WCET-Driven, Code-Size Critical Procedure Cloning

Paul Lokuciejewski, Heiko Falk, Peter Marwedel
Computer Science 12
Dortmund University of Technology
D-44221 Dortmund, Germany

Henrik Theiling
AbsInt Angewandte Informatik
Science Park 1
D-66123 Saarbrücken, Germany

Outline

- **Introduction**
  - Motivating examples
  - Standard Procedure Cloning
    - Problems
- **WCET-Driven Procedure Cloning**
  - Algorithm
  - Experimental Environment
- **Results**
- **Conclusions & Future Work**

Introduction

- **Embedded Systems used as Real-Time Systems**
- **WCET is a key parameter**
  - Crucial for safety-critical systems
  - Required for task scheduling
  - Enables effective design and utilization of hardware
- **Estimation by static WCET analysis**
  - Requires loop iteration counts (flow facts)
- **Common Flow Fact Format**
  - min/max interval
  - Global minimum/maximum of loop iteration counts

WCET Overestimation

```
int f(int n) { …
  for(i=0; i<n;++i) {…}}
```

- **Structure of typical ES applications**
  - Functions invoked with different constant arguments
  - Parameter-dependent loops
    - Context-dependent loop execution
- **These loops cannot be precisely analyzed**
  - Analysis must be conservative to guarantee safeness
    - Assume max. iteration counts for each loop execution
  - Results in safe but overestimated WCET
- **Exploit Procedure Cloning**

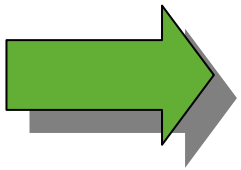
Standard Procedure Cloning

- **Well-known compiler optimization**
  - Main objective is ACET minimization
- **Creates specialized copies of function**
  - Propagates constant parameters into function body
  - Opportunities for further optimizations
  - Reduced calling overhead
  - Allows control-flow simplifications

Example of Procedure Cloning

```
int f(float *x, int n, int p) {
  // flow fact:
  // [ min(n), max(n) ]
  for (i=0; i<n; ++i) {
    x[i] = p * x[i];
    if(i==10) {...}
  }
  return x[n];
}

int main(void) {
  // multiple calls of f(x,5,2);
  return f(a,5,2); }
```

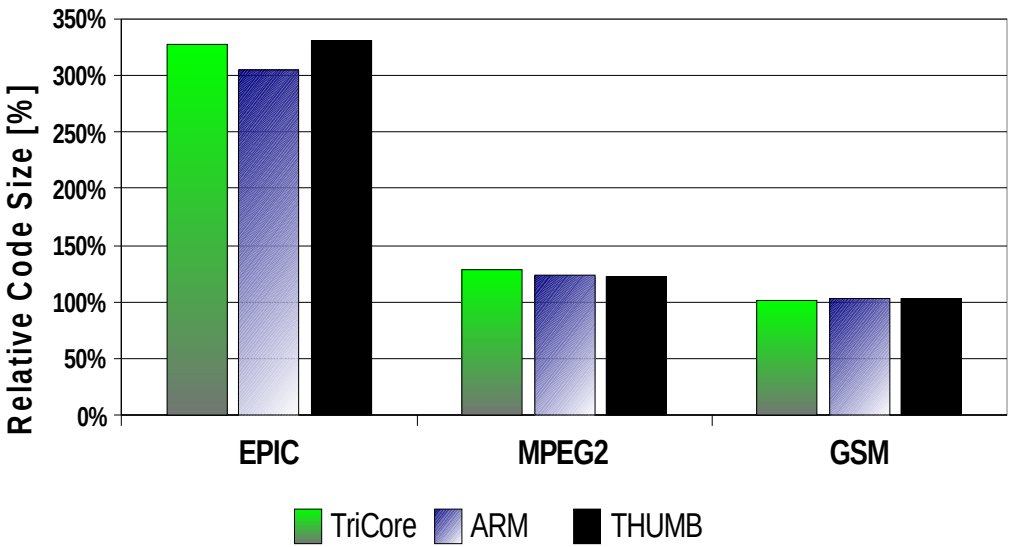


```
int f1(float *x) {
  // flow facts: [ 5, 5 ]
  for (i=0; i<5; ++i) {
    x[i] = 2 * x[i];
    if( i==10) {...}
  }
  return x[5];
}

int main(void) {
  // multiple calls of f1(x);
  return f1(a); }
```

