Abstract Analysis of Method-Level Speculation

Clark Verbrugge
McGill University

Allan Kielstra
IBM Toronto Lab

Christopher J.F. Pickett
McGill University

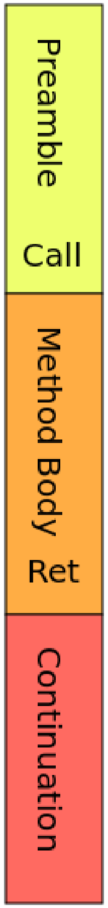
Compiler-Driven Performance, November 10, 2011

Contents

- Essential background
- Modeling MLS
    - In-order, out-of-order, nested
    - Signaling
- Abstraction
- Experiments
- Conclusion and future work

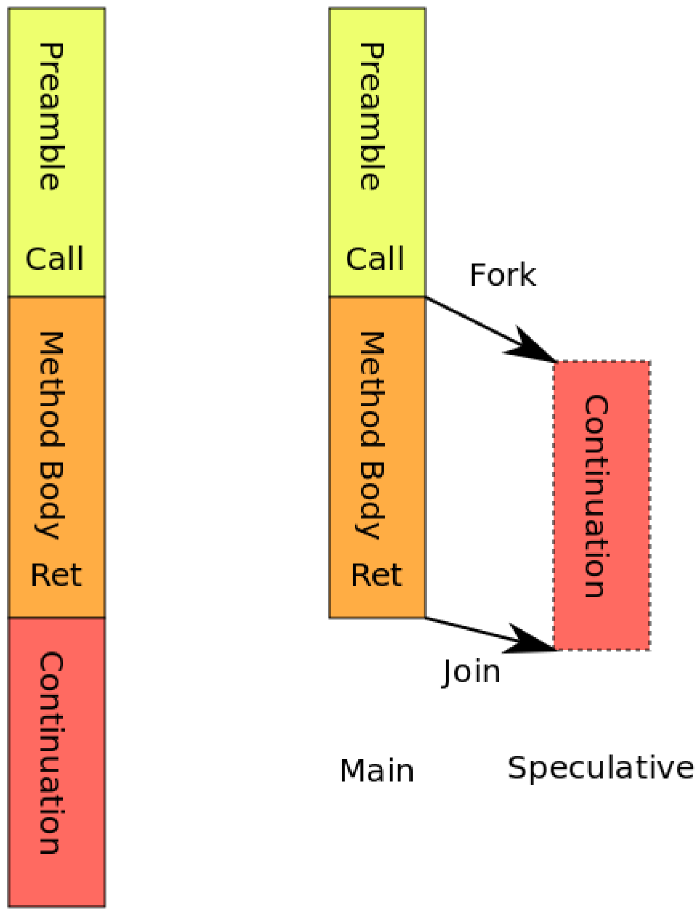
Background

- Method-Level Speculation



Background

- Method-Level Speculation



Background

- Issues
  - Safety: validate speculative thread
  - Overhead
    - Forking
    - Joining & validating
    - Speculative isolation
  - Parallel work
    - Length of method, continuation
    - Misspeculation
    - Fork points

Background

- Existing systems
  - Focus on data dependencies
  - Careful heuristics
  - Context-specific
  - Varying performance...
- Why?
  - Feedback; resource-limited.
  - Speculative “style” vs code

Modeling MLS

- MLS Constraint Graph

```
A() {
  work1
  B()
  work2
}
```

```
B() {
  work3
  C()
  work4
}
```

```
C() {
  work5
}
```

Modeling MLS

- MLS Constraint Graph

```
A() {
  work1
  B()
  work2
}

B() {
  work3
  C()
  work4
}

C() {
  work5
}
```

Execution: A $\rightarrow$ w1 $\rightarrow$ B $\rightarrow$ w3 $\rightarrow$ C $\rightarrow$ w5 $\rightarrow$ w4 $\rightarrow$ w2 $\rightarrow$ 0

Modeling MLS

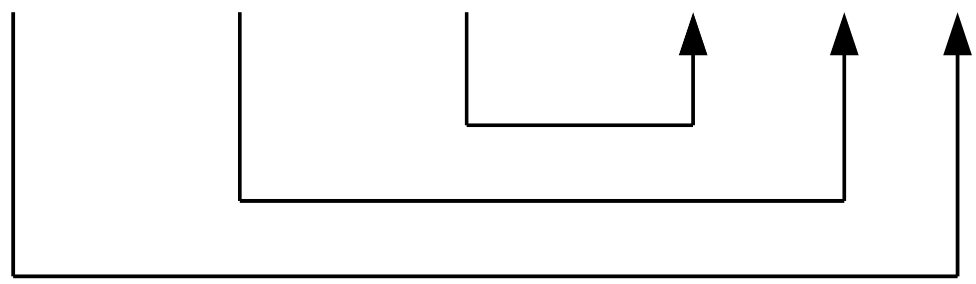
- MLS Constraint Graph

```
A() {
  work1
  B()
  work2
}
```

```
B() {
  work3
  C()
  work4
}
```

```
C() {
  work5
}
```

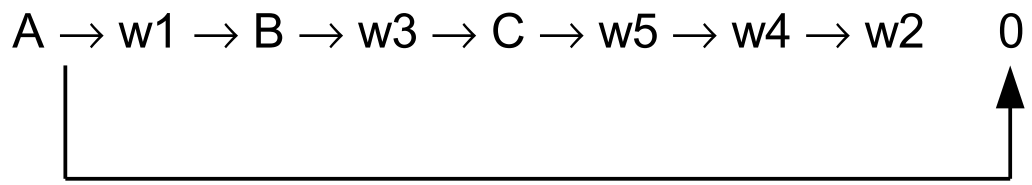
Execution: A → w1 → B → w3 → C → w5 → w4 → w2 → 0



