McNumJS: Enabling Fast Numeric Computation for JavaScript

Sujay Kathrotia,

Laurie Hendren

Sable Lab, McGill University

Agenda

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

Motivation

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:

Module	Example Properties or methods
Core	shape, stride, get, set, index
Generation	zeroes, ones, random
Unary Operations	fill, sum, log, transpose, sin, cos
Binary Operations	add, subtract, multiply, divide

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

```
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
```

buffer

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.

```
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

```
x = x|0;     // x is of type integer
y = +y;      // y is of type double
```

Asm.js

```
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

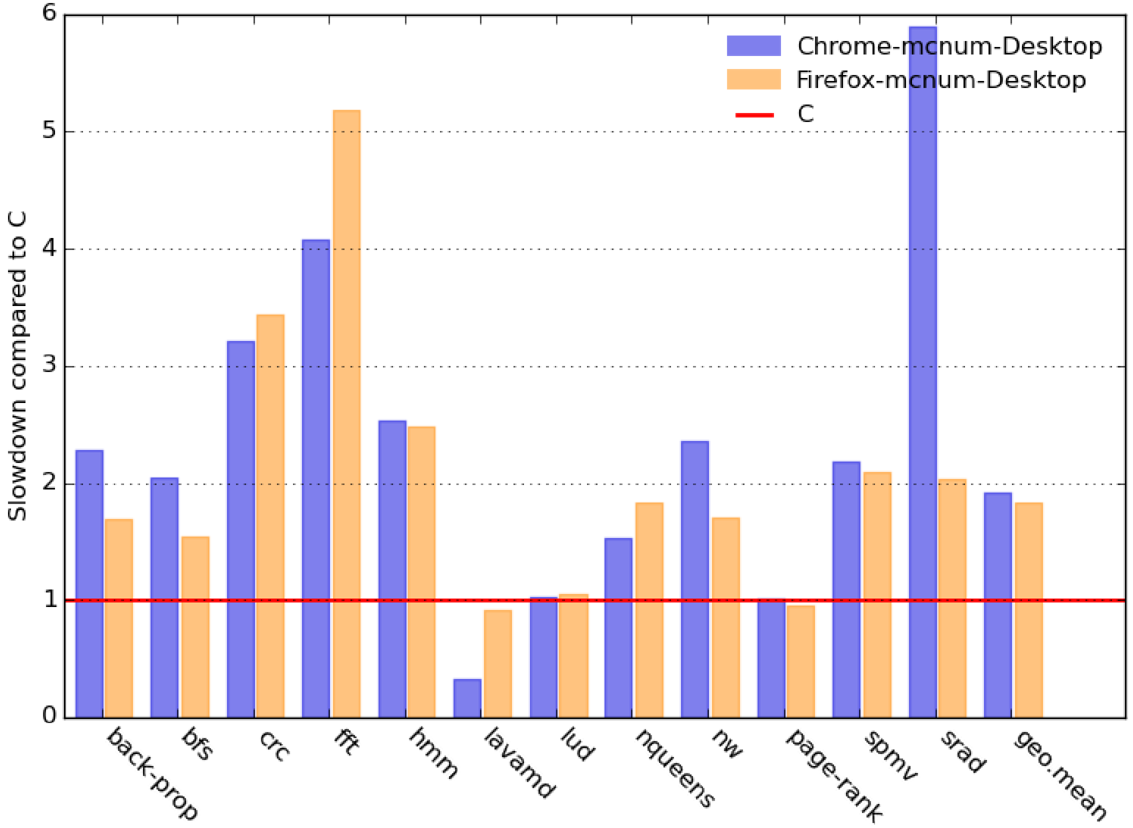
```
function add(v) {
    v = +v;    // double
    var i=0;   // int
    var l= values.length>>>0; // uint
    for (; i < l; (++i)|0) {
        values[i] = +(values[i] + v);
    }
}
```

