Efficient Path Profiling

Thomas Ball
Bell Laboratories
Lucent Technologies
tball@research.bell-labs.com

James R. Larus*
Dept. of Computer Sciences
University of Wisconsin-Madison
larus@cs.wisc.edu

Appeared in MICRO'96 Conference Proceedings

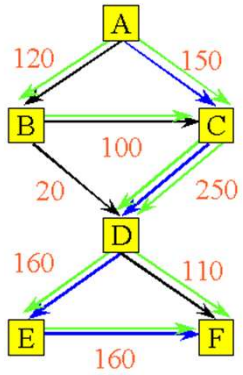
Presented by Long Cheng
10/06/2015

Background

- Profiling: analysis of program behavior based on run-time data
- Profiler: conceptual module whose purpose is to collect or analyze runtime data
- Profile: a set of frequencies associated with run-time events
- Static analysis Vs. Dynamic analysis
  - Programs behaviors are hard to understand statically
  - Dynamic analysis based on runtime data is needed

Path Profiling

- How often does a control flow path execute?
  - Before this, basic block and control flow edge profiling were used. Path profiling was assumed to be much more costly
    - Blocks ➔ statements & lines
    - Edges ➔ branches & blocks
    - Paths ➔ sequence of edges & blocks



Path	Profile A	Profile B
ACDF	90	110
ACDEF	60	40
ABCDF	0	0
ABCDEF	100	100
ABDF	20	0
ABDEF	0	20

- Why path profiling is needed?
  - Edge profiling does not identify the most frequently executed paths (and no cheaper)

Path Profile Usage

- Debugging and bug-isolation
- Feedback based optimization
  - Concentrate/favor frequently executed paths
- Performance tuning
  - Hardware metrics along path
- Software coverage testing
- Characterize program execution, understanding program/architecture interaction

Efficient Path Profiling

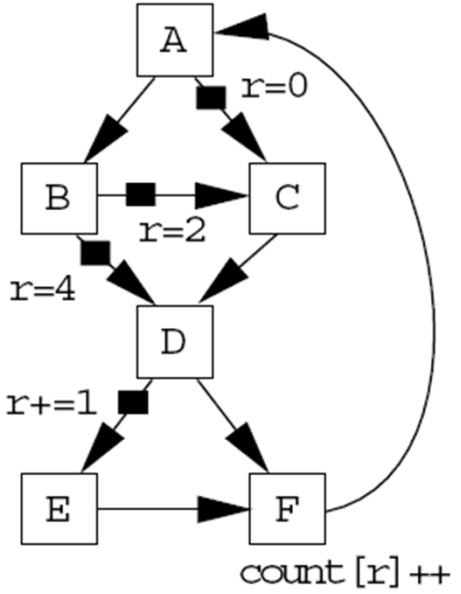
- This paper describes an efficient path profiling algorithm
  - Simple
  - Fast
  - Minimized run-time overhead
    - Efficient edge profiling : average overhead 16%
    - Efficient path profiling : average overhead 31%
    - Accurate path profiling overhead is only twice as compared to efficient edge profiling

Outline

- Path profiling of directed acyclic graphs (DAGs)
- Arbitrary control-flow graphs
- Experimental results

Path Profiling of DAGs

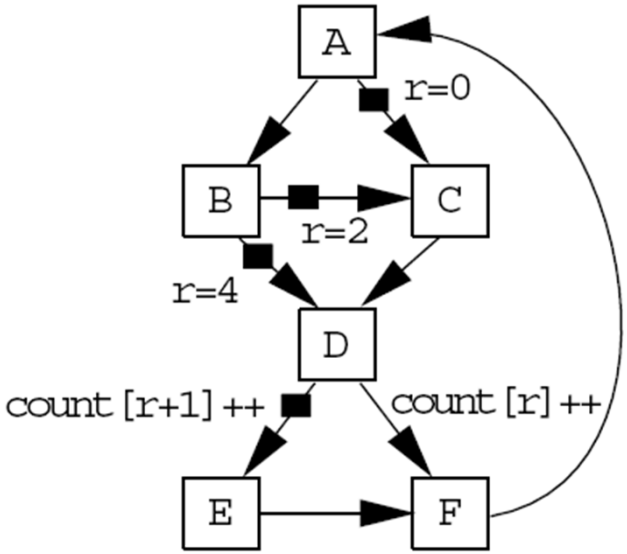
- Basic Idea
  - Paths are identified by unique integer (path identifier)
  - This integer is used to index an array of counter



