Refinement of Worst-Case Execution Time Bounds by Graph Pruning

Florian Brandner
Unité d'Informatique et d'Ing. des Systèmes
ENSTA-ParisTech

Alexander Jordan
Embedded Systems Engineering Sect.
Technical University of Denmark

This work is partially supported by the EC project T-CREST.

Real-Time Systems

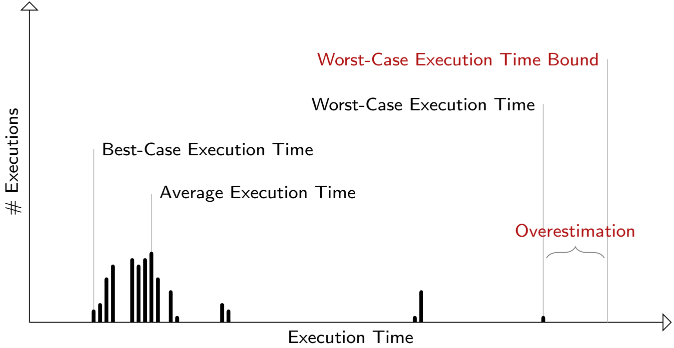
Strict timing **guarantees**

- Critical tasks have to be completed in time

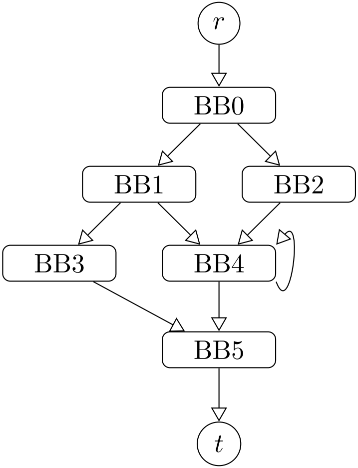
Real-Time Systems

Strict timing **guarantees**

- Critical tasks have to be completed in time
- Bound *Worst-Case Execution Time* (WCET)

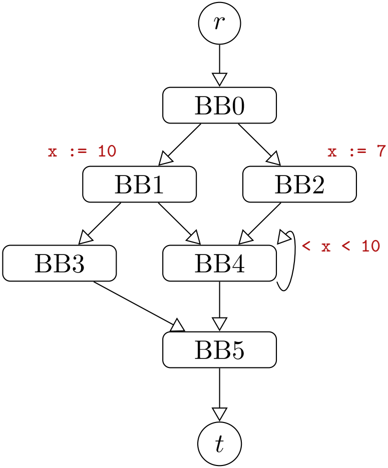


WCET Analysis (1)



Three analysis phases:

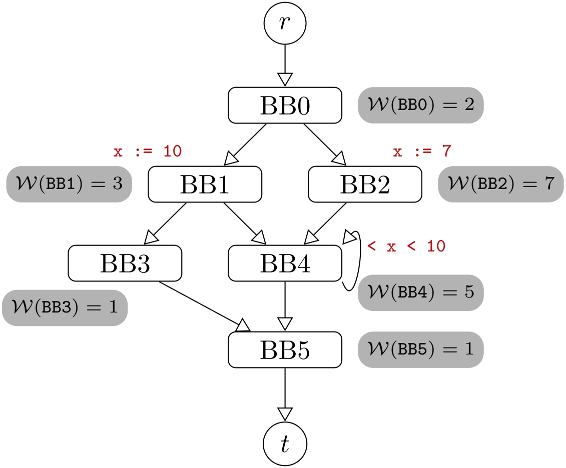
WCET Analysis (1)



Three analysis phases:

(1) Loop bounds & flow facts

WCET Analysis (1)

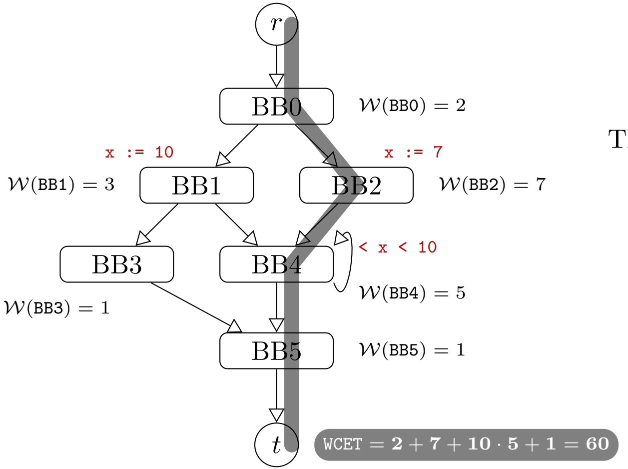


Three analysis phases:

(1) Loop bounds & flow facts

(2) Pipeline & caches

WCET Analysis (1)



Three analysis phases:

(1) Loop bounds & flow facts

(2) Pipeline & caches

(3) Longest path search (IPET)

WCET Analysis (2)

Bound longest possible execution time of a program

- Covering all potential execution paths
- Covering all potential program inputs
- Covering all potential hardware states

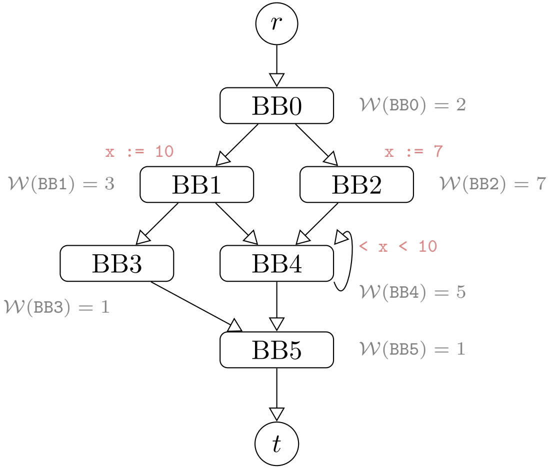
WCET Analysis (2)

Bound longest possible execution time of a program

- Covering all potential execution paths
- Covering all potential program inputs
- Covering all potential hardware states

A priori all executions are equally considered relevant

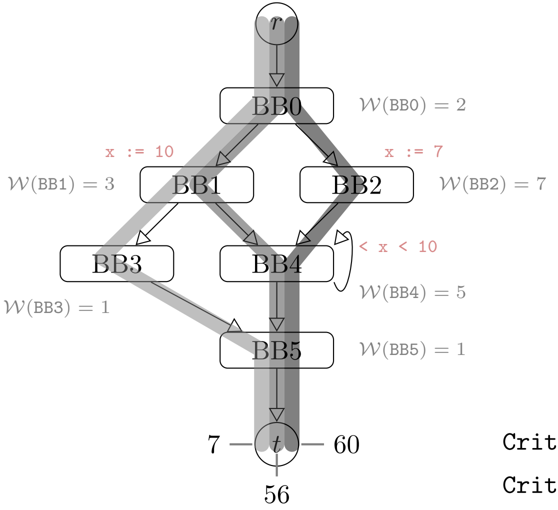
Criticalities



Criticality:

- $\texttt{WCET}(BBn)$: Longest path over BBn.
- $\texttt{WCET}$: Longest path in the graph (from r to t)
- $\texttt{Crit}(BBn) = \frac{\texttt{WCET}(BBn)}{\texttt{WCET}}$

Criticalities



Criticality:

- $\texttt{WCET}(BBn)$: Longest path <u>over</u> BBn.
- $\texttt{WCET}$: Longest path in the graph (from r to t)
- $\texttt{Crit}(BBn) = \frac{\texttt{WCET}(BBn)}{\texttt{WCET}}$

$$\texttt{Crit}(BB3) = \frac{7}{60} = 0.12$$

$$\texttt{Crit}(BB1) = \frac{56}{60} = 0.93$$

Criticality Distribution: Debie1

Problem	BBs	I_0	I_1	I_2	I_3	I_4	I_5
debie1	83	4	2	0	13	19	45
debie3a	74	16	0	0	0	1	57
debie3b	74	15	0	0	0	0	59
debie3c	74	15	0	0	0	0	59
debie4a	285	31	192	0	19	3	40
debie4b	285	236	3	14	0	3	29
debie4c	285	260	0	4	0	5	16
debie4d	285	264	0	4	0	1	16
debie5a	138	13	0	0	1	4	120
debie5b	138	5	0	0	0	1	132
debie6a	376	53	24	2	105	0	192
debie6b	376	52	22	4	106	0	192
debie6c	376	52	22	143	4	0	155
debie6d	376	12	24	2	0	144	194

*Intervals: $0 \leq I_0 < 0.25 < I_1 < 0.5 < I_2 < 0.75 < I_3 < 0.9 < I_4 < 0.99 < I_5 \leq 1$

