

for (i=0; i<20; i++)
  for (x=0; x<36; x++)
    for (y=0; y<49; y++)
      if (x>=10 || y>=14) // Splitting-If
        for (; y<49; y++) // Zweite y-Schleife
          for (vx=0; vx<9; vx++) ... {
            then_block_1; then_block_2; } // Keine If-Stmts
      else
        for (vx=0; vx<9; vx++) ... {
          if (0 || 4*x+x4>35 || 0 || 4*y+y4>48) // Alte
            then_block_1; else else_block_1; // If-Stmts
          if (4*x+vx+x4-4<0 || 4*x+vx+x4-4>35 || ...)
            then_block_2; else else_block_2; }

```

Optimierte Code-Struktur

- **Splitting-If:**
 - Wann immer die Bedingung des Splitting-If erfüllt ist, sind automatisch die Bedingungen aller ursprünglichen If-Statements erfüllt.
 - ☞ *Then-Zweig* des Splitting-If enthält keine originalen If-Statements mehr, sondern nur noch deren *then-Zweige*.
 - Wenn Bedingung des Splitting-If nicht erfüllt ist, ist keine Aussage über Wahrheitswert der ursprünglichen If-Statements möglich
 - ☞ *Else-Zweig* des Splitting-If enthält alle originalen If-Statements, um Korrektheit des Codes zu erhalten.

Warum Zweite Y-Schleife?

- Intuitiver Code:

```
for (x=0; x<36; x++)
  for (y=0; y<49; y++)
    if (x>=10 || y>=14)
      for (vx=0; vx<9; vx++) ...
```

- **Splitting-If:**

1 Ausführung für
jedes einzelne $y \in [14, 48]$

$y = 16$

- Optimierter Code:

```
for (x=0; x<36; x++)
  for (y=0; y<49; y++)
    if (x>=10 || y>=14)
      for (; y<49; y++)
        for (vx=0; vx<9; vx++) ...
```

- **Splitting-If:**

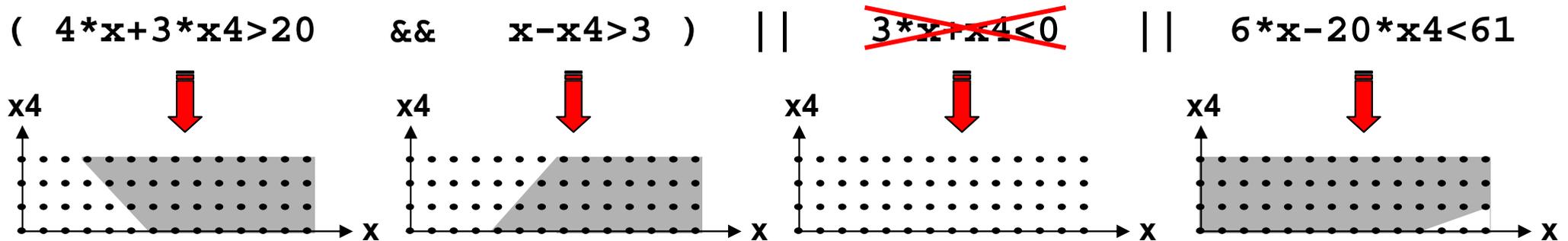
1 Ausführung für
alle $y \in [14, 48]$ *zusammen*

$y = 16$

Phasen von Loop Nest Splitting

- **Condition Satisfiability:** Finden einzelner stets erfüllbarer oder unerfüllbarer Bedingungen in If-Statements
- **Condition Optimization:** Für jede einzelne Bedingung C , finde eine einfacher auszudrückende Bedingung C' , so daß $C' \Rightarrow C$ gilt (wenn C' wahr ist, ist auch C wahr).
- **Search Space Generation:** Verknüpfe alle einzelnen Bedingungen C' zu einer Struktur G , die alle If-Statements mitsamt deren Struktur ($\&\&$, $|$) modelliert.
- **Search Space Exploration:** Bestimme anhand von G eine Bedingung für das Splitting-If, die zur Minimierung der Ausführung von If-Statements führt.

Ablauf von Loop Nest Splitting



Hinweis:

Dieses Beispiel entspricht nicht dem MPEG4-Code der vorigen Folien!

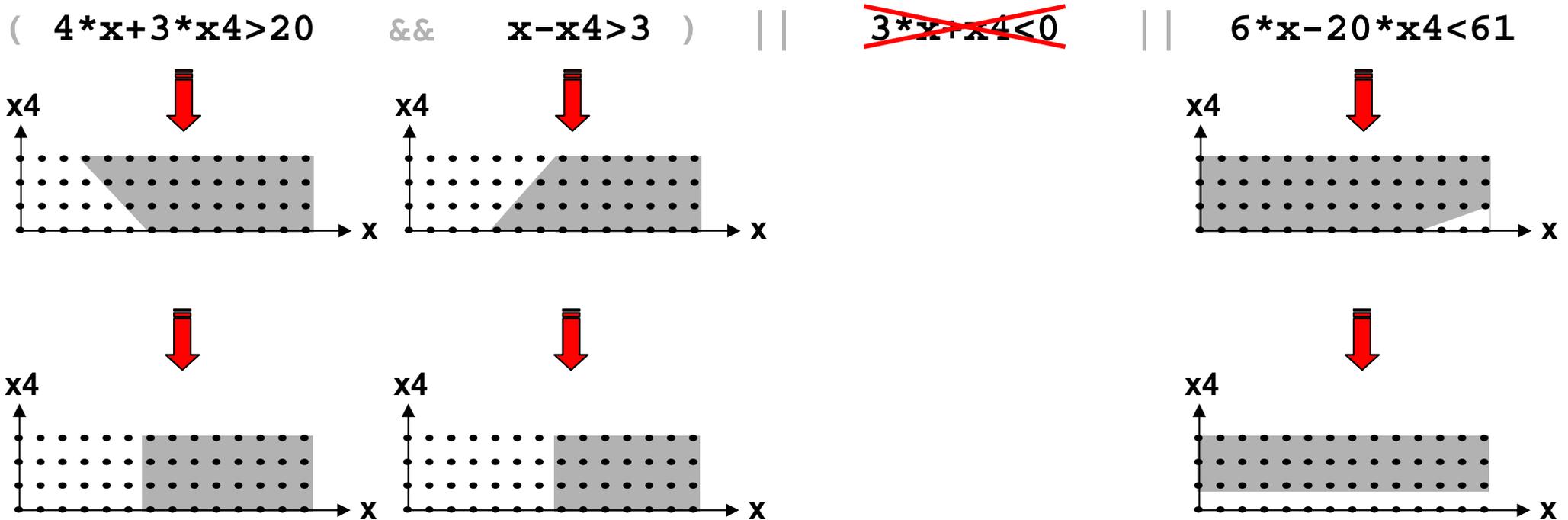
- **Angenommene Schleifengrenzen:**

$$0 \leq x \leq 13$$

$$0 \leq x4 \leq 3$$

1 - Condition Satisfiability

Ablauf von Loop Nest Splitting

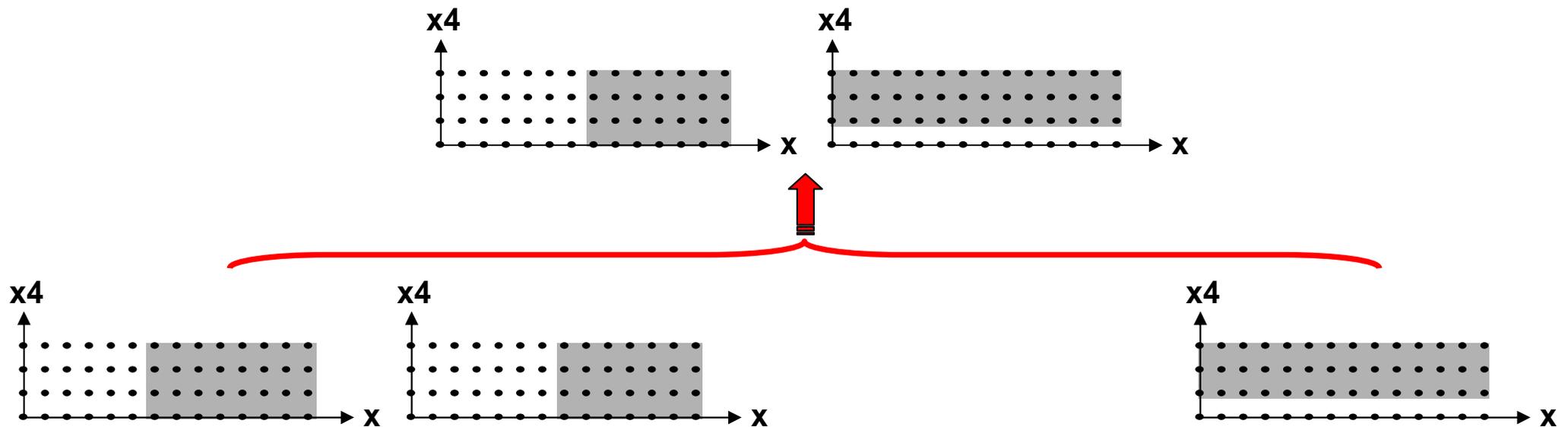


1 - Condition Satisfiability