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

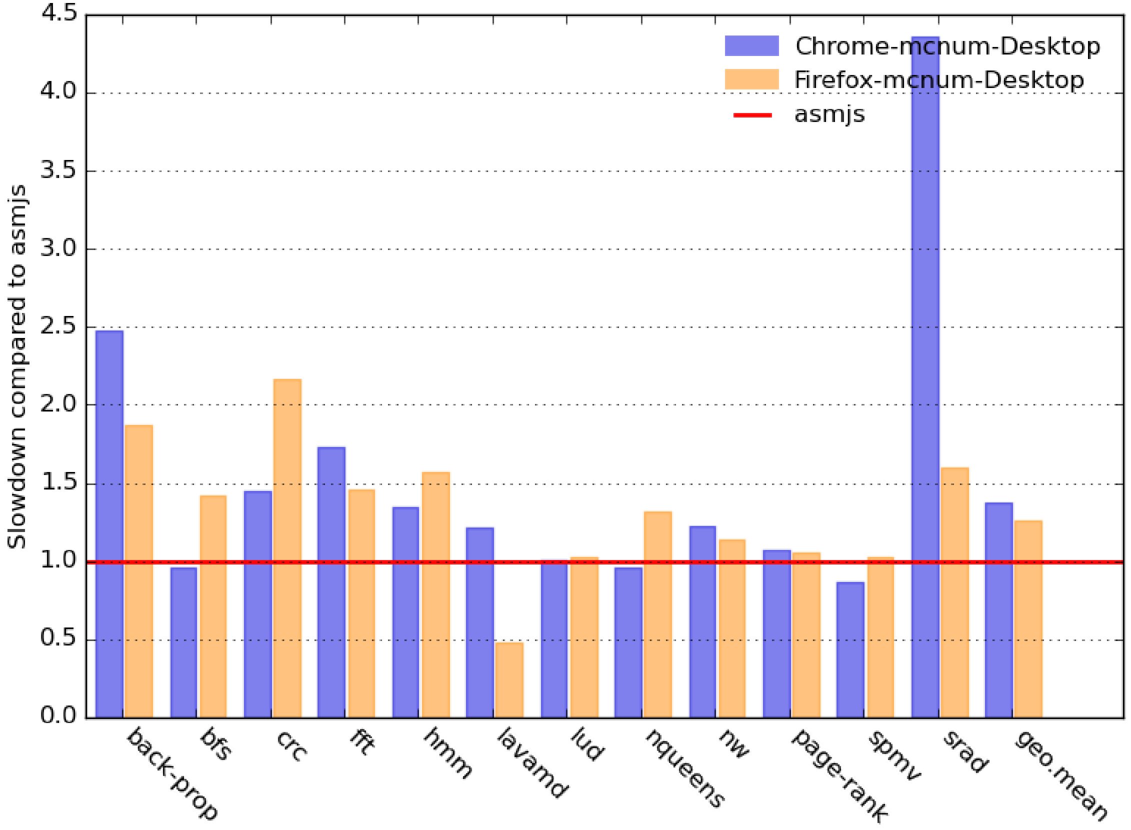
Processor	Intel Core i7 @3.2GHz x12
Operating System	64-bit Ubuntu 12.04
Physical Memory	16 GiB
Browsers	Chrome v38, Firefox v33

Performance: McNumJS vs. C



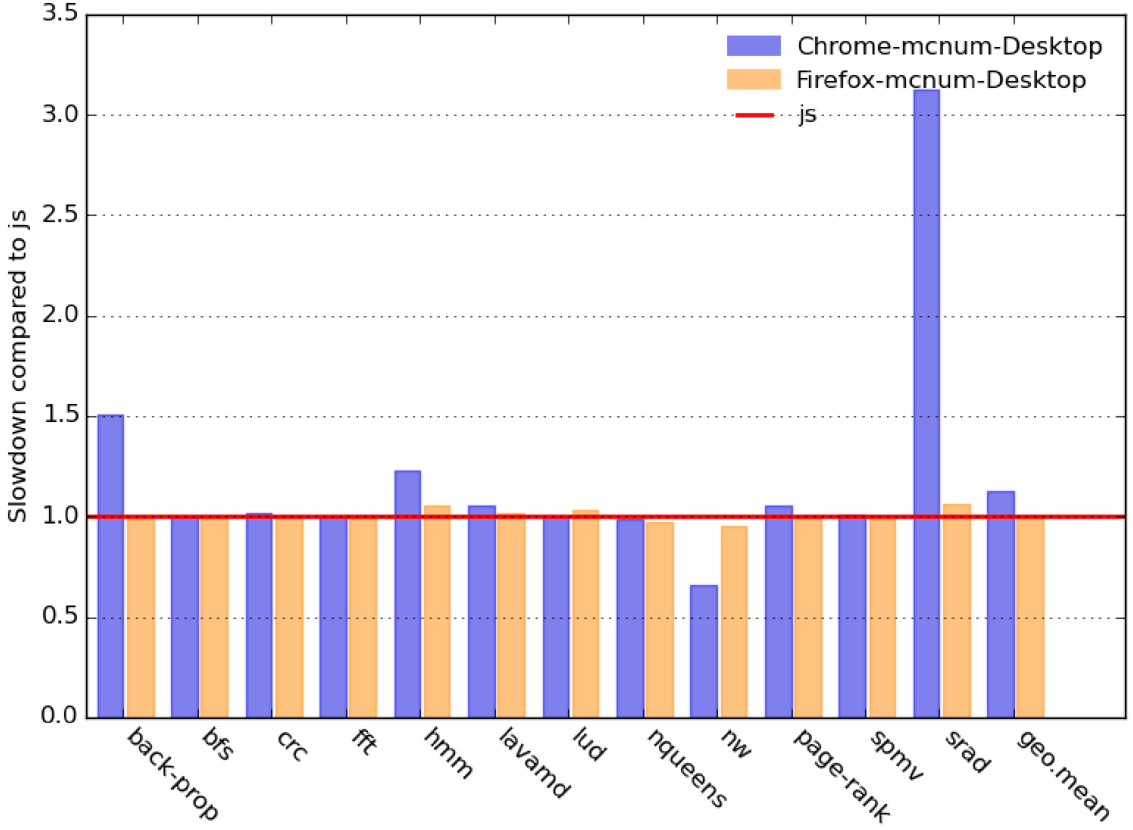
- Slowdown of McNumJS compared to C
- Average slowdown of 1.915, 1.831 on Chrome and Firefox

