Karlsruher Institut für Technologie

Improved integer conversion optimisation and VHDL code generation based on bit-width analysis

Marcel Hollerbach

Lehrstuhl Programmierparadigmen, IPD Snelting

Das Ziel

```
int arr[4];

for (int i = 0; i < 4; i++) {

        int x = i*i;
        int y = (i << 4);
        int res = x + y;
        arr[i] = res;
}
```

Wie viele Bits nutzen *i*, *x*, *y*, *res*?

Was ist eine Bitbreite?

Karlsruher Institut für Technologie

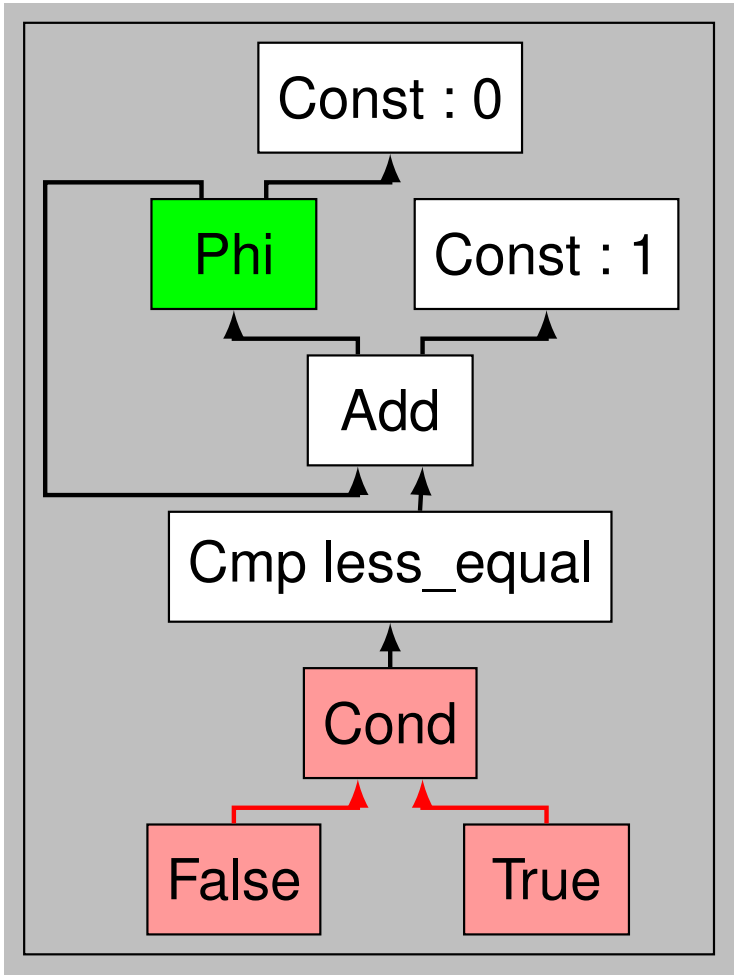
Definition

Bitbreite (stable_bits, is_positive)

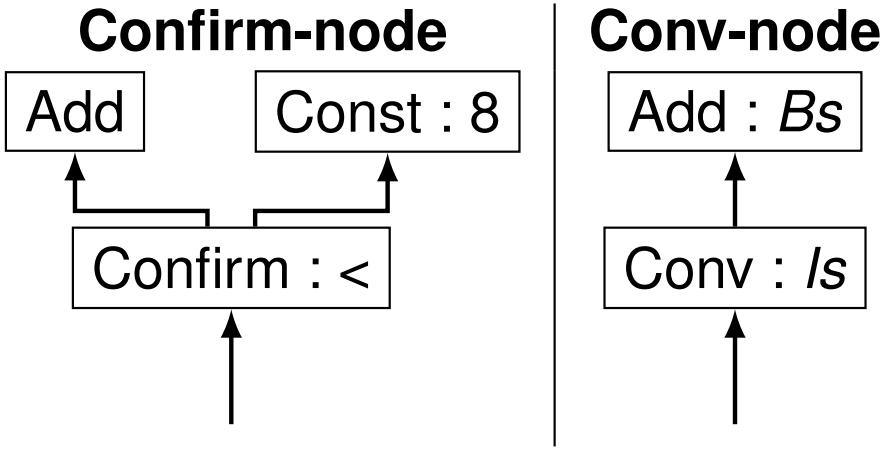
mode	value	bit representation								stable_bits	is_positive
Bs	5	0	0	0	0	0	1	0	1	4	true
Bs	-2	1	1	1	1	1	1	1	0	6	false
Bu	5	0	0	0	0	0	1	0	1	5	-