2 - Condition Optimization

Ablauf von Loop Nest Splitting

$$(4 * x + 3 * x4 > 20 \quad \&\& \quad x - x4 > 3) \quad || \quad \del{3 * x + x4 < 0} \quad || \quad 6 * x - 20 * x4 < 61$$



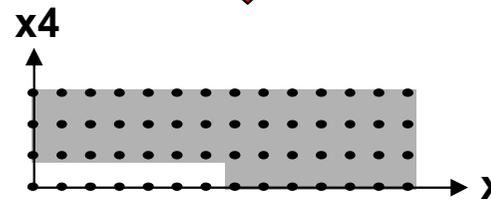
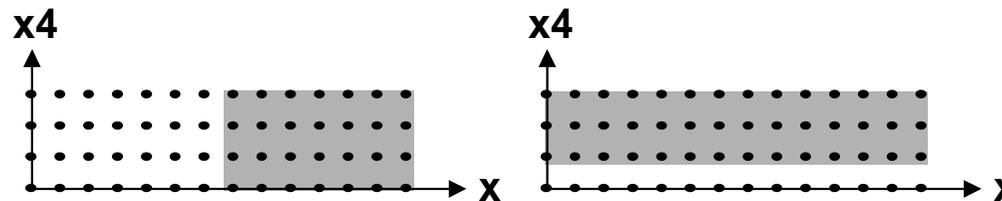
1 - Condition Satisfiability

2 - Condition Optimization

3 - Search Space Generation

Ablauf von Loop Nest Splitting

$$(4*x+3*x4>20 \quad \&\& \quad x-x4>3) \quad || \quad \cancel{3*x+x4<0} \quad || \quad 6*x-20*x4<61$$



$$x \geq 7 \quad || \quad x4 \geq 1$$

1 - Condition Satisfiability

2 - Condition Optimization

3 - Search Space Generation

4 - Search Space Exploration

Voraussetzungen

- **Schleifengrenzen:**

Alle unteren & oberen Grenzen (l_L, u_L) sind konstant.

- **If-Statements:**

Sequenz schleifenabhängiger Bedingungen, mit logischem UND bzw. ODER verknüpft.

- Format: $\text{if } (C_1 \otimes C_2 \otimes \dots) \quad \otimes \in \{ \&\&, | \}$

- **Schleifenabhängige Bedingungen:**

Lineare Ausdrücke über Indexvariablen i_L der Schleifen.

- Format: $C_x \cong \sum_{L=1}^N (c_L * i_L) + c \geq 0 \quad c_L, c \in \mathbb{Z}$

Polytope & Lineare Bedingungen

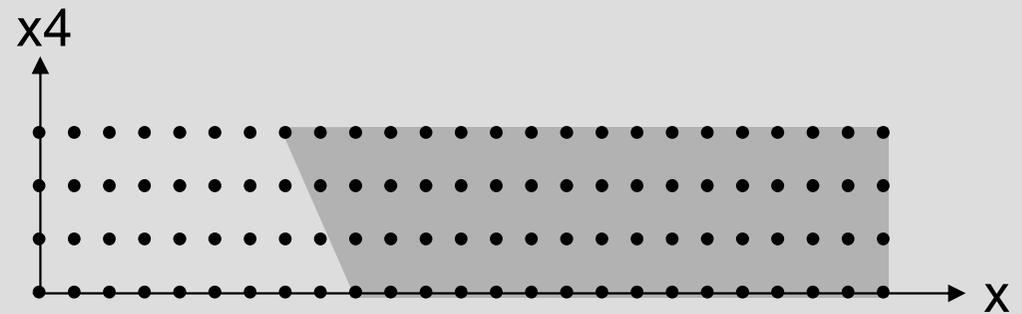
- **Definition (*Polyeder & Polytope*):**

- *Polyeder* $P = \{ x \in \mathbb{Z}^N \mid Ax \geq b \}$ $A \in \mathbb{Z}^{m \times N}, b \in \mathbb{Z}^m$
- *Polyeder* P heißt *Polytop* genau dann wenn $|P| < \infty$

- **Modell linearer Bedingungen in verschachtelten Schleifen:**

$4 \cdot x + 3 \cdot x_4 > 35$ für $x \in [0, 35], x_4 \in [0, 3]$ als Polytop

- $P = \{ p \in \mathbb{Z}^2 \mid \begin{pmatrix} 4 & 3 \\ 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} p \geq \begin{pmatrix} 36 \\ 0 \\ -35 \\ 0 \\ -3 \end{pmatrix} \}$



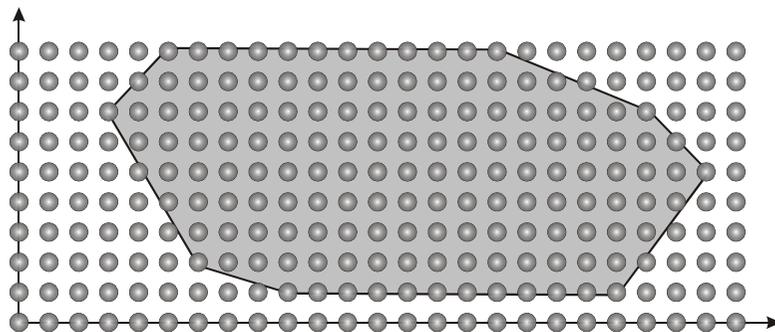
Konvexität von Polytopen

- **Definition (*Konvexität*):**

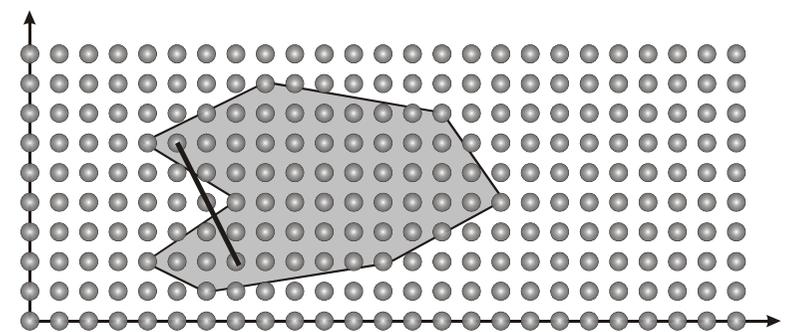
Eine Menge $S \subset \mathbb{Z}^N$ heißt *konvex*, wenn für alle $x, y \in S$ jede konvexe Kombination $ax + by$ von x und y ($a+b=1$; $a, b \geq 0$) in S ist.

- **Informal:** Jede Linie zwischen zwei beliebigen Punkten x und y aus S muss vollständig in S liegen.

☞ *Polytope sind konvex.*



Konvex



Nicht konvex

Operationen auf Polytopen

- Die **Schnittmenge** zweier Polytope ist auch ein Polytop.
- Die **Vereinigung** zweier Polytope ist kein Polytop, da die resultierende Menge nicht konvex ist.

- **Definition (Endliche Vereinigung v. Polyedern, EVP):**
 Für Polyeder P_1, \dots, P_n ($n \in \mathbb{N}$, $n < \infty$) heißt $V := \{P_1 \cup \dots \cup P_n\}$ endliche Vereinigung von Polyedern (EVP).

