Optimizing JShelter performance

Martin Zmitko

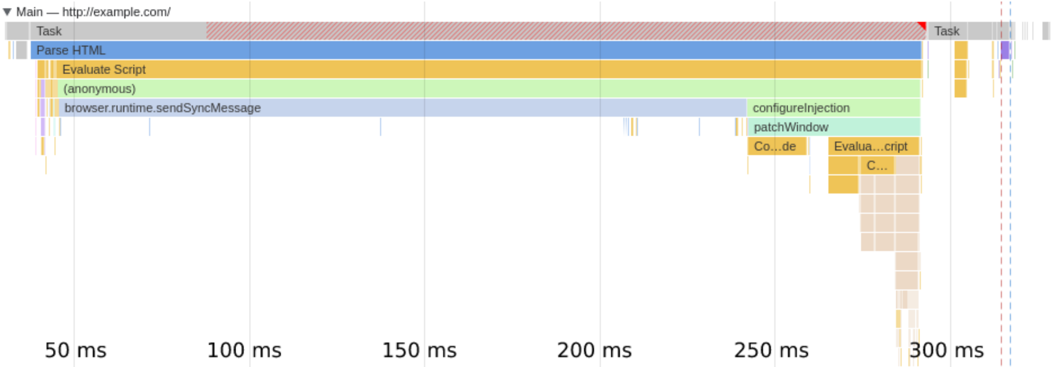
Supervisor: Ing. Radek Hranický, Ph.D.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

May 23, 2023

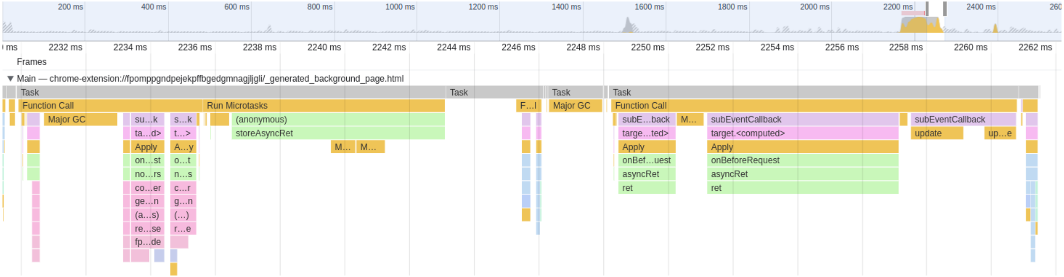
Chrome – Regular Injection Performance

- On almost every load
- 250 ms performance hit
- Slow SyncMessage – 180 ms on every load



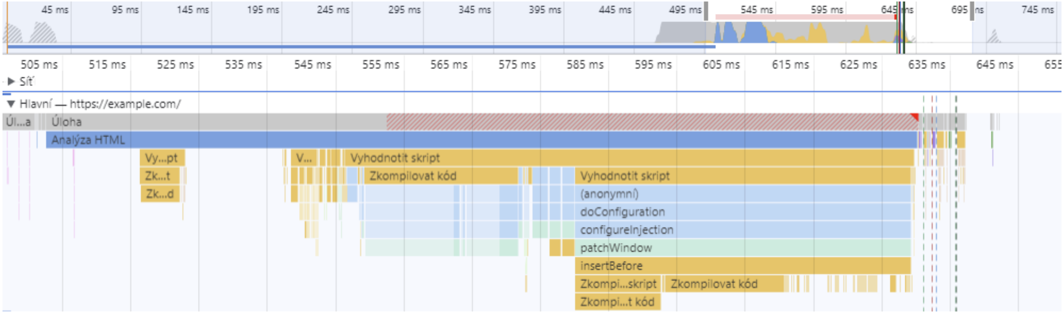
SyncMessage Performance

- Large payload size – 700 kB
- Necessary serialization and deserialization
- 30 ms spent by background script handler
- Slow internal browser processing
- Linear execution time increase



