JITProf: Pinpointing JIT-Unfriendly JavaScript Code

Liang Gong\textsuperscript{1}, Michael Pradel\textsuperscript{2}, Koushik Sen\textsuperscript{1}

\textsuperscript{1} UC Berkeley, \textsuperscript{2} TU Darmstadt
Motivation

JavaScript: One of the most popular languages

Performance: Crucial for responsiveness and energy-efficiency
JIT Compilers to the Rescue?

Just-in-time compilation:
Performance despite JavaScript’s dynamism
JIT Compilers to the Rescue?

Just-in-time compilation:
Performance despite JavaScript’s dynamism

But: Relies on regularity assumptions about code

Developers may write JIT-unfriendly code
Example

```javascript
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
  node.left = this.root_;  
  node.right = this.root_.right;
  ...
} else {
  node.right = this.root_;  
  node.left = this.root_.left;
  ...
}
```

From Google’s Octane-Splay benchmark
Example

```javascript
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
    node.left = this.root_;  
    node.right = this.root_.right;
    ...
} else {
    node.right = this.root_;  
    node.left = this.root_.left;
    ...
}
```

Assumption broken: Properties initialized in inconsistent order

From Google’s Octane-Splay benchmark
Example

```javascript
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
    node.left = this.root_;
    node.right = this.root_.right;
    ...
} else {
    node.right = this.root_;  
    node.left = this.root_.left;
    ...
}
```

Assumption broken: Properties initialized in inconsistent order

From Google’s Octane-Splay benchmark
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
    node.right = this.root_.right;
    node.left = this.root_;
    ...
} else {
    node.right = this.root_;  
    node.left = this.root__.left;
    ...
}
How To Find JIT-Unfriendly Code?

CPU profiling?
- Many hot functions
- Few optimization opportunities

Feedback from engine?
- Engine-specific
- For experts only
This Talk: JITProf

Profiler that pinpoints JIT-unfriendly code

Program

Profiler 1

Profiler N

JITProf framework

Code location + explanation
This Talk: JITProf

Profiler that pinpoints JIT-unfriendly code

JIT-unfriendly code patterns

Program

Profiler 1

Profiler N

JITProf framework

Code location + explanation
Pattern 1: Polymorphic Operations

function f(a, b) {
    return a + b;
}

f(2, 5);
f("ESEC", "FSE");
Function $f(a, b)$:

```javascript
function f(a, b) {
  return a + b;
}
```

- **Location** that applies operation to different types
- JIT compiler cannot specialize code for plus operation
Pattern 2: Mostly Numeric Arrays

```javascript
var array = [];
for (var i = 0; i < 1000; i++) {
    array[i] = i;
}
array[4] = "abc";
```
Pattern 2: Mostly Numeric Arrays

```javascript
var array = [];
for (var i = 0; i < 1000; i++) {
    array[i] = i;
}
array[4] = "abc";
```

- Stores **non-numeric value** into a numeric array
- Forces change of array representation
Pattern 3: Inconsistent Obj. Layouts

```javascript
var x = {}; 
x.a = "hello";
x.b = "world";
.. = x.a;
```
Pattern 3: Inconsistent Obj. Layouts

```javascript
var x = {};

x.a = "hello";
x.b = "world";

.. = x.a;
```

- Engine keeps track of "hidden class":

  ![Diagram showing properties a and b]

  Offset of a: 0

- Inline caching to optimize property access:
  Replace `x.a` with `x[0]`
Pattern 3: Inconsistent Obj. Layouts

```
var x = {};  // Engine keeps track of "hidden class":
if (..) {
    x.a = "hello";
    x.b = "world";
} else {
    x.b = "world";
    x.a = "hello";
}
.. = x.a;  // Offset of a: 0 or 1
```

- Inline caching fails:
  Cannot replace `x.a` with fixed offset
Framework and API

newObj(loc, val)
getProp(loc, base, prop, val)

etc.

Profiler 1

Profiler N

val.meta
loc.meta
incrCtr(loc)

JITProf framework
Framework and API

- `newObj(loc, val)`
- `getProp(loc, base, prop, val)`
- `etc.`

- `val.meta`
- `loc.meta`
- `inrCtr(loc)`

Profiler 1 :: Profiler N

JITProf framework

Keep track of runtime events
Framework and API

newObj(loc, val)
getProp(loc, base, prop, val)

etc.

