SCAF: A Speculation–Aware Collaborative Dependence Analysis Framework

Sotiris Apostolakis, Ziyang Xu, Zujun Tan, Greg Chan, Simone Campanoni†, and David I. August

PLDI 2020
Optimizations

Compiler

Source Code

Optimized Code
Memory Analysis limits applicability of optimizations
Memory Analysis limits applicability of optimizations

- undecidable in theory [Landi, LPLS’92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.
Memory Analysis limits applicability of optimizations

- **undecidable in theory** [Landi, LPLS’92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.

- **insufficiently precise in practice** [Hind, PASTE’01]
  especially for languages like C/C++.
Memory Analysis limits applicability of optimizations

- **Undecidable in theory** [Landi, LPLS’92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.

- **Insufficiently precise in practice** [Hind, PASTE’01]
  Especially for languages like C/C++.

- **Conservatively respects all possible inputs**
  Many real dependences rarely occur in practice.
Speculation enables optimization of the expected case
Speculation enables optimization of the expected case
State-of-the-art $[1,2,3,4]$ does not fully leverage speculation

1. Johnson et al., PLDI ’12
2. Kim et al., CGO ’12
3. Mehrara et al., PLDI ’09
4. Vachharajani et al., PACT ’07
State-of-the-art $[1,2,3,4]$ does not fully leverage speculation

1. Johnson et al., PLDI '12
2. Kim et al., CGO '12
3. Mehrara et al., PLDI '09
4. Vachharajani et al., PACT '07
State-of-the-art \([1,2,3,4]\) does not fully leverage speculation.

Composition by Confluence

1. Johnson et al., PLDI '12
2. Kim et al., CGO '12
3. Mehrara et al., PLDI '09
4. Vachharajani et al., PACT '07
Motivating Example

```
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
            ...
        i3: a = ...
```
Motivating Example

loop L:
  if (rare)
    // no writes to a
    ...
  else
    a = ...

i1: a = ...
i2: foo(a)
    ...
i3: a = ...

Control Flow Graph

Hot path

Rarely taken path
Motivating Example

loop L:
    if (rare)
        // no writes to a
        ...
    else
        a = ...
        foo(a)
        ...
        a = ...

Is there a cross-iteration data flow from i3 to i2?
Motivating Example

```
loop L:
  if (rare)
    // no writes to a
    ...
  else
    a = ...

i1: a = ...
i2: foo(a)
     ...
i3: a = ...
```

Is there a cross-iteration data flow from i3 to i2?
Motivating Example

loop L:
    if (rare)
        // no writes to a
        ...
    else
        a = ...
        foo(a)
        ...
        a = ...

Is there a cross-iteration data flow from i3 to i2?

Composition by Confluence cannot assert its absence.
Motivating Example

loop L:
  if (rare)
    // no writes to a
    ...
  else
    a = ...

i1: a = ...
i2: foo(a)
    ...
i3: a = ...

Is there a cross-iteration data flow from i3 to i2?

Control Flow Graph
Motivating Example

loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
            ...
        i3: a = ...

Is there a cross-iteration data flow from i3 to i2?
Motivating Example

```
loop L:
  if (rare)
    // no writes to a
    ...
  else
    a = ...
    foo(a)
    ...
  a = ...
```

Is there a cross-iteration data flow from i3 to i2?
Motivating Example

```plaintext
loop L:
  if (rare)
    // no writes to a
    ...
  else
    i1:  a = ...
    i2:  foo(a)
    ...
    i3:  a = ...
```

Is there a cross-iteration data flow from i3 to i2?

