Timing Analysis for Sensor Network Nodes of the Atmega Processor Family

By: Sibin Mohan,
Frank Mueller,
David Whalley, and
Christopher Healy

Introduction

- Networks of Embedded systems (EmNets)
- Atmel Atmega family CPUs
- Limited Research into timing constraints on these architectures
- Goal: Provide a timing framework of tools
  - Tool should give worst-case execution time (WCET)
  - WCET should be tight bound of actual timing
  - WCET should be safe (never underestimate actual)

Verification of result correctness

Use a 3 step system

- Compile and run code on actual hardware
- Run same code on cycle-accurate simulator (provided by hardware manufacturer)
- Run same code through developed timing analysis framework

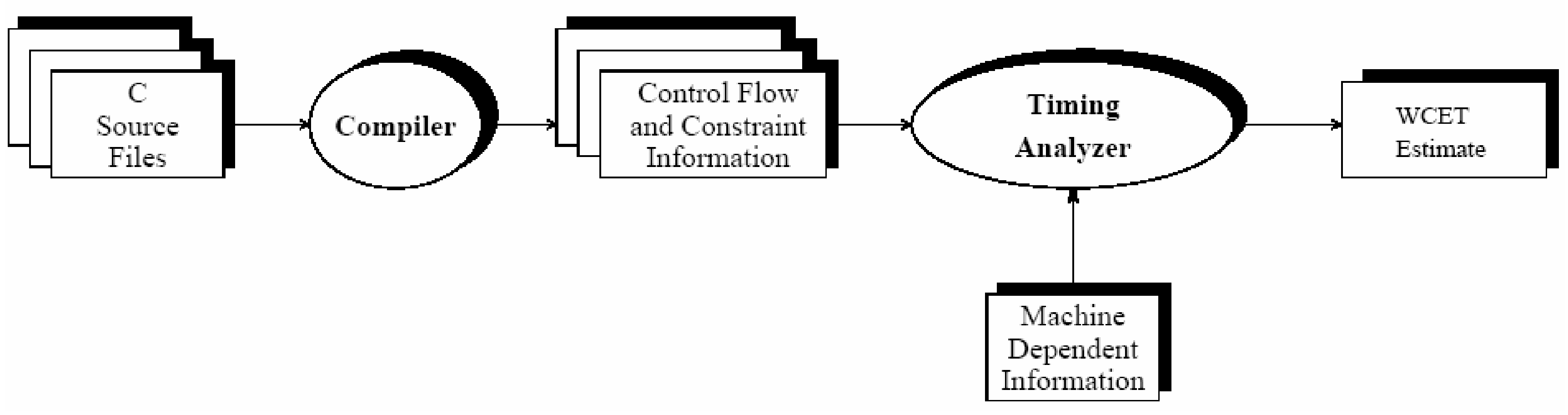
Types of Timing Analysis:

- Dynamic
  - Simulates execution on worst case input
  - WCET Safety can not be guaranteed
  - Can be difficult to determine worst input set
  - Hardware/Software interactions can hide worst-case
    - Architectural complexities: pipelines and cache
- Static
  - Examines code
  - Analyzes all possible execution paths
  - Combines paths to construct worst case execution
  - Does not trace code execution
  - Does not take variable state into consideration
  - Guarantees WCET

Timing Analysis Framework

- Performs Static Timing Analysis
- Originally designed for SPARC I
- Modified to support Atmel architecture
- Enhanced to provide tighter WCET bounds
- Uses Fixed-point algorithm to determine WCET of loops and functions
- WCET tree constructed:
  - Paths -> loops -> functions -> program

Timing Analysis Framework


C Source Files
Compiler
Control Flow and Constraint Information
Timing Analyzer
WCET Estimate
Machine Dependent Information

- Program information:
  - Compiler produces Control Flow
  - Loop bounds through analysis or programmer
- Hardware information
  - Cache description and behavior
  - Pipeline description

Timing Analyzer: Pipeline

- Pipeline Simulator handles for each path
  - Structural Hazards
  - Data Hazards
  - Branch Prediction
  - Cache Misses