libFIRM

1. Integer-modes sind genutzt
2. Nutzung von arithmetische/logische Ausdrücke
3. Behandlung von Confirm-nodes



libFIRM



Analyse

Implementiert mit worklist appraoch.

op	stable bits
add	$max(min(a_{stable}, b_{stable}) - 1, 0)$
minus	$max(a_{stable} - 1, 0)$
sub	$max(min(a_{stable}, b_{stable}) - 1, 0)$
mul	$max(2 * mode - (b_{stable} + a_{stable}), 0)$
div	$\begin{cases} a_{stable} & , \neg mode \text{ is signed} \\ max(a_{stable} - 1, 0) & , \text{otherwise.} \end{cases}$
mod	$\lfloor log2(max(a)) \rfloor$

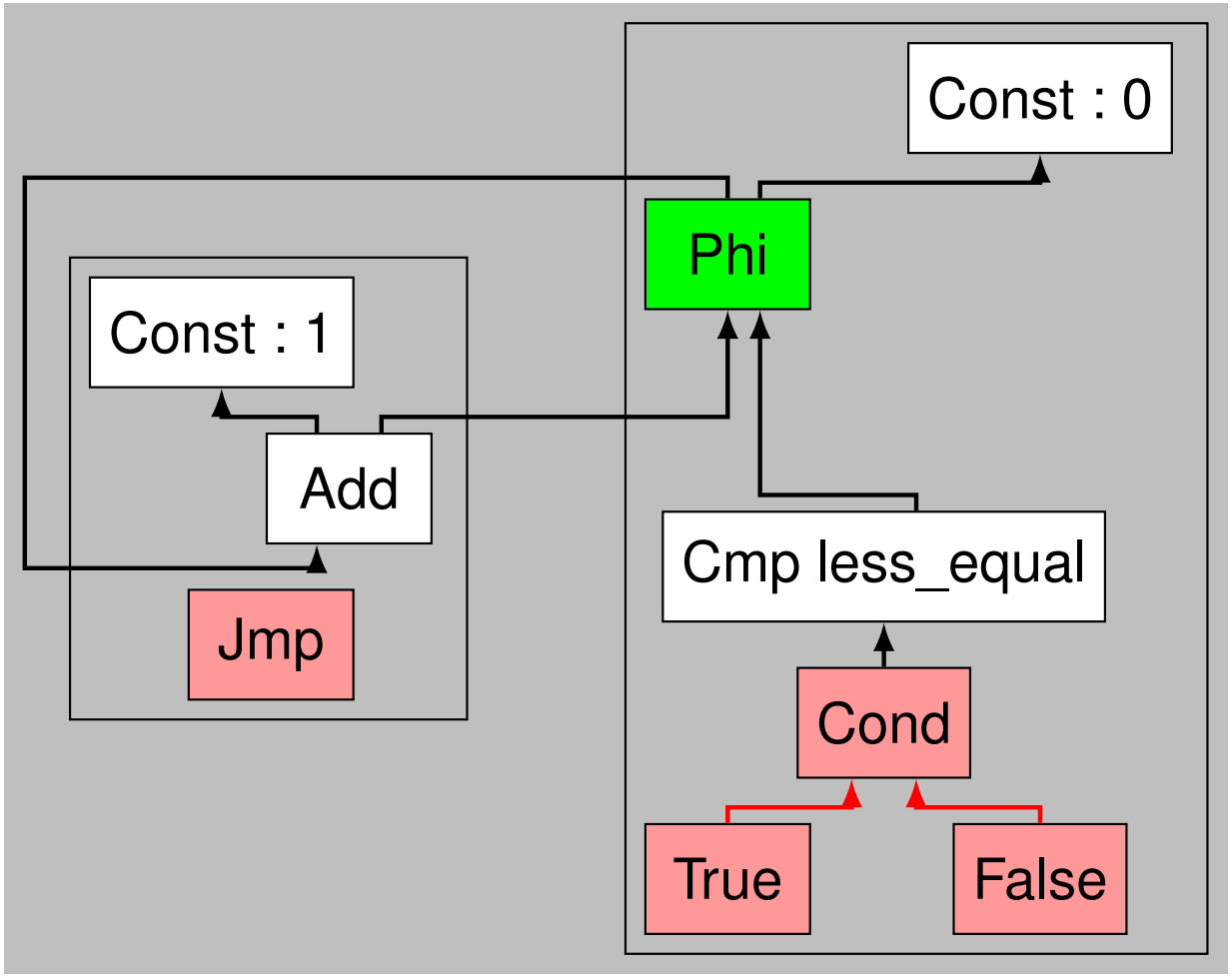
Analyse 2

