Parallel JavaScript Update

Intel and Mozilla

Agenda

Work since January

Overview

- Design goals
- Key insight
- API overview and examples

Differences with sequential methods

Lessons learned

Use cases

Summary Since January

Major Focus was on implementation

- Use case driven performance tuning

Harvey Mudd collaboration

- Project implementing OpenCV algorithms

Demos rewritten to use typed object API

- Deform and Blocks

Spec language draft

Design Goals

Ease of use

- Deterministic where possible
- Follow current syntax, semantics, and security

Platform independent

- Support all kinds of platforms, parallel or not
- Perform well on different parallel architectures (multi-core, GPUs, SIMD Vectors…)

Extract reasonable performance out of parallel hardware

- Extracting all performance a secondary goal

You Can't Break the Web

Key Insight: Temporal Immutability

- During concurrent execution
  - A computation can read or write its local data
  - A computation can read shared state
  - Parent waits patiently
  - Whitelist thread-safe/temporally immutability primitives
  - Violations or best effort failure result in a sequential schedule
- Otherwise
  - Nothing changes
  - Current JavaScript programs are unaffected

The sweet spot between Functional and OO

Parallel JavaScript API (ES7)
(Stay within developer's comfort zone)

- Extend JavaScript's Array type and Typed Objects API
- High-level parallel methods
  - *mapPar, reducePar, filterPar, scatterPar, scanPar, buildPar*
- Elemental Functions

```
input.mapPar(
    e => { var avg = (e[0] + e[1] + e[2]) / 3;
           return [avg, avg, avg, 255];
         })
```

Simple Yet Powerful

Sum using reducePar

Sequential

```
var i;
var a = [1, 2, 3, 4];
var sum = 0;
for (i=0; i<a.length; i++) {
        sum += a[i];
}
```

Data parallel

```
var pa = [1, 2, 3, 4];
var sum = pa.reducePar(
     (a, b) => a+b
 ); // 10
```

PrefixSum

```
var prefixSum = pa.scanPar(
     (a, b) => a+b
 ); // [1,3,6,10]
```

buildPar, scatterPar, filterPar

Reverse

```
var pa = Array.buildPar(4, (i) => i); //[0,1,2,3]
var reversedPa = pa.scatterPar(
     (e, index, c) => c.length – index – 1
 );                                          //[3,2,1,0]
```

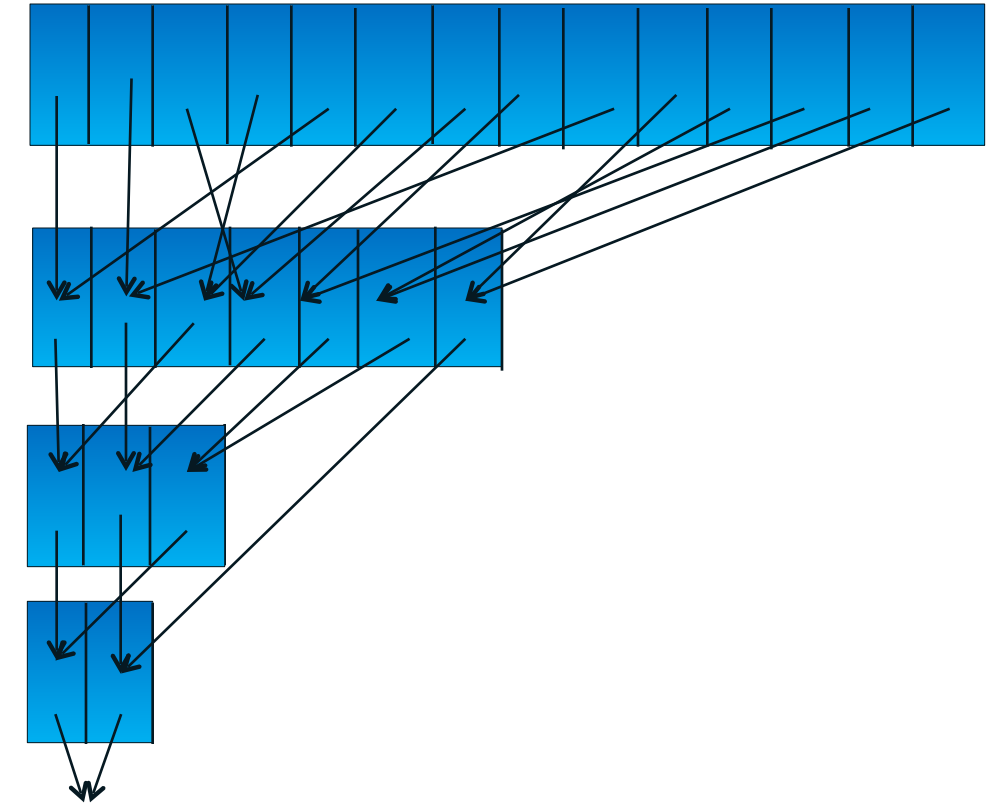
Positive

```
var pa = [1, -7, 3, 5]
var positivePa = pa.filterPar(
     (e) => e > 0;
 );                                          //[1,3,5]
```

