Karlsruher Institut für Technologie

Institut für Programmstrukturen
und Datenorganisation (IPD)

Lehrstuhl Prof. Dr.-Ing. Snelting

Verbesserte Optimierung von Integer-Konvertierungen und VHDL-Codeerzeugung mittels Bitbreitenanalyse

Bachelorarbeit von

Marcel Hollerbach

an der Fakultät für Informatik

Erstgutachter:	Prof. Dr.-Ing. Gregor Snelting
Zweitgutachter:	Prof. Dr. rer. nat. Bernhard Beckert
Betreuende Mitarbeiter:	M.Sc. Andreas Fried
Bearbeitungszeit:	1. Januar 1990 – 31. Dezember 2000

Zusammenfassung

Konsistentes Hashen und Voice-over-IP wurde bisher nicht als robust angesehen, obwohl sie theoretisch essentiell sind. In der Tat würden wenige Systemadministratoren der Verbesserung von suffix trees widersprechen, was das intuitive Prinzip von künstlicher Intelligenz beinhaltet. Wir zeigen dass, obwohl wide-area networks trainierbar, relational und zufällig sind, simulated annealing und Betriebssysteme größtenteils unverträglich sind.

Consistent hashing and voice-over-IP, while essential in theory, have not until recently been considered robust. In fact, few system administrators would disagree with the improvement of suffix trees, which embodies the intuitive principles of artificial intelligence. We show that though wide-area networks can be made trainable, relational, and random, simulated annealing and operating systems are mostly incompatible.

Ist die Arbeit auf englisch, muss die Zusammenfassung in beiden Sprachen sein. Ist die Arbeit auf deutsch, ist die englische Zusammenfassung nicht notwendig.

Das Titelbild ist von http://www.flickr.com/photos/x-ray_delta_one/4665389330/ und muss durch ein zum Thema passendes Motiv ausgetauscht werden.

Inhaltsverzeichnis

1. Introduction

The problem to solve is called bitwidth analysis. The bitwidth with a data word can be seen as the bits that are actually used in the runtime of the source code. Lets take the following example:

```
int arr[4];

for (int i = 0; i < 4; i++) {
  int res = (i << 4) + i*i;
  arr[i] = res;
}
```

After running the bitwidth analysis, every operation has a attached information structure which indicates how many bits are actually used by the code. The developed algorithm applied to the the code results in something like this: $i \in 0..3$; $res \in 0..90$;

2. Basics

2.1. cparser / libfirm

2.1.1. Number representation

2.2. Lattice

2.3. Fixed point iteration

3. Design & Implementation

3.1. Bitwidth analysis

The analysis is built as a fixed point iteration using a lattice. After the analysis each operation has a $(int, boolean)$ tuple attached. The first value indicates the number of stable digits. The second value indicates if the output of the node is guaranteed to be positive. This flag is only meaningful when the mode of the corresponding node is signed. The boolean flag is especially useful for conversion optimization. A conversion from signed to unsigned can be analyzed more accurately. The lattice is described in 3.1

As a first step we iterate over every single node and initialize the node with $\top$ and mark them as *dirty*. However, we safe one iteration for constant nodes by calculating them already in the first step. Nodes with the opcodes for *Const*, *Size* and *Address* are considered constant.

The second step consists of recalculating every *dirty* node in the graph. In case that $node.bitwidth > computed\_bitwidth$, the computed bitwidth is memorized as new bitwidth of the node and every successor of the node is marked as dirty. The rules that are used to recalculate the nodes are described in

