CerCo Work Package 5

Indexed Labels for Dependent Costs

Paolo Tranquilli

Unibo

March 16, 2012

Achievements

- Theoretical development of indexed labeling approach
- Application to the untrusted prototype:
  - Implementation of indexed labels
  - Optimisations (peeling, unrolling, constant/copy propagation, lazy code motion, simple peephole opt.)
  - Immediate arguments along compilation chain
  - Branch of Frama-C cost plugin

SEVENTH FRAMEWORK PROGRAMME

Quick recap on the labeling approach

- Inject cost labels at key points in the code
- Propagate them during compilation
- Assign costs to labels measuring the compiled code, lift them to source
- Paramount conditions for the labeling approach:
  in the compiled code labels occur
  - in each loop (for correctness)
  - at every branching (for preciseness)

SEVENTH FRAMEWORK PROGRAMME

Running example – Labeling

$$
\begin{array}{l}
p \leftarrow 1 \\
i \leftarrow 2 \\
\texttt{while } i < 7 \texttt{ do} \\
\quad p \leftarrow p * i \\
\quad i \leftarrow i + 1
\end{array}
\quad \mapsto \quad
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
\texttt{while } i < 5 \texttt{ do} \\
\quad \beta : p \leftarrow p * i \\
\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
$$

trace:	$\alpha \cdots\cdots \beta \cdots\cdots \beta \cdots\cdots \beta \cdots\cdots \gamma \cdots\cdots$				
costs:	122	484	484	484	41

Statically computed costs: $\kappa(\alpha) = 122,\ \kappa(\beta) = 484,\ \kappa(\gamma) = 41$

SEVENTH FRAMEWORK PROGRAMME

Limits of the previous approach

- Paramount conditions for the labeling approach:
  in the compiled code labels occur
  - in each loop (for correctness)
  - at every branching (for preciseness)
- If they are ensured in source code, the above can still fail if
  1. a high level instruction is mapped to a non-sequential block
  2. transformations rearrange the code (e.g. loop optimisations)
  3. the execution cost is context-dependent (e.g. cache)
- Common problem: cost labels occurring with different costs
- **Our solution:** dependent cost labels!

What loop optimisations?

Loop peeling: `while` b `do` S $\mapsto$ `if` b `then` S `while` b `do` S

Loop unrolling: `while` b `do` S $\mapsto$ `while` b `do` S $(\texttt{if}\ b\ \texttt{then}\ S)^*$

(more specialized form of loop unrolling are the norm...)

Running example – peeling

$$
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
\texttt{while } i < 5 \texttt{ do} \\
\quad \beta : p \leftarrow p * i \\
\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
\quad \mapsto \quad
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
\texttt{if } i < 5 \texttt{ do} \\
\quad \beta : p \leftarrow p * i \\
\quad \texttt{while } i < 5 \texttt{ do} \\
\quad\quad \beta : p \leftarrow p * i \\
\quad\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
$$

trace:	$\alpha \cdots\cdots \beta \cdots\cdots \beta \cdots\cdots \beta \cdots\cdots \gamma \cdots\cdots$				
costs:	42	41	246	246	31

Statically computed costs: $\kappa(\alpha) = 42$, $\kappa(\beta) = ???$, $\kappa(\gamma) = 31$

Variable costs occur also with caching

SEVENTH FRAMEWORK PROGRAMME

Dependent labels: peeling

$$
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
i_0 : \texttt{while } i < 5 \texttt{ do} \\
\quad \beta\langle i_0 \rangle : p \leftarrow p * i \\
\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
\quad \mapsto \quad
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
\texttt{if } i < 5 \texttt{ do} \\
\quad \beta\langle 0 \rangle : p \leftarrow p * i \\
\quad i_0 : \texttt{while } i < 5 \texttt{ do} \\
\quad\quad \beta\langle i_0 + 1 \rangle : p \leftarrow p * i \\
\quad\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
$$

trace:	$\alpha \cdots\cdots \beta\langle 0 \rangle \cdots\cdots \beta\langle 1 \rangle \cdots\cdots \beta\langle 2 \rangle \cdots\cdots \gamma \cdots\cdots$				
costs:	42	41	246	246	31

