

# Sparse matrices on the web : Characterizing the performance and optimal format selection of sparse matrix-vector multiplication in JavaScript and WebAssembly

Prabhjot Sandhu, *David Herrera*, and Laurie Hendren

Sable Research Group  
McGill University



September 6, 2018

# Outline

- 1 Introduction
- 2 Experimental Design
- 3 Can managed web languages' performance come closer to native C?
- 4 RQ2 : Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?
- 5 RQ3 : If the best storage format for C is known, will it be the best format for web languages too?
- 6 Summary and Future Work

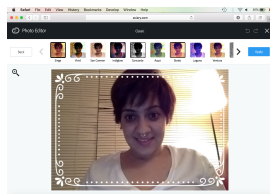
# Why Sparse Matrices on the Web?

- Web-enabled devices everywhere!
- Various compute-intensive applications involving sparse matrices on the web.
  - Image editing
  - Text classification (data mining)
  - Deep learning
- Recent addition of WebAssembly to the world of JavaScript.



# Why Sparse Matrices on the Web?

- Web-enabled devices everywhere!
- Various compute-intensive applications involving sparse matrices on the web.
  - Image editing
  - Text classification (data mining)
  - Deep learning
- Recent addition of WebAssembly to the world of JavaScript.



# Why Sparse Matrices on the Web?

- Web-enabled devices everywhere!
- Various compute-intensive applications involving sparse matrices on the web.
  - Image editing
  - Text classification (data mining)
  - Deep learning
- Recent addition of WebAssembly to the world of JavaScript.



# Background : Sparse Matrix Formats

- **A sparse matrix** : a matrix in which most of the elements are zero.
- **Basic sparse storage formats** :
  - Coordinate Format (COO)
  - Compressed Sparse Row Format (CSR)
  - Diagonal Format (DIA)
  - ELLPACK Format (ELL)

A

1	0	6	0
0	2	0	7
0	0	3	0
5	0	0	4

COO Format :

row	0	0	1	1	2	3	3
col	0	2	1	3	2	0	3
val	1	6	2	7	3	5	4

CSR Format :

row_ptr	0	2	4	5	7		
col	0	2	1	3	2	0	3
val	1	6	2	7	3	5	4

DIA Format :

	val	offset
X	1 6	-3 0 2
X	2 7	
X	3 X	
5	4 X	

ELL Format :

val	indices
1 6	0 2
2 7	1 3
3 X	2 X
5 4	0 3

## WebAssembly

- A typed low-level bytecode representation.
- Introduced to enable better performance.

## Sparse Matrix Vector Multiplication (SpMV)

- Computes  $y = Ax$ , where matrix  $A$  is sparse and vector  $x$  is dense.
- A performance-critical operation.
- Choice of storage format (data structure) matters.
- Depends on the structure of the matrix, machine architecture and language of implementation.



# This Paper

We explored the performance and choice of optimal sparse matrix storage format for **sequential SpMV** for both **JavaScript** and **WebAssembly**, as compared to **C** through the following three research questions :

## RQ1

Can managed web languages' performance come closer to native C?

## RQ2

Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?

## RQ3

If the best storage format for C is known, will it be the best format for web languages too?

# Outline

- 1 Introduction
- 2 Experimental Design**
- 3 Can managed web languages' performance come closer to native C?
- 4 RQ2 : Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?
- 5 RQ3 : If the best storage format for C is known, will it be the best format for web languages too?
- 6 Summary and Future Work

# Reference Implementations

Developed a reference set of sequential C and JavaScript implementations of SpMV for different formats on same algorithmic lines.

```
void spmv_coo(int *coo_row, int *coo_col, MYTYPE *coo_val, int nz, int N, MYTYPE *x,
             MYTYPE *y)
{ int i;
  for(i = 0; i < nz ; i++)
    y[coo_row[i]] += coo_val[i] * x[coo_col[i]];
}
```

Listing 1: SpMV COO reference C implementation

```
\\ efficient representation, using typed arrays
var coo_row = new Int32Array(nz)
var coo_col = new Int32Array(nz)
var coo_val = new Float32Array(nz)
var x = new Float32Array(cols)
var y = new Float32Array(rows);

\\ note the use of Math.fround in the loop body
function spmv_coo(coo_row, coo_col, coo_val, N, nz, x, y)
{
  for(var i = 0; i < nz; i++)
    y[coo_row[i]] += Math.fround(coo_val[i] * x[coo_col[i]]);
}
```

