TuBound – A Conceptually New Tool for Worst-Case Execution Time Analysis[1]

Adrian Prantl, Markus Schordan, Jens Knoop
{adrian,markus,knoop}@complang.tuwien.ac.at

FAKULTÄT FÜR !INFORMATIK

Institut für Computersprachen
Technische Universität Wien

computer languages

WCET'08, Prague
July 1, 2008

[1] This work has been partially supported by the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) under contract P18925-N13, *Compiler Support for Timing Analysis*, `http://costa.tuwien.ac.at/`, the ARTIST2 Network of Excellence, `http://www.artist-embedded.org/` and research project "Integrating European Timing Analysis Technology" (ALL-TIMES) under contract No 215068 funded by the 7th EU R&D Framework Programme.

Requirements for WCET analysis tools

WCET analysis

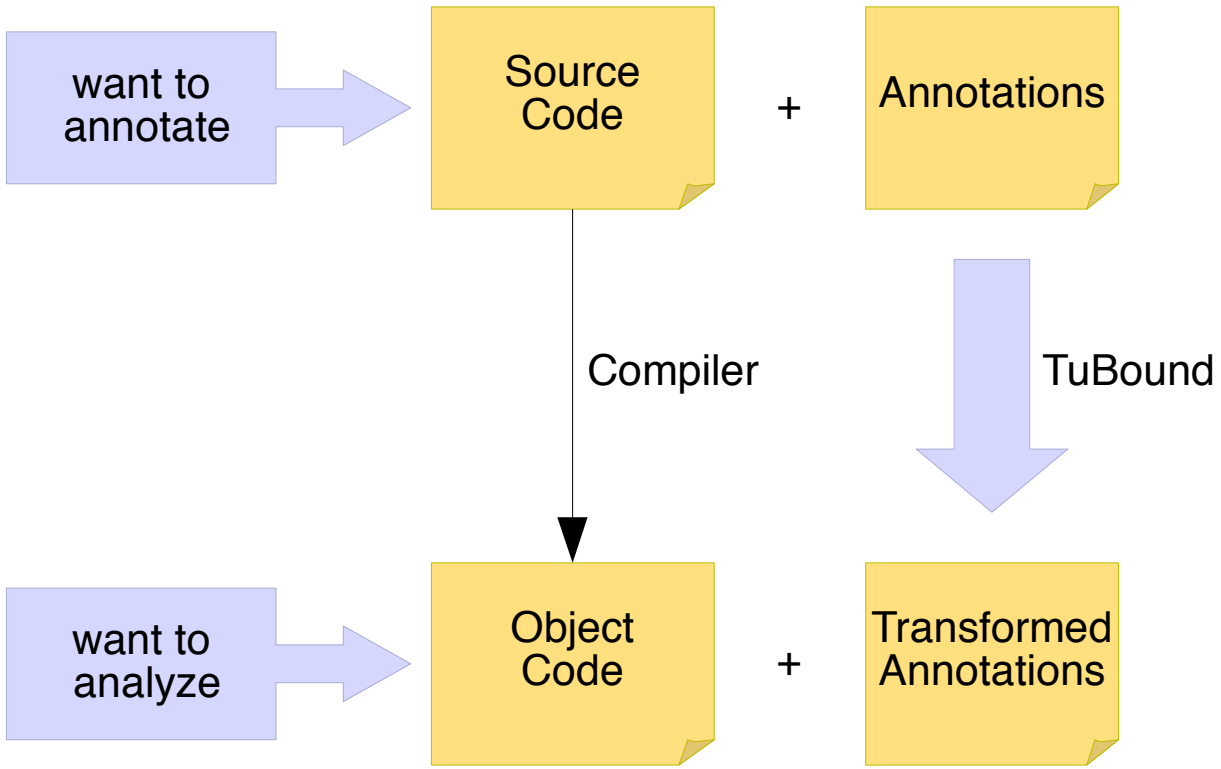
Depends on manual annotations

- Results from static analysis may not be sufficient
- User may have additional *domain-specific knowledge*

WCET analysis

Must be performed on the final low-level machine code

How to bring both worlds together



- Bridge the gap between desired annotation level and analysis level

TuBound

TuBound is a WCET analysis tool developed at TU Vienna.

