Improving WCET by Optimizing Worst-Case Paths

Wankang Zhao [1],
William Kreahling [1], David Whalley [1],
Christopher Healy [2], Frank Mueller [3]

1. Florida State University
2. Furman University
3. North Carolina State University

Motivation

Benefits gained from reducing Worst-Case Execution Time (WCET).

- More likely to meet timing constraints for embedded applications.
- Allow a developer to use a lower clock rate to reduce power consumption while still meeting the timing constraints.

Outline

- Related work
- Basic idea
- Research framework
- Path optimization techniques
  - superblock formation
  - path duplication
  - loop unrolling
- Experiments
- Conclusions

Related Work

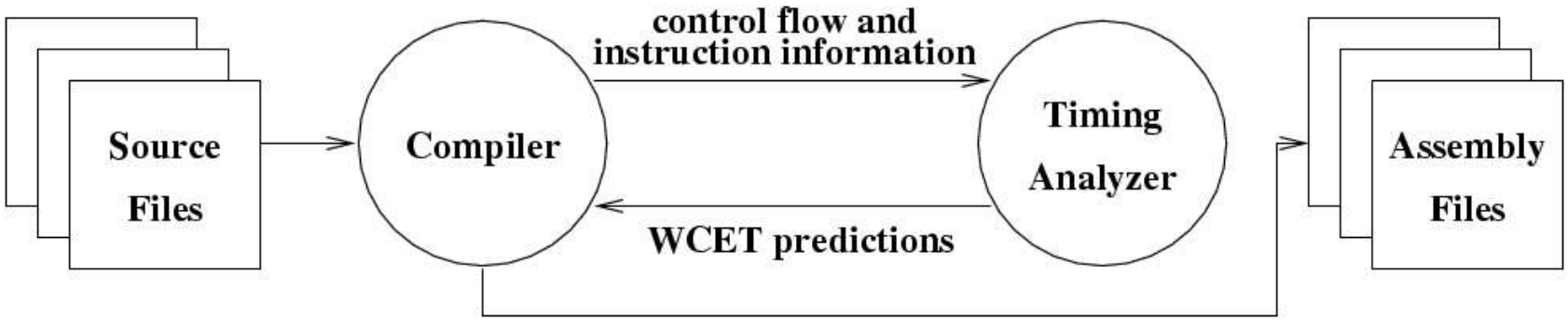
- Methods to reduce WCET in critical sections.
  - Marlowe and Masticola, System Integration '92
  - Hong and Gerber, PLDI '93
- Reduce WCET on a dual instruction set processor.
  - Lee, et al, WCET '03 and SCOPES '04
- Tuning WCET by searching for efficient optimization phase sequences.
  - Zhao et al, RTAS '04
- WCET Code Positioning.
  - Zhao et al, RTSS '04

Basic Idea

- Traditional path optimization uses profiling data to determine the frequent path to optimize.
- Our WCET path optimization uses worst-case path information from a timing analyzer to select the path to optimize.
- WCET path optimizations are more complex than traditional path optimizations since the worst-case (WC) path can change after each optimization.

Research Framework

- We retargeted the VPO compiler and our worst-case timing analyzer to the StarCore SC100 processor.
- The compiler obtains the WC path information to select which paths to optimize and to ensure the WCET improves before committing to a code size increase.

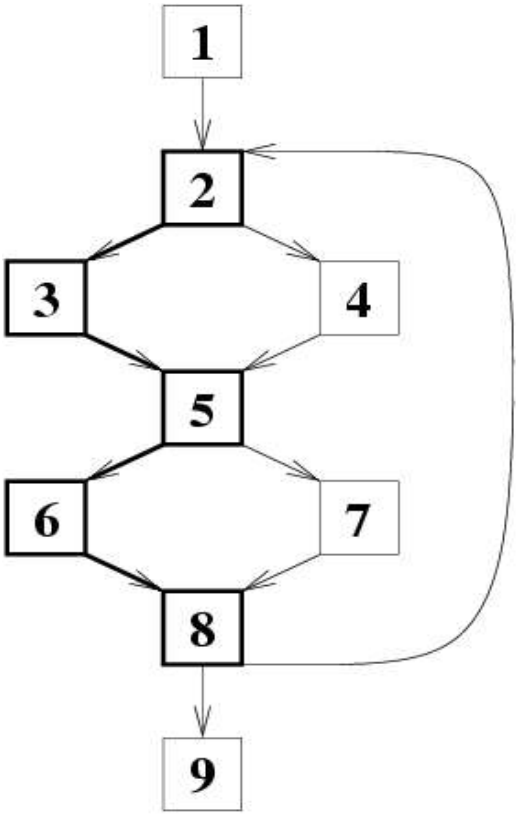


Path Optimization Techniques

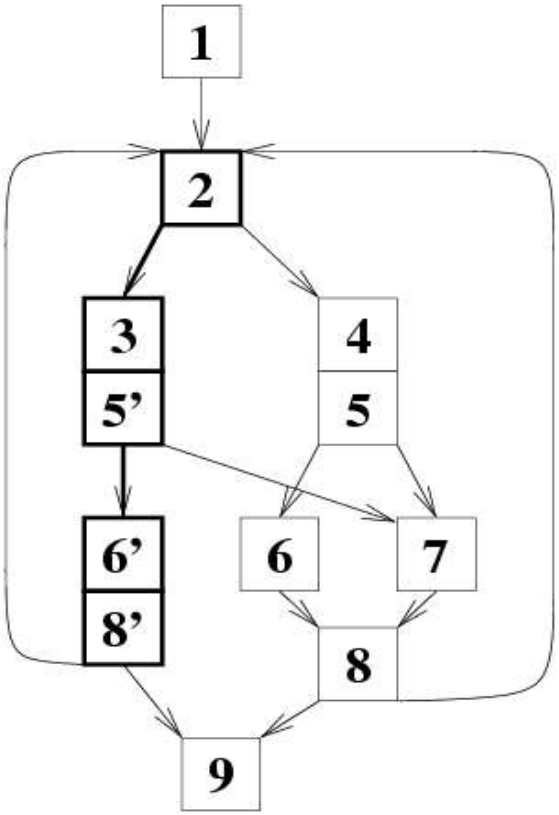
- Superblock Formation
  - Make a superblock along the worst-case path.
- Path Duplication
  - Duplicate the worst-case path.
- Loop Unrolling
  - Unroll the loop by a factor of two to reduce the number of branches of executed and transfer of control stalls.
- Apply other optimizations to exploit fewer joins in the control flow.

Superblock Formation

Creates a path of basic blocks where there is a single entry and possibly more than one exit.