Continuation edges

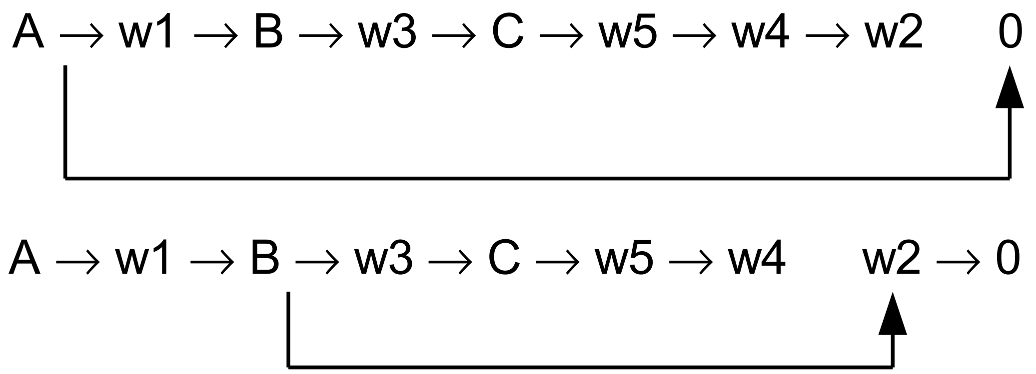
Modeling MLS

- All possible MLS executions



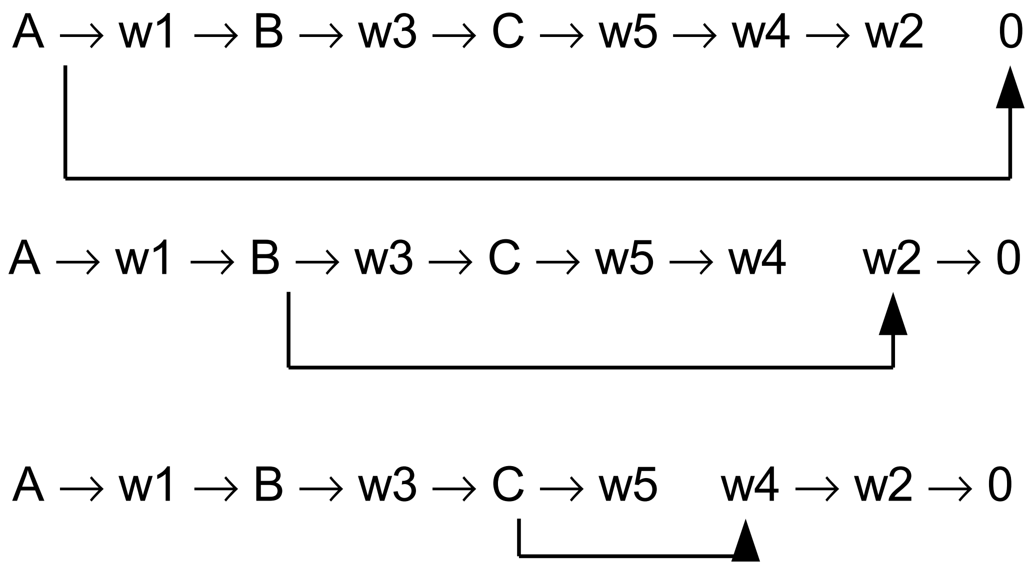
Modeling MLS

- All possible MLS executions



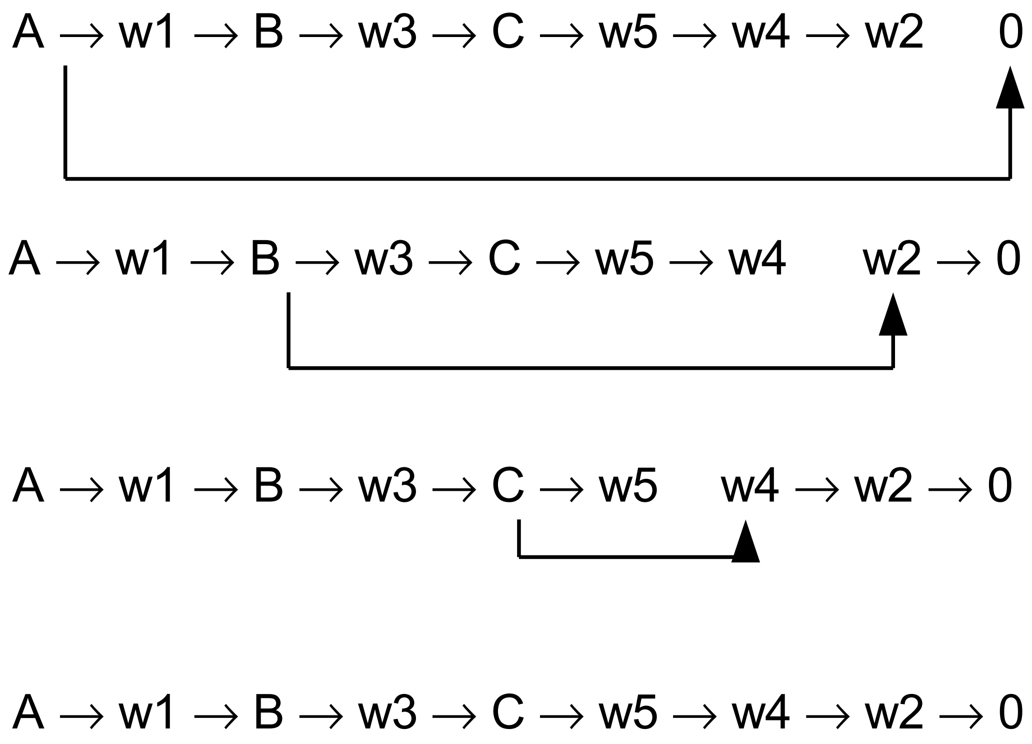
Modeling MLS

- All possible MLS executions



Modeling MLS

- All possible MLS executions



Modeling MLS

- Speculation Styles
  - Usually more than 1 speculative thread
- Out-of-order
  - Create multiple spec children from a thread
- In-order
  - Spec children can create spec children
- Nested
  - Both

