APPROACHES TO OPTIMIZE V8 JAVASCRIPT ENGINE

Dmitry Botcharnikov

2015/12/02

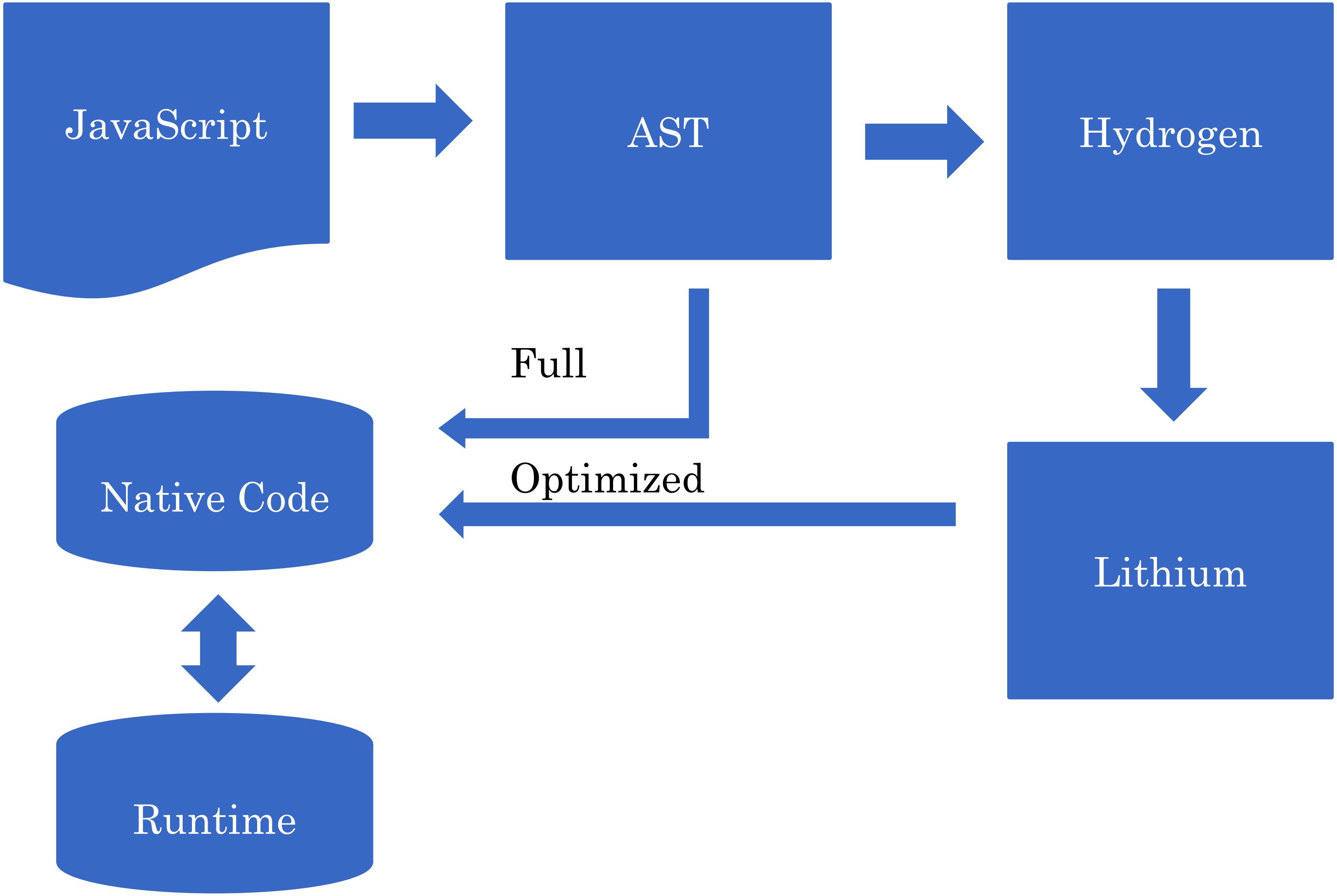
AGENDA

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

JAVASCRIPT ENGINE OPTIMIZATION

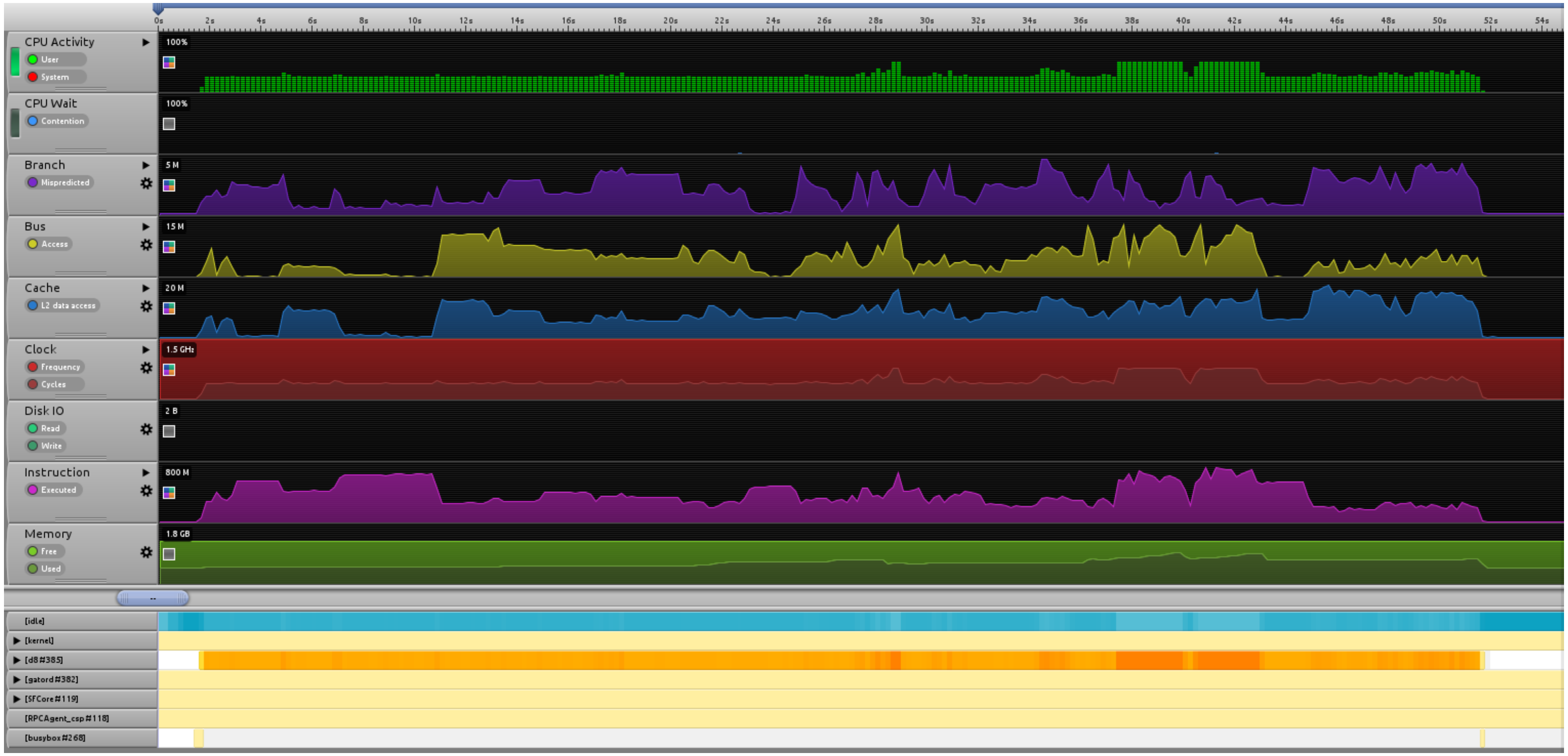
- Why optimization of JavaScript matters?
  - JavaScript is one of the most popular programming languages
  - Samsung produces millions of devices running JavaScript => better utilize the hardware
- **BUT:** JavaScript is dynamically typed prototype based object oriented interpreted language => complicated 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



