Timing Analysis for Hard Real-Time Systems

Reinhard Wilhelm
Saarland University
Saarbrücken

ARTIST2 Cluster on
Compilation and Timing Analysis

Structure of the Talk

1. Timing Analysis, the problem,
2. architecture, static program analysis
3. Industrial experience
4. Future Work in ARTIST2

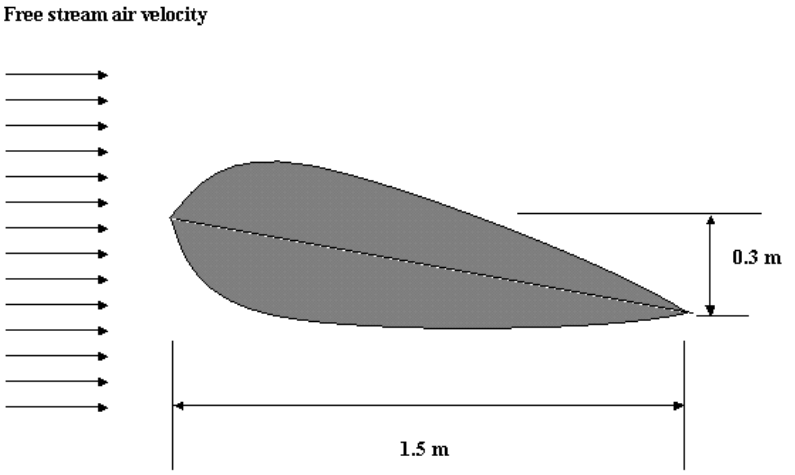
Industrial Needs

Hard real-time systems, often in safety-critical applications abound

- Aeronautics, automotive, train industries, manufacturing control

Sideairbag in car,
Reaction in <10 mSec

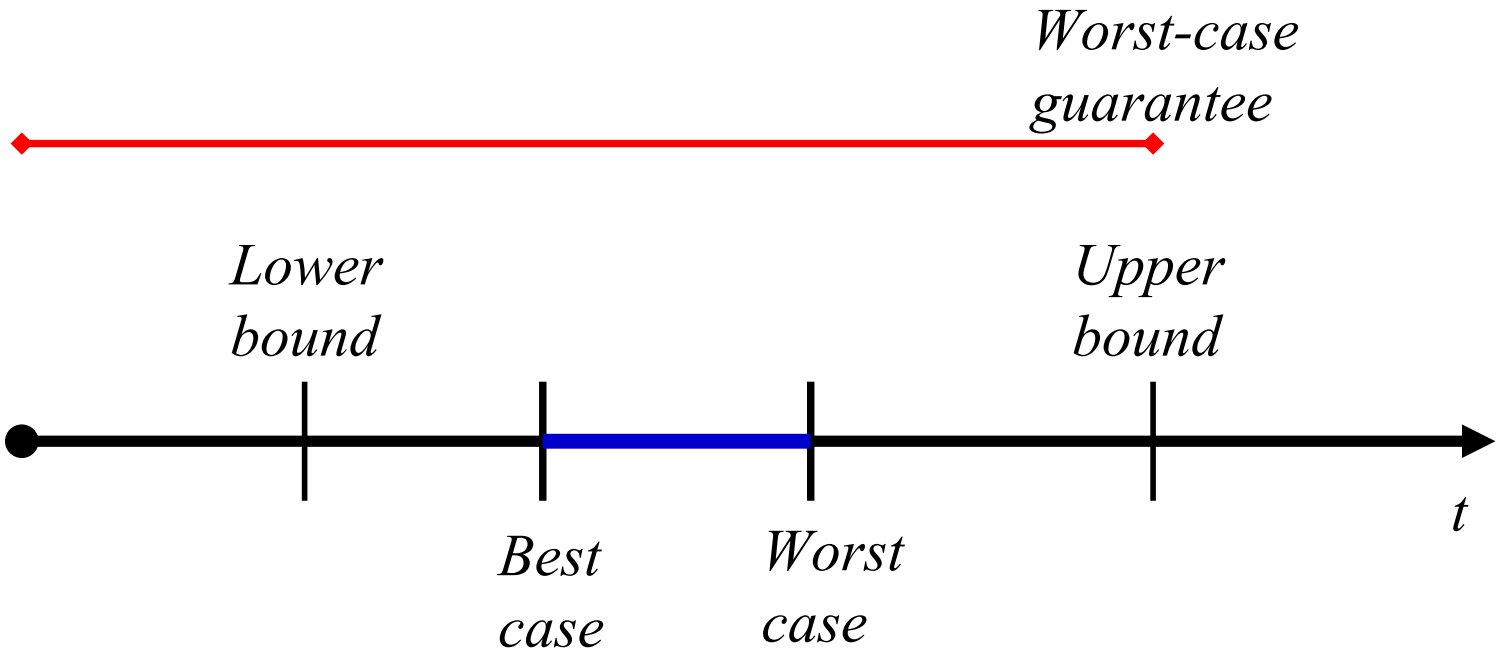
Wing vibration of airplane,
sensing every 5 mSec



