Optimizing Compilers
4th Lecture

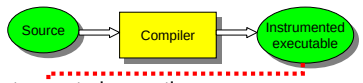
Bernhard Scholz
Institut f. Computersprachen
Argentinierstr. 8
scholz@complang.tuwien.ac.at

Profiling

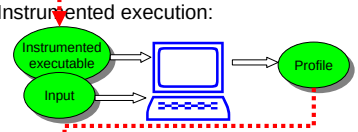
- Why ?
  - to produce high-quality code based on runtime behavior
  - superior to static analysis
- Different types of profiling
  - profiling for control flow graphs
    - basic block, edges, acyclic paths, whole program paths
  - memory accesses (for cache analysis, etc.)
  - domain specific profiles depending on language and application
- In dynamic environments (e.g. JITs) profiling is crucial!

Feedback-Directed Optimization

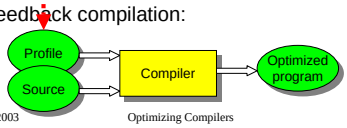
1. Instrumented compilation:



2. Instrumented execution:



3. Feedback compilation:

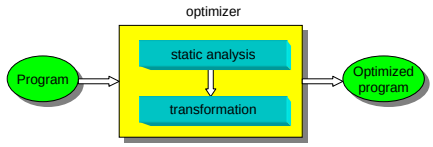


FDO(2)

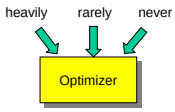
- Feedback-Directed Optimization is expensive to implement.
- Instrumented program can be significantly slower than optimized program.
- Solution: A small training input-set is used for obtaining profile information.
- Problem: Which training set is representative?
  - open research topic
  - statistical techniques are currently employed
  - several executables depending on input?

Classical Approach

- Classical Program Optimization:

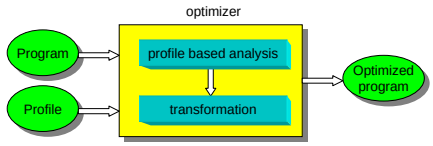


- Drawback:

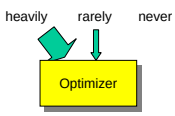


Profiling Approach

- Optimization based on profiling

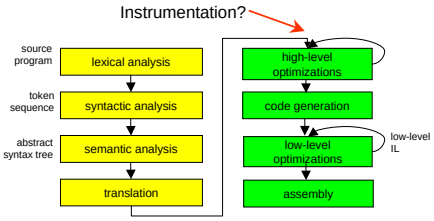


- Advantage:



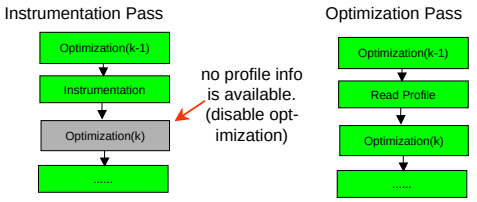
Phase Ordering for Profiling

Problem: Transformations modify input program and invalidate profile information. When and where should be profiled?



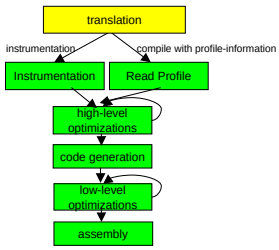
Phase Ordering(2)

- There is only one optimization that uses profile information (e.g. Optimization k) and the optimization is only performed once:



Phase Ordering(3)

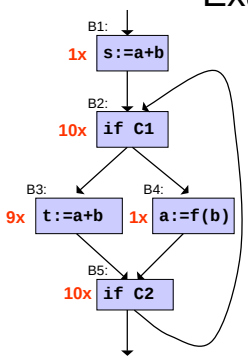
- Alternative: Transformations are *profile-aware*.
- Every transformation updates profile information
  - Is that possible for all transformations?



Basic Block Profiling

- Measure execution frequencies of basic blocks
- Instrumentation Pass
  - introduce for each basic block a basic block counter
  - increment counter when block is executed
  - dump profile information at the end of program execution
- Optimization Pass
  - read profile information
  - annotate basic blocks with frequency information
  - run optimization based on frequency information
- Applications
  - code generation issues, register allocation, etc.