Performance: McNumJS vs. Asm.js



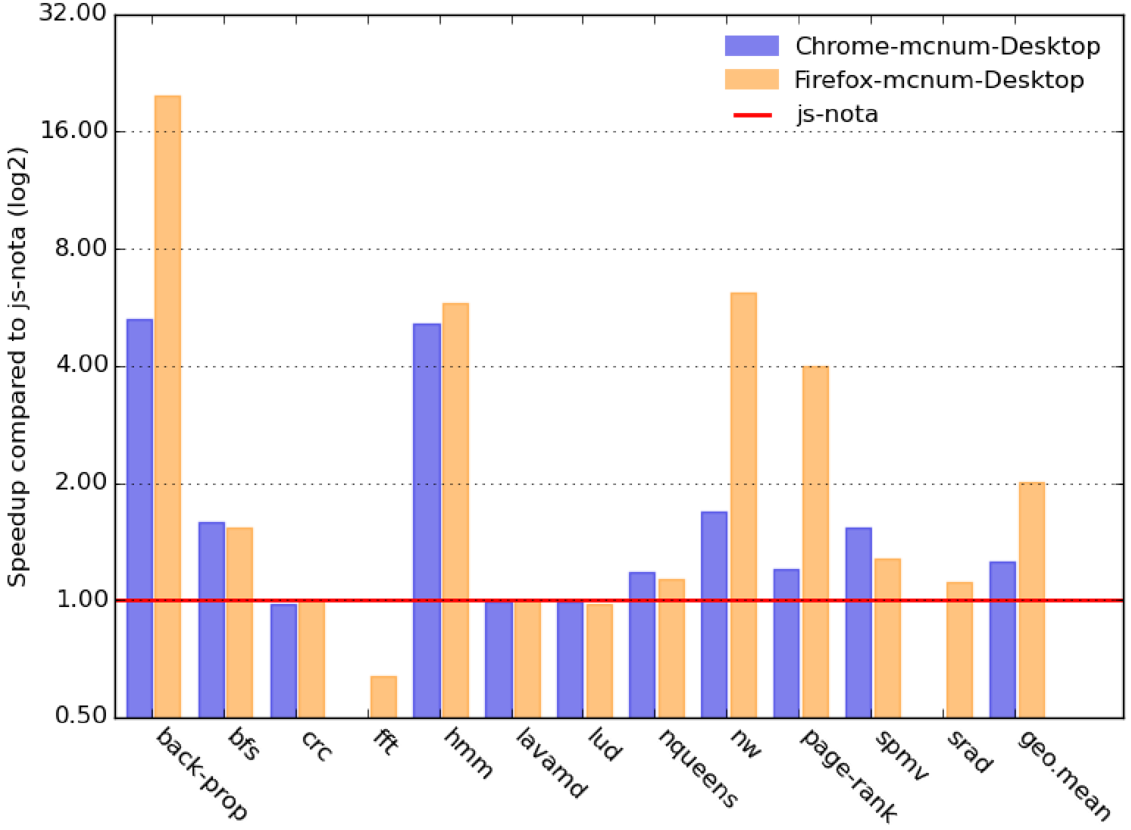
- Slowdown of McNumJS compared to Asm.js
- Average slowdown of 1.376, 1.264 on Chrome and Firefox