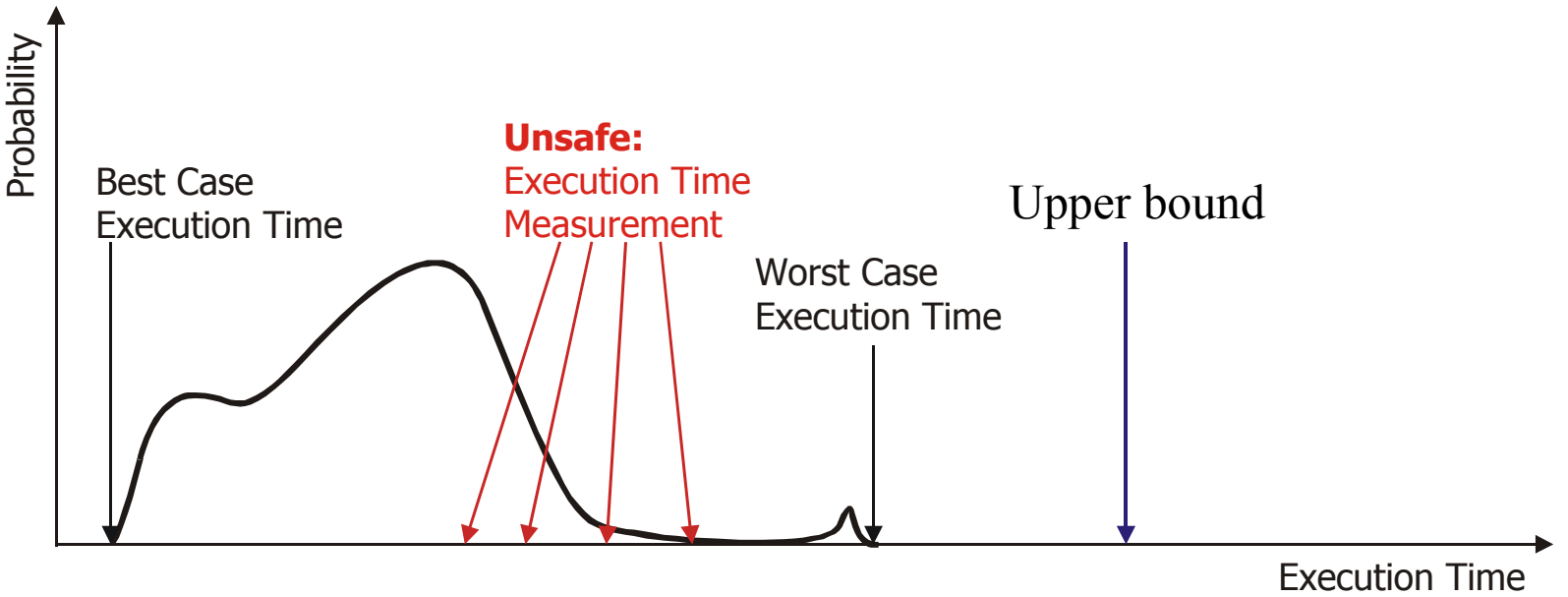
Hard Real-Time Systems

- Embedded controllers are expected to finish their tasks reliably within time bounds.
- Task scheduling must be performed
- Essential: upper bound on the execution times of all tasks statically known
- Commonly called the Worst-Case Execution Time (WCET)
- Analogously, Best-Case Execution Time (BCET)

Basic Notions



Measurement vs. Analysis



Once upon a time,

the world was compositional

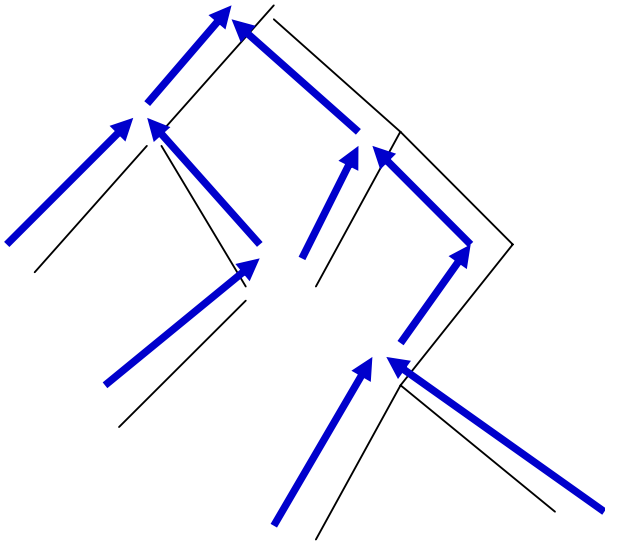
u_bound(**if** *c* **then** s_1 **else** s_2) =
u_bound(*c*) +max{u_bound(s_1), u_bound(s_2)}

