

# PARALLEL JAVASCRIPT

Norm Rubin (NVIDIA)

Jin Wang (Georgia School of Technology)



# JAVASCRIPT

- Not connected with Java
- Scheme and self (dressed in c clothing)
- Lots of design errors (like automatic semicolon insertion)
- Has inheritance but not objects

# JAVASCRIPT

- Widely used, Might be the most widely used programming language. 5-20 million programmers
- No need to declare types
- Friendly to amateurs
- Runs everywhere (all browsers)
- Installed everywhere
- Functional language
- Feels deterministic
  - Uses events so races are not a problem
- Cares about security

# PERFORMANCE

- For the last four years cpu performance of JS has gone up 30 times, but it is still single threaded
  - One of the great successes of compilers
- Extra performance has expended usage
- node.js - js on the sever
- Js as a delivery mechanism
  - (30 or more compilers that target js)
- Node-copter - js as device controller
- ....

# JAVASCRIPT ON MODERN ARCHITECTURES



PC



Server



Mobile



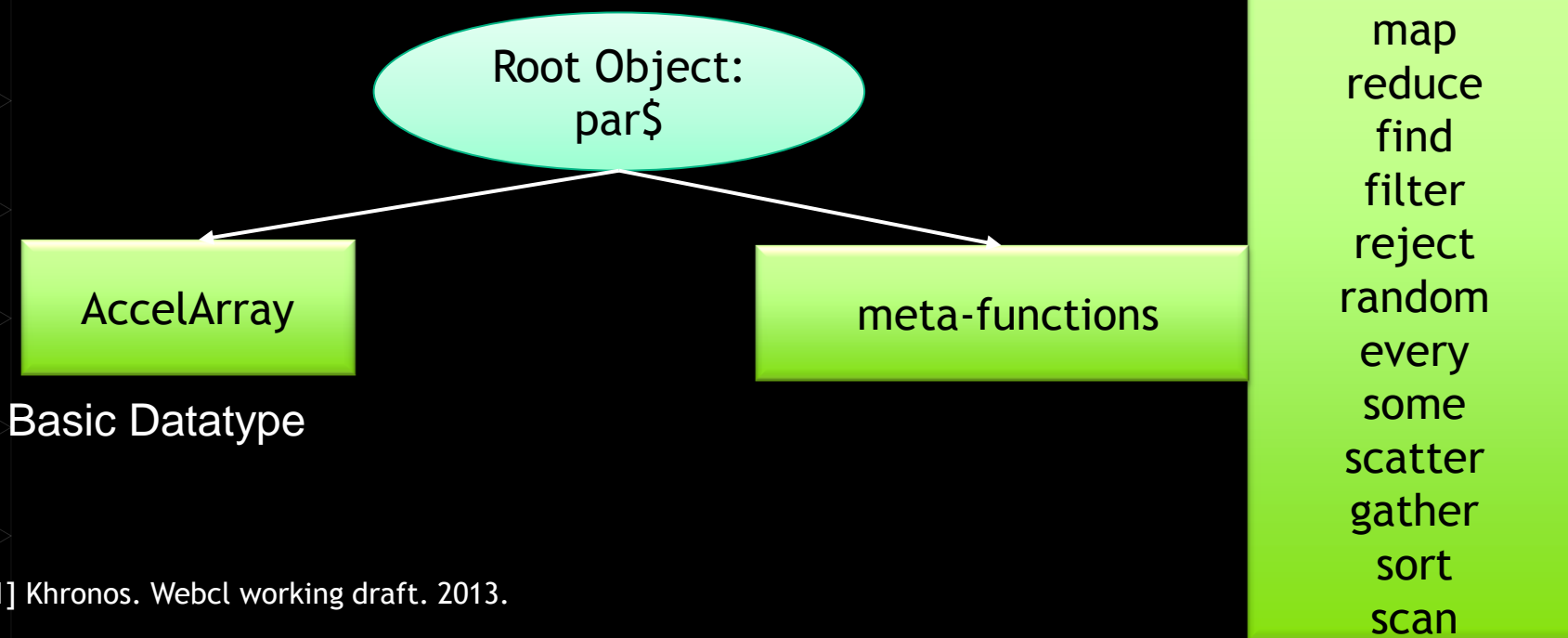
High level

# GOALS OF THIS PROJECT

- Can adding very high level primitives remove the need for GPU specific knowledge?
  - No shared memory
  - No hierarchical parallelism
  - No thread id, no blocks
- Can high level primitives give us device performance portability ?
- Can we extend JS without needing to add a second language so that it is still easy to use?
- Can we do all this in a jit?

