

Accelerating WCET-driven Optimizations by the Invariant Path Paradigm – a Case Study of Loop Unswitching

Paul Lokuciejewski, Fatih Gedikli, Peter Marwedel

Computer Science 12

Dortmund University of Technology

D-44221 Dortmund, Germany

Outline

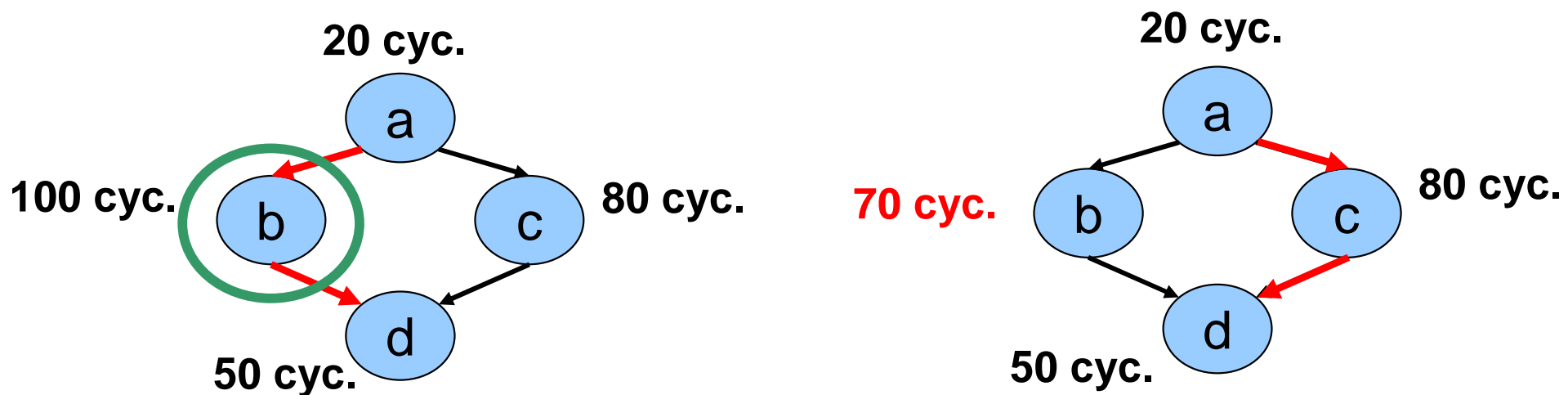
- **Introduction**
 - Motivation
 - Problems in WCET-driven Optimizations
- **Invariant Path Paradigm**
 - Idea, Concepts
- **WCET-driven Loop Unswitching**
 - Extension of standard optimization
- **Experimental Environment & Results**
- **Conclusions & Future Work**

Motivation

- **Embedded Systems used as Real-Time Systems**
- **Worst-case execution time (WCET) is a key parameter**
 - Crucial for safety-critical systems
 - Required for task scheduling
 - Helps to design hardware platform
- **Complex optimization problems solved by compilers**
 - WCET-driven optimizations rely on worst-case timing model
 - Data provided by static WCET analysis
 - **New challenges** imposed to WCET-aware compilers

Switching Worst-Case Execution Path (WCEP)

- Reduce WCET by optimizing the WCEP
- Path might change after code transformation



- Effective WCET reduction must be aware of path switch
- Common approach:** update by WCET analysis
- Optimization possibly infeasible due to long run times

Outline

- Introduction
- **Invariant Path Paradigm**
- WCET-driven Loop Unswitching
- Experimental Environment & Results
- Conclusions & Future Work

Accelerating WCET-driven Optimizations

Goal

- Speed-up WCET-driven optimizations by **reducing number of *updating WCET analysis***

Motivation

- Does **each code transformation** imply a potential path switch?
- **Branches introduce mutually exclusive CFG paths**
 - Represent potential candidates for WCEP switch
- **Not all branch paths prone to WCEP switch**

