Ideas in Profiling and Feedback-Directed Compilation

rich vuduc

CS 265

Spring 2000

- Can we make measurements of our programs that characterize their behavior on some set of inputs? (profiling)
- How can programmers use this information to tune their applications? (hardware/software tools)
- To what extent can we automate the process of tuning based on empirical data? (e.g., feedback-directed compilation)

Outline

- Introduction and Brief History
- Assisting "Classic" Code Optimizations
- Path Profiling
- Tools
- Zany Ideas (and More Zany Ideas)
- To Profile or Not to Profile?

- **Introduction and Brief History**
- Assisting “Classic” Code Optimizations
- Program Paths
- Tools
- Zany Ideas (and More Zany Ideas)
- To Profile or Not to Profile?

A Brief History of Profiling

- "Empirical analysis of FORTRAN programs" (Knuth 1971)
  - let's measure something about the code we write
  - target a particular language and/or suite of applications
  - developer oriented tools
- `prof`; `gprof` (Graham, et al., 1982)
- Hardware analyses, e.g., branch prediction (1980s)
- Profile information for compiler optimization (1990s)
- Extension to paths (Larus, Ball, et al., mid-1990s); hardware counter-based metrics
- Today: Lots of zany ideas! (later)

- *profile, v.*: measure something "interesting," e.g.,
  - execution frequencies of basic blocks
  - execution times of blocks, paths
  - number of cache misses
  - high frequency data and address variable values
- *trace, v.*: select "interesting" execution paths, e.g.,
  - most (and least!) frequently executed path
  - path containing the most cache misses

- Goals
  - low overhead (run-time & storage)
  - accurate measurements
  - minimal compiler changes
  - minimal intervention by programmer

Profiling Issues & Goals (2)

- High-level issues
  - Collect what?
  - At what granularity?
  - On what inputs? (Wall '91)
  - "Quantum" measurement/observation effects?
  - Aggregation of multiple profiles? (Kistler, Franz '97)
  - What to do with resulting data?

Example: prof/gprof

- `gprof` (Graham, Kessler, McKusick, '82)
  - tool for developers
  - measured subroutine execution frequencies and times
  - "call graph perspective"
  - execution times: randomly sampled
  - compiler must insert a little bit of code
  - programmer compiles with a flag!

Outline

- Introduction and Brief History
- **Assisting “Classic” Code Optimizations**
- Path Profiling
- Tools
- Zany Ideas (and More Zany Ideas)
- To Profile or Not to Profile?

Assisting classic code optimizations

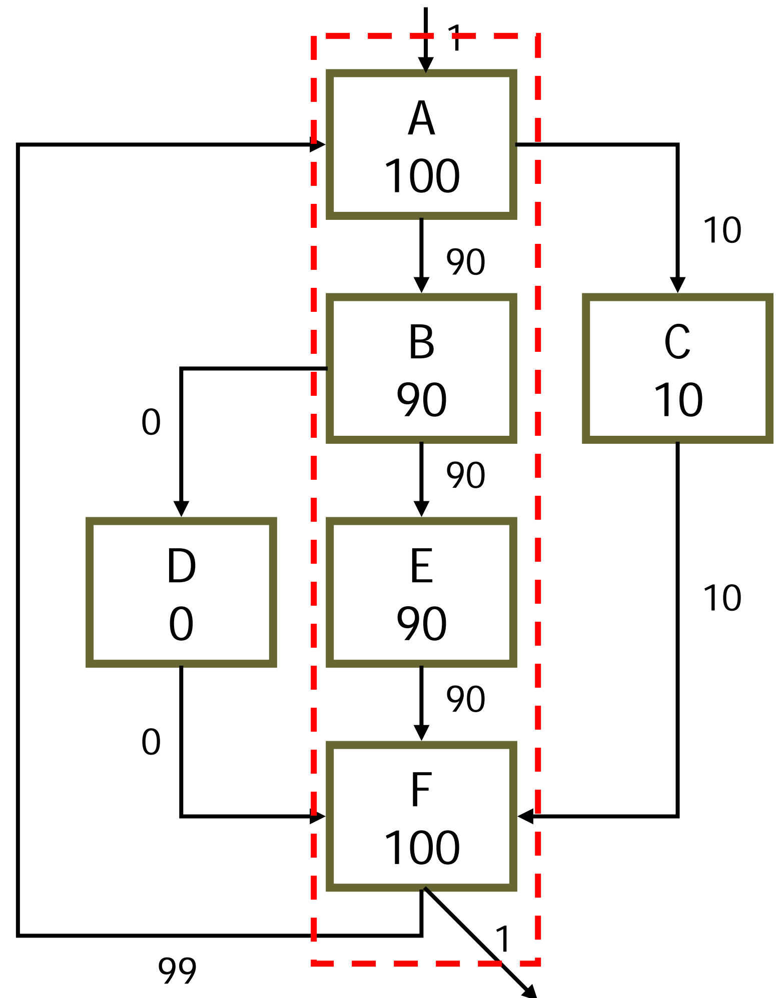
- IMPACT compiler (UIUC, Chang, et al., ‘91)
  - constant propagation; copy propagation
  - **common subexpression elimination**
  - redundant load/store elimination
  - **dead code removal**
  - loop invariant code removal
  - loop induction variable elimination
  - global variable migration
- Idea: optimize frequently executed paths of a “weighted control flow graphs”

Weighted Control Flow Graphs

- Intraprocedural
- Single entry, single exit
- Basic blocks A-F
- #'s = exec. freqs.
- also, “edge profile”