New Concepts

- Source-based Annotations
- Source-based Optimization
- FlowTrans to transform flow constraints alongside the compiler

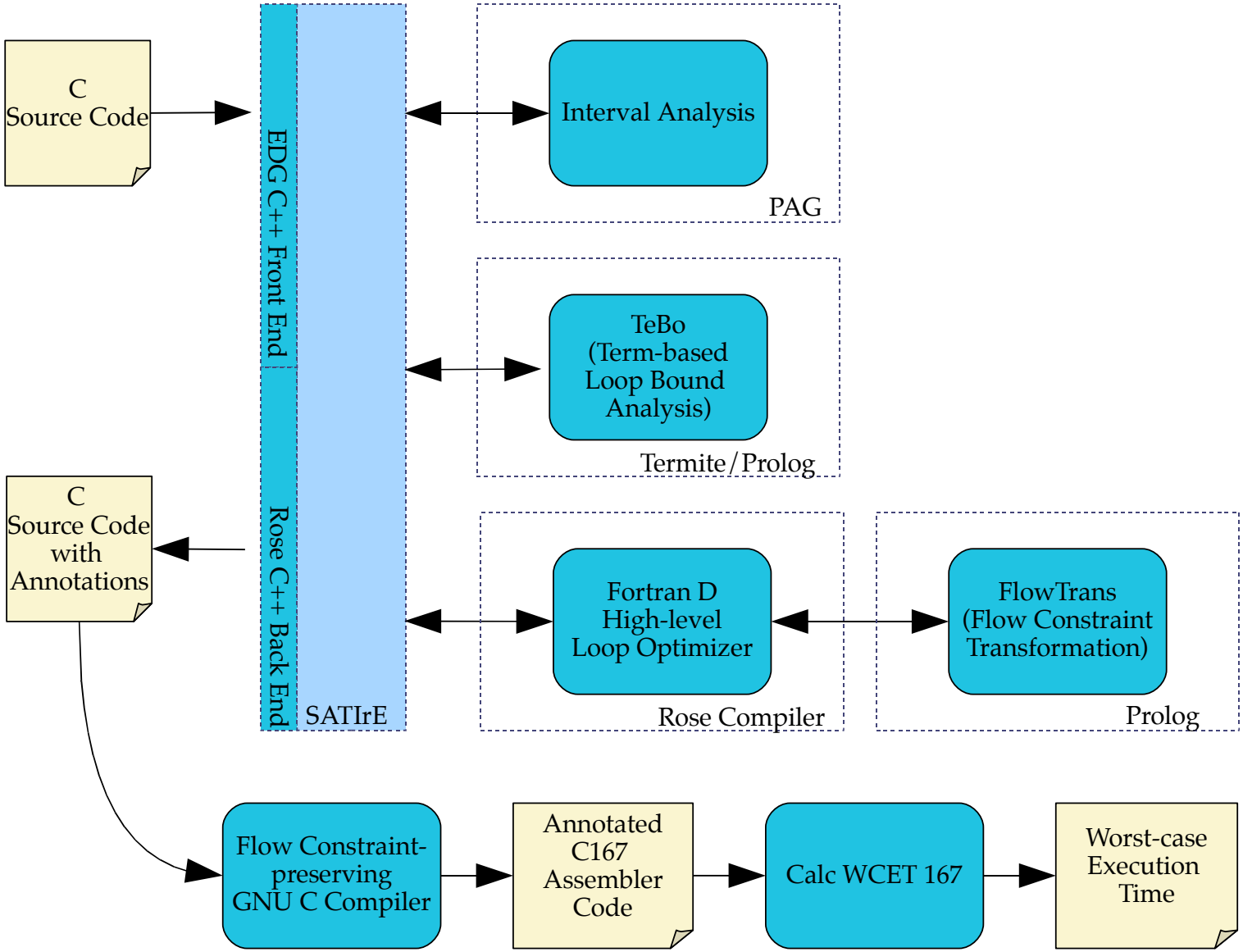
Architecture Overview

Three Phases

Start-up and Annotation

Program Optimization and WCET Annot. Transformation

Compilation and WCET calculation



I. Start-up and Annotation 1/2

1. Parsing

- EDG C++ front end

➥ Output: Rose abstract syntax tree (AST)

