


Reconciling compilers and timing analysis

Peter Marwedel, Heiko Falk

TU Dortmund/Informatik 12

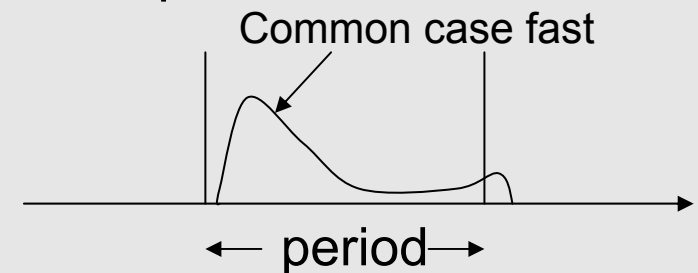
& ICD/ES

Germany


- Importance of timing recognized by some real-time specialists
- First commercially successful timing analysis tools
- But: $WCET_{EST}$ computed **after** software generation
- Even if we use save $WCET_{EST}$ guarantees 

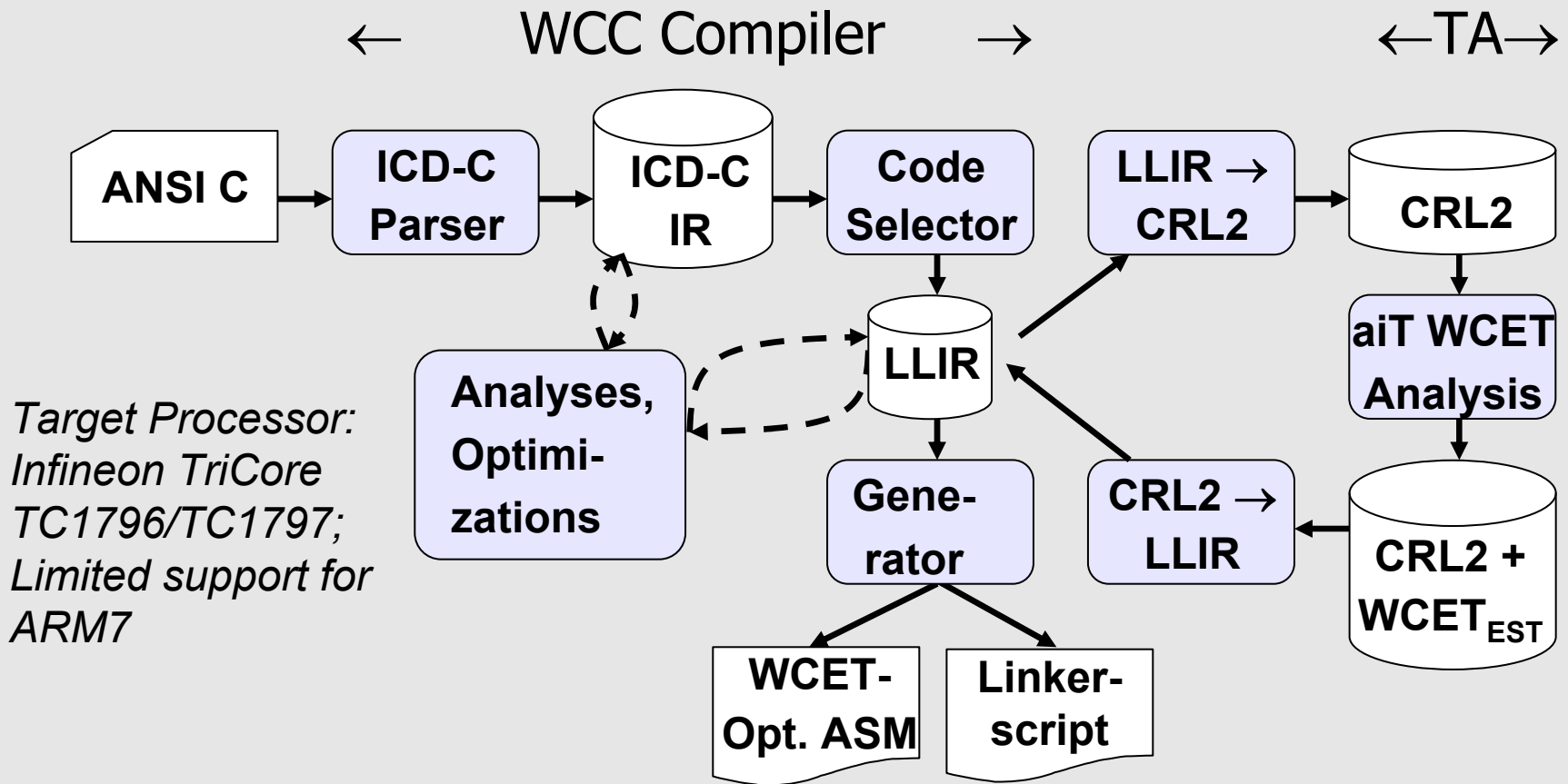
1. Specification of real-time software
2. Generation of Code (ANSI-C or similar)
3. Compilation for given target processor
4. Computation of $WCET_{EST}$
5. If “ $WCET_{EST}$ ” $>$ real-time constraint: change some detail, go back to 1 or 2.