Procedure Cloning for WCET minimization

- **Allows a more precise WCET analysis**
  - Makes calling contexts explicit
  - Removal of infeasible paths
- **But**
  - Cloning can result in high code size increase



[**P. Lokuciejewski et al.,** Influence of Procedure Cloning on WCET Prediction, ISSS 2007]

- **Introduction**
  - Motivating examples
  - Standard Procedure Cloning
    - Problems
- **WCET-Driven Procedure Cloning**
  - Algorithm
  - Experimental Environment
- **Results**
- **Conclusions & Future Work**

WCET-Driven Procedure Cloning (1)

- **How can an optimization minimize the WCET?**
  - WCET corresp. to length of longest execution path (WC path)
  - Optimization must operate on the WC path
    - Transformation on other paths irrelevant for WCET
    - Must consider *WC path switching*
- **Standard Cloning not designed for WCET minimization**
  - Optimization is not aware of WC path
  - Function properties (parameter-dependent loops…) leading to a reduced WCET not exploited

✘ **Code size increase without WCET minimization**

WCET-Driven Procedure Cloning (2)

- **Modified Cloning of standard version**
- **Focusing on systematical improvement of the WCET**
- **Avoids heavy code size increase**
  - Suited for memory-restricted embedded systems
- **Our greedy algorithm works in three phases**
- **First Phase: Finding the WC path**
  - Allows effective WCET minimization
  - For original functions with context-independent loop bound specifications

Evaluation and Data Collection

- **Second Step: WCET and Code Size collection**
  - Successive evaluation of functions on WC path
  - Clone functions *iff* its parameter is used
    - Inside a loop statement OR
    - Inside a conditional expression OR
    - As argument in the function's callee
  - After Cloning
    - Adjust flow facts automatically
    - Removal of infeasible paths
  - Perform WCET analysis of modified code
  - Collect relative WCET and code size changes