# PARALLELJS PROGRAM

- Define High-level Parallel Constructs
- Use same syntax defined by regular JavaScript
- No extra language binding like WebGL<sup>[1]</sup>



[1] Khronos. Webcl working draft. 2013.

# AN EXAMPLE - SUM OF SQUARES

Classic JS:

```
function square(x){return x * x;}  
function plus(x, y){return x + y;}  
var numbers = [1,2,3,4,5,6];  
var sum = numbers.map(square).reduce(plus);
```

One line -change to go accelerate!

```
function square(x){return x * x;}  
function plus(x, y){return x + y;}  
var numbers = new par$.AccelArray([1,2,3,4,5,6]);  
var sum = numbers.map(square).reduce(plus);
```



# CUDA CODE IS MUCH LARGER

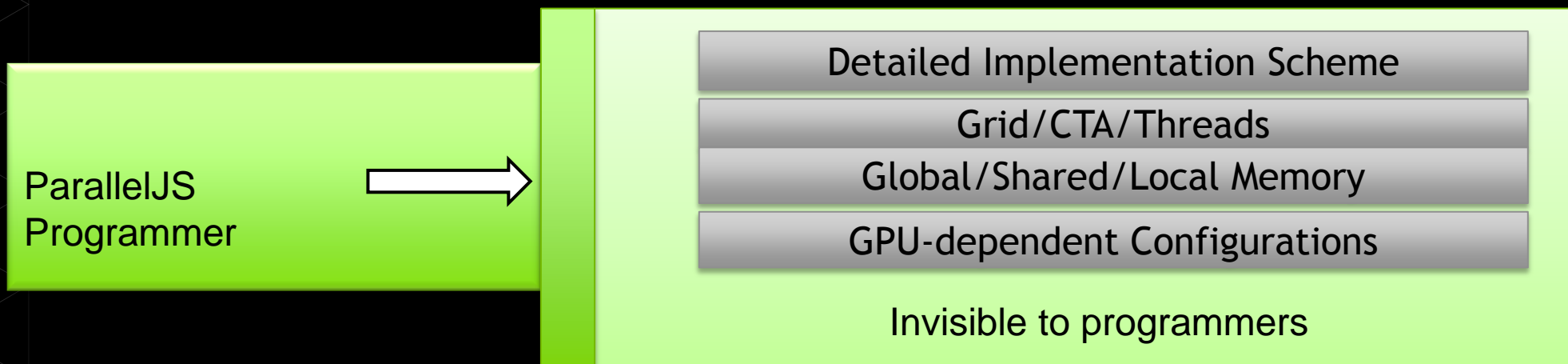
```
cudaMalloc((void **) &gpudata, sizeof(int) * DATA_SIZE);
cudaMalloc((void **) &result, sizeof(int) * THREAD_NUM * BLOCK_NUM);
cudaMemcpy(gpudata, data, sizeof(int) * DATA_SIZE, cudaMemcpyHostToDevice);
sumOfSquares<<<BLOCK_NUM, THREAD_NUM, THREAD_NUM * sizeof(int)>>>
    (gpudata, result, DATA_SIZE);
cudaMemcpy(&sum, result, sizeof(int) * BLOCK_NUM, cudaMemcpyDeviceToHost);
...
#define BLOCK_NUM      32
#define THREAD_NUM     512

__global__ static void sumOfSquares(int * num, int * result, int DATA_SIZE)
{
    extern __shared__ int shared[];
    const int tid = threadIdx.x;
    const int bid = blockIdx.x;
        shared[tid] = 0;
    ....
```