- **Mengenoperationen auf EVPs:** $A = \cup_i A_i$, $B = \cup_j B_j$
- $A \cap B := (\cup_i A_i) \cap (\cup_j B_j) = \cup_{i,j} (A_i \cap B_j)$
- $A \cup B := (\cup_i A_i) \cup (\cup_j B_j)$

Phase 1 – Condition Satisfiability

- **Ziel:** Finden einzelner schleifenabhängiger Bedingungen C_x , die für alle Werte der Indexvariablen aller umgebender Schleifen konstant 'true' oder konstant 'false' ergeben.
- **Vorgehen:**
 - Überführen jeder Bedingung C_x in Polytop P_x (siehe Folie 22)
 - Test auf leere Menge: $P_x == \emptyset \Rightarrow C_x$ stets 'false'
 - Test auf Allmenge / Universum: $P_x == U \Rightarrow C_x$ stets 'true'
- **Quellcode-Modifikation:** Ersetzung solcher konstant wahrer oder falscher Bedingungen im Quellcode durch Konstanten '0' bzw. '1'.

Phase 2 – Condition Optimization

- **Gegeben:** N Schleifen & eine Bedingung $C = \sum_{L=1}^N (c_L * i_L) + c \geq 0$

- **Angenommene hypothetische Situation:**

```
for ( loop L1 )
    ...
    for ( loop LN )
        if ( C ) ...
```

Annahme: Die Schleifen enthalten nur 1 If-Statement mit nur 1 Bedingung. Weitere If-Statements & Bedingungen sind quasi „ausgeblendet“.

- **Ziel:** Bestimme Intervalle $[l_{C,1}, u_{C,1}] \dots [l_{C,N}, u_{C,N}]$ mit:
 - C ist erfüllt für alle Schleifen-Iterationen in diesen Intervallen
 - Minimierung der Ausführung von If-Statements nach hypothetischem Loop Nest Splitting anhand dieser Intervalle.

Optimierung einer Bedingung C

- $C = \sum_{L=1}^N (c_L * i_L) + c \geq 0$
 - Bestimmung von Werten $l_{C,L}$ und $u_{C,L}$ für jede Schleife L
 - Interpretation: C ist erfüllt für alle $l_{C,L} \leq i_L \leq u_{C,L}$
 - Optimierungsziel:
Alle Werte $l_{C,L}$ und $u_{C,L}$ minimieren Gesamt-Anzahl ausgeführter If-Statements.

- Vereinfachung: Wegen Linearität von C
Entweder $l_{C,L} = l_L$ oder $u_{C,L} = u_L$
- Konsequenz: Bestimmung nur eines Wertes $v_{C,L}$
Entweder erfüllt $[v_{C,L}, u_L]$ oder $[l_L, v_{C,L}]$ Bedingung C

Ein paar Definitionen...

- **Gegeben:** $C = \sum_{L=1}^N (c_L * i_L) + c \geq 0$ $l_L, u_L \quad v_{C,L}$
- **Totaler Iterationsraum:** *(#Ausführungen d. innersten Schleifenkörpers)*

$$TIR = \prod_{L=1}^N (u_L - l_L + 1)$$
- **Beschränkter Iterationsraum:** *(#Ausführungen d. innersten Schleifenkörpers, eingeschränkt auf durch $v_{C,L}$ definierte Bereiche)*

$$BIR = \prod_{L=1}^N r_L \quad \text{mit}$$

$$r_L = \begin{cases} u_L - l_L + 1 & \text{für } c_L = 0, \\ u_L - v_{C,L} + 1 & \text{für } c_L > 0, \\ v_{C,L} - l_L + 1 & \text{sonst} \end{cases}$$
- **Splitting-Schleife:** *(Index der Schleife, wo Splitting stattfindet)*

$$\lambda = \max\{ i \mid L_i \in \Lambda, r_L \neq u_L - l_L + 1 \}$$

Veranschaulichung (1)

Schleife 1 `for (i=0; i<20; i++)`

`for (x=0; x<36; x++)`

`for (y=0; y<49; y++)`

`for (vx=0; vx<9; vx++)`

`for (vy=0; vy<9; vy++)`

`for (x4=0; x4<4; x4++)`

Schleife N=7

`for (y4=0; y4<4; y4++) {`

`if (4*x+vx+x4-4>35)`

`then_block_1; else else_block_1; }`

■ Beachte:

- Schleifen von 1 ... N von außen nach innen durchnummeriert.
- Nur 1 Bedingung im if-Statement, statt vieler wie auf Folie 10!

Veranschaulichung (2)

Schleife 1 `for (i=0; i<20; i++)`

`for (x=0; x<36; x++)`

`for (y=0; y<49; y++)`

`for (vx=0; vx<9; vx++)`

`for (vy=0; vy<9; vy++)`

`for (x4=0; x4<4; x4++)`

Schleife N=7

`for (y4=0; y4<4; y4++) {`

`if (4*x+vx+x4-4>35)`

`then_block_1; else else_block_1; }`

- **TIR:**

- Anzahl d. Ausführungen des Codes in geschweiften Klammern.
- Hier: $20 * 36 * 49 * 9 * 9 * 4 * 4 = 45.722.880$

Veranschaulichung (3)

```
Schleife 1 for (i=0; i<20; i++)
    for (x=0; x<36; x++)
        for (y=0; y<49; y++)
            for (vx=0; vx<9; vx++)
                for (vy=0; vy<9; vy++)
                    for (x4=0; x4<4; x4++)
                        Schleife N=7
                            for (y4=0; y4<4; y4++) {
                                if (4*x+vx+x4-4>35)
                                    then_block_1; else else_block_1; }
```

- Seien $v_{C,1} = 0$; $v_{C,2} = 10$; $v_{C,3} = \dots = 0$ vorgegeben.
 - Bedingung C ist erfüllt für alle $x \geq 10$ und $i, y, \dots, y4 \geq 0$.
 - BIR: $20 * (36 - 10) * 49 * 9 * 9 * 4 * 4 = 33.022.080$

Veranschaulichung (4)

```

for (i=0; i<20; i++)
  for (x=0; x<36; x++)
    if (x >= 10)
      for (; x<36; x++) ...
        for (y4=0; y4<4; y4++) {...}
    else
      for (y=0; y<49; y++) ...
        for (y4=0; y4<4; y4++) {
          if (4*x+vx+x4-4>35) ... }

```

BIR: Anzahl der Ausführungen des Codes in geschweiften Klammern für $x \geq 10$.

- **Hypothetischer Code für $v_{C,2} = 10$**
 - Splitting-Schleife $\lambda = 2$, da Splitting-If in 2. Schleife steht.
 - Frage: Wie oft würden alle hier vorkommenden If's ausgeführt?