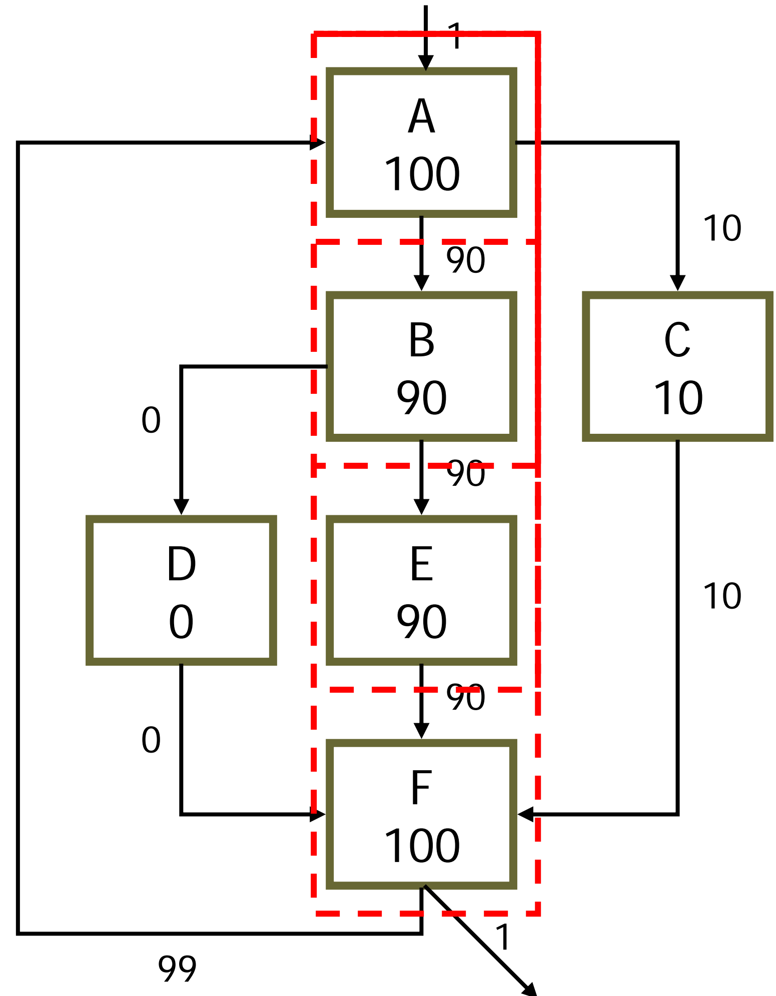
Super-blocks

- Representation of a frequent path
  - Sequence of blocks
  - Single entry
- Optimize together



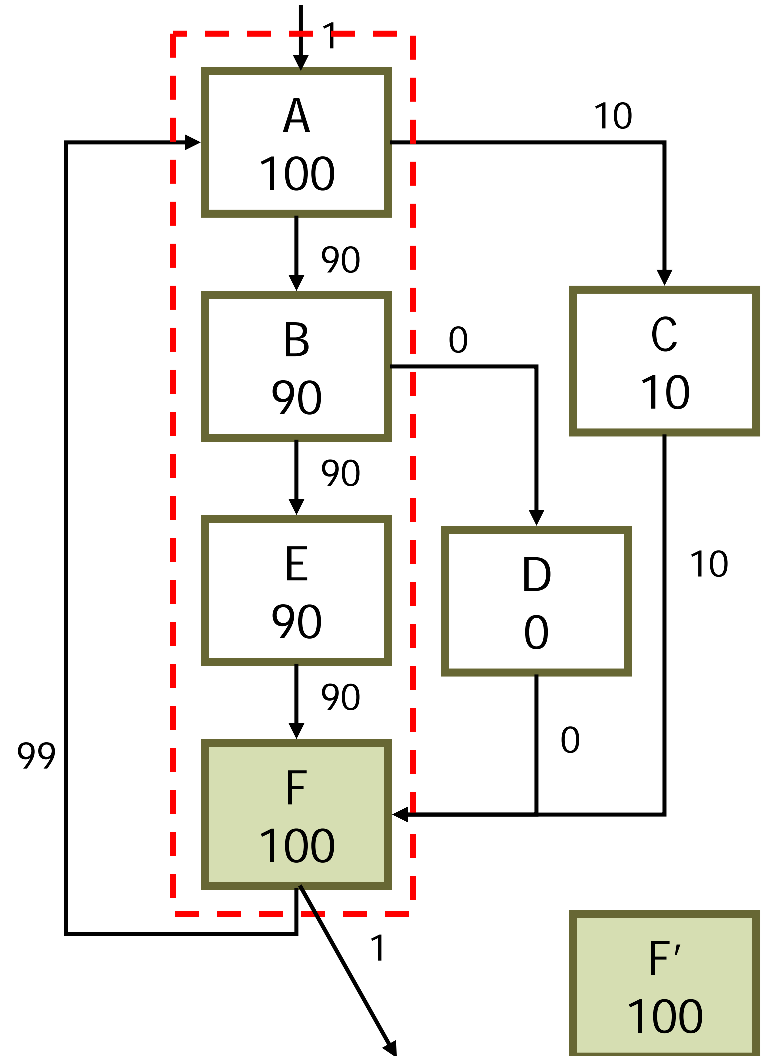
Transforming the WCFG (1)

- Trace selection
  - Start at most frequent block (A)
  - Add blocks along most likely out-going edge
  - Do in both directions
  - Repeat on remaining nodes



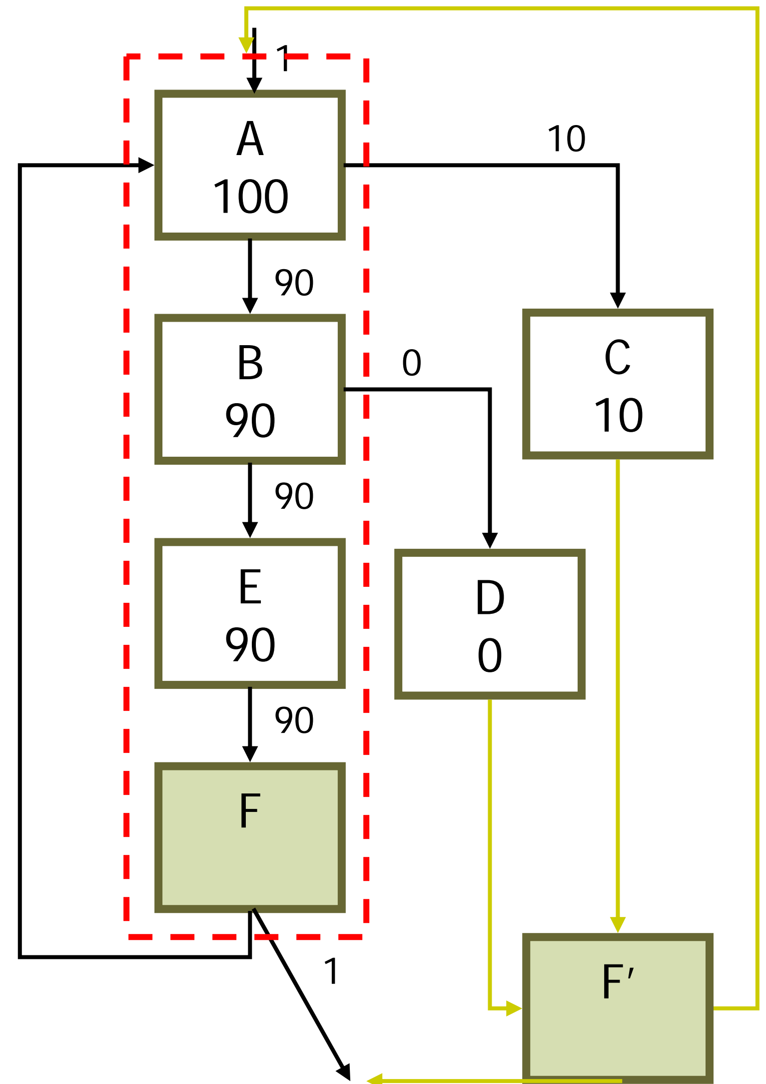
Transforming the WCFG (2)

