Towards Automated Generation of Time-Predictable Code[1]

Daniel Prokesch, Benedikt Huber,
Peter Puschner
{daniel,benedikt,peter}@vmars.tuwien.ac.at

Institute of Computer Engineering
Vienna University of Technology, Austria

July 8th, 2014
WCET 2014, Madrid

[1] Part of this work was supported by the FP7-ICT Project 288008 Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST), http://www.t-crest.org/

Motivation

Uncertainty in program execution time

Hardware

Duration of operations dependent on internal state

Software

Different inputs lead to paths of varying execution time

The Single-Path Approach

A proposed solution

- Singleton execution path
- Execution time stable w.r.t. varying input data

Benefits

…of single-path / low execution time jitter

- Reduce complexity of path analysis
- Increase predictability
- Control loops: performance and stability
- Security: Avoid timing-based side-channel attacks

The Single-Path Approach

Basic Idea

Eliminate input-data dependent control flow.

Predicated execution

```
(guard) instruction;
```

If-conversion

```
if (cond) {
  x = a + 1;
} else {
  x = b - 2;
}
```

```
( cond) x = a + 1;
(!cond) x = b - 2;
```

The Single-Path Approach

Input-data independent control flow

Treatment of loops:

```
while (cond)
  // max N times
  {
    stmts;
  }
```

```
finished := false;
for i = 1 .. N {
  if (!cond)
    finished := true;
  if (!finished) {
    stmts;
  }
}
```

Single-Path Transformation Rules

Set of rules to transform from high-level language to single-path sequence of predicated statements [Puschner et al., 2012]

- Clear conceptual understanding
- BUT:

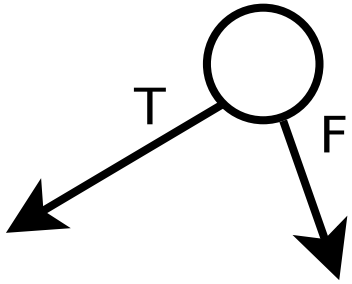
Not sufficient for implementation in a compiler backend!

This work

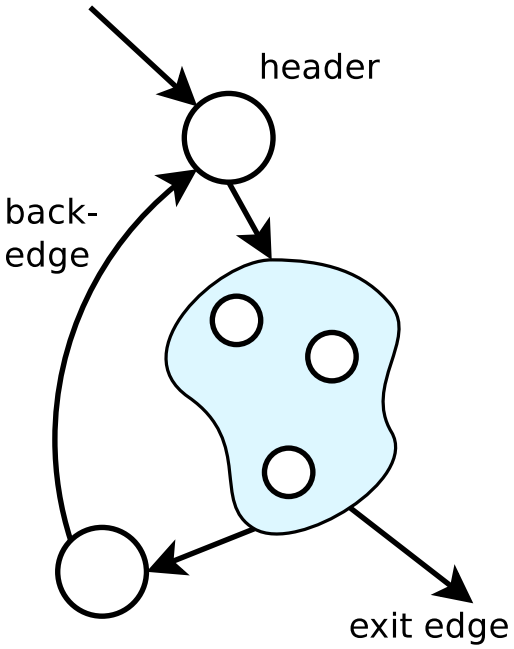
Single-Path Transformation from a low-level perspective as graph transformation on control-flow graphs

Control-flow Graph

Assumptions



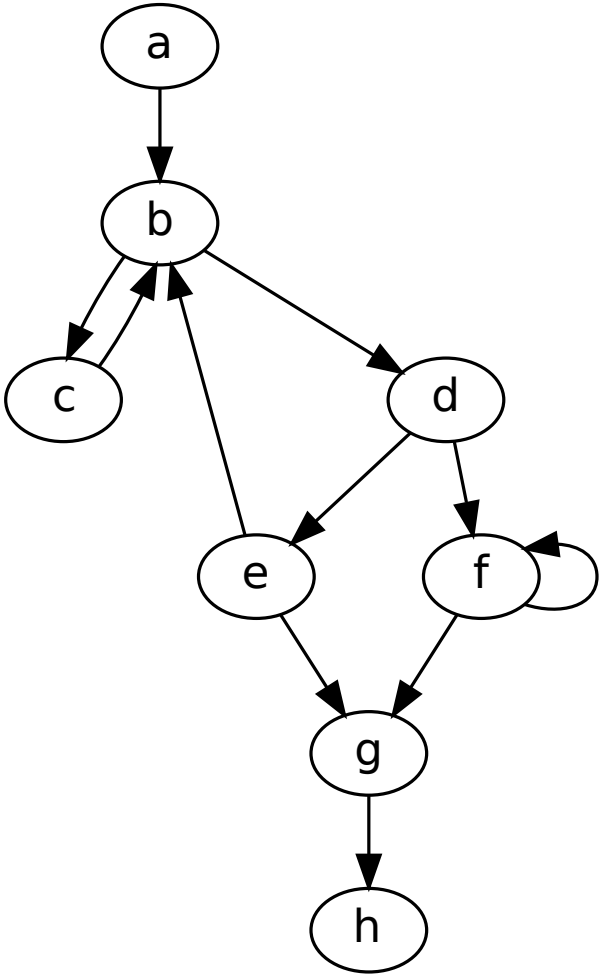
- Nodes with at most two successors
- Branch condition