Modeling MLS

- Signaling Disciplines
  - Support thread reuse
- Forward-signaling
  - Parent signals child to stop
  - Improves parallelism, mostly for out-of-order
- Backward-signaling
  - Child signals parent
  - Improves parallelism, mostly for in-order
    - But must retain child states

Modeling MLS

- Assume T = SABC
  - S is the sequential preamble
  - A method body
  - B continuation (pre-join)
  - C continuation (post-join)
- Full formula:

MLS(T=SABC) = S ; MLS(A) | MLS(B) + MLS(C)

Abstraction

$T = t_1, t_2, \ldots, t_n$

MLS(T,d,time) =

  for all S = preamble(T,d) s.t. time(S) < time

    let $(t_{|S|+1}, t_b)$ be a continuation edge

    $T_A = t_{|S|+1}, \ldots, t_{b-1}$

    for all d_1, d_2 = d-1,0 // out-of-order

      0,d-1 // in-order

      split(d-1) // nested

    for all A = MLS(T_A,d_1,time-time(S)-F)

      $T_B = t_b, \ldots, t_n$

      for all B = MLS(T_B,d_2,time(A))

        $T_C = t_{|S|+|A|+|B|+1}, \ldots, t_n$

        time(S;A|B) = time(S) + F + max(time(A),time(B)) + J

        for all C = MLS(T_C,d,time-time(S;A|B))

          time(T) = time(S;A|B) + time(C)

          return S ; A | B + C

Abstraction

- Exhaustive analysis
  - Model in-order, out-of-order, nested
- Show maximum parallel potential
  - Interaction of spec design and code
  - Assume no misspeculation
    - Adds overhead, reduces available threads

Experiments

- Basic coding idioms
  - Iteration
    - for(...) { work(); }   (10 iters)
  - Head-recursion
    - head() { head(); work(); } (10 levels)
  - Tail-recursion
    - tail() { work(); tail(); } (10 levels)
  - Tree-add: double head-recursion
    - ta() { ta(); ta(); work; } (3 levels)