- Tail-duplication (roughly)
  - Find first block with external entry edges (but not the head)
  - Duplicate
  - Edges: redirect incoming, duplicate outgoing
  - Repeat on its successors within the trace



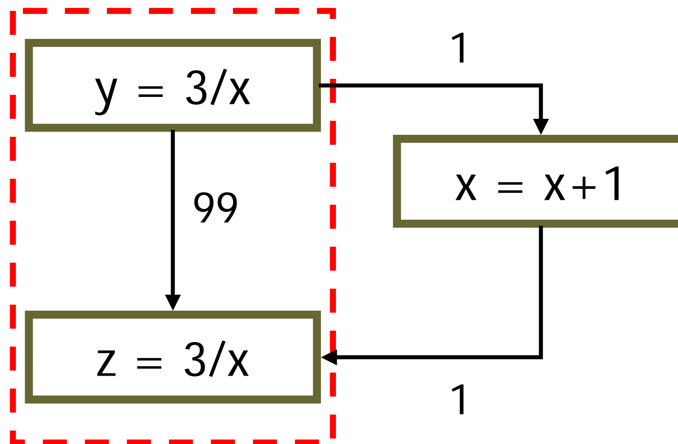
Transforming the WCFG (2)

- Tail-duplication (roughly)
  - Find first block with external entry edges (but not the head)
  - Duplicate
  - Edges: redirect incoming, duplicate outgoing
  - Repeat on its successors within the trace
- Fix-up frequencies if desired
- Lots of code duplication

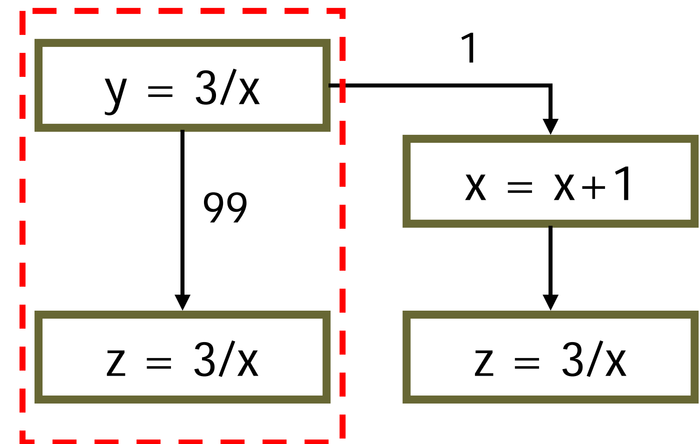


Example: Common Sub-expressions

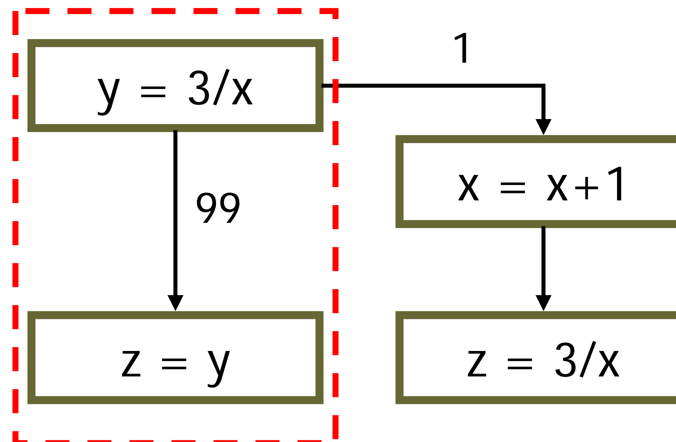
Original:



Transformed:



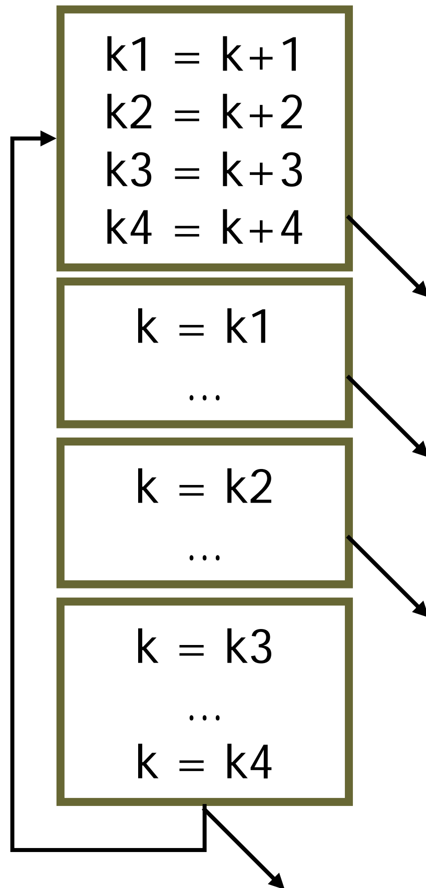
After CSE:

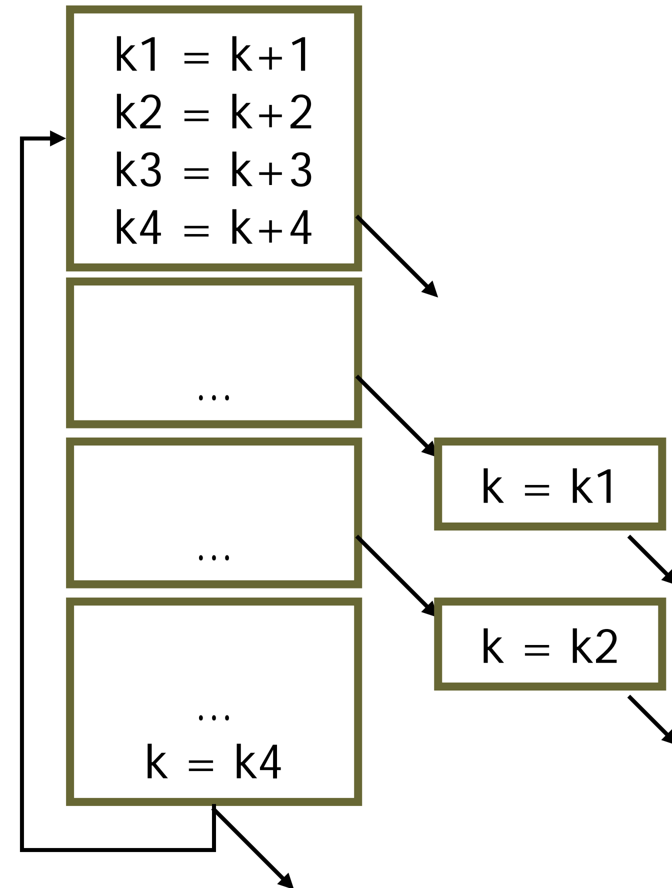


Example: Dead code "removal"

Original: Unrolled loop, index k

Final: after moving statements





Impact of IMPACT Optimizations

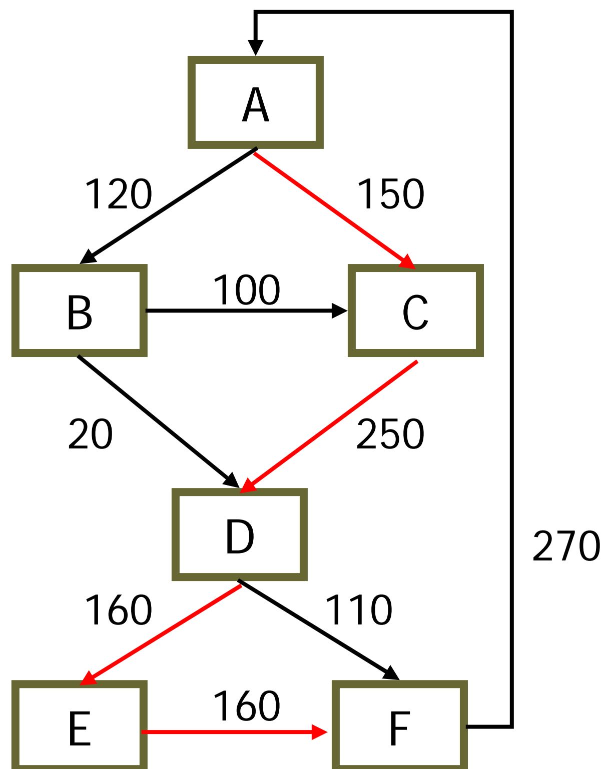
name	lines	+size	IMPACT	+prof	Vendor	gcc
mpla	38k	13%	1	1.18	.95	.87
xlisp	8k	20%	1	1.16	.88	.76
espresso	7k	7%	1	1.03	.98	.87
cccp	5k	3%	1	1.04	.93	.92
lex	3k	8%	1	1.01	.99	.96
tbl	3k	6%	1	1.03	.98	.93
eqn	3k	10%	1	1.25	.92	.91
yacc	2k	9%	1	1.08	1.00	.90

Outline

- Introduction and Brief History
- Assisting “Classic” Code Optimizations
- **Path Profiling**
- Tools
- Zany Ideas (and More Zany Ideas)
- To Profile or Not to Profile?

Edge Profiling Inaccuracies

- Overlapping paths can lead to inaccurate frequent path identification



Estimated frequent path: ACDEF

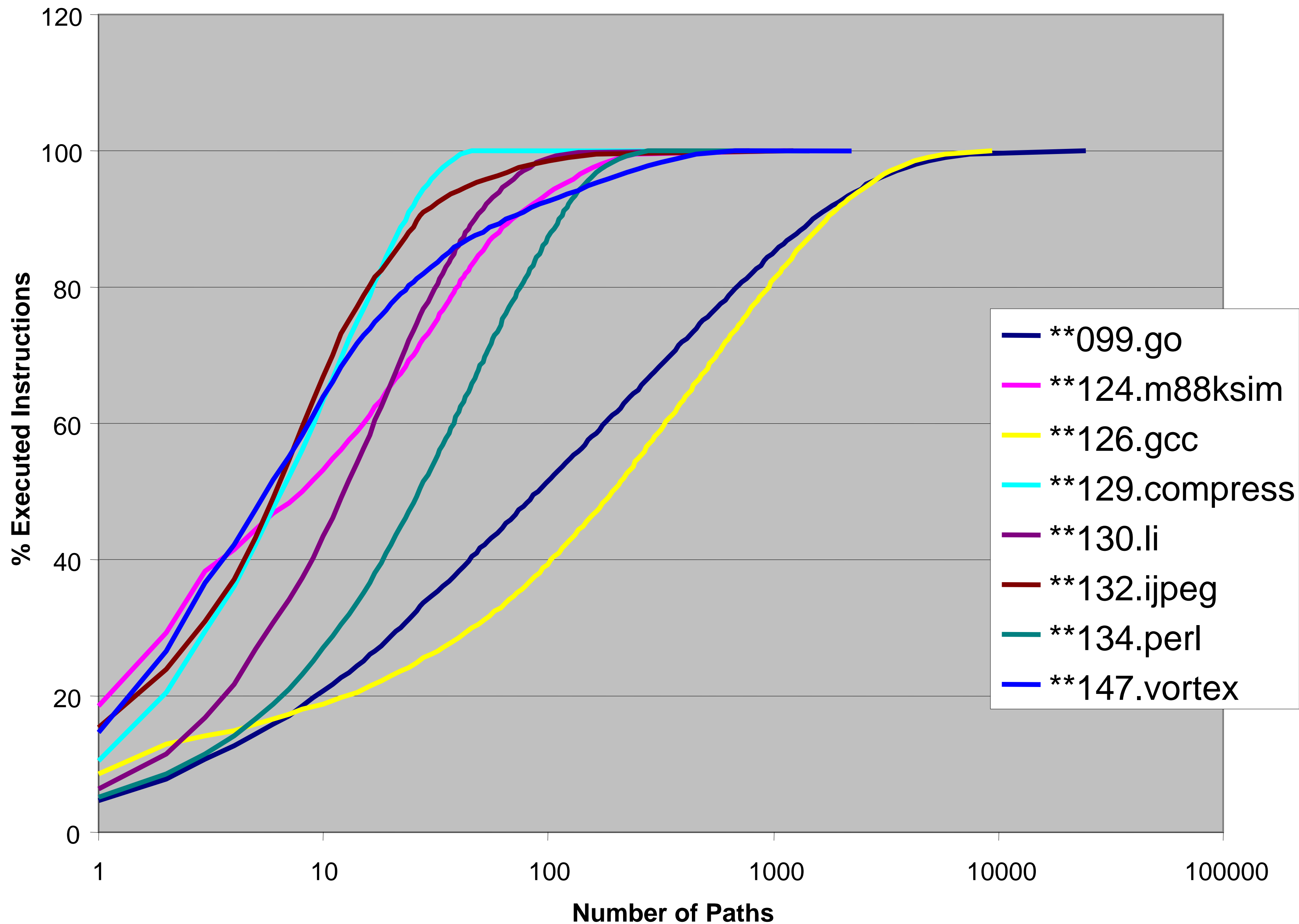
But these two path profiles **also** satisfy this edge profile!