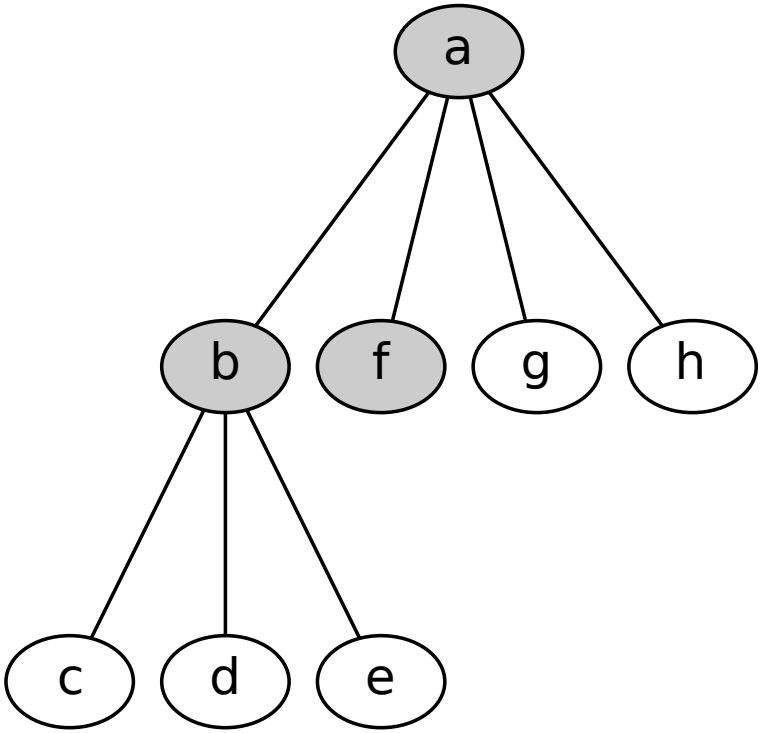
- Loops: assume reducible control flow graph