Original Code

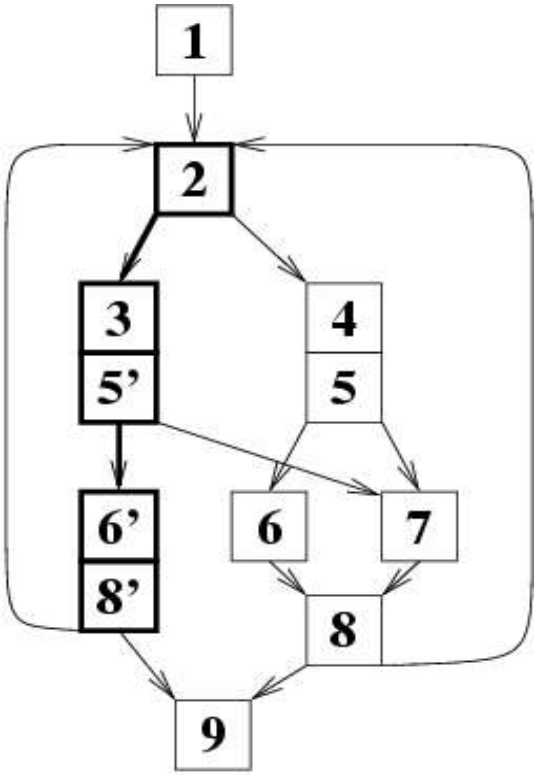


After Superblock Formation

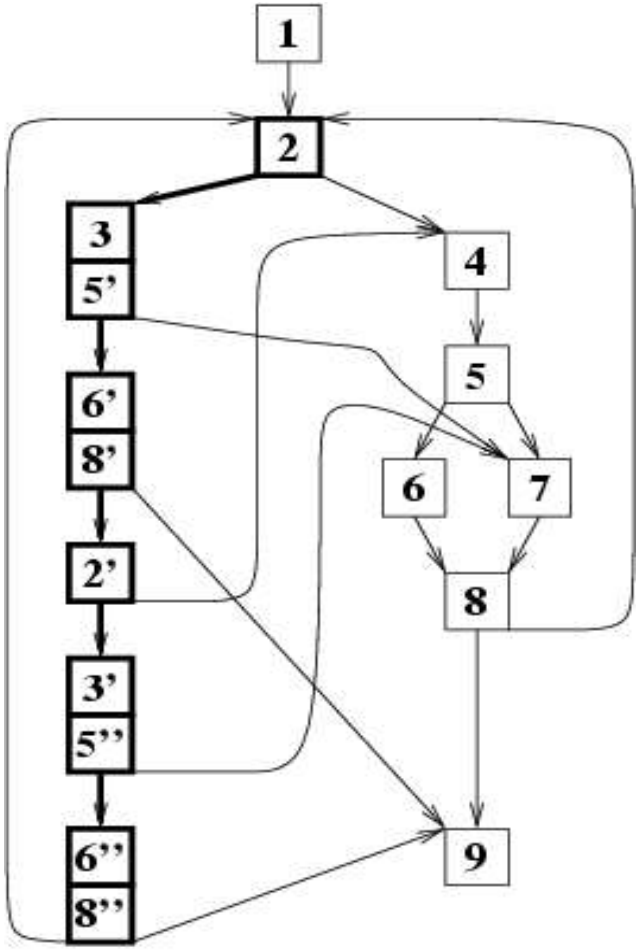
WC Path Duplication

- After superblock formation, duplicate the WC path to further reduce the WCET along that path.
- Superblock formation should be performed before path duplication to eliminate any joins along the WC path.
- Path duplication complicates the timing analysis since some paths represent two original loop iterations and other paths represent one.

WC Path Duplication Example



After Superblock Formation

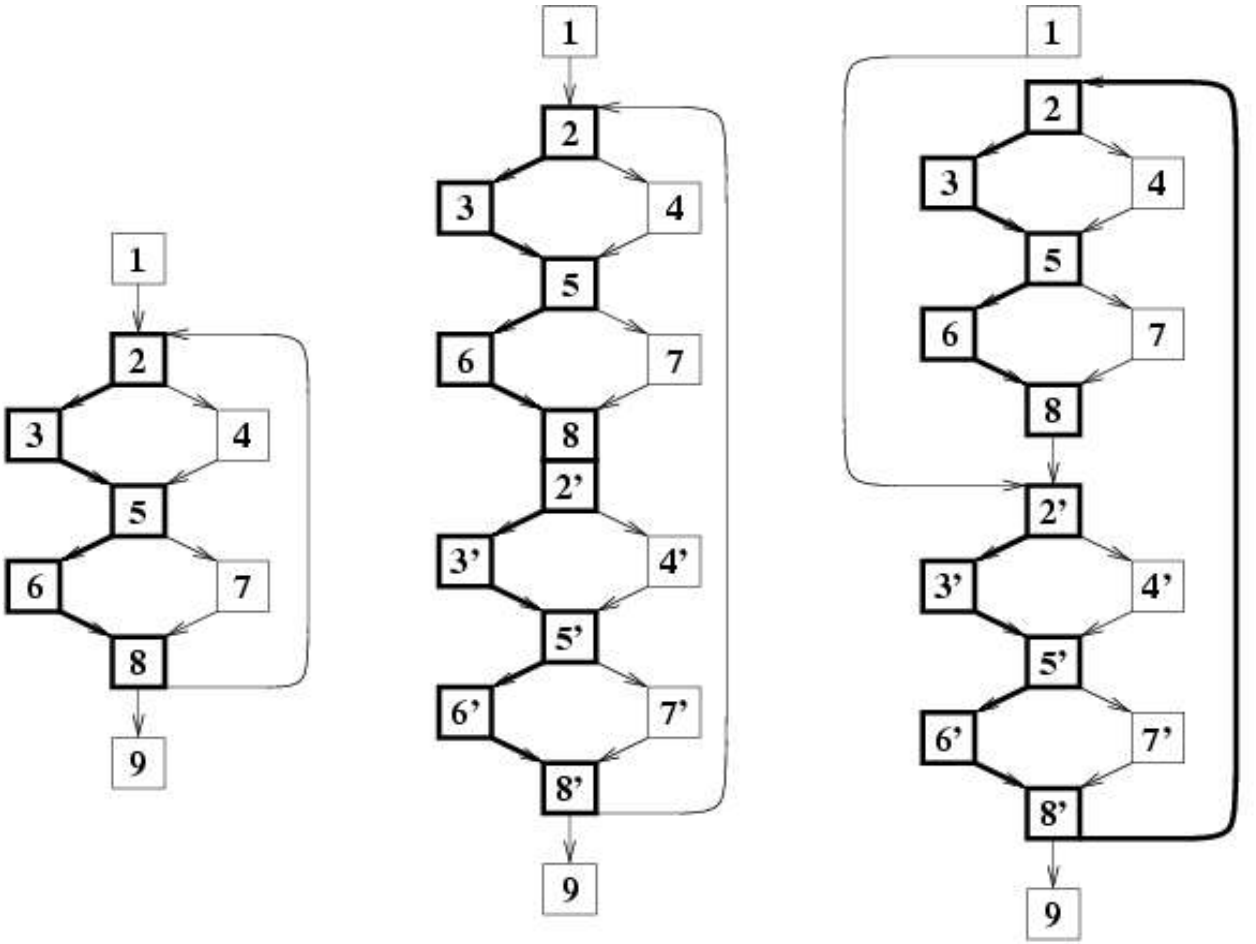


After WC Path Duplication

Loop Unrolling

- Duplicates the loop body to reduce the loop overhead.
- We use a new technique to duplicate the loop body for loops with an odd number of iterations.
- Provides more opportunities for superblock formation and other optimizations.

Loop Unrolling (Cont.)