Iterative Graph Pruning

Improving WCET bounds

- Many basic blocks turn out to be *uncritical*

Iterative Graph Pruning

Improving WCET bounds

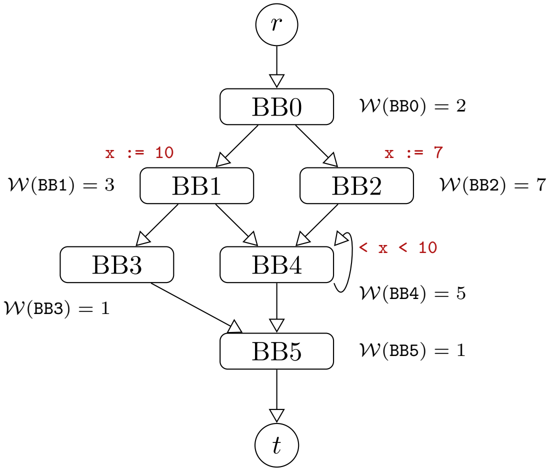
- Many basic blocks turn out to be *uncritical*
- Why do we then analyze them?

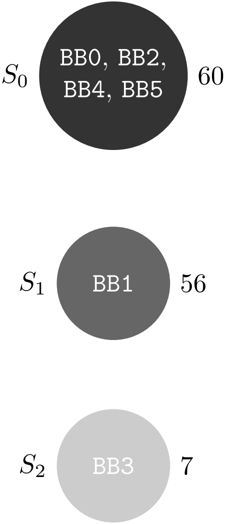
Iterative Graph Pruning

Improving WCET bounds

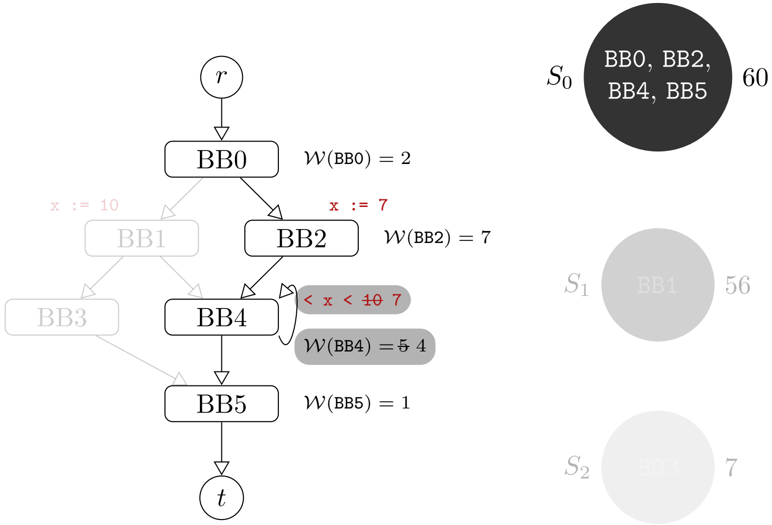
- Many basic blocks turn out to be *uncritical*
- Why do we then analyze them?
- Can we remove uncritical blocks?
  - Focus on relevant code only
  - More precise WCET
  - Faster analysis