Timing Analyzer: Path Analysis

- Takes path info from pipeline simulator
- Longest execution path selected
- Fixed-Point algorithm for loops
  - Uses longest path of loop body
  - Faster each iteration (benefit from cache)
  - Stop when cache stops improving execution of body
  - Can now bound WCET of loop

Modifications to Architecture

Variable Cycle Instructions

Example:

- Branch: 1 cycle if not taken
  - Fall through to next instruction
- Branch: 2 cycle if taken
  - Memory lookup of target instruction required

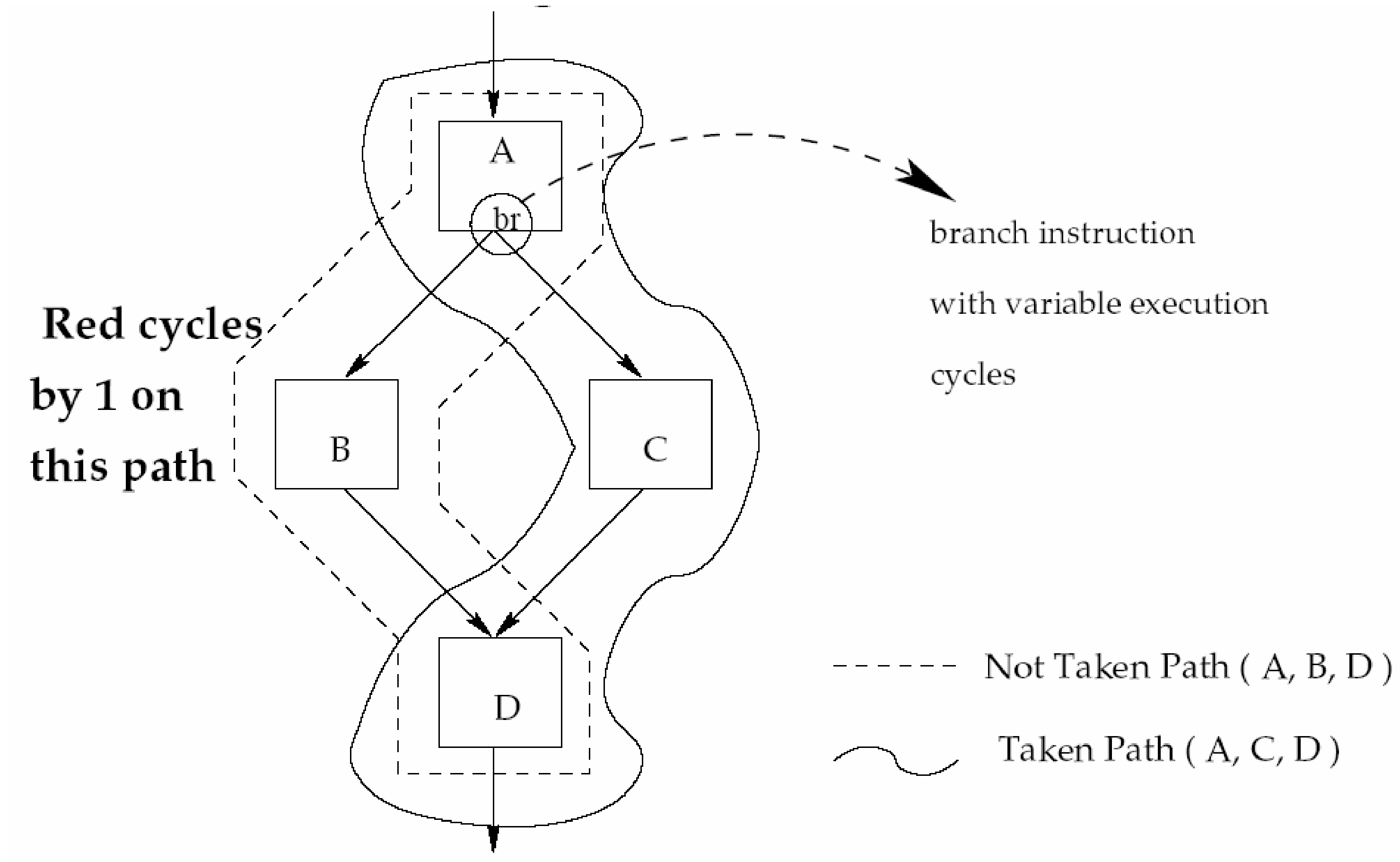
Past approaches would assume always max

- Overly pessimistic, especially when in loops
- Unnecessarily bloats WCET estimates

