A Sampling Profiler for JITing in Ř

Masters Presentation II

Andreas Wälchli
Supervisor: Olivier Flückiger

26. 05. 2020
Problem: Polluted Type Feedback in JIT-Compiler

```r
a <- 1
f <- function() { a + 1 }
f()
...
a <- 1L
f()
...
a <- 1
f()
...
```

run time
Compilation with Dynamic Typing

• Problem: no good type information from static analysis
  • need to run code to get type information
  • types may change between executions

• Solution: Warm-Up Phase
  • run code in interpreter
  • record type information during execution
  • compile with collected type information after several runs
Research Question

• (How) Can a sampling profiler be used to improve JIT-compiler output?

• Approach:
  • detect LLVM compiled code and sample dynamic types at runtime
  • trigger recompilation when sufficient data is collected
R Compilation Pipeline

R Source -> RIR Bytecode -> PIR SSA-IR -> LLVM

No Type Information

Interpreted Execution
Records Type Feedback
Sampling Values for Optimisation
Sampling Profiler

• Stop program execution at regular intervals
• Sample the program state
• Usually uses signals and signal handlers
Sample Triggering

**POSIX Timer**
- Precise Timing
- generate a SIGNAL on timeout
- can deadlock with SYSCALL at high frequency (>100Hz)

**Performance Measuring Unit (PMU)**
- Imprecise Triggering
- generate a SIGNAL after n Instructions in User Mode
- Sampling Rate not consistent between different CPUs
- Interference with SYSCALL unlikely
- Higher Possible Sample Rate
<table>
<thead>
<tr>
<th>Measurement</th>
<th>Sample Count per 2E6 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer 100Hz</td>
<td>7</td>
</tr>
<tr>
<td>PMU 1kOps</td>
<td>1055</td>
</tr>
<tr>
<td>PMU 10kOps</td>
<td>2366</td>
</tr>
<tr>
<td>PMU 100kOps</td>
<td>2045</td>
</tr>
<tr>
<td>PMU 1MOps</td>
<td>453</td>
</tr>
<tr>
<td>PMU 10MOps</td>
<td>33</td>
</tr>
</tbody>
</table>
Sample Collection

• Boxed Values live on a Stack
• Inspect the top-most Stack Frame
• We have a Mapping from Stack Slots to TypeFeedback Entries
• When sampling, take any live Slots and fill their TypeFeedback Entry
Ř JIT-Compiler

function call

already compiled?

N

function hot?

Y

should recompile?

N

recompile

run compiled version

compile

run in interpreter
Triggering Compilation

• When is a function ready for recompilation?
  • At least 50% of slots have at least 10 samples
  • At least one “ready” slot has more precise feedback than during compilation
  • Discard entry after 100 samples (prevent pollution)

• When compiling, the compiler takes “our” feedback for every slot that is “ready” (even when inlining the function somewhere else)
Results
2 Examples

Microbenchmark

```r
a <- 1
f <- function() a+a+a+a+1L
f()
f()
a <- 1L
f()
f()
system.time(for (i in 1:100000) f())
```

RSA Encryption

```r
p1 <- 971
p2 <- 383
n <- p1 * p2
e <- 17

encrypt <- function(msg) {
  p <- 1
  a1 <- msg
  for(i in 1:e) {
    p <- p*a1
    p <- p%%n
    i <- i+1
  }
p
}

system.time(for(i in 1:100000) encrypt(i))
n <- 371893L
system.time(for(i in 1:100000) encrypt(i))
```
Performance over Time - RSA encryption

- Warmup, Initial Compilation
- De-opt and Recompilation
- Profiler Triggered Recompilation

run time per 1E5 runs [s]

Iteration
Demo – RSA Encryption
Conclusion

• A Sampling profiler can improve performance in JIT-compiled code
• Large gains with type changes in global variables
Ř Compilation Pipeline

- **R Source**
- **RIR Bytecode**
- **PIR SSA-IR**
- **LLVM**

**Interpreted Execution**
Records Type Feedback

---

**Eventual Performance - RSA encryption**

**Performance over Time - RSA encryption**

- Warmup, Initial Compilation
- De-opt and Recompilation
- Profiler Triggered Recompilation