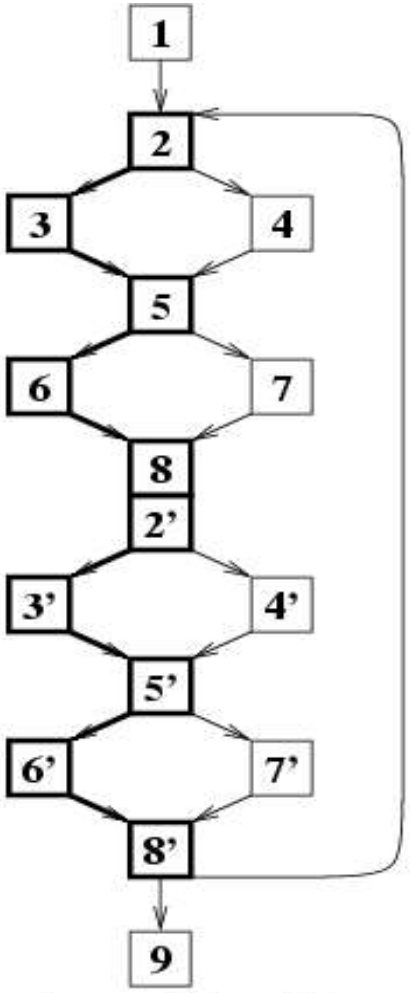


Original Control Flow

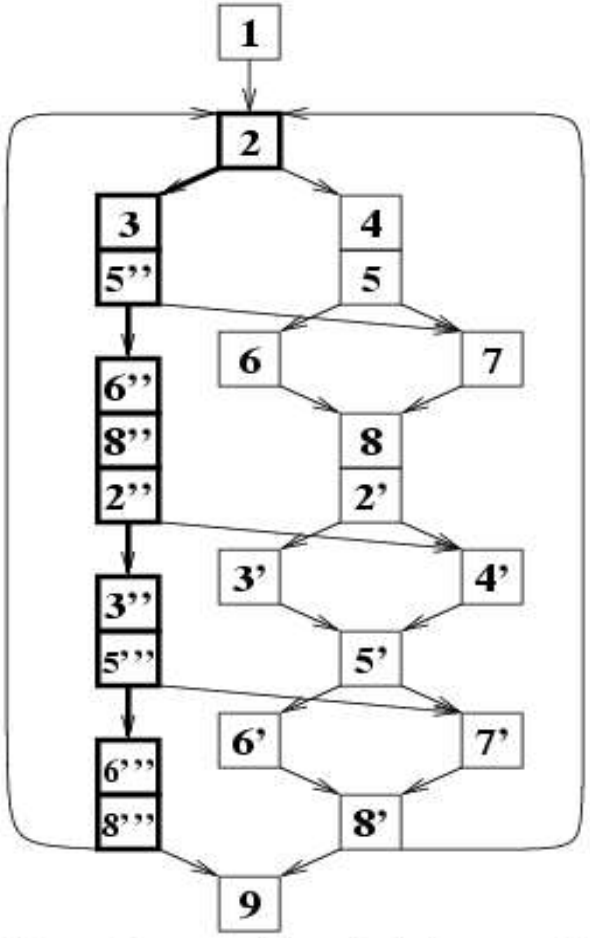
Unrolling for an Even Number of Iterations

Unrolling for an Odd Number of Iterations

Loop Unrolling (Cont.)



After Loop Unrolling



After Superblock Formation

Path Duplication vs. Loop Unrolling

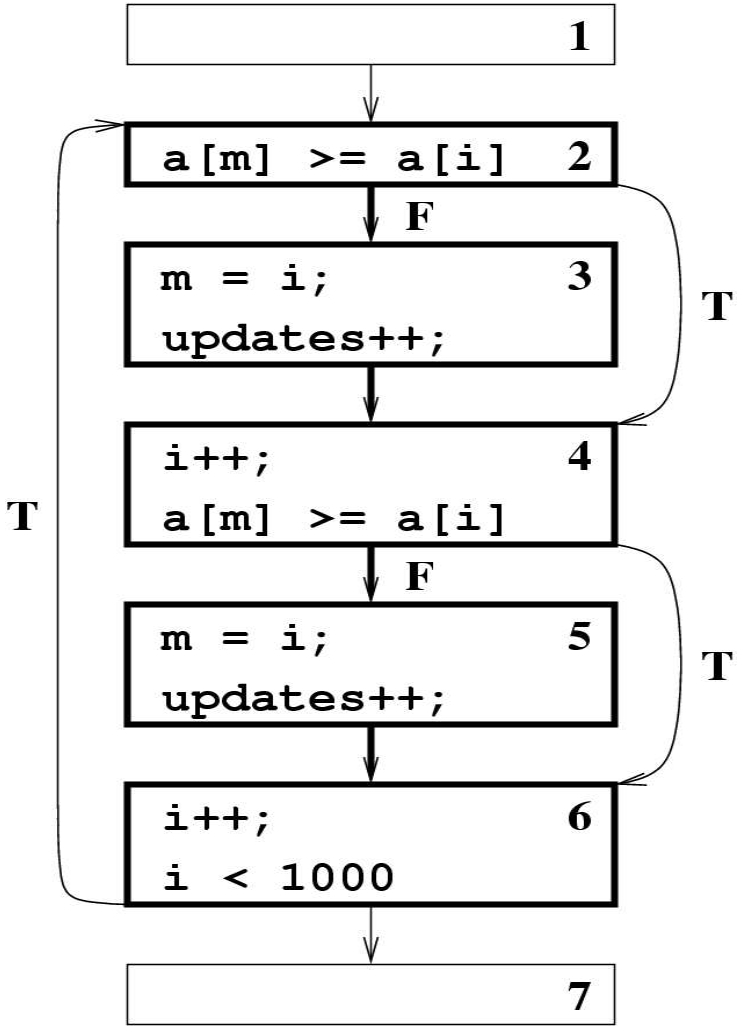
- Path duplication is performed after superblock formation and only duplicates the WC path within the loop.
  - less code size increase
  - smaller decrease in WCET
- Loop unrolling is performed before superblock formation but duplicates the entire loop.
  - greater code size increase
  - greater decrease in WCET

Source Code Example

- Finds index of the maximum value in an array.
- WC path enters the *if* statement, frequent path does not.