Variable Cycle Instructions

- During fixed-point algorithm, instructions that modify control flow are analyzed
- Solution: modify length of path chosen by this instruction
- For branch example: assume instruction takes 2 cycles and reduce not-taken path by one cycle to compensate.
- Fixed-point algo will operate as before and produce tighter WCET bound

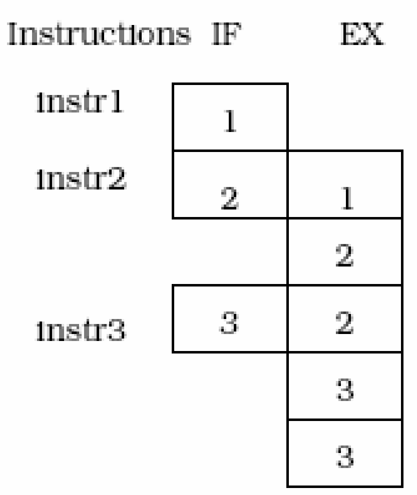
Variable Cycle Instructions


A
br
B
C
D
Red cycles by 1 on this path
branch instruction with variable execution cycles
Not Taken Path (A, B, D)
Taken Path (A, C, D)

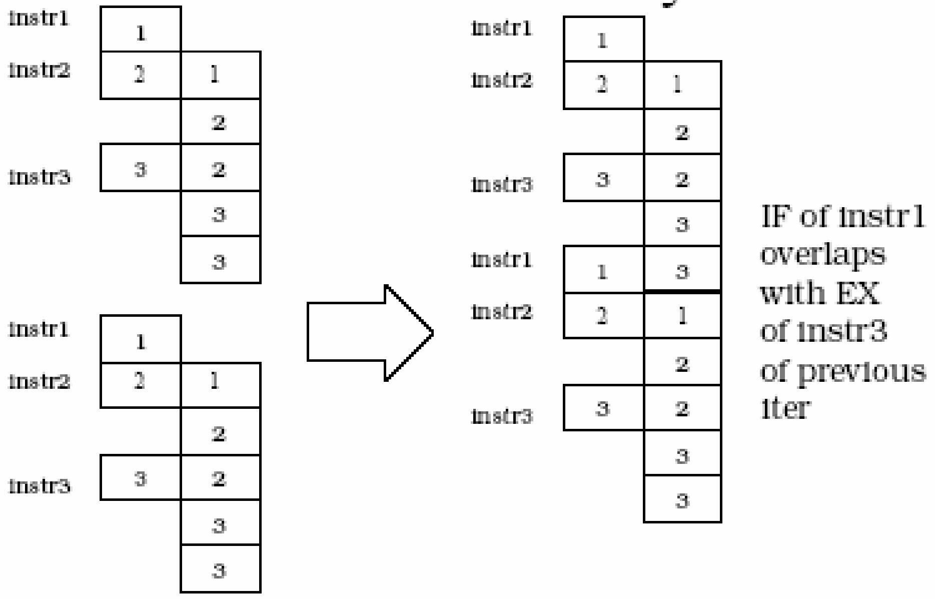
Modifications to Architecture

Pipeline Modeling across Loops Iterations

