CS510 Software Engineering

Program Profiling

Asst. Prof. Mathias Payer

Department of Computer Science
Purdue University

TA: Scott A. Carr
Slides inspired by Xiangyu Zhang

http://nebelwelt.net/teaching/15-CS510-SE

Spring 2015

Table of Contents

Definition of Profiling

Profiling

Profiling is a lossy technique that aggregates execution information to finite entries.

While tracing is lossless (i.e., records every detail of the execution), profiling is lossy. Think: counting versus sampling.
Examples: control-flow profiling based on instruction/edge/function frequency or value profiling based on value frequency.

Table of Contents

Use-cases for profiling

- Optimization: identify hot program paths;
- Optimization: data compression;
- Optimization: value speculation;
- Optimization: data locality detection (for caching);
- Optimization: performance tuning;
- Testing: code coverage;
- Debugging: check execution frequencies.

Table of Contents

GNU gprof Profiler

- gprof is a profiler for C programs, profiling execution times for functions and procedures with frequency-annotated call edges.
- Compile a program with `-g -pg` to enable profiling and debug information.
- gcc then instruments function entry and exit of each function to record per-function calling frequencies.
- Per-function execution time is sampled (leading to imprecision).
- After execution a gmon.out file is created which can be viewed using `gprof ./exec`.

gprof Example

Switch to demo.

grpof Example 2

Switch to demo 2.

Address Sanitizer: Quick look

- The run-time library replaces the malloc and free functions.
- The memory around malloc-ed regions (red zones) is poisoned.
- The free-ed memory is placed in quarantine and also poisoned.
- Every memory access in the program is transformed by the compiler in the following way:
  `if (isPoisoned(address)) { error(); } stmt;`.

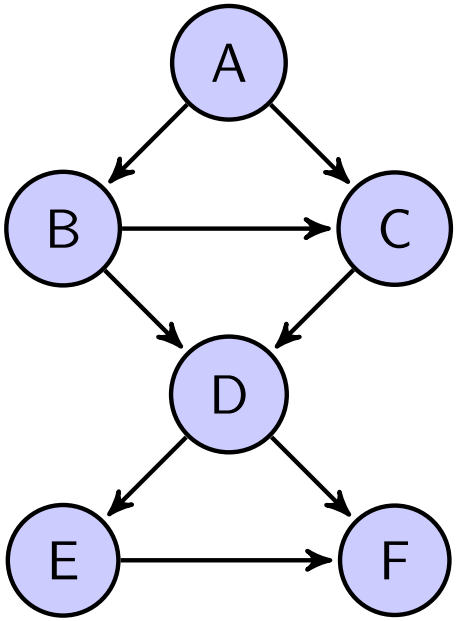
ASan Example

Switch to demo 3.

Table of Contents

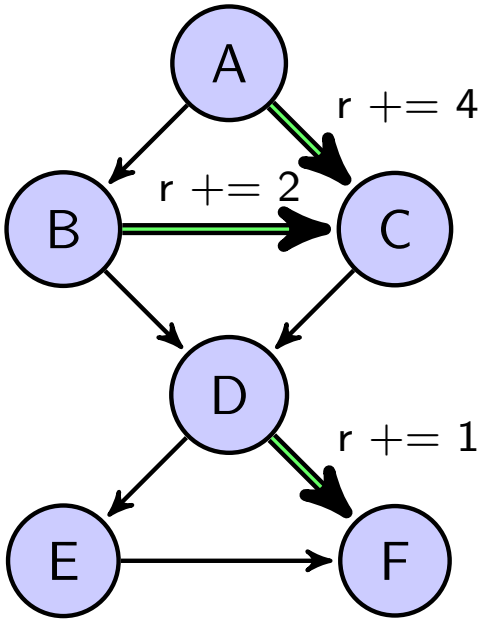
Path Profiling

How often is a certain control-flow path executed?