Zählen von If-Statement Ausführungen

- **#If-Statements nach Splitting:** #Originaler If's + #Splitting If's

$$IF_{\text{Total}} = IF_{\text{Orig}} + IF_{\text{Split}}$$

- **#Originaler If's:** Totale Iterationen ohne Eingeschränkte Iterationen

$$IF_{\text{Orig}} = TIR - BIR$$

- **#Splitting-If's:**

$$IF_{\text{Split}} = \# \text{Then-blocks} + \# \text{Else-blocks} = TB + EB$$

- **#Then-Blocks:** BIR ohne Iterationen von Schleifen innerhalb von λ

$$TB = BIR / \left(\prod_{L=\lambda+1}^N (u_L - l_L + 1) * r_\lambda \right)$$

- **#Else-Blocks:** #Originaler If's ohne Schleifen-Iterationen in λ

$$EB = IF_{\text{Orig}} / \prod_{L=\lambda+1}^N (u_L - l_L + 1)$$

Erzeugen der Werte $v_{C,L}$

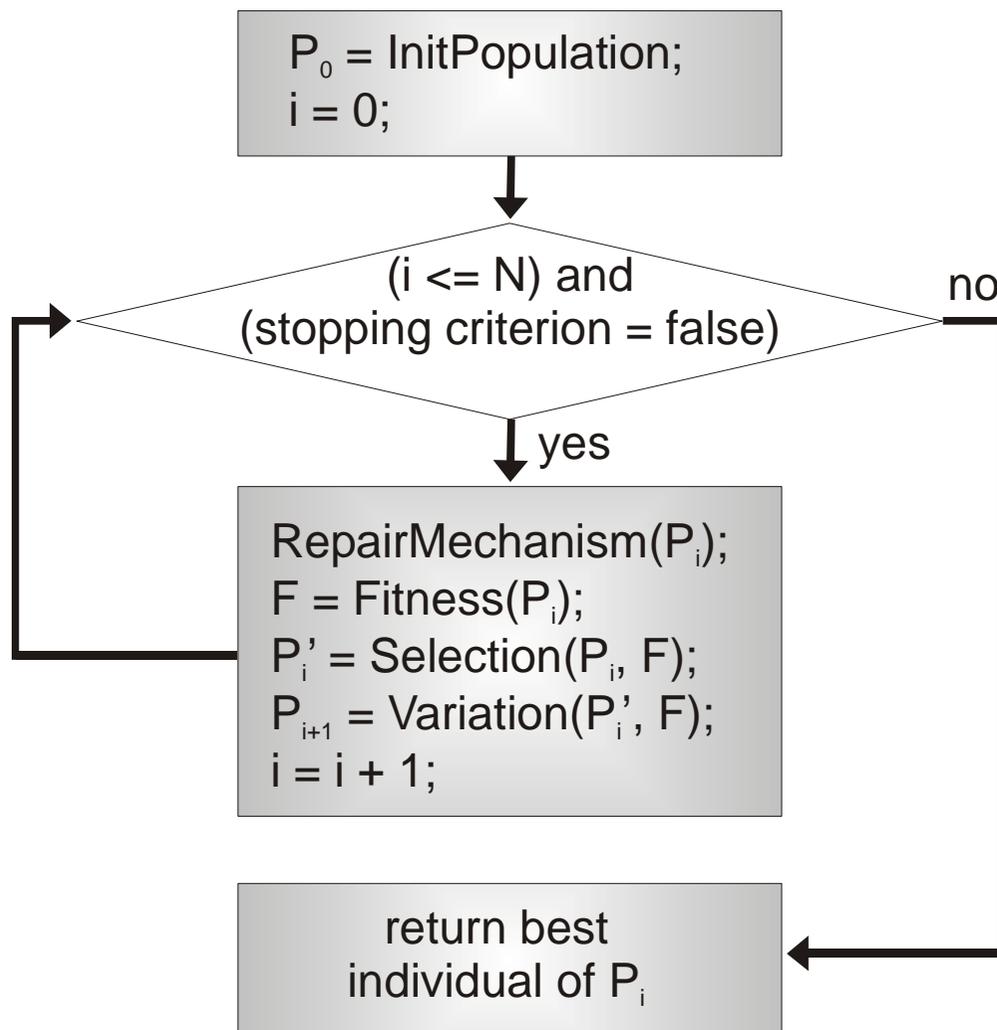
- **Bisher Erreichtes:**
 - Für eine Bedingung C und gegebene Werte $v_{C,L}$ kann berechnet werden, wie viele If-Statements nach Splitting anhand von $v_{C,L}$ ausgeführt werden würden.

Sehr schön, aber...

...wer erzeugt gute Werte für $v_{C,L}$?

☞ **Ein Genetischer Algorithmus**

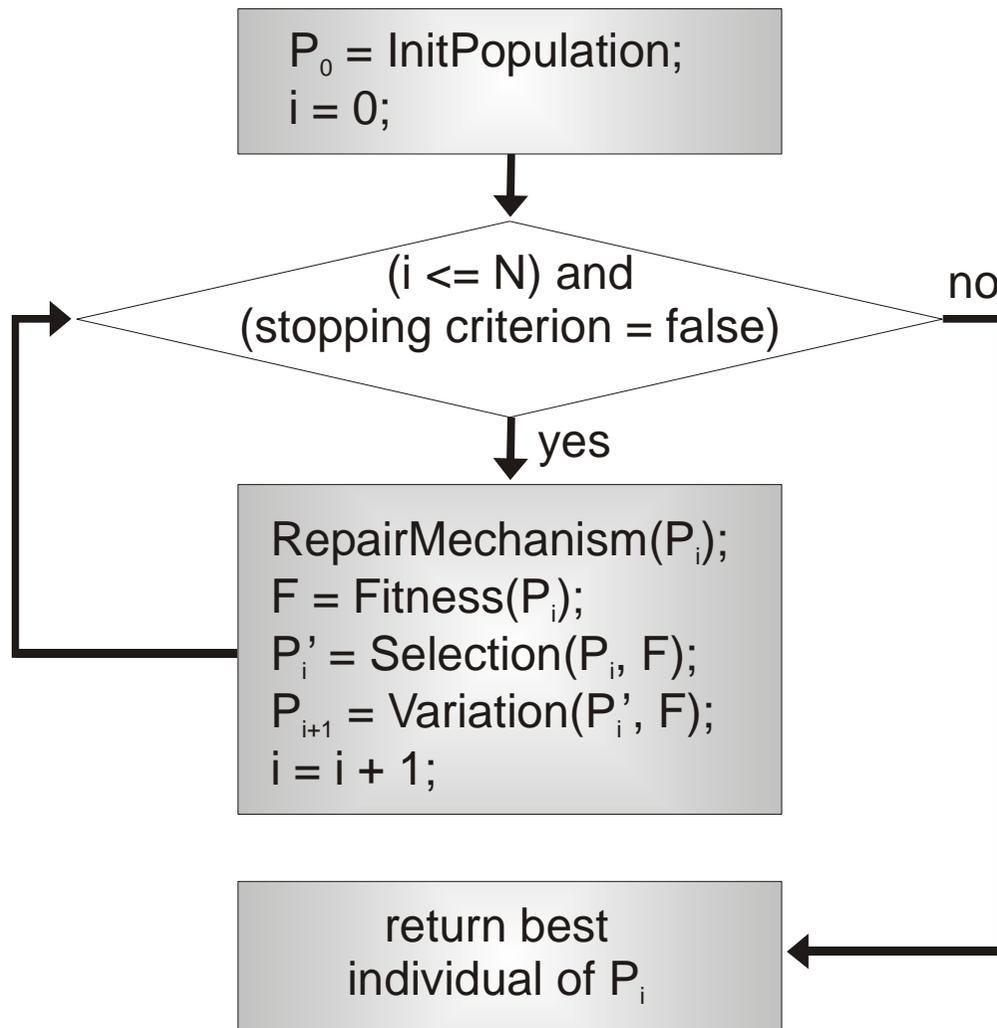
Ablauf Genetischer Algorithmen (1)



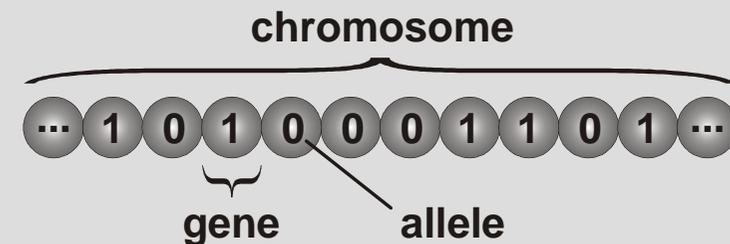
- Anlehnung an natürliche Evolution, “*Survival of the fittest*”
- Optimierung in Schleife $i=0,1,\dots$
- Jede Iteration i verwaltet *Population* P_i ; eine Population besteht aus vielen *Individuen*.
- Ein Individuum repräsentiert eine Lösung für das modellierte Optimierungsproblem.