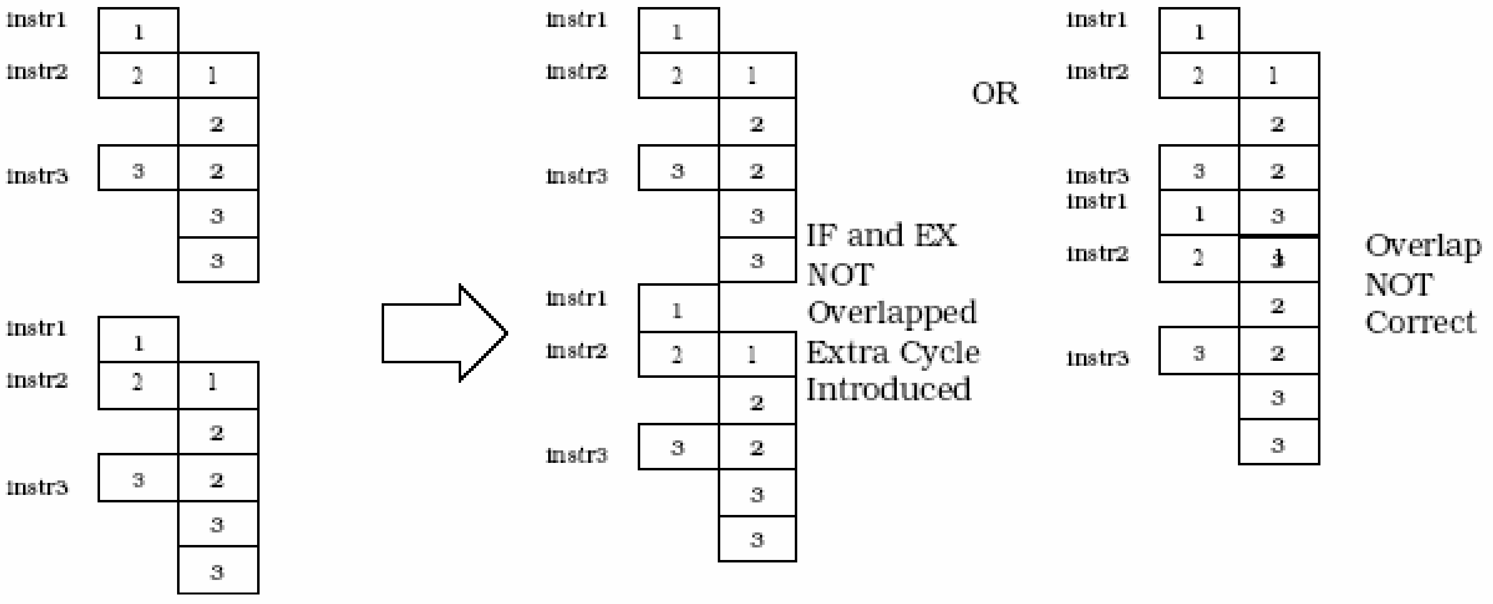
- Simulators tend to estimate WCET of loops by *(loop count)* * *(body WCET)*
- Ignores pipeline state between iterations
- Common errors when compensating
  - Place blocks end to end producing noop (ci)
  - Place IF phase end to end overlapping EX phase of pipeline (cii)



(a) Single Iteration of Loop



(b) Correct Handling of Loop Iterations



(i) Over−estimation of WCET (ii) Under−estimation of WC

(c) Incorrect Handling of Loop Iterations

Atmega Architecture

- Atmega128 / Atmega103 processors
- CMOS 8-bit RISC controller
- Separate memory for program and data
- Separate bus for program and data
- Two stage pipeline IF & EX
- No cache