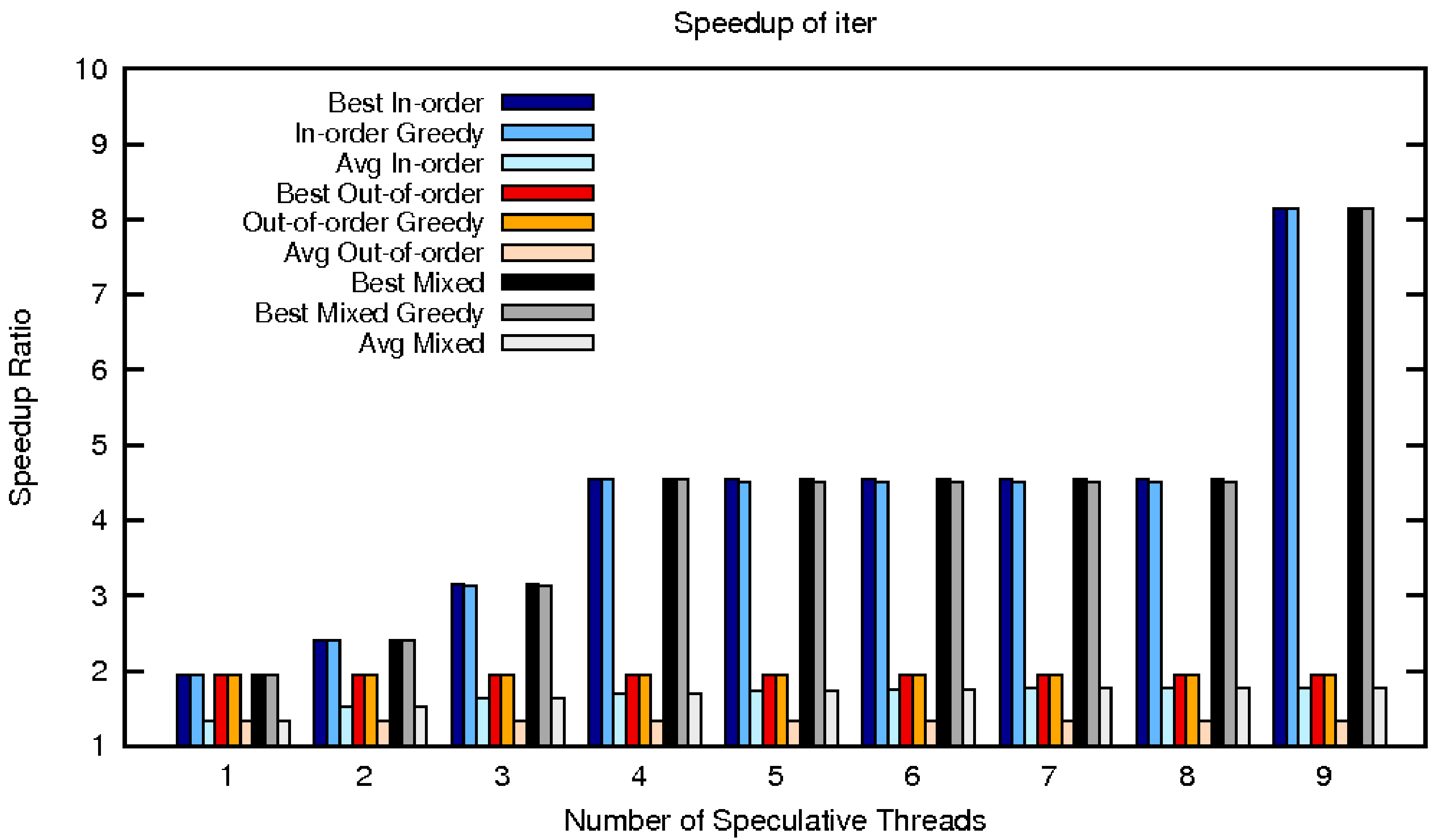
Experiments

- Abstract time units
  - Method-call: 5 units
  - Fork: 5 units
  - Join: 20 units
  - Work: 1000 units
- Maximal parallelism; no misspeculation

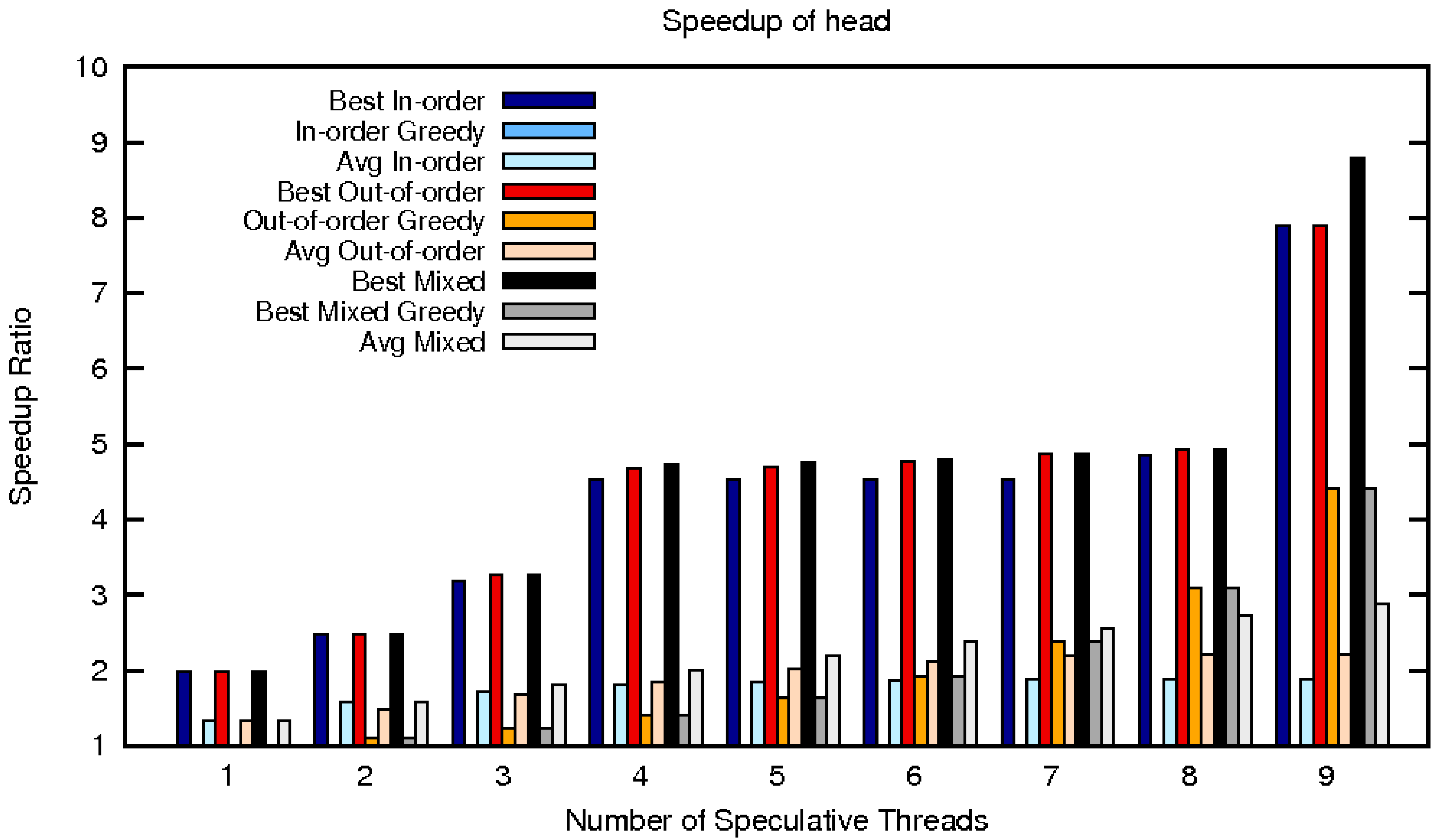
Experiments

- Measurements
  - Speedup
    - In-order, out-of-order, nested (forward-signaling)
    - Max, average, “greedy” fork heuristic
  - Weight sensitivity
    - Scale fork/join overhead 0...10000 units
    - (not shown)
  - Code structure
    - Simple code changes