path	prof 1	prof 2
ACDF	90	110
ACDEF	60	40
ABCDF	0	0
ABCDEF	100	100
ABDF	20	0
ABDEF	0	20

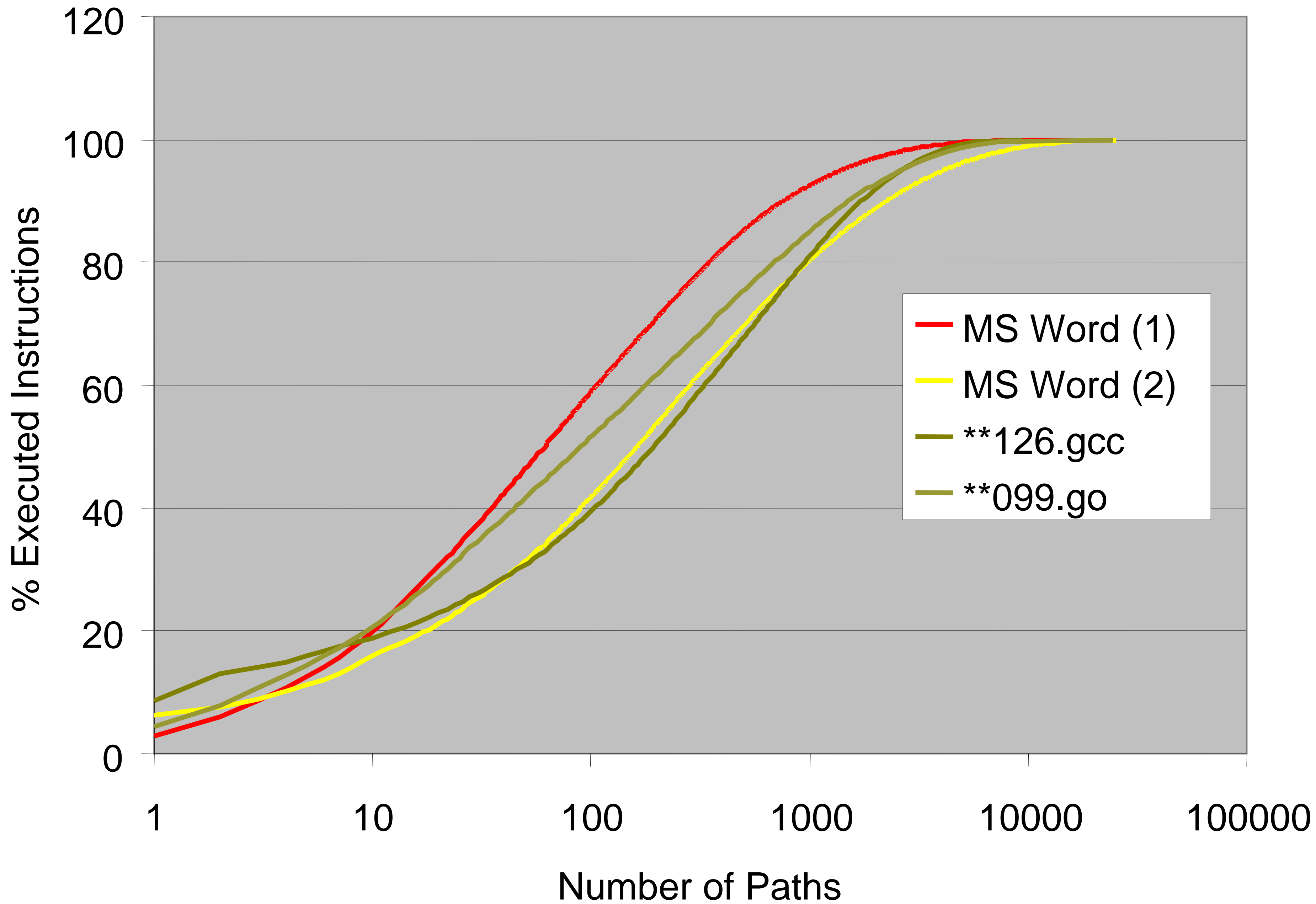
The Case for Path Profiling (1)

- Greater accuracy
- Enables new measurements via hardware counters
- BUT there are lots of paths!
  - Combinatorial explosion in number of paths
  - What about self-loops? Cycles in general?

➔ Need a way to track paths efficiently



Cumulative distribution of instructions in SPEC95 integer benchmarks (Larus & Ball '99).



Throwing Microsoft Word into the stew… (Larus & Ball '99)

The Case for Path Profiling (2)

Int Benchmark	# of paths	**% EP correct**
099.go	24.4k	4.3
124.m88ksim	1.1k	29.8
126.gcc	9.3k	20.8
129.compress	249	43.0
130.li	770	38.1
132.ijpeg	1.2k	36.4
134.perl	1.4k	24.5
147.vortex	2.2k	39.5

The Case for Path Profiling (3)

