# Approximation of Worst-case Execution Time for Preemptive Multitasking Systems

Matteo Corti, Roberto Brega, Thomas Gross
ETH Zurich

# Outline

- Environment
- Other approaches
- Worst-case execution time approximation
- Results
- Conclusions

# Environment: User Needs

- Complex mechatronic applications
- Timing correctness
- Concurrency (RT and non-RT tasks)
- Rapid development: dynamic system
- Modern programming languages
- Modern processors

# Environment: System

- XOberon:
  - Loading/unloading of modules (tasks) at runtime
  - Deadline driven scheduler with admission testing
  - Resources are shared between RT and non-RT tasks
  - Preemptive scheduling
- Modern RISC processors: PowerPC 604e
- Modern language: Oberon-2
  - Automatic garbage collection
  - Strong type checking

# Problem Description

- Admission test
  - **deadline**: determined by the problem
  - **max. duration**: determined by the task and the system
- Preemptive scheduling and processor complexity hinder a precise computation of the worst-case execution time (WCET)
- The system is able to stop safely if the given duration is to small (w/o damaging the robot or the operator)

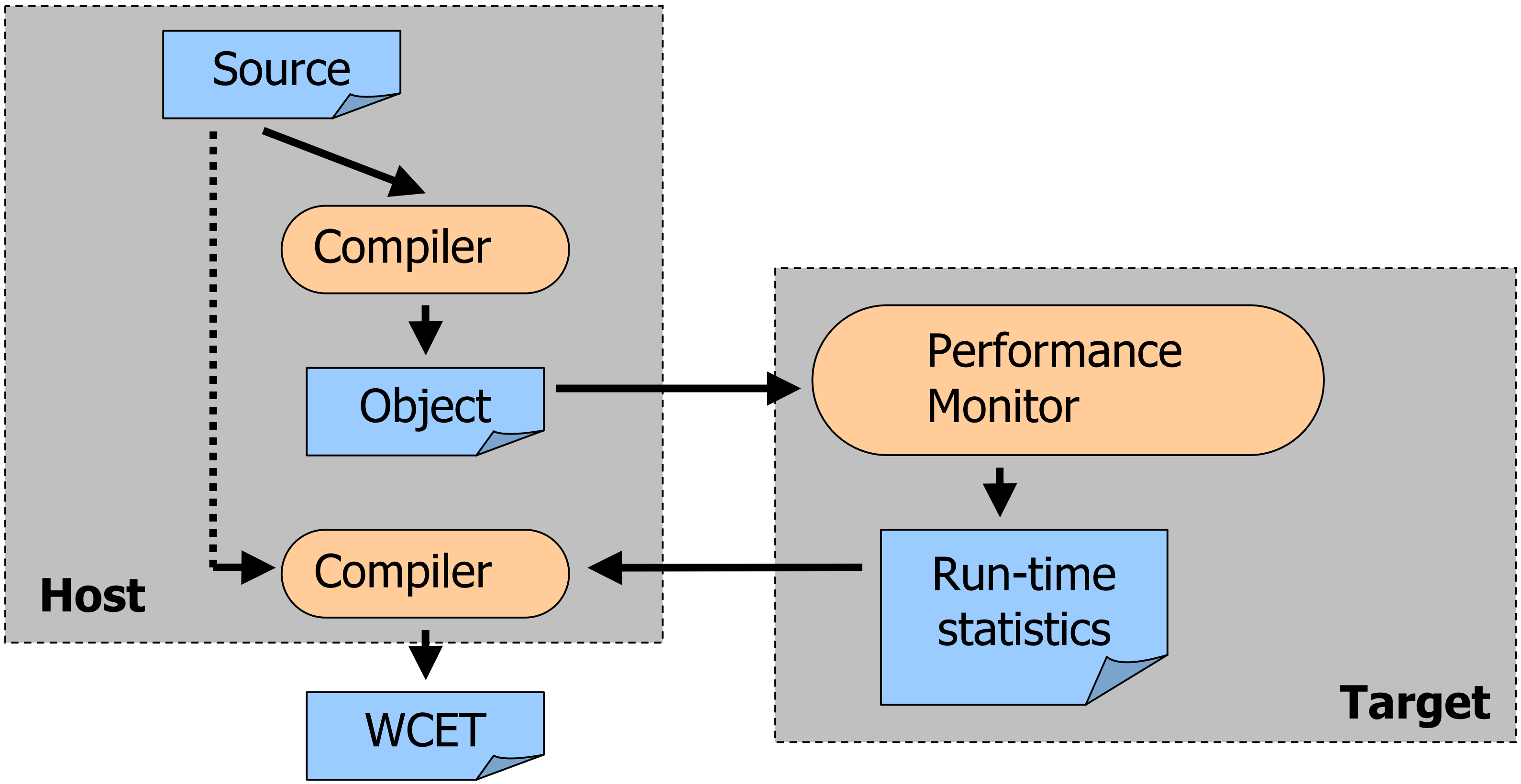
# Issues

- Static program analysis
  - automatic loop bounding
  - false paths
  - infeasible paths
- Instruction length computation
  - caches (instruction and data)
  - pipelines

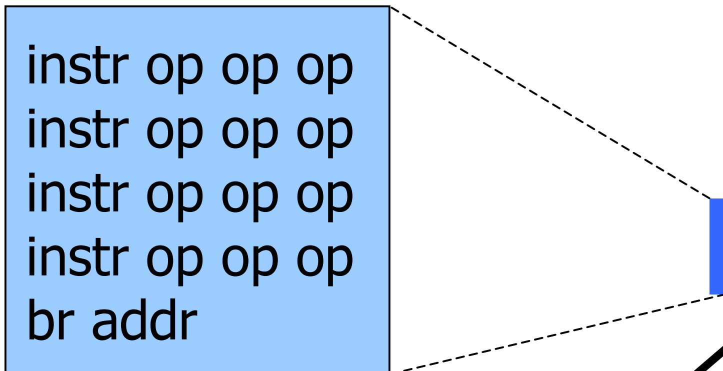
# Other approaches

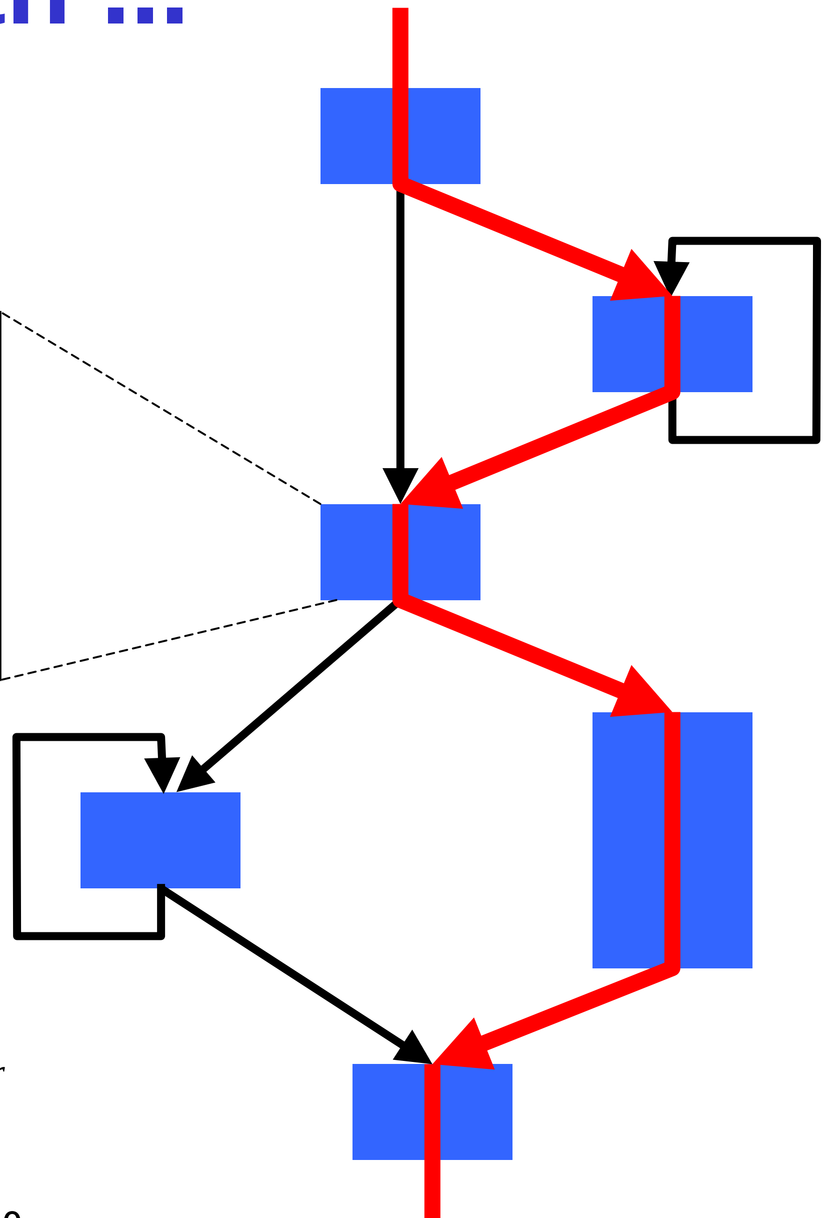
- Longest path:
  - user annotations
  - automatic tools (loop bounding, false paths, …)
- Instruction length (w/o preemption):
  - cache prediction
  - active cache management
  - pipelines prediction
- Dynamic systems:
  - trial-and-error experimentation