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

Path Profiling of DAGs

- Pre-execution
  - Assign edge values
  - Minimize edge increments
  - Place instrumentation
- Execution
  - Record path profile
- Post-execution
  - Associate path with number (Path Regenerating)



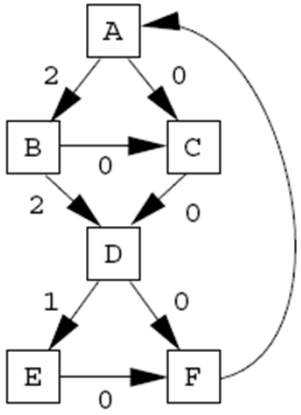
Path Profiling of DAGs

- Terminology
  - Control-flow graphs (CFGs) have been converted into directed acyclic graphs (DAG) with a unique source vertex ENTRY and sink vertex EXIT
    - Basic algorithm assumes that control flow graph is DAG
    - Later show how to transform an arbitrary CFG into a DAG

Path Profiling of DAGs

- First Step---Edge Assignment
  - Assign a non-negative constant value Val(e) to each edge e in a DAG

```
foreach vertex v in reverse topological order {
  if v is a leaf vertex {
    NumPaths(v) = 1;
  } else {
    NumPaths(v) = 0;
    for each edge e = v->w {
      Val(e) = NumPaths(v);
      NumPaths(v) = NumPaths(v) + NumPaths(w);
    }
  }
}
```



Vertex v	NumPaths(v)
A	6
B	4
C	2
D	2
E	1
F	1

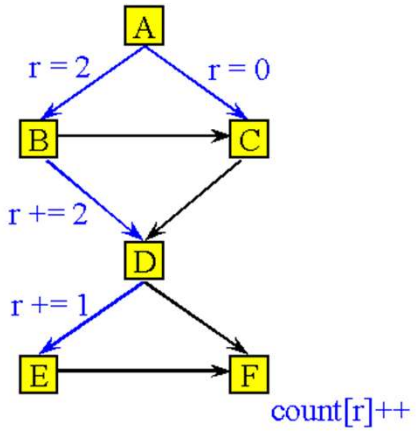
Path	Encoding
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

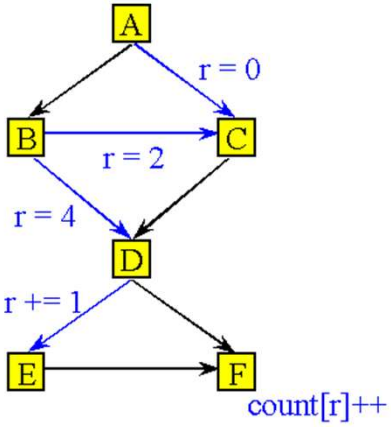
Any vertex with a single outgoing edge e, such as C and E, always has Val(e) = 0

Path identifier is a sum of edge values through the path

Path Profiling of DAGs

- Second Step---Edge Selection for Efficiently Computing Sums





Many ways to compute sums

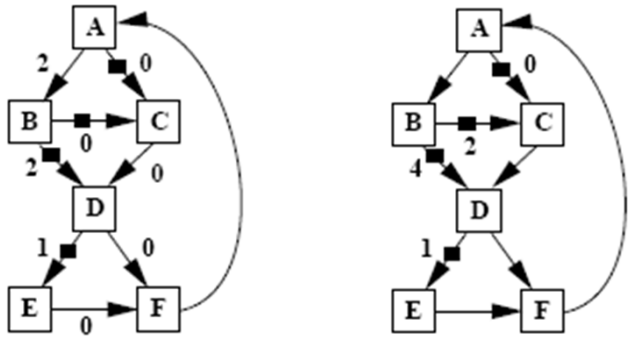
Find minimum operations to compute sums ?