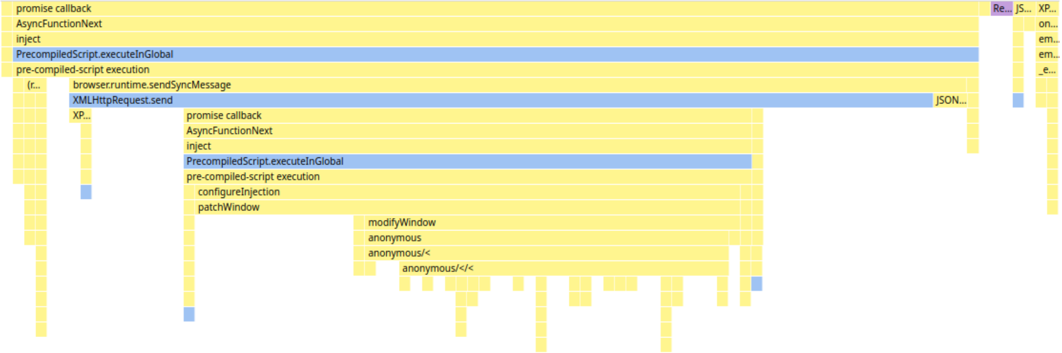
Chrome – Early Injection Performance

- Uncommon
- 80 ms performance hit
- 10 ms for evaluation, 20 ms added to total time



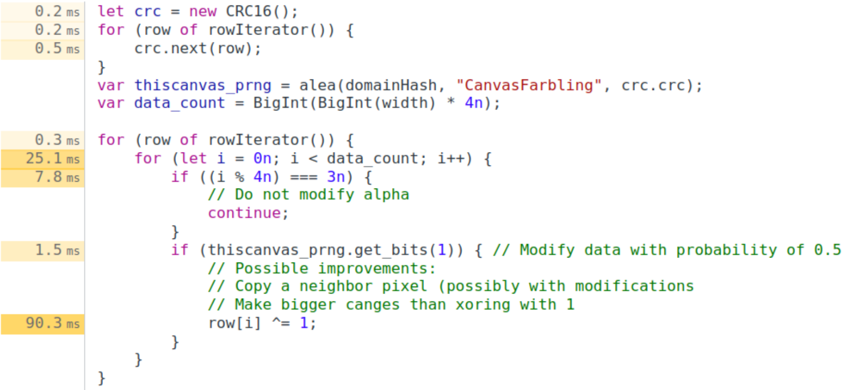
Firefox Injection Performance

- Always the same
- 80 ms performance hit
- Injected as a content script – patchWindow executes during SyncMessage handling
- Necessary to complete request, additional 20 ms



Wrapper Performance

- Small performance hit for all
- Most couldn´t be further optimized
- Large performance hit on farbling
- Inefficient iteration



```
  0.2 ms  let crc = new CRC16();
  0.2 ms  for (row of rowIterator()) {
  0.5 ms      crc.next(row);
          }
          var thiscanvas_prng = alea(domainHash, "CanvasFarbling", crc.crc);
          var data_count = BigInt(BigInt(width) * 4n);

  0.3 ms  for (row of rowIterator()) {
 25.1 ms      for (let i = 0n; i < data_count; i++) {
  7.8 ms          if ((i % 4n) === 3n) {
                      // Do not modify alpha
                      continue;
                  }
  1.5 ms          if (thiscanvas_prng.get_bits(1)) { // Modify data with probability of 0.5
                      // Possible improvements:
                      // Copy a neighbor pixel (possibly with modifications
                      // Make bigger canges than xoring with 1
 90.3 ms              row[i] ^= 1;
                  }
              }
          }
```

