Making Different JIT Compilations
Dancing to the Same Tune, Acting in the Meta-level

Yusuke Izawa and Hidehiko Masuhara
izawa@prg.is.titech.ac.jp, masuhara@is.titech.ac.jp, Tokyo Institute of Technology

Context

JIT Compilation

Two major JIT strategies:

- Trace-based compilation
- Method-based compilation

Problem

Trade-offs between the two

Good

- trace-based
  - inline automatically
  - convert non-tail rec. call to loop

Bad

- method-based
  - need carefully management for inlining

Our Approach

(1) Apply different strategies to different area

(2) Realize as a "meta-JIT compiler framework"

Proposal

Meta-hybrid JIT Compiler Framework

Virtual Machine Generation

language designer

Meta-interpreter

BacCaml

trace-based

Method-based

Execution

(Meta) JIT Compilation

Interpreter

Runtime

Machine code

Result

Standalone JIT Microbenchmark:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Tracing JIT</th>
<th>method JIT</th>
<th>MinCaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hybrid JIT Microbenchmark:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Hybrid JIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Future work

- Add optimizations for resulting traces
- Investigate profiling scheme for automatically selecting a suitable JIT strategy
- Apply our approach for RPython

References