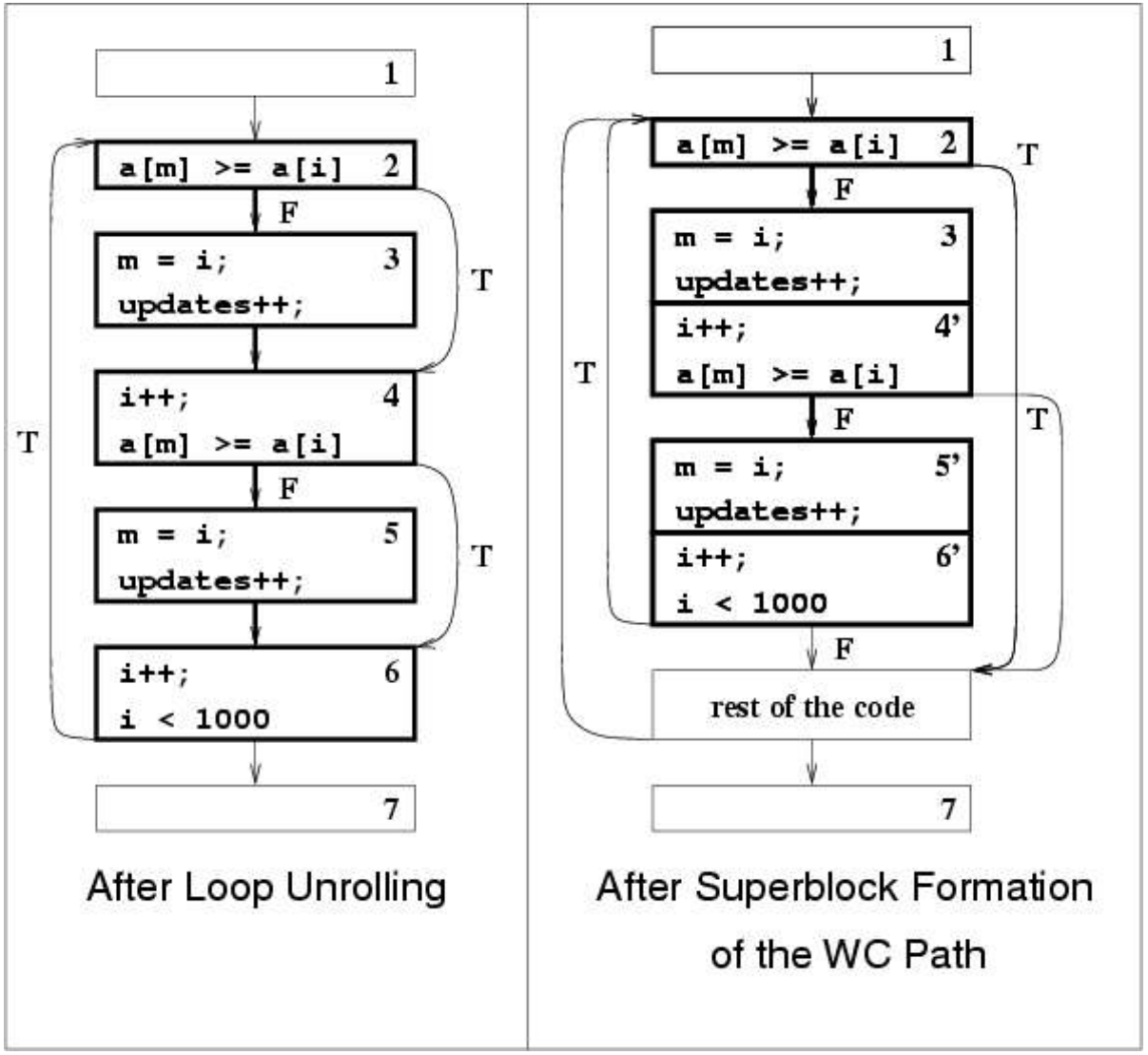
```
m = 0;
updates = 0;
for (i=0; i < 1000; i++)
    if (a[m] < a[i]) {
        m = i;
        updates++;
    }
```

Source Code Example (Cont.)