u_bound(x:=y+z) =
time(**mv** y R1) +
time(**mv** z R2) +
time(**add** R1 R2) +
time(**mv** R1 x)

add	4
mv m Reg	12
mv Reg m	14
mv Reg Reg	1

Structure-based Approaches

- Historically first approaches to Timing Analysis
- Based on the structure of the program
- Easy to implement ☺
- Need compositionality ☹
- Do not deliver precision on modern processors ☹



Modern Hardware Features

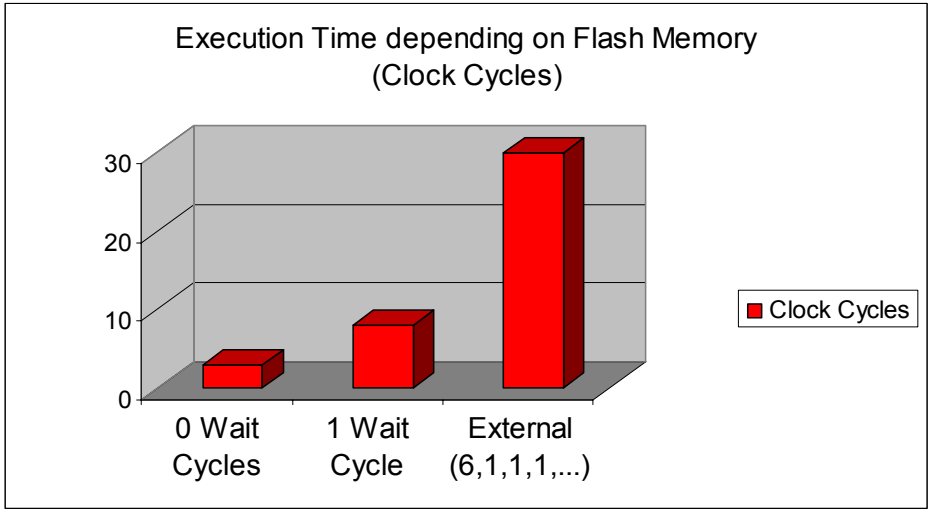
- Modern processors increase performance by using:
  Caches, Pipelines, Branch Prediction
- These features make WCET computation difficult:
  Execution times of instructions vary widely
  - Best case - everything goes smoothely: no cache miss, operands ready, needed resources free, branch correctly predicted
  - Worst case - everything goes wrong: all loads miss the cache, resources needed are occupied, operands are not ready
  - Span may be several hundred cycles

Access Times

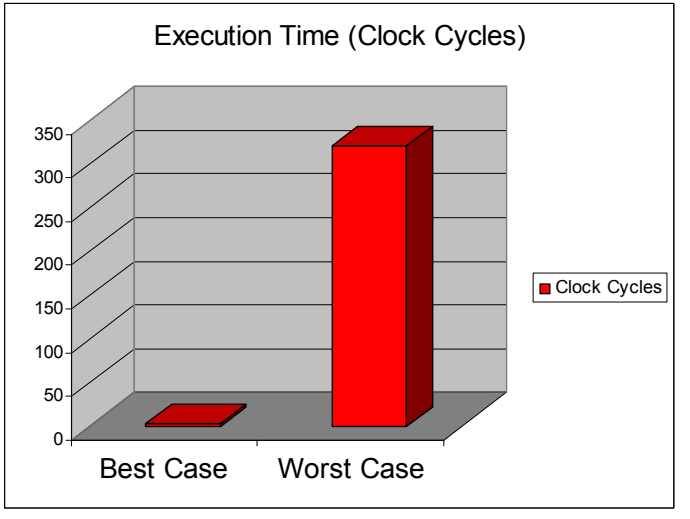
x = a + b;

```
LOAD    r2, _a
LOAD    r1, _b
ADD     r3,r2,r1
```