Ablauf Genetischer Algorithmen (2)



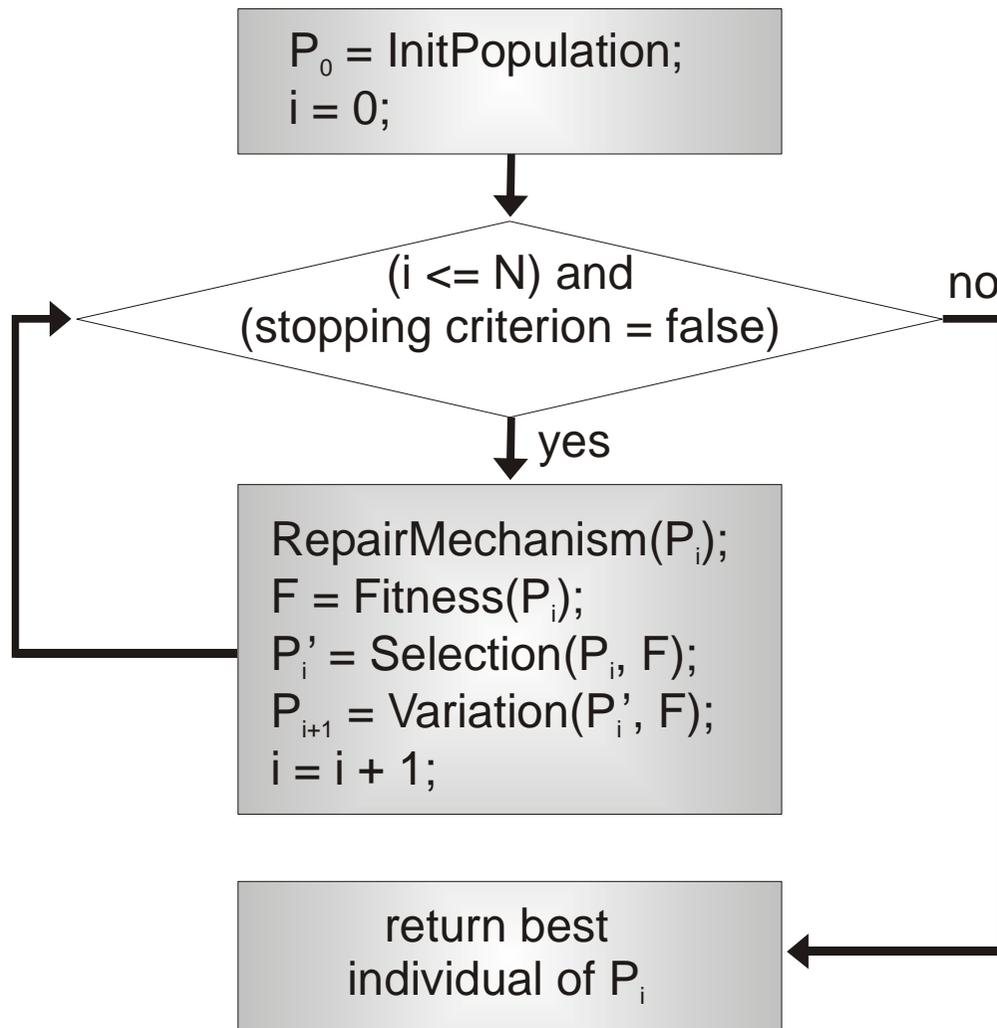
- Die Datenstruktur eines Individuums heißt *Chromosom*.



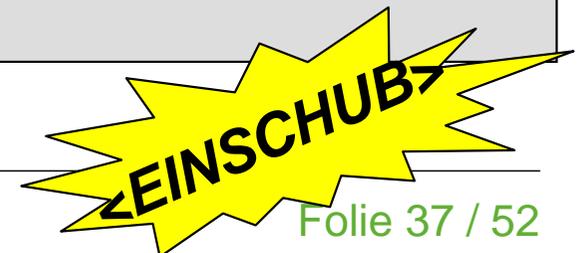
- Ein Chromosom besteht aus vielen *Genen*, die Daten speichern.
- Ein konkreter in einem Gen gespeicherter Wert heißt *Allel*.



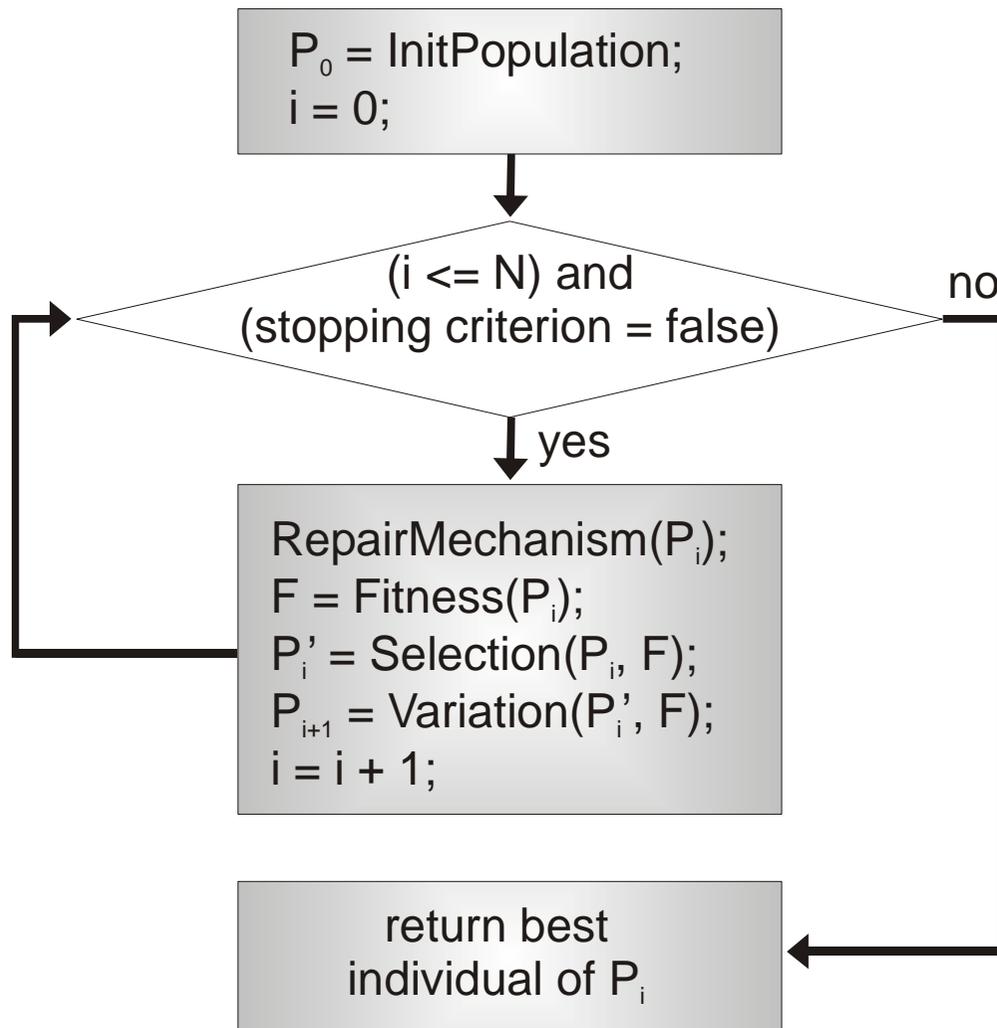
Ablauf Genetischer Algorithmen (3)



- Eine Fitness-Funktion berechnet für jedes Individuum in P_i dessen *Fitness*.
- Aus P_i wird eine Teilmenge P_i' mit höchster / niedrigster Fitness ausgewählt (*Selektion*, abhängig ob Minimierungs- oder Maximierungsproblem).
