Loop Selection for Thread-Level Speculation

Shengyue Wang, Xiaoru Dai, Kiran S. Yellajyosula, Antonia Zhai, Pen-Chung Yew

Department of Computer Science & Engineering
University of Minnesota

Chip Multiprocessors (CMPs)

IBM Power5

- CMPs:
  - IBM Power5
  - Sun Niagara
  - Intel dual-core Xeon
  - AMD dual-core Opteron

Improve program performance with parallel threads

Thread-Level Speculation (TLS)

Automatic parallelization is difficult

- Ambiguous data dependences
- Complex control flow

TLS facilitates automatic parallelization by:

- Executing potentially dependent threads in parallel
- Preserving data dependences via runtime checking

Where do we find speculative parallel threads?

Parallelizing Loops under TLS

Loops are good candidates for parallelism

- Regular structure
- Significant coverage on dynamic execution time

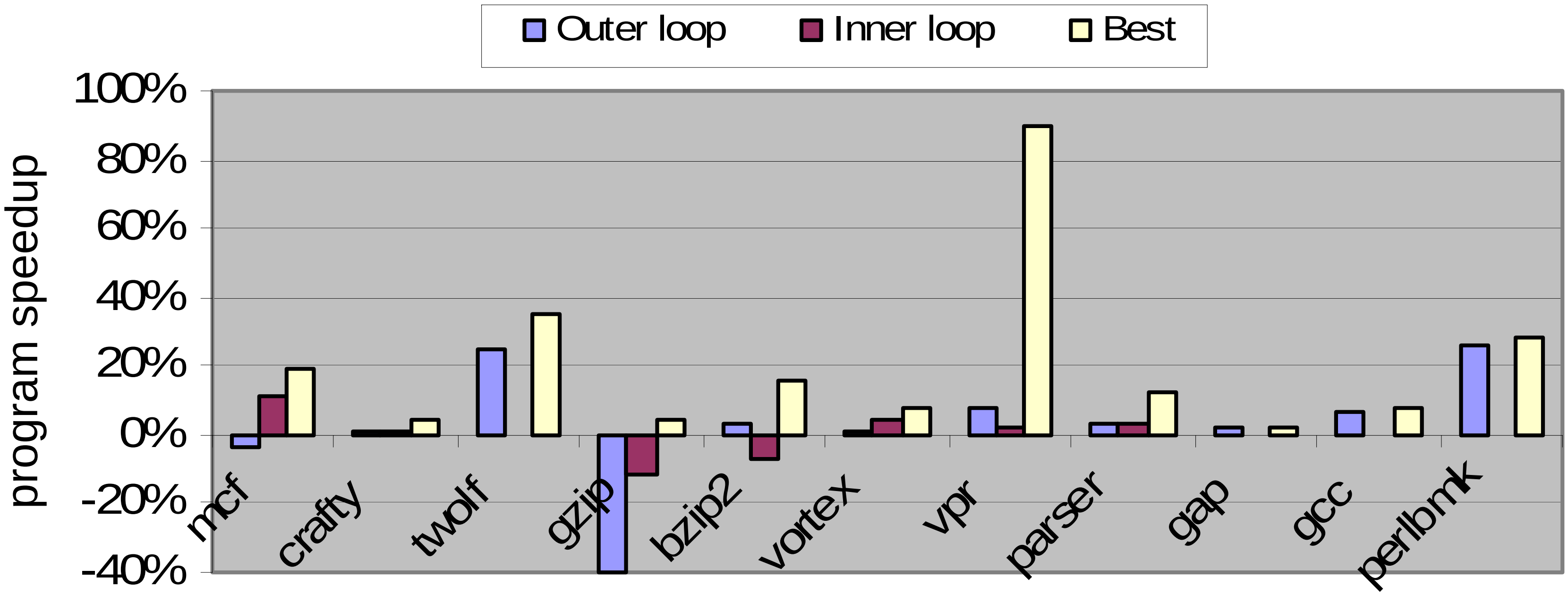
General purpose applications are complicated

Facts about SPECINT 2000

- Average number of loops: 714
- Average dynamic loop nesting: 8

Loop selection: which loops should be parallelized?

Potential of Loop Selection



Carefully selected loops can improve performance significantly!

