Motivation: Von Verifikation über Analyse zur Transformation – Optimierung am Beispiel “code motion”-basierter Transformationen (Teil 1)

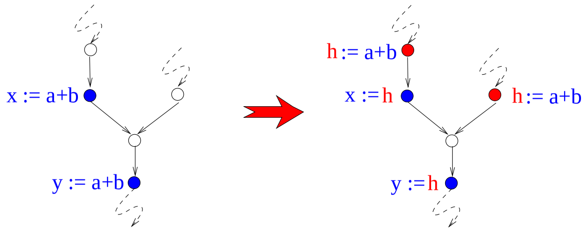
Analyse und Verifikation (WS 2005/2006) / 5. Teil (21.11.2005)

Jens Knoop

Technische Universität Wien

CM – What’s it all about?

...essentially, CM aims at avoiding recomputing values



Why Considering Code Motion (CM)? (1)

...because it is

- Relevant ...widely used in practice
- General ...a family of optimizations rather than a single one
- Well-understood ...manually proven correct and optimal
- Challenging ...conceptually simple, but exhibits lots of thought-provoking phenomena

Why Considering Code Motion (CM)? (2)

Last but not least, it is...

- Truly classical ...has a long history
  - Morel, E. and Renvoise, C. *Global Optimization by Suppression of Partial Redundancies.* CACM 22 (2), 96 - 103, 1979.
  - Ershov, A. P. *On Programming of Arithmetic Operations.* CACM 1 (8), 3 - 6, 1958.

Conceptually

...code motion can be considered a two-stage process

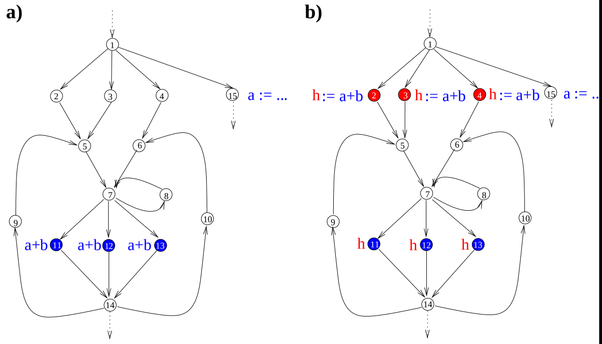
1. Expression hoisting

   ...hoisting expressions to “earlier” safe computation points

2. Total redundancy elimination

   ...eliminating computations which became totally redundant

The Effect of CM applied to a More Complex Example



Correctness, Optimality

We can prove...

- Correctness

  Theorem [“Essence”]

  ...at every use site of a temporary, recomputing the expression yields the same value which is stored in the temporary

- Optimality

  Theorem [Earliestness principle]

  ...hoisting expressions to their earliest safe computation points leads to computationally optimal programs

  $\leadsto$ ...known as Busy Code Motion (PLDI’92, Knoop et al.)

Note

Traditionally,

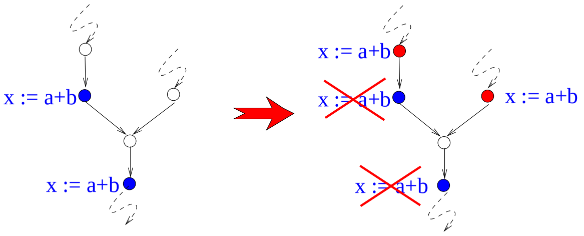
- Code (C) means expressions
- Motion (M) means hoisting

But...

- CM is more than hoisting of expressions and PR(E)E!

For example, code...

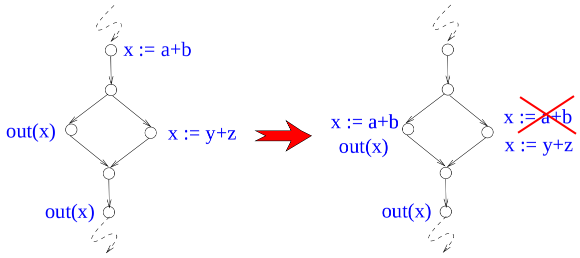
...can be assignments, too.



- Here, CM means partially redundant assignment elimination (PRAE)

In contrast to expressions, assignments...

...might also be sunk.