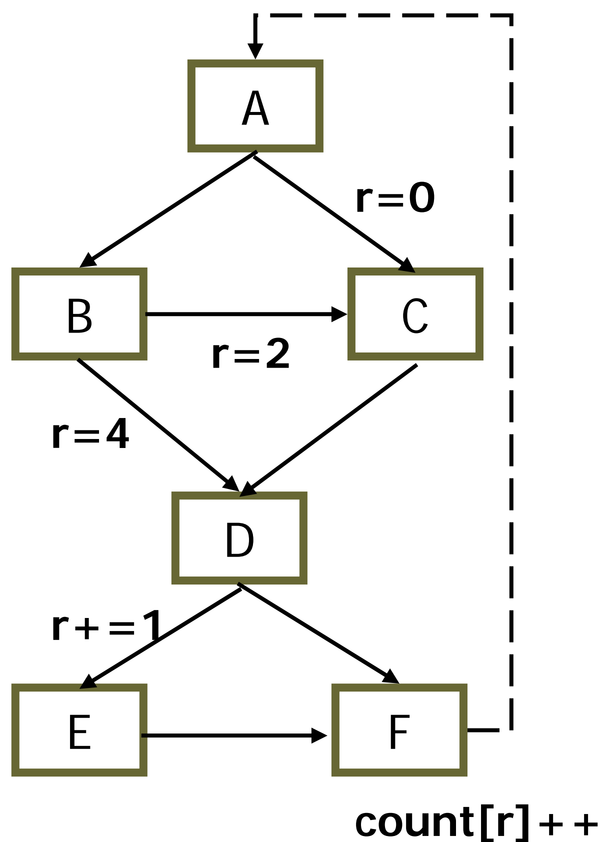
FP Benchmark	# of paths	**% EP correct**
101.tomcatv	421	49.6
102.swim	378	57.1
103.su2cor	905	45.9
104.hydro2d	1.4k	33.2
107.mgrid	589	44.8
110.applu	619	54.1
125.turb3d	674	46.6
145.fpppp	821	25.8

Efficient path profiling

- (Ball and Larus, ‘95)

- Assume: DAG, single entry, single exit
- Intraprocedural path finding algorithm
  - Add dummy edge from exit → entry
  - Uniquely and compactly number paths
  - Select edges to instrument
  - Insert instrumentation
- Special transformations for cyclic graphs

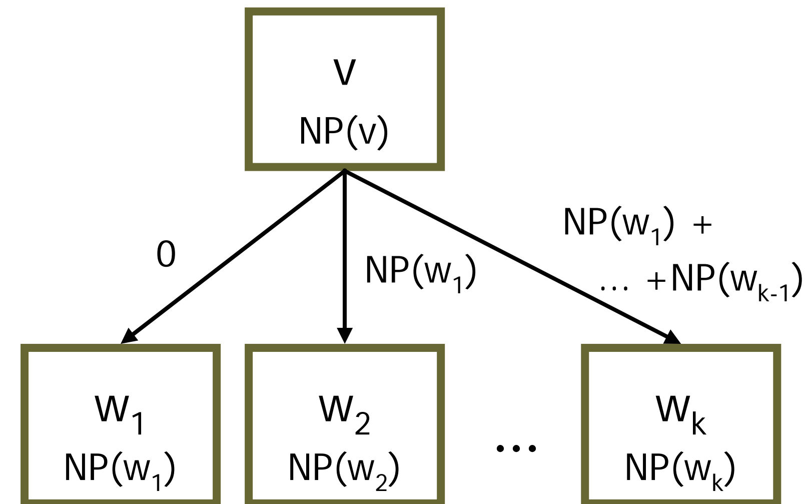
EPP: Goal



- For now, ignore loop-back edge
- *r* is the path number
- *count[r]* is the frequency of path *r*
- Each acyclic path from A to F has a unique number
- The path numbering is compact

EPP: Numbering paths

- Let NP(v) be the number of paths from v to the exit
- Maintain Val(e) for each edge e
- Sum of Val(e) along a path is the path number
- NP(exit) = 1, NP(v)=0

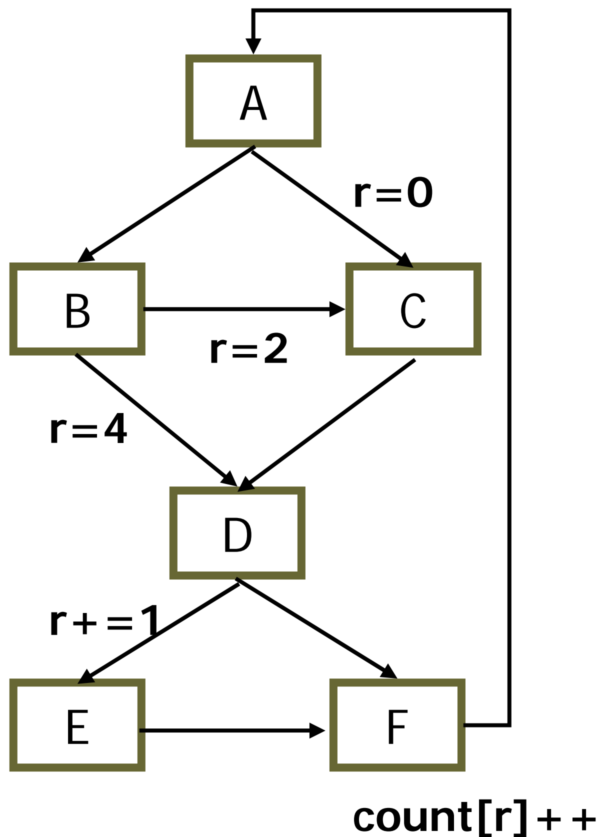


- In reverse topological order, visit each v, and for each edge $v \to w_i$
  - $Val(v \to w_i) = NP(v)$
  - $NP(v) += NP(w_i)$

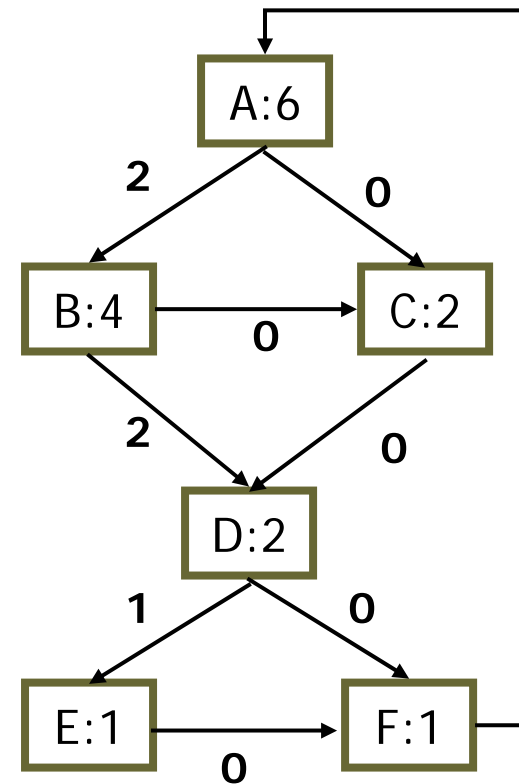
- When done, the sum of *Val(e)* along any path is the path's unique number