MPC 5xx



PPC 755



(Concrete) Instruction Execution

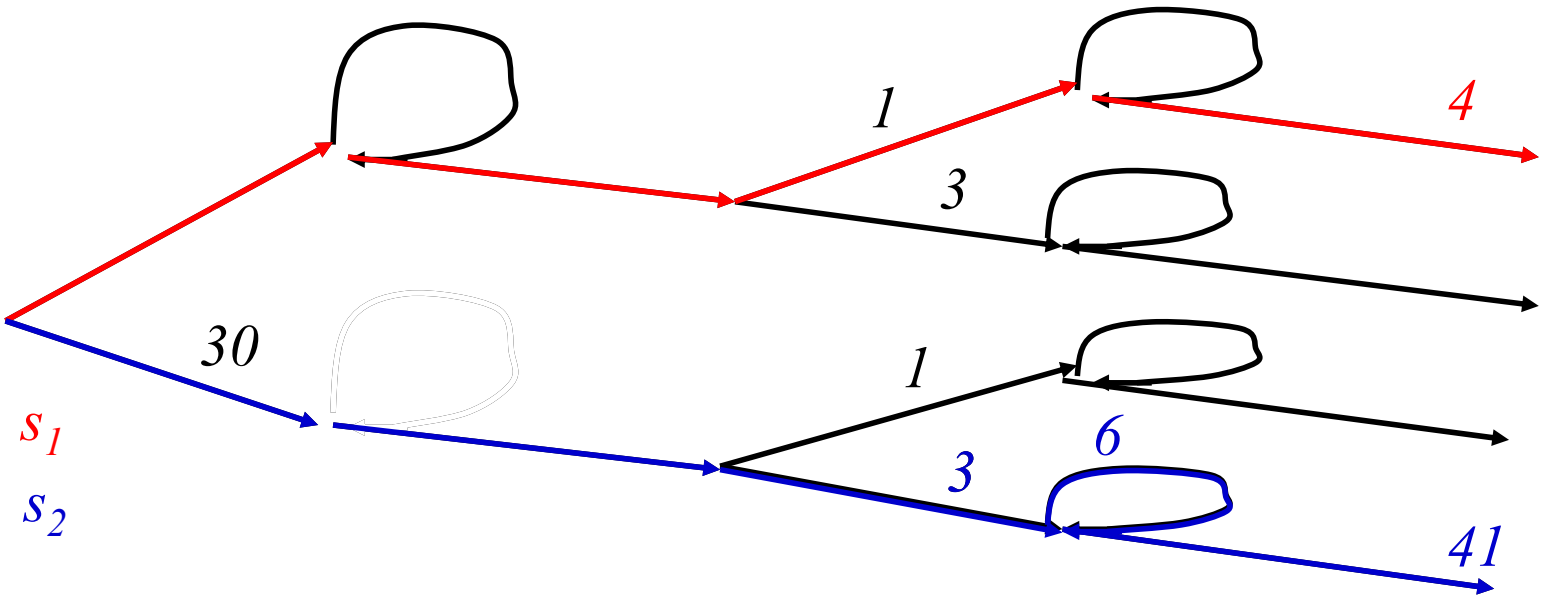
`mul`

Fetch
I-Cache miss?

Issue
Unit occupied?

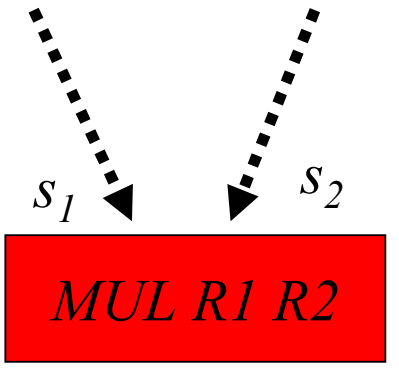
Execute
Multicycle?

Retire
Pending instructions?