- Uses the Event Counting Algorithm in the paper

  Thomas Ball. "Efficiently counting program events with support for on-line queries", ACM Transactions on Programming Languages and Systems, Sep 1994

Path Profiling of DAGs

- Second Step--- Event Counting Algorithm
  - Path identifier is preserved
    - Ensure that the sum of Incrementing values for any path P from ENTRY to EXIT is identical to the sum of Val(e) values for P
  - Transition events in the paths are reduced
    - Weigh edges by execution frequency
    - Instruments the least traveled edges
  - Example:
    - Transition number changes from 3 to 2 in the path (ABDEF)



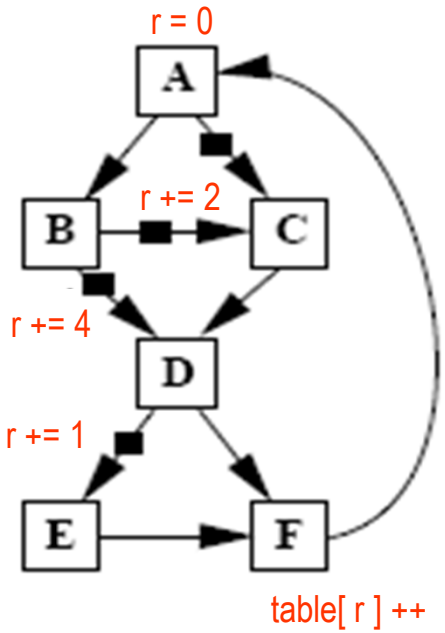
Path Profiling of DAGs

- Third Step---Inserting Instrumentation
  - Basic
    - Initialize: r =0 at ENTRY
    - Increment: r+=Inc(c) along chord c
    - Record: count[r]++ at EXIT
    - Postlude: Array is written out to permanent storage
  - Optimization (reduce memory access)
    - Initialize & Increment: r=Inc(c)
    - Increment & Record: count[r+ Inc(c)]++

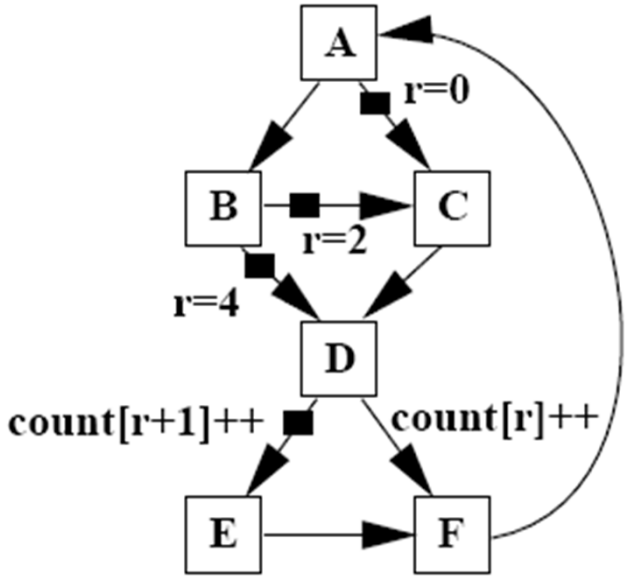
Path Profiling of DAGs

- Third Step---Inserting Instrumentation

Basic



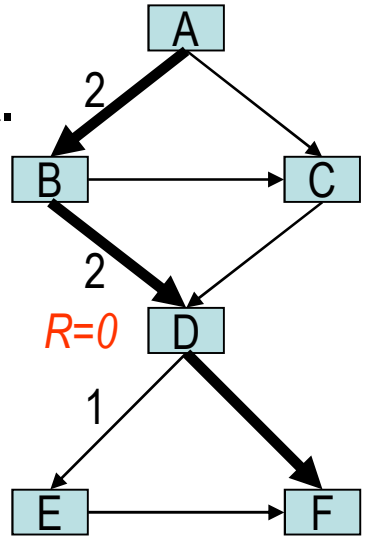
Optimization



Path Profiling of DAGs

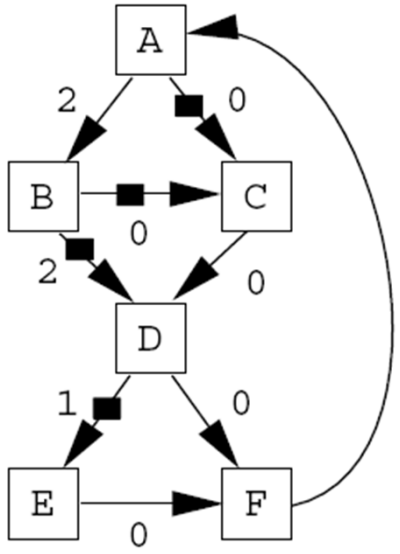
- Path Generation
  - Given information
    - R = path identifier
    - v = current block (initialized to entry block)
    - e = outgoing edge from the vertex v to w
    - Val(e) = edge value of the edge e
  - At each block, find e (v→w) , which is outgoing edge of v with the largest Val(e) ≤ R.