EPP: Number paths (example)

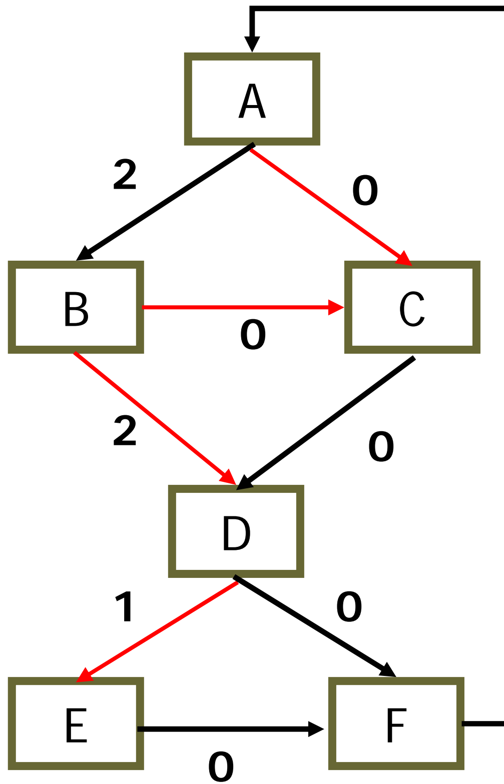
- Recall goal:



- So far:

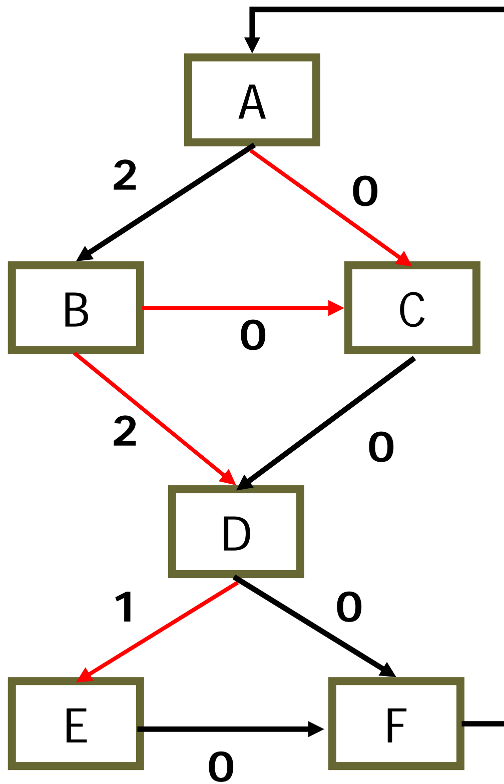


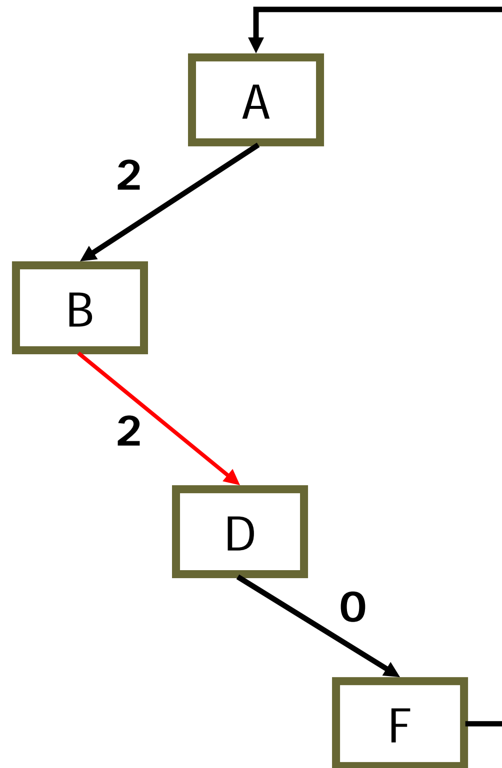
EPP: Efficient sums

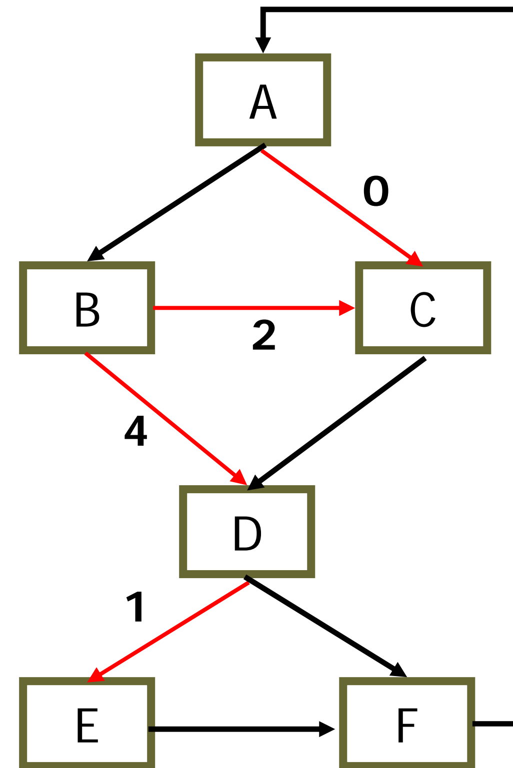


- Want to avoid summing (instrumenting) along every edge or every block
- Find a spanning tree (dark), chords (red)
- For each chord *c*, let Inc(*c*) be the sum of edges along the unique cycle it creates
- Instrument the chords

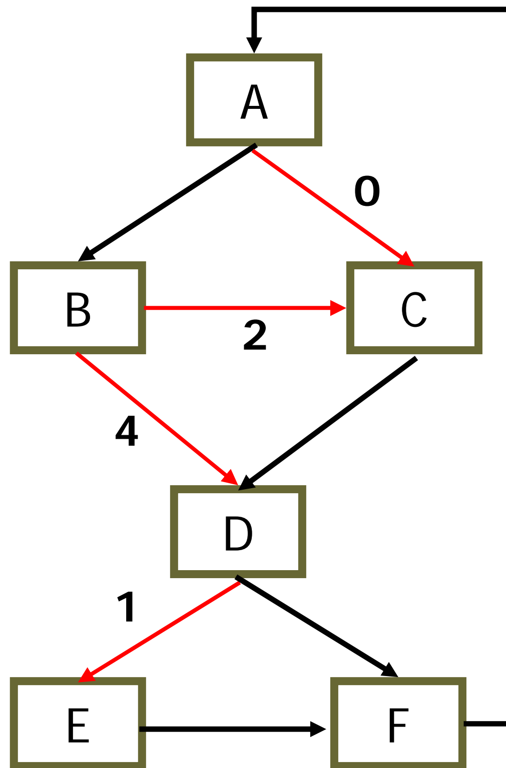
EPP: Efficient sums (example)