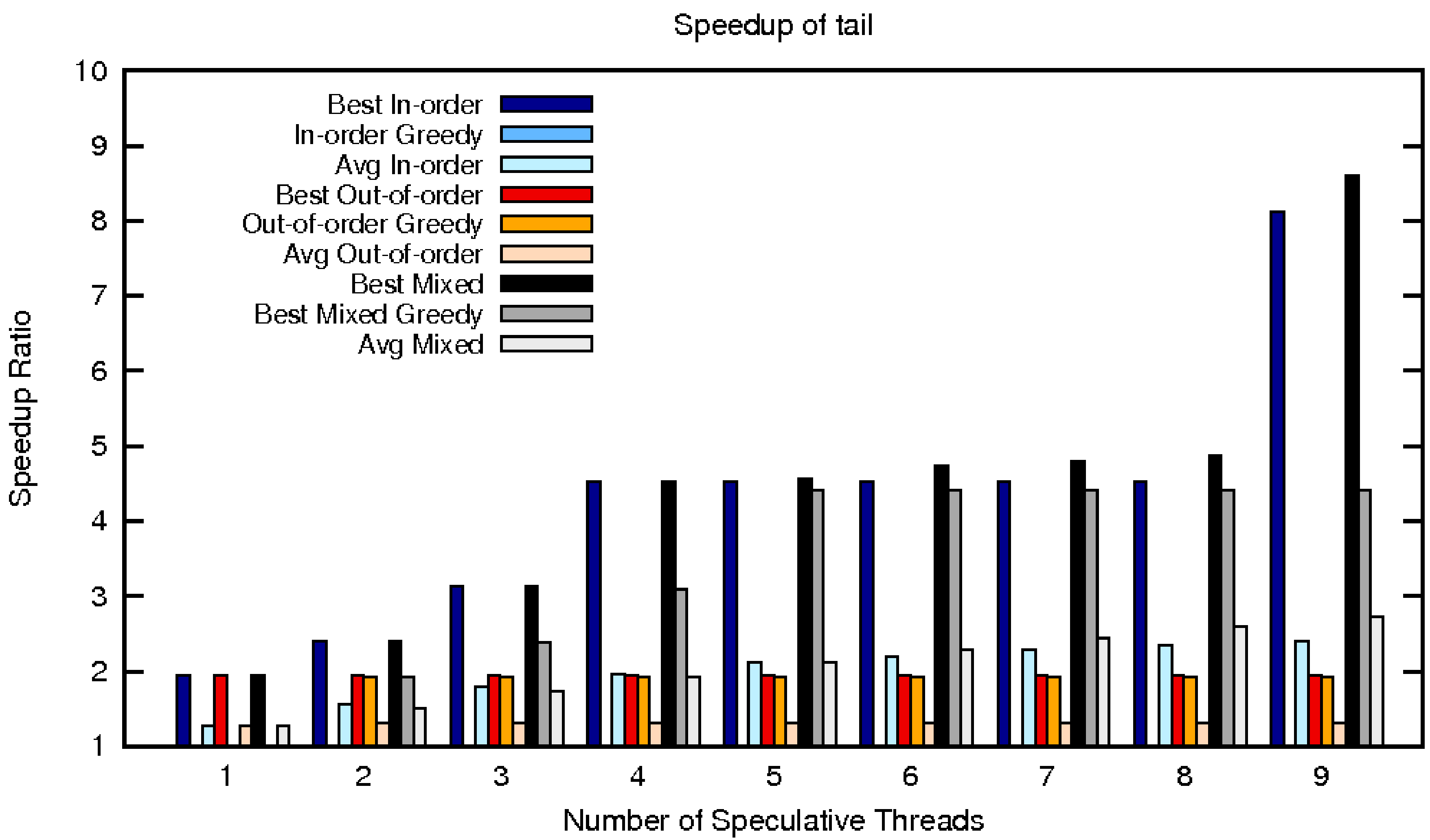
Experiments: Speedup



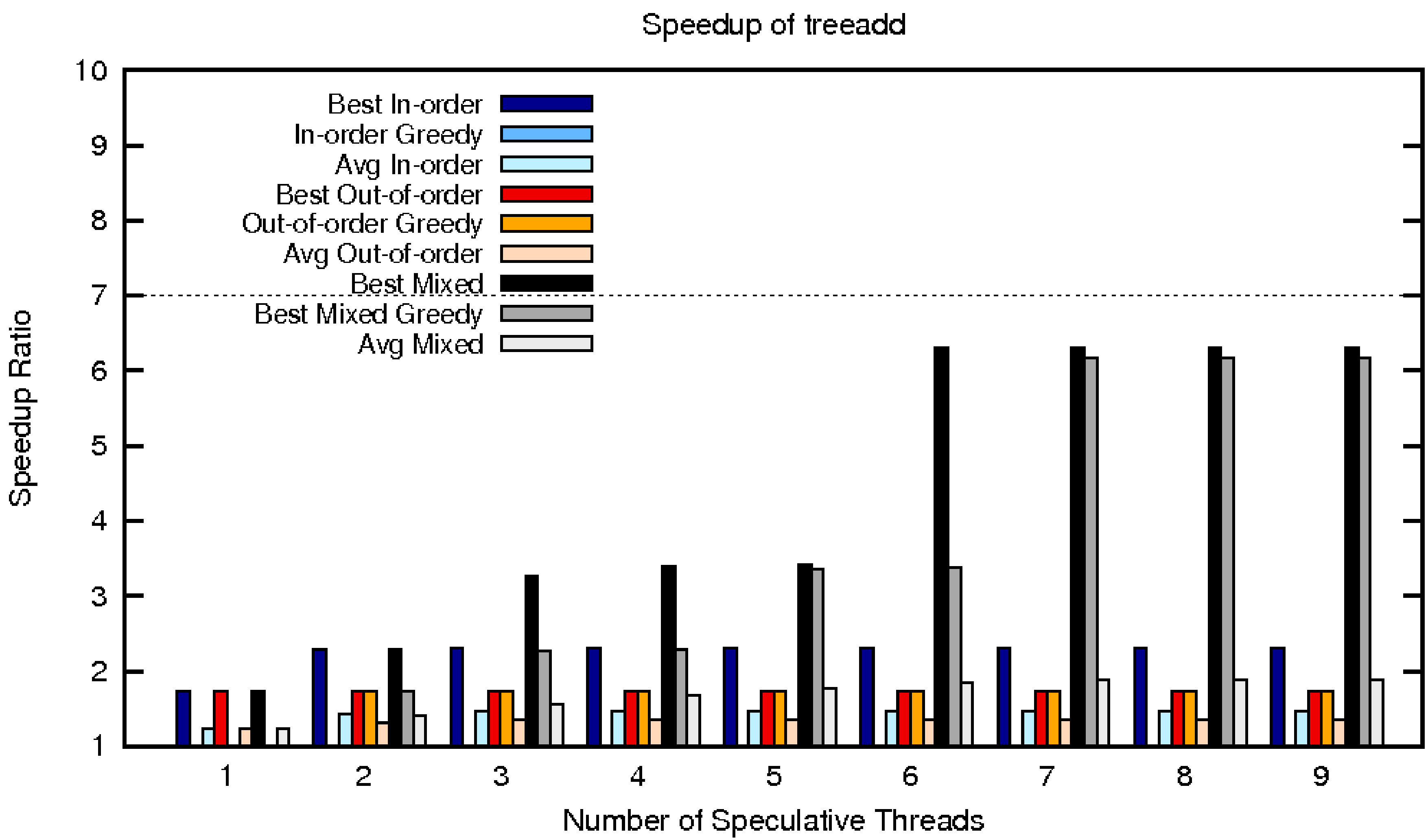
Experiments: Speedup



Experiments: Speedup



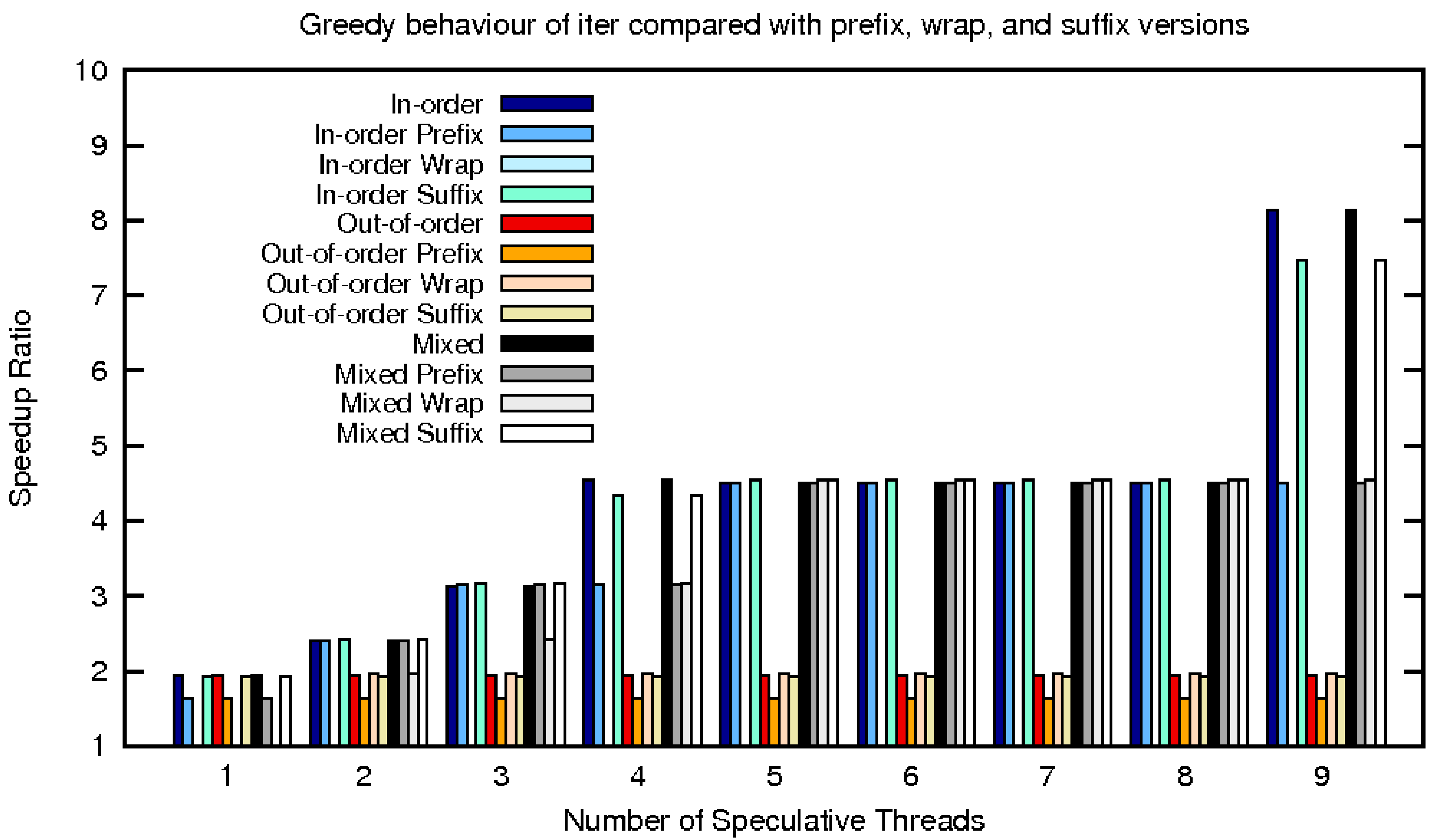
Experiments: Speedup