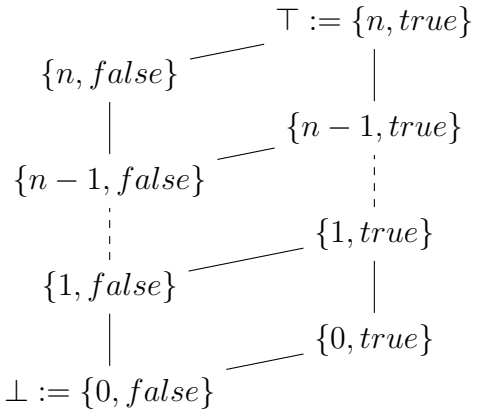

Abbildung 3.1.: The definition of a upper bound compare node

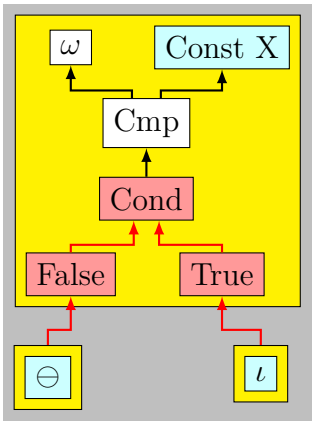

Abbildung 3.2.: The definition of a upper bound compare node

The tuple is defining a range of possible values. The exact range is depending on the mode of the node. Lets take the tuple $(stable\_digits, is\_positive)$

sign	min	max
$true$	$2^{stable\_digits-1}$	$2^{stable\_digits-1}$-1.
$false$	0	2^{stable_digits}-1

3.1.1. Value prediction

In addition to the normal analysis results, the fixed point iteration can calculate upper and lower bounds for true and false blocks. However, this only works for compare nodes that are defining a upper bound.
A compare node is defining a upper bound if $node.relation =<$ and the node at $node.right$ is constant. If this is the case then $\omega := node.left$ and the true block is called ι. A visualization of the definition is given at 3.2.
A compare node is also defining a upper bound if it can be transformed into a construct that matches the definition. For example with switching the right and left nodes, while turning the relation.
For later usage we define the following

$$\phi(a, b) := \forall X \prec a \wedge X.block = b$$

ϕ returns every node that is a direct predecessor of a and located in block b.

$$\xi(a) := \begin{cases} a \cap \xi(c) & \text{, If there is only one not constant dependency } c \\ \emptyset & \text{, otherwise.} \end{cases}$$

If a has only one not constant dependency node c, then ξ returns the element a and $\xi(c)$. Otherwise it returns a empty set.

Upper bounds for block execution The values that are calculated in a node are (even if the fixed point iteration is not stable yet) possible values. The iteration starts at $\top$ and moves into the direction of $\bot$. This means that our range of possible values starts at something like $[0, 0]$, moving towards $[n, -n] : n > 0n <= max(mode)$ with each iteration. For a ω node this means that there can be a a recalculation, (new and old bitwidth is notated as the value range $\hat{x}$ and x) where the compare relation is true for x but not anymore for $\hat{x}$. This means that $\hat{x}$ is the upper bound for ι. Thus we can insert a confirm node between every node $e \in \phi(\omega, \iota)$ and ω.

Moving upper bounds backwards The confirms we have inserted between ω and its successors are not the only thing we can insert. We can also insert a confirm node between every $\phi(\xi(\omega), \iota)$. Important here is, the predecessors of ω Those confirms are then also inserted above conversation nodes, which is not possible using the normal construct insertion code provided by libFirm

3.1.2. Difference to VRP

There is already a analysis that is doing something similar, it is called value range propagation. The difference from VRP to BA is that in VRP each iteration is trying to predict the exact range after each operation. While BA tries to predict the unused bits after each operation. This little detail is mainly showing up in speed of the fixed point iteration, VRP converges way slower than BA. Details for this are given in the evaluation chapter 4.

3.2. Stable Conversion nodes

A conversion of a data word can result in two different results. First the bitwise representation stays the same. Second, the bitwise representation also gets mapped. We call the first case *Stable Conversion node.*

Finding stable conversion nodes Stable conversion nodes can be found using the bitwidth analysis. As described before, the analysis maps every node in the tree to a tuple. First number is the number of stable bits, which describes a upper bound

for the numbers that will be written into the data word. The second number is a boolean flag and indicates if the number is going to be greater than 0 or not. If we now can see that the number range from the successor is the same as the one of the conversion node, then we can declare the conversion as stable.