Outline

- Loop selection
  - Algorithm
  - Parallel performance prediction
- Dynamic loop behavior
- Conclusions

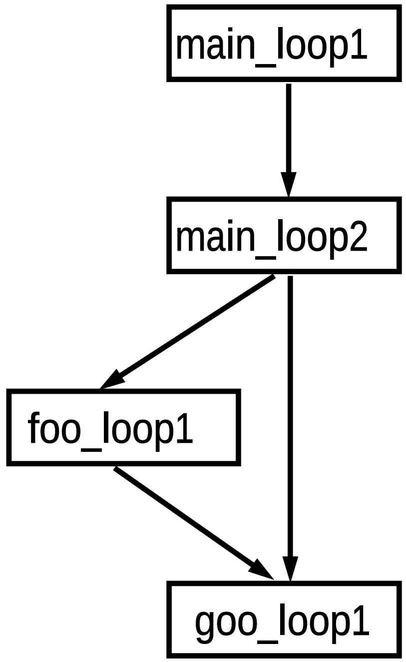
Loop Nesting

```
main( ) {
  while ( condition1 ) {
    while ( condition2 ) {
      foo( );
      goo( );
    }
  }
}
```

```
foo( ) {
  while ( condition3 ) {
    goo( );
  }
}
```

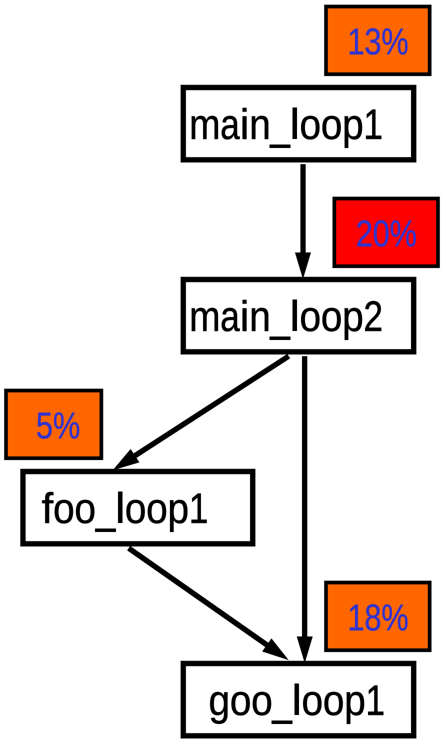
```
goo( ) {
  while ( condition4 ) {
  }
}
```

Source code



Loop graph

Benefit of Parallelizing a Single Loop



Coverage	Loop Speedup	Benefit
80%	1.2	13%
70%	1.4	20%
30%	1.2	5%
50%	1.6	18%

benefit = % program execution time saved
= coverage × (1 – 1 / loop speedup)

Program speedup = 1 / (1 - benefit) = 1.25

Loop Selection: Problem Definition

Goal:

Select the set of loops that maximizes the overall program performance when parallelized

Constraint:

The set cannot contain loops with nesting relationship

Loop selection is **NP-complete**!

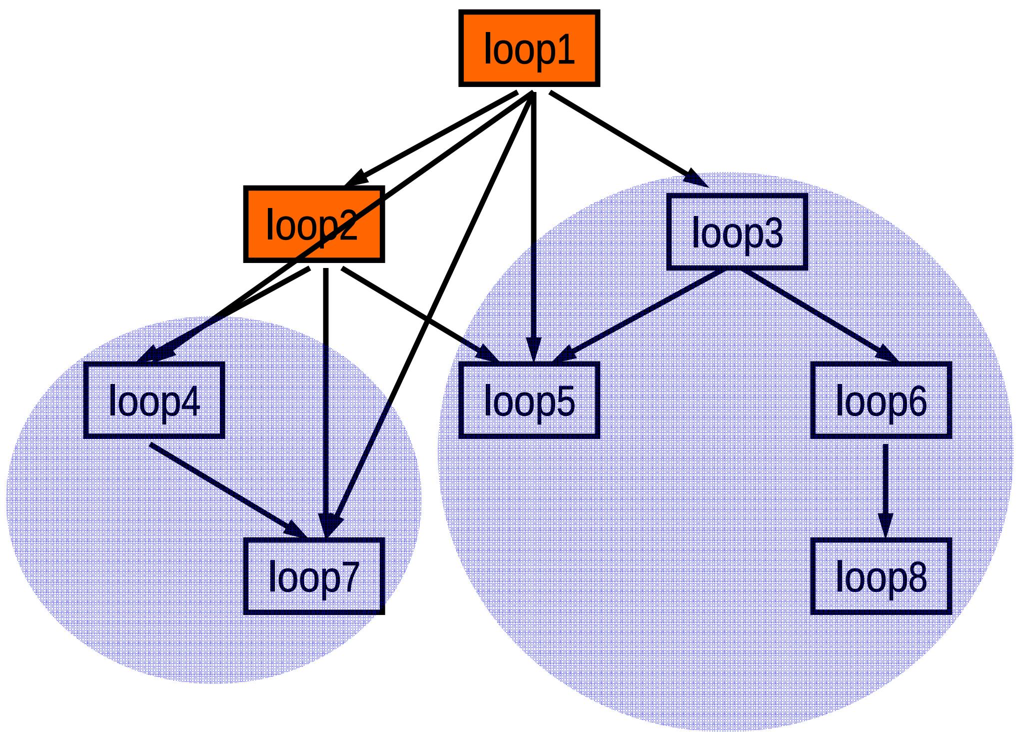
Weighted maximum independent set

Loop Selection: Algorithm

- Exhaustive search ($\leqslant$ 50 nodes)
  - Try all possible combinations of loops
- Greedy algorithm (> 50 nodes)
  - In descending order of benefit
    - Check for nesting relation
    - Add the loop to the set if no nesting