# MAP(INPUTS[,OUTPUTS][,FUNCTION]...)

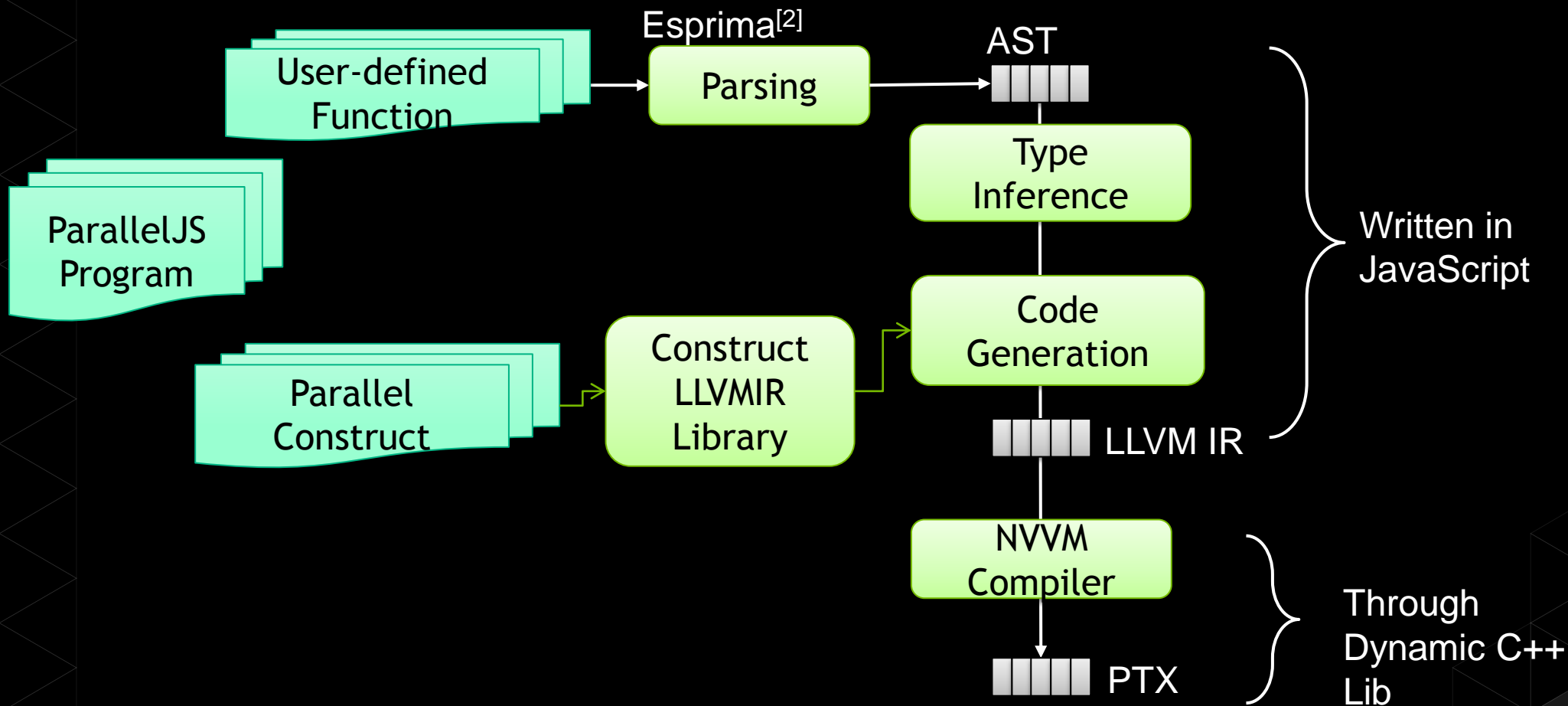
- Inputs - one or more accelArrays
- Outputs -Zero or more AccelArrays
  - Will provide an output if none is specified
  - Each parallel thread owns an output location, only the owner can write its location, (checked by compiler)
  - No output can also be an input (checked by run-time)
- Any valid JavaScript function provided it has no side effects.
  - Might run on GPU or on host- transparently
  - Becomes a loop on the host

# HIGH-LEVEL PROGRAMMING METHODOLOGY



ParallelJS programs are typically smaller than CUDA codes  
Productivity and Performance portability