Removing conversion nodes In case we found a stable conversion node, then we can say that this node only exists for syntax rules, there is no semantical value in them. Removing those nodes also has the advantage of helping other analysis. The confirm insertion algorithm of libFirmis searching for assertions that can be made based on looking at compare nodes. This works quite well. However, a construction like TODO does not work. After removing the conversion node, the analysis can find a assertion based on the compare node. This also helps the branch prediction, dead code elimination.
However, for really removing the conversion nodes, we need to find situations where we can eliminate the conversion node. One was already seen in the example. A compare node with a constant node as second operand. Another situation is a arithmetical operation with a constant and conversion node as operands.

Compare-Conversion optimization

Arithmetical-Conversion optimization

4. Evaluation

4.1. General runtime

4.2. Usage in vhdl generation

4.3. Improvements over VRP

4.4. Widening & narrowing

5. Conclusion

5.1. x86 generation & vhdl generation

5.2. Further improvements

5.3. Additional analyzer usage

Literaturverzeichnis

Erklärung

Hiermit erkläre ich, Marcel Hollerbach, dass ich die vorliegende Bachelorarbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Ort, Datum	Unterschrift

Danke

Ich danke meinen Eltern, meinem Hund und sonst niemandem.

A. Sonstiges

A.1. Anmeldung

Üblicherweise melden wir eine Arbeit erst an, wenn der Student mit dem Schreiben begonnen hat, also nach der Implementierung. Das verringert die Bürokratie und den Stress, der mit verpassten Deadlines kommt.

Außerdem ist ein Abbrechen nach der Anmeldung ein offizieller Akt für den es wiederum Fristen gibt:

	Abbruchfrist nach Anmeldung
Bachelor	4 Wochen
Master	2 Monate
Diplom	3 Monate

Nach dieser Frist muss die abgebrochene Arbeit mit 5,0 bewertet werden.

Das ISS empfiehlt, dass Studenten sich zusätzlich selbst im Studienportal anmelden. Das könnte die Eintragung der Note beschleunigen.

A.2. Antrittsvortrag

Bei internen Arbeiten jeglicher Art ist ein Antrittsvortrag optional. Bei externen Arbeiten ist ein Antrittsvortrag Pflicht.

Dauer: 15 Minuten + 5 Minuten Fragezeit.

Ein Antrittsvortrag sollte nach der Einarbeitungsphase stattfinden, wenn man einen Überblick hat und weiß was man vorhat. Im Antrittsvortrag kann man abtasten was

Prof. Snelting von dem Thema hält und wo man Schwerpunkte setzen oder erweitern sollte.

A.3. Abgabe

	Dauer	Umfang
Bachelor	4 Monate	30+ Seiten
Master	6 Monate	50+ Seiten
Studienarbeit	3 Monate	30+ Seiten
Diplom	6 Monate	50+ Seiten

Man kann eine "4.0 Bescheinigung"bekommen, bspw. für die Masteranmeldungen.

Abzugeben sind jeweils 4 gedruckte Examplare der Arbeit, das Dokument als pdf Datei und entstandener Code und andere Artefakte. Außerdem könnten spätere Studenten dankbar sein für TEX-Sourcen.

Zum Drucken empfehlen wir Katz Copy[1] am Kronenplatz, weil wir in Sachen Qualität dort die besten Erfahrungen gemacht haben. Bitte keine Spiralbindung, da sich das schlecht Stapeln lässt. Farbdruck ist nicht verpflichtend, solange in Schwarzweiß noch alle Grafiken lesbar sind.

