McNumJS: Enabling Fast Numeric Computation for JavaScript

Sujay Kathrotia,
Laurie Hendren
Sable Lab, McGill University
Agenda

- Motivation
- Introduction
- Typed Arrays
- Asm.JS
- Performance
Motivation

RUN IT ANY DEVICE

JS
Motivation

- High performance numeric application
- Scientist/Engineers to write fast numeric application
- Compiler writers compiling Scientific Languages like MATLAB/R to JavaScript
Introduction

- API similar to NumPy

- McNumJS modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Example Properties or methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>shape, stride, get, set, index</td>
</tr>
<tr>
<td>Generation</td>
<td>zeroes, ones, random</td>
</tr>
<tr>
<td>Unary Operations</td>
<td>fill, sum, log, transpose, sin, cos</td>
</tr>
<tr>
<td>Binary Operations</td>
<td>add, subtract, multiply, divide</td>
</tr>
</tbody>
</table>
Introduction

- Performance improvement
  - Typed arrays
  - Type coercion similar to asm.js
Typed Arrays

- Provide a mechanism for accessing raw binary data.
- Typed array architecture: Buffers and views

```javascript
var buffer = new ArrayBuffer(8); // Allocates 8 bytes
var intView = new Int16Array(buffer); // [0,0,0,0] : Int16
var floatView = new Float64Array(buffer); // [0] : Float64
```
Typed Arrays

- **Advantages:**
  - Facilitate typing in JavaScript
  - Fast Access compared to regular arrays

- **Disadvantages:**
  - Initialization is expensive
  - Cannot grow or shrink dynamically
  - Only one dimension
McNumJS Typed Arrays

- Override View Object to facilitate dimensions
- Add Properties: shape, stride, etc.
- Add Methods: get, set, etc.

```javascript
var intView = new Int32Array(64, [8, 8]); // 8x8 Int32 Matrix
console.log(intView.shape); // [8, 8]
console.log(intView.stride); // [8, 1]
console.log(intView.get(3, 2)); // 0 // 3*8+2*1 = 26
```
Asm.js

- Low level, strict subset of JavaScript for compilation target
- Asm.js is defined by static typing
- Allows ahead-of-time (AOT) optimizing compilation

```javascript
x = x|0;    // x is of type integer
y = +y;     // y is of type double
```
function AsmMath(stdlib, foreign, buffer) {
    "use asm";
    /* Globals */
    var values = new stdlib.Float64Array(buffer);
    /* Module body */
    function add(v) {
        v = +v;                  // double
        var i=0, l=0;            // int
        for (i = 0, l = values.length << 3; (i|0) < (l|0); i=(i+8)|0) {
            values[i>>3] = +(values[i>>3] + v);
        }
    }
    /* Export */
    return {
        add: add
    };
}
Asm.js

- Non-trivial to write manually
  - Manual memory management using single heap
  - Byte addressing
  - Restricted subset of standard JavaScript library
  - No bounds checking

- Use static typing rules to allow better JIT compilation

```javascript
function add(v) {
    v = +v; // double
    var i=0; // int
    var l= values.length>>>0; // uint
    for (; i < l; (++i)|0) {
        values[i] = +(values[i] + v);
    }
}
```
Performance

- **Ostrich Benchmark suite**
  - Using JavaScript and WebCL for numerical computations: a comparative study of native and web technologies (DLS’14)
    (Faiz Khan, Vincent Foley-Bourgon, Sujay Kathrotia, Erick Lavoie, Laurie Hendren McGill University)
  - 12 Benchmarks with C, JS with typed arrays, JS with no-typed arrays, Asm.js, OpenCL, WebCL implementations

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Core i7 @3.2GHz x12</td>
</tr>
<tr>
<td>Operating System</td>
<td>64-bit Ubuntu 12.04</td>
</tr>
<tr>
<td>Physical Memory</td>
<td>16 GiB</td>
</tr>
<tr>
<td>Browsers</td>
<td>Chrome v38, Firefox v33</td>
</tr>
</tbody>
</table>
Performance: McNumJS vs. C

- Slowdown of McNumJS compared to C
- Average slowdown of 1.915, 1.831 on Chrome and Firefox
Performance: McNumJS vs. Asm.js

- Slowdown of McNumJS compared to Asm.js
- Average slowdown of 1.376, 1.264 on Chrome and Firefox
Performance: McNumJS vs. JS

- Slowdown of McNumJS compared to JS with TypedArrays (manual indexing)
- Average slowdown of 1.128, 1.002 on Chrome and Firefox
Performance: JS-NOTA Vs. McNumJS

- Speedup of McNumJS compared to JavaScript with Regular Arrays with single dimension (manual indexing)
- Average speedup of 1.254, 2.009 on Chrome and Firefox
Conclusion

- Easy to use API similar to NumPy
- Extend typed arrays to support multi-dimension
- Good performance by using typed arrays and type coercion
- Performance is within 2x of native C
Thank you!

http://www.sable.mcgill.ca/mclab/projects/mcnumjs
Backup slides

I KNOW YOU GOT QUESTIONS
JavaScript Performance

![Graph showing slowdown compared to Native C for Chrome, Firefox, and Safari.](image-url)
## JavaScript vs. C

### JavaScript (Typed Arrays)

<table>
<thead>
<tr>
<th>Machine</th>
<th>Chrome</th>
<th>Firefox</th>
<th>Safari</th>
<th>IE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>1.62</td>
<td>1.65</td>
<td>-</td>
<td>3.11</td>
</tr>
<tr>
<td>Mac</td>
<td>1.51</td>
<td>1.55</td>
<td>2.69</td>
<td>-</td>
</tr>
</tbody>
</table>

### asm.js

<table>
<thead>
<tr>
<th>Machine</th>
<th>Chrome</th>
<th>Firefox</th>
<th>Safari</th>
<th>IE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>1.35</td>
<td>1.17</td>
<td>-</td>
<td>2.81</td>
</tr>
<tr>
<td>Mac</td>
<td>1.37</td>
<td>1.11</td>
<td>2.15</td>
<td>-</td>
</tr>
</tbody>
</table>

### Special Cases:

<table>
<thead>
<tr>
<th>Benchmark(s)</th>
<th>Browser</th>
<th>Machine</th>
<th>Issue</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>lavamd</td>
<td>Chrome</td>
<td>Windows</td>
<td>4x faster than c</td>
<td>Exponential function is faster in Chrome than GCC</td>
</tr>
<tr>
<td>crc</td>
<td>IE</td>
<td>Windows</td>
<td>35x slower than c</td>
<td>Missed performance optimization</td>
</tr>
<tr>
<td>crc, fft</td>
<td>Safari</td>
<td>Mac</td>
<td>crc: 17x slower ft: 10x slower</td>
<td>Missed performance optimization</td>
</tr>
</tbody>
</table>