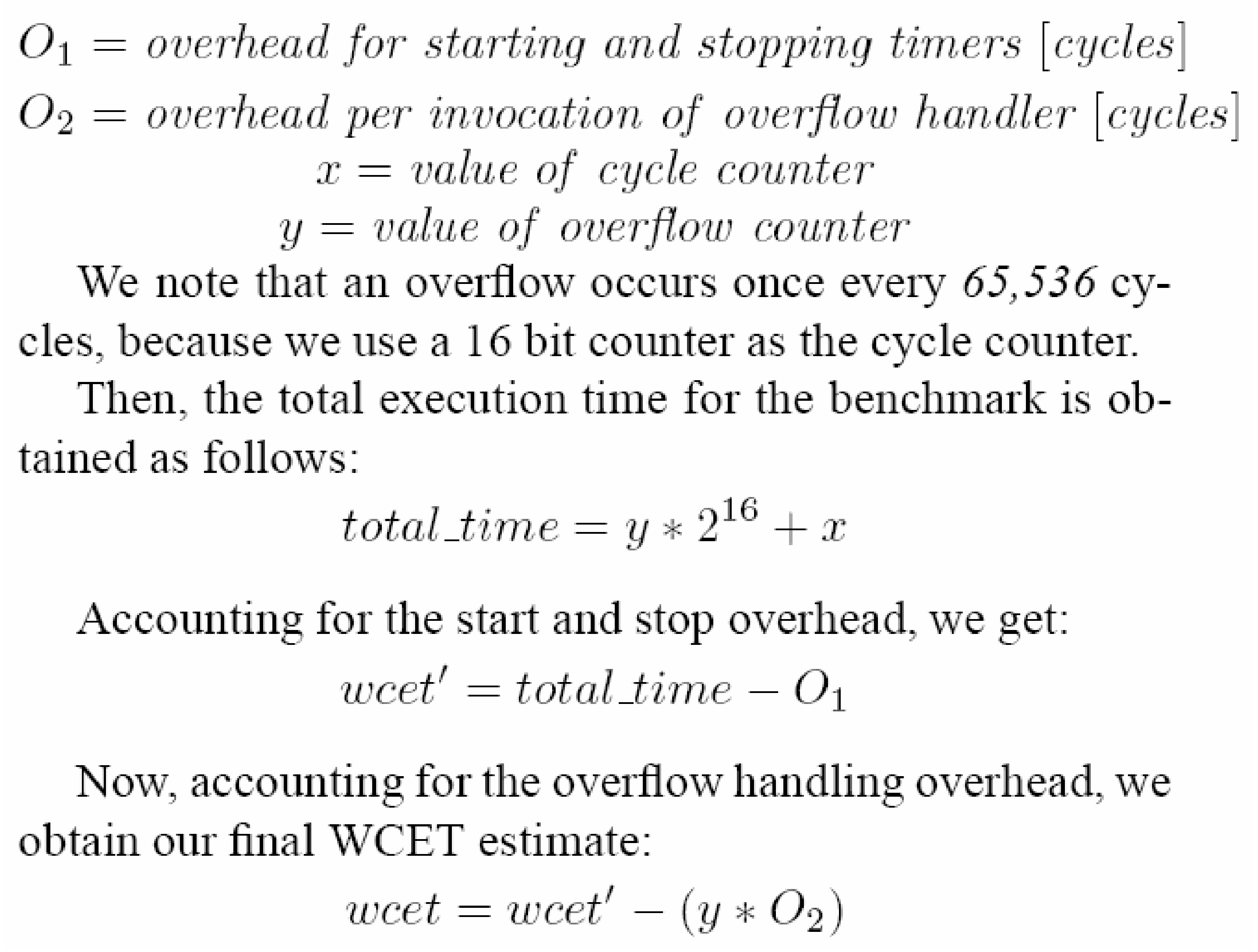
Instruction Set

- 16 bit or 32 bit wide instructions
- Integer based, floating point in emulation only
- Almost all instructions are 1 or 2 cycles
  - organized into 2 categories for analysis
- Some variable cycle instructions (loads, compares, branches…)
  - Handled through modifications described

Experimentation

- Benchmarks
  - C-Lab embedded WCET suite
  - NesC benchmarks
- Worst-case measurements in terms of processor cycle count
- Hardware timing obtained using interrupt-driven routines and hardware counters
  - Two hardware counters initialized to 0
  - Increment counter 1 at each cycle
  - At overflow of counter 1, counter 2 incremented
- Worst-case input sets manually constructed
- Same assembly output from complier used for all 3 levels of experiment

Hardware overhead compensation



$$O_1 = overhead\ for\ starting\ and\ stopping\ timers\ [cycles]$$
$$O_2 = overhead\ per\ invocation\ of\ overflow\ handler\ [cycles]$$
$$x = value\ of\ cycle\ counter$$
$$y = value\ of\ overflow\ counter$$

We note that an overflow occurs once every *65,536* cycles, because we use a 16 bit counter as the cycle counter.