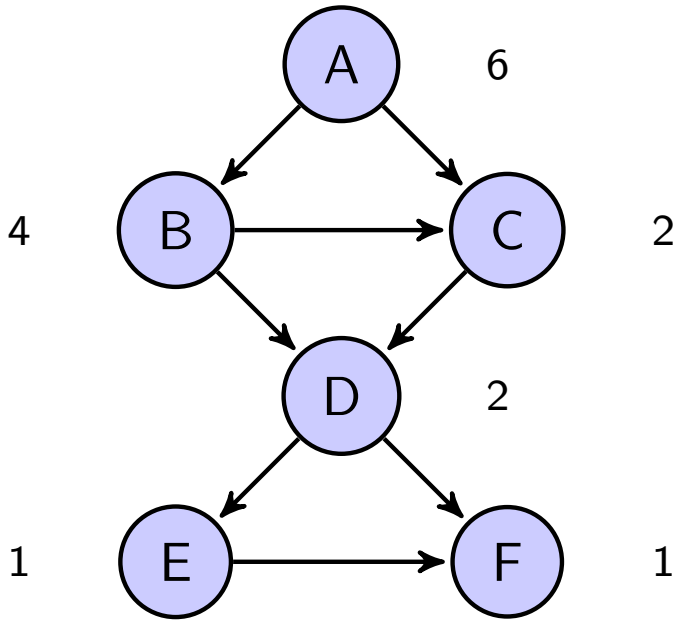
Naive solution: record sequence of executed basic blocks, then increment frequency for given path.

Efficient Path Profiling



Path	Encoding
ABDEF	0
ABDF	1
ABCDEF	2
ABCDF	3
ACDEF	4
ACDF	5

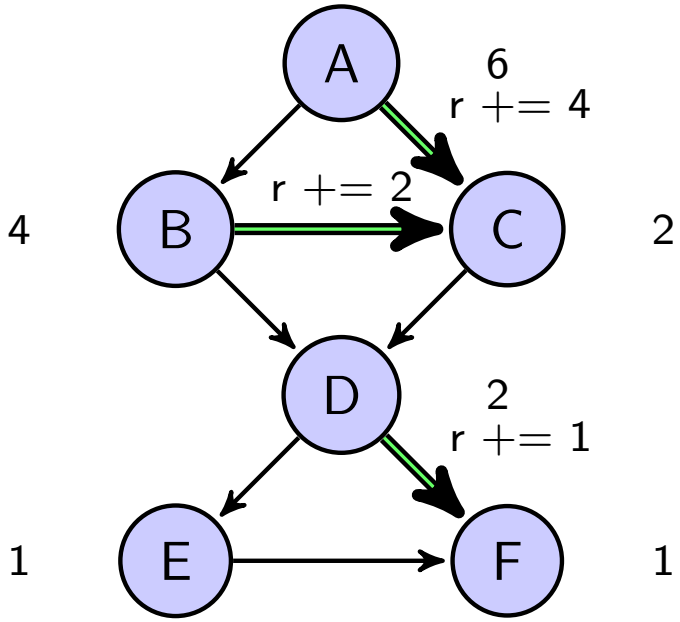
Efficient Path Profiling



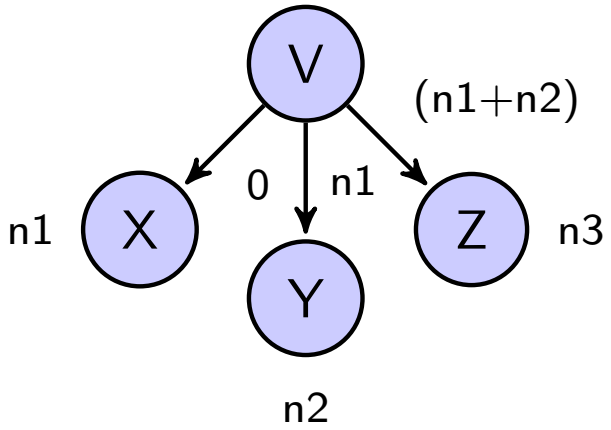
Each node is annotated with the number of paths from that node to the end:

$num(n) = \sum_{child\_i} num(i)$

Efficient Path Profiling (EPP)



Given path sums, encode edge increments as follows:



Path Regeneration

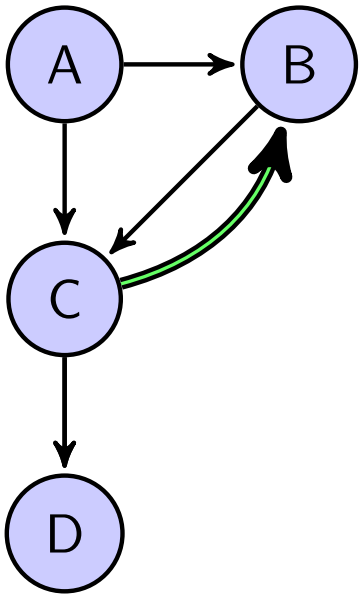
- Start at the top of the graph with count P.
- Follow highest possible path so that P does not become negative.
- Repeat until you reach the bottom with $P = 0$.

Handling Loops

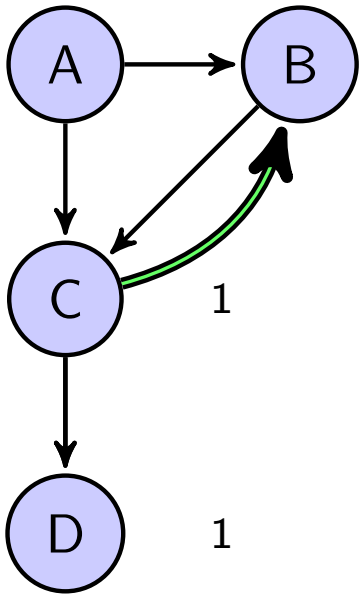
How could loops be handled in this approach?
Some part of the graph is executed multiple times, we need to define a way to capture this behavior.

1. add extra nodes that capture entry and exit of a function;
2. connect entry to loop beginnings;
3. connect loop endings to exit;
4. remove loop edge;
5. store each loop iteration as its own path.