Memory analysis and speculation combined can assert its absence.
Monolithic Integration $^{[1,2,3]}$

Source Code → Optimizations → Speculation–Aware Dependence Analysis → Optimized Code

Compiler

1. Apostolakis et al., ASPLOS ’20
2. Devecsery et al., ASPLOS ’18
3. Fernandez et al., PACT ’02
Proposed Approach: Composition by Collaboration
Proposed Approach is both **Modular & Collaborative**

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Supported Forms of Collaboration</th>
<th>Memory Analysis Decoupled from Speculation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Among Speculative Techniques</td>
<td>Between Memory Analysis and Speculative Techniques</td>
</tr>
<tr>
<td>Monolithic Integration [1,2,3]</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Composition by Confluence [4,5,6,7]</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Composition by Collaboration (This Work)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1 Apostolakis et al., ASPLOS ’20  2 Devecsery et al., ASPLOS ’18  3 Fernandez et al., PACT ’02  4 Johnson et al., PLDI ’12  5 Kim et al., CGO ’12  6 Mehrara et al., PLDI ’09  7 Vachharajani et al., PACT ’07
CAF*: Collaborative Dependence Analysis Framework

Collaborative resolution of analysis queries by simple analysis algorithms

* Nick P. Johnson et al., Collaborative Dependence Analysis Framework in CGO ‘17
CAF*: Collaborative Dependence Analysis Framework

Collaborative resolution of analysis queries by simple analysis algorithms

Isolate propositions beyond ones module's logic as premise queries.

* Nick P. Johnson et al., Collaborative Dependence Analysis Framework in CGO '17
SCAF: Speculation-Aware Collaborative Dependence Analysis Framework

* Nick P. Johnson et al., Collaborative Dependence Analysis Framework in CGO '17
SCAF: Speculation-Aware Collaborative Dependence Analysis Framework

*Nick P. Johnson et al., Collaborative Dependence Analysis Framework in CGO '17
SCAF’s Query Language
SCAF’s Query Language

Query Response:
Analysis result might be predicated on speculative assertions
SCAF’s Query Language

Query Response:
Analysis result might be predicated on speculative assertions

New Query Parameters:
Control-flow parameter in the form of dominance information
SCAF’s Query Language

Query Response:
Analysis result might be predicated on speculative assertions

New Query Parameters:
Control-flow parameter in the form of dominance information
Desired result parameter for quick bail-out
loop L:
    if (rare)
        // no writes to a
        ...
    else
i1:    a = ...
i2:    foo(a)
        ...
i3:    a = ...
SCAF in action

```plaintext
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1:  a = ...
        i2:  foo(a)
            ...
        i3:  a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```java
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
        ...
        i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
        ...
        i3: a = ...

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```python
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
        ...
        i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```plaintext
loop L:
  if (rare)
    // no writes to a
    ...
  else
    i1: a = ...
    i2: foo(a)
    ...
    i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```python
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
            ...
        i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?

---

**Diagram:**

1. Client
   - modref(i3,i2,ci,dt)
   - Orchestrator
SCAF in action

```java
loop L:
    if (rare)
        // no writes to a
        ...
    else
        a = ...
```

```java
i1: a = ...
i2: foo(a)
    ...
i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
    i2: foo(a)
        ...
    i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
loop L:
  if (rare)
    // no writes to a 
    ...
  else
  i1:   a = ...
  i2:   foo(a)
    ...
  i3:   a = ...

Is there a cross-iter data flow from i3 to i2?
SCAF in action

loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
        ...
        i3: a = ...

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```plaintext
loop L:
  if (rare)
    // no writes to a
    ...
  else
    i1: a = ...
    i2: foo(a)
    ...
    i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```plaintext
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
            ...
        i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```c
loop L:
  if (rare)
    // no writes to a
    ...
  else
    a = ...

i1: a = ...
i2: foo(a)
...
i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?
SCAF in action

loop L:
    if (rare)
       // no writes to a
       ...
    else
        a = ...
    i1: a = ...
    i2: foo(a)
       ...
    i3: a = ...

Is there a cross-iter data flow from i3 to i2?

Client

Kill Flow Module

Orchestrator

Control Spec

modref(i3,i2,ci,specdt)

premise query

spec-dt encodes assertion that branch never taken

modref(i3,i2,ci,specdt)

modref(i3,i2,ci,dt)

modref(i3,i2,ci,dt)
SCAF in action

Is there a cross-iter data flow from i3 to i2?