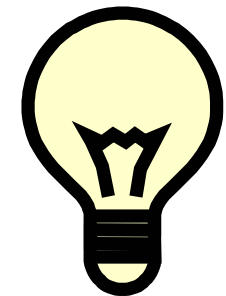
Invariant Path

Definition of Invariant Path Paradigm

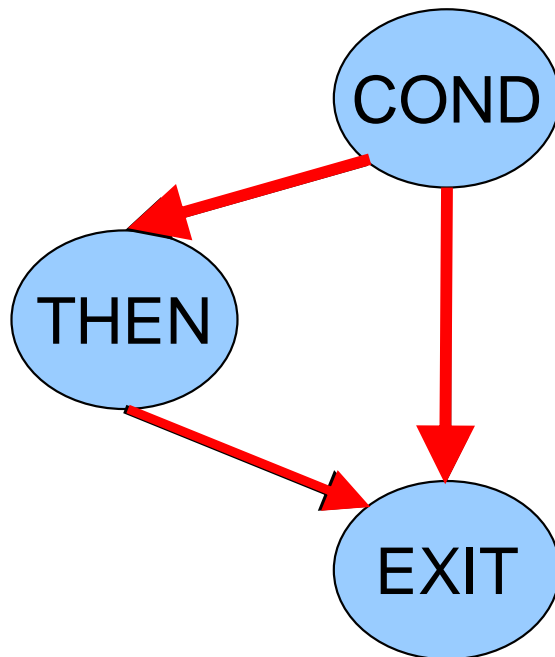
- Defines sub-paths of the WCEP which will always remain part of WCEP

Preliminaries

- Invariant Path considered for branches
 - Relevant pseudo-statements: *if*, *if-else*, (*switch*)
 - Code on WCEP and not on *branched* code is Invariant Path
- Safe and precise static WCET analysis
 - Longer path assumed when branch not statically evaluable
 - Context-sensitivity assumed



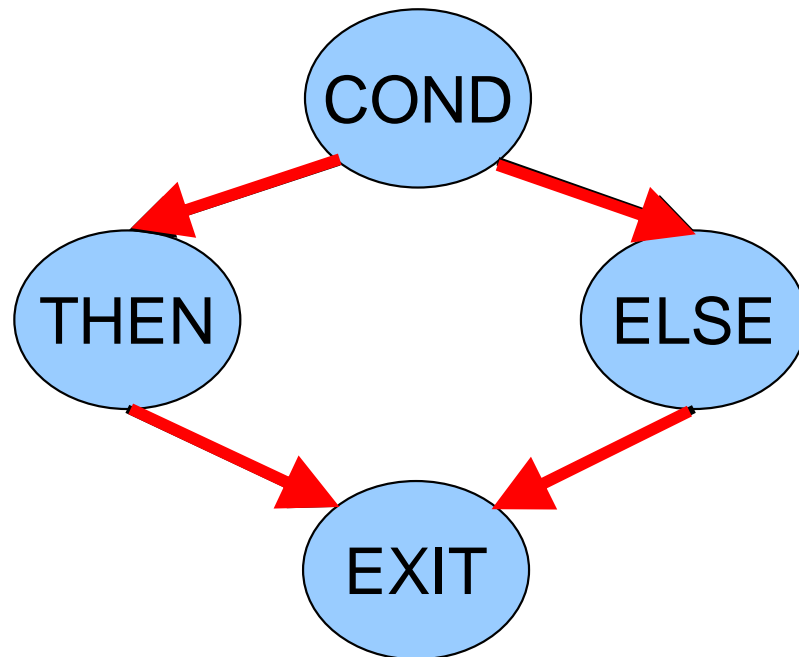
IF with feasible WCEPs



1. **COND** could not be statically evaluated
2. **COND** could be statically evaluated
3. Both paths taken in different calling contexts

- **No crucial WCEP switches due to code transformations**
- **IF classified as Invariant Path**

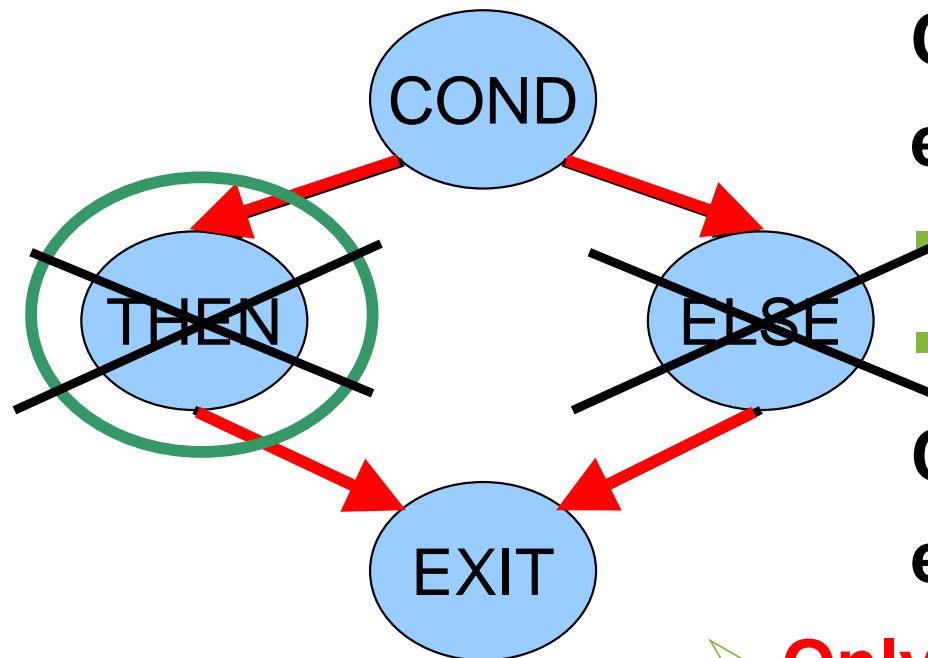
IF-ELSE with feasible WCEPs – Common Case