$$add(const(5), x) \equiv max(min(4, 5) - 1, 0) \equiv 3$$

stable_bits	min	max
5	-4	+3
4	-8	+7
3	-16	+15

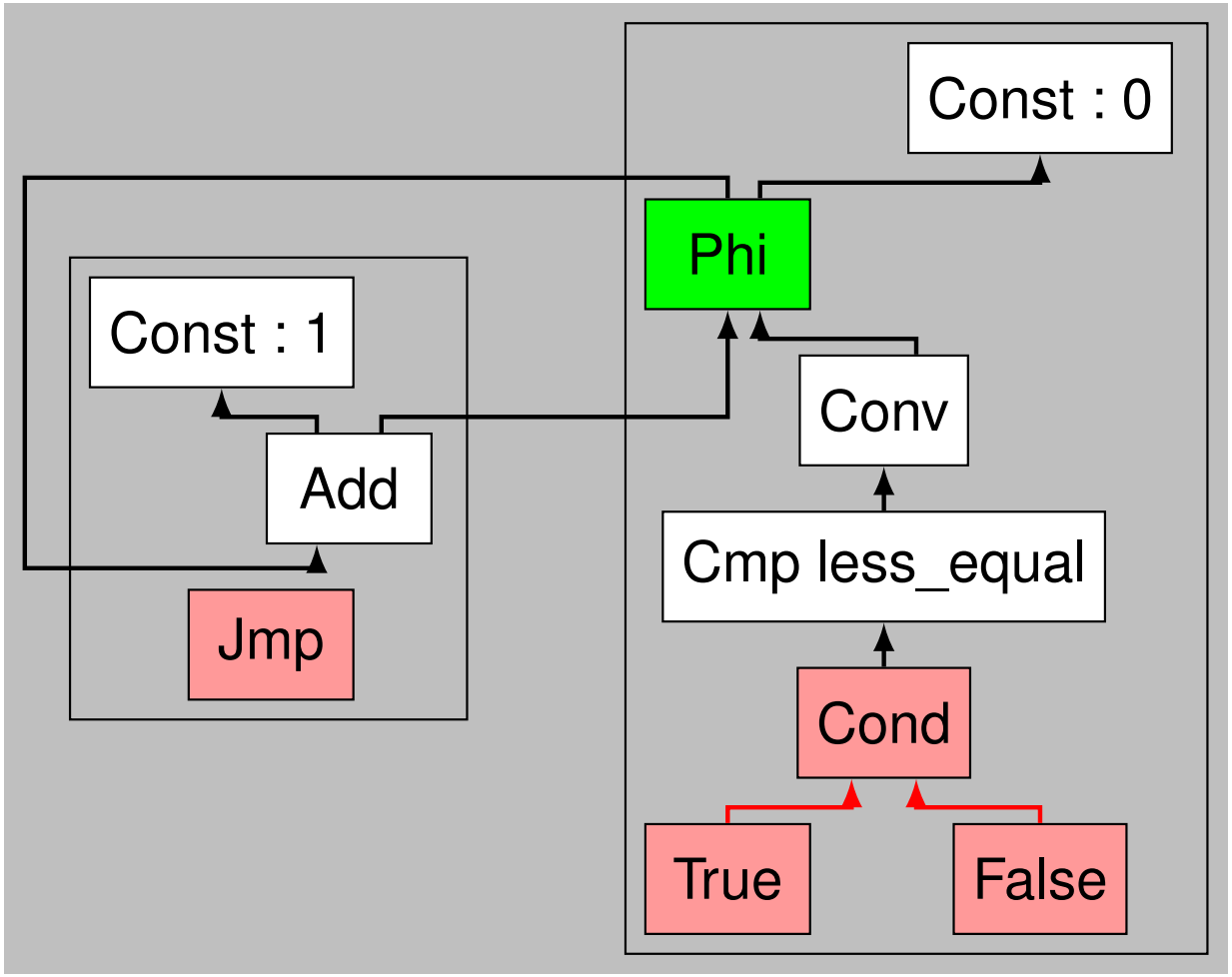
Schleifen im Graphen

Mit möglichem Confirm-node:



Schleifen im Graphen

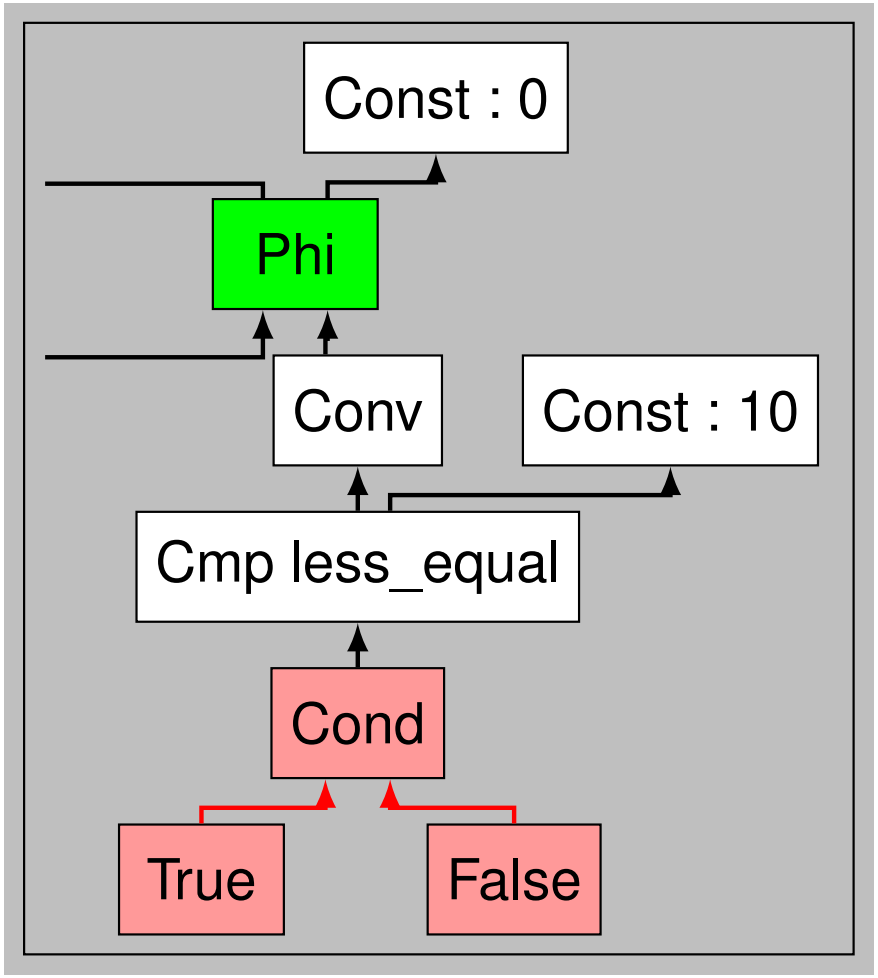
Ohne möglichen Confirm-node:



Zusätzliche Confirm-nodes

Stable bits von add:

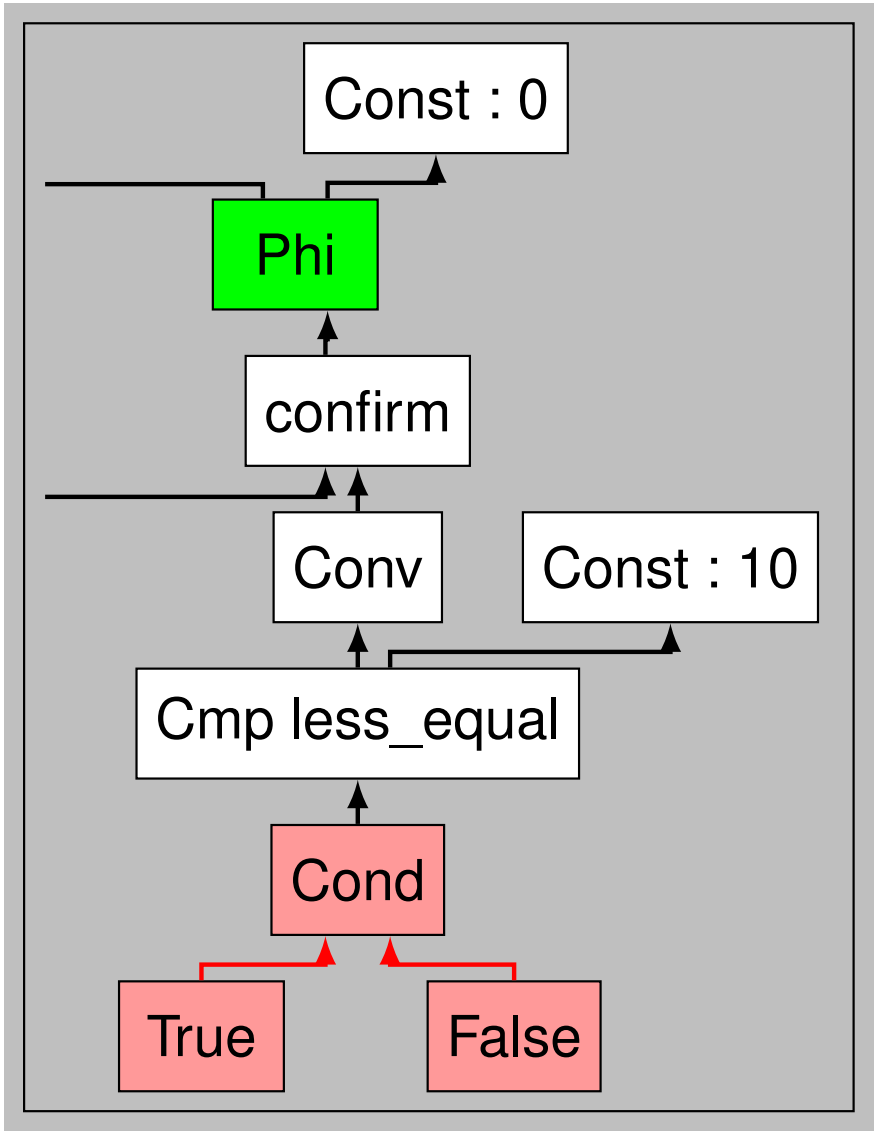
1. (32, true)
2. (30, true)
3. (29, true)
4. (28, true) [7,-8]
5. (27, true) [15,-16] -> insert Confirms



Zusätzliche Confirm-nodes

Stable bits von add:

1. (32, true)
2. (30, true)
3. (29, true)
4. (28, true) [7,-8]
5. (27, true) [15,-16] -> insert Confirms