Listing 2: SpMV COO reference JavaScript implementation

# Reference C versus Intel MKL and Python Sparse.SciPy

		COO		CSR		DIA	
		n	Speedup	n	Speedup	n	Speedup
MKL	single	97	1.04	221	0.76	103	0.97
	double	49	1.09	174	1.078	22	0.92
SciPy	single	122	0.95	399	1.03	32	2.28
	double	53	0.96	790	1.09	23	1.90

Table: Speedup of the reference C implementation versus Intel MKL and Python SciPy (greater than 1 means our implementation performs better than the corresponding library implementation)

The performance of our implementation is close to both Intel MKL and Python SciPy, in most cases.

# Target Languages and Runtime

- **Machine Architecture**

Intel Core i7-3930K with 12 3.20GHz cores, 12MB last-level cache and 16GB memory, running Ubuntu Linux 16.04.2

- **C**

Compiled with gcc version 7.2.0 at optimization level -O3

- **JavaScript**

Used the latest browsers Chrome 66 (Official build 66.0.3359.139 with V8 JavaScript engine 6.6.346.26) and Firefox Quantum (version 59.0.2)

- **WebAssembly**

Automatically generated from C using Emscripten version 1.37.36, with optimization flag -O3.

# Measurement Setup

- **Benchmarks** : Around 2000 real-life sparse matrices from The SuiteSparse Matrix Collection.
- **Sparse Storage Formats** : COO, CSR, DIA, ELL
- Measured SpMV execution time for C, JavaScript and WebAssembly in GFLOPS.

A

1	0	6	0
0	2	0	7
0	0	3	0
5	0	0	4

COO Format :

row	0	0	1	1	2	3	3
col	0	2	1	3	2	0	3
val	1	6	2	7	3	5	4

CSR Format :

row_ptr	0	2	4	5	7		
col	0	2	1	3	2	0	3
val	1	6	2	7	3	5	4

DIA Format :

	val	offset
X	1 6	-3 0 2
X	2 7	
X	3 X	
5	4 X	

ELL Format :

val	indices
1 6	0 2
2 7	1 3
3 X	2 X
5 4	0 3

# How to choose the best format?

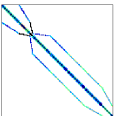
Input matrix	Graph	COO	CSR	DIA	ELL
CurlCurl_0		1.268 $\pm 0.027$	1.216 $\pm 0.029$	0.026 $\pm 0.0079$	1.161 $\pm 0.032$

Table: SpMV performance in GFLOPS for a matrix *CurlCurl\_0*.

Will you choose COO or CSR or ELL as the best format? What is your criteria? Did you consider the measurement error?

## Definition

We say that an input matrix  $A$  has an  $x\%$ -affinity for storage format  $F$ , if the performance for  $F$  is at least  $x\%$  better than all other formats and the performance difference is greater than the measurement error.

## Example

For example, if input array  $A$  in format CSR, is more than 10% faster than input  $A$  in all other formats, and 10% is more than the measurement error, then we say that  $A$  has a 10%-affinity for CSR.



# How to choose the best format?

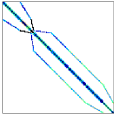
Input matrix	Graph	COO	CSR	DIA	ELL
CurlCurl_0		1.268 $\pm 0.027$	1.216 $\pm 0.029$	0.026 $\pm 0.0079$	1.161 $\pm 0.032$

Table: SpMV performance in GFLOPS for a matrix *CurlCurl\_0*.

For 10%-affinity criteria, we will choose a *combination-format* category, *COO-CSR-ELL* for this matrix. In this case, the matrix can be stored in any one of these formats for optimal performance.

# Outline

- 1 Introduction
- 2 Experimental Design
- 3 Can managed web languages' performance come closer to native C?**
- 4 RQ2 : Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?
- 5 RQ3 : If the best storage format for C is known, will it be the best format for web languages too?
- 6 Summary and Future Work

# RQ1 : JavaScript versus C

- *best-vs-best* : performance comparison of the best performing format in C and the best performing format in JavaScript.
- *best-vs-same* : performance comparison of the best performing format in C and the same format in JavaScript.