Both paths taken in different calling contexts

- **WCEP switch not possible**
- **IF-ELSE classified as Invariant Path**

IF-ELSE with infeasible WCEP



COND could be statically evaluated:

- **One path is WCEP**
- **Other path is dead code**

COND could not be statically evaluated

➤ **Only case prone to WCEP switch**

- **Due to dead code, WCEP switch not possible**
- **IF-ELSE classified as Invariant Path**

Exploiting Invariant Path for Optimizations

First Step: Recursive Construction of Invariant Path

- **Start at entry point of CFG**
- **Traverse CFG in top-down manner**
- **Nested branches separately analyzed**

Second Step: Make Use of Invariant Path

- **If code transformed by WCET-driven optimization is part of Invariant Path**
 - Continue with next optimization step without redundant update of WCET data
 - **Significant optimization time reduction**

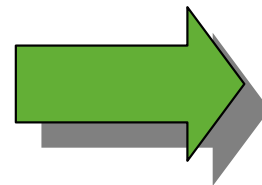
Outline

- Introduction
- Invariant Path Paradigm
- **WCET-driven Loop Unswitching**
- Experimental Environment & Results
- Conclusions & Future Work

Standard Loop Unswitching

- ACET optimization with trade-off between speed and size
- Shifts loop-invariant conditions out of loop at cost of loop body duplication

```
for(i=0; i<100; i++) {
  x[i] = x[i] + y[i];
  if( w )
    y[i] = y[i] * 2;
  else
    y[i] = 1;
}
```



```
if( w )
  for(i=0; i<100; i++) {
    x[i] = x[i] + y[i];
    y[i] = y[i] * 2; }
else
  for(i=0; i<100; i++) {
    x[i] = x[i] + y[i];
    y[i] = 1; }
```

WCET-driven Loop Unswitching

- **Code expansion crucial for embedded systems**
- **Effective unswitching relies on suitable opt. order**
 - Most frequently executed loop-invariant conditions
- **Many compilers lack execution counts**