PROFILING V8 ENGINE

Profiling of the engine reveals almost uniform distribution of work without ‘hot’ regions

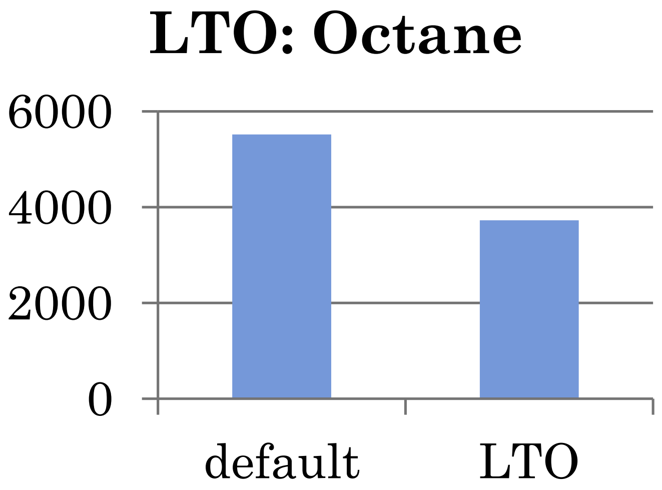


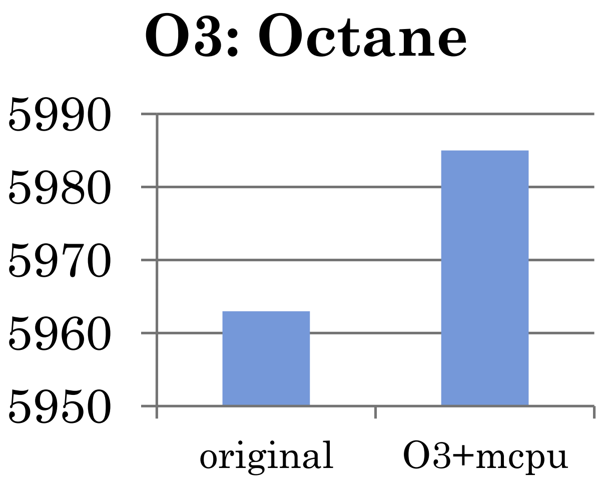
APPROACHES TO SPEED UP V8 ENGINE