- Now, CM means partially dead code elimination (PDCE)

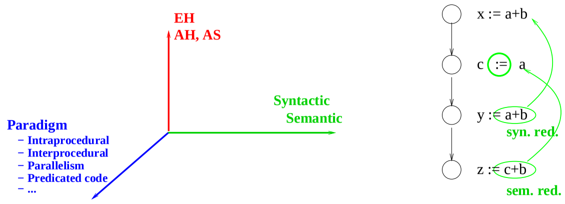
Towards the Design Space of CM-Algorithms...

More generally...

- Code means expressions/assignments
- Motion means hoisting/sinking

Code / Motion	Hoisting	Sinking
Expressions	EH	·/·
Assignments	AH	AS

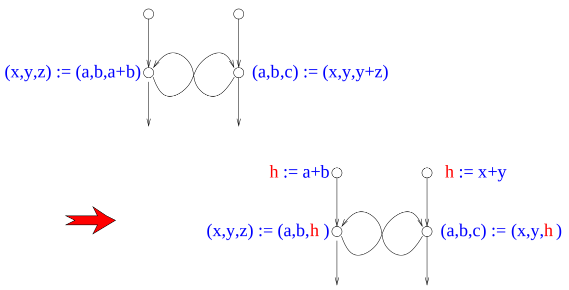
Refining the Design Space of CM-Algorithms Further...



Introducing semantics... !

Semantic Code Motion...

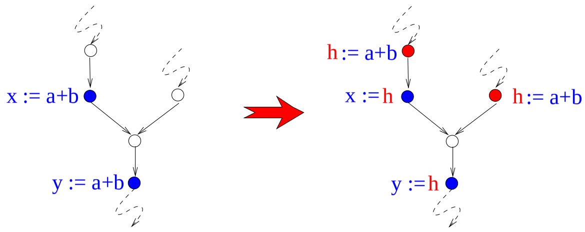
allows more powerful optimizations!



(example by B. Steffen, TAPSOFT'87)

In the following...

...we first focus on (syntactic) PRE(E), the basic variant of CM-based program optimizations.



...while taking New Challenges of Program Optimization into Account

...there is more than speed!

1999 World Market for Microprocessors

Chip Category	Number Sold
Embedded 4-bit	2000 million
Embedded 8-bit	4700 million
Embedded 16-bit	700 million
Embedded 32-bit	400 million
DSP	600 million
Desktop 32/64-bit	150 million

... David Tennenhouse (Intel Director of Research). Keynote Speech at the *20th IEEE Real-Time Systems Symposium* (*RTSS'99*), Phoenix, Arizona, December 1999.

1999 World Market for Microprocessors

Chip Category	Number Sold
Embedded 4-bit	2000 million
Embedded 8-bit	4700 million
Embedded 16-bit	700 million
Embedded 32-bit	400 million
DSP	600 million
Desktop 32/64-bit	150 million

$\sim$ 2%

... David Tennenhouse (Intel Director of Research). Keynote Speech at the *20th IEEE Real-Time Systems Symposium* (*RTSS'99*), Phoenix, Arizona, December 1999.

Think of...

... domain-specific processors as used in embedded systems

- **Telecom**
  - Cell phones, pagers, ...
- **Consumer Electronics**
  - MP3 player, cameras, pocket games, ...
- **Automative**
  - GPS navigation, airbags, ...
- ...

Code for Embedded Systems

Requirements...

- Performance (often real-time constraints)
- Code size (system-on-chip, on-chip RAM/ROM)
- ...

For embedded systems...

...code size is often more critical than speed!

Code for Embedded Systems (Cont'd)

Requirements ...and how they are commonly addressed:

- Assembly programming
- Manual postpass optimizations

Shortcomings...

- Error-prone
- Extended time-to-market

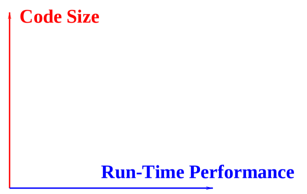
... problems getting even worse with increasing complexity.

Hence, we observe...

...a trend towards HLL programming (C/C++)

Given this trend...

...how does traditional compiler and optimizer technology support the specific requirement profile of code for embedded systems?



Unfortunately, little.

As a Matter of Fact...