Example



- Control Flow
  - 9 times left branch
  - 1 times right branch
  - terminate loop
- Profiling Result

Node	Freq
B1	1
B2	10
B3	9
B4	1
B5	10

Implementation

- Implementation details:
  - Introduce for each function an array
    `number __freq_<func>[NUM_BB];`
  - insert code for each basic block at the entry
    `__freq_<func>[<block>] ++;`
    where every basic block has a unique index.
  - a finalize handler dumps the basic block information in a file.
- Problem: counter overflows might happen
  - either 64-bit number (overflow is quite unlikely) or
  - an overflow-check: if(f < (unsigned)(-1)) f++;

Edge Profiling

- Measure execution frequencies of edges
- Instrumentation
  - introduce for each jump (or transition between basic blocks) a counter
  - increment counter when edge (jump) is executed
  - dump profile information at the end of program execution
- Edge profiling is more expensive than block profiling -- still moderate costs!
- Edge profiling is superior to basic block profiling
  - basic block counts can be derived
  - more optimization applications, e.g. basic block reordering

Flow Properties

- Frequency of a node / flow conditions

$$f(u) = \sum_{v \in pred(u)} f(v \to u) \quad \text{for all nodes } v \in N \backslash \{\text{start}\}$$

$$f(u) = \sum_{v \in succ(u)} f(u \to v) \quad \text{for all nodes } u \in N \backslash \{\text{end}\}$$

  - the incoming flow amount must be equal to the outgoing one

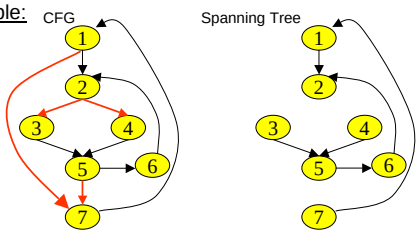
$$\sum_{v \in pred(u)} f(v \to u) = \sum_{v \in succ(u)} f(u \to v) \quad \text{for all nodes } u \in N \backslash \{\text{start,end}\}$$

- Refinement
  - not all edges need be instrumented to determine flow
  - speed up execution time of instrumented program

Refinement

Lemma: Not-measured edge frequencies can be determined if not-measured edges form a spanning tree.

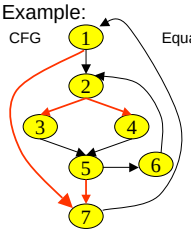
Example:



Refinement(2)

- Flow Properties allow to compute not-measured edges, i.e. linear equation system.
- Solved by substitution
- Example:

CFG



Equations:

$$f(b_3 \to b_5) = f(b_2 \to b_3)$$
$$f(b_4 \to b_5) = f(b_2 \to b_4)$$
$$f(b_5 \to b_6) = f(b_3 \to b_5) + f(b_4 \to b_5)$$
$$f(b_6 \to b_2) = f(b_5 \to b_6)$$
$$f(b_7 \to b_1) = f(b_1 \to b_7) + f(b_5 \to b_7)$$
$$f(b_1 \to b_2) = f(b_7 \to b_1) - f(b_1 \to b_7)$$

Static Profiling

- What is if feed-back loop cannot be set in place?
- Approach based on heuristics to assess frequencies
- *Advantage:*
  - There is no phase-ordering problem, i.e. re-computations of frequencies after every program transformation.
- Two phases:
  1. Compute branch probabilities based on heuristics
  2. Use Markov model to compute frequencies

Markov Chains

- A Markov Model consists of
  - set of states, i.e. S={s_1,s_2,...,s_k}
  - transitions between states with a certain probability $p(s_1 \to s_2)$
- Properties
  - the sum of the probabilities for outgoing transitions must be one:

$$\sum_{v \in succ(u)} p(u \to v) = 1$$

  - the transitions are independent of each other, such that

$$p(s_1 \to s_2 \to s_3 \to \ldots s_k) = p(s_1 \to s_2)\, p(s_2 \to s_3) \ldots p(s_{k-1} \to s_k)$$