$\kappa(\alpha) = 42,\ \kappa(\beta) = ???$, $\kappa(\gamma) = 31$

SEVENTH FRAMEWORK PROGRAMME

Dependent labels: peeling

$$
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
i_0 : \texttt{while } i < 5 \texttt{ do} \\
\quad \beta\langle i_0 \rangle : p \leftarrow p * i \\
\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
\quad \mapsto \quad
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
\texttt{if } i < 5 \texttt{ do} \\
\quad \beta\langle 0 \rangle : p \leftarrow p * i \\
\quad i_0 : \texttt{while } i < 5 \texttt{ do} \\
\quad\quad \beta\langle i_0 + 1 \rangle : p \leftarrow p * i \\
\quad\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
$$

trace:	$\alpha \cdots\cdots \beta\langle 0 \rangle \cdots\cdots \beta\langle 1 \rangle \cdots\cdots \beta\langle 2 \rangle \cdots\cdots \gamma \cdots\cdots$				
costs:	42	41	246	246	31

$$\kappa(\alpha) = 42,\ \kappa(\beta) = (i_0 == 0)?41 : 246,\ \kappa(\gamma) = 31$$

SEVENTH FRAMEWORK PROGRAMME

Dependent labels: unrolling

$$
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
i_0 : \texttt{while } i < 5 \texttt{ do} \\
\quad \beta\langle i_0 \rangle : p \leftarrow p * i \\
\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
\quad \mapsto \quad
\begin{array}{l}
\alpha : p \leftarrow 1 \\
i \leftarrow 2 \\
i_0 : \texttt{while } i < 5 \texttt{ do} \\
\quad \beta\langle 2 * i_0 \rangle : p \leftarrow p * i \\
\quad i \leftarrow i + 1 \\
\quad \texttt{if } b \texttt{ then} \\
\quad\quad \beta\langle 2 * i_0 + 1 \rangle : p \leftarrow p * i \\
\quad\quad i \leftarrow i + 1 \\
\gamma : \texttt{skip}
\end{array}
$$

trace:	$\alpha \cdots\cdots \beta\langle 0 \rangle \cdots\cdots \beta\langle 1 \rangle \cdots\cdots \beta\langle 2 \rangle \cdots\cdots \gamma \cdots\cdots$				
costs:	42	246	230	246	31

$$\kappa(\alpha) = 42,\ \kappa(\beta) = (i_0 \% 2 == 0)?246 : 230,\ \kappa(\gamma) = 31$$

SEVENTH FRAMEWORK PROGRAMME

The loop indexed labels approach in brief

- Annotate loops with indexes, which parametrize labels
- Loop optimisations transform these parameters
- Semantics keeps track of indexes, and compilation propagates them
  (no added difficulty to proofs of compilation passes)
- Dependent costs for labels are given with conditional expressions

SEVENTH FRAMEWORK PROGRAMME

The loop indexed labels approach in brief

- Annotate loops with indexes, which parametrize labels
- Loop optimisations transform these parameters
- Semantics keeps track of indexes, and compilation propagates them
  (no added difficulty to proofs of compilation passes)
- Dependent costs for labels are given with conditional expressions

SEVENTH FRAMEWORK PROGRAMME

Indexed labeling

- Labeling function $\mathcal{L}$ maps to labeled code
- It is parametrized with fresh indexes, initially unmodified:

$$\mathcal{L}\langle l\rangle(\texttt{while}\ b\ \texttt{do}\ S) := \begin{array}{l} i_k\texttt{: while}\ b\ \texttt{do} \\ \quad \alpha\langle l, i_k\rangle\texttt{:}\ \mathcal{L}\langle l, i_k\rangle(S) \\ \beta\langle l\rangle\texttt{: skip} \end{array}$$

where

- the loop is single-entry
  (important with presence of `goto`s)
- i_k is different from indexes of containing loops
  (in fact, i_k sequence of fresh identifiers, k loop nesting)

The loop indexed labels approach in brief

- Annotate loops with indexes, which parametrize labels
- Loop optimisations transform these parameters
- Semantics keeps track of indexes, and compilation propagates them
  (no added difficulty to proofs of compilation passes)
- Dependent costs for labels are given with conditional expressions

Loop transformations

Loop peeling

$$i_k : \texttt{while}\ b\ \texttt{do}\ S \mapsto$$

```
if b then
  S[i_k ↦ 0]
  i_k : while b do
    S[i_k ↦ i_k + 1]
```

Loop unrolling

$$i_k : \texttt{while}\ b\ \texttt{do}\ S \mapsto$$

```
i_k : while b do
  S[i_k ↦ 2 * i_k]
  if b then
    S[i_k ↦ 2 * i_k + 1]
```