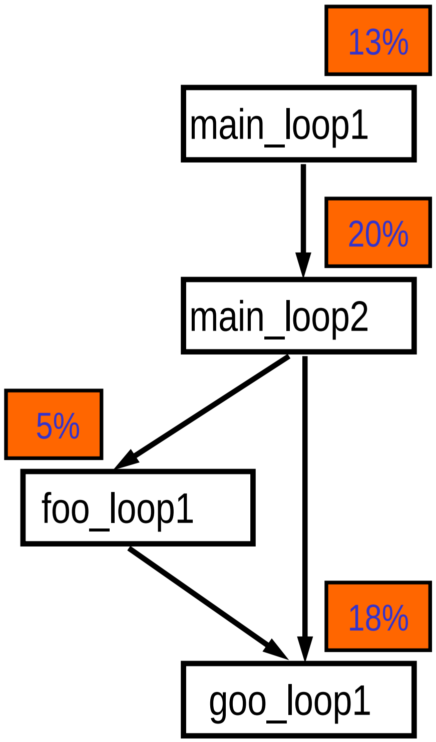
Average number of loops for SPECINT 2000: 714

Loop Pruning



Only resort to greedy algorithm for *gcc* and *parser*

Benefit of Parallelizing a Single Loop



Loop graph

Coverage	Speedup	Benefit
80%	1.2	13%
70%	1.4	20%
30%	1.2	5%
50%	1.6	18%

How can we estimate the speedup?

Outline

- ➢ Loop selection
  - Algorithm
  - ➢ Parallel performance prediction
- Dynamic loop behavior
- Conclusions

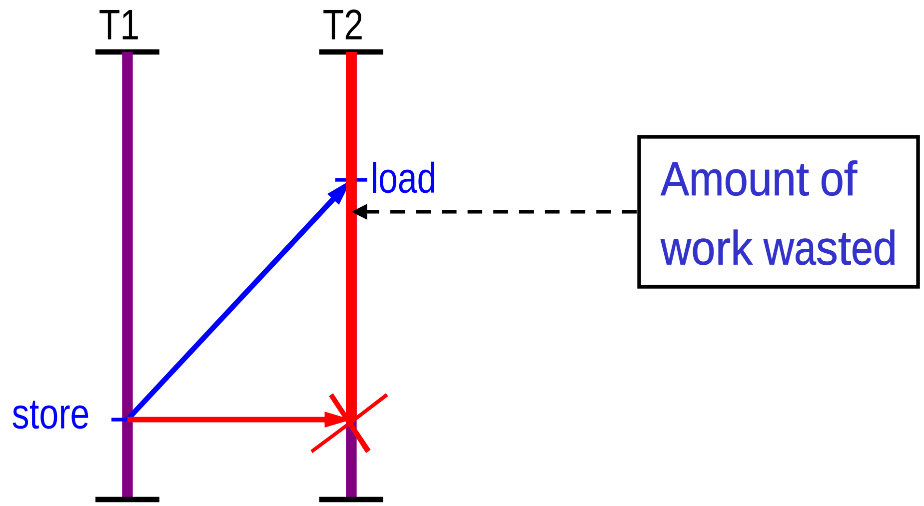
Estimating Parallel Performance

Communicating value between speculative threads adds significant overhead to parallel execution

- Synchronization:
  - Resolves frequently occurring data dependences
- Speculation:
  - Resolves infrequently occurring data dependences