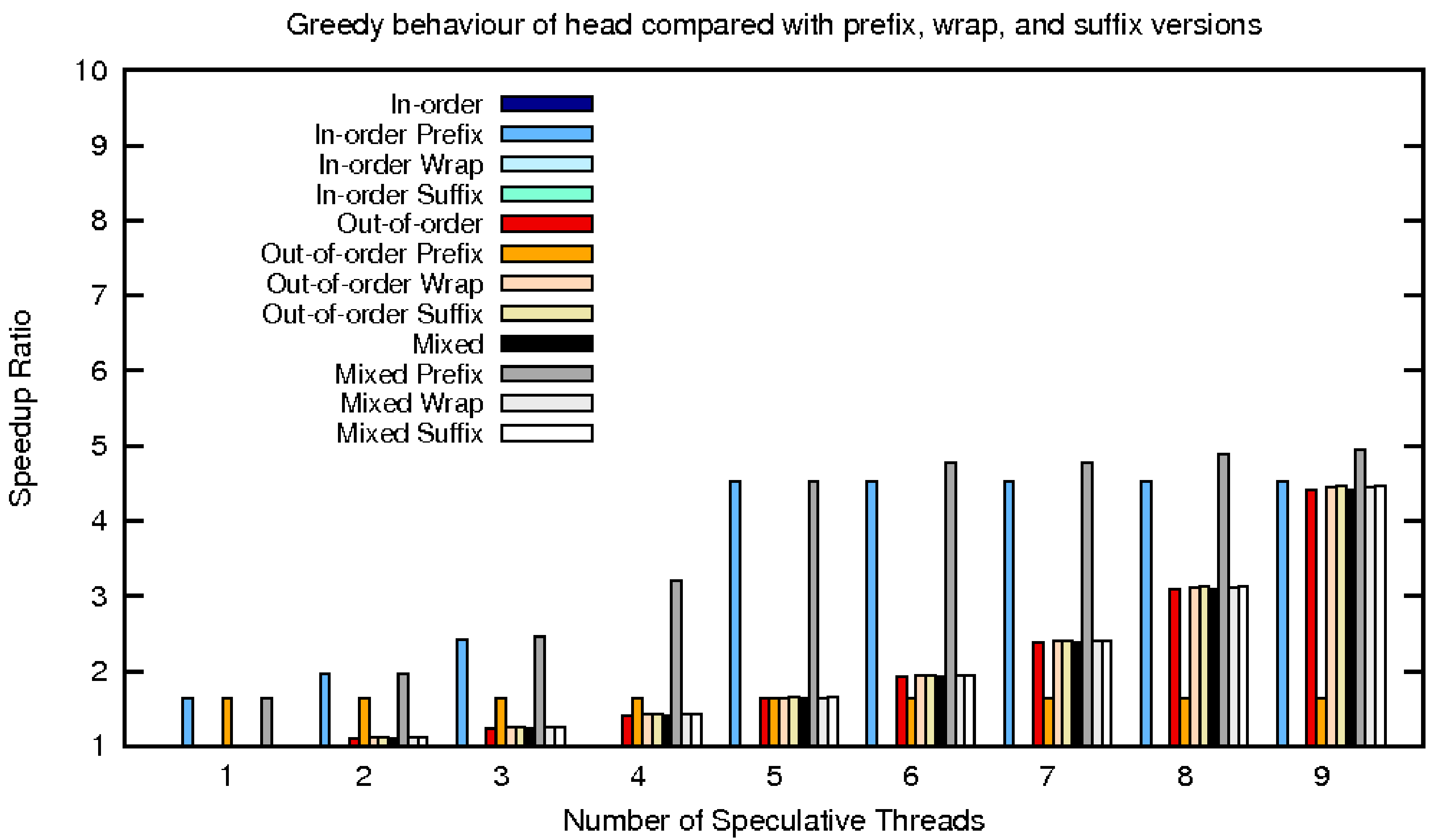
Experiments: Structure

- Greedy forking
- Prefix:
  - prefix() { work; }; benchmark();
- Wrap:
  - wrap { benchmark(); work; }
- Suffix:
  - benchmark(); suffix() { work; }

Experiments: Structure



Experiments: Structure



Conclusions

- Improve understanding of TLS
  - Interaction of speculation-style and code
  - Feedback properties
- Abstraction
  - Exhaustive analysis
  - Greedy behaviour
- Step to further abstraction