Algorithm

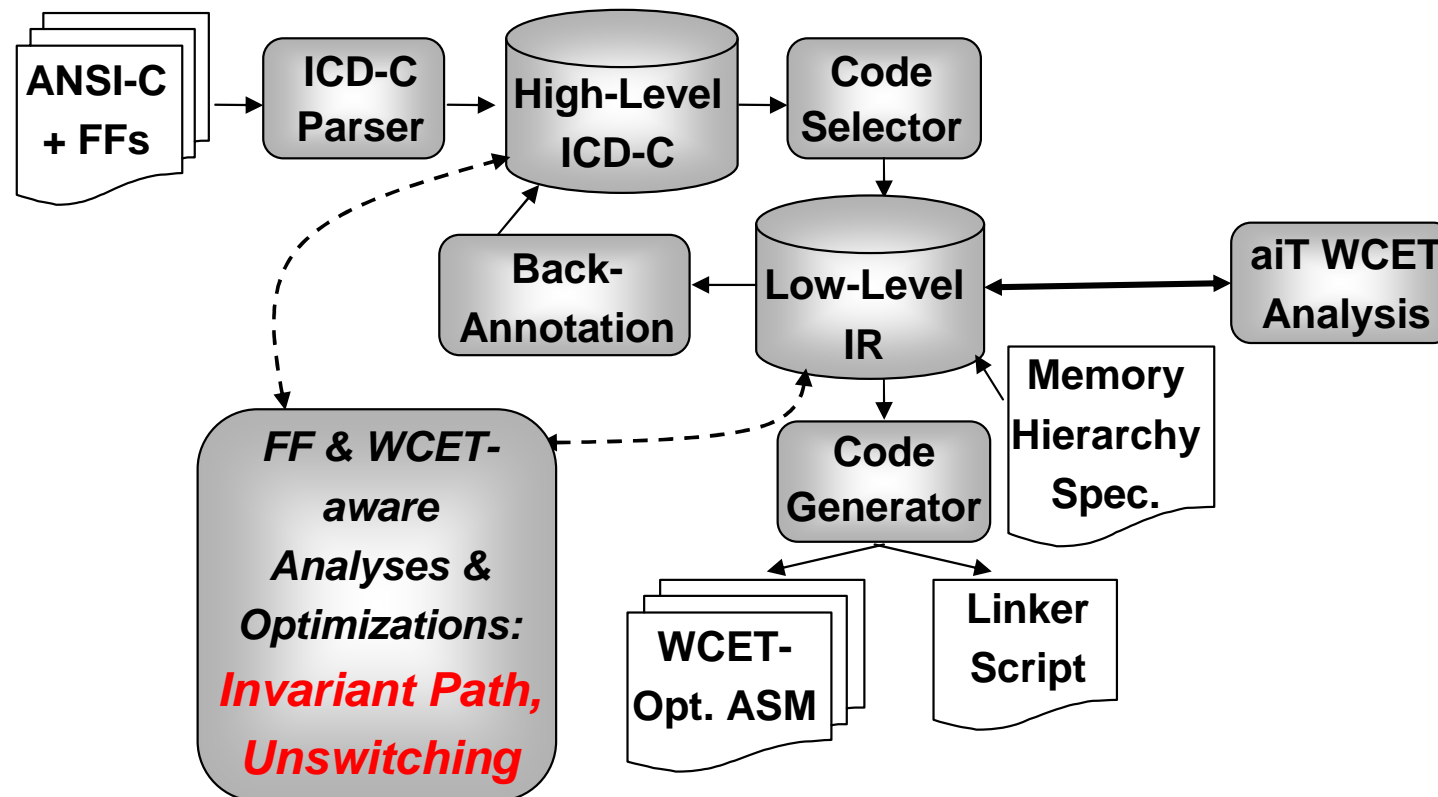
- **WCET-driven Loop Unswitching aims at WCET reduction**
- **Heuristics based on worst-case execution counts**
 - Choose best candidate based on profit calculation
- **Exploit Invariant Path information**
 - Only unswitching on non-IP require updating WCET analysis

Outline

- Introduction
- Invariant Path Paradigm
- WCET-driven Loop Unswitching
- **Experimental Environment & Results**
- Conclusions & Future Work

Experimental Environment

- **WCC** * – WCET-aware C compiler for Infineon TC1796



[*<http://ls12-www.cs.tu-dortmund.de/research/activities/wcc/>]

Results

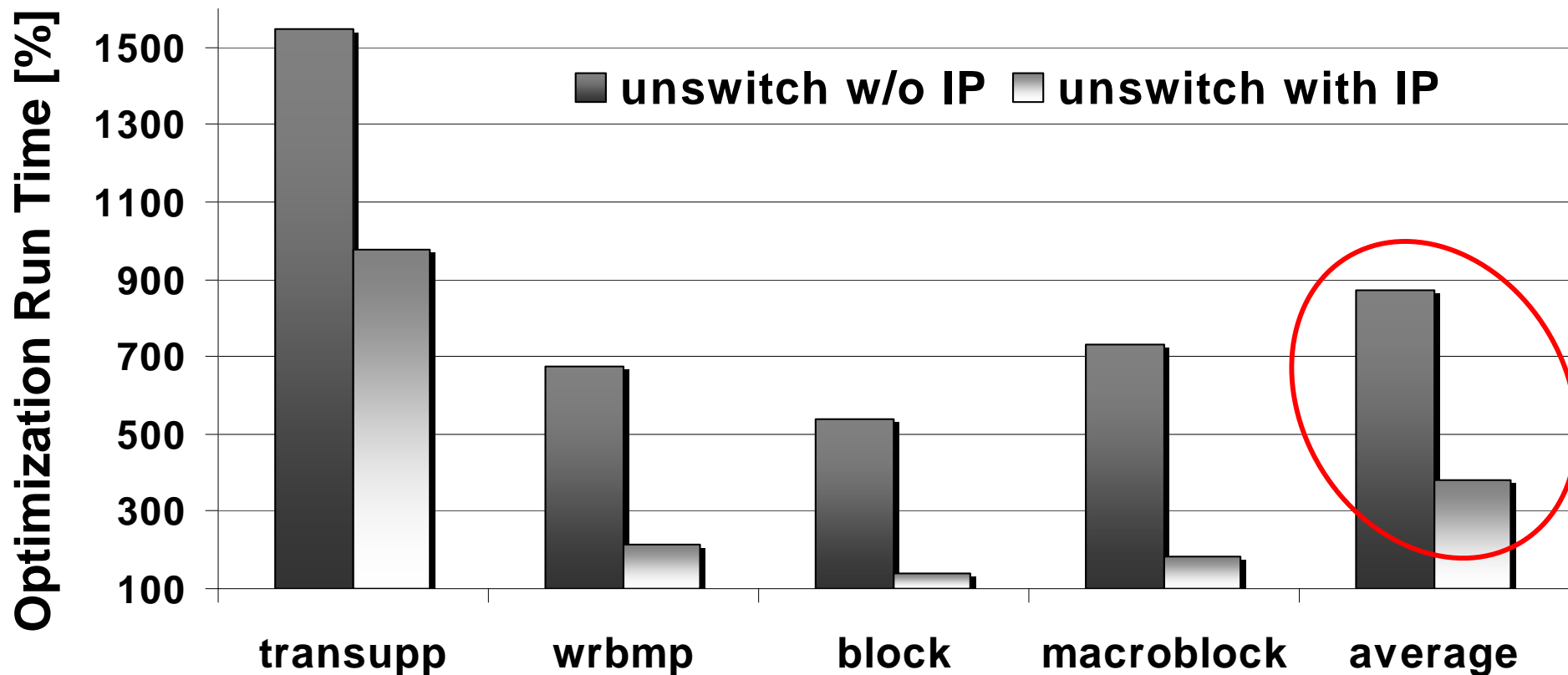
Invariant Path Ratio for 45 benchmarks (for [-O0 .. -O3])