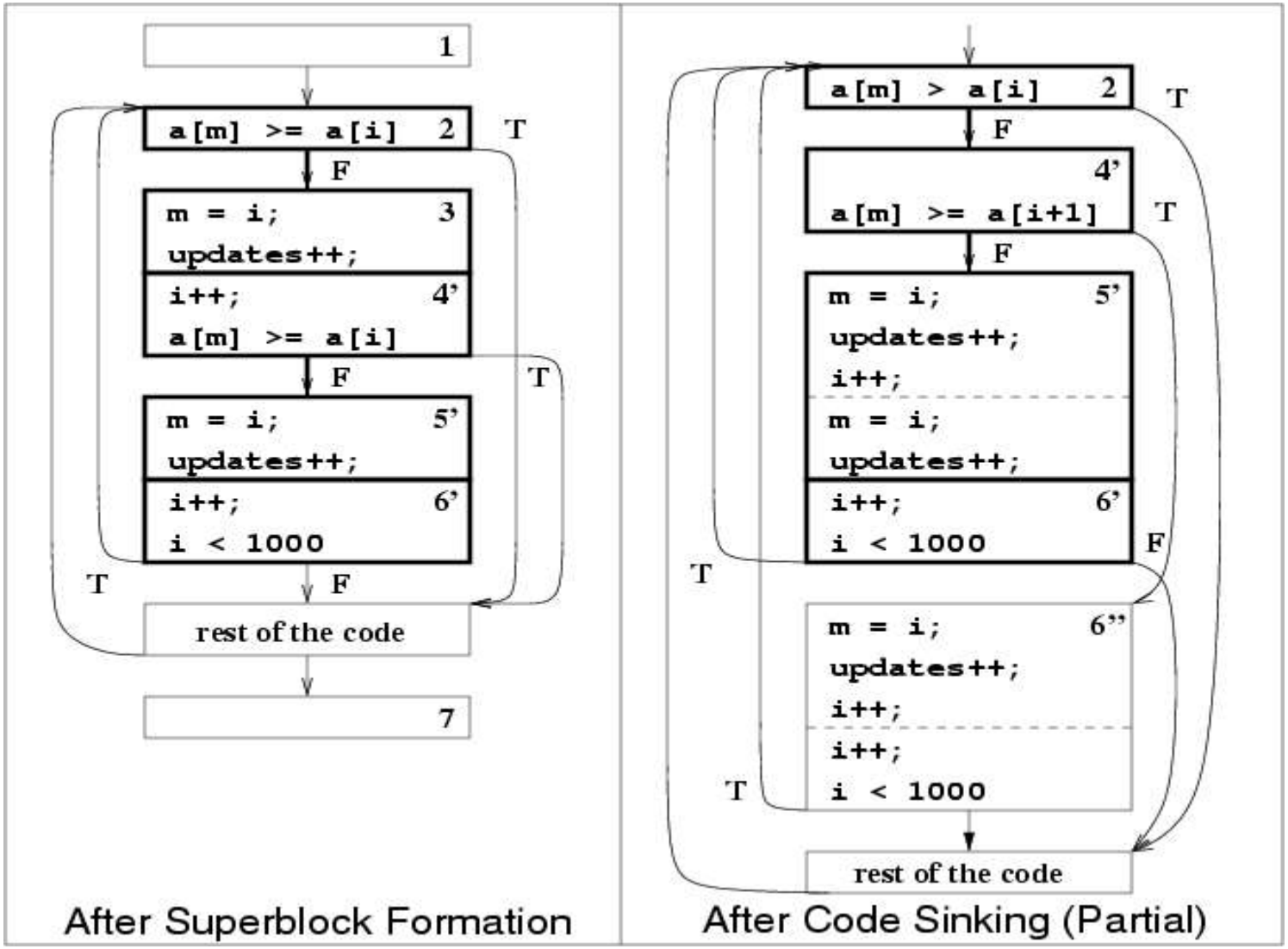


After Loop Unrolling

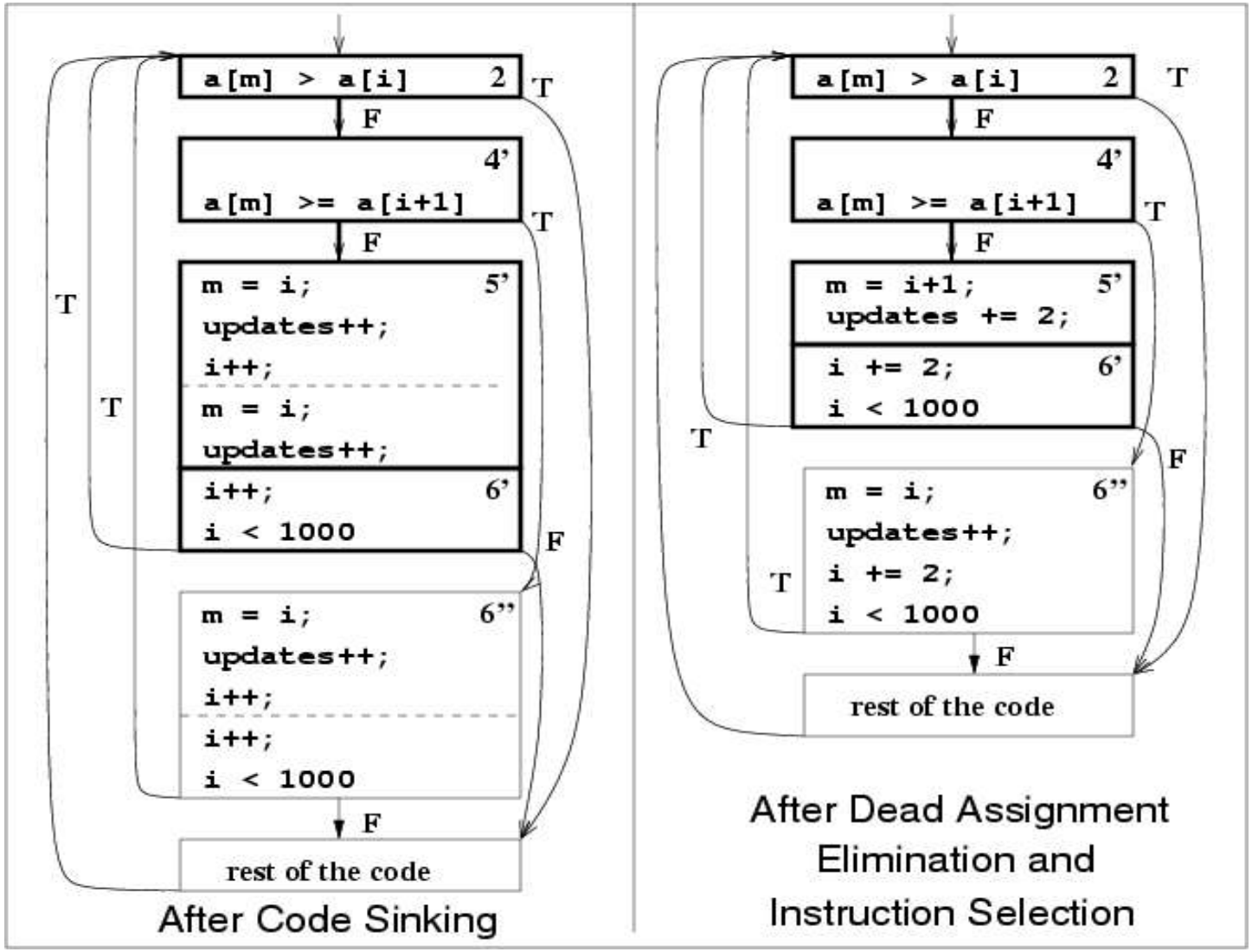
Source Code Example (Cont.)



Source Code Example (Cont.)



Source Code Example (Cont.)

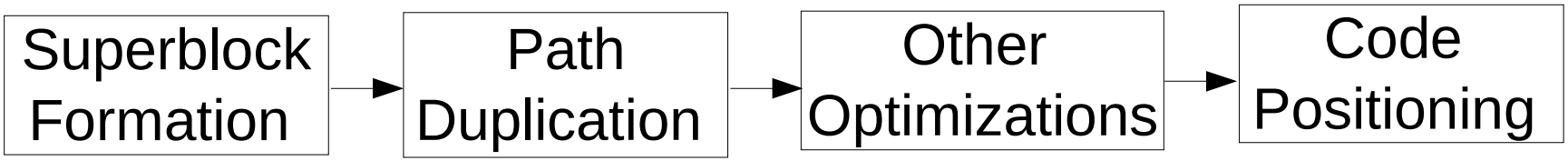


Benchmarks

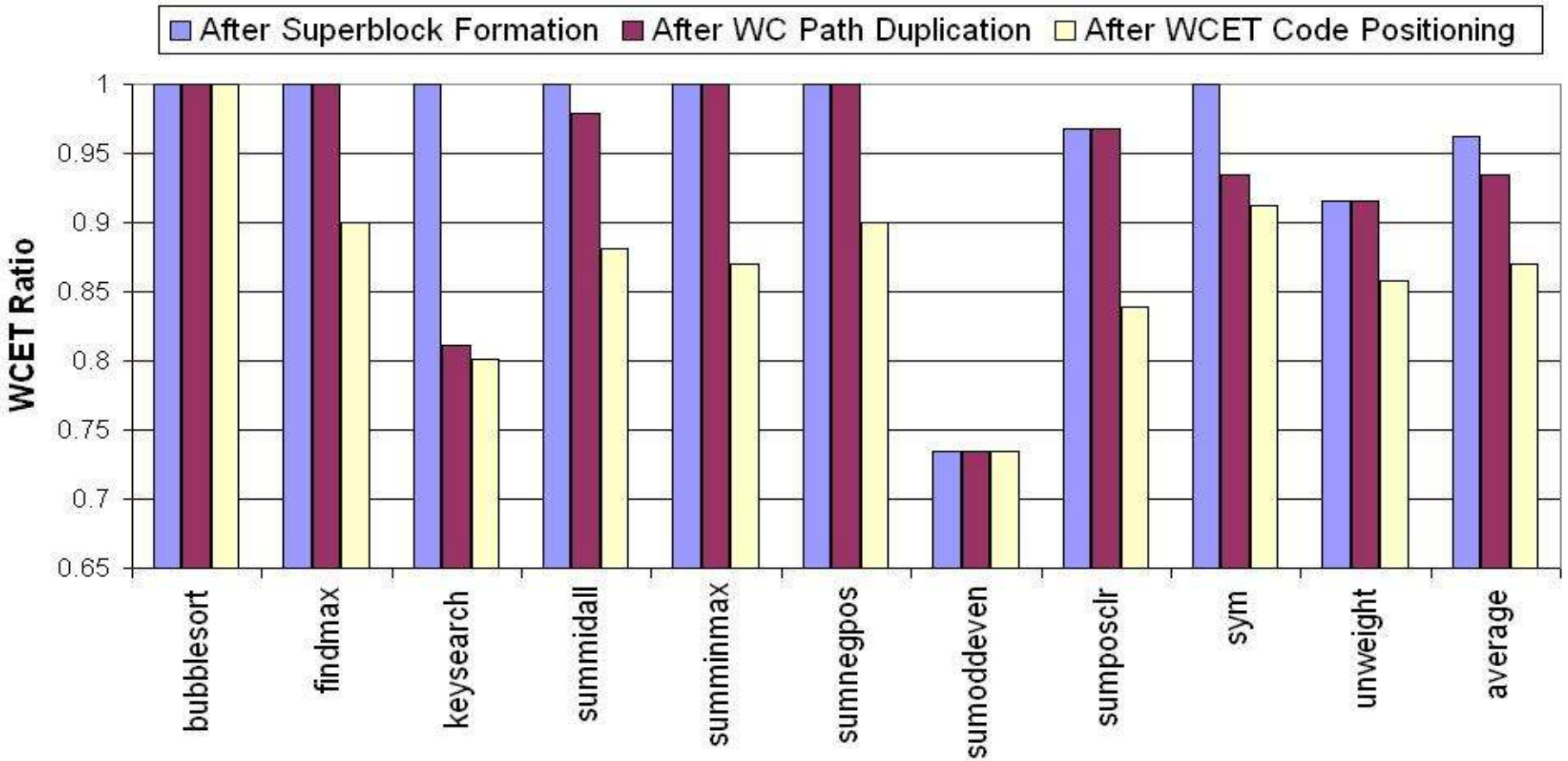
Program	Description
Bubblesort	performs a bubble sort on 500 elements
Findmax	find the maximum elements in a 1000 element array
Keysearch	performs a linear search involving 4 nested loops for 625 elements
Summidall	sums the middle half and all elements of a 1000 integer vector
Summinmax	sums the minimum and maximum of the corresponding elements of two 1000 integer vectors
Sumoddeven	sums the odd and even elements of a 1000 integer vector
Sumnegpos	sums the negative, positive, and all elements of a 1000 integer vector
Sumposclr	sums positive values from two 1000 element arrays and sets negative values to zero
Sym	tests if a 50x50 matrix is symmetric
Unweight	converts an adjacency 100x100 matrix of a weighted graph to an unweighted graph