- P_i' wird durch zufällig generierte neue Individuen zur nächsten Population P_{i+1} ergänzt (*Variation*)

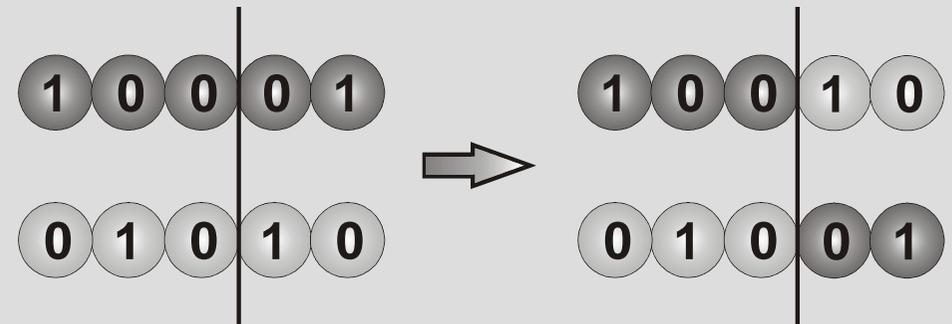


Ablauf Genetischer Algorithmen (4)

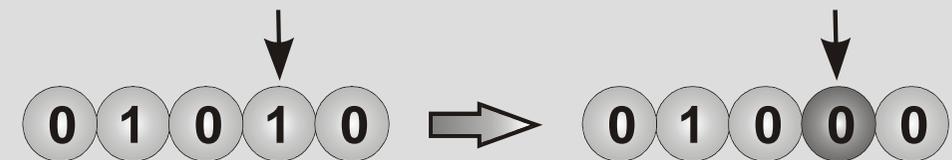


- Variation beruht auf zwei Genetischen Operatoren:

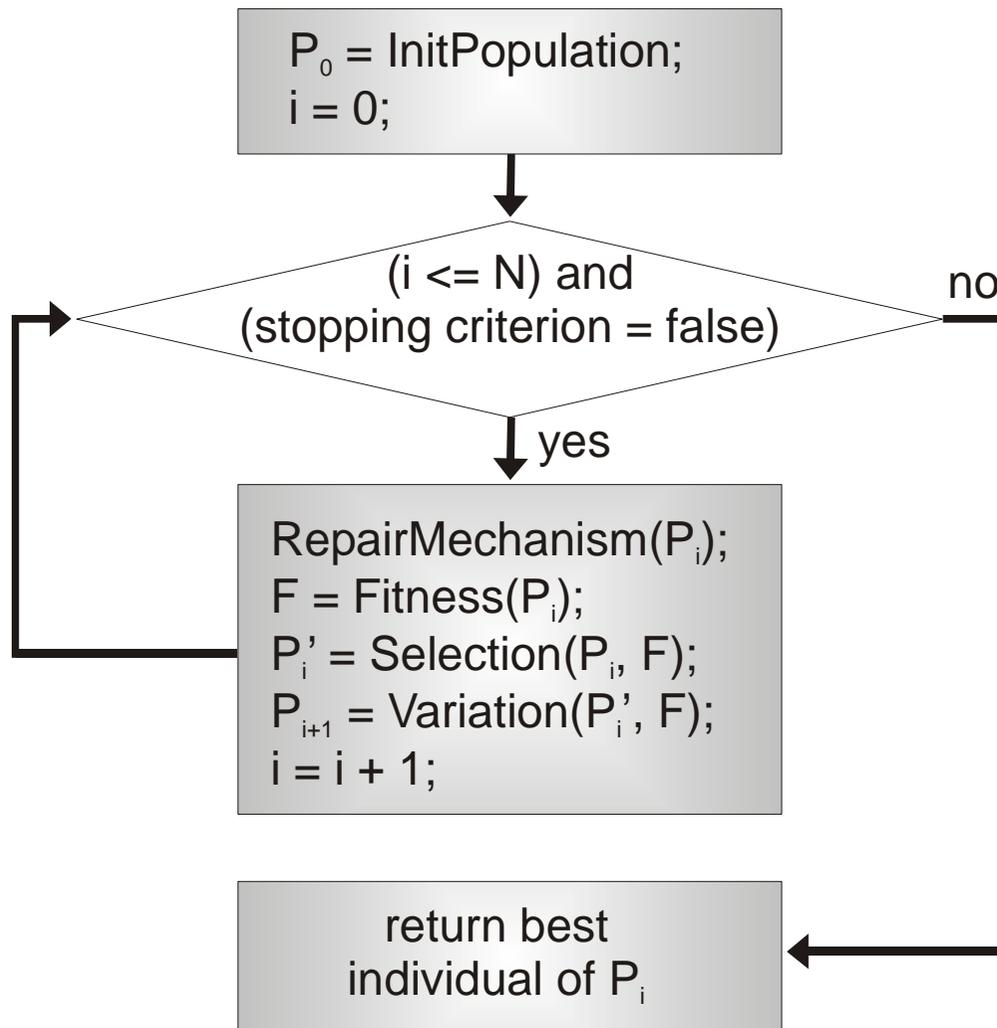
- Crossover:*



- Mutation:*



Ablauf Genetischer Algorithmen (5)



- Wg. zufälliger Variation kann P_i Individuen enthalten, die keine gültige Lösung repräsentieren. Die müssen *repariert* werden.
- Abbruch des GAs, wenn
 - max. X Iterationen erreicht,
 - Beste erreichte Fitness über y Iterationen unverändert,
 - ...
- Rückgabe des Individuums mit bester Fitness aus letzter Population als Endergebnis.



Der GA zur Condition Optimization

■ Chromosomale Repräsentation:

- Bei N verschachtelten Schleifen hat jedes Chromosom N Gene.
- Jedes Gen stellt eine integer-Zahl dar.
- Gen L repräsentiert zu optimierenden Wert $v_{C,L}$
- Wertebereich jedes Gens L beschränkt auf $[l_L, u_L]$.

■ Fitness:

- Die Fitness eines Individuums $\cong IF_{\text{Total}}$
- Ungültige Individuen ($v_{C,1}, \dots, v_{C,N}$) repräsentieren Iterationen, in denen Bedingung C nicht immer erfüllt ist.
- Ungültige Individuen erhalten sehr schlechte Fitness.

Ergebnis von Condition Optimization

- **Eingabe der Condition Optimization:**
 - Lineare Bedingung C
 - Schleifengrenzen $[l_L, u_L]$
- **Ausgabe des Genetischen Algorithmus:**
 - Werte $(v_{C,1}, \dots, v_{C,N})$ des Individuums mit bester Fitness.

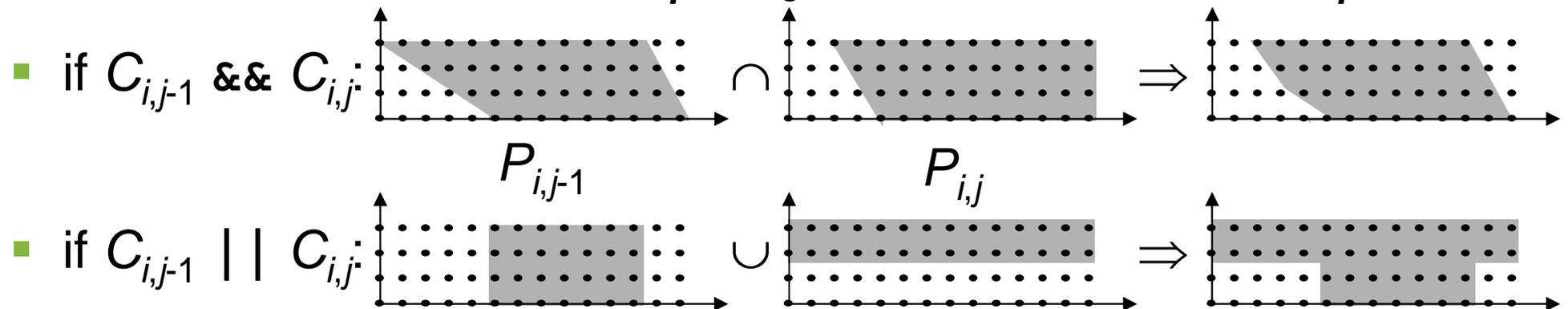
- **Ausgabe der Condition Optimization:**
 - Polytop $P'_C = \{(x_1, \dots, x_N) \in \mathbb{Z}^N \mid$
 \forall Schleifen $L: l_L \leq x_L \leq u_L,$
 $x_L \geq v_{C,L}$ falls $c_L > 0,$
 $x_L \leq v_{C,L}$ falls $c_L < 0 \}$

Phase 3 – Search Space Generation

- **Gegeben:** If-Statements, Bedingungen & Polytope

$$IF_i = (C_{i,1} \otimes C_{i,2} \otimes \dots \otimes C_{i,n}), \otimes \in \{\&\&, ||\} \quad \forall C_{i,j} \rightsquigarrow P_{i,j}$$

- **Konstruktion von EVPs P_i für jedes If-Statement IF_i :**



- **Konstruktion einer Globalen EVP (*Global Search Space*):**

EVP G modelliert Iterationsraum, in dem alle If-Statements wahr sind.

- $G = \cap P_i$

Phase 3 – Struktur der EVP G

- **Konsequenz der Verwendung des \cup Operators:**
 - G ist Endliche Vereinigung von Polyedern
 - $G = R_1 \cup R_2 \cup \dots \cup R_M$
 - Interpretation:
Jedes einzelne Polytop $R_r \rightarrow$
Region im Iterationsraum, für die alle If-Statements wahr sind.