# JIT COMPILATION FOR GPU

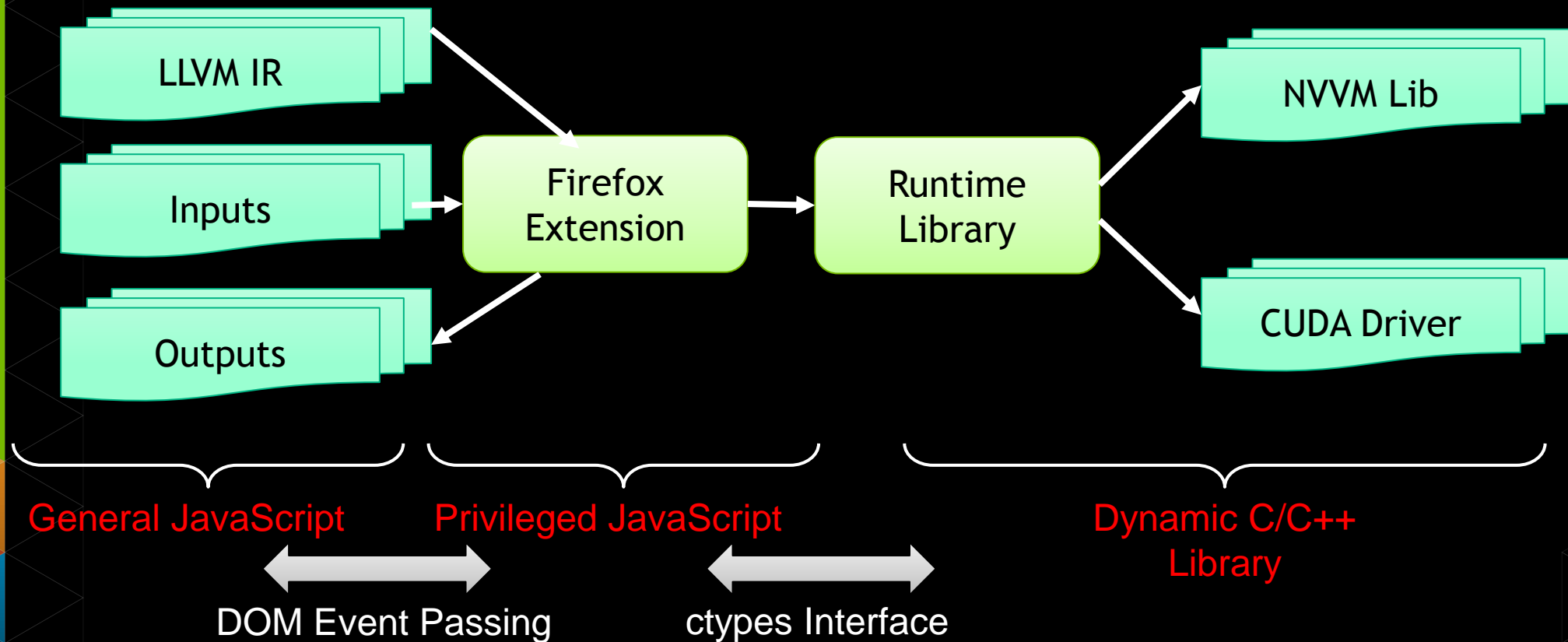


[2] A. Hidayat. Esprima: Ecmascript parsing infrastructure for multipurpose analysis. <http://esprima.org/>.

# COMPILATION CONT.

- AST: stored in JavaScript objects
- Type Inference
  - JavaScript is dynamically typed
  - Type propagation from input types
  - Avoid iterative Hindley-Milner algorithm
- Code Generation
  - Add Kernel Header and Meta to comply with NVVM
  - AST->LLVM Instructions
- NVVM: Generate PTX from LLVM

## RUNTIME SYSTEM (FIREFOX)



# RUNTIME SYSTEM CONT.

- Data Transfer Management
  - Data transfer is minimized between CPU and GPU
  - Garbage collector is used to reuse GPU memory
- Code Cache
  - ParallelJS Construct code is hashed
  - Future invocation of the same code does not need recompilation

# EXAMPLE PROGRAM

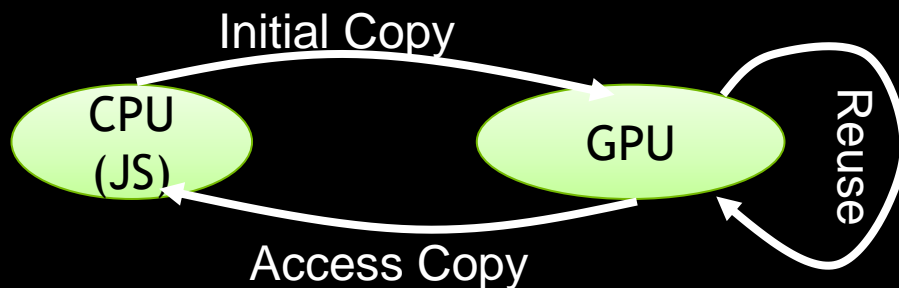
- Boids Simulation
  - Chain (Spring connection)
  - Mandelbrot
- } Use `par$.map`
- Reduce Use `par$.reduce`
  - Single Source Shortest Path (SSSP) Use `par$.map` and `par$.reduce`