Estimating communication costs with the compiler

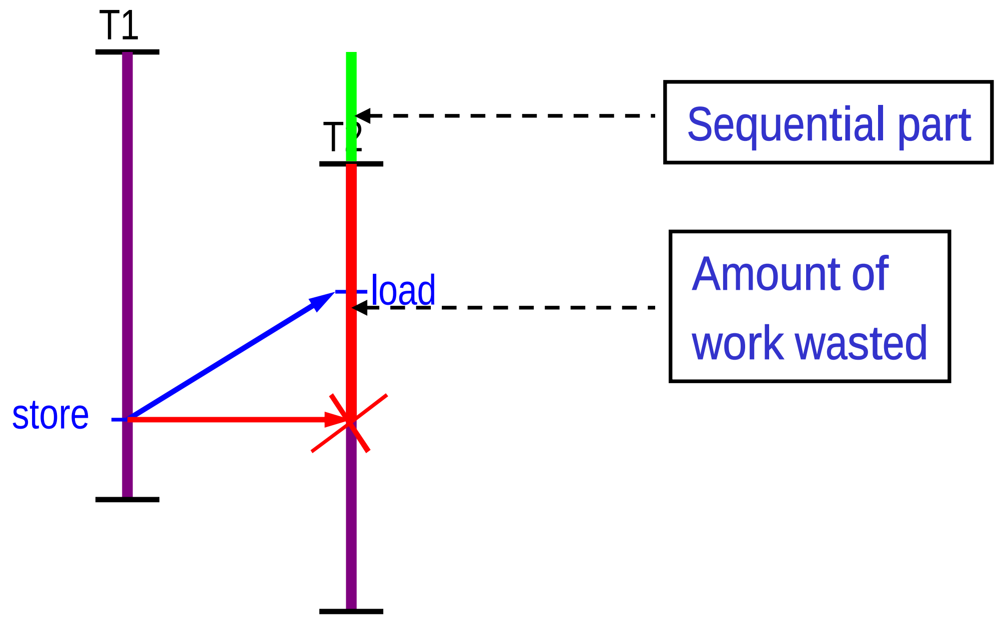
Cost of Mis-speculation



Cost of mis-speculation

= amount of work wasted $\times$ prob. of mis-speculation

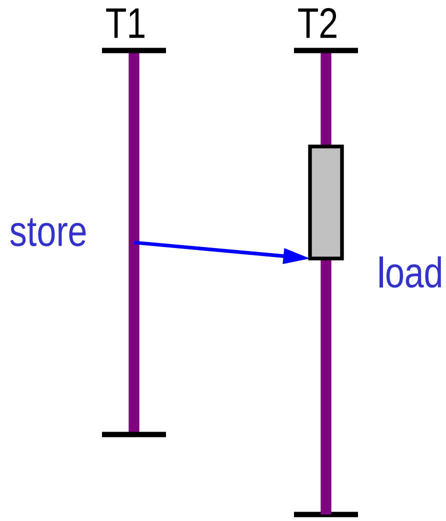
Cost of Mis-speculation



Cost of mis-speculation

= amount of work wasted $\times$ prob. of mis-speculation

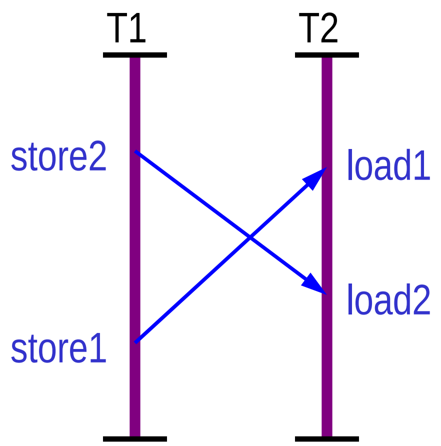
Synchronization



Synchronization serializes parallel execution

Cost of Synchronization

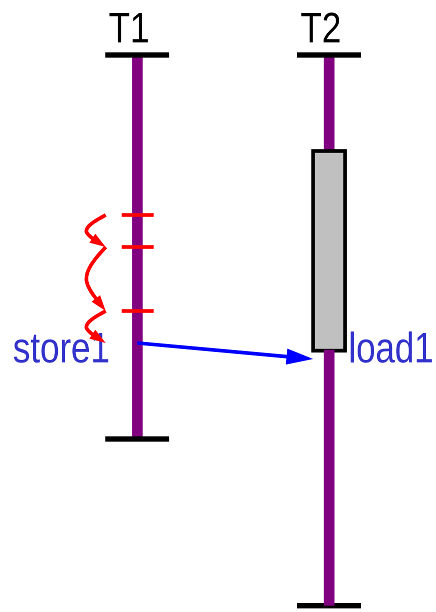
Est. I



Synchronization Cost =

