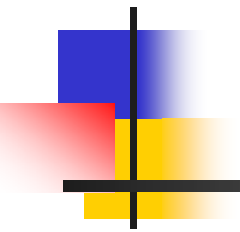
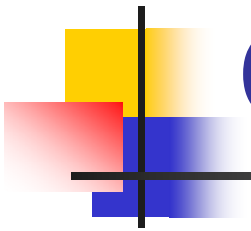
Efficient Path Profiling

Thomas Ball James R. Larus
Bell Laboratories Univ. of Wisconsin
(now at Microsoft Research)

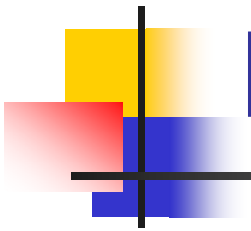


Presented by
Ophelia Chesley



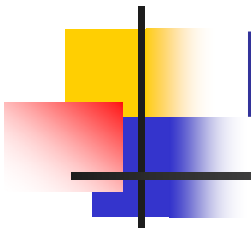
Overview

- Motivation
- Path Profiling vs. Edge Profiling
- Path Profiling Algorithm
- Arbitrary Control Flow
- Experimental Results
- Summary



Motivations

- Identify heavily executed paths for performance tuning (cache misses, page faults, etc.)
- Profile-directed compilation to focus optimization on frequently executed paths
- Software test coverage



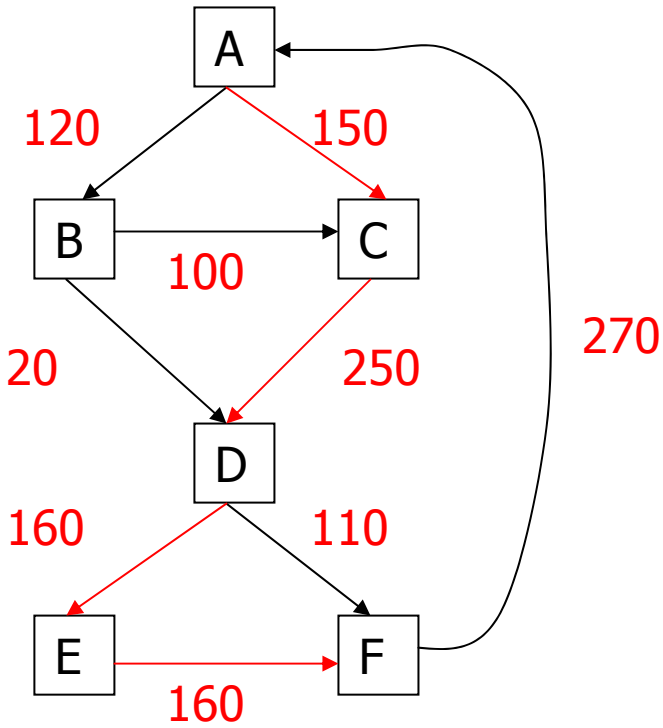
Edge/Block

- Inexpensive
- Inaccurate in predicting frequencies of overlapping paths
- Blocks/Edges are finite and linear to program size
- Increments memory (?)

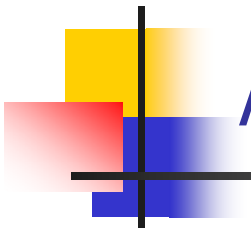
Path

- Could be expensive
- Potential acyclic paths is exponential to program size
- More accurate
- Induce edge profile
- Increments registers (for small # of paths, hash table otherwise)