- 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

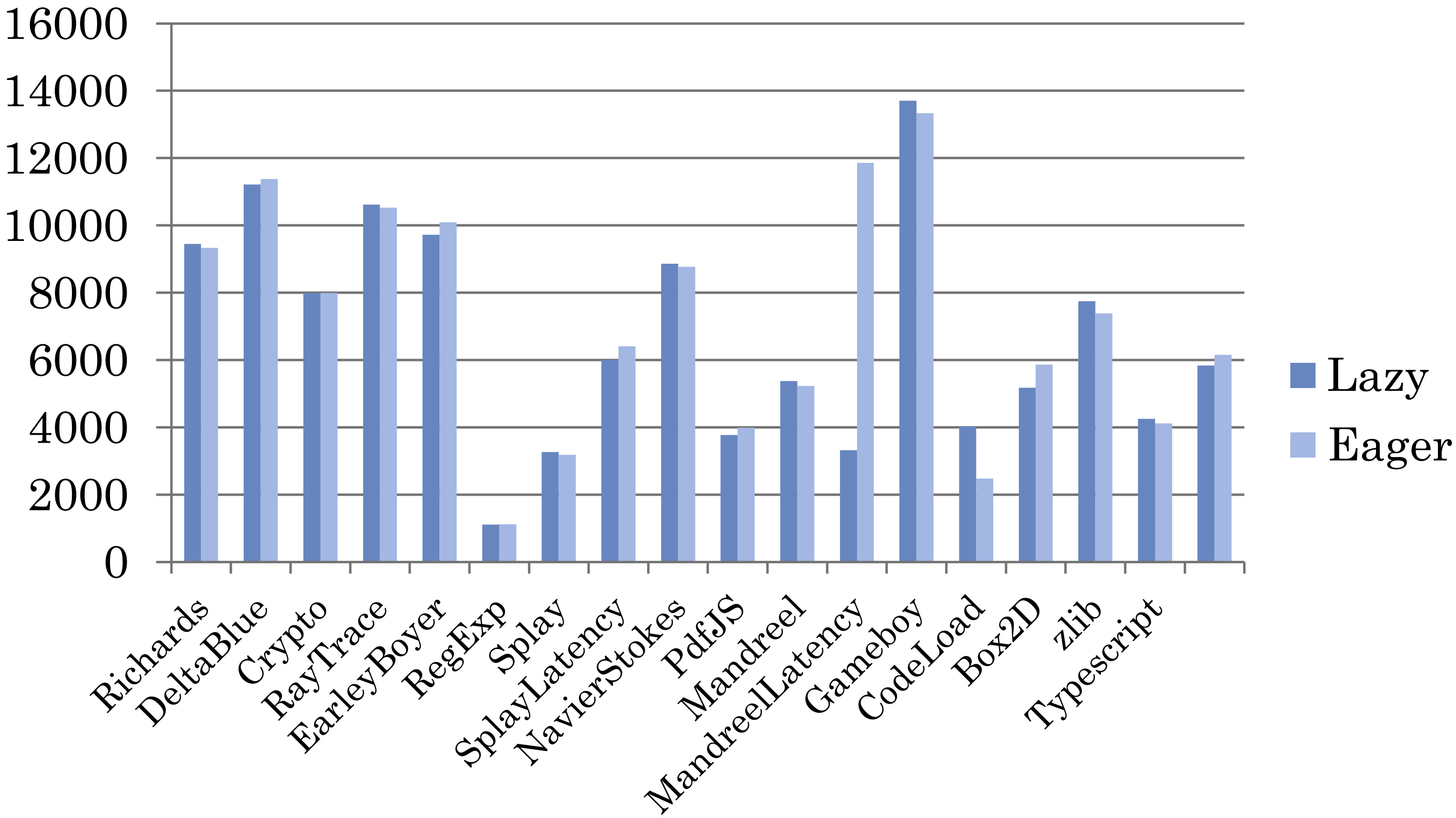
- Link-time optimization: **FAIL**
- Platform options tuning: **SUCCESS**
  - -O3 for highest optimization level
  - -mcpu=cortex-a15 for target CPU
- …