Non-determinism

Inherent in reduction: reducePar, scanPar, scatterPar

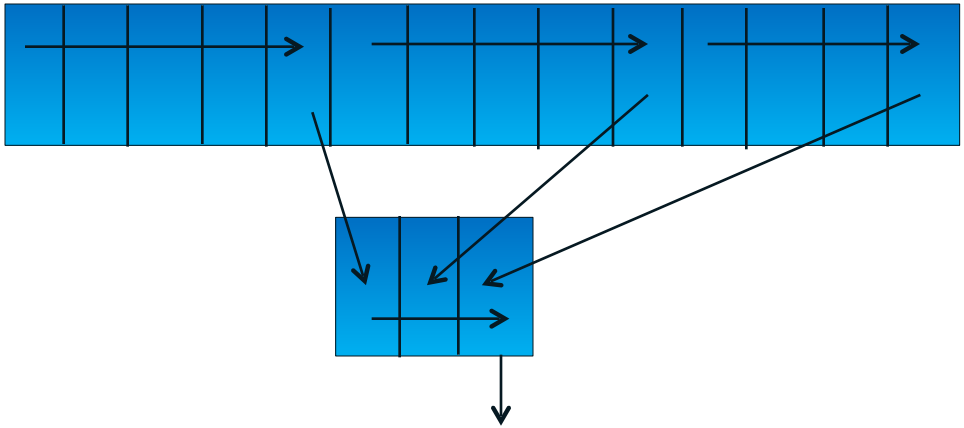
"It's all about the scheduler" Guy Blelloch



Non-determinism

Inherent in reduction: reducePar, scanPar, scatterPar

"It's all about the scheduler" Guy Blelloch



Result

Associativity provides determinism

Result from a legal schedule:
No out of thin air results

Spec wording

reducePar and scanPar use values from the Original O array and results pushed onto an A array

Repeat in an arbitrary and implementation dependent order len-1 times

- *Select 2 previously unselected indices, k1 and k2 from O or A*

Why parallel versions?

Sufficiently sophisticated compiler argument

- New semantics to reduce, scan, and scatter
- Mental Model is temporal immutability and
  - developing parallel algorithms instead of
  - parallelizing sequential algorithms
- Auto-parallelization is a long time unsolved problem
- Intent improves tooling and feedback
  - Formalizing Parallelizable subset definition difficult

What we have learned

We can see the horizon and there are no show stoppers

- Multiple prototypes: Intel(FF, V8/Crosswalk)
- Production: Mozilla closely tracking spec

Scaling is achievable in parallelizable parts of application

Falling back to sequential schedule better than throw

Out pointers to kernel functions are useful for reducing memory pressure and avoiding copying

Allocation pressure is crucial to performance in larger kernels

Pressure on Memory Management Latency

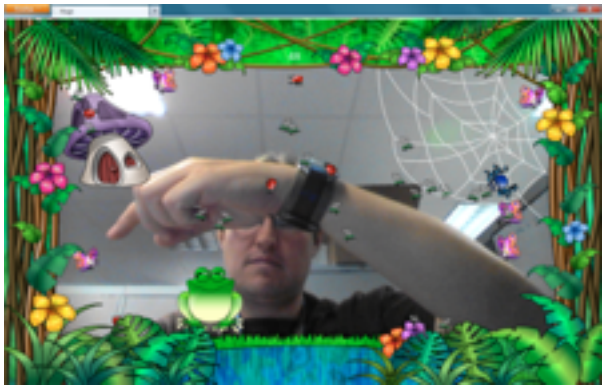
- During parallel computation global heap is immutable
- Only the return value escapes an elemental function
- Rollback to GC safe point trivial

- Simple concurrent approach
  - Each elemental function gets a local heap
  - Elemental function copies return value to the global heap
  - Local heap recycled immediately
  - No global synchronization required

Semantics Enable GC Concurrency

Lessons from use cases

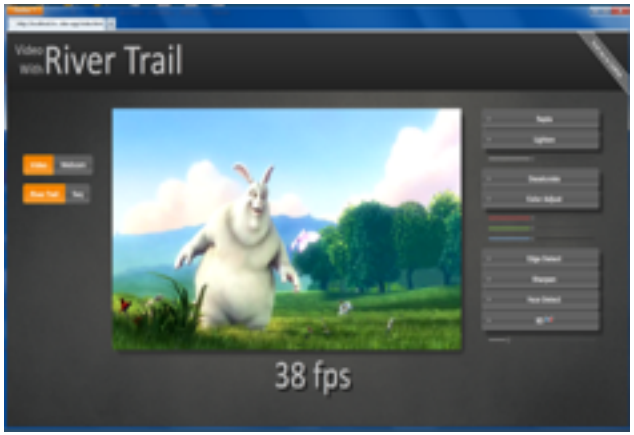
Visual Computing



3D Animation

Physics-based Gaming

Video



Next steps

More use cases and use-case driven performance work

Typed Object and PJS are being co-designed

Firming up spec language
https://github.com/RLH/ParallelJavaScript/wiki

Three dog tug by Nate Bolt, from http://www.flickr.com/photos/boltron/623602756/

Q&A

Backup