- **Static ratio** – number of basic blocks on Invariant Path
 - between 74.1% and 77.9%
- **Dynamic ratio** – number of cycles on Invariant Path
 - between 85.4% and 88.8%
- **Large amount of code not crucial to WCEP switches**

Benchmark Characteristics

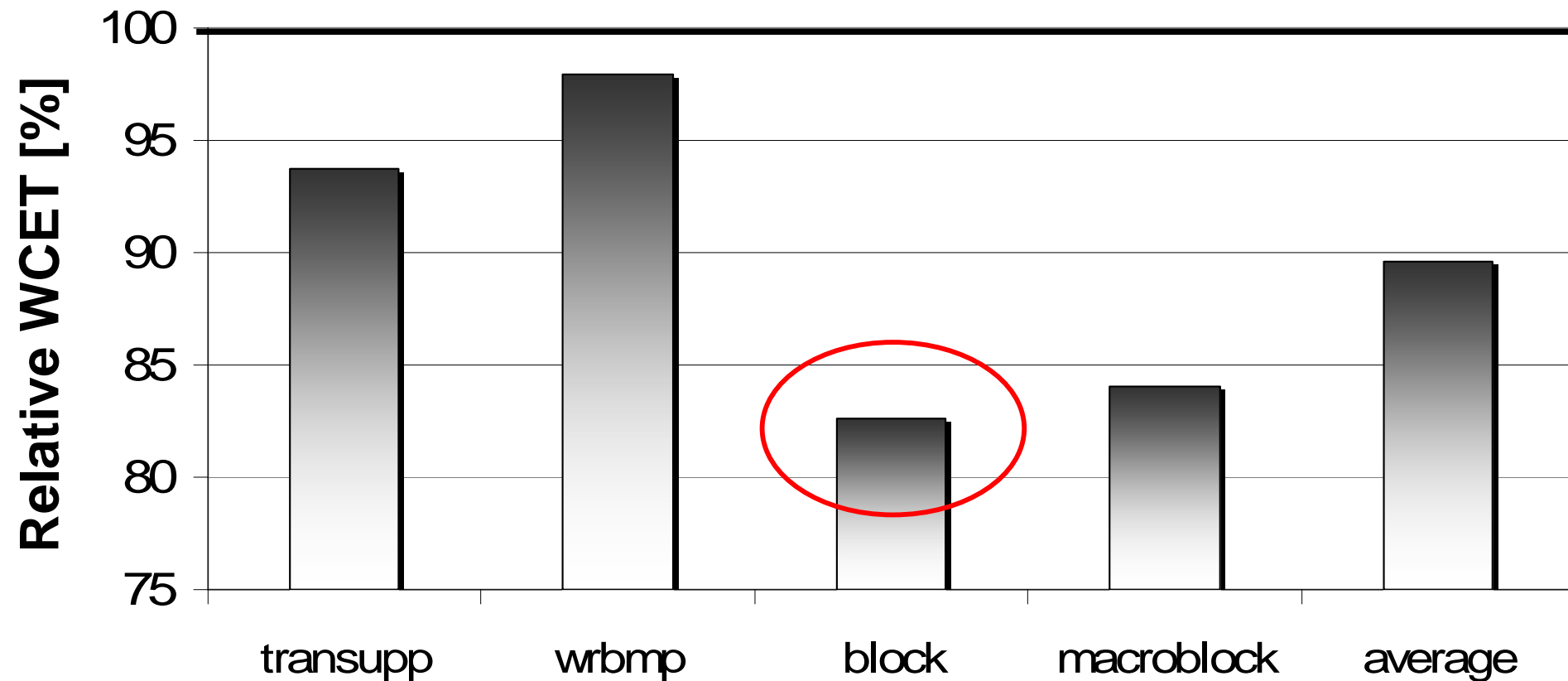
- 4 kernels from MediaBench for JPEG/H.264 compression
- Average code size: 10.6 kByte
- Unswitching candidates: between 4 and 13