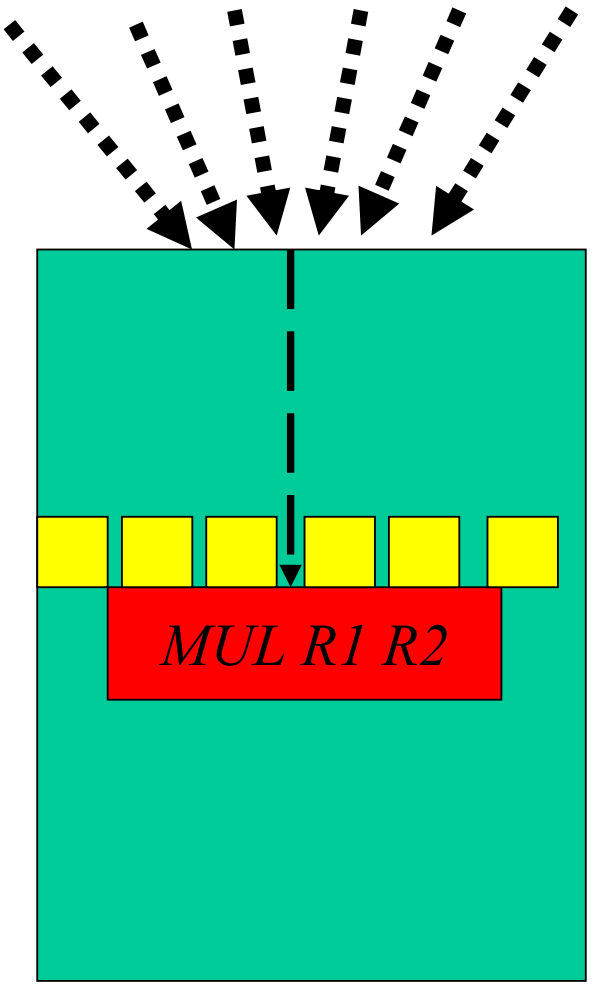
History Dependence

- Execution time of individual instructions is (extremely) history dependent:
  Has execution reached state s_1 or s_2?
- Needs consideration of the paths to this instruction
- Invariant about a set of paths to this instruction describes common properties about execution states – cache contents, pipeline occupancy, etc.



Determination of Invariants

- Static program analysis determines invariants
- Differentiation of contexts – partition of this set of paths – is important for precision. Example: caches in loops



Timing Accidents and Penalties

Timing Accident – cause for an increase of the execution time of an instruction

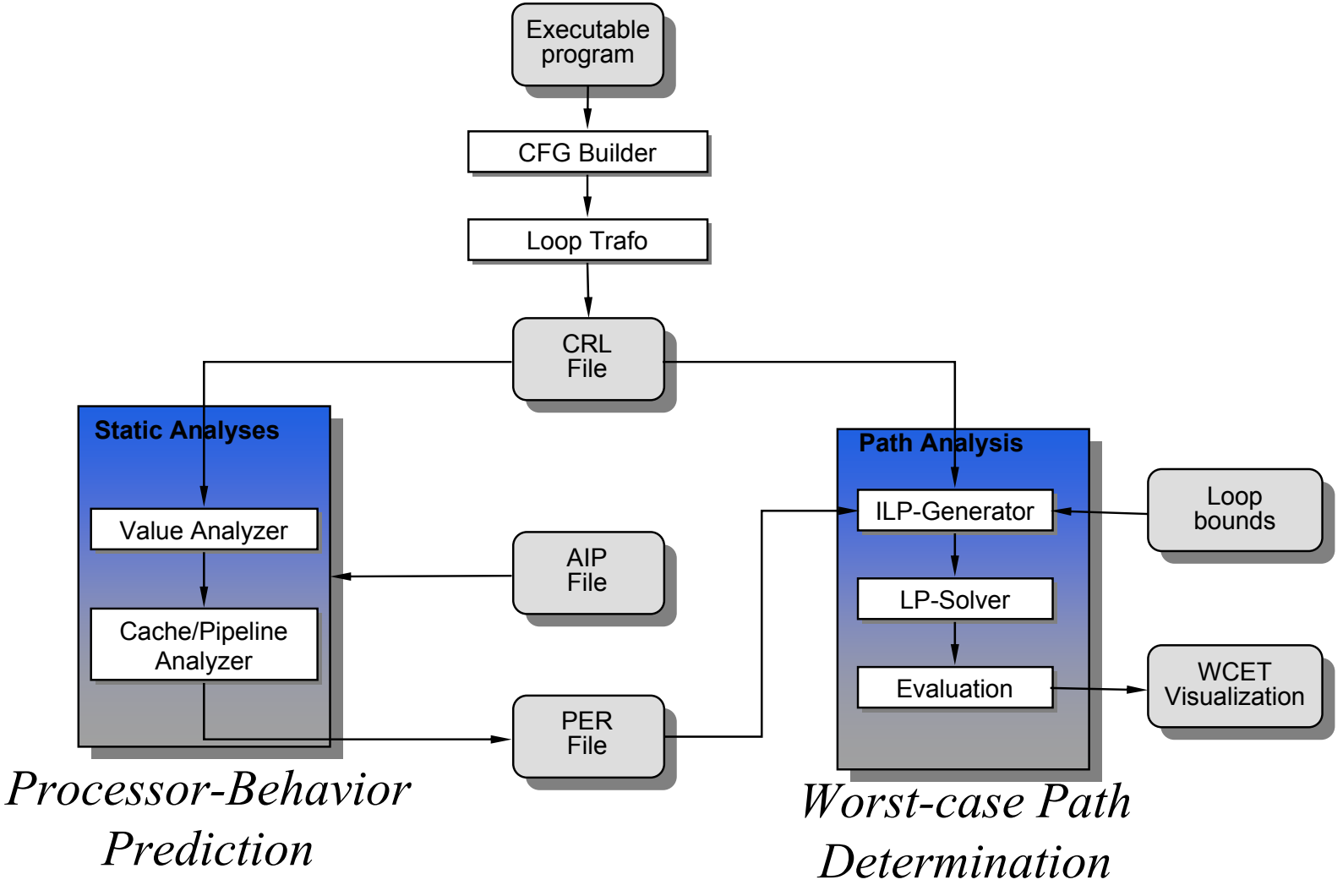
Timing Penalty – the associated increase