RUNTIME PARAMETERS TUNING

Octane score: **↑5.4%**



SCALAR OPTIMIZATIONS

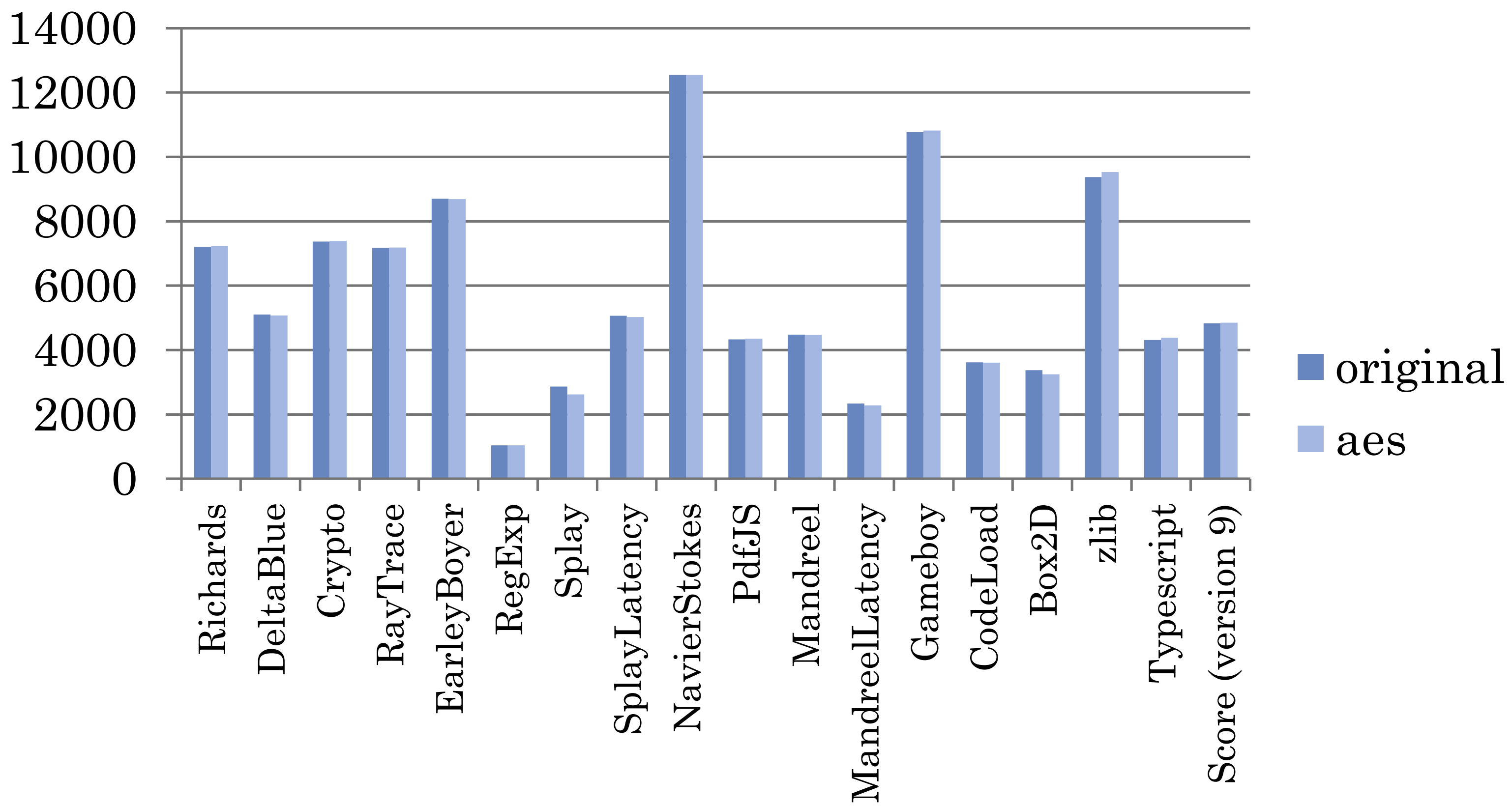
- Algebraic Expression Simplification
  - uses algebraic identities to simplify expressions:

$$x + 1 = x, \; y * 1 = y, \; z \mid 0 = z$$

- Common Subexpression Elimination
  - path sensitive elimination of common subexpression
- Fast call frame
  - use ARMv7 specific call frame

