APPROACHES TO OPTIMIZE V8 JAVASCRIPT ENGINE

Dmitry Botcharnikov

2015/12/02
AGENDA

- JavaScript engine optimization
- V8 engine architecture
- Approaches to speed up V8 engine
  - Optimized build
  - Runtime parameters tuning
  - Scalar optimizations
- Conclusion
Why optimization of JavaScript matters?

- JavaScript is one of the most popular programming languages
- Samsung produces millions of devices running JavaScript => better utilize the hardware

**BUT:** JavaScript is dynamically typed prototype based object oriented interpreted language => complicated to optimize

We were involved in optimizing open source V8 engine (part of Samsung stock browser for Android)

**About 10% total performance improvement** on major benchmark suites: Octane, Kraken, SunSpider => now runs on Samsung mobile devices
V8 ENGINE ARCHITECTURE

JavaScript → AST → Hydrogen

Native Code ← Optimized ← Runtime

Full ← Lithium
Profiling of the engine reveals almost uniform distribution of work without ‘hot’ regions
APPROACHES TO SPEED UP V8 ENGINE

- We decided to focus on the following approaches
  - Optimized build of V8 engine itself
  - Tuning of V8 runtime options
  - Implementation of additional scalar optimizations
SELECTION OF BUILD PARAMETERS

- Link-time optimization: FAIL
- Platform options tuning: SUCCESS
  - -O3 for highest optimization level
  - -mcpu=cortex-a15 for target CPU

...
RUNTIME PARAMETERS TUNING

Octane score: ↑5.4%
SCALAR OPTIMIZATIONS

- Algebraic Expression Simplification
  - uses algebraic identities to simplify expressions:
    \[ x + 1 = x, \ y \times 1 = y, \ z \mid 0 = z \]

- Common Subexpression Elimination
  - path sensitive elimination of common subexpression

- Fast call frame
  - use ARMv7 specific call frame
ALGEBRAIC EXPRESSION SIMPLIFICATION

Octane score: \(\uparrow0.3\%\)
COMMON SUBEXPRESSSION ELIMINATION

Octane score: $\uparrow 1.8\%$
FAST CALL FRAME

Prologue:

func:
  
  \texttt{stmdb} \quad \texttt{sp!}, \{r4-r5, fp, lr\}
  
  \texttt{add} \quad \texttt{fp, sp, #N}

Epilogue:

\texttt{mov} \quad \texttt{sp, fp}

\texttt{ldmia} \quad \texttt{sp!}, \{r4-r5, fp, lr\}

\texttt{bx} \quad \texttt{lr}

Prologue:

func:
  
  \texttt{sub} \quad \texttt{sp, sp, #16}
  
  \texttt{stm} \quad \texttt{sp, \{r4,r5,fp, lr\}}
  
  \texttt{add} \quad \texttt{fp, sp, #N}

Epilogue:

\texttt{mov} \quad \texttt{sp, fp}

\texttt{ldm} \quad \texttt{sp, \{r4, r5, fp, lr\}}

\texttt{add} \quad \texttt{sp, sp, #16}

\texttt{bx} \quad \texttt{lr}
FAST CALL FRAME

2M calls: ↑10%

original

fast frame
CONCLUSION

- Application of traditional scalar optimizations in JavaScript gives diminishing returns
- Successful application of optimized build gives us evidence that there is a space for optimizations in JavaScript engines
Thank you