# of dependent instructions

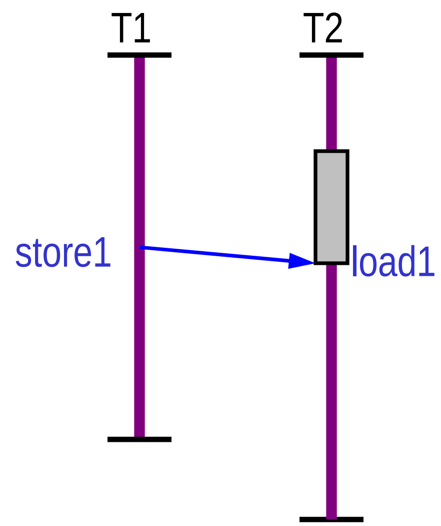
Est. II



Synchronization Cost =

longest stall

Est. III



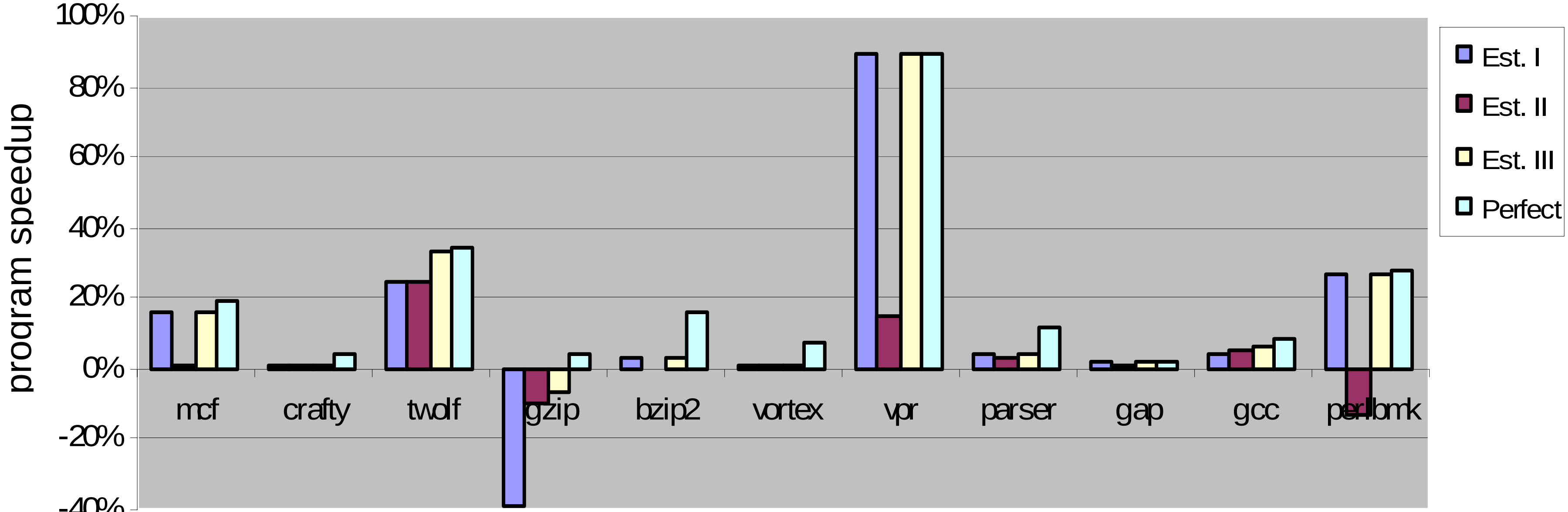
Synchronization Cost =

longest stall based on dependent instructions

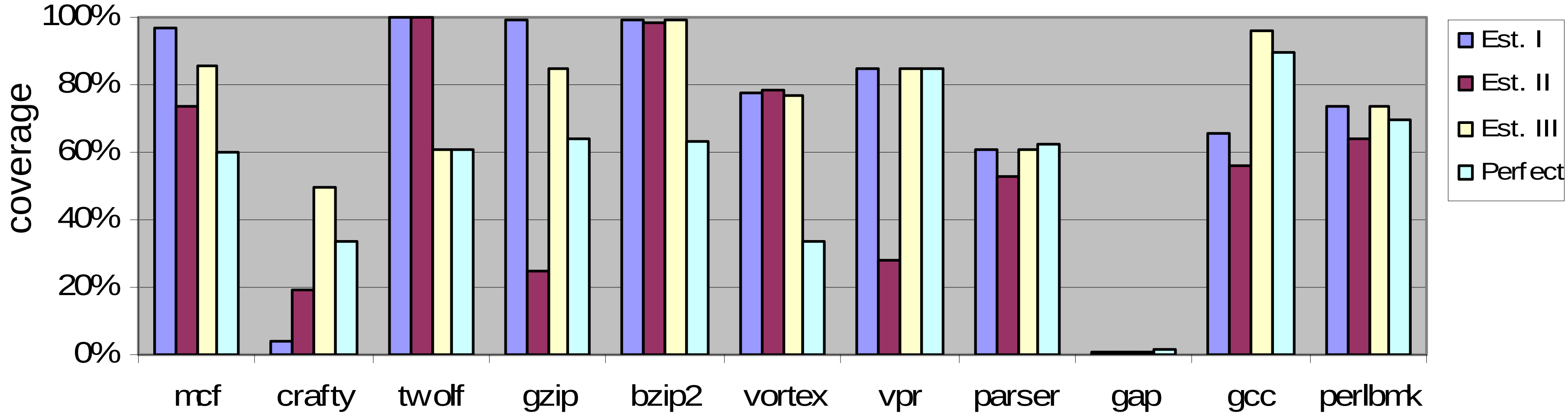
Experimental Framework

- Machine model
  - 4 single-issue in-order processors
  - Private L1 data cache (32K, 2-way, 1 cycle)
  - Shared L2 data cache (2M, 4-way, 10 cycles)
  - Speculation support (write buffer, address buffer)
  - Synchronization support (comm. buffer, 10 cycles)
- Compiler optimizations using ORC 2.1