Path versus Edge Profiling



Path	Prof1	Prof2
ACDF	90	110
ACDEF	60	40
ABCDEF	100	100



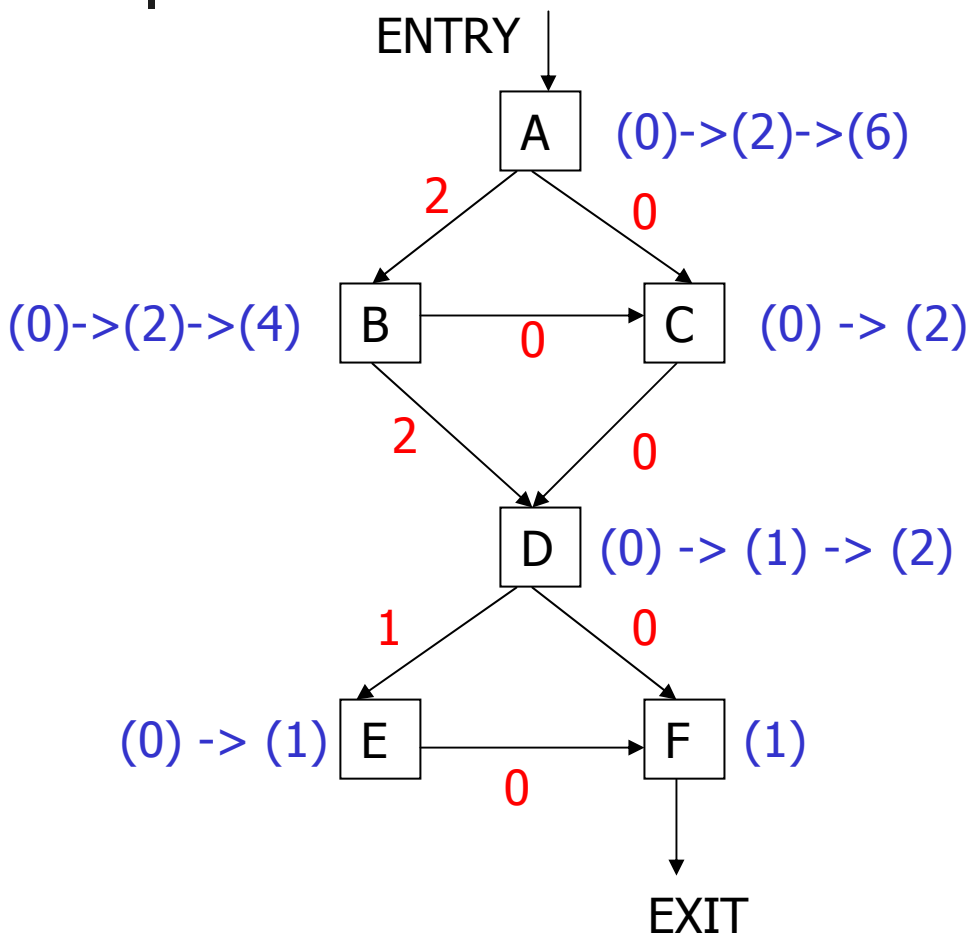
Algorithm for Intra-Procedural Path Profiling

- Assign unique path sum
- Select edges with minimal instrumentation
- Compute increments for these edges
- Instrumentation
- Use profiled data to derive paths

Path Representation

- CFG → DAG with EXIT and ENTRY
- v = vertex; e = edge;
- NumPaths(v) = # of paths from v to EXIT
- NumPaths(leaf vertex) = 1

Path Sum

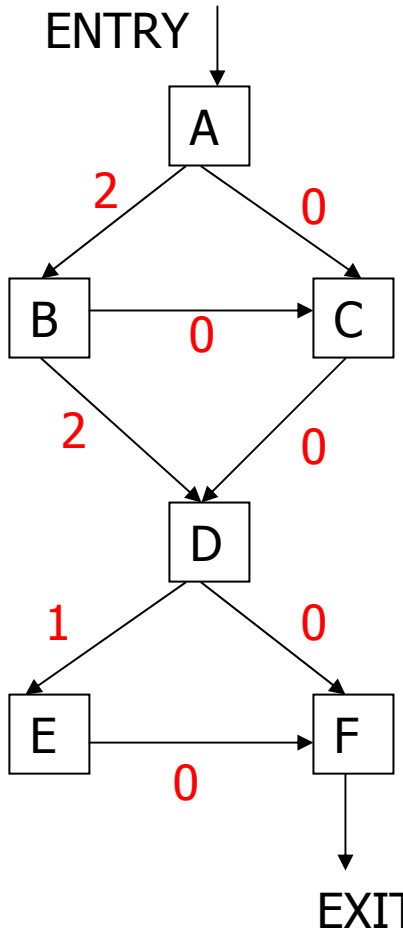


Reverse Topological Order

(NumPaths(v))