- Types of timing accidents
  - Cache misses
  - Pipeline stalls
  - Branch mispredictions
  - Bus collisions
  - Memory refresh of DRAM
  - TLB miss

Overall Approach: Natural Modularization

1. Processor-Behavior Prediction:
   - Uses static program analysis
   - Excludes as many Timing Accidents as possible
   - Determines WCET for basic blocks (in contexts)
2. Worst-case Path Determination
   - Maps control flow graph to an integer linear program
   - Determines upper bound and associated path

Overall Structure



Murphy's Law in Timing Analysis

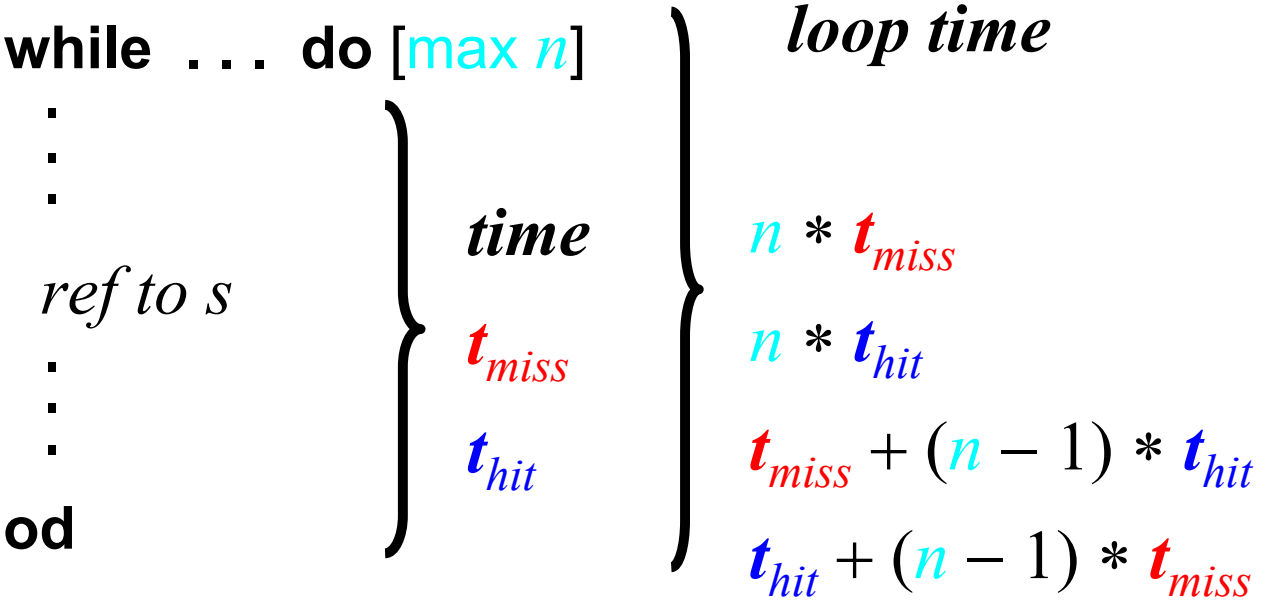
- Naïve, but safe guarantee accepts Murphy's Law:
  Any accident that may happen will happen
- Consequence: hardware overkill necessary to guarantee timeliness
- Example: Alfred Rosskopf, EADS Ottobrunn, measured performance of PPC with all the caches switched off (corresponds to assumption 'all memory accesses miss the cache')
  Result: Slowdown of a factor of 30!!!

Fighting Murphy's Law

- Static Program Analysis allows the derivation of Invariants about all execution states at a program point
- Derive Safety Properties from these invariants :
  Certain timing accidents will never happen.
  Example: At program point p, instruction fetch will never cause a cache miss
- The more accidents **excluded**, the **lower** the **upper** bound
- (and the more accidents **predicted**, the **higher** the **lower** bound)

Contribution to WCET

Information about cache contents sharpens timings.