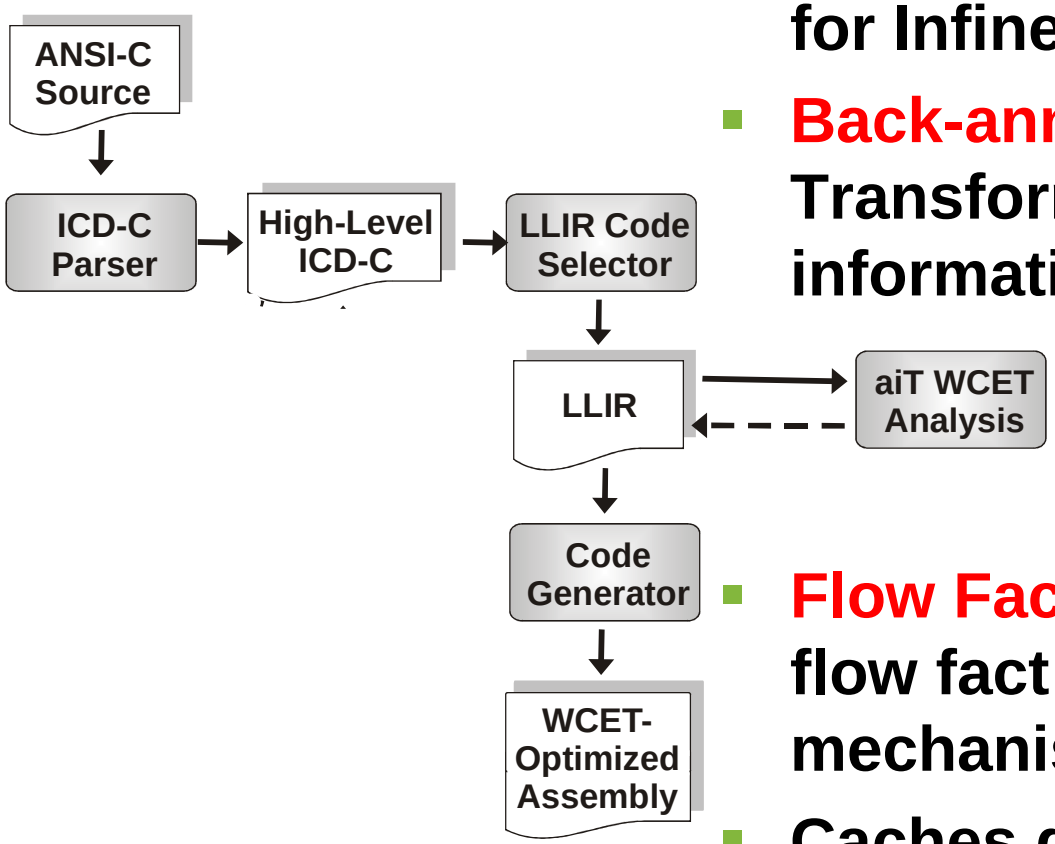
Benefit Calculation

- **Third Step: Determining fittest function**
  - Evaluate collected data by calculating benefit

$$benefit_{function} = \frac{WCET_{original} - WCET_{optimized}}{code\ size_{optimized} - code\ size_{original}}$$

- **Additional parameter monitors code size increase**
- **Find fittest function and apply Procedure Cloning**
- **Update WC path and restart algorithm**

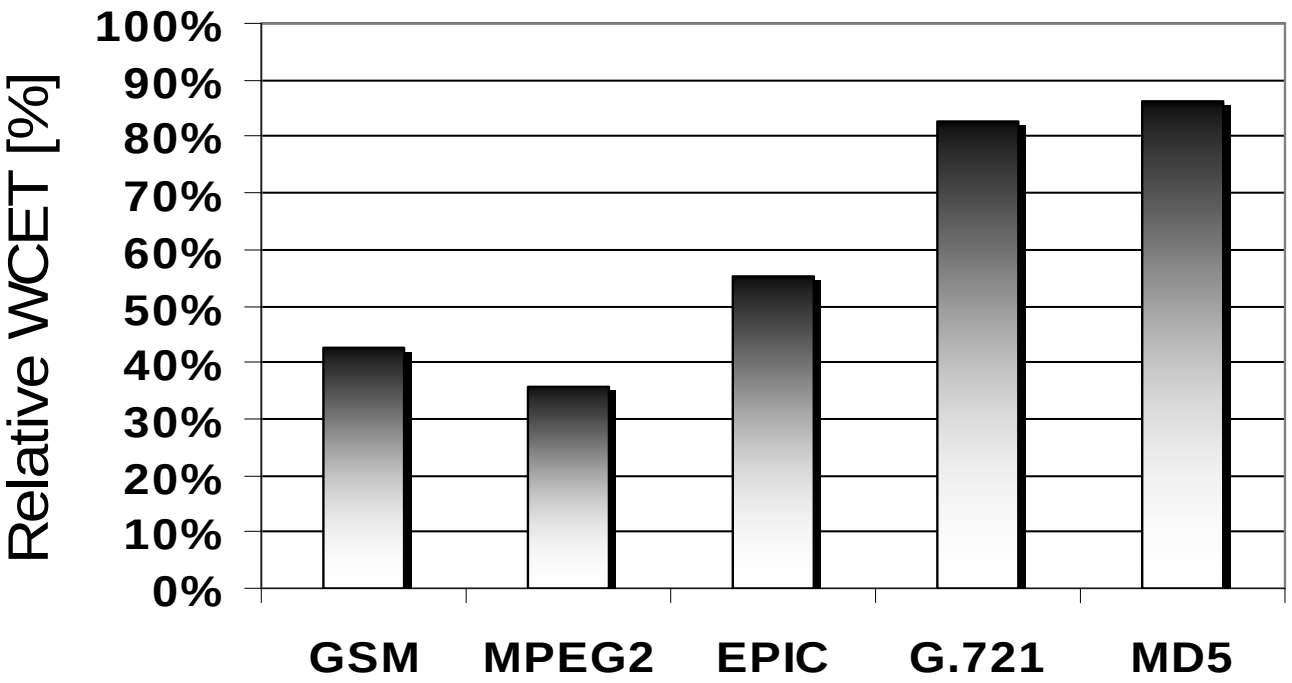
Experimental Environment



- **WCC – WCET-aware C compiler for Infineon TriCore 1796**
- **Back-annotation: Transformation of WCET information to High-Level IR**
- **Flow Fact Manager: High-level flow fact analyses and update mechanisms**
- **Caches disabled**