# Predictor Structure



# Longest Path ...



$$len_b = iter_b \cdot \sum_{instr} length_{instr}$$

# Block Iterations

- Static program analysis
  - loop iteration bounds
- Real-time tasks are relatively well structured
  ➡ minimal compiler support
  - automatic loop bounding for simple loops
  - user annotations (driver calls, *difficult* loops, polymorphism, library calls)
  - user hints can be checked at run-time

# Instruction Length

- Preemption, dynamic set of processes ➡ no exact knowledge of the cache and pipeline status
- Maximal instruction lengths (caches are always empty, instructions always stall, ...) are not useful: the WCET is too high to be used in practice

- Instruction length **approximation** using **run-time information** about the processor usage during the task's execution

# Performance Monitor ...

- The PowerPC 604e provides **hardware assist** to monitor and count predefined events (cache misses, mispredicted branches, issued instructions, ...)
- Processes can be marked for runtime profiling
- Events book-keeping is done in the scheduler (small overhead)
- No code instrumentation

# Performance Monitor

- Not specifically designed to help in program analysis:
  - event counting is not precise (out-of-order execution)
  - many events are not disjoint
  - only four different events can be monitored in parallel
- The instruction length must be **approximated** dealing with the performance monitor (PM) inaccuracies

# Statistics Gathering

- Problem: choose representative traces
- Solution:
  - profile different input sets
  - conservative approximation
- The tests confirmed a certain homogeneity within different execution traces for the same tasks

# Cycles Per Instruction (CPI) …

- The instruction length can be divided in several components:
  - ICP: infinite cache performance (CPU busy and stall time)
  - FCE: finite cache performance (effects of memory hierarchy)

$$CPI = ICP + FCE$$

$$CPI = busy + stall + FCE$$

$$CPI = \frac{exec_{unit} + stall_{unit}}{parallelism} + stall_{pipeline} + FCE$$