# RUNTIME SYSTEM CONT.

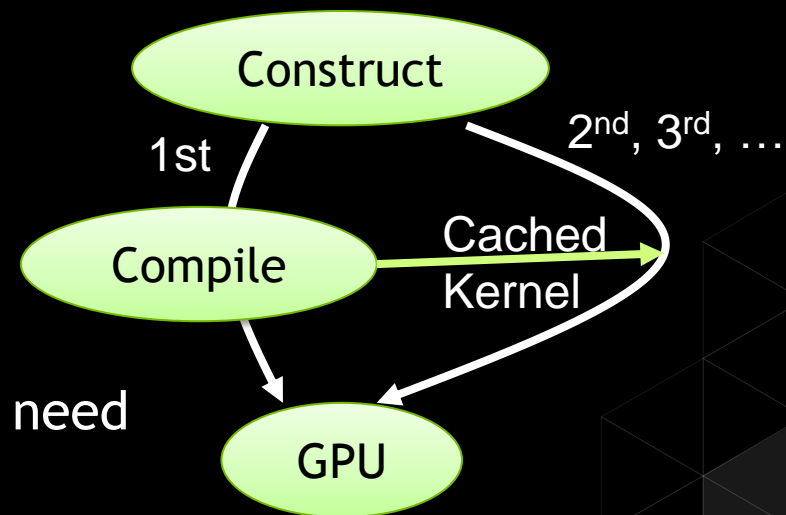
## ■ Data Transfer Management

- Copy data only when accessed by CPU or GPU
- Garbage collector is used to reuse GPU memory



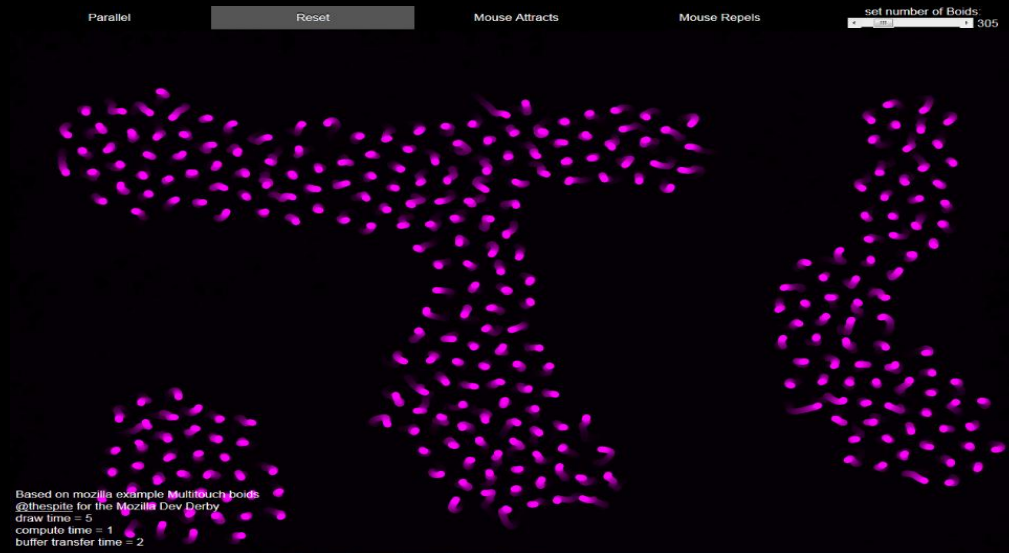
## ■ Code Cache

- ParallelJS Construct code is hashed
- Future invocation of the same code does not need recompilation