Control-flow Graph

Loop header tree



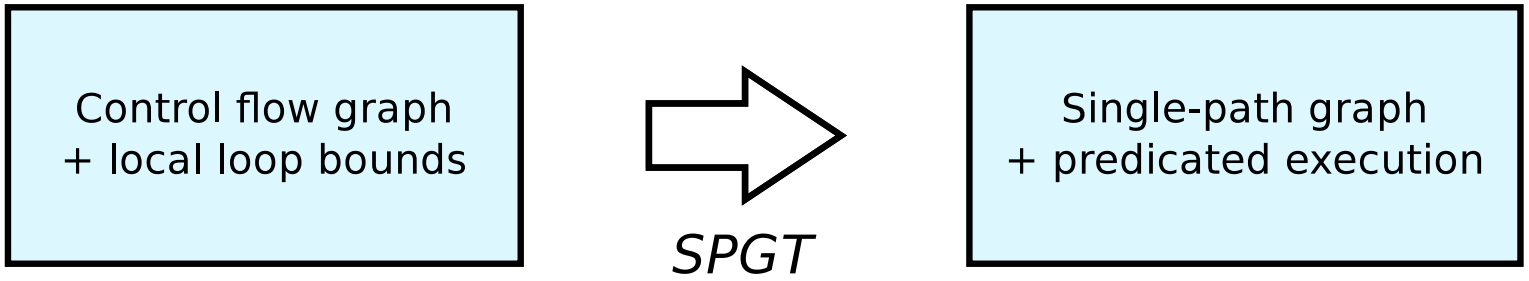
Control flow graph



Loop header tree

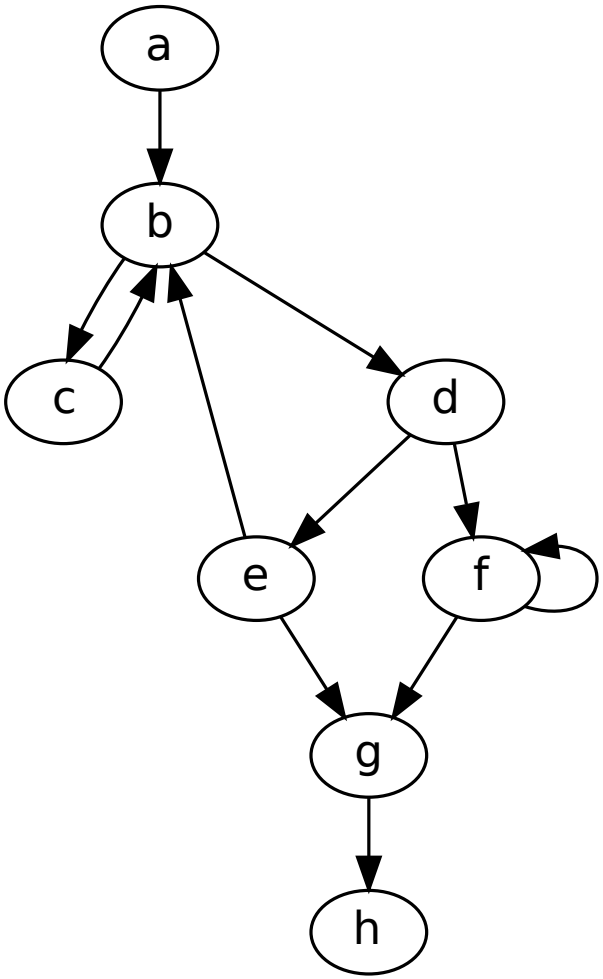
The Single-Path Graph Transformation

- Admissible paths in CFG: local loop bounds
- Predicated Nodes
- Semantic actions on predicates
  - Node may set predicate to (negated) branch condition
  - Edge may set/clear predicates
  - Edge may copy value of predicate