Results – Optimization Run Time



- **100%: optimization time of standard ACET unswitching**
- **Avg. time for WCET Unswitching w/o Invariant Path: 872%**
- **Exploiting Invariant Path reduces opt. time by 57.5%**

Results - WCET



- **100%: WCET of -O0 with dead code elimination**
- **Maximal WCET reduction of 18.3%**

Outline

- Introduction
- Invariant Path Paradigm
- WCET-driven Loop Unswitching
- Experimental Environment & Results
- **Conclusions & Future Work**

Conclusions & Future Work

- **WCET-driven optimizations prone to WCEP switches**
 - Update of WCET data by time-consuming WCET analysis
- **Invariant Path Paradigm** presented to avoid redundant updates
- **Development of WCET-driven Loop Unswitching**
 - Exploits Invariant Path → 57.5% opt. time reduction
 - Maximal WCET reduction of 18.3%

Future Work

- **Translate Invariant Path to low-level code**
- **Accelerate further optimizations by new paradigm**

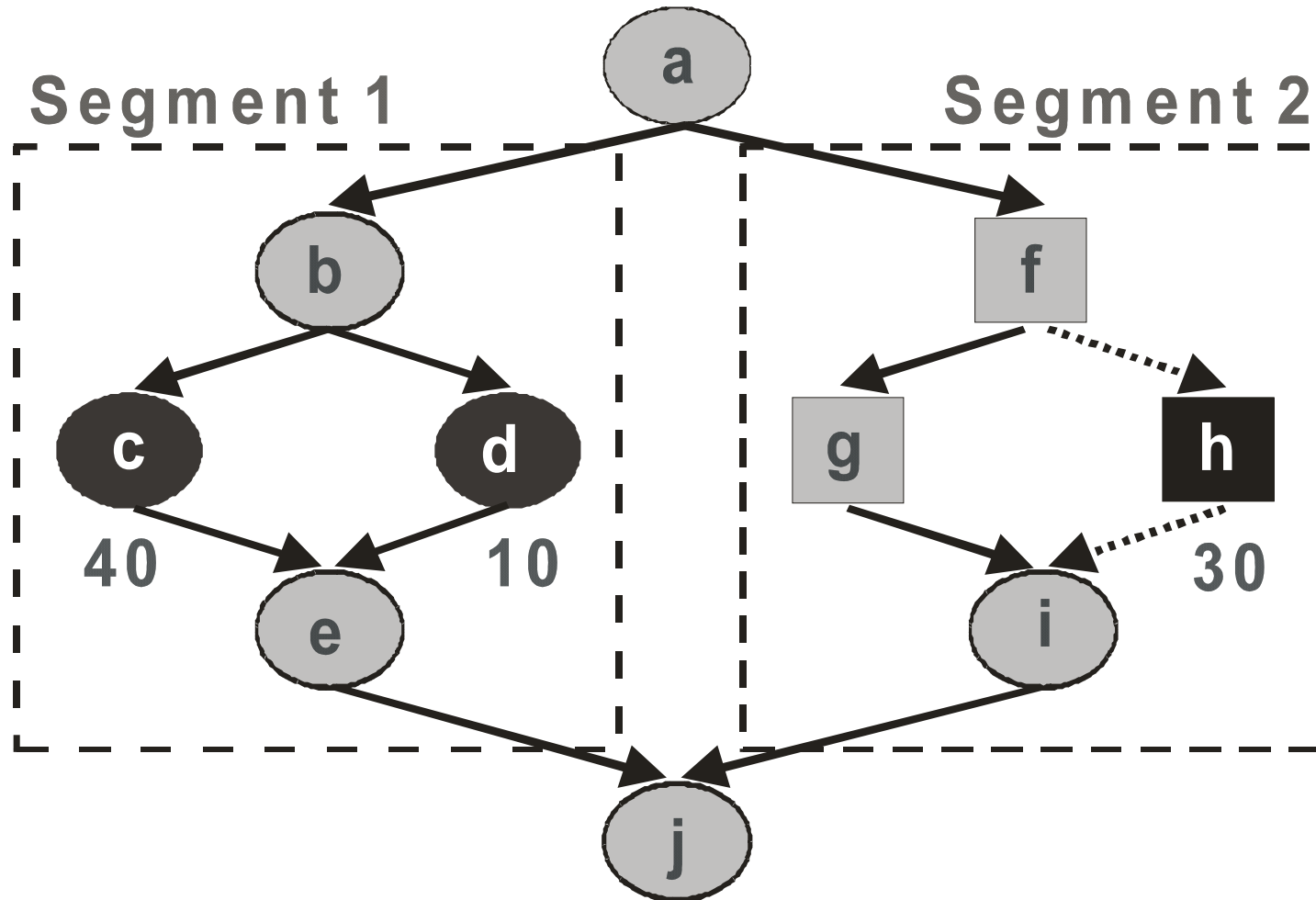
**Thank you for your
attention.**

PREDATOR 

Loop Unswitching Algorithm

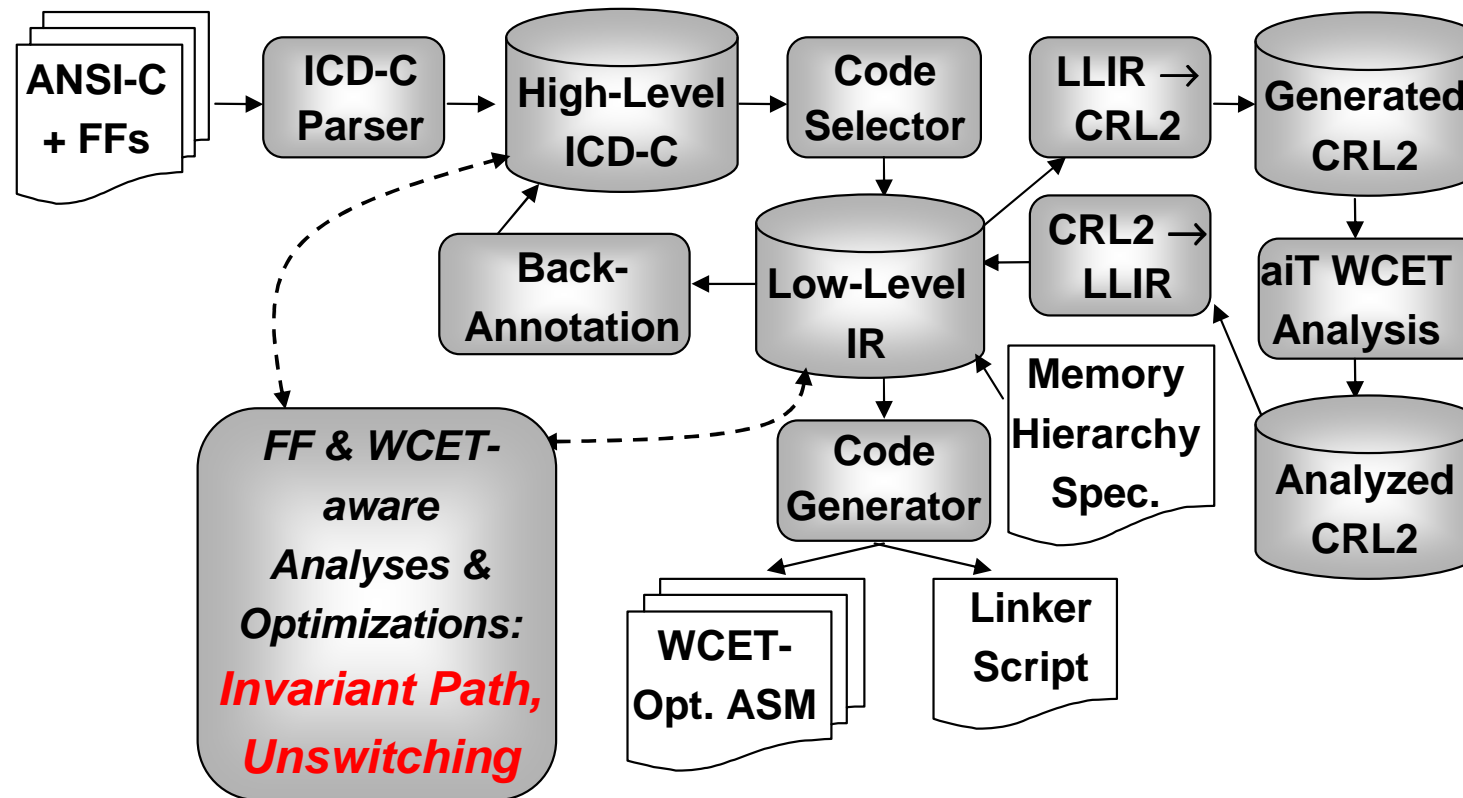
```
4 begin
5 performWCETAnalysis(P)
6 set<Loop> S := FindUnswitchCand(P)
7 while(!S.empty()) do
8   boolean allOnIP := CheckInvariance(S)
9   repeat
10     Loop bestCand := FindBest(S)
11     LoopUnswitching(bestCand)
12     DeleteCandidate(S,bestCand)
13     if(CodeSizeIncrease(P) > MAX)
14       return P
15     fi
16   until(!S.empty() && allOnIP == true)
17   performWCETAnalysis(P)
18   S := FindUnswitchCand(P)
19 od
```