Handling Loops

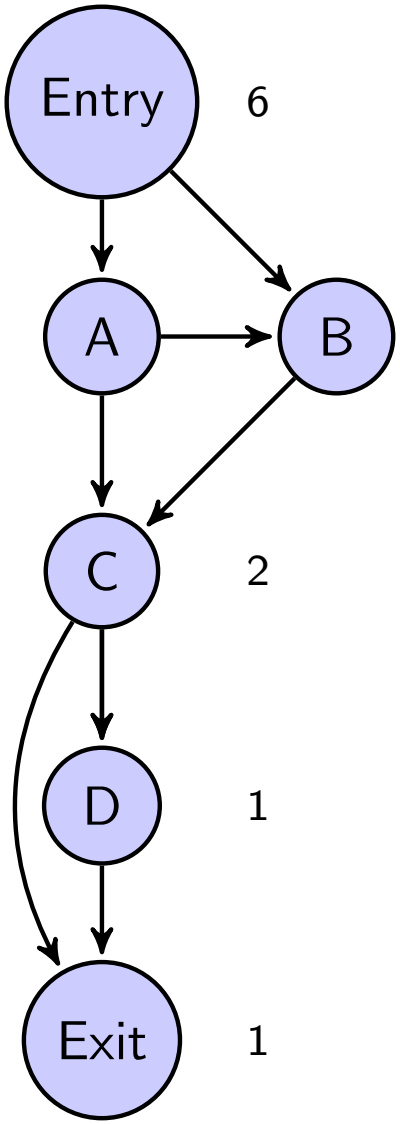


2

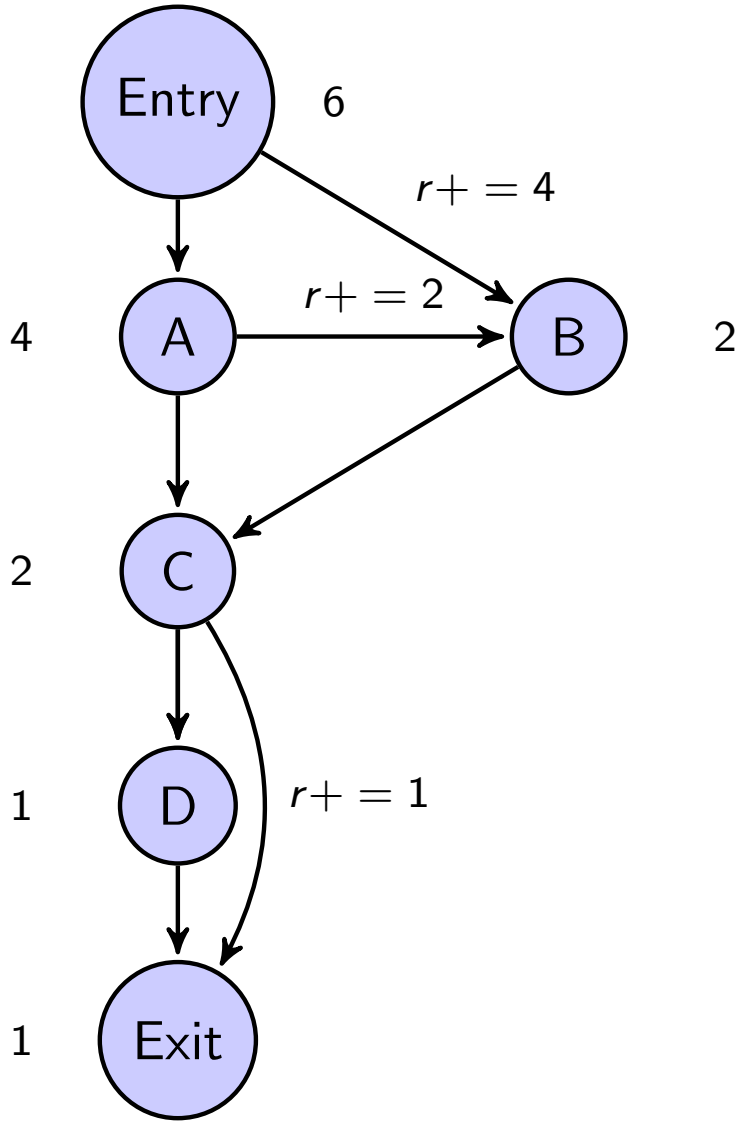


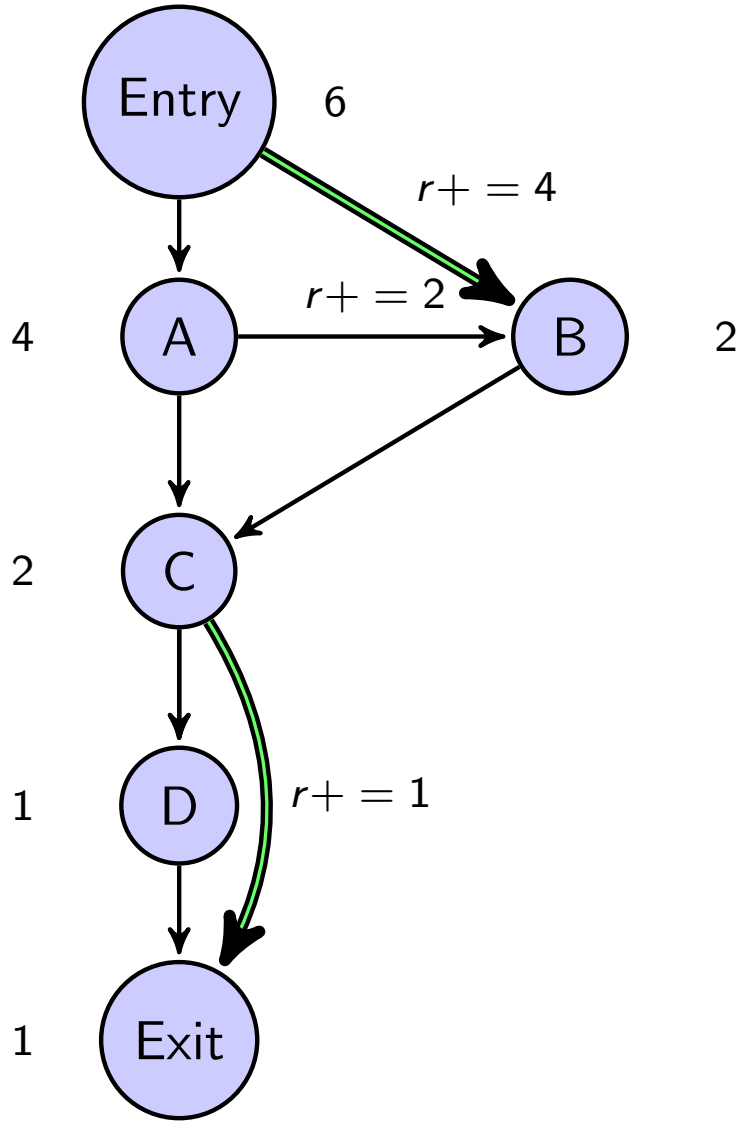
1

4

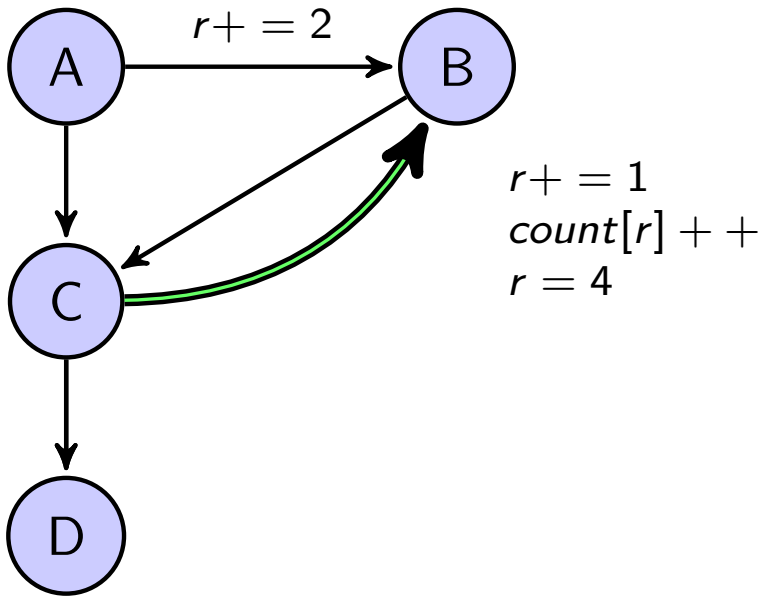


Handling Loops (2)





Handling Loops (3)



EPP Characteristics

- EPP causes 40% overhead on average.
- Problem of path explosion: number of paths becomes too large to enumerate for complex functions, resort to hash maps.
- Allows efficient tracing (compared to alternatives).
- Reading assignment: Efficient Path Profiling, by T. Ball and J. Larus, Micro 1996.

Table of Contents

Optimization: Object Equality Profiling

- Observation: object creation and destruction are expensive.[1]
- Observation: some (many?) objects are equal.
- Idea: merge equivalent objects into single object.
- Requires: all objects have the same field values.
- Requires: all references are redirected to a single object.
- Requires: updates are synchronized/valid (safe: for read-only objects only).
- Merging objects reduces memory usage, improves cache locality, reduces GC overhead, and reduces allocation cost.

[1] Reading assignment: Darko Marinov and Robert O'Callahan, Object Equality Profiling, OOPSLA'03

Mergeability requirements

- Both objects are of the same class.
- Each pair of corresponding field values is either identical or are references to objects that are mergeable themselves (recursive definition).
- Neither object is mutated in the future.
- The objects have (partially) overlapping lifetimes.

Mergeability Profiling

- Profiler produces triplets:
  (*class*, *allocation_site*, *estimated_saving*)
- Information needed: allocation times, last references, field values.
- All memory allocation must be traced.
- Results: memory footprint reduction of 37% and 47% for two SpecJVM programs.
- Only small program changes needed (1 line for DB).

OO Best Practices

The following best practices are loosely based on IBM WebSphere Application Server best practices[2].

- Don't allocate large objects too frequently.
- Reuse $\{datasources, connections, objects\}$
- Release objects when done.
- Avoid string concatenation $(x+ = y)$.
- Minimize cross-thread synchronization.

[2] `https://www-01.ibm.com/software/webservers/appserv/ws_bestpractices.pdf`

Questions?

?