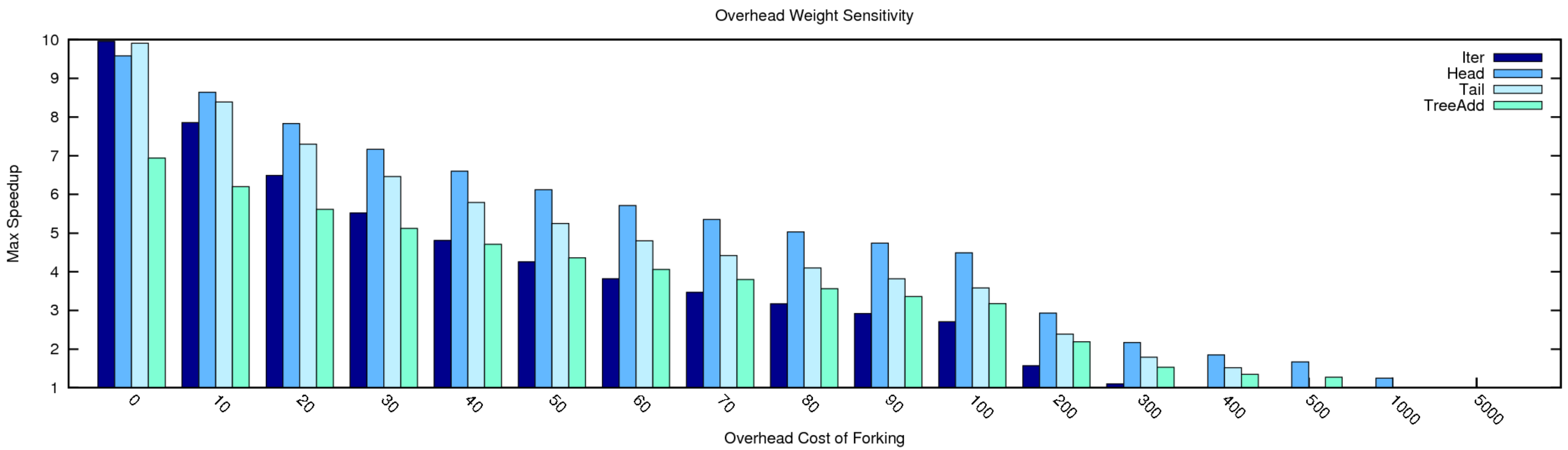
Future Work

- Examine other factors
  - Misspeculation due to data-dependencies
  - Non-spec instructions
  - Backward-signaling; mixed signaling
  - Different fork heuristics
- Real program workloads
- Basis for new fork heuristics

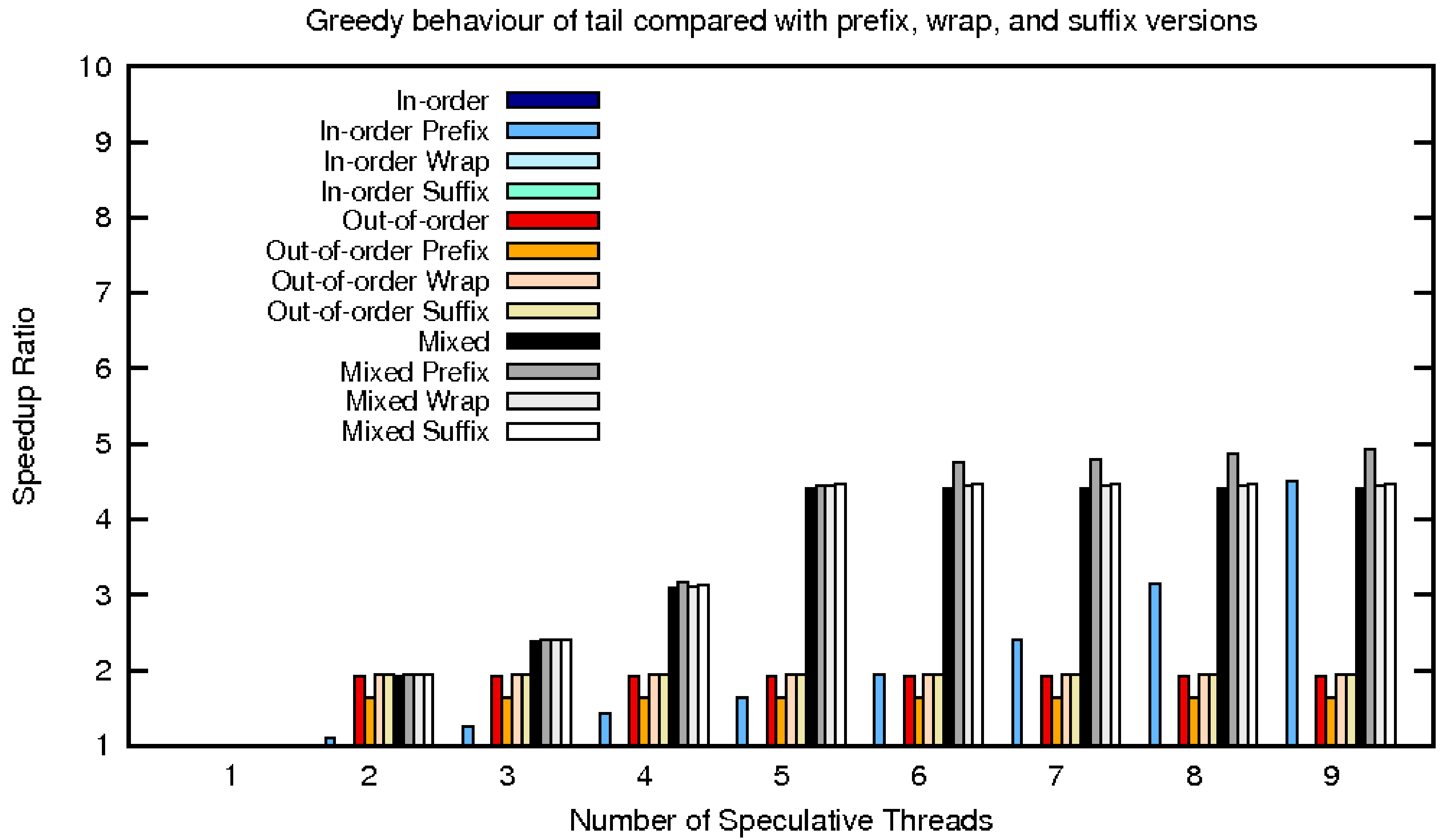
Done!

- Questions?

Experiments: Weight



Experiments: Structure



Experiments: Structure