Example: path register is 4
- Regenerated path is ABDF



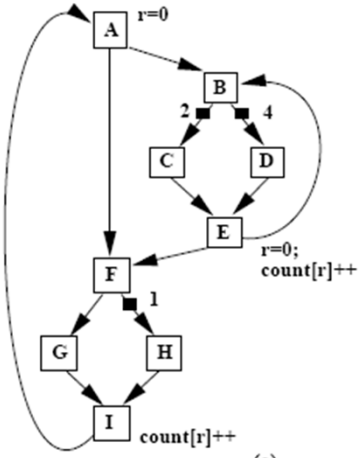
Arbitrary Control-Flow Graphs

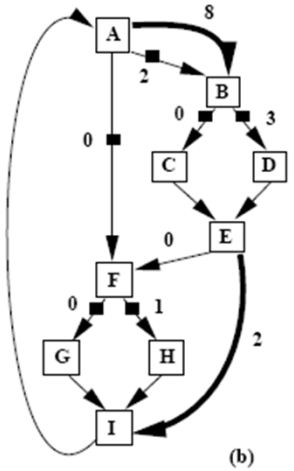
- Transforming general CFG to DAG
  - Control flow graphs generally contain cycles
  - Approach: Break cycles at loop backedge
  - Instrument each backedge with [count[r]++; r=0], which records the path upto the backedge and prepares to record the path after the backedge.

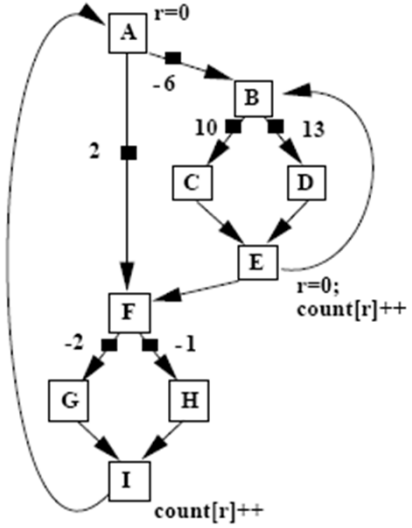


Arbitrary Control-Flow Graphs

- If there is a backedge (E → B)
  - Insert dummy edge (ENTRY→ B)
  - Insert dummy edge (E → EXIT)
  - Remove the backedge (E→B) (Eliminate all backedges except EXIT to ENTRY)
  - Apply the first two steps of Path Profiling Algorithm (Edge value assignment and chord increment)







Path	Path Sum
AFGI	*0*
AFHI	*1*
ABCEFGI	*2*
ABCEFHI	*3*
ABDEFGI	*5*
ABDEFHI	*6*
ABCE	4
ABDE	7
BCE	*10*
BDE	*13*
BCEFGI	8
BCEFHI	9
BDEFGI	11
BDEFHI	12

Arbitrary Control-Flow Graphs

- Dealing with Self Loops
  - Self loops are backedges with same source and target vertex
  - Approach: Add a counter along them to record the number of times they execute

Experimental results

Overhead is the increase in execution time due to profiling

Benchmark	Base Time (sec)	PP Overhead %	QPT2 Overhead %	PP/ QPT	Path Inc (million)	Edge Inc (x Path)	Hashed Inc %	Inst/ Inc
099.go	885.0	53.4	24.1	2.2	1002.4	1.5	27.7	33.2
124.m88ksim	571.0	35.6	18.7	1.9	4824.9	1.2	3.9	16.2
126.gcc	322.0	96.9	52.8	1.8	9.4	1.7	16.8	15.1
129.compress	351.0	19.4	21.9	0.9	3015.7	1.5	0.0	16.6
130.li	480.0	25.4	26.7	1.0	3282.4	1.4	1.2	16.8
132.ijpeg	749.0	17.4	16.3	1.1	1164.9	1.1	1.2	31.0
134.perl	332.0	72.9	51.5	1.4	1133.0	1.9	23.4	22.2
147.vortex	684.0	37.7	34.1	1.1	3576.3	1.5	23.7	20.3
CINT95 Avg:		44.8	30.8	1.4	22251.1	1.5	12.2	21.4
101.tomcatv	503.0	19.9	2.8	7.1	574.6	1.1	95.8	93.0
102.swim	691.0	8.4	0.6	14.5	163.4	1.0	0.2	162.9
103.su2cor	465.0	10.1	5.8	1.7	558.1	1.2	21.5	92.8
104.hydro2d	811.0	37.7	5.8	6.5	1690.7	1.7	77.8	43.1
107.mgrid	872.0	6.3	3.2	2.0	1035.2	1.0	7.7	133.5
110.applu	715.0	71.0	12.0	5.9	2111.4	1.1	99.7	44.8
125.turb3d	1066.0	5.5	7.4	0.7	2952.8	1.1	0.0	56.5
141.apsi	492.0	7.7	1.8	4.2	599.3	1.1	3.5	84.0
145.fpppp	1927.0	14.6	-2.6	-5.6	395.0	1.8	42.5	636.0
146.wave5	620.0	16.9	6.1	2.8	737.3	1.3	65.0	74.1
CFP95 Avg:		19.8	4.3	4.0	1081.8	1.2	41.3	142.1
Average:		30.9	16.1	2.8	1601.5	1.3	28.4	88.4