Traditional optimizations...

- ...are strongly biased towards performance optimization
- ...are not code-size sensitive and usually don't offer any control about their impact on code size

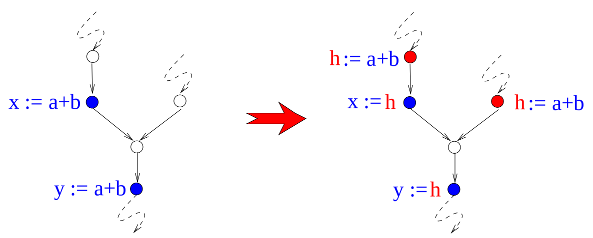
In the following, we demonstrate this considering

- **Partial Redundancy (Expression) Elimination (PR(E)E)**

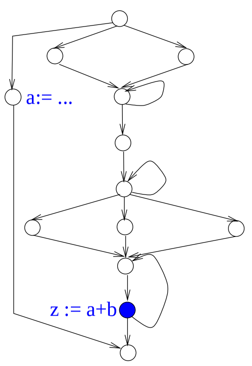
as example.

Recall the Essence of PRE

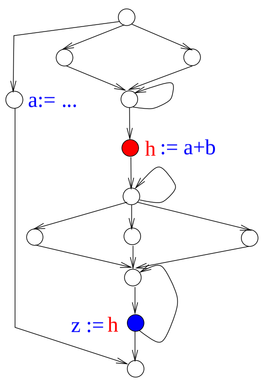
...essentially, PRE aims at avoiding recomputing values



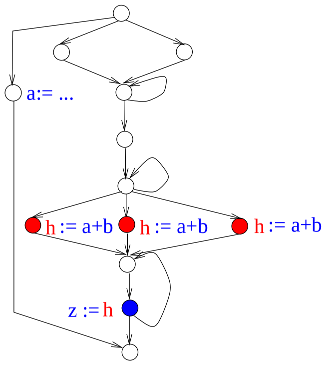
PRE – especially profitable if loops are involved...



Of course, we would like to have...



What we receive using a state-of-the-art compiler...



As we have observed already: Conceptually...

PRE can be considered a two-stage process...

1. Expression hoisting

   ...hoisting expressions to "earlier" safe computation points

2. Total redundancy elimination

   ...eliminating computations becoming totally redundant

Extreme Strategy – Earliestness Principle

Placing computations as early as possible...

- Theorem [Computational Optimality]

  ...hoisting expressions to their earliest safe computation points yields computationally optimal programs

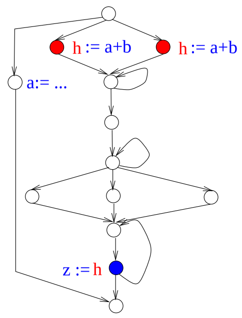
  ⇝ ...known as Busy Code Motion (PLDI'92, Knoop et al.)

...already known to Morel and Renvoise (though no theorem or proof).

Earliestness Principle

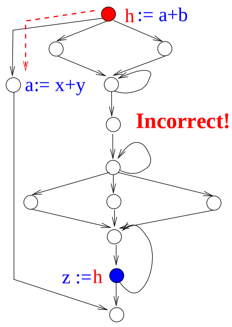
Placing computations as early as possible...

...yields computationally optimal programs.



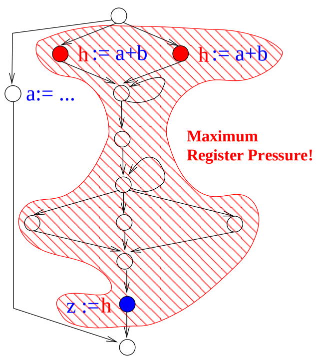
Note: Earliestness means in fact...

...as early as possible, but not earlier!



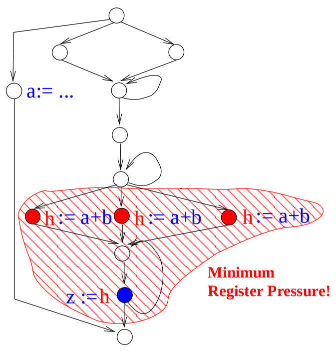
Earliestness Principle: Important Drawback

...computationally optimal, but maximum register pressure



For comparison, the previous program

...computationally optimal, too, but mininum register pressure!