Val(e)

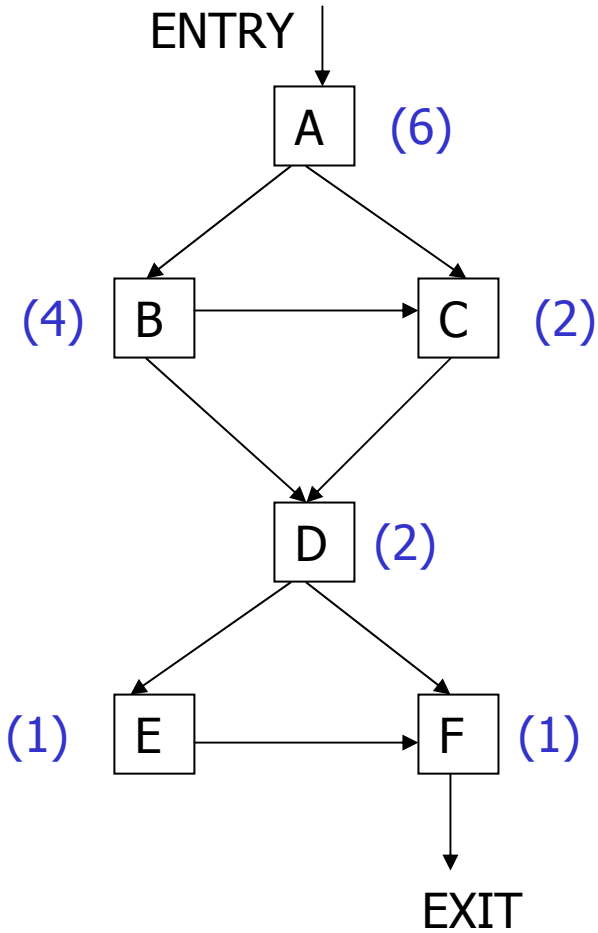
Path Sum



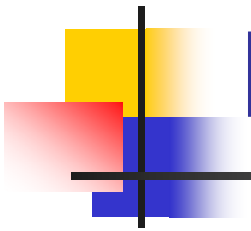
Path (sum of all edges)	Unique Encoding
ACDF	0
ACDEF	1
ABCDF	2
ABCDEF	3
ABDF	4
ABDEF	5

Index an array of counters

Path Sum



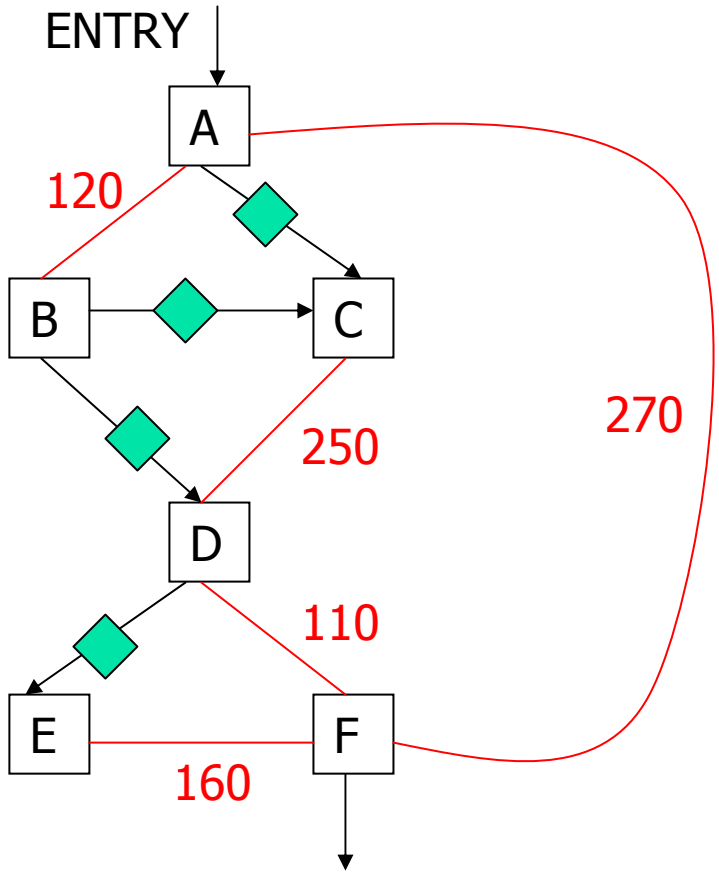
Vertex v	NumPaths(v)
A	6
B	4
C	2
D	2
E	1
F	1