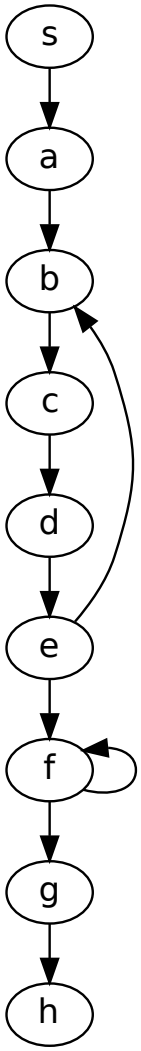


The Single-Path Graph Transformation

Example



Control flow graph

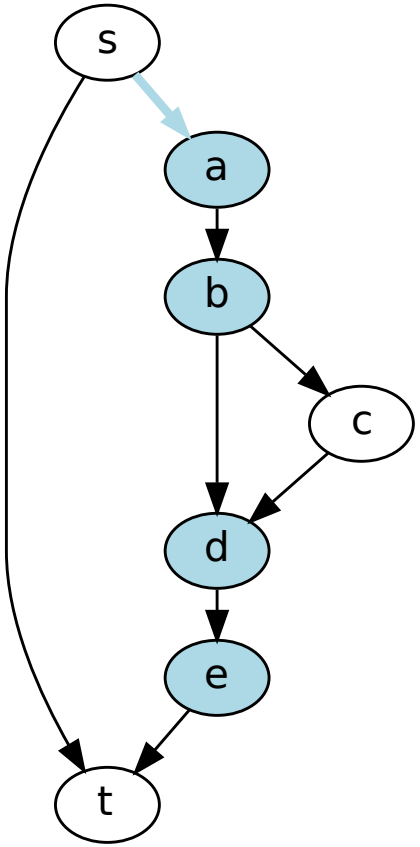


Single-path CFG

Acyclic regions

The RK algorithm

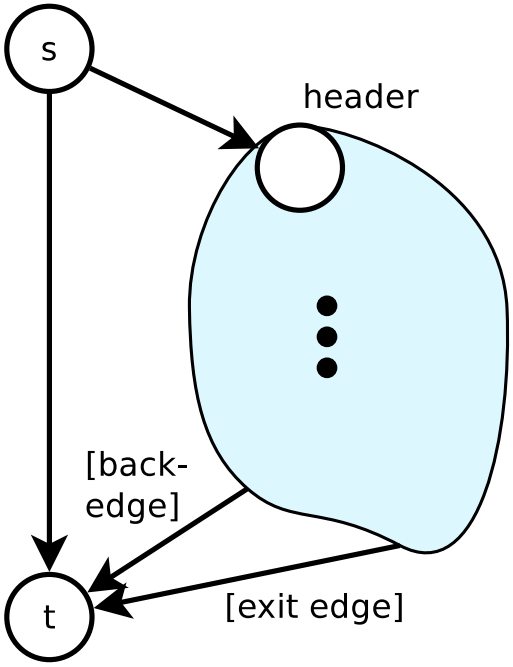
[Park and Schlansker, 1991] describe transformation in acyclic case



- Partitioning by Control Dependence
- Nodes with same set of CD: same predicate
- Predicate assignments at source of CD edges
- Initialization with *false*
- Topological sort order

Loops

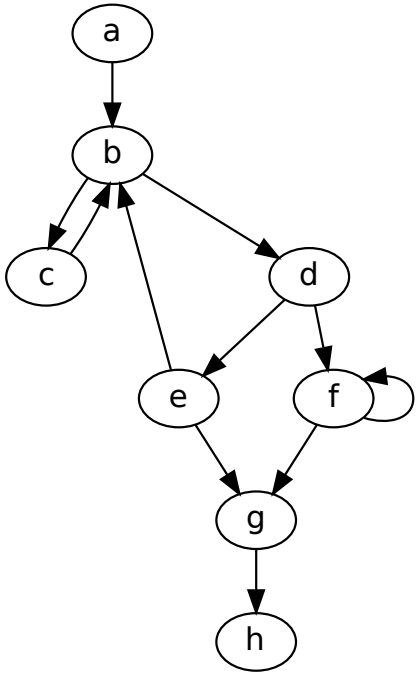
Key observation: Represent loops compacted into a single node



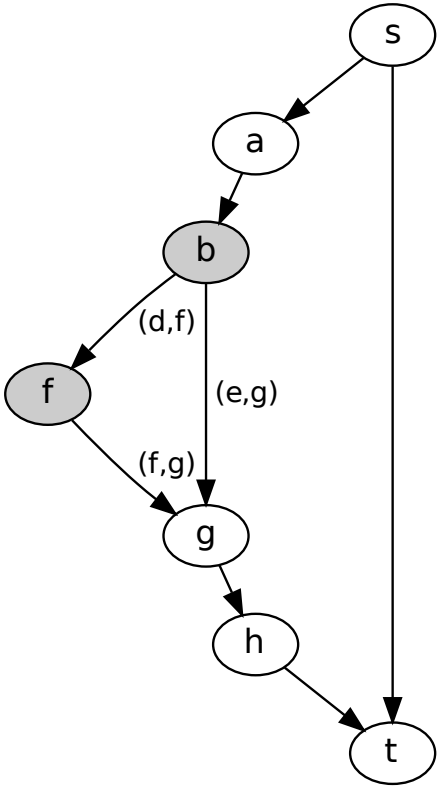
- Forward control flow graph of loop nodes
- Extend by entry, exit node
- Inner loops compacted
- Proceed as in acyclic case