Dual Extreme Strategy – Latestness Principle

Placing computations as late as possible...

- Theorem [Optimality]

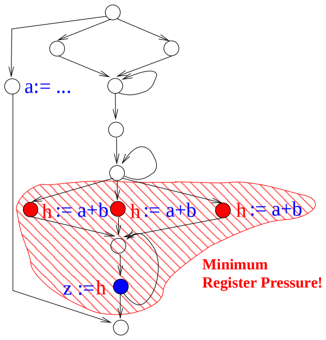
  ...hoisting expressions to their latest safe computation points yields computationally optimal programs with minimum register pressure

  ⇝ ...known as Lazy Code Motion (PLDI'92, Knoop et al.)

33

That's what we have seen...

LCM: ...computationally optimal with mininum register pressure!



34

These days...

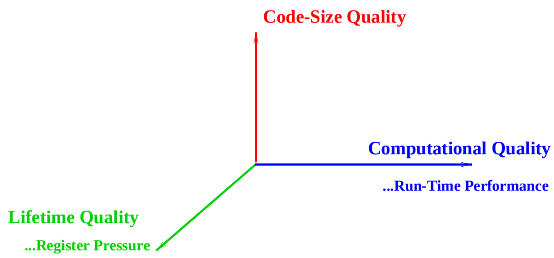
Lazy Code Motion is...

- ...the de-facto standard algorithm for PRE used in contemporary state-of-the-art compilers
  - Gnu compiler family
  - Sun Sparc compiler family
  - ...

35

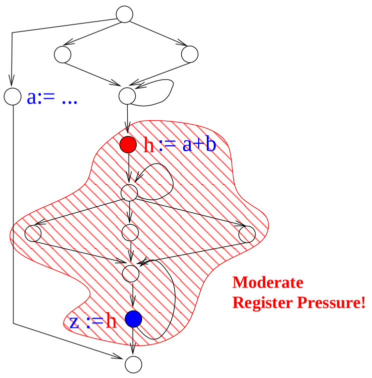
This lecture...

...enhancing LCM to take a user's priorities into account!



36

...rendering possible this transformation, too:



37

Towards Code-Size Sensitive PRE...

- **Background: Classical PRE**

  ⇝ Busy CM (BCM) / Lazy CM (LCM) (Knoop et al., PLDI'92)

  - Received the *ACM SIGPLAN Most Influential PLDI Paper Award 2002 (for 1992)*
  - Selected for *"20 Years of the ACM SIGPLAN PLDI: A Selection"* (60 papers out of ca. 600 papers)

- **Code-Size Sensitive PRE** (Knoop et al., POPL'00)

  ⇝ ...modular extension of BCM/LCM

  ∗ Modelling and Solving the Problem

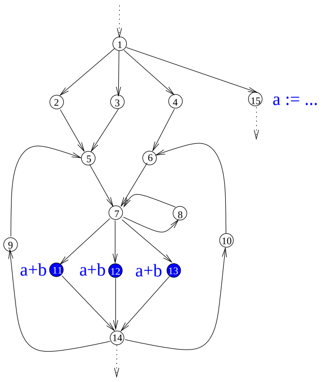
  ...based on graph-theoretical means

  ∗ Main Results

  ...**correctness, optimality**

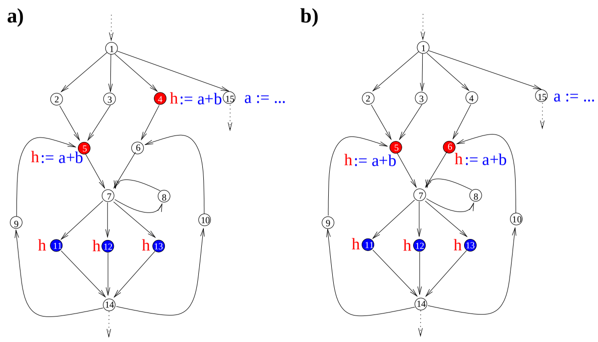
38

The Running Example



39

The Running Example (Cont'd)