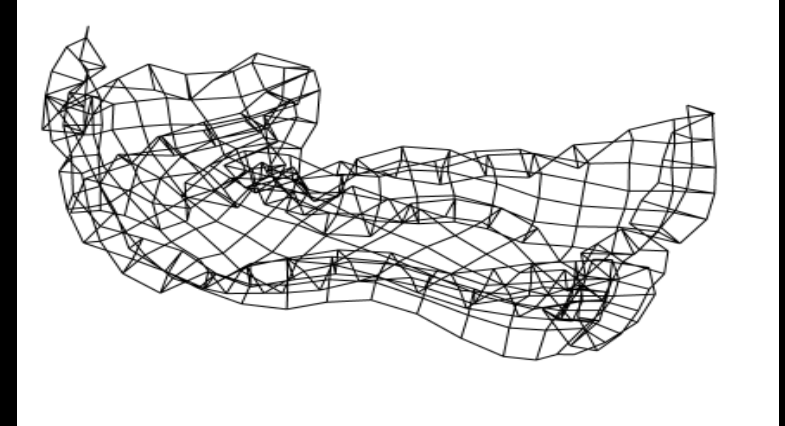


# Example Programs Cont.

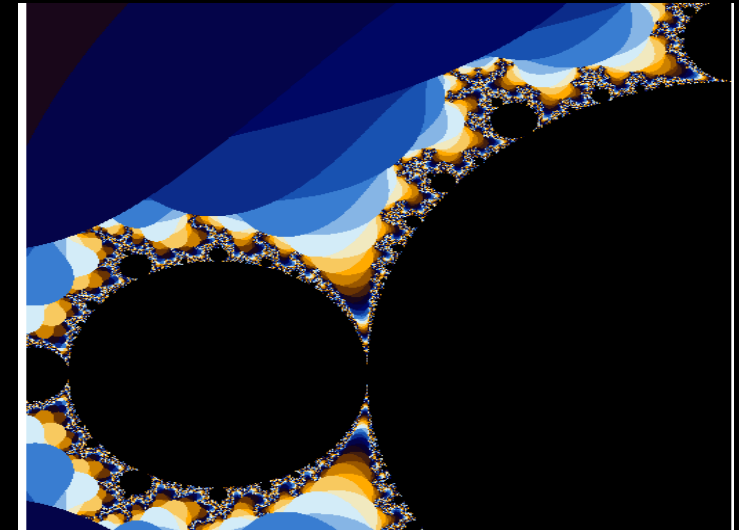
Boids



Chain



Mandel

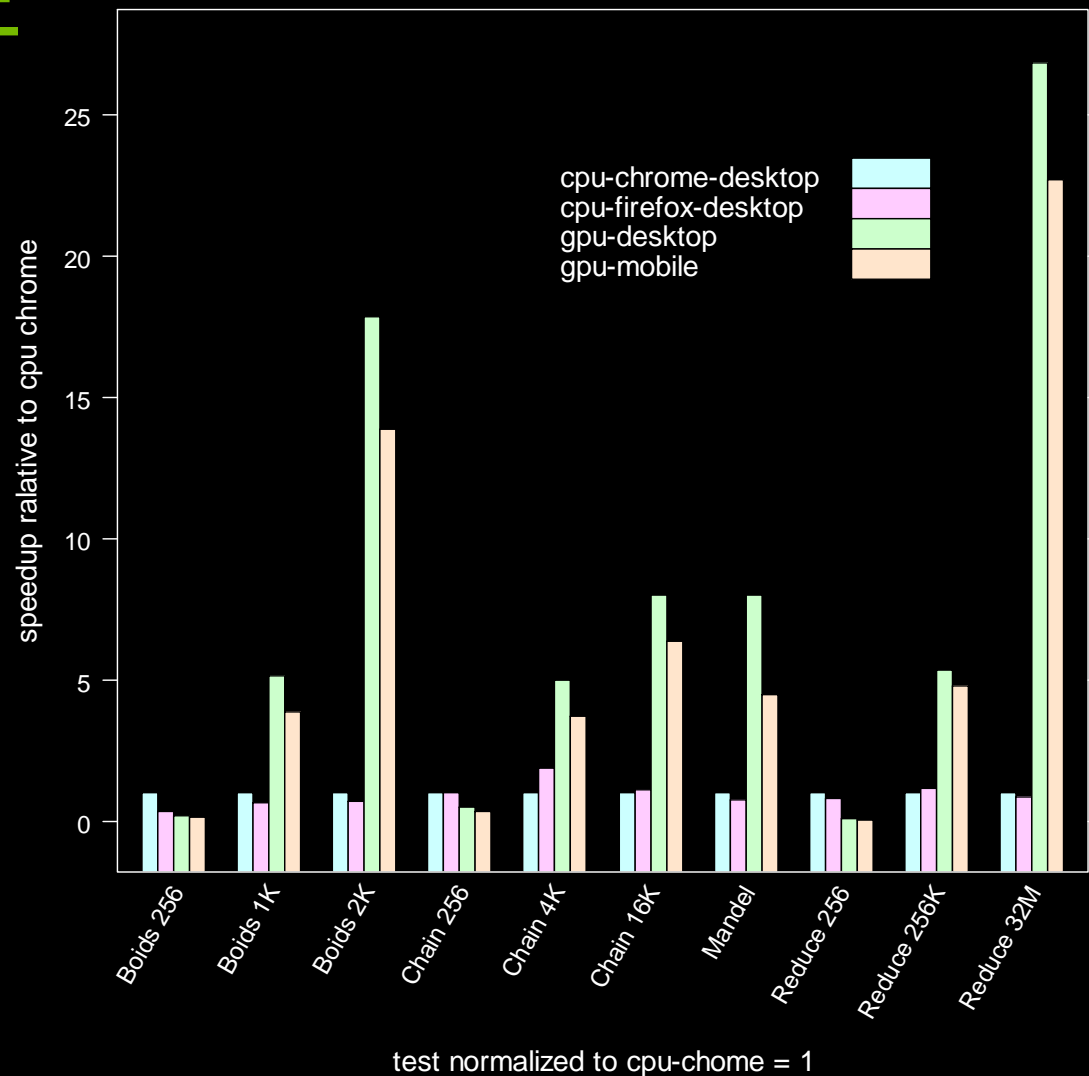


# PERFORMANCE

- Looked at
- discrete GK110 (titan) 2688 cores and i7-4771 3.5 gh
- Gtx 525M (fermi) 96 cores and i7-2630 2 gh

