Loop Quasi-Invariant Peeling
A method to optimize programs

Assya Sellak
Advisor: Dr. Clément Aubert

AUGUSTA UNIVERSITY
Programs

- Programs = set of instructions to perform a task
- Types of instructions:
  - Data modification
  - Control flow
Control-flow Instructions

If...else...

Loops: while, do...while, for
for i in batch {
    oven.preheat()
    bowl.add(dry_ingredients)
    bowl.mix()
    ....
    cookies.bake()
}
Example: Optimize Baking Time

For each batch:
- Preheat oven
- Mix dry stuff ...
Example: Optimize Baking Time

For each batch:
- Preheat oven
- Mix dry stuff ...

peel unnecessary instruction

Preheat oven
For each batch:
- Preheat oven
- Mix dry stuff...
Loop Invariant Code Motion (LICM)

• Peeling: move commands that do not change within the loop to occur before the loop

```plaintext
i = 0;
while (i < n) {
    w = 20;
    x = y + z;
    i++;
}
```
Loop Invariant Code Motion (LICM)

- Peeling: move commands that do not change within the loop to occur before the loop

```
i = 0;
while (i < n) {
  w = 20;
  x = y + z;
  i++;
}
```

```
i = 0;
w = 20;
x = y + z;
while (i < n) {
  w = 20;
  x = y + z;
  i++;
}
```
Proof of Equivalence

- Guarantee the optimized program performs the same tasks as the original

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performs specified task(s)</td>
<td>Performs specified task(s)</td>
</tr>
<tr>
<td>Runs at a slower speed</td>
<td>Runs at a faster speed</td>
</tr>
</tbody>
</table>
**Algorithm Definition**

**Definition 6.** Let $C := \text{while } E \text{ do } [C_1; C_2; \ldots; C_n]$ be a command. We define the directed graph $\text{Dep}(C)$ as follows:

- the set of vertices $V^{\text{Dep}(C)}$ is equal to $\{C_1, \ldots, C_n\}$ (the set of commands in the loop);
- the set of edges $E^{\text{Dep}(C)}$ is equal to $\forall m=1, \forall i \in \text{In}(C_m) \text{ PrD}_i(C_m)$ (the set of all principal dependencies);
- the source $s(i)$ of the edge $C_k \in \text{PrD}_i(C_m)$ is $C_k$;
- the target $t(i)$ of the edge $C_k \in \text{PrD}_i(C_m)$ is $C_m$.

The invariance degree $\deg_C(C_m)$ of a command $C_m$ w.r.t. $C$ is then defined as follows. When clear, we will avoid writing the subscript $C$ to ease notations. If $C_m$ is a source in $\text{Dep}(C)$, then $\deg(C_m) = 1$. If $C_m$ has a reflective edge in $\text{Dep}(C)$, then $\deg(C_m) = \infty$. Otherwise, we write $\text{Fib}(C_m)$ — the fiber over $C_m$ — the set of vertices in $\text{Dep}(C)$ defined as $\{C_k \mid \exists e \in E^{\text{Dep}(C)}, s(e) = C_k, t(e) = C_m\}$, and define $\deg(C_m)$ by the following equation, where $\chi_{>m}(i) = 1$ if $i > m$ and $\chi_{>m}(i) = 0$ otherwise:

$$\deg(C_m) = \max \{\deg(C_i) + \chi_{>m}(i) \mid C_i \in \text{Fib}(C_m)\}$$

In particular, if $C_m$ is part of a cycle in $\text{Dep}(C)$, its degree is equal to $\infty$.

For all $i \in \mathbb{N} \cup \{\infty\}$, we define the inverse image $\deg^{-1}(i)$, i.e. $\deg^{-1}(i) = \{C_k \mid \deg(C_k) = i\}$, and we note $\maxdeg(C)$ the largest integer (i.e. not equal to $\infty$) such that $\deg^{-1}(\maxdeg(C)) \neq \emptyset$. The following lemma will be used in the proof of the main theorem.
Algorithm Snippet

def comput_deg(tabDeg, i, lldep):
    if tabDeg[i] == 0:  // if deg(Ci) has not been computed
        if (len(lldep[i]) == 0):  // if Ci has no dependencies, set deg(Ci) in tabDeg to 1
            tabDeg[i] = 1
        else:  // else compute max degree of Ci's dependencies(Cl)
            tabDeg[i] = -1
            deg = -1
            for l in lldep[i]:
                // compute degree for each dependency and update tabDeg
                tabDeg[l] = comput_deg(tabDeg, l, lldep)
                // if Ci is also a dependency of Cl, then there is a loop
                if tabDeg[l] == -1: return -1
                // if deg(Cl) is the max and Cl precedes Ci
                if (tabDeg[l] > deg) and l < i: deg = tabDeg[l]
                // if deg(Cl) is the max if Cl follows Ci
                if (tabDeg[l] >= deg) and l > i: deg = tabDeg[l] + 1
                // do nothing if the current degree is the max
            tabDeg[i] = deg  // set deg(Ci) in tabDeg to max degree

    return tabDeg[i]
Application

- Simplification
- Automatically generated
- No conflict
Results

• Improve original implementation
• Added tests
• Overall goal = transform
Limitations

• Proof of concept:
  • Programming language = C
  • Only certain types of instructions
  • Rest are ignored
Future work

• Add optimization to compiler
• Parallelization: split loops to run simultaneously

```java
i = 0, j = 10, k = 0;
while (i < n) {
    j = j - 1;
    k = k + 1;
    i = i + 1;
}
```

```java
i = 0, j = 10, k = 0;
while (i < n) {
    j = j - 1;
    i = i + 1;
}
```
Conclusion

- Method for removing unnecessary instruction from loops

Thank you!