Iterative Graph Pruning: Example

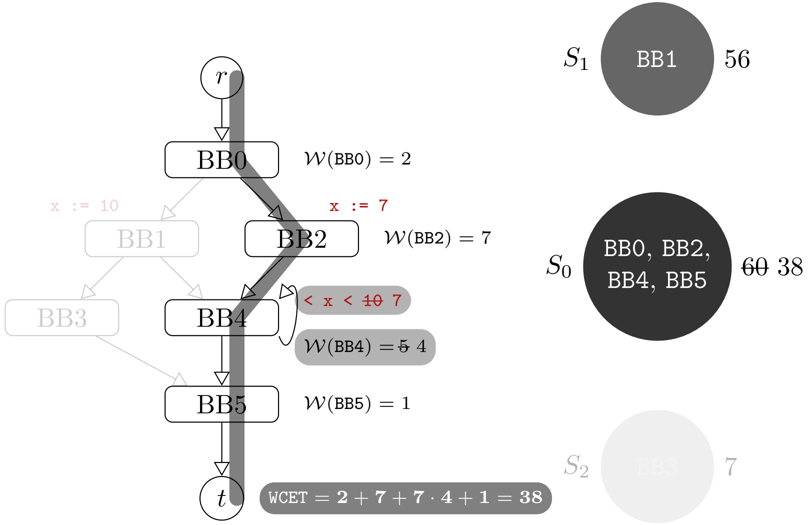




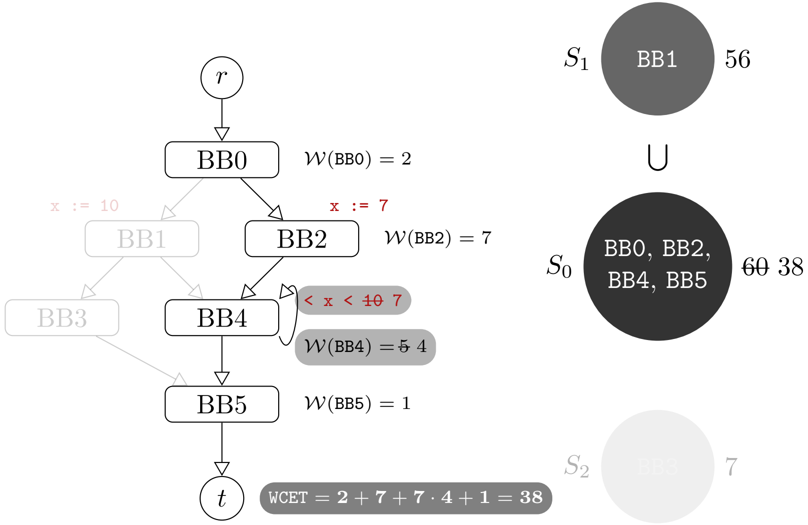
Iterative Graph Pruning: Example



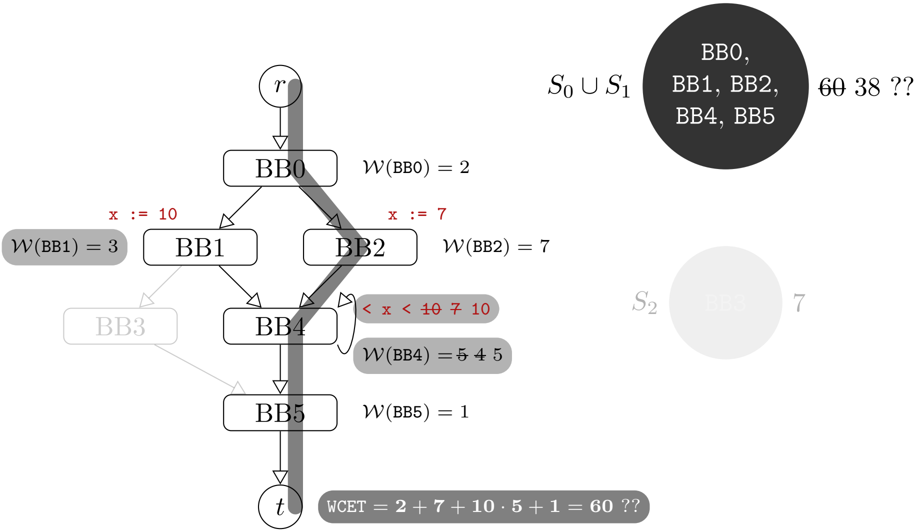
Iterative Graph Pruning: Example



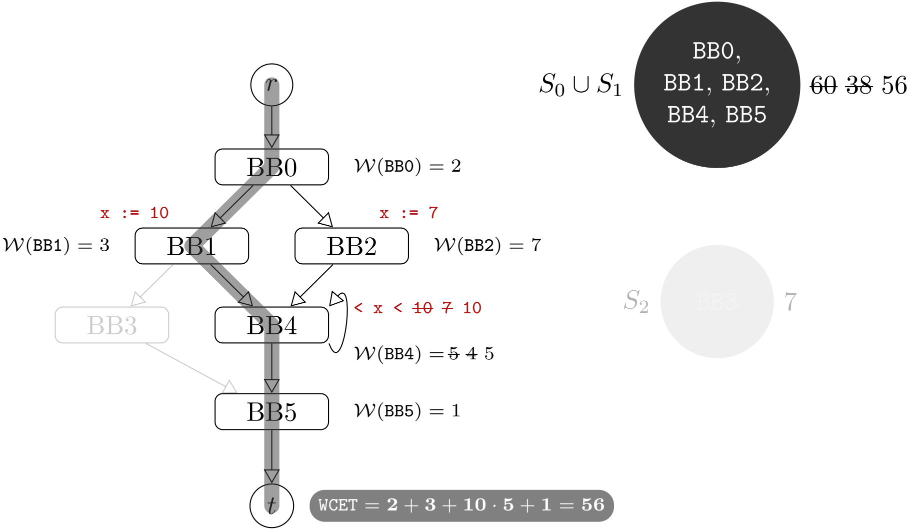
Iterative Graph Pruning: Example



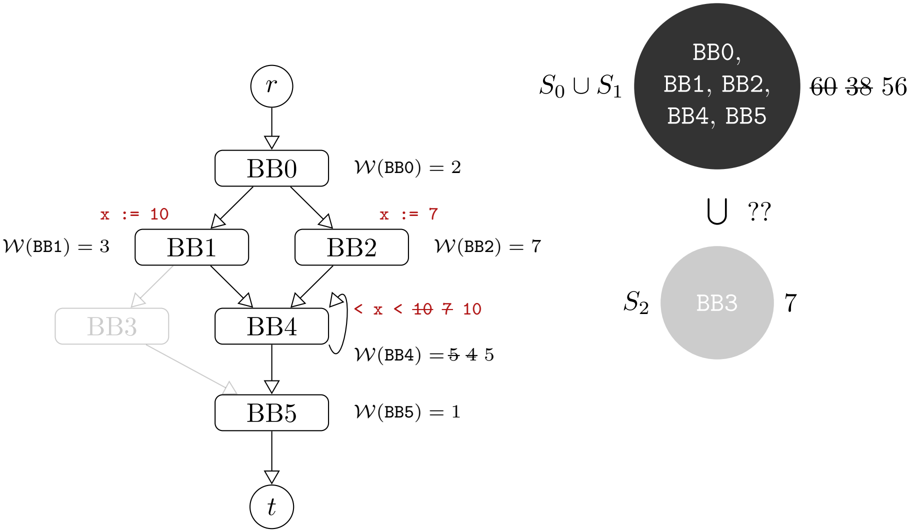
Iterative Graph Pruning: Example



Iterative Graph Pruning: Example



Iterative Graph Pruning: Example



Iterative Graph Pruning: Algorithm

Input: $G = (V, E)$ The program's control-flow graph
$S_0, \ldots, S_n$ Block sets sorted by path length

$ub_{\mathtt{WCET}}$ `:= 0`
`for` i `= 1 to` n
    `if` $ub_{\mathtt{WCET}}$ `>= pathlength(`S_i`)`
        `return` $ub_{\mathtt{WCET}}$
    `let` $V' = S_0 \cup \ldots \cup S_i$, $G' = (V', E \cap V' \times V')$ `in`
        $ub_{\mathtt{WCET}}$ `:= max(`$ub_{\mathtt{WCET}}$`, WCEToverAny(`G'`,` S_i`))`
`return` $ub_{\mathtt{WCET}}$

Fast vs. Precise WCET Analysis

Two-Stage WCET analysis

- Combine fast with precise analysis
- Fast analysis
  - Compute block sets
  - Check WCET increase while iterating
- Precise analysis to verify

Fast vs. Precise WCET Analysis

Two-Stage WCET analysis

- Combine fast with precise analysis
- Fast analysis
  - Compute block sets
  - Check WCET increase while iterating
- Precise analysis to verify

Non-Iterative Pruning

- Heuristically construct a pruned graph
  - Using Criticality?
  - Using Criticality estimates?
- Apply precise analysis to pruned graph

Experiments

Setup

- Commercial WCET analysis tool[a] (AbsInt aiT, 12.10)
- Freescale mpc5554 and mpc755s (PowerPC)

Experiments

Setup

- Commercial WCET analysis tool[a] (AbsInt aiT, 12.10)
- Freescale mpc5554 and mpc755s (PowerPC)

- Two real-time benchmarks
  - Debie1: satellite instrument control
  - Papabench: flight control

Experiments

Setup

- Commercial WCET analysis tool[a] (AbsInt aiT, 12.10)
- Freescale mpc5554 and mpc755s (PowerPC)