  - Instruction scheduling to improve parallelism

Comparison: Speedup Estimation Techniques



Average program speedup: 20%, coverage: 70%



Outline

- Loop selection
  - Algorithm
  - Parallel performance prediction
- ➢ Dynamic loop behavior
- Conclusions

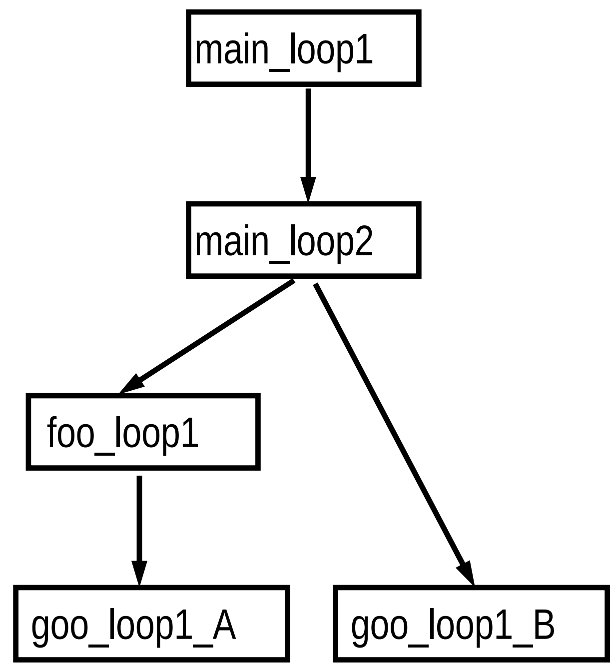
Loop Behavior May Change

```
main( ) {
   while ( condition1 ) {
      while ( condition2 ) {
         foo( );
         goo( );
      }
   }
}
```

```
foo( ) {
   while ( condition3 ) {
      goo( );
   }
}
```

```
goo( ) {
   while ( condition4 ) {
   }
}
```