- Design time
 - How to find necessary changes?
 - How many iterations until successful?
- “Make the common case fast” a wrong approach for RT-systems
 - Computer architecture and compiler techniques focus on average speed
 - Circuit designers know it’s wrong
 - Compiler designers (typically) don’t
- “Optimizing” compilers unaware of cost functions other than code size



How to reconcile compilers and timing analysis?

- Timing analysis tools already very complex
- Adding timing analysis to compilers would add to their complexity
- Follow a golden principle of computer science:
Implement a certain functionality once and only once! 



Optimization for WCET_{EST}!

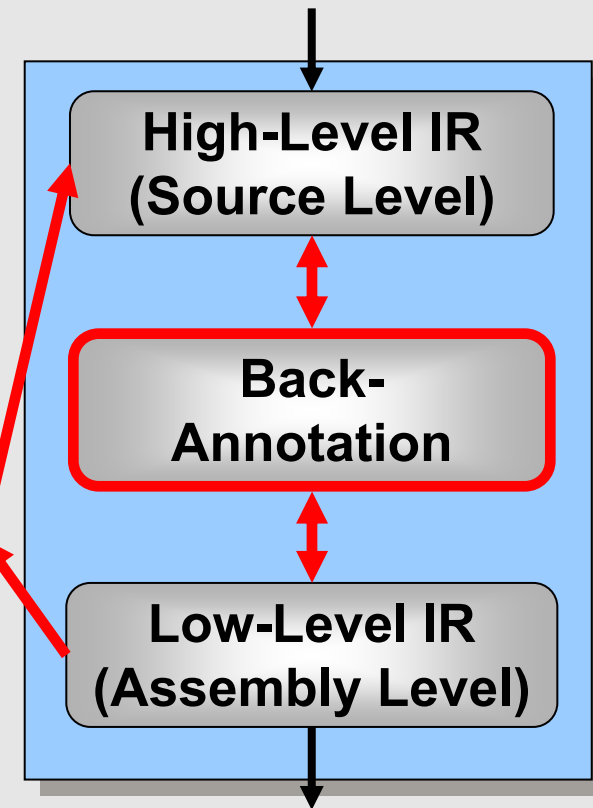
- Instruction cache locking (CODES/ISSS 07)
- Cache partitioning (WCET-Workshop 09)
- Procedure positioning (ECRTS 08)
- Procedure cloning (... , CODES/ISSS 07, SCOPES 08)
- Function inlining (SMART 09)
- Loop unswitching/invariant paths (SCOPES 09)
- ➔ • Loop unrolling (ECRTS 09)
- Register allocation (DAC 09)
- Scratchpad optimization (DAC 09)
- Loop invariant code motion/machine learning (SMART 10)
- Multi-objective optimization (RTSS 08, ISORC 10)