PP : path profiling

QPT : edge profiling

Path profiling overhead - 30.9% (5.5 to 96.9%)

Edge profiling overhead - 16.1% (-2.6 to 52.8%)

Experimental results

The fraction of paths predicted entirely correctly by edge profiling

Benchmark	Path Profile						Edge Profile Paths					Routines		
	Num Path	Longest Edge	Longest Inst	Avg Edge	Avg Inst	% Correct	Longest Edge	Longest Inst	Avg Edge	Avg Inst	Exec	Max Path	Avg Path
099.go	24414	105	314	10.9	33.2	4.3	84	252	5.0	19.3	407	1574	60.0
124.m88ksim	1113	138	360	5.8	16.2	29.8	138	360	4.3	9.1	220	70	5.1
126.gcc	9319	711	1074	7.4	15.1	20.8	711	1074	4.6	10.5	1027	163	9.1
129.compress	249	80	146	6.5	16.7	43.0	80	146	4.6	8.8	69	34	3.6
130.li	770	153	252	9.0	16.8	38.1	62	109	7.0	14.6	216	64	3.6
132.ijpeg	1199	139	416	7.0	31.0	36.4	139	202	5.1	22.2	252	194	4.8
134.perl	1421	123	305	10.7	22.2	24.5	115	207	7.3	16.7	233	125	6.1
147.vortex	2223	584	841	8.9	20.3	39.5	584	841	8.5	13.6	627	65	3.5
CINT Avg:	5088	254	464	8.3	21.4	29.5	239	399	5.8	14.4	381	286	12.0
101.tomcatv	421	83	326	4.1	93.0	49.6	82	201	3.9	25.2	146	56	2.9
102.swim	378	106	310	2.3	162.9	57.1	106	310	2.2	57.6	143	25	2.6
103.su2cor	905	136	954	6.7	92.8	45.9	73	781	5.5	59.3	209	127	4.3
104.hydro2d	1456	344	488	6.5	43.1	33.2	344	436	5.8	36.2	227	434	6.4
107.mgrid	589	83	320	2.3	133.5	44.8	78	320	2.1	15.8	160	73	3.7
110.applu	619	240	3557	3.7	44.8	54.1	240	3557	2.4	26.6	144	82	4.3
125.turb3d	674	162	692	7.1	56.5	46.6	162	692	5.1	28.2	189	39	3.6
141.apsi	1064	712	1196	6.1	84.0	40.8	136	734	4.6	69.0	242	54	4.4
145.fpppp	821	85	11455	14.9	636.0	25.8	76	11455	9.0	122.6	143	322	5.7
146.wave5	896	90	1180	5.2	74.1	47.8	90	1180	4.8	49.5	212	69	4.2
CFP Avg:	782	204	2048	5.9	142.1	44.6	139	1967	4.5	49.0	182	128	4.2
Average:	2696	226	1344	6.9	88.4	37.9	183	1270	5.1	33.6	270	198	7.7

Acknowledgement

- *http://drona.csa.iisc.ernet.in/~muralikrishna/teaching/spring2014/Papers2014/Ball_Larus_Presentation.pdf*
- http://pages.cs.wisc.edu/~larus/Talks/path_talk/sld001.htm
- *https://cse.sc.edu/~mgv/csce531sp10/presentations/531Sethia2010.**ppt***

Thanks