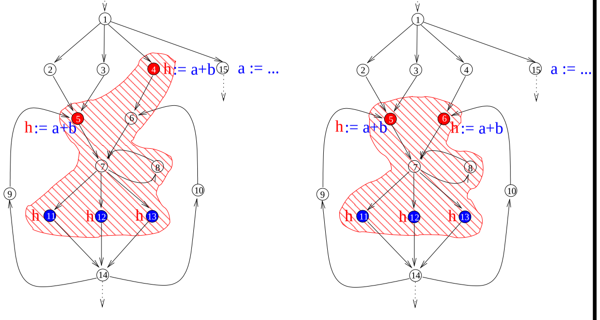


Two Code–size Optimal Programs

40

The Running Example (Cont'd)

a) b)

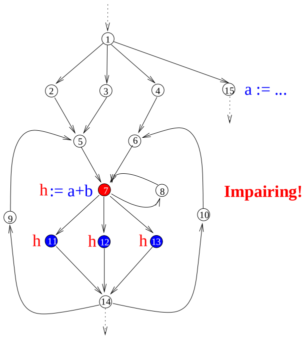


SQ > CQ > LQ

SQ > LQ > CQ

The Running Example (Cont'd)

Note, we do not want the following transformation: It's no option!



Code-Size Sensitive PRE

⇝ **The Problem**

...how to get a code-size minimal placement of computations, i.e., a placement which is

- admissible (semantics & performance preserving)
- code-size minimal

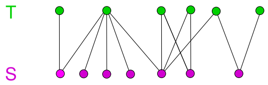
⇝ **Solution: A Fresh Look at PRE**

...considering PRE a trade-off problem: trading the original computations against newly inserted ones!

⇝ **The Clou: Use Graph Theory!**

...reducing the trade-off problem to the computation of tight sets in bipartite graphs based on maximum matchings!

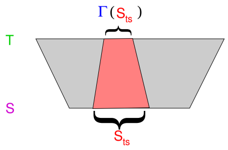
Bipartite Graph



Tight Set

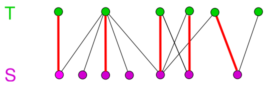
... of a bipartite graph $(S \cup T, E)$ is a subset $S_{ts} \subseteq S$ such that

$$\forall\, S' \subseteq S.\ |S_{ts}| - |\Gamma(S_{ts})| \;\geq\; |S'| - |\Gamma(S')|$$



Two Variants: (1) **Largest** Tight Sets (2) **Smallest** Tight Sets

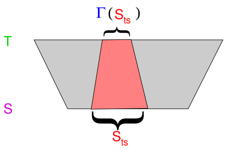
Bipartite Graph



Tight Set

... of a bipartite graph $(S \cup T, E)$ is a subset $S_{ts} \subseteq S$ such that

$$\forall\, S' \subseteq S.\ |S_{ts}| - |\Gamma(S_{ts})| \;\geq\; |S'| - |\Gamma(S')|$$



Two Variants: (1) **Largest** Tight Sets (2) **Smallest** Tight Sets

Apparently

Off-the-shelf algorithms of graph theory can be used to compute...

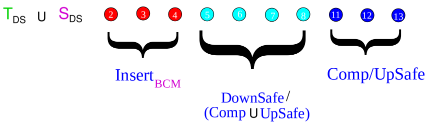
- **Maximum matchings** and
- **Tight sets**

Hence, our PRE problem boils down to...

...constructing the bipartite graph modelling the problem!

Modelling the Trade-Off Problem

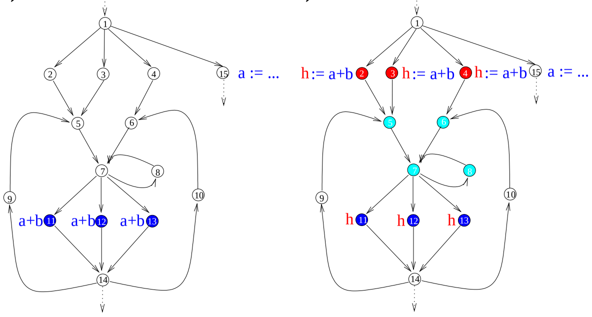
The Set of Nodes



The Set of Edges...