View of loop based on spec-dt

loop L:
    if (rare)
        // no writes to a
        ...
    else
    i1:    a = ...
    i2:    foo(a)
            ...
    i3:    a = ...

p"
SCAF in action

```plaintext
loop L:
  if (rare)
    // no writes to a
    ...
  else
    i1:  a = ...
    i2:  foo(a)
    ...
  i3:  a = ...
```

Is there a cross-iter data flow from i3 to i2?

View of loop based on spec-dt

---

premise query

1. Client
2. Orchestrator
3. Kill Flow Module
4. Control Spec
5. modref(i3,i2,ci,specdt)
6. modref(i3,i2,ci,specdt)
7. (NoModRef, ∅)

spec-dt encodes assertion that branch never taken

loop L:
  i1:  a = ...
  i2:  foo(a)
  ...
  i3:  a = ...

SCAF in action

Orchestrator

Control Spec

Client

Is there a cross-iter data flow from i3 to i2?
SCAF in action

```
loop L:
    if (rare)
        // no writes to a
            ...
    else
        i1:  a = ...
        i2:  foo(a)
            ...
        i3:  a = ...
```

Is there a cross-iter data flow from i3 to i2?

View of loop based on spec-dt

---

---

spec-dt encodes assertion that branch never taken
loop L:
    if (rare)
        // no writes to a
        ...
    else
        i1: a = ...
        i2: foo(a)
        ...
        i3: a = ...

Is there a cross-iter data flow from i3 to i2?

View of loop based on spec-dt

SCAF in action

Orchestrator

Spec

Kill Flow Module

Control Spec

spec-dt encodes assertion that branch never taken

Speculation assertion A: branch never taken

Client

premise query

modref(i3,i2,ci,specdt)

(NoModRef, ∅)

(NoModRef, {A})

modref(i3,i2,ci,dt)

modref(i3,i2,ci,dt)

modref(i3,i2,ci,dt)

modref(i3,i2,ci,dt)
SCAF in action

```plaintext
loop L:
  if (rare)
    // no writes to a
    ...
  else
    i1: a = ...
    i2: foo(a)
    ...
    i3: a = ...
```

Is there a cross-iter data flow from i3 to i2?

View of loop based on spec-dt

---

Final outcome:
(NoModRef, {A})

Speculation assertion A: branch never taken
SCAF’s Evaluation Methodology
SCAF’s Evaluation Methodology

Empirically Evaluated Claim

SCAF reduces the need for memory speculation
SCAF’s Evaluation Methodology

Empirically Evaluated Claim
SCAF reduces the need for memory speculation

Benchmarks
16 C/C++ benchmarks from SPEC CPU
SCAF’s Evaluation Methodology

Empirically Evaluated Claim
SCAF reduces the need for memory speculation

Benchmarks
16 C/C++ benchmarks from SPEC CPU

State-of-art Baseline
Composition by Confluence: analysis results are the confluence of results of individual components [1,2,3,4]

1Johnson et al., PLDI ’12  2Kim et al., CGO ’12  3Mehrara et al., PLDI ’09  4Vachharajani et al., PACT ‘07
SCAF reduces need for expensive memory speculation
SCAF reduces need for **expensive memory speculation**
### SCAF enables various Forms of Beneficial Collaboration

<table>
<thead>
<tr>
<th></th>
<th>Benchmark Coverage</th>
<th>Loop Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Among Speculation Modules</td>
<td>87.5%</td>
<td>53.6%</td>
</tr>
<tr>
<td>Between Memory Analysis and Speculation Modules</td>
<td>93.8%</td>
<td>42.9%</td>
</tr>
<tr>
<td>All</td>
<td>100.0%</td>
<td>66.1%</td>
</tr>
</tbody>
</table>

Beneficial Collaboration: two or more modules collaboratively resolve more queries than in isolation.
New Desired Result Parameter reduces Query Latency

28% geomean reduction
Conclusion

- SCAF is a modular and collaborative dependence analysis framework that computes the full impact of speculation on memory dependence analysis.

- SCAF dramatically reduces, compared to the state-of-the-art, the need for expensive-to-validate memory speculation.

- SCAF is essential for memory analysis sensitive clients and a necessary step toward robust automatic parallelization.

Artifact available: https://doi.org/10.5281/zenodo.3751586