Then, the total execution time for the benchmark is obtained as follows:

$$total\_time = y * 2^{16} + x$$

Accounting for the start and stop overhead, we get:

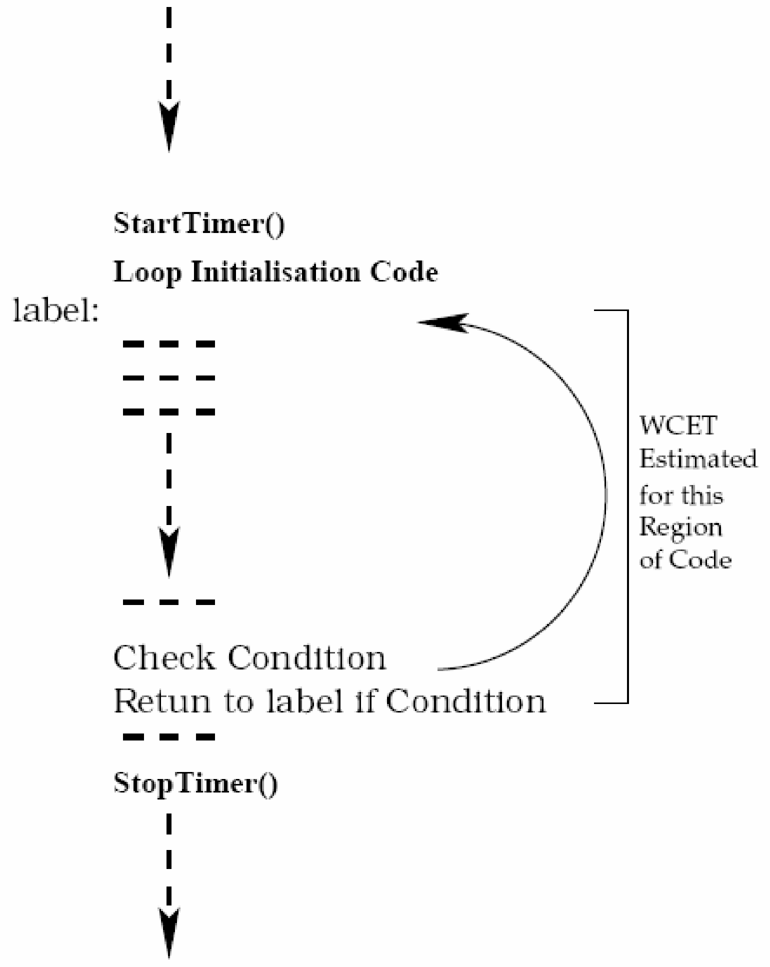
$$wcet' = total\_time - O_1$$

Now, accounting for the overflow handling overhead, we obtain our final WCET estimate:

$$wcet = wcet' - (y * O_2)$$

Hardware/Simulator vs. Timing Analyzer Mismatch

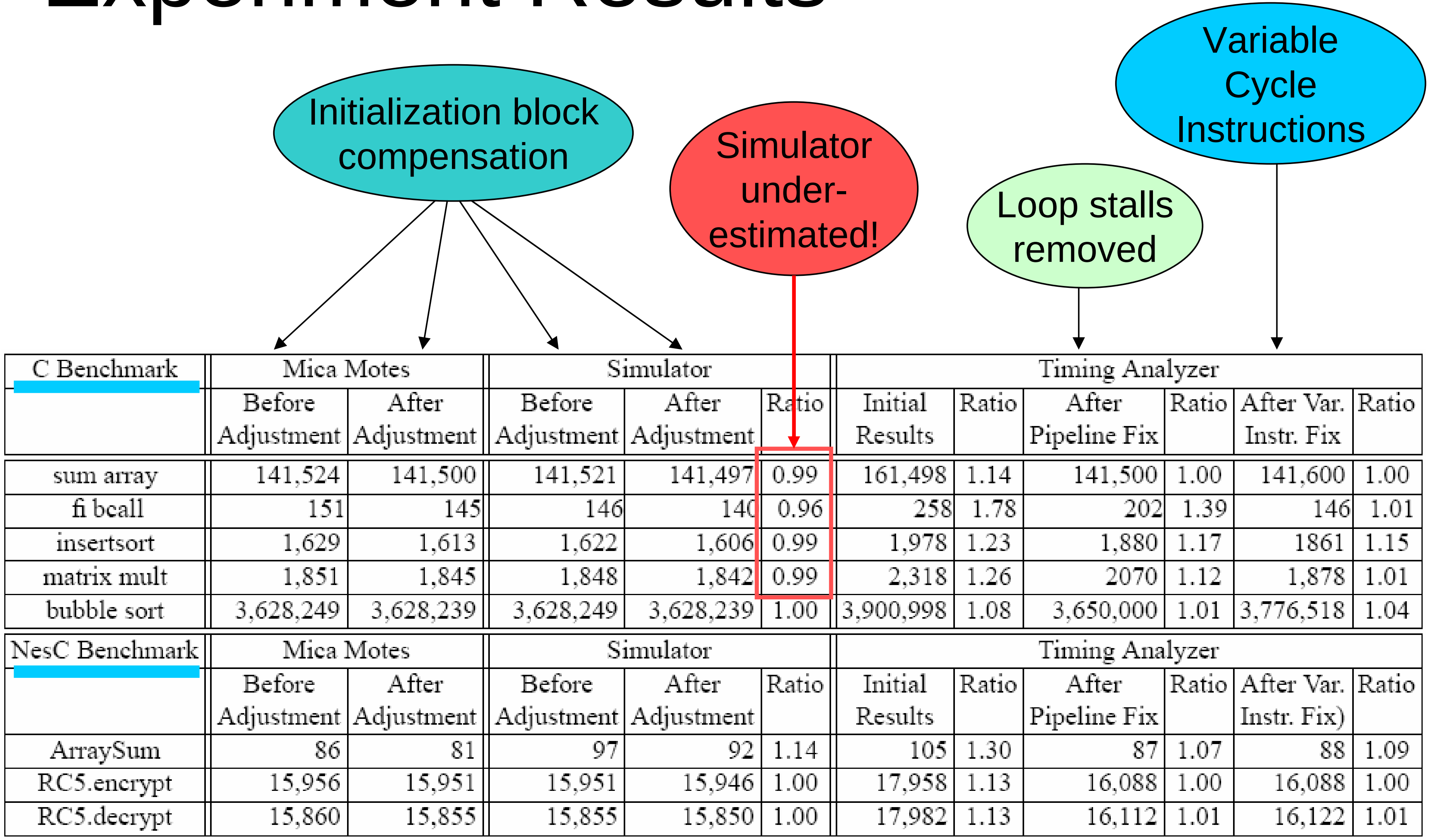
- Analyzer provides WCET for loops and functions which can leave out initialization blocks
- Hardware and simulator can provide arbitrary block WCET
- Causes result discrepancies
- Must be compensated for when comparing WCET



Timing Analysis for NesC

- Programming Language for applications running on the TinyOS platform
- Defined especially for distributed embedded wireless sensor networks
- Built on C
- NesC compiler converts to intermediate C code
- Timing analysis can be performed on intermediate code
- Loop bounds can be determined manually from C code