Experiment 1

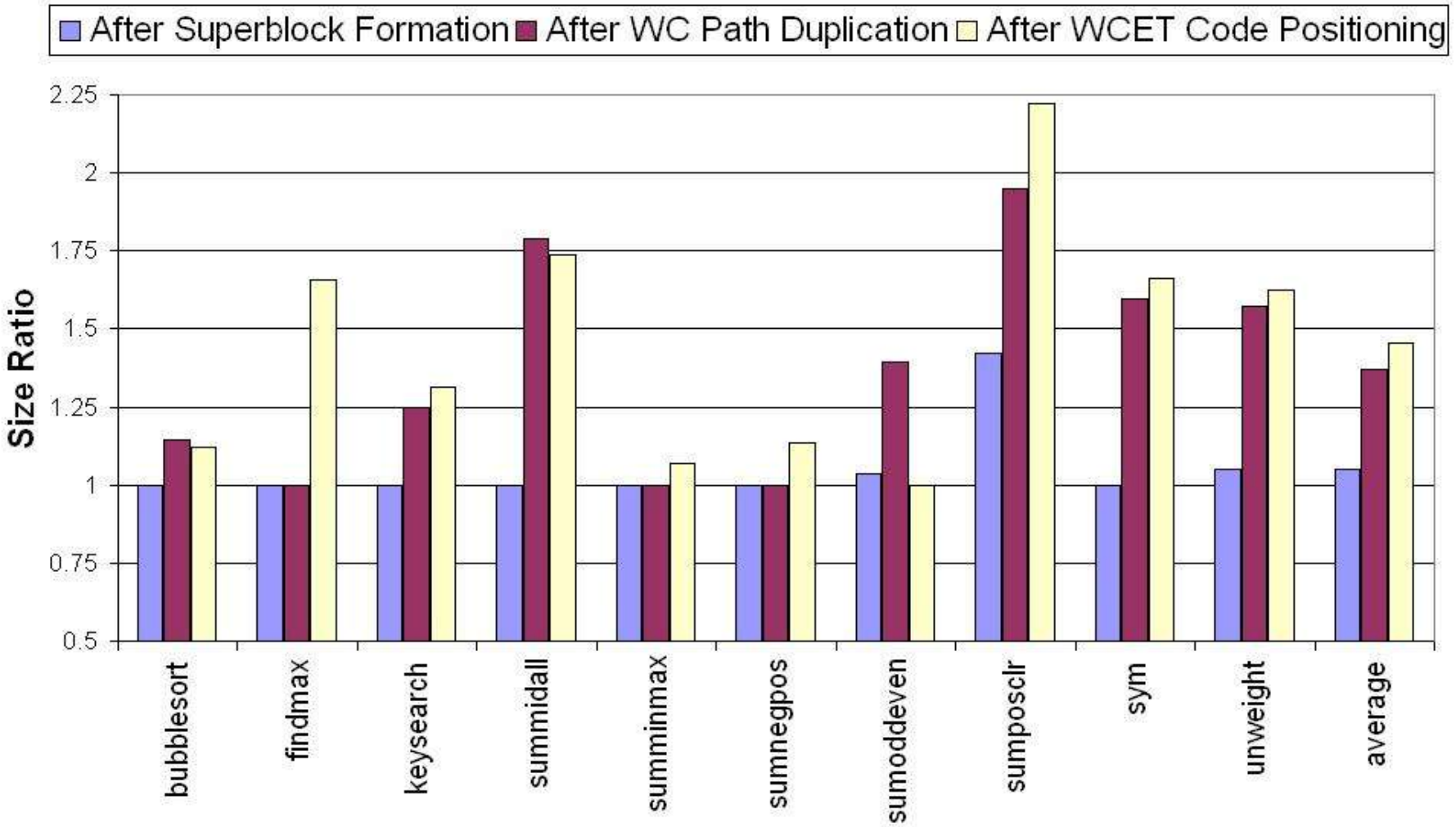
- Apply these optimizations on the WC path in the innermost loops.
- Roll back to a previous state if there is no benefit.