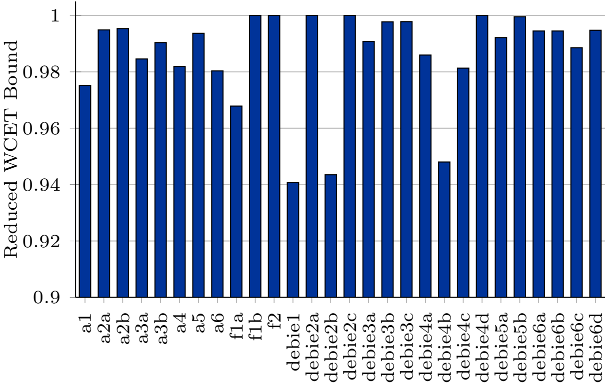
- Two real-time benchmarks
  - Debie1: satellite instrument control
  - Papabench: flight control

- 28 analysis problems[b]

---

[a] http://www.absint.com/ait/
[b] http://www.mrtc.mdh.se/projects/WCC/2011/

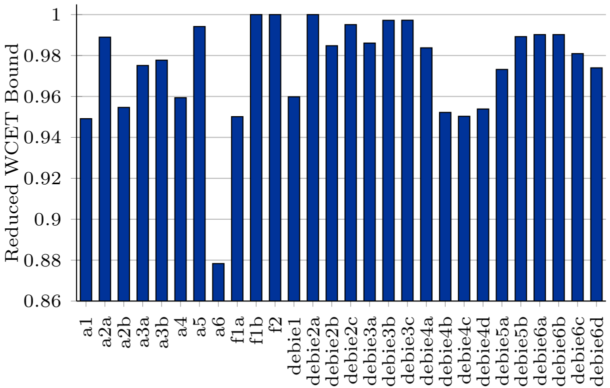
WCET Reductions (mpc5554)



WCET Reductions up to 6%.

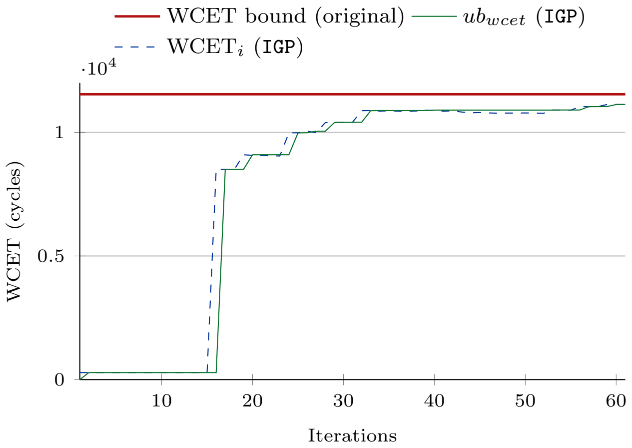
aiT is usually already close to measured bounds.

WCET Reductions (mpc755s)

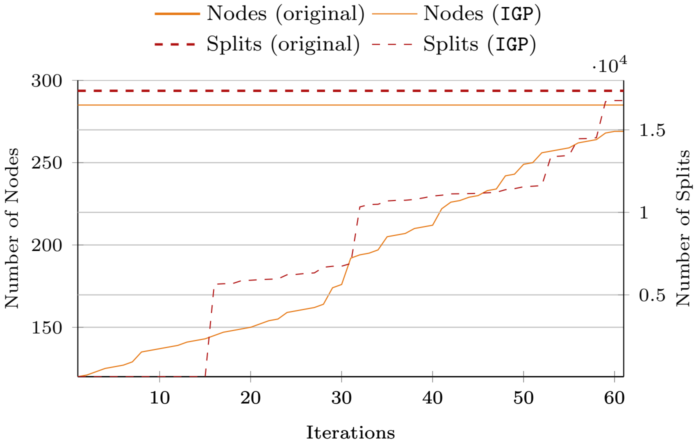


WCET Reductions up to 12%.

Iterations of f1a: WCET (mpc5554)



Iterations of f1a: Problem Size (mpc5554)



Conclusion

- Criticality
  - Novel compiler-centric metric
  - Proved interesting for WCET analysis
  - Cheap yet accurate estimation

Conclusion

- Criticality
  - Novel compiler-centric metric
  - Proved interesting for WCET analysis
  - Cheap yet accurate estimation
- Iterative Graph Pruning
  - Based on Criticality
  - Allows elimination of uncritical code
  - Successfully reduces overestimation
  - Causes quite some overhead (9x on average)
    - Proof-of-Concept implementation
    - Treats WCET analysis as black box
    - Incremental analysis techniques needed