$$CPI = \ldots$$

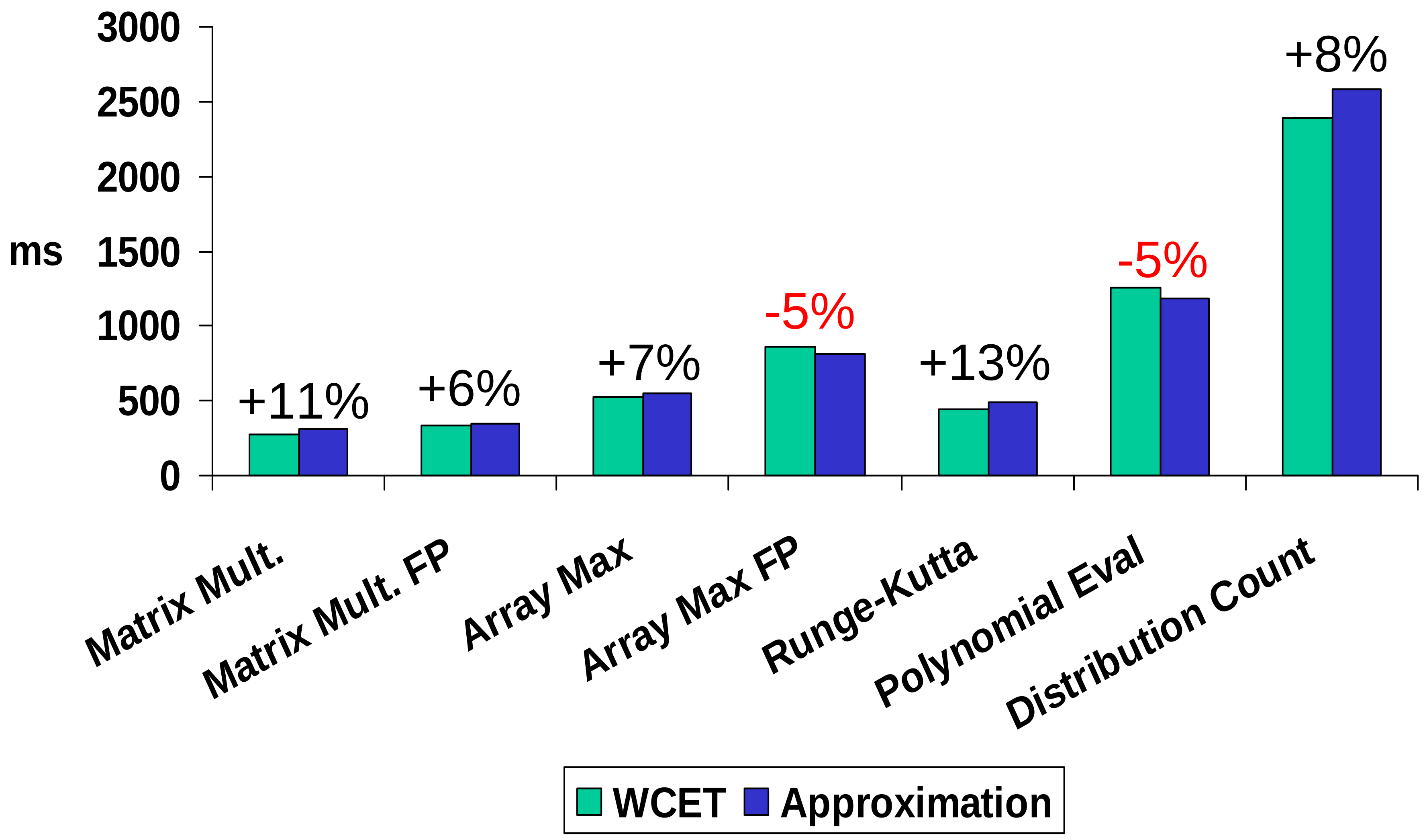
# Cycles Per Instruction (CPI)

- Instruction length components:
  - From the processor architecture
    - execution time
    - miss penalty
  - Estimated with help of run-time data
    - stalls
    - cache misses
    - instruction parallelism
  - Estimated by the program structure
    - distance between instructions of the same type

# Testing the Predictor

- First phase: approximation tuning
  - simple tests with known WCET (matrix multiplication, Runge-Kutta, …)
  - different components of the approximator and of the processor can be tested separately
- Second phase: real applications
  - longest path and exact WCET unknown
  - not all the paths can be tested

# Results: Simple Tests



# Results: Approximations

- Worst case assumptions about caches and pipeline produce non usable durations
- Example: no cache approximation (but all other included)

| Test | Matr. Mul. | Array Max. | Pol. Eval. |
|---|---|---|---|
| Measured value | 280 ms | 520 ms | 1252 ms |
| Full predictor | 311 ms | 555 ms | 1188 ms |
| No cache hits | **1403 ms** | **1901 ms** | **3193 ms** |