Global Search Space für MPEG-Code

- $IF_1 = 4*x+x^4 < 0 \quad || \quad 4*x+x^4 > 35 \quad || \quad 4*y+y^4 < 0 \quad || \quad 4*y+y^4 > 48$
 $P_{1,1} = \emptyset, P_{1,2} = \{x \geq 9\}, P_{1,3} = \emptyset, P_{1,4} = \{y \geq 13\}$
 $P_1 = \{x \geq 9\} \cup \{y \geq 13\}$
- $IF_2 = 4*x+vx+x^4 < 4 \quad || \quad 4*x+vx+x^4 > 39 \quad || \quad 4*y+vy+y^4 < 4 \quad || \quad 4*y+vy+y^4 > 52$
 $P_{2,1} = \{x = 0 \wedge vx = 0\}, P_{2,2} = \{x \geq 10\},$
 $P_{2,3} = \{y = 0 \wedge vy = 0\}, P_{2,4} = \{y \geq 14\}$
 $P_2 = \{x = 0 \wedge vx = 0\} \cup \{x \geq 10\} \cup \{y = 0 \wedge vy = 0\} \cup \{y \geq 14\}$

- $G = IF_1 \cap IF_2 =$
 $\{x = 0 \wedge vx = 0 \wedge y \geq 13\} \cup \{x \geq 10\} \cup$
 $\{y = 0 \wedge vy = 0 \wedge x \geq 9\} \cup \{y \geq 14\}$

Global Search Space & Splitting-If

- $$G = IF_1 \cap IF_2 =$$

$$\{x = 0 \wedge vx = 0 \wedge y \geq 13\} \cup \{x \geq 10\} \cup$$

$$\{y = 0 \wedge vy = 0 \wedge x \geq 9\} \cup \{y \geq 14\}$$

- Direkte Übersetzung von G in Splitting-If:**

```
if ( ( x == 0 && vx == 0 && y >= 13 ) || ( x >= 10 ) ||
     ( y == 0 && vy == 0 && x >= 9 ) || ( y >= 14 ) )
```

- Keine gute Lösung:**

- ☞ Dieses Splitting-If müsste in $\forall y$ -Schleife plaziert sein (5-innerste!)
- ☞ Führt zu 10.103.760 Ausführungen von If-Statements.

Global Search Space & Splitting-If

- $$G = IF_1 \cap IF_2 =$$

$$\{\mathbf{x} = 0 \wedge \mathbf{v}\mathbf{x} = 0 \wedge \mathbf{y} \geq 13\} \cup \{\mathbf{x} \geq 10\} \cup$$

$$\{\mathbf{y} = 0 \wedge \mathbf{v}\mathbf{y} = 0 \wedge \mathbf{x} \geq 9\} \cup \{\mathbf{y} \geq 14\}$$

- Alternativ:** Nur Teil-Polytope R_r von G für Splitting-If benutzen.
Legal, da jedes Teil-Polytop R_r für sich bereits alle If's erfüllt.

`if (x >= 10)`

- Dieses Teil-Polytop ist auch keine gute Lösung:**

☞ Führt zu 25.401.820 Ausführungen von If-Statements.

`if ((x >= 10) || (y >= 14))`

- Diese Teil-Polytope sind eine gute Lösung:**

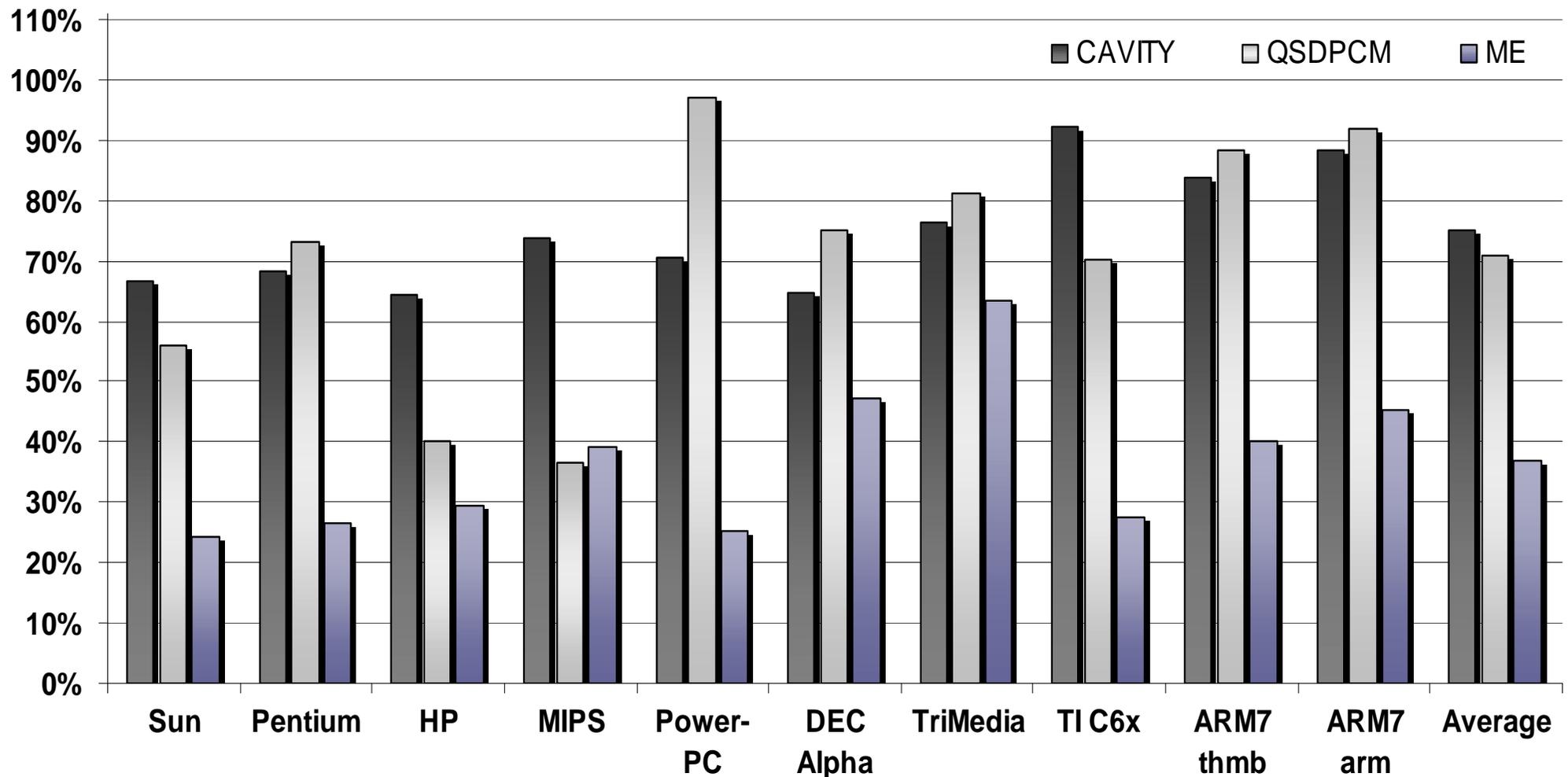
☞ Führen zu 7.261.120 Ausführungen von If-Statements.