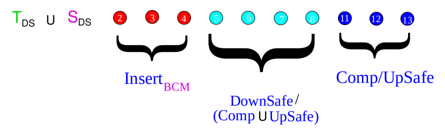
The Set of Nodes

a) b)

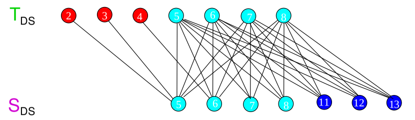


Modelling the Trade-Off Problem

The Set of Nodes



The Bipartite Graph



The Set of Edges ... $\forall n \in S_{DS} \; \forall m \in T_{DS}$.

$$\{n, m\} \in E_{DS} \iff_{df} m \in \textbf{Closure}(pred(n))$$

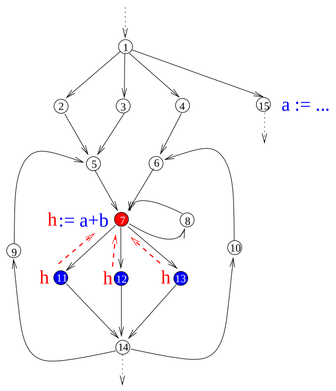
DownSafety Closures

DownSafety Closure

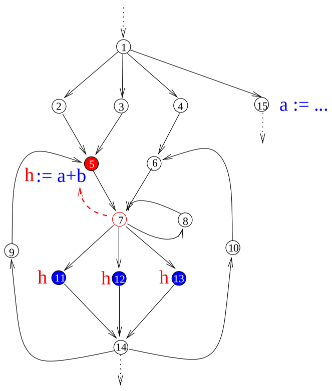
For $n \in$ *DownSafe/Upsafe* the DownSafety Closure $Closure(n)$ is the smallest set of nodes satisfying

1. $n \in Closure(n)$
2. $\forall\, m \in Closure(n) \setminus$ *Comp.* $succ(m) \subseteq Closure(n)$
3. $\forall\, m \in Closure(n).\ pred(m) \cap Closure(n) \neq \emptyset \Rightarrow pred(m) \setminus$ *UpSafe* $\subseteq Closure(n)$

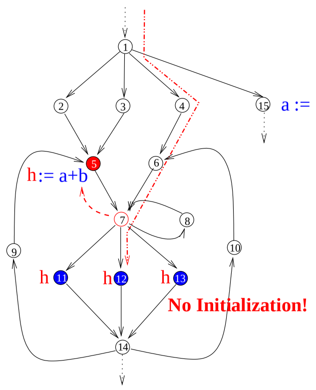
DownSafety Closures – The Very Idea 1(4)



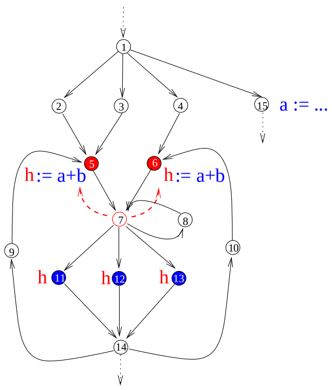
DownSafety Closures – The Very Idea 2(4)



DownSafety Closures – The Very Idea 3(4)



DownSafety Closures – The Very Idea 4(4)



DownSafety Closures

DownSafety Closure

For $n \in$ *DownSafe/Upsafe* the DownSafety Closure $Closure(n)$ is the smallest set of nodes satisfying

1. $n \in Closure(n)$
2. $\forall\, m \in Closure(n) \setminus$ *Comp.* $succ(m) \subseteq Closure(n)$
3. $\forall\, m \in Closure(n).\ pred(m) \cap Closure(n) \neq \emptyset \Rightarrow pred(m) \setminus$ *UpSafe* $\subseteq Closure(n)$

DownSafety Regions

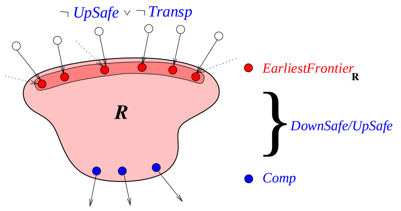
Some subsets of nodes are distinguished. We call each of these sets a DownSafety Region...

- A set $\mathcal{R} \subseteq N$ of nodes is a DownSafety Region if and only if
  1. *Comp*$\setminus$*UpSafe* $\subseteq \mathcal{R} \subseteq$ *DownSafe*$\setminus$*UpSafe*
  2. $Closure(\mathcal{R}) = \mathcal{R}$

Fundamental...