Event Counting

- Needs to determine the minimal cost set for instrumentation
- Use Event Counting Algorithm (previous research):
  - Compute weight (execution frequency) for each edge
  - Identify the maximum spanning tree wrt edge weights
  - Use cords (edges not in MST) for instrumentation

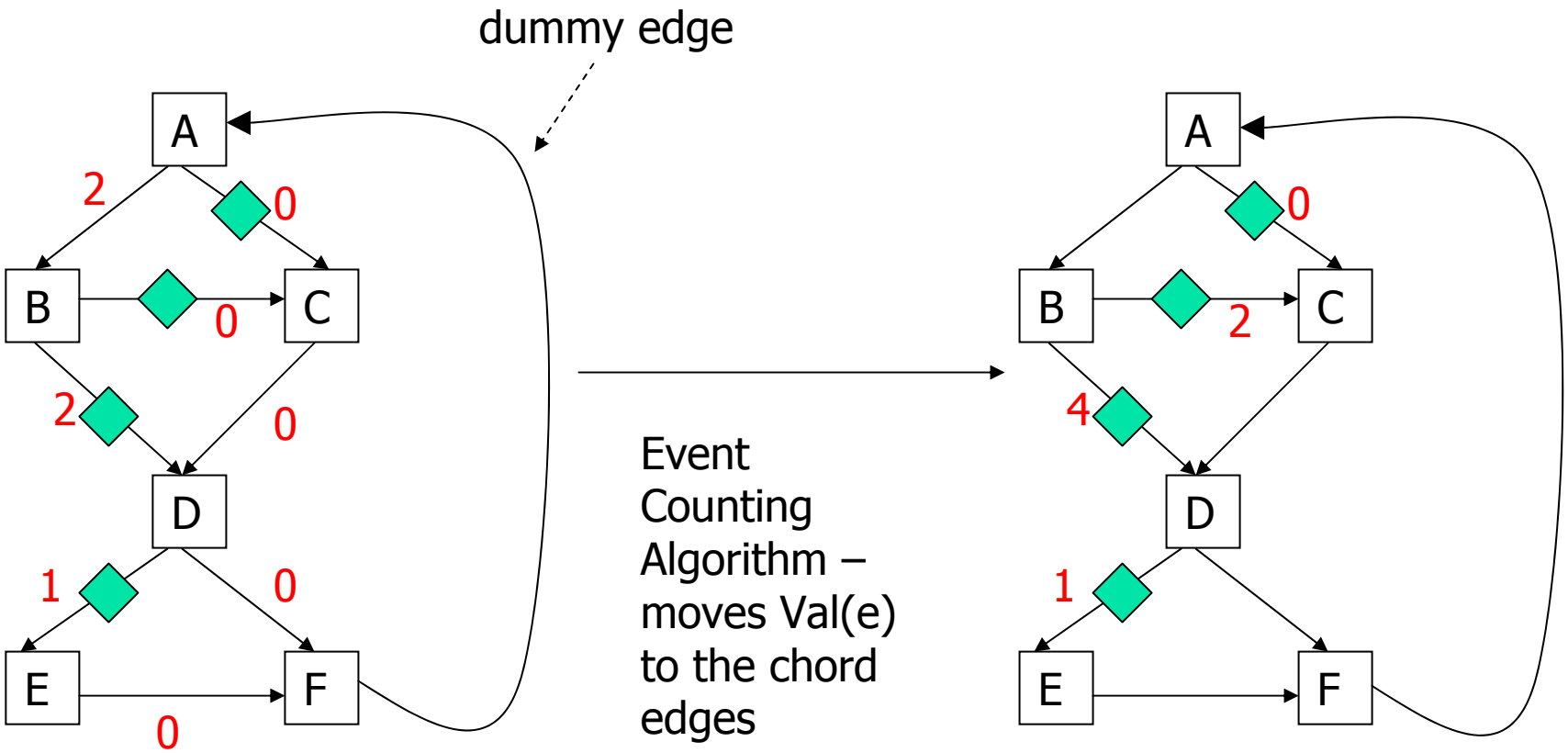
Event Counting



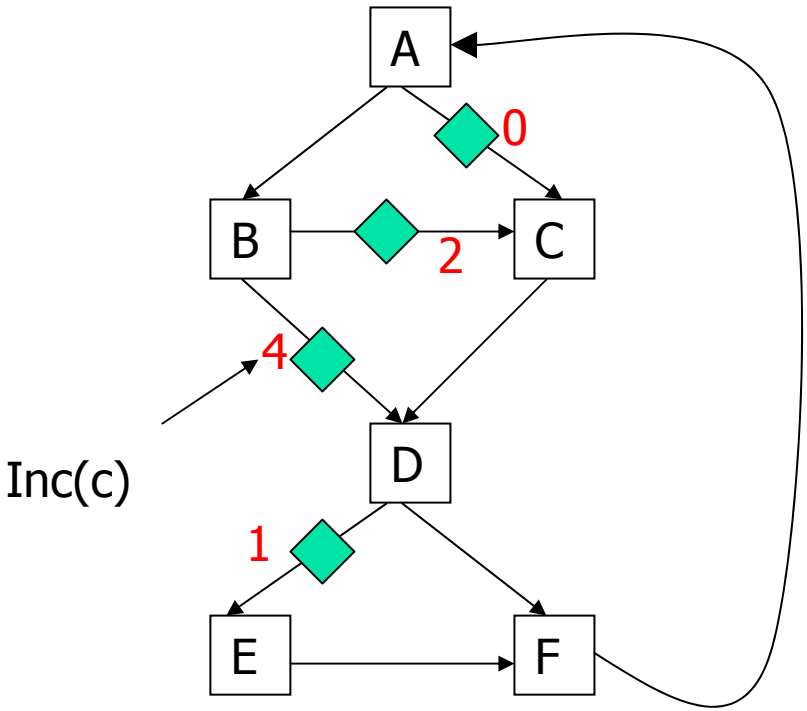
Max Spanning tree edge wrt edge weights

Min Spanning tree chord (least travelled edges)