Injection optimization

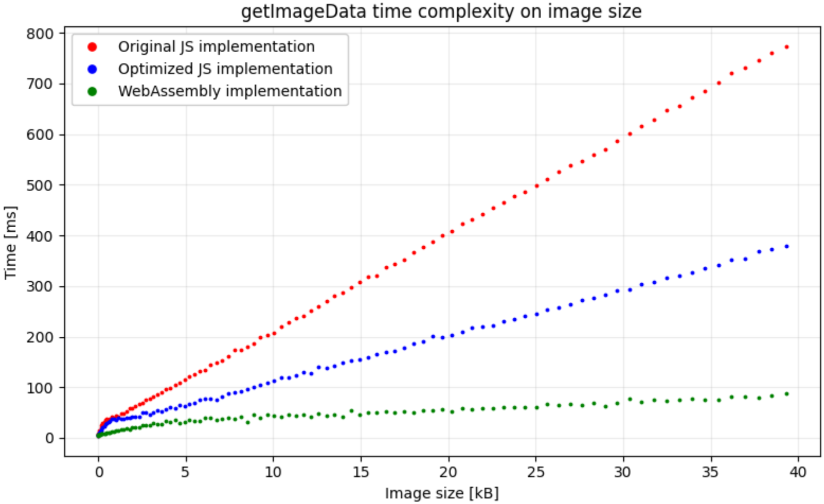
- Decrease SyncMessage payload size – don’t send code
- Split configuration and code generation logic
- Move code generation to content scripts
- Generate code in `document_start.js`
- Wrapper definition evaluation adds 10 ms, code generation takes 15 ms
- Final SyncMessage payload size is 12 kB and executes under 10 ms
- Code size optimizations

WebAssembly optimized farbling

- Allows efficient data processing
- Subject to CSP on Chrome
- Modify CPS headers, adjustable level
- Inconsistent initialization
- JS implementation used as a fallback, optimized implementation must always provide same results
- Not subject to CSP on Firefox, possible to use WebAssembly only
- Reimplemented Canvas, WebGL and WebAudio farbling in AssemblyScript – TypeScript syntax, for compiling into WebAssembly
- Differences between number types and operations
- Automatic build process
- Unit tests
- Known bug: floating point CRC provides different results

Optimized canvas farbling measurement

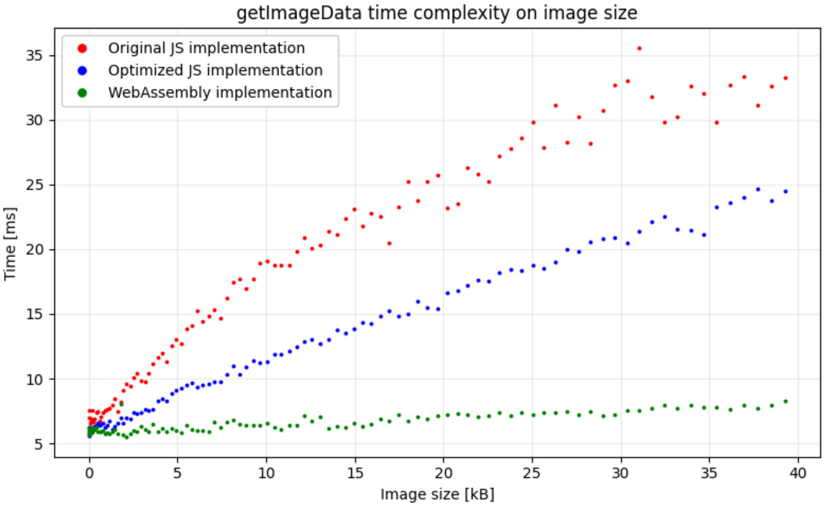
Measured on Chrome for square canvas with data in range 0.4-4000 kB, 5.3 times



faster

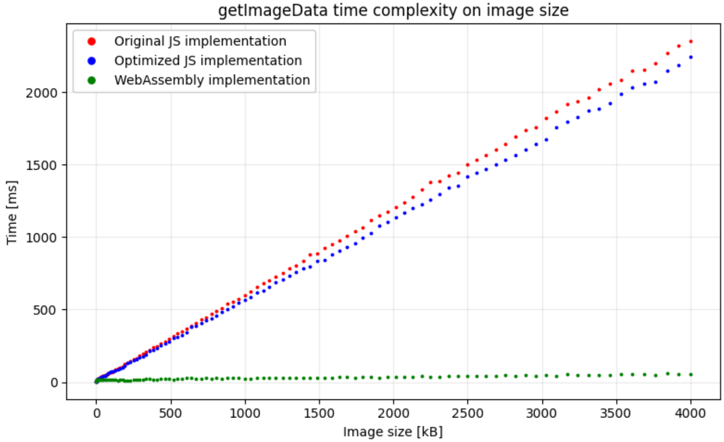
Optimized canvas farbling measurement detail