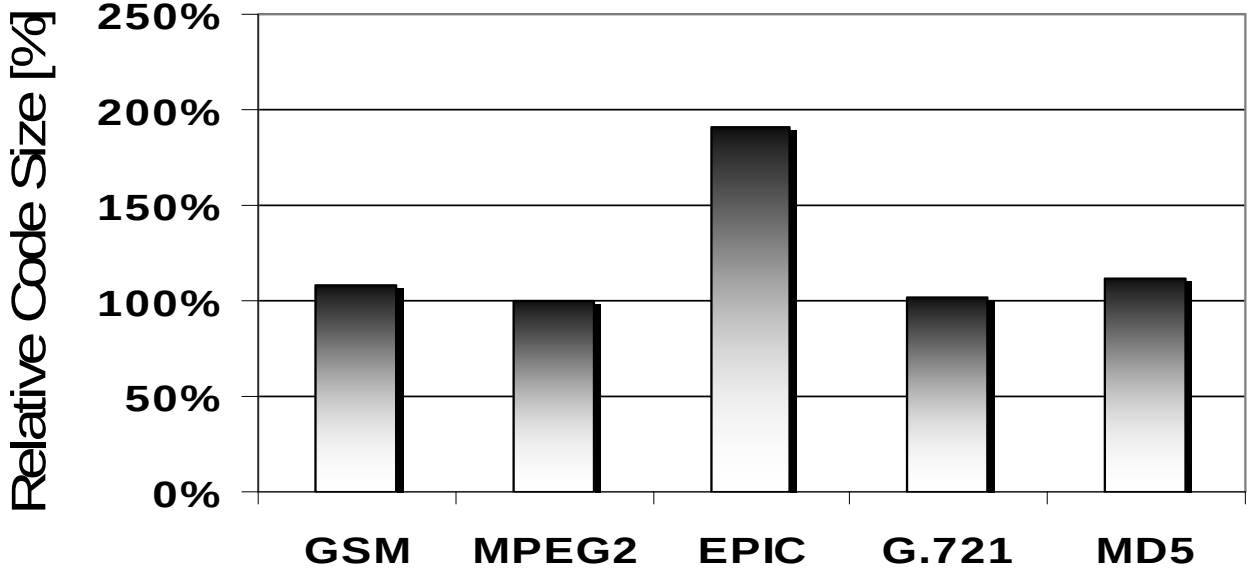
- **Introduction**
  - Motivating examples
  - Standard Procedure Cloning
    - Problems
- **WCET-Driven Procedure Cloning**
  - Algorithm
  - Experimental Environment
- **Results**
- **Conclusions & Future Work**

Results – Relative WCET after Cloning



- **100% correspond to WCET before Procedure Cloning**
- **WCET improvements of up to 64.2%**

Results – Relative Code Size after Cloning



- **100% correspond to Code Size before Procedure Cloning**
- **Code size increase for EPIC reduced from over 300% to less than 200%**
- **Simulated program's run-time decreased by 3% on average**
- **Optimization run-time acceptable (few min. for most benchmarks)**

- **Introduction**
  - Motivating examples
  - Standard Procedure Cloning
    - Problems
- **WCET-Driven Procedure Cloning**
  - Algorithm
  - Experimental Environment
- **Results**
- **Conclusions & Future Work**

Conclusions & Future Work

- **Novel WCET-Driven Procedure Cloning presented**
- **Exclusive cloning of functions which promise improvements of WCET estimation**
- **Better WCET results with less code size increase achieved than for standard Cloning**

- **Future Work**
  - Improve static loop analysis to allow better automatic adjustment of loop bound specifications
  - Advanced placement of clones for better I-cache utilization

Thank you for your attention.