Meta information about objects and locations

Profiler 1

::

Profiler N

val.meta
loc.meta

incrCtr(loc)

JITProf framework
Framework and API

newObj(loc, val)
getProp(loc, base, prop, val)

etc.

val.meta
loc.meta

incrCtr(loc)

Profiler 1

Profiler N

JITProf framework

How often a location is involved in a JIT-unfriendly operation
Framework and API

newObj(loc, val)
getProp(loc, base, prop, val)

Implementation:
- Source-to-source transformation
- Based on Jalangi

[Sen et al., FSE’13]

JITProf framework

Implementation:
- Source-to-source transformation
- Based on Jalangi

[Sen et al., FSE’13]
Profiler 1: Polymorphic Operations

function f(a, b) {
    return a + b;
}
f(2, 5);
f("ESEC", "FSE");

- Track types of operands of unary and binary operations
- Different than last time? Increment unfriendliness counter
Profiler 2: Mostly Numeric Arrays

```javascript
var array = [];
for (var i = 0; i < 1000; i++) {
    array[i] = i;
}
array[4] = "abc";
```

- Track **state of each array**
- Numeric becomes non-numeric? Increment unfriendliness counter
Profiler 3: Inconsistent Obj. Layouts

```javascript
x = {};  
if (...) {
    x.a = "hi";
    x.b = "world";
} else {
    x.b = "world";
    x.a = "hi";
}
.. = x.a;
```

- Track **hidden classes of objects** and simulate inline caching
- Cache miss? Increment unfriendliness counter
Other Patterns and Profilers

- Binary operations on \texttt{undefined}
- Non-contiguous arrays
- Undefined array elements
- Unnecessary use of generic arrays
Other Patterns and Profilers

- Binary operations on undefined
- Non-contiguous arrays
- Undefined array elements
- Unnecessary use of generic arrays

53 – 278 lines of code per profiler
Easy to extend with new profilers
Sampling

Control overhead at two levels

- **Function level sampling:**
  Selectively run original or instrumented code

- **Instruction level sampling:**
  Selectively report events to profilers

Decaying sampling strategy
Evaluation

Setup
- Top 50 web sites
- Octane & SunSpider benchmarks
- Firefox + Chrome

Questions
- Prevalence of JIT-unfriendly code
- Effectiveness
- Compare: CPU profiling
- Overhead
Prevalence of JIT-Unfriendly Code

Total of top 50 web sites

- Polym. op.
Prevalence of JIT-Unfriendly Code

Total of top 50 web sites

- Incons. obj. lay.
- Polym. op.
- Mostly num. arrays
Optimization Opportunities

Found 15 optimization opportunities in benchmarks

Avoiding JIT-unfriendliness improves performance

- 1.1%–26.3% improvement (med.: 6.5%)
- 1–18 changed lines of code (med.: 2)
Examples (1)

```javascript
var node = new SplayTree.Node(key, value);
if (key > this.root_.key) {
    node.left = this.root_;  // Modify
    node.right = this.root_.right;
    ...
} else {
    node.right = this.root_;  // Modify
    node.left = this.root_.left;
    ...
}
```

Improvement:  ■ 3.5% in Firefox
              ■ 15.1% in Chrome
Examples (2)

```javascript
String.leftPad = function (val, size, ch) {
    var result = new String(val);

    if (ch == null) { ch = " "; }
    while (result.length < size) {
        result = ch + result;
    }
    return result;
}
```
Examples (2)

```javascript
String.leftPad = function (val, size, ch) {
    var result = val + "";
    var tmp = new String(val);
    if (ch == null) { ch = " "; }
    while (result.length < size) {
        result = ch + result;
    }
    return result;
}
```

Improvement:
- 19.7% in Firefox
- 22.4% in Chrome
## Comparison to CPU Profiling

<table>
<thead>
<tr>
<th></th>
<th>Rank of optimization opportunities</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>JITProf</td>
<td>1 or 2</td>
<td>Statement</td>
</tr>
<tr>
<td>CPU profilers</td>
<td>Higher for 76% of found opportunities</td>
<td>Function</td>
</tr>
</tbody>
</table>
Effect of Sampling

Overhead imposed by profiler

- Without sampling: 627x (median)
- With sampling: 18x (median)
  (no loss of optimization opportunities)

Acceptable for in-house performance analysis
Conclusion

JITProf:
Profiling to find JIT-unfriendly code

- Framework + 7 profilers
- Easily extensible
- Engine-independent

https://github.com/Berkeley-Correctness-Group/JITProf
Conclusion

JITProf:
Profiling to find JIT-unfriendly code

- Framework + 7 profilers
- Easily extensible
- Engine-independent

Thanks!

https://github.com/Berkeley-Correctness-Group/JITProf