Simple expressions generated by these transformations:

$$s ::= a * i_k + b$$

SEVENTH FRAMEWORK PROGRAMME

The loop indexed labels approach in brief

- Annotate loops with indexes, which parametrize labels
- Loop optimisations transform these parameters
- Semantics keeps track of indexes, and compilation propagates them
  (no added difficulty to proofs of compilation passes)
- Dependent costs for labels are given with conditional expressions

SEVENTH FRAMEWORK PROGRAMME

Indexes in source semantics

- Separate store for indexes: constant indexings C
- Needed operations:
  - $L \circ C$ evaluates a label (e.g. $\alpha\langle 2 * i_0 + 1\rangle \circ (i_0 \mapsto 2) = \alpha\langle 5\rangle$)
  - $C[i_k \downarrow 0]$ denotes setting i_k as 0 in C
  - $C[i_k \uparrow]$ denotes increment of i_k in C
- Unexciting management of indexes with active loops etc.
- $L : S \overset{L \circ C}{\rightarrow} S$: labels are emitted relative to C

Intermediate and target languages

- As loop structure is lost along compilation, indexes need to be managed elsewhere
- In each language (starting from `Cminor`), add explicit pseudo-instructions:

$$\begin{array}{rrcl} \text{emit cost label:} & \texttt{emit}\ L & \leftrightarrow & L \circ C \\ \text{index reset:} & \texttt{reset}\ i_k & \leftrightarrow & C[i_k \downarrow 0] \\ \text{index increment:} & \texttt{inc}\ i_k & \leftrightarrow & C[i_k \uparrow] \end{array}$$

SEVENTH FRAMEWORK PROGRAMME

Semantics preservation

- For every transformation $\mathcal{T}$ there is an extension of $\mathcal{T}$ to statuses $\mathbb{S}$:

$$\frac{P, \mathbb{S} \xrightarrow{\lambda}{}^{*} P', \mathbb{S}'}{\mathcal{T}(P), \mathcal{T}(\mathbb{S}') \xrightarrow{\lambda}{}^{*} \mathcal{T}(P'), \mathcal{T}(\mathbb{S}')}$$

  Optimisation are particular kinds of transformations
- Apart from optimisations, semantics preservation proofs are parametric in the type of cost labels: no added difficulty

The loop indexed labels approach in brief

- Annotate loops with indexes, which parametrize labels
- Loop optimisations transform these parameters
- Semantics keeps track of indexes, and compilation propagates them
  (no added difficulty to proofs of compilation passes)
- Dependent costs for labels are given with conditional expressions

Loop indexed costs

- All $\alpha\langle l\rangle$ in compiled code get a cost $\tau(\alpha\langle l\rangle) \in \mathbb{N}$
- Costs lifted to α giving expression $\tau(\alpha)$.
  That depends on the set of transformations
- E.g. $\alpha\langle 2 * i_0 + 1\rangle$ contributes when $i_0\%2 == 1$

Simple expressions: $s ::= a * i_k + b$

$$\updownarrow$$

Simple conditions: $\begin{cases} i_k == a \\ i_k >= a \\ i_k\%a == b \;\&\&\; i_k >= b' \end{cases}$

- Proof obligation:
  $\forall \alpha, l$ s.t. $\alpha\langle l\rangle$ occurs.$\forall C.\kappa(\alpha) \circ (l \circ C) = \kappa(\alpha\langle l\rangle)$

SEVENTH FRAMEWORK PROGRAMME

Towards cache analysis

- To exploit cache analysis in loops virtual loop peeling is performed
- Indexed labels allow to handle such virtual loop peeling
- Global abstract interpretation yields a cost per instruction
- Analysis categorizes variables in:
  - Always hit
  - Persistent: every access but the first is a hit
  - Other
- We can implement cache analysis for 8051 extensions, and integrate the results with dependent costs.

SEVENTH FRAMEWORK PROGRAMME

Conclusions

Not showed here: instrumentation, dependent cost simplifications, implementation details

Perspectives:

- Abstract algebra for simple expressions/conditions?
- Loop optimisation is interesting in this framework, as it can be driven by cost annotations
- Dependency could be extended to variables. For example: loop reversing ($i_k \mapsto n - i_k$) or simple instructions compiled with branching code (e.g. shift in 8051)