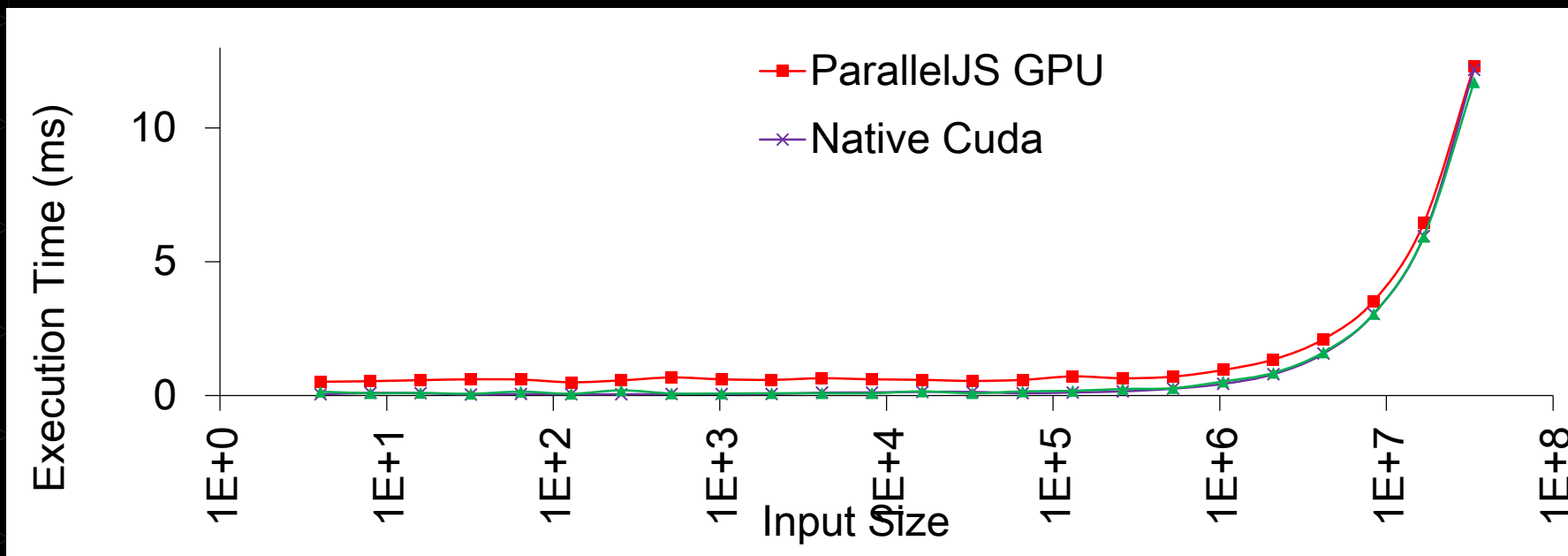
# PERFORMANCE RESULTS

- Small Input: ParallelJS is slower
- Larger Input: ParallelJS is **5.0x-26.8x** faster on desktop, **3.8x-22.7x** faster on laptop