2. Interval Analysis

- SATIrE ICFG
- Program analyzer generator (PAG)
- Interprocedural interval analysis

➥ Output: Variable value ranges attached to the AST

I. Start-up and Annotation 2/2

3. Loop Bound Analysis (TeBo)

- Based on the SATIrE term representation of the AST
- Implemented in Prolog
- Solves equation system
- Constraint logic programming

➥ Output: Flow constraint annotations (as `#pragma` nodes) in the AST

II. Program Optimization and WCET Annotation Transformation

1. Fortran D loop optimizer

- Part of the Rose compiler
- Source-to-source loop optimizations

➥ Output: Optimized program and optimization trace

2. FlowTrans

- Flow constraint transformation rules

➥ Output: Annotated, optimized program

III. Compilation and WCET Calculation

1. Compilation to Assembler Code

Flow constraint-aware Compiler:

- Customized GCC 2.7 for the Infineon C167 microcontroller
- Input: annotated C sources
- Preserves the annotations

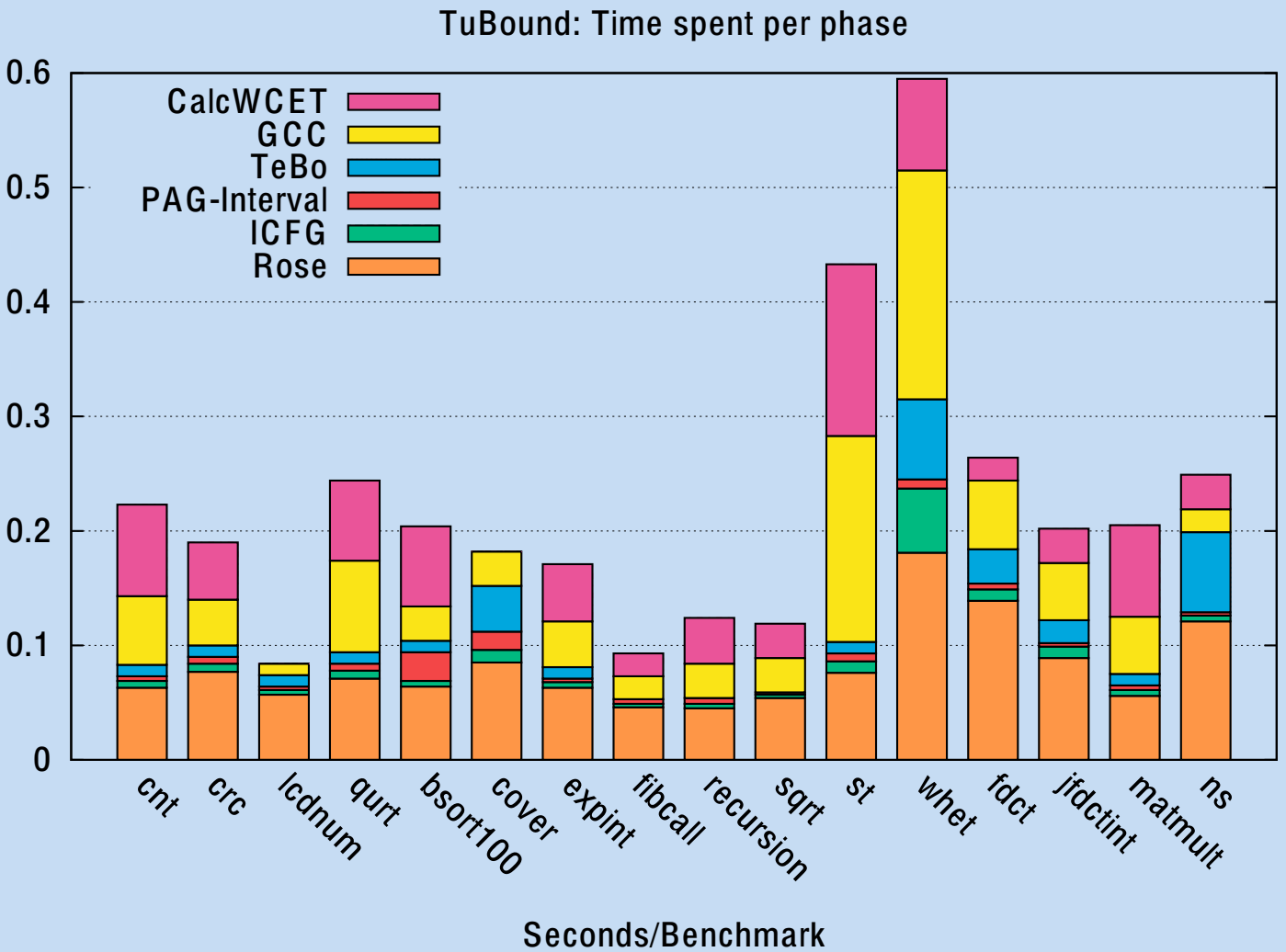
➥ Output: Annotated C167 assembler code

2. WCET Calculation (CalcWCET$_{167}$)

- Instruction table lookup to yield WCET of basic blocks
- Combined equation system:
  - WCET of basic blocks (exec. times)
  - ICFG (connections)
  - Flow constraints (relative frequencies)
- Integer linear programming (ILP) to find maximal solution