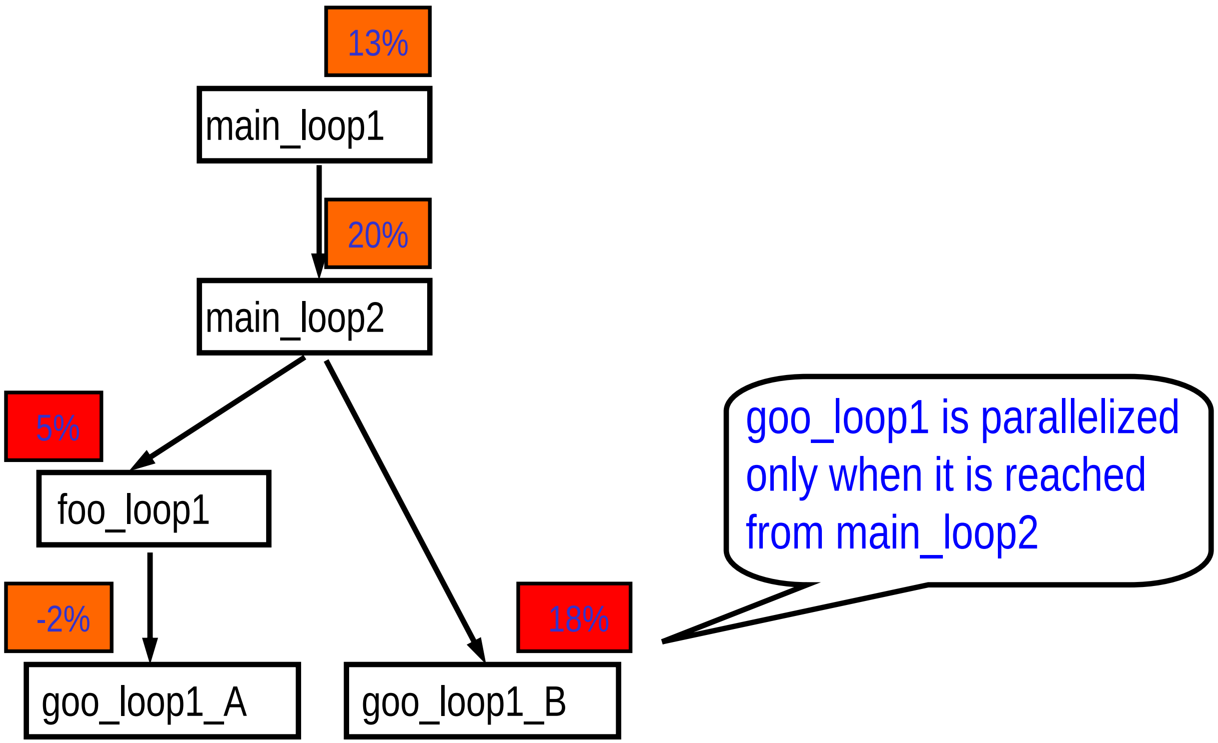
Source code



Loop tree

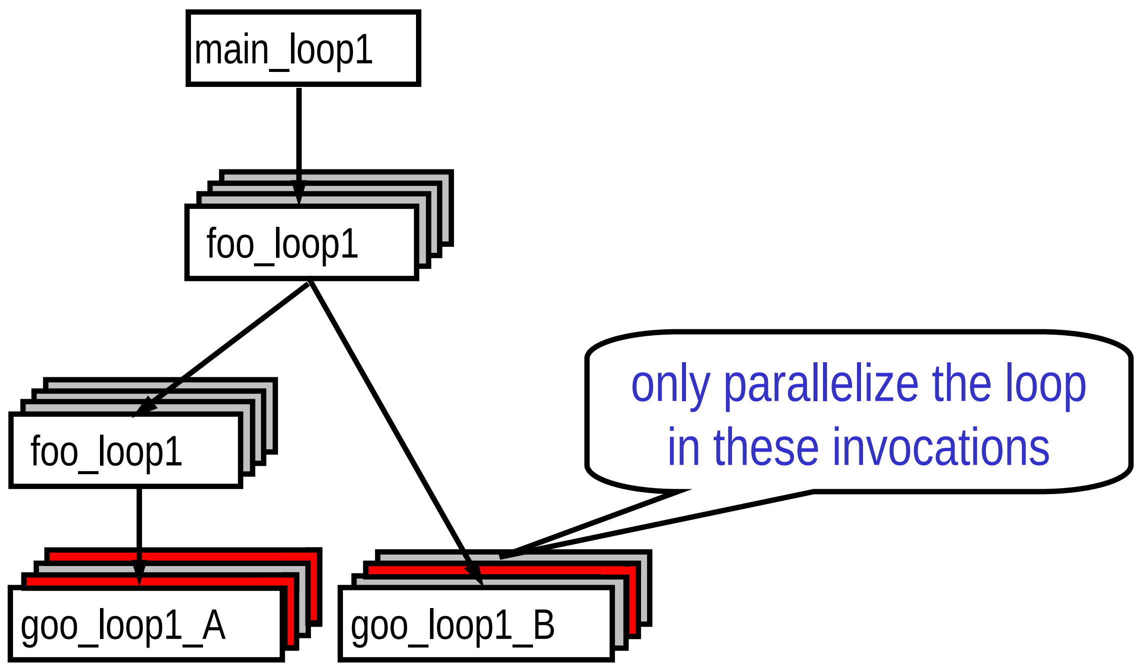
Calling context of a loop:
the path from the root to that loop