Event Counting

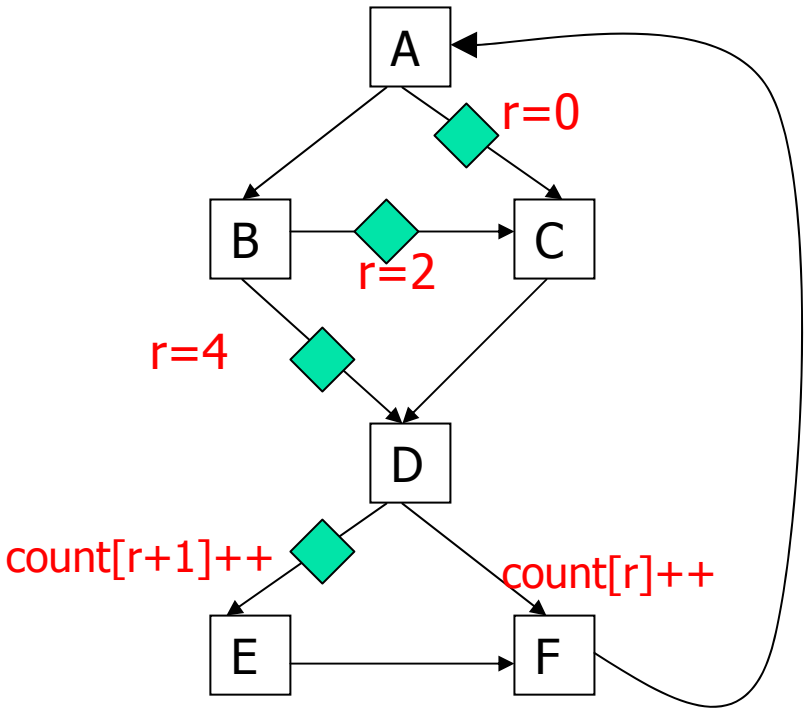


Path Instrumentation



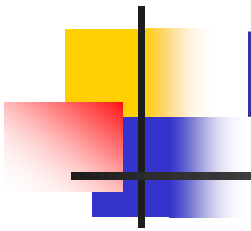
- Tasks:
  - Allocate and initialize an array of counters: count[]
  - Initialize and counter increment at each chord c: count[Inc(c)]++
  - count[r+Inc(c)]++ at last chord

Path Instrumentation

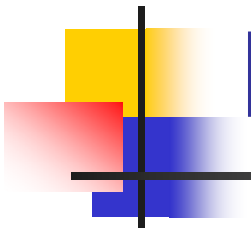


- Tasks:
  - Allocate and initialize an array of counters: count[]
  - Initialize and counter increment at each chord c: count[Inc(c)]++
  - count[r+Inc(c)]++ at last chord
  - count[r]++ at EXIT

Path Instrumentation



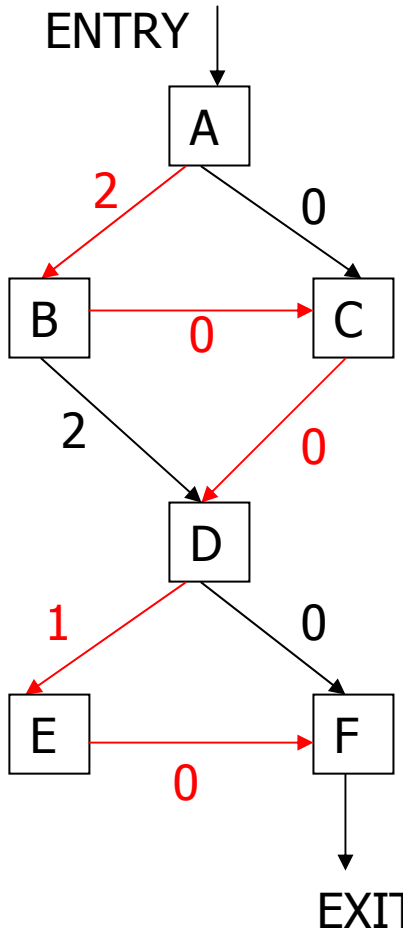
- Inc(c) is the counter's address in the array located in the path register at run time
- Inc(c) is limited by the instruction's immediate field
- During path sum calculation, visit the successor with the largest NumPath last → Val(e) is minimized → minimized Inc(c)



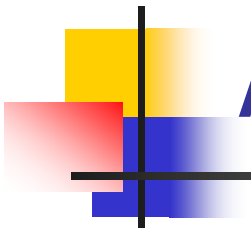
Regenerating Paths

- Find out which path using its path value (R) and Val(edge) computed earlier
  - Start with v = ENTRY
  - Select an outgoing edge v→w with largest Val(edge) <= R
  - R = R-Val(v→w); v = w