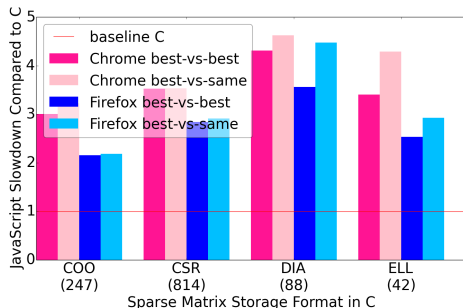


Figure: Slowdown of JavaScript relative to C for double-precision SpMV using the 10%-affinity criteria

# RQ1 : JavaScript versus C

## Observations

- Overall slowdown factor for JavaScript compared to C is less than 5.
- Firefox performs better than Chrome.

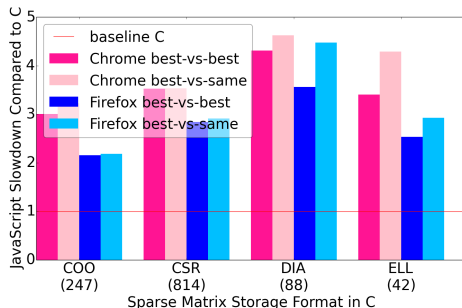


Figure: Slowdown of JavaScript relative to C for double-precision SpMV using the 10%-affinity criteria

# RQ1 : WebAssembly versus C

## Observations

- WebAssembly performs similar or better than C for Firefox.
- Overall slowdown factor for Chrome is around 2.

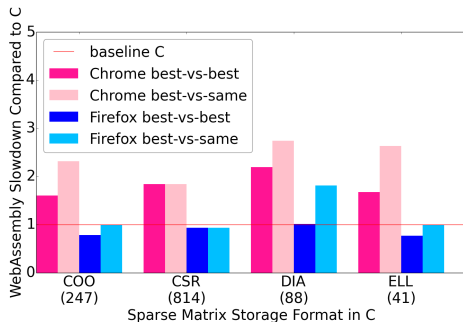


Figure: Slowdown of WebAssembly relative to C for double-precision SpMV using the 10%-affinity criteria

# RQ1 : JavaScript versus WebAssembly

## Observations

- WebAssembly performs significantly better than JavaScript.
- More performance improvement for Firefox from JavaScript to WebAssembly.

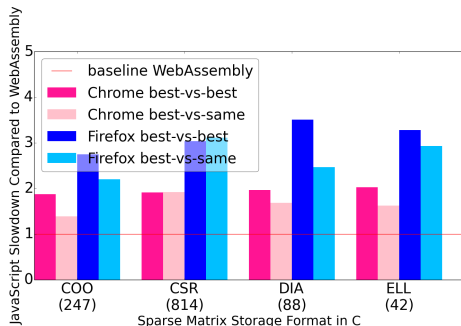


Figure: Slowdown of JavaScript relative to WebAssembly for double-precision SpMV using the 10%-affinity criteria

# Outline

- 1 Introduction
- 2 Experimental Design
- 3 Can managed web languages' performance come closer to native C?
- 4 RQ2 : Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?**
- 5 RQ3 : If the best storage format for C is known, will it be the best format for web languages too?
- 6 Summary and Future Work

## RQ2 : Performance Comparison between Single- and Double-precision for C

- In single-precision, a 32-bit number takes half the space compared to a 64-bit number in double-precision.
- Doubling the memory requirement for each floating-point number increases the load on cache and memory bandwidth.
- Effectiveness of SIMD (Single Instruction, Multiple Data) optimizations.

Format	n	GFLOPS	Speedup
		Single	
COO	212	1.03	1.08
		0.95	
CSR	366	1.88	1.08
		1.74	
DIA	90	3.59	1.96
		1.83	
ELL	18	1.44	1.21
		1.19	



## RQ2 : Performance Comparison between Single- and Double-precision for Chrome JavaScript

- Double-precision performs better than single-precision.
- JavaScript natively only supports double-precision.

Format	n	GFLOPS Single	Speedup
		Double	
COO	48	0.23	0.82
		0.28	
CSR	960	0.35	0.79
		0.44	
DIA	20	0.34	0.77
		0.44	
ELL	2	0.18	1.0
		0.18	

## RQ2 : Performance Comparison between Single- and Double-precision for Firefox WebAssembly

- Double-precision performs better than single-precision.
- WebAssembly natively supports both single- and double-precision.