Measured on Chrome for square canvas with data in range 40 B to 40 kB



Optimized canvas farbling measurement

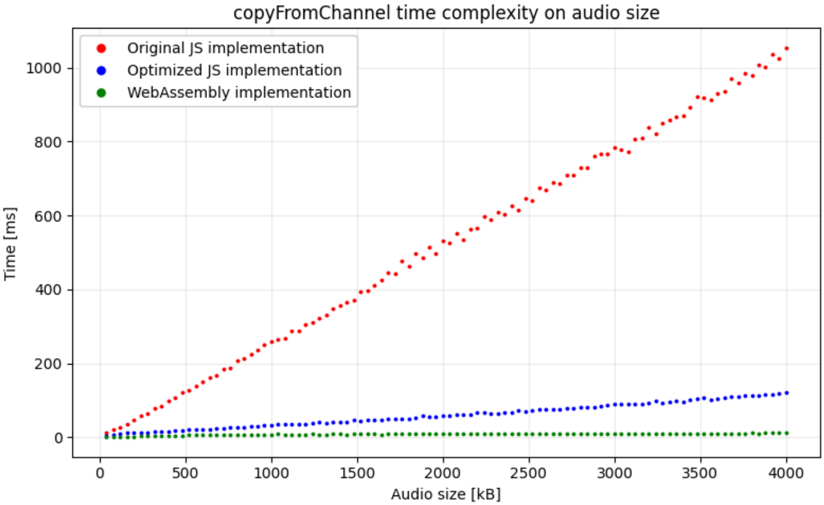
Measured on Firefox for square canvas with data in range 0.4-4000 kB, 53 times



faster

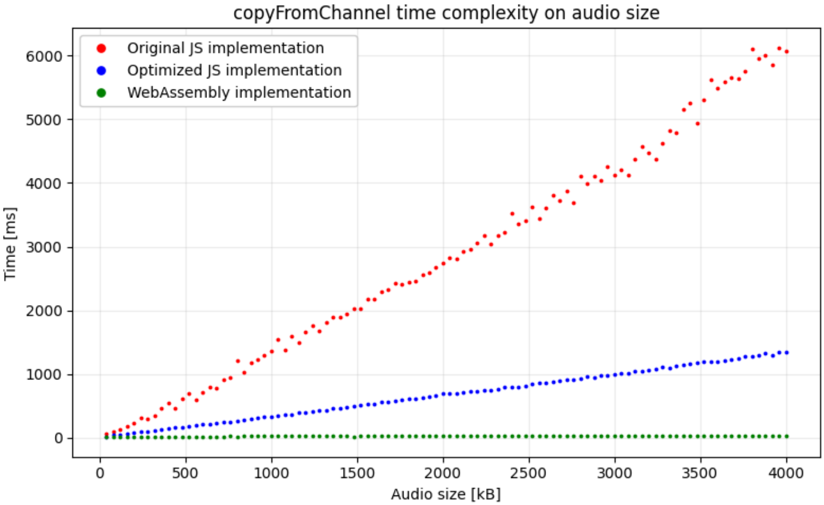
Optimized audio farbling measurement

Measured on Chrome for audio in range 0.4-4000 kB



Optimized audio farbling measurement

Measured on Firefox for audio in range 0.4-4000 kB



Lighthouse loading analysis

- Tool for measuring user percieved loading performance
- No similiar tools found
- Implemented own CLI tool for collecting performance data on set URLs with set extensions
- JSON output for analysis
- Measured on 50 top Tranco domains
- Performance of clean browser was 83.5, original JShelter 69.2 and optimized JShelter 78.5
- That is 13,5% increase from original version
- Original version decreased performance by 17,2 %, optimized version decreased performance just by 6,1 %