Loops

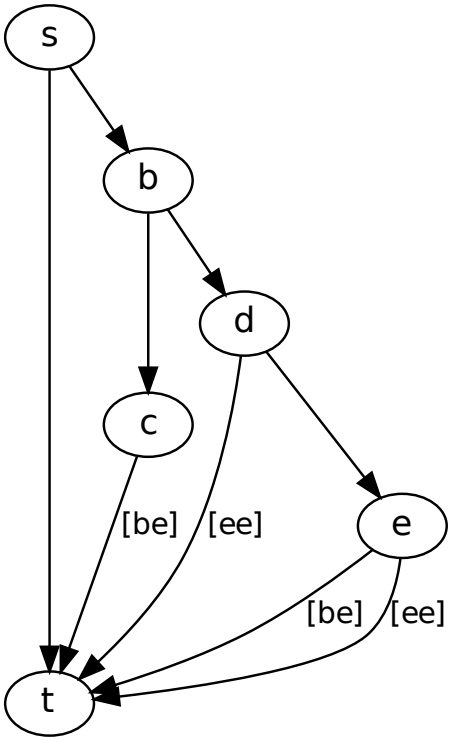
Example



Control flow graph



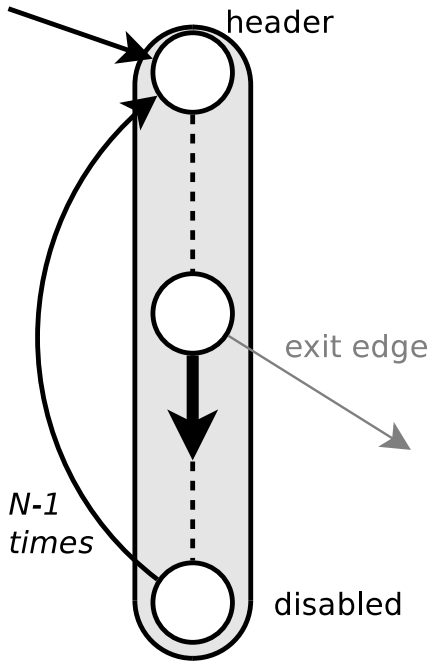
FCFG for loop with header *a*



FCFG for loop with header *b*

Loops

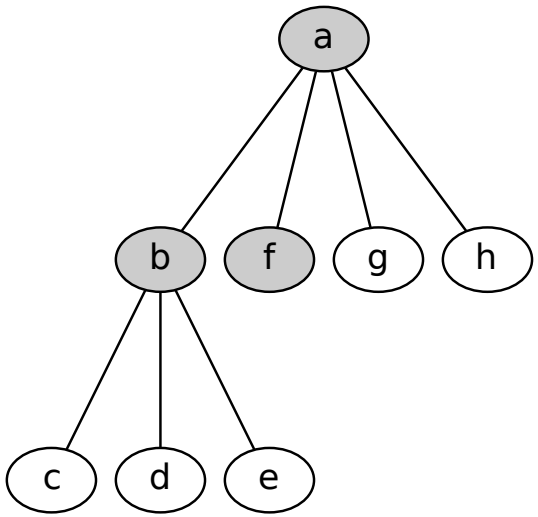
Input-data independent iteration count

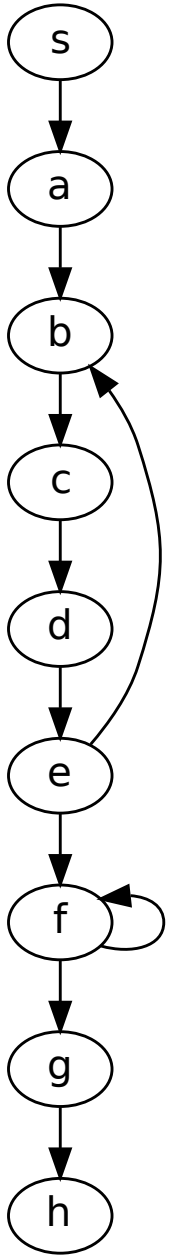


- Entering header initializes predicates
- Input-data independent branch condition
- Former exit edge causes clearing of header predicate
- From then on all nodes are disabled

Composition of the SP-Graph

- Preorder traversal of the loop header tree





Experiments

Simulation Framework

1. Experimental validation of technique
2. Evaluate estimated execution cost

Benchmark/Function	Max	SP	Ratio
adpcm/upzero	96	125	1.30
compress/compress	3589	4200	1.17
cover/swi120	1570	2655	1.69
expint/expint	1984	2736	1.38
fir/fir_filter_int	2618	3236	1.24
insertsort/main	750	832	1.11
janne_complex/complex	1135	1381	1.22
lcdnum/num_to_lcd	36	190	5.28
lms/main	103839	150209	1.45
ludcmp/ludcmp	328187	415544	1.27
minmax/main	70	83	1.19
minver/minver	197066	371563	1.89
qsort-exam/sort	7253	12004	1.66
qurt/qurt	1838	2208	1.20
select/select	7196	11219	1.56
statemate/FH_DU	305	360	1.18

Summary

- Single-path conversion as graph transformation
- Single-path graph with predicated execution model
- Validated technique
- Amenable for implementation in a compiler backend
- Automated generation of predictable machine code

http://github.com/t-crest/

References I

Park, J. C. and Schlansker, M. (1991).
On predicated execution.
Technical report, Hewlett Peckard Software and Systems Laboratory.

Puschner, P., Kirner, R., Huber, B., and Prokesch, D. (2012).
Compiling for time predictability.
In *Proc. SAFECOMP 2012 Workshops (LNCS 7613)*, pages 382–391. Springer.