Format	n	GFLOPS Single	Speedup
		Double	
COO	16	1.0	1.04
		0.96	
CSR	1002	1.41	0.82
		1.70	
DIA	0	-	-
		-	
ELL	8	1.17	0.86
		1.36	

# RQ2 : Format Difference between Single- and Double-precision for Firefox WebAssembly

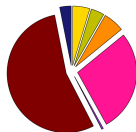
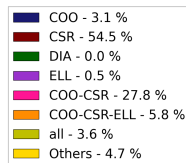


Figure: Single-precision for 10%-affinity

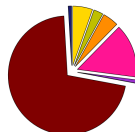
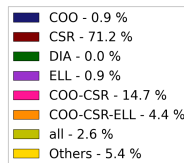


Figure: Double-precision for 10%-affinity

## Observations

- CSR format is prevalent for WebAssembly.
- None of the matrices have affinity for DIA format.

## RQ2 : Format Difference between Single- and Double-precision for C

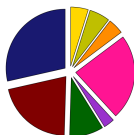
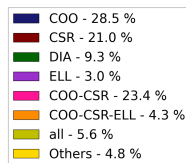


Figure: Single-precision for 10%-affinity

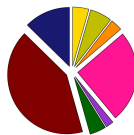
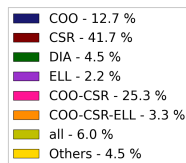


Figure: Double-precision for 10%-affinity

### Observations

- DIA format appears more important for single-precision as compared to double-precision.
- COO is more prevalent in single-precision (66.6%), while CSR is more prevalent in double-precision(80.8%).

# Outline

- 1 Introduction
- 2 Experimental Design
- 3 Can managed web languages' performance come closer to native C?
- 4 RQ2 : Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?
- 5 RQ3 : If the best storage format for C is known, will it be the best format for web languages too?**
- 6 Summary and Future Work

## RQ3 : JavaScript versus C

- Affinity greatly differs between C and JavaScript.
- SIMD optimizations in C make DIA to become the optimal format for some of the matrices.
- JavaScript lacks SIMD capabilities.

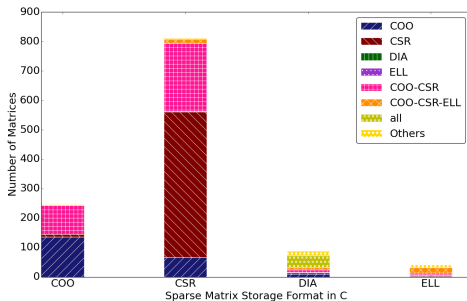


Figure: Affinity of matrices towards different format(s) for JavaScript relative to C using the 10%-affinity criteria for double-precision Firefox

# RQ3 : WebAssembly versus C

- CSR format takes precedence for WebAssembly.
- SIMD instruction set is in the future plans for WebAssembly.

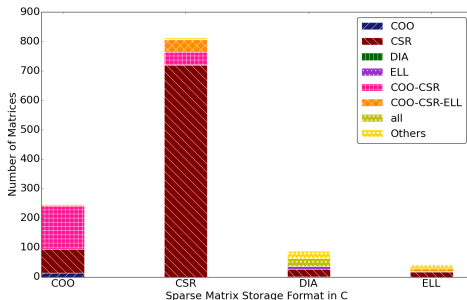


Figure: Affinity of matrices towards different format(s) for WebAssembly relative to C using the 10<sup>0</sup>-affinity criteria for double-precision Firefox

# Outline

- 1 Introduction
- 2 Experimental Design
- 3 Can managed web languages' performance come closer to native C?
- 4 RQ2 : Single-precision operations are usually faster than double-precision for C. Is it the case for web languages as well?
- 5 RQ3 : If the best storage format for C is known, will it be the best format for web languages too?
- 6 Summary and Future Work**



# Summary

- WebAssembly performs similar or better than C for Firefox, and overall slowdown factor for Chrome is around 2.
- WebAssembly performs at least 2x faster than JavaScript.
- Unlike C, double-precision SpMV is faster than single-precision in most cases for the web.
- The best format choices are different between C, JavaScript and WebAssembly, and also between the browsers.

# Takeaways

- Sequential SpMV on the web is reasonably performant.
- Realistic to utilize web-connected devices for compute-intensive applications using SpMV.
- Use WebAssembly for efficient kernel implementations.

# Future Work

- Explore the new optimization opportunities through hand-tuned WebAssembly implementations.
- Develop parallel versions of SpMV based on upcoming multithreading and SIMD features.
- Examine the impact of other factors like nnz, N, cache size etc. on SpMV performance.
- Develop automatic techniques to choose the best format for web-based SpMV.