Loop Selection in a Tree



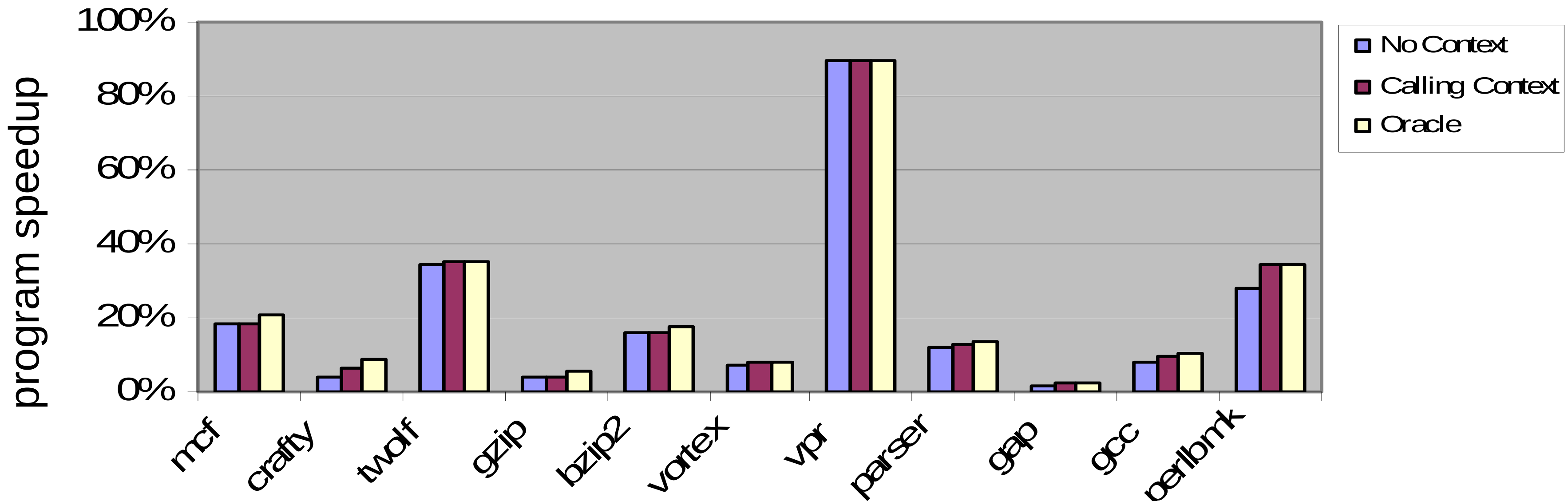
Loop cloning can be used

Loop Behavior May Change

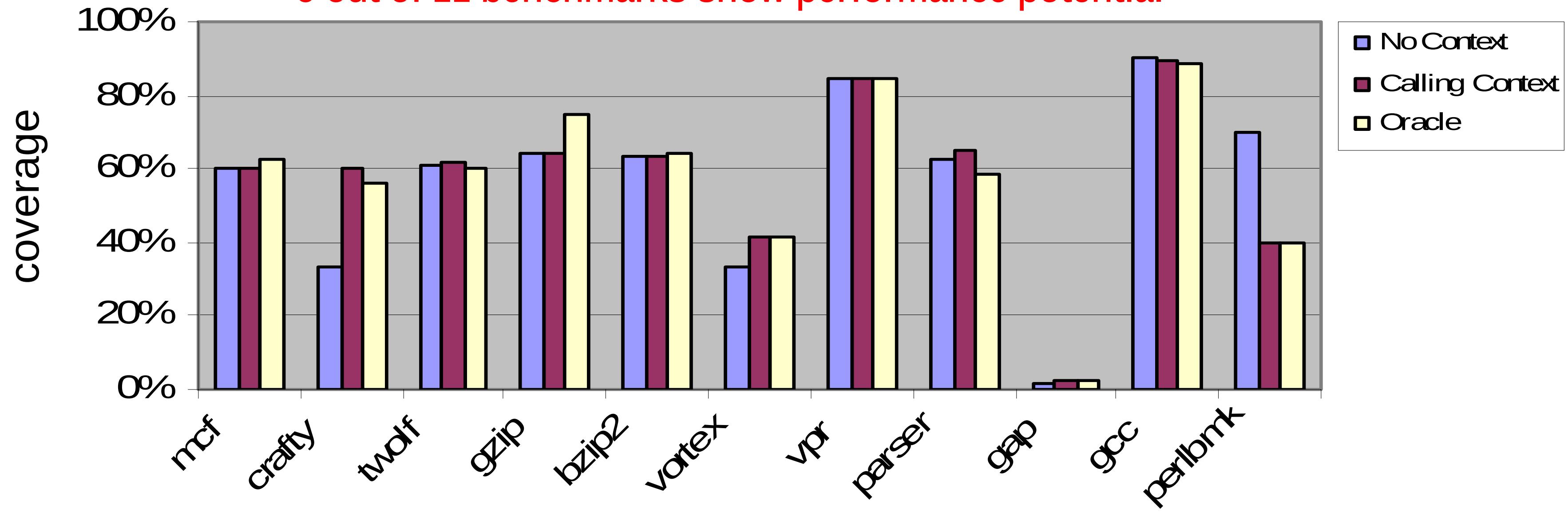


Exploit loop behavior dynamically

Potential of Exploiting Dynamic Behavior



5 out of 11 benchmarks show performance potential



Conclusions

Loop selection is important for TLS

- Compiler-based loop selection
  - Speedup 20%, Coverage 70%
- Exploiting dynamic behavior offers performance potential

Thank You!