Markov Chains(2)

- Expected frequency of going from one state to another one is defined as:

$$\hat{f}(u,v) = \sum_{\pi \in Path(u,v)} p(\pi)$$

where $\pi = s_1 \to s_2 \to s_3 \to \ldots s_k$ and

$$p(s_1 \to s_2 \to s_3 \to \ldots s_k) = p(s_1 \to s_2)\, p(s_2 \to s_3) \ldots p(s_{k-1} \to s_k)$$

- Computed by a linear equation system

Markov Chains & CFG

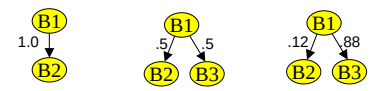
- Control-flow graph as Markov model
  - we need heuristics to estimate branch probabilities
  - imprecise: jumps are not independent of each other (who cares we estimate!)
  - compute expected frequency $\hat{f}(\text{start}, u)$ for all nodes u.
  - linear equation system for expected frequencies

$$\hat{f}(\text{start}) = 1$$

$$\hat{f}(u) = \sum_{v \in pred(u)} p(v \to u)\, \hat{f}(v) \quad \text{for all nodes } u \in N \backslash \{\text{start}\}$$

Branch Probabilities

- Simple loop heuristic
  - back edges are taken with a probability of 0.88
  - if no outgoing edge is a back edge => distribute equally
- Cases for OIL

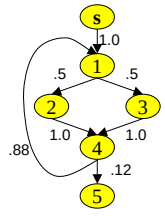


one successor node | two successor nodes w/o back-edge | two successor nodes with back-edge

- Special case:
  - both outgoing edges are back edges => inner loop 88%
  - i.e. B2 dom B3

Example

Given CFG with branch probabilities:



Equations: $\hat{f}(s) = 1$

$\hat{f}(b_1) = 1.0\,\hat{f}(s) + 0.88\,\hat{f}(b_4)$

$\hat{f}(b_2) = 0.5\,\hat{f}(b_1)$

$\hat{f}(b_3) = 0.5\,\hat{f}(b_1)$

$\hat{f}(b_4) = 1.0\,\hat{f}(b_2) + 1.0\,\hat{f}(b_3)$

$\hat{f}(b_5) = 0.12\,\hat{f}(b_4)$

Solution: $\hat{f}(s) = 1.0,\ \hat{f}(b_1) = 8.33,\ \hat{f}(b_2) = 4.167,$

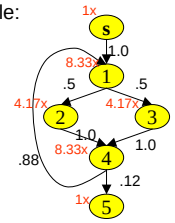
$\hat{f}(b_3) = 4.167,\ \hat{f}(b_4) = 8.33,\ \hat{f}(b_5) = 1.0$

Edge Frequencies

- Based on branch probability and basic block frequency

  $\hat{f}(u \to v) = \hat{f}(u)\,p(u \to v)$

- Example:



Edges	Freq
s-1	1.0
1-2	4.17
1-3	4.17
2-4	4.17
3-4	4.17
4-1	7.33
4-5	1.0

Refinement

- Add additional heuristics
  - predict that a comparison of a pointer against null or of two pointers will fail => 60%
  - Predict that a comparison of an integer for less than (or equal to) zero will fail => 84%
  - Return as a successor will not be taken => 72%
  - etc....
- How can we compose one probability for an edge based on several heuristics?

Refinement(2)

- Theory of Dempster-Shafer [Shafer-76]
- Several heuristics
  - heuristic H
  - two probabilities pH(e1) and pH(e2) for successors
- Algorithm

```
p(b-s1)=0.5;
p(b-s2)=0.5;
for all heuristics H
  d=p(b-s1)*pH(b-s1)+
    p(b-s2)*pH(b-s2);
  p(b-s1)=p(b-s1)*pH(b-s1)/d;
  p(b-s2)=p(b-s2)*pH(b-s2)/d;
```


Stop

- Next lecture: 8.5.2003, 13:45 – 14:45
- 3rd Assignment!