Experimental Results – WCET

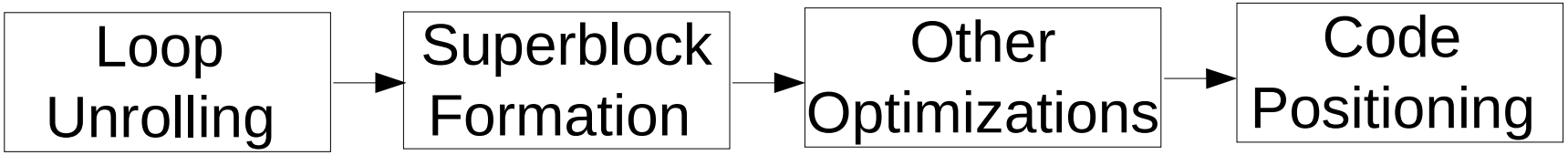


Experimental Results – Code Size

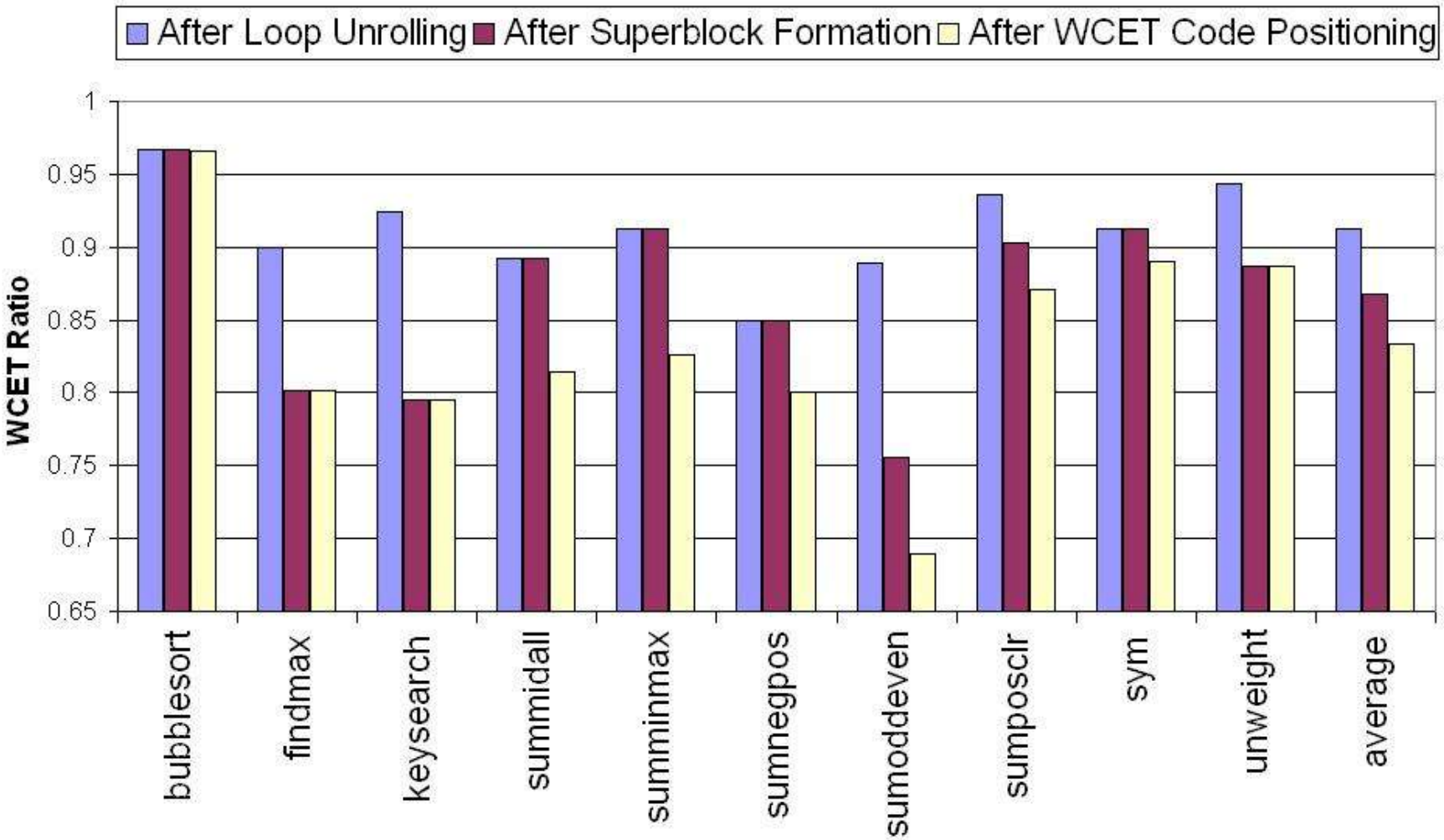


Experiment 2

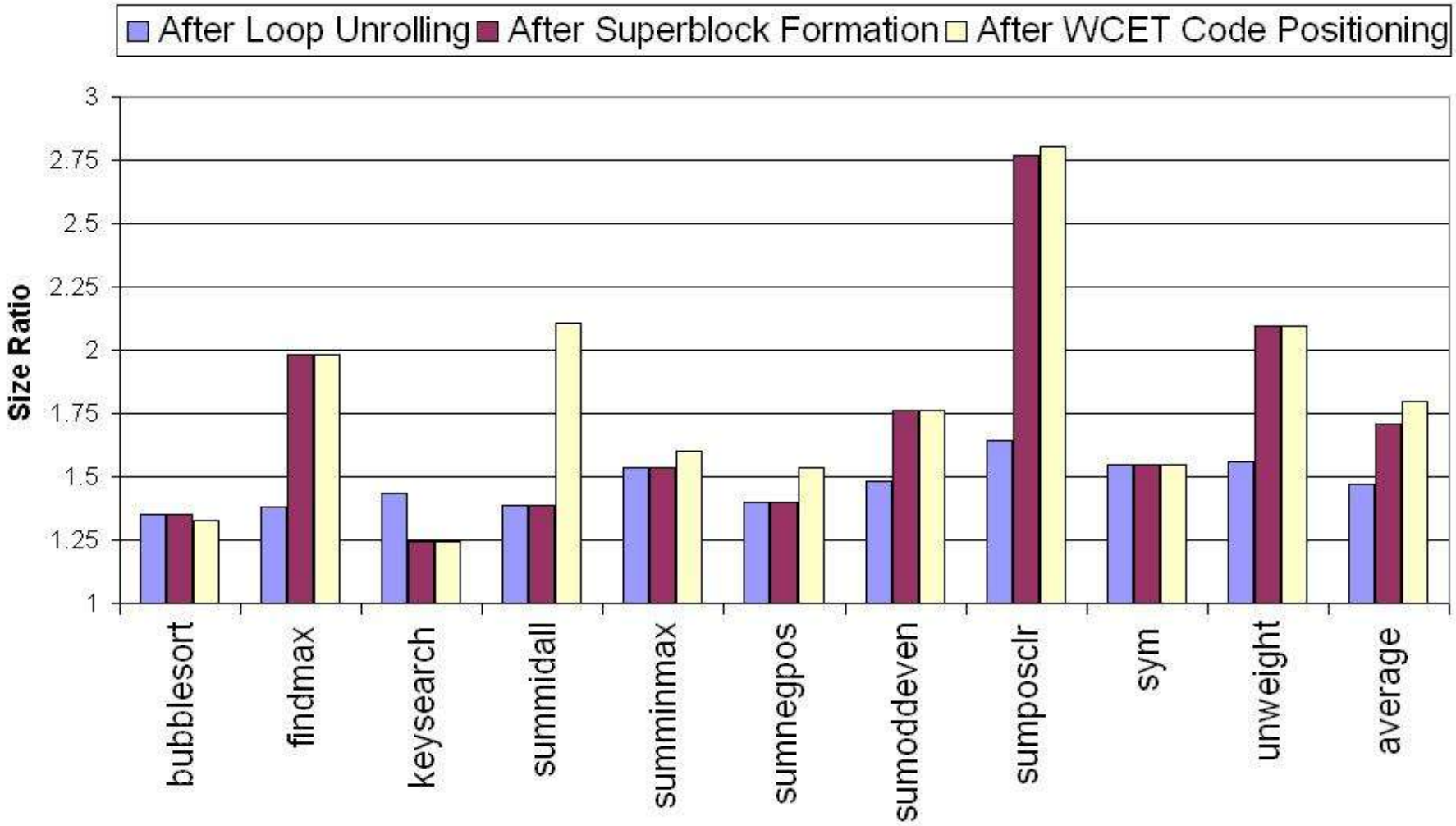
- Apply these optimizations on the innermost loops.
- Roll back to a previous state if there is no benefit.