Phase 4 – Search Space Exploration

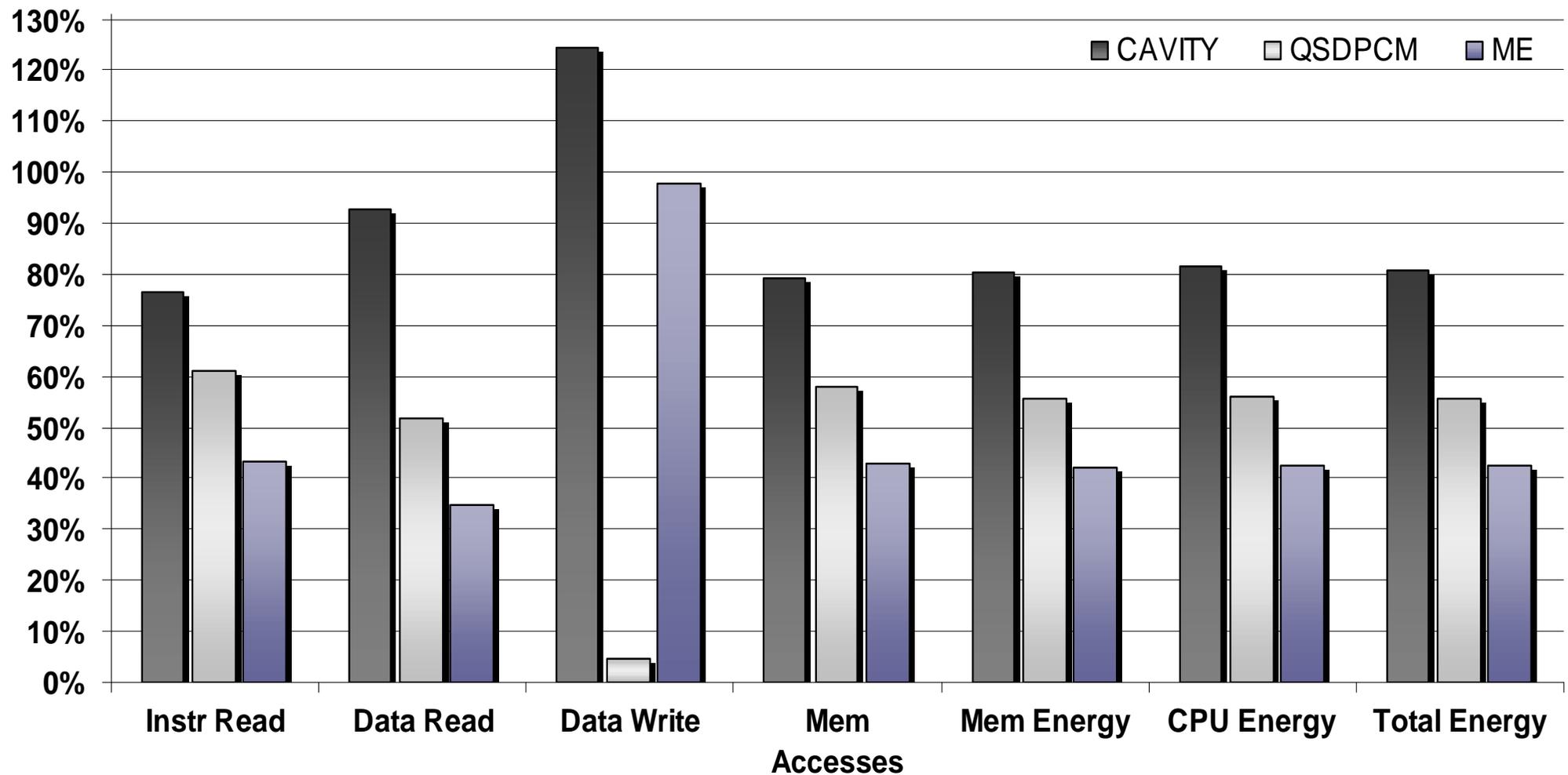
- **GA:** Wählt Regionen aus $G = R_1 \cup R_2 \cup \dots \cup R_M$ aus
 - Individuum: Bitvektor, der die ausgewählten Regionen codiert
Individuum $I = (I_1, \dots, I_M)$, $I_r = \begin{cases} 1 & \text{wenn Region } R_r \text{ gewählt,} \\ 0 & \text{sonst} \end{cases}$
 - Fitness-Funktion berechnet wieder
#{If-Statement Ausführungen} nach Loop Nest Splitting
 - Fitness-Funktion wird durch GA minimiert

- **Resultierendes Splitting If-Statement:**
 - So weit wie möglich in äußerster Schleife plaziert.
 - Enthält alle Bedingungen & Operatoren wie durch die gewählten Regionen R_r vorgegeben.

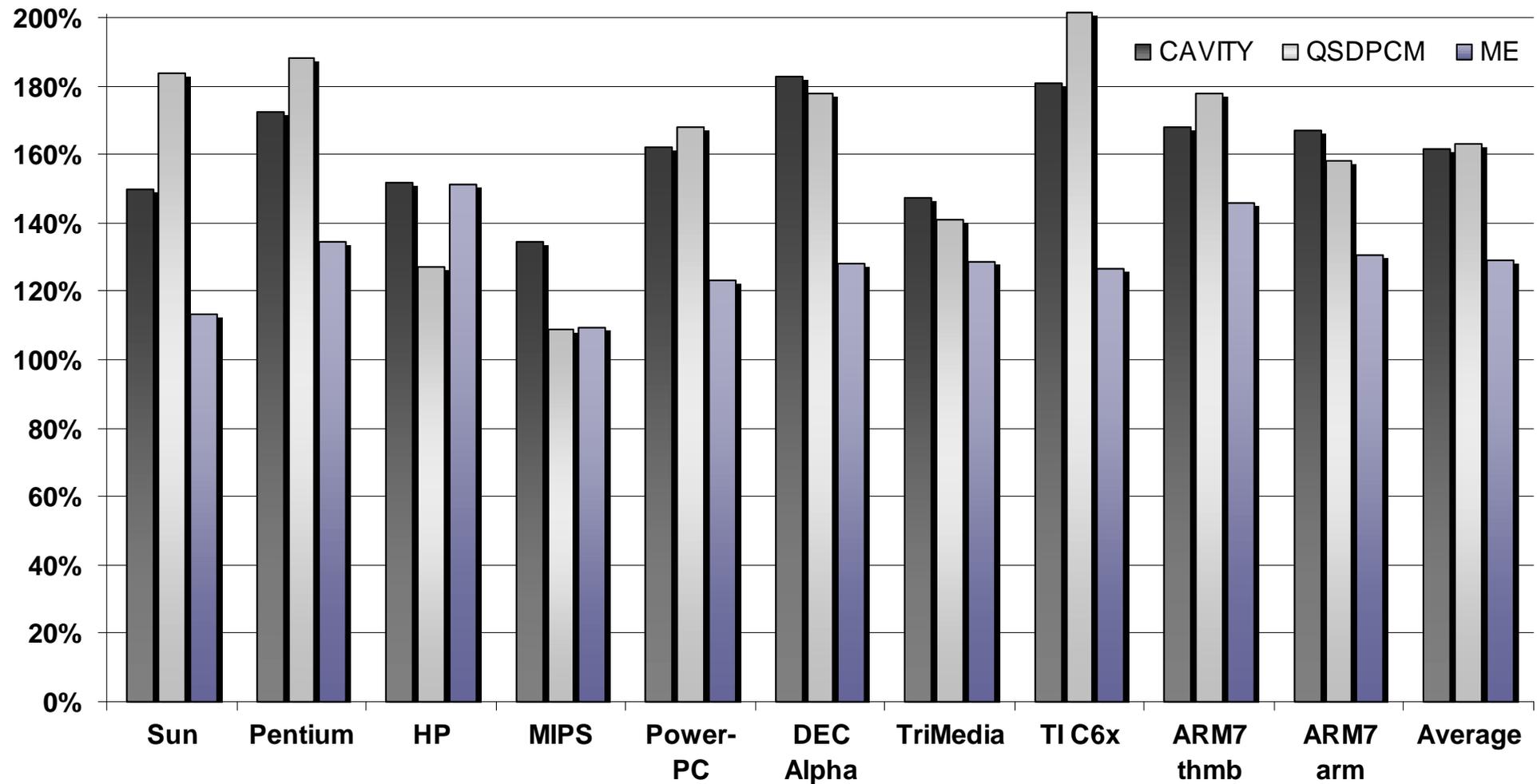
Relative Laufzeiten nach LNS



Relativer Energieverbrauch (ARM7) nach LNS



Relative Codegrösse nach LNS



Literatur

- **Loop Nest Splitting:**
 - H. Falk, P. Marwedel, *Control Flow driven Splitting of Loop Nests at the Source Code Level*, DATE Conference, München, 2003.
 - H. Falk, *Control Flow Optimization by Loop Nest Splitting at the Source Code Level*, Universität Dortmund, Forschungsbericht Nr. 773, Dortmund 2003.

Zusammenfassung

- **Non-Compiler-Optimierungen**
 - Postpass nach Compiler, z.B. auf Linker-Niveau
 - Prepass vor Compiler, auf Quellcode-Niveau
- **Loop Nest Splitting**
 - Kontrollfluß-Optimierung in Datenfluß-dominierten Multimedia-Anwendungen
 - Polytope zur Modellierung linearer Bedingungen & Schleifen
 - Genetische Algorithmen zur Optimierung der Polytop-Modelle
 - Enorme Einsparungen in ACET und Energie (*und WCET*), aber z.T. beträchtliche Code-Vergrößerung