Experiment Results



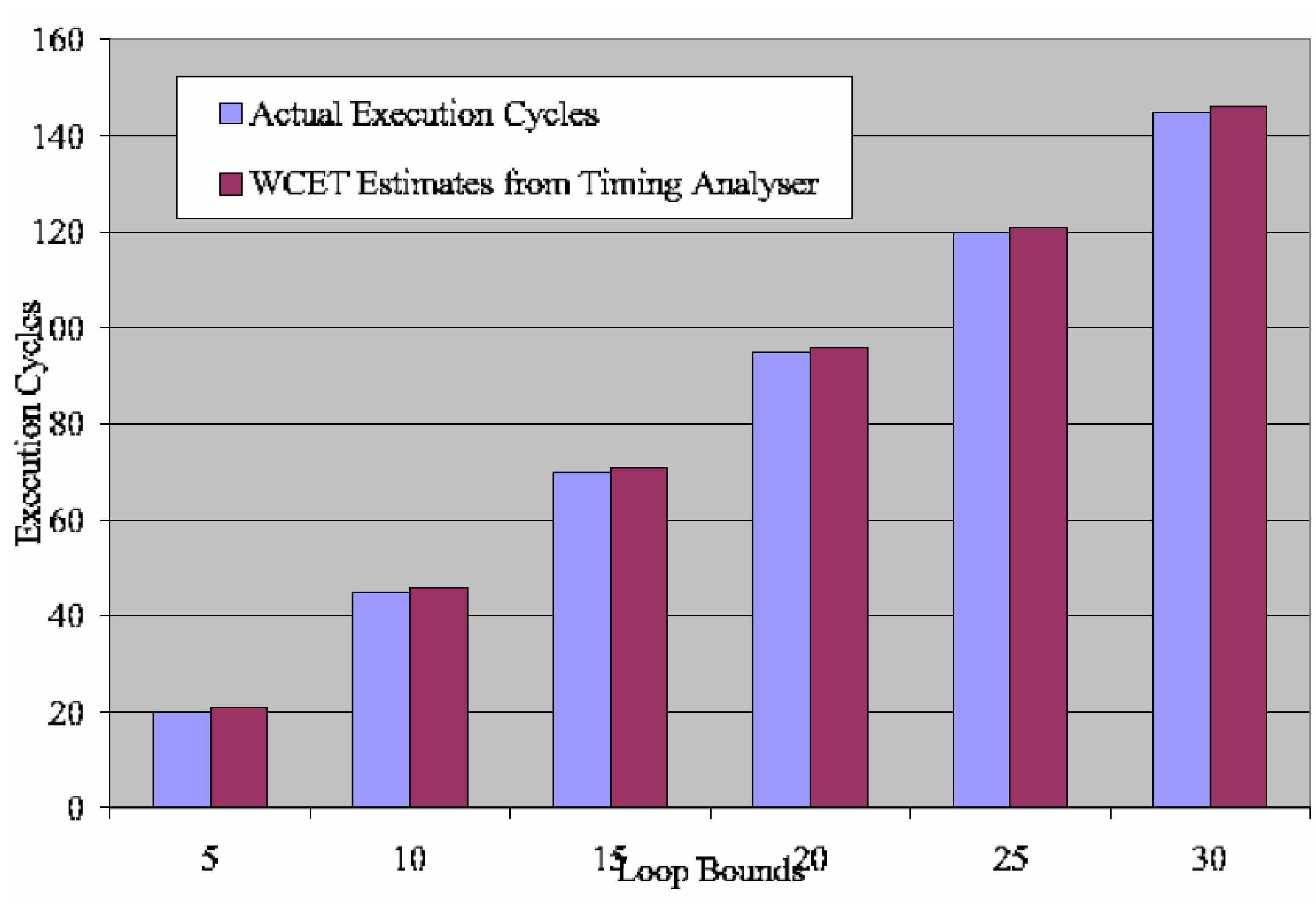
Initialization block compensation
Simulator under-estimated!
Loop stalls removed
Variable Cycle Instructions

C Benchmark	Mica Motes		Simulator			Timing Analyzer					
	Before Adjustment	After Adjustment	Before Adjustment	After Adjustment	Ratio	Initial Results	Ratio	After Pipeline Fix	Ratio	After Var. Instr. Fix	Ratio
sum array	141,524	141,500	141,521	141,497	0.99	161,498	1.14	141,500	1.00	141,600	1.00
fi bcall	151	145	146	140	0.96	258	1.78	202	1.39	146	1.01
insertsort	1,629	1,613	1,622	1,606	0.99	1,978	1.23	1,880	1.17	1861	1.15
matrix mult	1,851	1,845	1,848	1,842	0.99	2,318	1.26	2070	1.12	1,878	1.01
bubble sort	3,628,249	3,628,239	3,628,249	3,628,239	1.00	3,900,998	1.08	3,650,000	1.01	3,776,518	1.04

NesC Benchmark	Mica Motes		Simulator			Timing Analyzer					
	Before Adjustment	After Adjustment	Before Adjustment	After Adjustment	Ratio	Initial Results	Ratio	After Pipeline Fix	Ratio	After Var. Instr. Fix)	Ratio
ArraySum	86	81	97	92	1.14	105	1.30	87	1.07	88	1.09
RC5.encrypt	15,956	15,951	15,951	15,946	1.00	17,958	1.13	16,088	1.00	16,088	1.00
RC5.decrypt	15,860	15,855	15,855	15,850	1.00	17,982	1.13	16,112	1.01	16,122	1.01

Note: All ratios are with respect to Mica Motes "After Adjustment"

Scalability of Timing Analyzer



Input size scaled for *fibcall* benchmark