Path Regeneration



- R = 3
- v = ENTRY
- Select A→ B
- v = B; R = 3-2 = 1
- Select B→ C
- v = C; R = 1-0 = 1
- Select C→ D
- v = D; R = 1-0 = 1
- Select D→ E
- v = E; R = 1-1 = 0
- Select E→ F
- ABCDEF



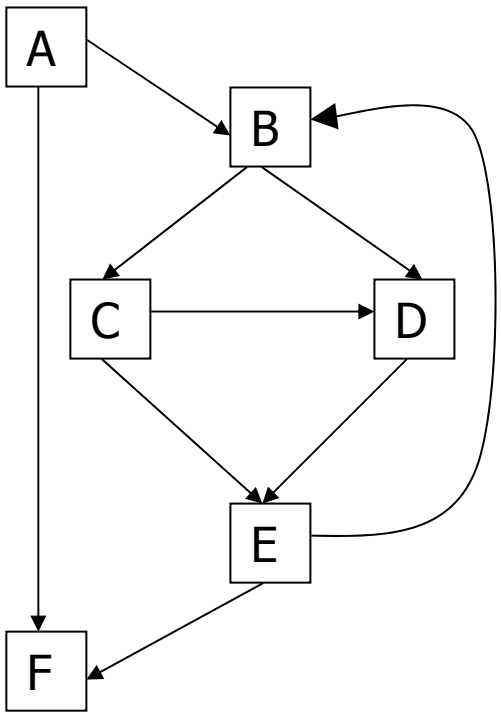
Arbitrary Control-Flow

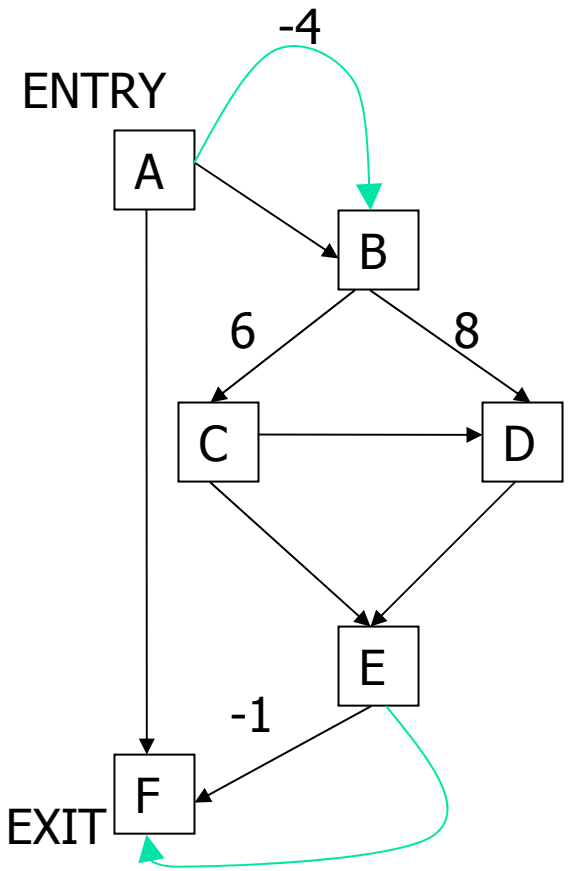
- Existence of backedges
- Existence of self loops
- Algorithm for DAG does not work
- Multiple paths could be assigned the same path value

Arbitrary Control-Flow

- Solutions for backedges:
  - Identify backedges w→v
  - Add dummy edge ENTRY→ v
  - Add dummy edge w→EXIT
  - Eliminate the backedges
  - Perform value assignment and chord increments as before

