Loop Quasi-Invariant Peeling

A method to optimize programs

Assya Sellak

Advisor: Dr. Clément Aubert

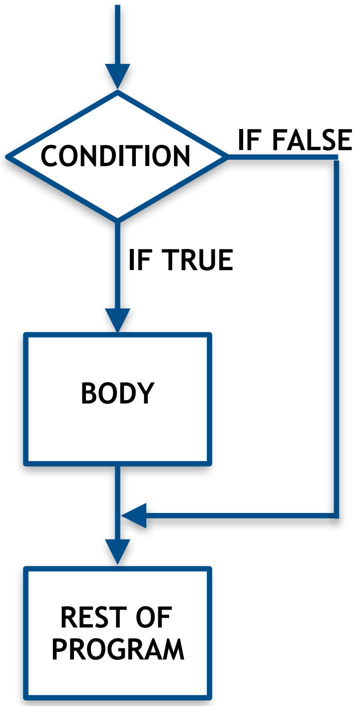
AUGUSTA UNIVERSITY

Programs

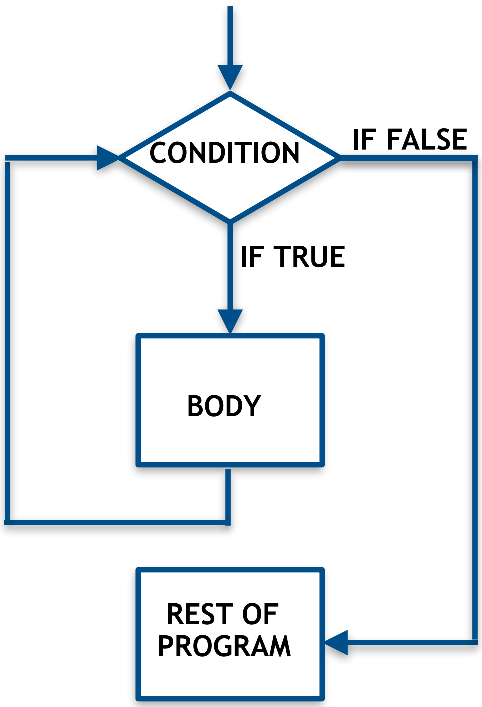
- Programs = set of instructions to perform a task
- Types of instructions:
  - Data modification
  - Control flow

Control-flow Instructions

If…else…

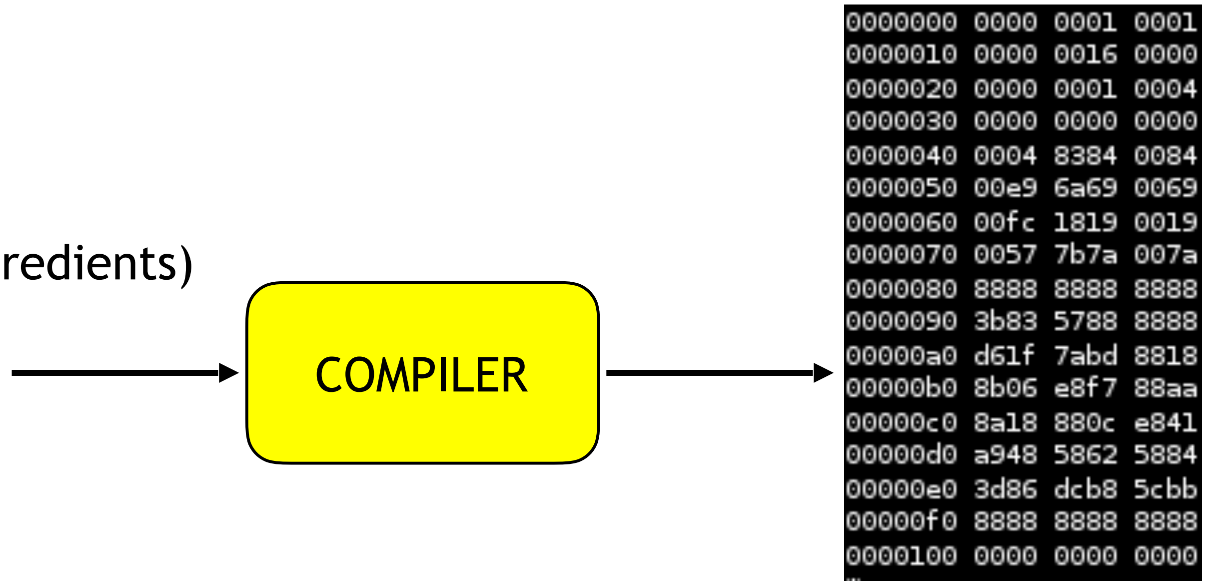


Loops: while, do…while, for



Programming Language **Machine Language**

```
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

```
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

INPUT	OUTPUT
Performs specified task(s)	Performs specified task(s)
Runs at a slower speed	Runs at a faster speed

Algorithm Definition

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

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

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

$$\deg(\mathtt{C_m}) = \max\left(\{\deg(\mathtt{C_i}) + \chi_{>m}(i) \mid \mathtt{C_i} \in \mathrm{Fib}(\mathtt{C_m})\}\right)$$

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

For all $i \in \mathbf{N} \cup \{\infty\}$, we define the inverse image $\deg^{-1}(\mathtt{i})$, i.e. $\deg^{-1}(\mathtt{i}) = \{\mathtt{C_k} \mid \deg(\mathtt{C})_k = i\}$, and we note $\mathrm{maxdeg}(\mathtt{C})$ the largest integer (i.e. not equal to ∞) such that $\deg^{-1}(\mathrm{maxdeg}(\mathtt{C})) \neq \emptyset$. 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 = tranform

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

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

→

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

while (i < n) {
    k=k+1;
    i=i+1;
}
```

Conclusion

- Method for removing unnecessary instruction from loops

Thank you!