Insertion Theorem

Insertions of admissible PRE-Transformations are always at **"earliest-frontiers"** of **DownSafety regions**.



...characterizes for the first time all semantics preserving PRE-transformations.

The Key Questions

...concerning correctness and optimality:

1. Where to insert computations, why is it correct?
2. What is the impact on the code size?
3. Why is it optimal, i.e., code-size minimal?

...three theorems answering one of these questions each.

Main Results / First Question

1. Where to insert computations, why is it correct?

Intuitively, at the earliestness frontier of the DS-region induced by the tight set...

Theorem 1 [Tight Sets: Insertion Points]

Let $TS \subseteq S_{DS}$ be a tight set.

Then $\mathcal{R}_{TS} =_{df} \Gamma(TS) \cup (\mathit{Comp} \setminus \mathit{UpSafe})$

is a DownSafety Region with $\mathit{Body}_{\mathcal{R}_{TS}} = TS$

Correctness

...immediate corollary of Theorem 1 and Insertion Theorem

Main Results / Second Question

2. What is the impact on the code size?

Intuitively, the difference between computations inserted and replaced...

Theorem 2 [DownSafety Regions: Space Gain]

Let $\mathcal{R}$ be a DownSafety Region

with $\mathit{Body}_{\mathcal{R}} =_{df} \mathcal{R} \setminus \mathit{EarliestFrontier}_{\mathcal{R}}$

Then

- **Space Gain of Inserting at EarliestFrontier$_{\mathcal{R}}$:**

  $|\mathit{Comp} \setminus \mathit{UpSafe}| - |\mathit{EarliestFrontier}_{\mathcal{R}}| =$

  $|\mathit{Body}_{\mathcal{R}}| - |\Gamma(\mathit{Body}_{\mathcal{R}})| \;\; {}_{df}= \mathit{defic}(\mathit{Body}_{\mathcal{R}})$

Main Results / Third Question

3. Why is it optimal, i.e., code-size minimal?

Due to an inherent property of tight sets (non-negative deficiency!)...

Optimality Theorem [The Transformation]

Let $TS \subseteq S_{DS}$ be a tight set.

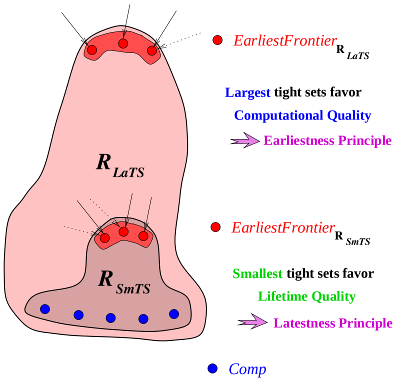
- **Insertion Points:**

  $\mathit{Insert}_{SpCM} =_{df} \mathit{EarliestFrontier}_{\mathcal{R}_{TS}} = \mathcal{R}_{TS} \setminus TS$

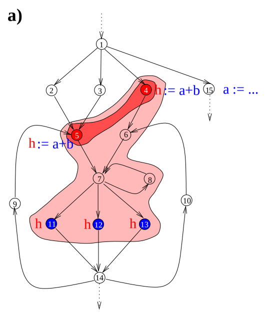
- **Space Gain:**

  $\mathit{defic}(TS) =_{df} |TS| - |\Gamma(TS)| \geq 0$ max.

Largest vs. Smallest Tight Sets: The Impact



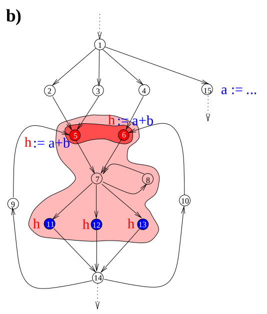
Recall the Running Example



Largest Tight Set

(SQ > CQ)

Earliestness Principle

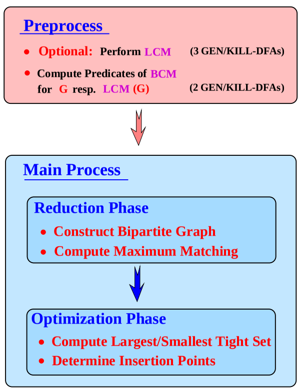


Smallest Tight Set

(SQ > LQ)