WCET Switch in Different Segments



Experimental Environment

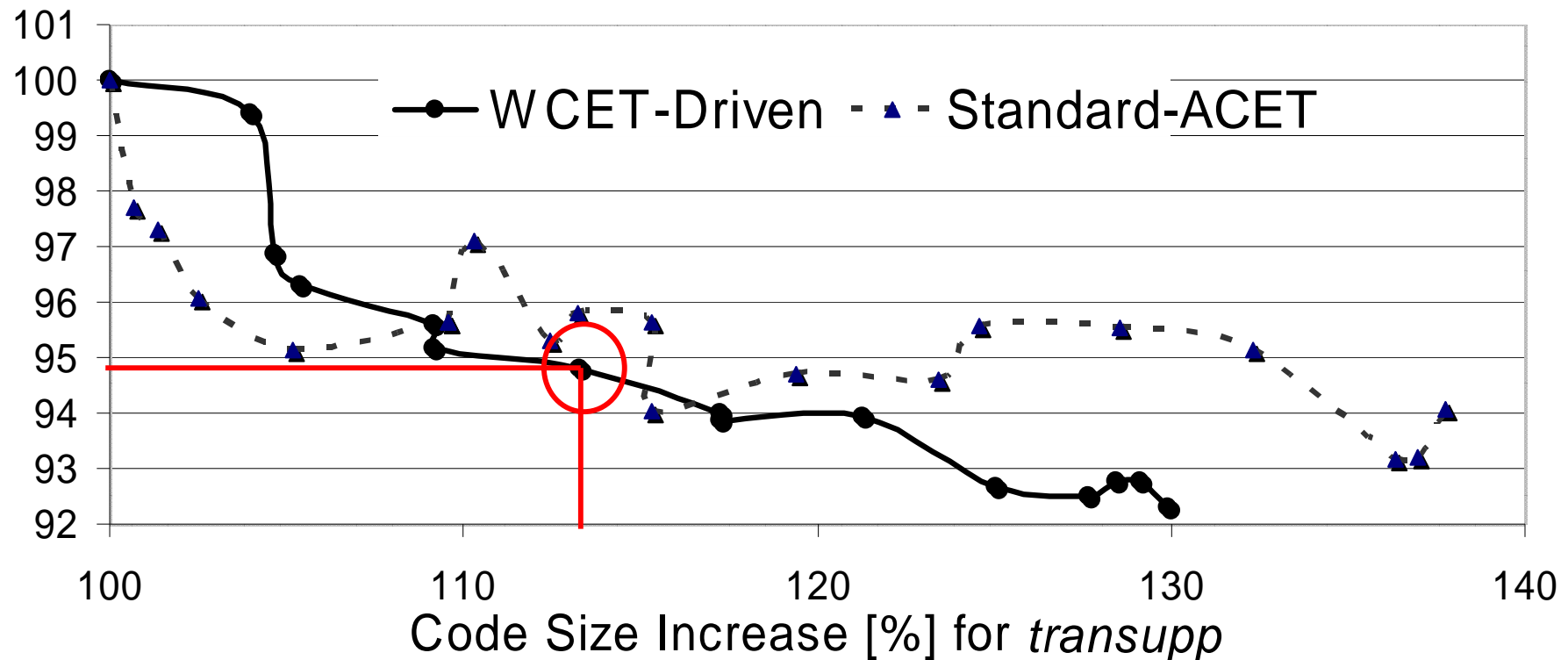
- **WCC** * – WCET-aware C compiler for Infineon TC1796



[*<http://ls12-www.cs.tu-dortmund.de/research/activities/wcc/>]

Results – Code Size

- 19.7% code size increase on average



- Optimization tailored towards effective WCET minimization