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

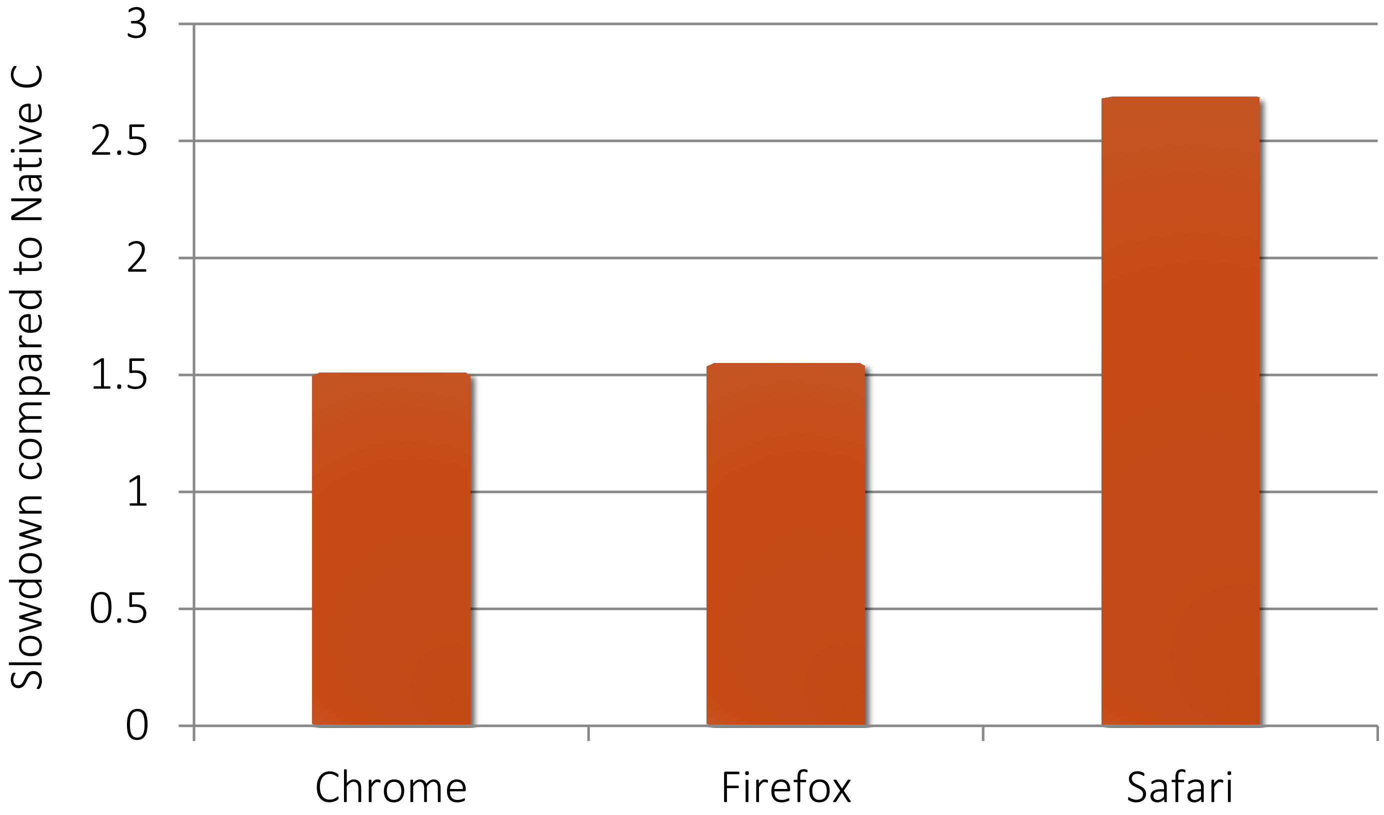
<Thank you!>

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

Backup slides

I KNOW YOU GOT QUESTIONS

JavaScript Performance



JavaScript vs. C

JavaScript (Typed Arrays)

	Chrome	Firefox	Safari	IE
Windows	1.62	1.65	-	3.11
Mac	1.51	1.55	2.69	-

asm.js

	Chrome	Firefox	Safari	IE
Windows	1.35	1.17	-	2.81
Mac	1.37	1.11	2.15	-

Special Cases:

Benchmark(s)	Browser	Machine	Issue	Reason
lavamd	Chrome	Windows	4x faster than c	Exponential function is faster in Chrome than GCC
crc	IE	Windows	35x slower than c	Missed performance optimization
crc, fft	Safari	Mac	crc: 17x slower fft: 10x slower	Missed performance optimization

JavaScript Competitive Against C