Latestness Principle

Code-Size Sensitive PRE at a Glance



Choice of Priority	Apply	To	Using	Yields	Auxiliary Information Required
$\mathcal{LQ}$	Not meaningful: The identity, i.e., G itself is optimal !				
$\mathcal{SQ}$	Subsumed by $\mathcal{SQ} > \mathcal{CQ}$ and $\mathcal{SQ} > \mathcal{LQ}$!				
$\mathcal{CQ}$	BCM	G			UpSafe(G), DownSafe(G)
$\mathcal{CQ} > \mathcal{LQ}$	LCM	G		LCM(G)	UpSafe(G), DownSafe(G), Delay(G)
$\mathcal{SQ} > \mathcal{CQ}$	SpCM	G	Largest tight set	SpCM_{LTS}(G)	UpSafe(G), DownSafe(G)
$\mathcal{SQ} > \mathcal{LQ}$	SpCM	G	Smallest tight set		UpSafe(G), DownSafe(G)
$\mathcal{CQ} > \mathcal{SQ}$	SpCM	LCM(G)	Largest tight set		UpSafe(G), DownSafe(G), Delay(G) UpSafe(LCM(G)), DownSafe(LCM(G))
$\mathcal{CQ} > \mathcal{SQ} > \mathcal{LQ}$	SpCM	LCM(G)	Smallest tight set		UpSafe(G), DownSafe(G), Delay(G) UpSafe(LCM(G)), DownSafe(LCM(G))
$\mathcal{SQ} > \mathcal{CQ} > \mathcal{LQ}$	SpCM	DL(SpCM_{LTS}(G))	Smallest tight set		UpSafe(G), DownSafe(G), Delay(SpCM_{LTS}(G)), UpSafe(DL(SpCM_{LTS}(G))), DownSafe(DL(SpCM_{LTS}(G)))

Conclusions, Perspectives

A brief survey of PRE...

- 1958: *...first glimpse of PRE*
  - ⤳ Ershov's work on *On Programming of Arithmetic Operations.*
- **1979:** *...origin of contemporary PRE*
  - ⤳ Morel/Renvoise's seminal work on PRE
- **1992:** *...LCM* [Knoop et al., PLDI'92]
  - ⤳ ...first to achieve comp. optimality with minimum register pressure
  - ⤳ ...first to rigorously be proven correct and optimal
- **2000:** *...origin of code-size sensitive PRE* [Knoop et al., POPL 2000]
  - ⤳ ...first to allow prioritization of goals
  - ⤳ ...rigorously be proven correct and optimal
  - ⤳ ...first to bridge the gap between traditional compilation and compilation for embedded systems

Conclusions, Perspectives (Cont'd)

- ca. since 1997: *...a new strand of research on PRE*
  - ⤳ Speculative PRE: Gupta, Horspool, Soffa, Xue, Scholz, Knoop,...
- **2005**: *...another fresh look at PRE* (*as maximum flow problem*)
  - ⤳ Unifying PRE and Speculative PRE [Jingling Xue and J. Knoop]

Another Look at the History of PRE

- **< 1979** ... Special Techniques
  - ⤳ Total Redundancy Elimination, Loop Invariant Code Motion
- **1979** ... Partial Redundancy Elimination
  - ⤳ **Pioneering** ... Morel/Renvoise's bidirectional algorithm [1979]
  - ⤳ **Heuristic improvements** ... Dhamdhere [1988, 1991], Drechsler/Stadel [1988], Sorkin [1989], Dhamdhere/Rosen/Zadeck [1992], ...
- **1992** ... BCM & LCM [Knoop et al., PLDI'92]
  - ⤳ BCM ... first to achieve Computational Optimality: Earliestness Principle
  - ⤳ LCM ... first to achieve Comp. & Lifetime Optimality: Latestness Principle

  ... first to be purely unidirectional, however, not yet code-size sensitive.
- **2000/2004**: Code-Size Sensitive PRE [Knoop et al., POPL 2000, LCTES 2004]
- **2005**: Unifying PRE and Speculative PRE [Jingling Xue and Knoop]