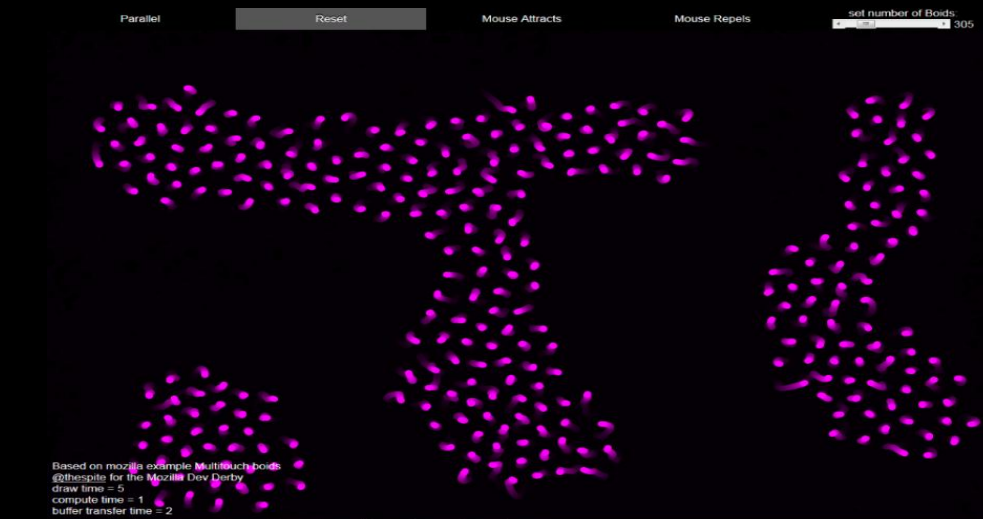
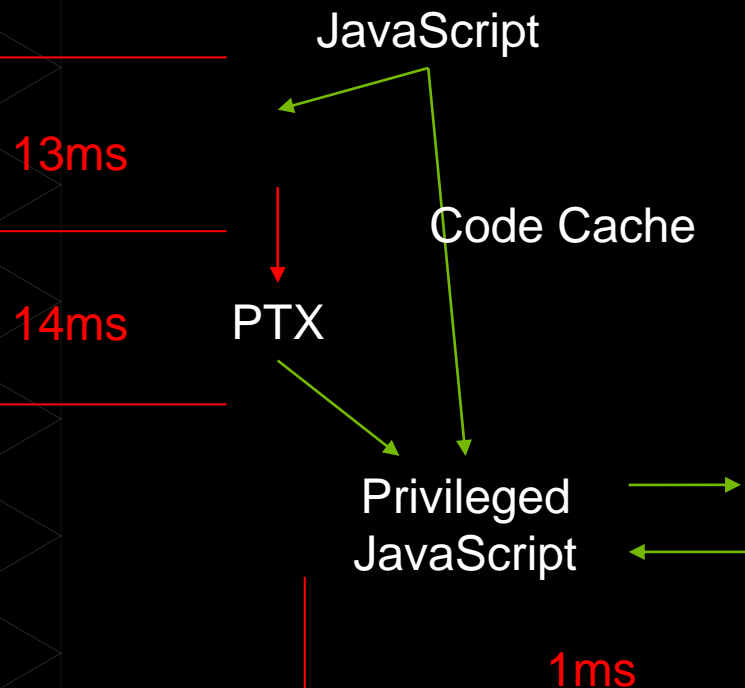
# PERFORMANCE RESULT CONT.

- Performance of Reduce compared with native CUDA, and CUB library



- For large input, ParallelJS has similar performance, but incredibly smaller code size

# PERFORMANCE BREAKDOWN FOR BODYS\_2K EXAMPLE



: Measured in JS  
\*: Measure in CUDA

# PERFORMANCE OF SSSP

- Compare with CUDA SSSP on CPU

Graph	nodes	edges	Time (ms)	Speedup
USA-Road	1070376	2712798	18685	0.86
r4-2e20	1048576	4194304	1215	0.60
rmat20	1048576	8259994	3645	0.59

- ParallelJS is always worse
  - Use backward algorithm instead of forward algorithm used by CUDA
    - Inevitable because JavaScript does not support atomic
- Cuda code is the lonestar benchmark

# COMPARE WITH RELATED WORK

- WebCL [1]
  - Foreign language binding into JavaScript
  - Programmers need to learn OpenCL
- JSonGPU[2]
  - Expose GPU concepts to JavaScript programmer
  - Complicated programming api
- RiverTrail[3] (prototype and production)
  - Prototype: uses openCL backend, tends to use extra memory
  - Production: more focus on SSE, and multi-core

[1] Khronos. Webcl working draft. 2013

[2] U. Pitambare, A. Chauhan, and S. Malviya. Just-in-time acceleration of javascript.

[3] S. Herhut, R. L. Hudson, T. Shpeisman, and J. Sreeram. River trail: A path to parallelism in javascript. SIGPLAN Not., 48(10):729{744, Oct. 2013.



# CONCLUSION

- ParallelJS: a framework for compile high-level JavaScript programs and execute them on heterogeneous systems
  - Use high-level constructs to hide GPU-specific details
  - Automating the decision of where to run the code
  - Productivity and performance portability
  - Limited use of advanced features of GPU
- Future directions
  - Currently always run on gpu we can, but the runtime should use better rules, how does this effect power efficiency?
  - Support of advanced GPU features

Thank you!

Questions?