Path Analysis

by Integer Linear Programming (ILP)

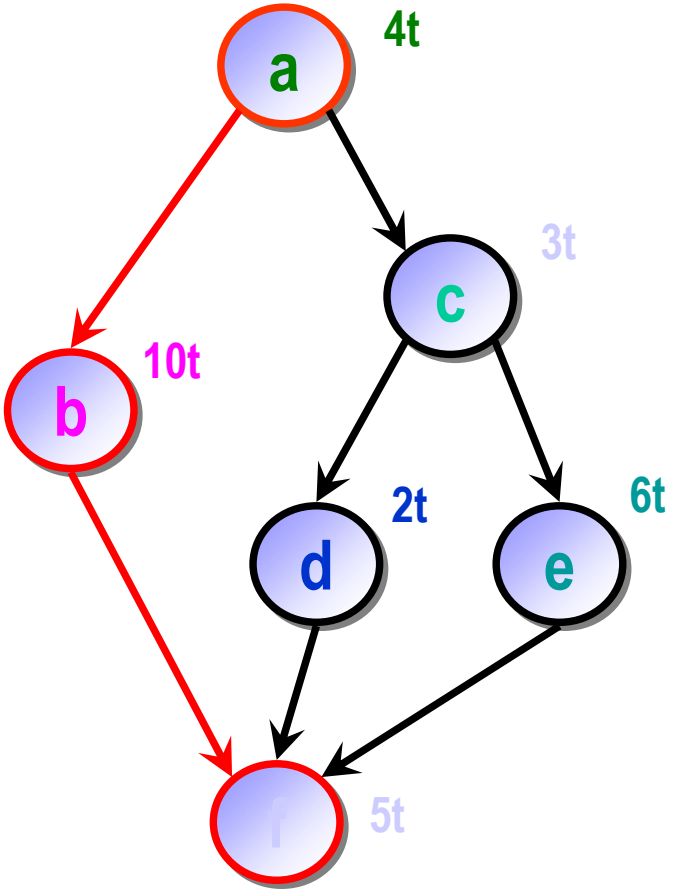
- Execution time of a program =

$$\sum_{\text{Basic\_Block } b} \text{Execution\_Time(b)} \times \text{Execution\_Count(b)}$$

- ILP solver maximizes this function to determine the WCET
- Program structure described by linear constraints
  - automatically created from CFG structure
  - user provided loop/recursion bounds
  - arbitrary additional linear constraints to exclude infeasible paths

Example (simplified constraints)

```
if  a  then
    b
elseif  c  then
    d
else
    e
endif
f
```



max: $4\,x_a + 10\,x_b + 3\,x_c +$

$2\,x_d + 6\,x_e + 5\,x_f$

where $x_a = x_b + x_c$

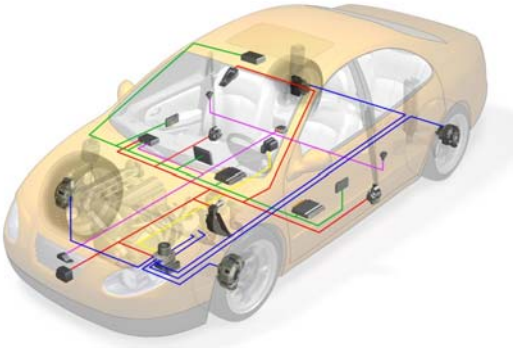
$x_c = x_d + x_e$

$x_f = x_b + x_d + x_e$

$x_a = 1$

Value of objective function: 19	
x_a	1
x_b	1
x_c	0
x_d	0
x_e	0
x_f	1

Industrial Experience



Tools used in the aeronautics, astronautics, and automotive domains

Analysis Results (Airbus Benchmark)

Task	Airbus' method	**aiT**'s results	precision improvement
1	6.11 ms	5.50 ms	10.0 %
2	6.29 ms	5.53 ms	12.0 %
3	6.07 ms	5.48 ms	9.7 %
4	5.98 ms	5.61 ms	6.2 %
5	6.05 ms	5.54 ms	8.4 %
6	6.29 ms	5.49 ms	12.7 %
7	6.10 ms	5.35 ms	12.3 %
8	5.99 ms	5.49 ms	8.3 %
9	6.09 ms	5.45 ms	10.5 %
10	6.12 ms	5.39 ms	11.9 %
11	6.00 ms	5.19 ms	13.5 %
12	5.97 ms	5.40 ms	9.5 %

Interpretation

- Airbus' results obtained with legacy method: measurement for blocks, tree-based composition, added safety margin
- ~30% overestimation
- aiT's results were between real worst-case execution times and Airbus' results

Tidorum's Bound-T Tool in Space Applications

- Mission-critical systems
- Experiments gave hints time leaks, led to redesign
- Supports rather simple processors as used in space applications, no caches, no deep pipelines, no speculation