Zusätzliche Confirm-nodes (Beispiel)

```
unsinged int l = 20;

for (int i = 0; i < l; i++) {
  data[i] = calculate(i);
}
```

Verwendungen der Analyse

1. Conversion-node Entfernung (Cmp)
2. VHDL Generierung

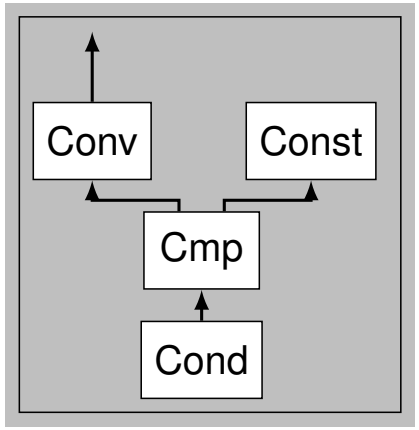
Conversion-node Entfernung (Cmp)

Idee:

1. Entfernen von Conv-node
2. Unterstützung anderer Analysen

Voraussetzung:

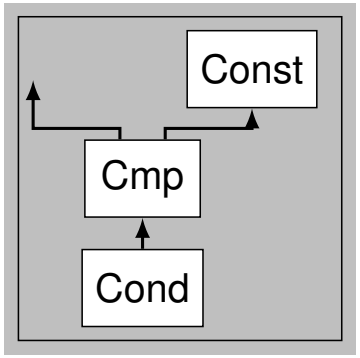
1. Cmp-node mit Const-node und Conv-node
2. Used bits von Conv-node muss unverändert sein



Conversion-node Entfernung (Cmp)

Optimierung:

1. Conv-node entfernen
2. Mode von Const anpassen



Evaluierung der Optimierung

Verbesserte Bitbreite:

Library	mode usage(1)	bitwidth usage(1)
zlib	40.6729	31.4571
libjpeg	40.7650	27.8380
libpng	32.7320	23.5180
libgif	36.5323	27.7742
libtiff	37.1373	27.6623

Erfahrungen:

1. Loop-Unroller verbessert sich
2. Geringe Auswirkung auf die Assembler Generierung

VHDL Generierung

Vorgehen:

1. C Quellcode kompilieren zu IR Format
2. Kompiliere IR Format zu VHDL

Beispielcode:

```
variable node199 : signed(31 downto 0):= (others => '0');
variable node194 : signed(31 downto 0):= (others => '0');
variable node193 : unsigned(31 downto 0):= (others => '0
[...]
node199 := node172 xor node198;
node194 := node199 + node193;
node193 := unsigned (resize(node194,32));
```

VHDL Generierung

Idee:

1. Nutzung der minimalen Variablenbreite
2. Dadurch verminderte LUT Anzahl

Beispielcode:

```
variable node199 : signed(9 downto 0):= (others => '0');
variable node194 : signed(16 downto 0):= (others => '0');
variable node193 : unsigned(31 downto 0):= (others => '0
[...]
node199 := resize(node172,10) xor resize(node19,10);
node194 := resize(node199,17) + resize(node193,17);
node193 := resize(unsigned(resize(node194,32)),32);
```

VHDL Generierung (Evaluation)

Karlsruher Institut für Technologie

Vorgehen der Evaluation:

1. Übersetzen von C Quellcode zu VHDL ohne Optimierung
2. Nochmaliges Übersetzen mit Optimierung
3. Vergleich der beiden Ergebnisse

VHDL Generierung (Evaluation)

IDE	Unoptimiert	Optimiert	Hand optimiert
Vivado	108	112	109
Quartus	54	54	-

Testcode:

	Unoptimiert	Optimiert
Genutzte Bitbreite	928	609

Verbesserungen

Analyse:

1. Nicht terminierende Schleifen gleich erkennen
2. Genauere Berechnung für arithmetische Ketten

Optimierung:

1. Weniger redundanten VHDL code erzeugen
2. Conv-nodes einfügen anstatt löschen
3. Kleinste mögliche Anzahl an Bits finden

Ende