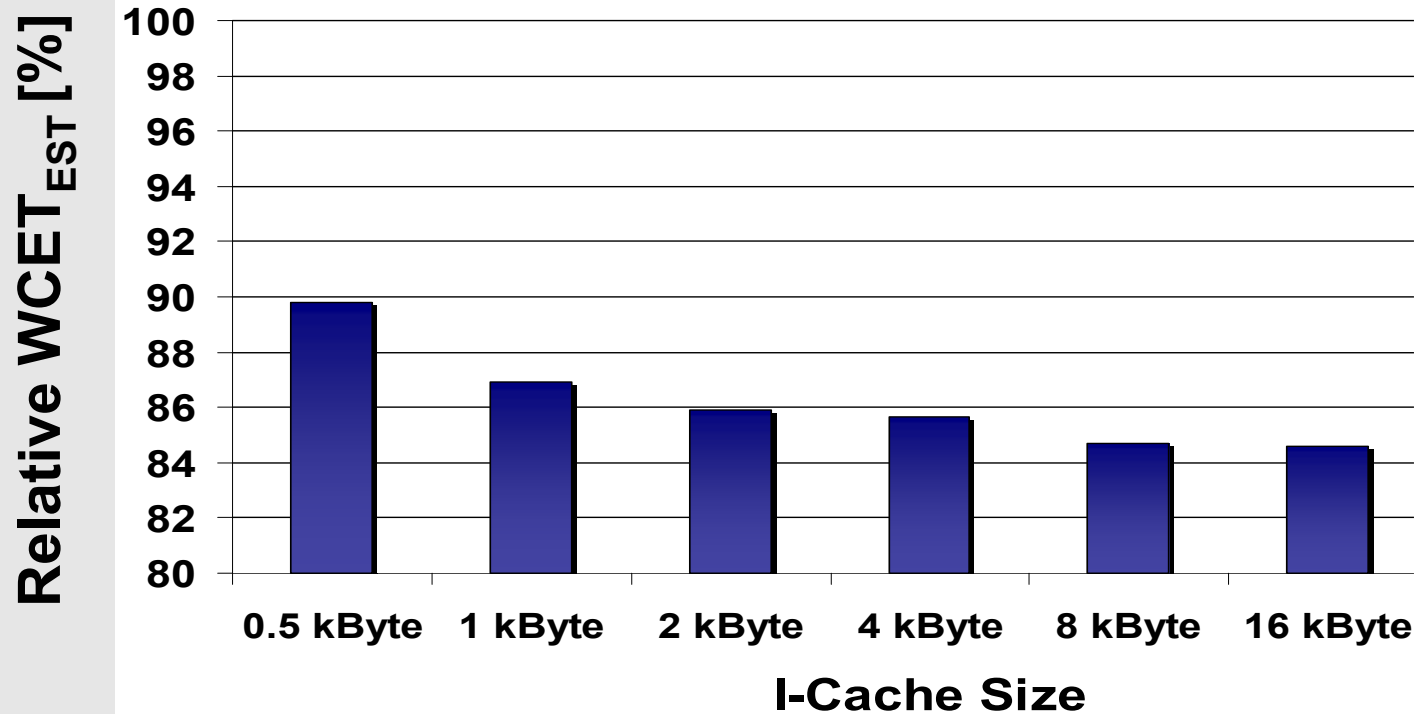
- Unrolling replaces the original loop with several instances of the loop body
- **Positive Effects**
 - Reduced overhead for loop control
 - Enables instruction level parallelism (ILP)
 - Offers potential for following optimizations
- Unroll *early* in optimization chain
- **Negative Effects**
 - Aggressive unrolling leads to I-cache overflows
 - Additional spill code instructions
 - Control code may cancel positive effects

Consequences of transformation hardly known

- WCET-information available at assembly level
- Unrolling to be applied at internal representation of source code
- Solution: Back-annotation: WCC allows feeding information from assembly level back to source level
 - WCET data
 - Assembly code size
 - Amount of spill code
- Memory architecture info available