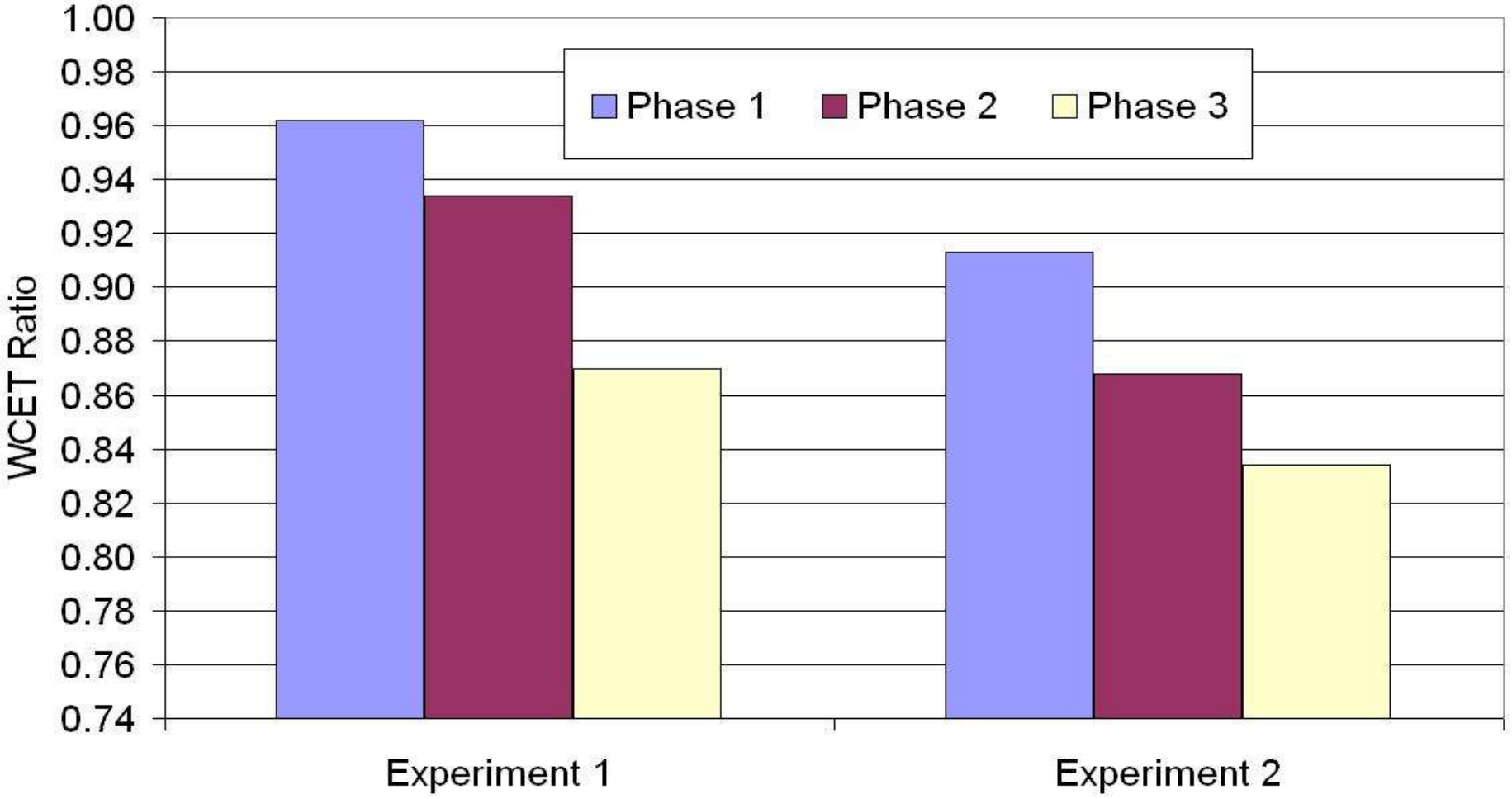
Experimental Results – WCET



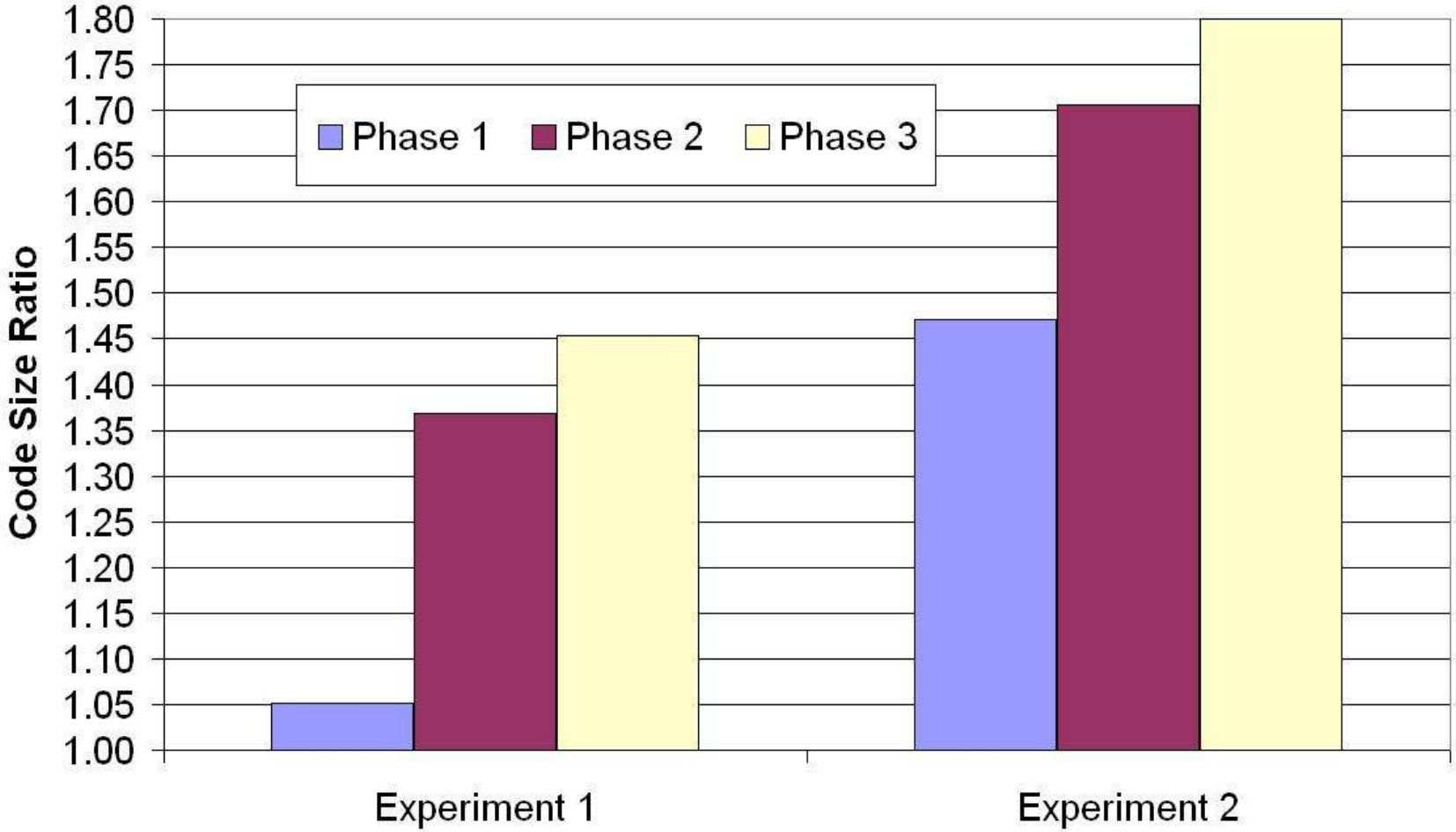
Experimental Results – Code Size



Average Improvement on WCET



Average Improvement on Code Size



Conclusions

- Our compiler uses information from a timing analyzer to automatically:
  - detect the WC paths in a function
  - determine the effect of the WC path optimization on these paths
  - ensure the WCET improves before committing to a code size increase
- Showed that traditional frequent path optimizations can be adapted to reduce WCET.
- Developed new WC path optimizations to improve WCET while attempting to limit code growth.

Any Questions?