Arbitrary Control-Flow

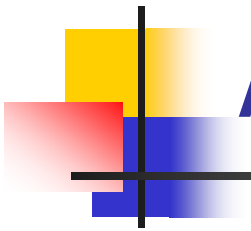




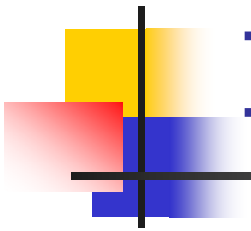
Path Values

AF	0
ABCEF	1
ABCE	2
ABDEF	3
ABDE	4
BCEF	5
BCE	6
BDEF	7
BDE	8

Arbitrary Control-Flow

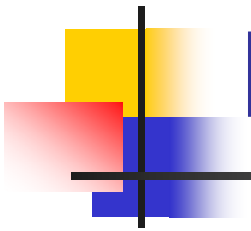


- Self-loop:
  - Cannot remove the edge or else there is no edge to instrument
  - Add a counter along the self loop to record the number of times they execute



Implementation

- Uses a profiling tool, PP builts on EEL(Executable Editing Library)
- EEL finds dead registers or spill the least used register for path profiling
- For more than 4000-6000 paths, PP replaces the array of counters with hash table
- When > 100,000,000 paths reachable from a node, PP removes outgoing edges before returning to value computation.



Experimental Results

- Compare PP to QPT2 (edge profiling tool built with EEL also)
- PP's average overhead 31%, as compare to QPT2's 16%
- In general, small programs have comparable overhead
- Comparable overhead for programs with long paths and path increments execute infrequently

PP Overhead

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	99.1	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

Minimal hashing

Hashing but infrequent path increments

Path Profiling versus Edge Profiling

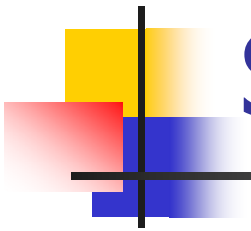
Benchmark	Path Profile						Edge Profile Paths					Routines		
	Num Path	Longest Edge	Longest Inst	Avg Edge	Avg Inst	% Correct	Longest Edge	Longest Inst	Avg Edge	Avg Inst	Exec	Max Path	Avg Path
099.go	24414	105	314	10.9	33.2	4.3	84	252	5.0	19.3	407	1574	60.0
124.m88ksim	1113	138	360	5.8	16.2	29.8	138	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

Edge profiling predicts shorter paths

Fraction of paths predicted by edge profiling

ref versus *train* datasets

Benchmark	Common Paths			Common Instructions	
	Number	Static	Dynamic	Number	Dynamic
099.go	13670	52.8%	99.5%	32823357655	98.6%
124.m88ksim	800	71.1%	97.1%	68794935104	93.3%
126.gcc	9058	63.5%	94.6%	140747413	88.9%
129.compress	201	78.2%	99.2%	49206874856	98.0%
130.li	522	67.8%	87.4%	48285966561	87.7%
132.ijpeg	1099	85.3%	99.9%	36098785257	100.0%
134.perl	687	41.8%	71.9%	16041579783	63.8%
147.vortex	2160	96.0%	100.0%	72589443060	100.0%
101.tomcatv	418	90.9%	100.0%	53422202701	100.0%
102.swim	373	98.2%	100.0%	26607743709	100.0%
103.su2cor	880	95.5%	100.0%	51766673091	100.0%
104.hydro2d	1336	90.1%	100.0%	72892099182	100.0%
107.mgrid	577	97.3%	100.0%	138157253535	100.0%
110.applu	582	87.5%	100.0%	94622106074	100.0%
125.turb3d	651	94.5%	100.0%	166724577396	100.0%
141.apsi	992	91.0%	91.2%	37648860455	74.3%
145.fpppp	764	91.9%	99.9%	250229811735	99.6%
146.wave	5821	88.5%	98.8%	52996842799	96.9%



Summary

- Presented a path profiling algorithm with comparable cost to edge profiling
  - Compact encoding of paths
  - Minimal instrumentation
- Identify longer paths with more instructions than edge profiling
- Short training dataset can cover most of the paths
- Potential to improve profile driven compilation