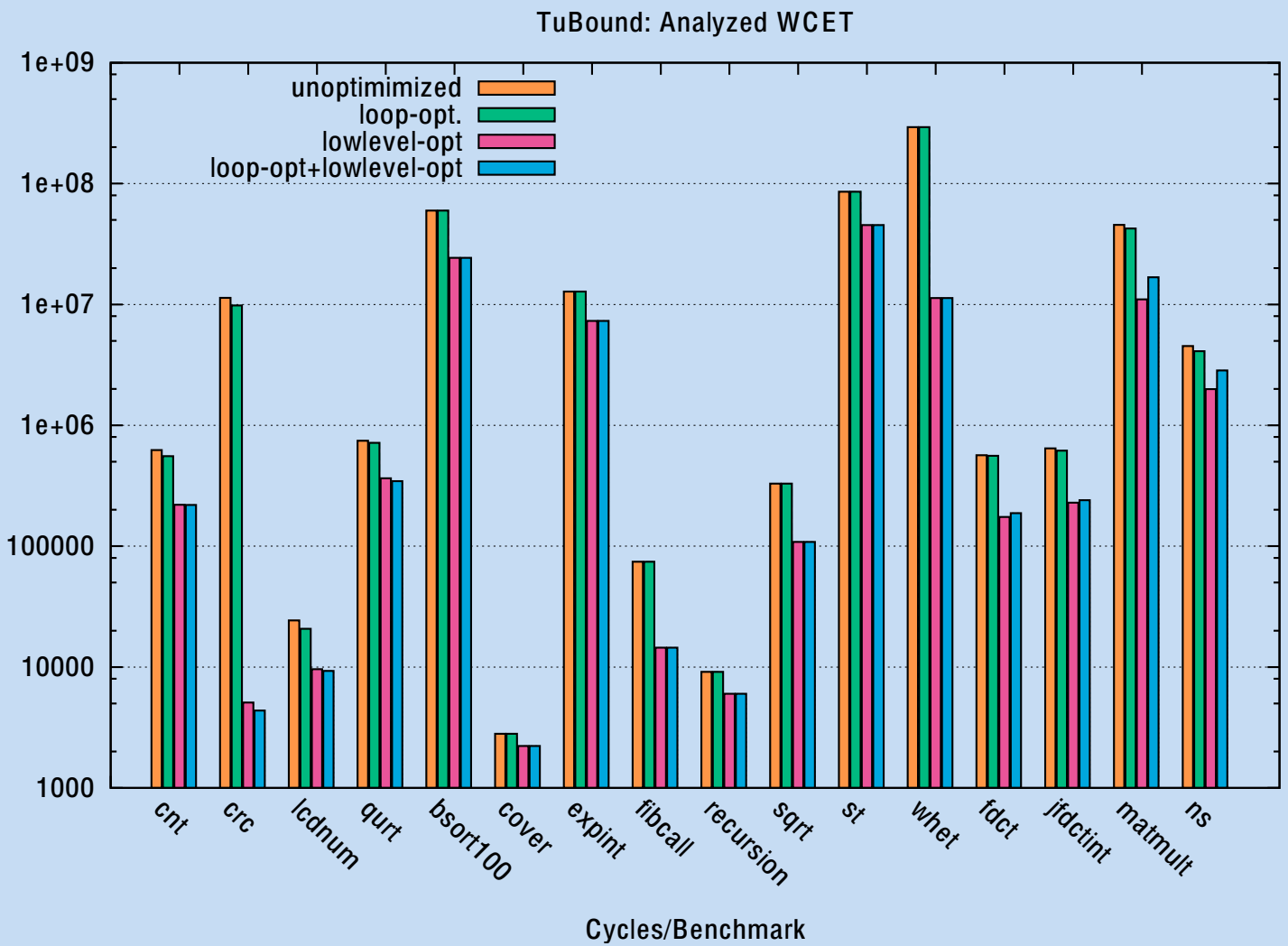
Measurements

Analysis Runtime



Measurements

Automatically calculated WCET



Conclusion

Source-based annotations

- Portable
- Easy to use
- Flexible integration of back ends
- Allow for perpetual refinement

Measurements

- Acceptable performance
- Optimizations improve WCET

➥ WCET Tool Challenge 2008

Thank You!

The CoSTA Team

Albrecht Kadlec Adrian Prantl

Jens Knoop Markus Schordan

Raimund Kirner Ingomar Wenzel

TU VIENNA

Example 1/3

Example

Original program

```
1  int sort(int* a, int n) {
2    int i,j;
3    for (i = 0; i < n; i++) {
4      for (j = 0; j < i; j++) {
5        if (a[i] > a[j]) SWAP(a[i],a[j]);
6      }
7    }
8  }
9
10 int main()
11 {
12   int a[100];
13   sort(a, 100);
14 }
```

Example 2/3

Example

After loop bounding

```
1  int sort(int* a, int n) {
2    int i, j;
3    for (i = 0; i < n; i++) {
4  #pragma wcet_loopbound(100)
5      for (j = 0; j < i; j++) {
6  #pragma wcet_loopbound(99)
7        if (a[i] > a[j])
8          SWAP(a[i],a[j]);
9      }
10   }
11   return 0;
12 }
13
14 int main()
15 {
16   int a[100];
17   sort(a, 100);
18 }
```

Example 3/3

Example

After loop constraint analysis

```
1  int sort(int* a, int n) {
2  #pragma wcet_marker(m1)
3    int i, j;
4    for (i = 0; i < n; i++) {
5  #pragma wcet_marker(m2)
6  #pragma wcet_loopbound(100)
7  #pragma wcet_constraint(m2=<m1*100)
8      for (j = 0; j < i; j++) {
9  #pragma wcet_marker(m3)
10 #pragma wcet_loopbound(99)
11 #pragma wcet_constraint(m3=<m1*4950)
12       if (a[i] > a[j])
13         SWAP(a[i],a[j]);
14     }
15   }
16   return 0;
17 }
```

```
18
19 int main()
20 {
21   int a[100];
22   sort(a, 100);
23 }
```