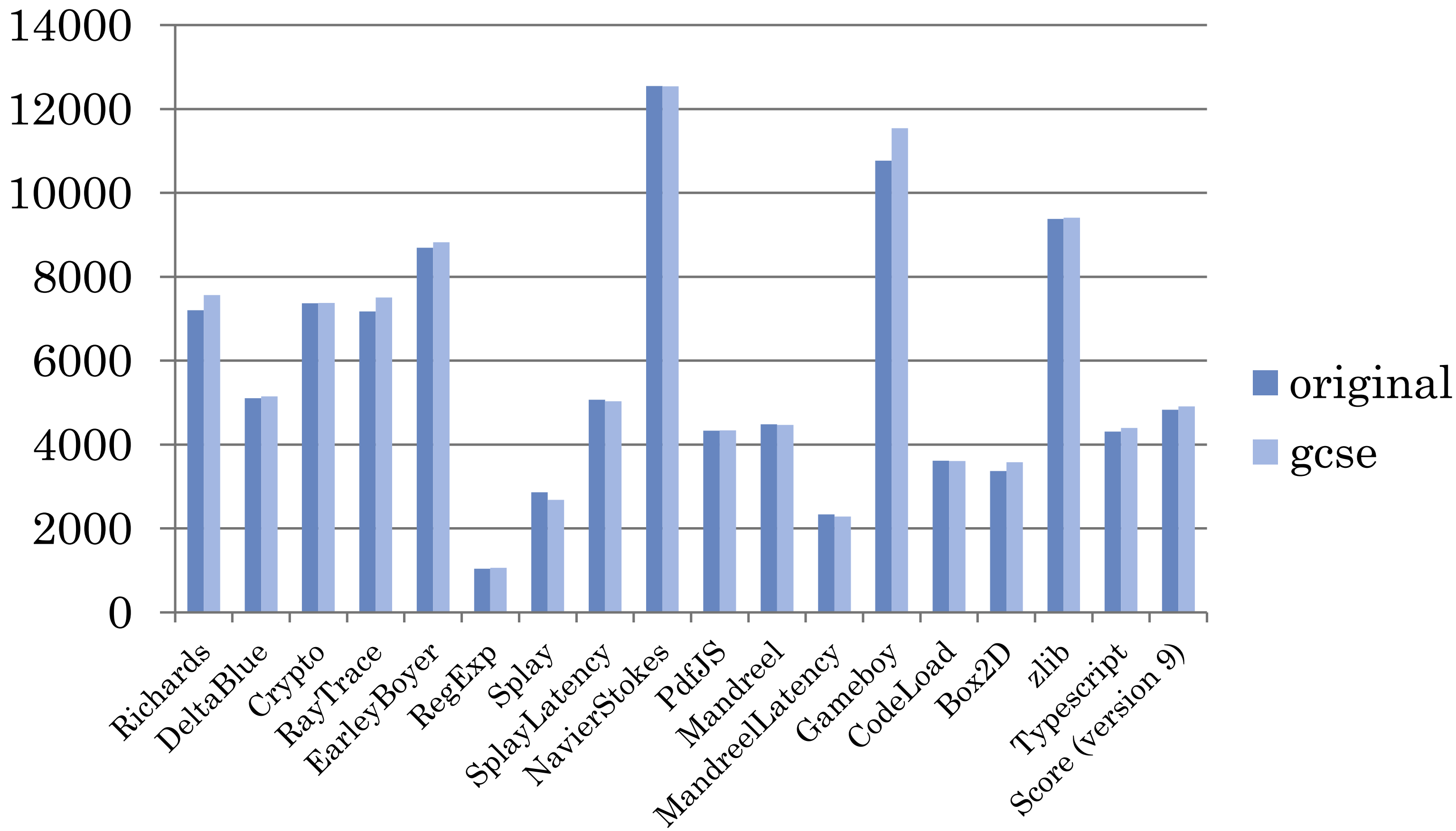
ALGEBRAIC EXPRESSION SIMPLIFICATION

Octane score: ↑**0.3%**



COMMON SUBEXPRESSION ELIMINATION

Octane score: ↑**1.8%**



FAST CALL FRAME

Prologue:

func:

stmdb sp!, {r4-r5, fp, lr}

add fp, sp, #N

Epilogue:

mov sp, fp

ldmia sp!, {r4-r5, fp, lr}

bx lr

Prologue:

func:

sub sp, sp, #16

stm sp, {r4,r5,fp, lr}

add fp, sp, #N

Epilogue:

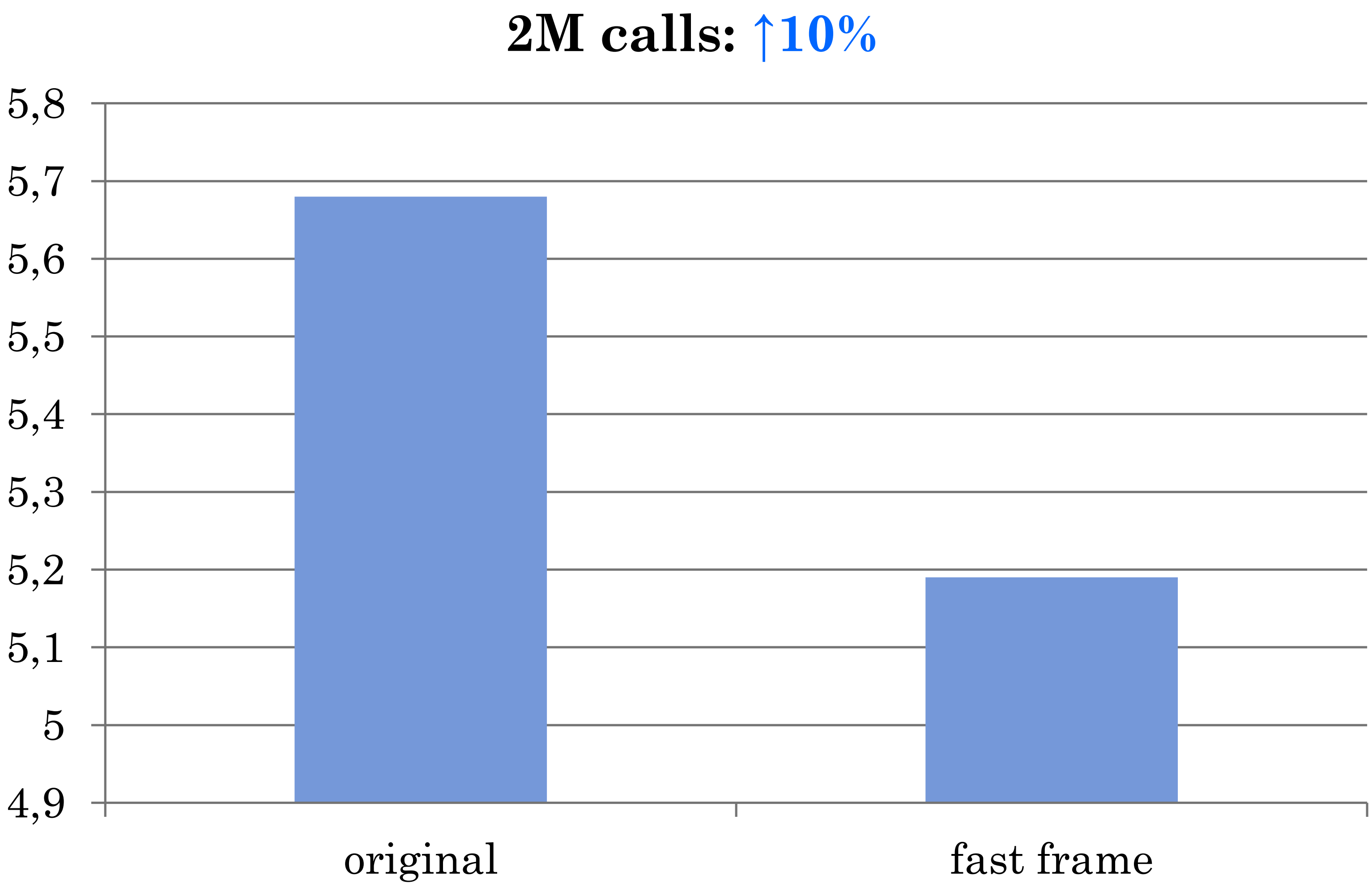
mov sp, fp

ldm sp, {r4, r5, fp, lr}

add sp, sp, #16

bx lr

FAST CALL 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