Timing-Analysis Case Studies in Sweden

Performed by Mälardalen University with aiT

Analysis of different types of code

- time-critical, but not hard real time
- often parametric in system parameters

ENEA operating system on ARM7TDMI

Time-critical code parts:

System calls, context switches, disable interrupt regions

Results (published in ISOLA 2004)

- Much manual annotation and expert knowledge of code needed
- Overestimation of 0 – 7% against measured execution times
- very parametric bounds – depending on system state, e.g. number of tasks

Not the typical application of Timing Analysis

Volcano Communications Technologies

Data-commun. Software on MC9S12DP256

(MC68HCS12)

Results (submitted to Euromicro 2005):

- Quite parametrical behavior, e.g. in system parameters
- Hard to automate analysis, annotations needed
- Interesting to analyze bounds under certain system conditions
- Lower bounds interesting for determining jitter

Conclusions (from Volcano study)

MS Thesis S. Byhlin, Mälardalen

Dynamic Analysis (measurement)	
Advantages	**Disadvantages**
Simple	Time Consuming
	Error Prone
	Target machine required
	Program execution required
	Undetailed results
	Difficult for small code snippets
	Unsafe WCET results

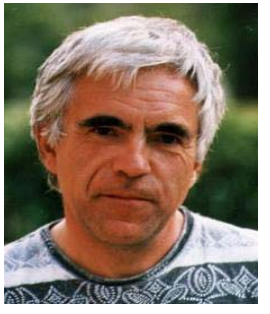
Conclusions (from Volcano study)

Static Analysis	
Advantages	**Disadvantages**
Theoretically guaranteed **safe** WCETs	
Correct annotations => correct results	Incorrect annotations => incorrect results
No program execution required	Time consuming in the beginning
User—friendly tools available	Good knowledge about the analyzed system required
Detailed results	Expensive if different analysis tools are required
Simple for small code snippets	Compiler dependent

Subject of ongoing work

Who needs Timing-Analysis Tools?

- TTA
- Synchronous languages
- Stream-oriented people
- UML real-time profile
- Hand coders



Conclusion

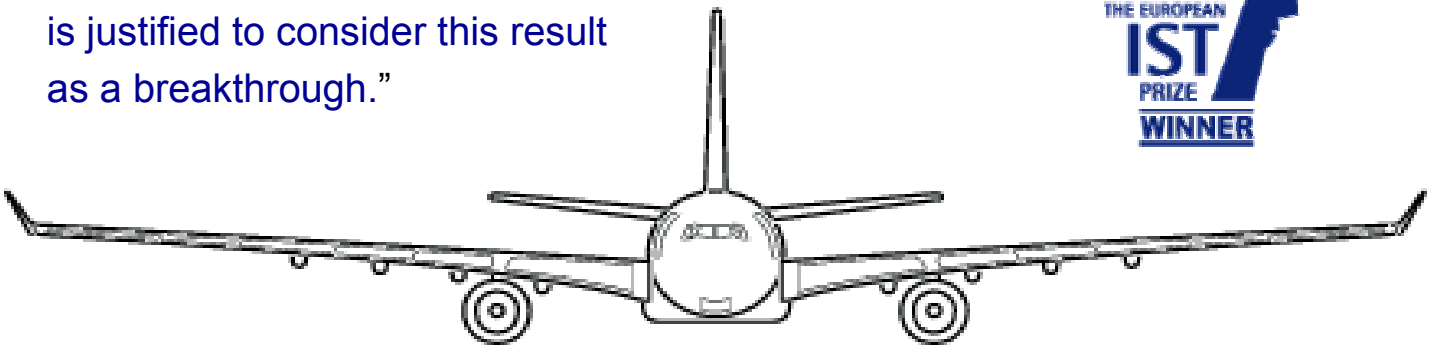
Existing Timing-Analysis tools enable the development of complex hard real-time systems on state-of-the-art hardware

- **Increase safety**
- **Save development time**
- **No over-provisioning**

  precise timing predictions enable the use of cost-efficient hardware

aiT WCET Analyzer
European Perspective

IST Project DAEDALUS final review report:
"The AbsInt tool is probably the best of its kind in the world and it is justified to consider this result as a breakthrough."



- Europe is leading in program analysis and WCET research.
- AbsInt actively participates in the definition of a European WCET research framework (NoE).

Conclusion II

- **ARTIST2** gathers most of the Timing-Analysis research worldwide and most available tools
- Problem for the mono-processor solved
- Tool development complex – Support in German transregional research center AVACS
- Usability needs improvement – Work in ARTIST2 cluster on Timing Analysis