EPP: Instrumenting



- r = 0 at entry A
- count[r]++ at exit F
- r += Inc(c) at each chord
- Optimizations:
  - r = Inc(c) if c is the first chord on every path from A to F containing c
  - Can update count[r+Inc(c)] if c is the last chord on every path from A to F w/ c

(See paper for pseudo-code)

EPP: Fixing Cyclic Graphs

- Add counter reset code to back edges
  - e.g., count[r]++ ; r = 0
  - But staying in loop duplicates path numbers!
- For all back edges $v \rightarrow w$, add dummy edges
  - start $\rightarrow$ w
  - v $\rightarrow$ exit
  - This creates new paths from entry to exit that will get unique encoding numbers
- Remove back edges, and run the DAG-based algorithm
- Special case for self-loops if desired

- Whole Program Paths (Larus '99)
  - Complete, compact record of entire program's control flow (wow)
  - Automatic identification of so-called *hot sub-paths*
- Accumulate a hardware counter metric (Ball and Larus '97)
  - Like cache misses
  - Data structure extensions to efficiently record calling context information for procedure-level

EPP: Improving Data-flow Analysis

- Ammons and Larus '98; compare to WZ91 constant propagation stuff
- Uses *qualified data-flow analysis*
  - (Holley and Rosen '81)
  - Traditional DFA: "What can be said about the data-flow value x at vertex v?"
  - QDFA: Attach a finite automaton to CFG to track states (in this case, paths) to ask: "What can be said about x at v given that we are in state q?"
- Good results (10% speedup on large programs like gcc, go, mk88sim), but mixed results on other programs (sometimes slower!)

More Optimization Ideas

- Edge profiling
  - Inlining C (IMPACT '92)
  - Inlining C++ (Aigner & Holzle '95)
  - Instruction scheduling (IMPACT '94)
- Path profiling
  - Gupta, Berson, Fang
    - Path profile assisted dead code elimination and partial redundancy elimination using speculation ('97)
    - Resource-sensitive profile-directed data flow analysis ('98)
- Value profiling (Calder, et al. '99)
- Path spectra (Larus '99)
  - To direct function cloning (Way and Pollock '98)

- Introduction and Brief History
- Assisting “Classic” Code Optimizations
- Path Profiling
- **Tools**
- Zany Ideas (and More Zany Ideas)
- To Profile or Not to Profile?

A Visual Tool for Programmers

Hot Path Browser

File Profile View Go Bookmarks Options

Open Select Close Home Back Forward Add Bookmark Bookmarks

Browser View Source View

Path I	Proc	Frequen	Lengtl	Executed Ir
53	RBDelete__7IntSet0	18	28	1350
6	***RBDelete__7IntSet***	***21***	***20***	***1218***
7	RBDelete__7IntSet0	18	22	1116
11	RBDelete__7IntSet0	17	23	1088
57	RBDelete__7IntSet0	12	29	924
10	RBDelete__7IntSet0	15	21	900
56	RBDelete__7IntSet0	12	27	876
52	RBDelete__7IntSet0	9	26	639
47	RBDelete__7IntSet0	3	26	216
43	RBDelete__7IntSet0	3	25	210

Procedure Name	Total Pa	Executed F	Executed Instr
RBDelete__7IntSe	***72***	***10***	***8537.0***

Paths

File: IntSet0.cpp

```
RBNode* IntSet0::RBDelete(RBNode* z)
{
  RBNode* y;
  RBNode* x;

  if ((z->left == nilNode) || (z->right == nilNode)) {
// z is a leaf or has exactly one child
    y = z;
  } else {
// z is an interior node
    y = treeSuccessor(z);
  }

  if (y->left != nilNode) {
    x = y->left;
  } else {
    x = y->right;
  }

  x->parent = y->parent;

  if (y->parent == nilNode) {
    root = x;
  } else if (y == y->parent->left) {
    y->parent->left = x;
```

- Profilers (a la gprof) widely available
- Executable editor: EEL
- Viz tool: Hot Path Browser (Ball ‘98)
- Path spectra for debugging and testing (Larus ‘99)
  - Y2K debugging
  - Automated testing procedures
- Idea: educational tools?

- Introduction and Brief History
- Assisting "Classic" Code Optimizations
- Path Profiling
- Tools
- **Zany Ideas (and More Zany Ideas)**
- To Profile or Not to Profile?

Zany Ideas: Automatic Tuning

- Target: Numerical kernels (matmul, FFTs, LU)
- Idea
  - generate many versions of kernel, each optimized differently
  - time each version
  - pick "the best"
- Examples
  - PHiPAC (Bilmes, Asanovic, Demmel, et al., @ Berkeley)
  - ATLAS (Dongarra, Whaley @ UTenn)
  - FFTW (Frigo & Johnson, @ MIT) [1999 Wilkinson Prize Winner]

- Statistical machine-learning for branch prediction
  - Determining feature sets for better branch prediction based on a corpus of code and profile data
  - Calder, et al., ‘97

- Genetic algorithms for optimization
  - Determining a good order in which to apply a sequence of transformations for a parallel application
  - Nisbet ‘97

Outline

- Introduction and Brief History
- Assisting “Classic” Code Optimizations
- Path Profiling
- Tools
- Zany Ideas (and More Zany Ideas)
- **To Profile or Not to Profile?**

The Same and Not the Same

- Central Idea: Use Data to Guide Decision Making
- Question: How does profiling and feedback-directed compilation relate to dynamic compilation and JITs?
- Answer (maybe): Complementary
  - Run-time Code Generation: profiling can inform decisions on, e.g., when to specialize
  - Just-in-time Compilation: profiling can inform decisions on, e.g., when to inline

(Conclusion & Opportunities)

- A good idea, simple analyses practical
- Data is plentiful; how do we use it?
- For compiler optimization
  - Several existing uses, but not much by way of empirical data
  - How to use hardware counter-based metrics
  - What about benchmarks
- Opportunity for better tools

- Profiling meets data mining?