**Mem.
Spec.**



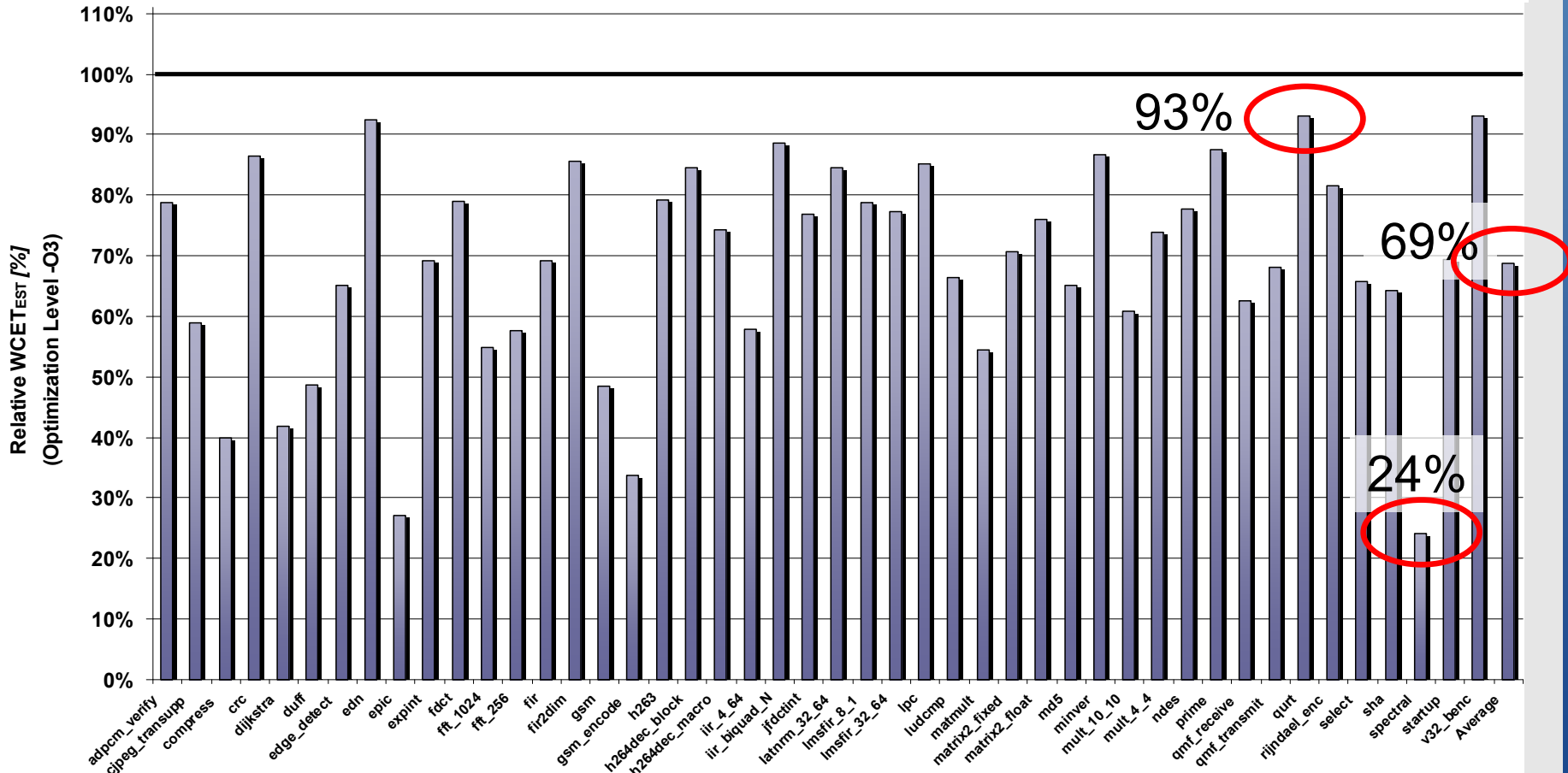


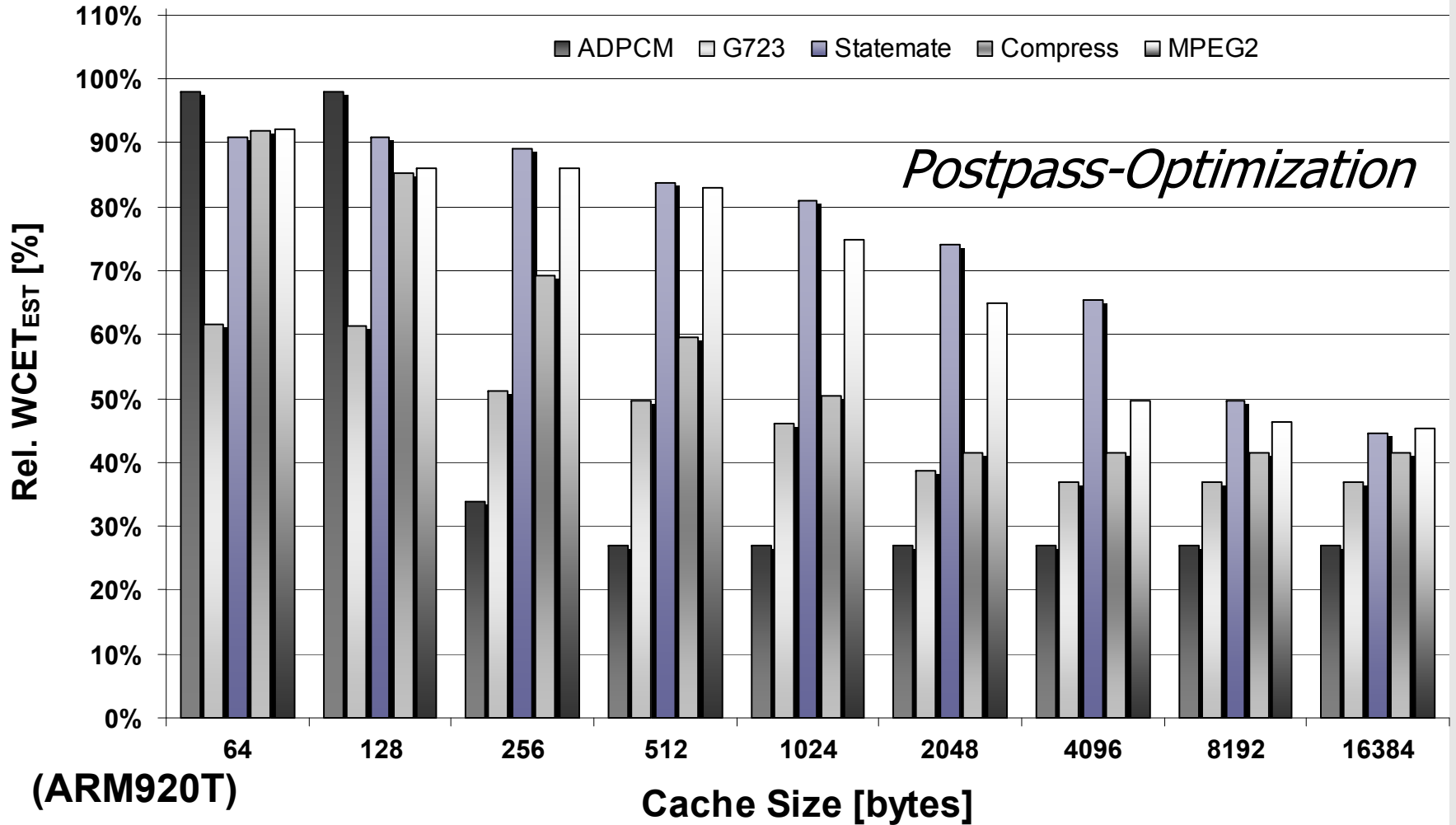
- 100%: Avg. WCET for all benchmarks with `-O3` & no unrolling
- WCET-driven unrolling outperforms standard unrolling by ave. **13.7%**
- WCET-driven unrolling outperforms std. by **10.2%** to **15.4%**



WCET_{EST}-reduction by WCET_{EST}⁻ directed graph coloring

100% = WCET_{EST} using Standard Graph Coloring (highest degree)





- Automotive: slower and cheaper microcontrollers
- Real-time sensors (physics): increased sampling rate
- Multimedia: better algorithms/less energy
- Control loops: improved stability

Exploitation

- In cooperation with ICD (a local spin-off) for maintenance etc.

- Completely new look at the use of compilers in real-time systems design
- Reflects the trend to “cyber-physical systems”
- Significant potential for increased efficiency
- WCC is a WCET-aware compiler for commercial processors (no toy examples)
- Developed in cooperation with industrial partners
- Exploitation scheduled