# Results: Real Applications

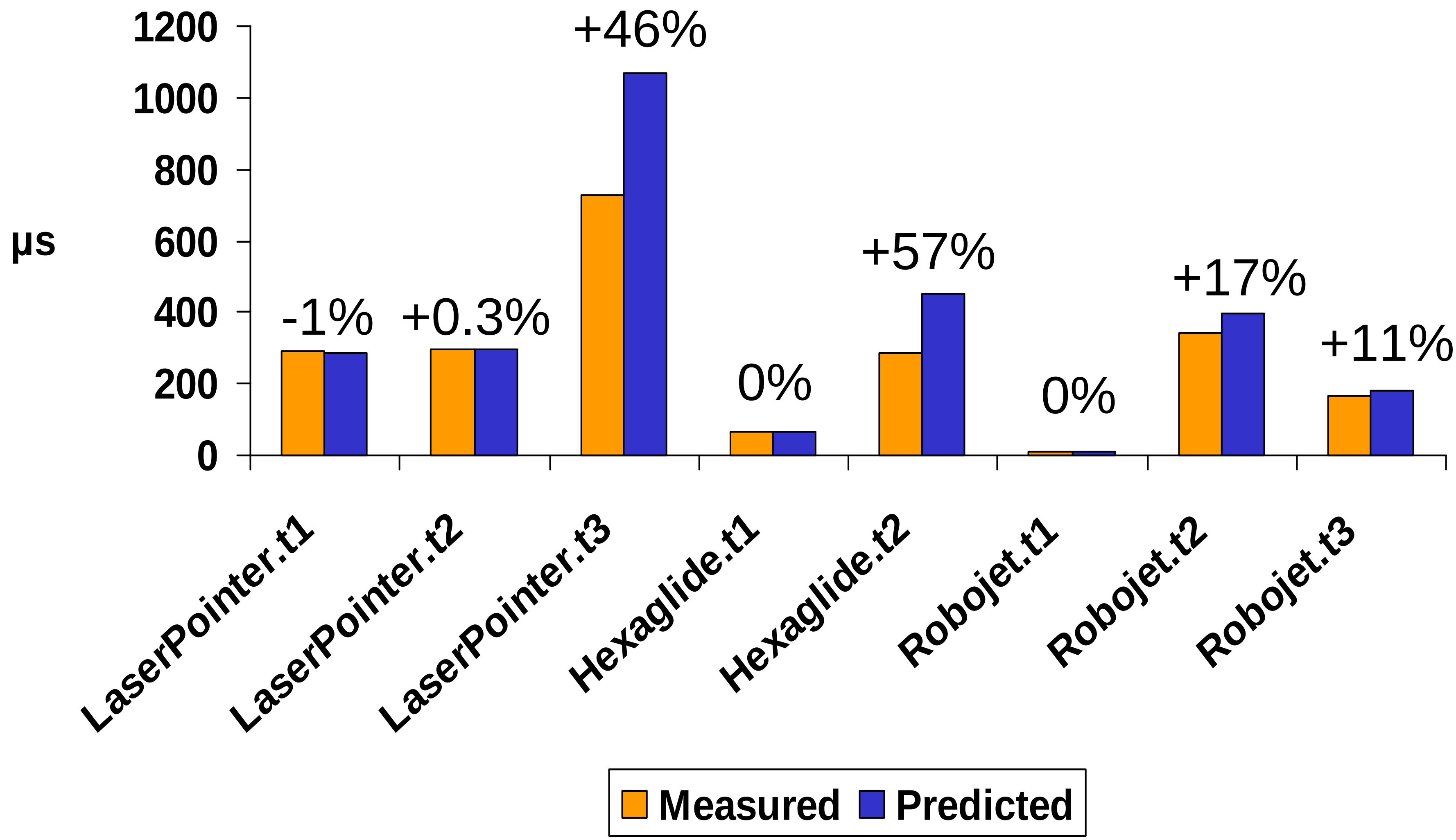
- **LaserPointer**: laboratory machine that moves a laser pen applied on the tool-center point of a 2-joints manipulator
- **Hexaglide**: a parallel manipulator with 6 DOF used as a high speed milling machine
- **Robojet**: a hydraulically actuated manipulator used in the construction of tunnels

# Results: Real Applications

- Only a few loops had to be manually bounded

| Application | User annotations | | Code Size |
|---|---|---|---|
| | Calls | Bounds | |
| LaserPointer | 5 | 0 / N.a. | 1000 LOC |
| Hexaglide | 4 | 2 / 258 | 2200 LOC |
| Robojet | 17 | 0 / 207 | 1600 LOC |

# Results: Real Applications



# Comments …

- Performance monitors are not designed to help in program analysis (coarse-grain information)
- Many CPI components are gathered using statistical methods
- There is no hard guarantee the result is correct
- Architecture dependent (different performance monitors, and processor architectures)

# Comments

- Simple approach: minimal user interaction needed (suitable for application experts)
- No special hardware tools needed
- Useful in complex environments with preemptive multitasking (dynamic constellation of real-time tasks)
- Big and real applications can be analyzed

# Conclusions

- The WCET can be approximated using run-time data
  - little or no user assistance is required
- Processor's performance monitors can help in program analysis
  - better support desirable
- Approximations are good enough for many dynamic real-time systems