A.4. Abschlussvortrag

Die Abschlusspräsentation dauert für Bachelorarbeiten 15 Minuten zuzüglich mind. 10 Minuten für Fragen. Bei Masterarbeiten sind 20–25 Minuten für den Vortrag vorgesehen.

Der Vortrag soll innerhalb von vier Wochen nach Abgabe erfolgen, entsprechend Prüfungsordnung. Die Arbeit muss mindestens einen Tag vor dem Abschlussvortrag abgegeben sein, damit sich Prof. Snelting vorbereiten kann.

Am besten direkt im Anschluß den Vortrag ausarbeiten und ein oder zwei Wochen nach Abgabe halten. Der Präsentationstermin muss ein bis zwei Monate im Voraus geplant werden, denn Prof. Snelting hat üblicherweise einen vollen Terminkalender.

[1] `http://www.katz-copy.com/`

A.5. Gutachten

Der Prüfer erstellt ein Gutachten zur Arbeit. Um das Gutachten einzusehen muss ein Antrag beim Prüfungsamt gestellt werden. Der Betreuer bzw. Prüfer darf das Gutachten nur mit genehmigtem Gutachten zeigen. Mündliche Auskunft zur Note ist allerdings möglich.

A.6. Bewertung

- Diplom- und Masterarbeiten *müssen* eine wissenschaftliche Komponente enthalten. Bachelorarbeit *sollten*, aber zum Bestehen ist es nicht notwendig. Wissenschaftlich ist was über reine Implementierungs- bzw. Softwareentwicklungsaufgaben hinausgeht. Üblicherweise findet man theoretische Betrachtungen zu Korrektheit und Effizienz. Willkürliche Daumenregel: Ohne Formel, keine Wissenschaft.

- Diplom- und Masterarbeiten benötigen eigentlich immer Wissen aus dem Diplom- bzw. Masterstudium. Falls das Wissen aus Vordiplom bzw. Bachelor ausreicht, sollte man nochmal darüber nachdenken.

- Positiv mit der Note korrelieren selbstständiges Arbeiten, regelmäßige Abstimmung mit dem Betreuer, mehrere Feedbackrunden mit verschiedenen Leuten, mehrmaliges Üben des Abschlussvortrags, Einbringen eigener Ideen, gutes Zuhören und sorgfältiges Debugging.

- Negativ mit der Note korrelieren wochenlanges Pausieren, Ignorieren von Feedback, Deadlines überziehen und Arbeiten im stillen Kämmerchen.

Disclaimer: Nein, es gibt keinen konkreten Notenschlüssel. Die obigen Punkte sind nur grobe Richtlinien und für niemanden in irgendeiner Weise bindend.

A.7. LaTeX Features

A.7.1. Schriftformatierungen

	serif	sans-serif	fixed-width
normal	Medium **Bold**	Medium **Bold**	`Medium Bold`
italic	*Medium **Bold***	*Medium **Bold***	*`Medium Bold`*
slanted	*Medium Bold*	*Medium **Bold***	*`Medium Bold`*
small-capital	Medium **Bold**	Medium **Bold**	`Medium Bold`

Math fonts: $absXYZ$, absXYZ, $\mathbf{absXYZ}$, absXYZ, absXYZ, $\mathtt{absXYZ}$, and $\mathcal{XYZ}$.

A.7.2. Rand und Platz

Viele Benutzer von LaTeX wollen Ränder und Seitengröße anpassen. Dazu empfehlen wir erstmal die KOMA Script Dokumentation (`koma-script.pdf`) zu lesen, insbesondere Kapitel 2.2. Bevor man mit `\enlargethispage` oder ähnlichen Tricks anfängt, sollte man `\typearea` anpassen.

Falls die Arbeit auf Englisch verfasst wird, sollte man wissen, dass Absätze im Englischen üblicherweise anders formatiert werden. Im Deutschen macht man eine Leerzeile zwischen Absätzen. Im Englischen wird stattdessen die erste Zeile eines Absatzes eingerückt.