

One day of life in V8

Vyacheslav Egorov

Can V8 do that?!

Vyacheslav Egorov

```
function dot (a, b) {  
    var d = 0;  
    for (var i = 0; i < a.length; i++) {  
        d += a[i] * b[i];  
    }  
    return d;  
}
```

```
dot(new Float64Array([1.1, 2.2, 3.3]),  
    new Float64Array([0.1, 0.2, 0.3]));
```

Can it be as efficient as C?

```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}
```

```
dot([1.1, 2.2, 3.3], [0.1, 0.2, 0.3]);
```

What about this?

```
function Klass (proto) {  
  function ctor() {  
    this.init.apply(this, arguments);  
  }  
  ctor.prototype = proto;  
  return ctor;  
}
```

```
var klass = Klass({  
  init: function (x, y, z) {  
    this.x = x;  
    this.y = y;  
    this.z = z;  
  }  
});
```

What about this?

```
function Klass (proto) {  
  function ctor() {  
    this.init.apply(this, arguments);  
  }  
  ctor.prototype = proto;  
  return ctor;  
}
```

```
var klass = Klass({  
  init: function (x, y, z) {  
    this.x = x;  
    this.y = y;  
    this.z = z;  
  }  
});
```

```
function OldSchool (x, y, z) {  
  this.x = x;  
  this.y = y;  
  this.z = z;  
}
```

Wh

```
var len = [1,2,3].map(function (i) {  
    return i * i;  
}).reduce(function (a, b) {  
    return a + b;  
}, 0);
```

VS

```
var arr = [1,2,3];  
var len = 0;  
for (var i = 0; i < arr.length; i++) {  
    len += arr[i] * arr[i];  
}
```

...and this?

**Can V8 show code it
generates?**

**Can V8 show code it
generates?**

Yes

```
$ make ia32.release objectprint=on \  
disassembler=on
```

```
$ out/ia32.release/d8 --print-opt-code \  
--code-comments \  
--trace-hydrogen \  
test.js
```

**print generated code
with comments**

**write intermediate
representation (IR) into
hydrogen.cfg.
can be viewed by C1Visualizer**

```
function dot (a, b) {
  var d = 0;
  for (var i = 0; i < a.length; i++) {
    d += a[i] * b[i];
  }
  return d;
}

var a = new Float64Array([1, 2, 3]);
var b = new Float64Array([0.1, 0.2, 0.3]);

while (true) dot(a, b);
```

```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}
```

```
var a = new Float64Array([1, 2, 3]);  
var b = new Float64Array([0.1, 0.2, 0.3]);
```

```
while (true) dot(a, b);
```



???

```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}  
  
var a = new Float64Array([1, 2, 3]);  
var b = new Float64Array([0.1, 0.2, 0.3]);  
  
while (true) dot(a, b);
```

Can V8 hoist **a.length**?

```
function dot (a, b) {  
    var d = 0;  
    for (var i = 0; i < a.length; i++) {  
        d += a[i] * b[i];  
    }  
    return d;  
}  
  
var a = new Float64Array([1, 2, 3]);  
var b = new Float64Array([0.1, 0.2, 0.3]);  
  
while (true) dot(a, b);
```

Is array access fast like in C?

```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}
```

```
var a = new Float64Array([1, 2, 3]);  
var b = new Float64Array([0.1, 0.2, 0.3]);  
  
while (true) dot(a, b);
```

Can V8 avoid boxing numbers?


```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}  
  
var a = new Float64Array([1, 2, 3]);  
var b = new Float64Array([0.1, 0.2, 0.3]);  
  
while (true) dot(a, b);
```

Can V8 differentiate int vs. double?


```
;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx
168 jnc 0x5b60a03c
174 movsd xmm1,[edx+esi*8]
179 cmp esi,eax
181 jnc 0x5b60a046
187 movsd xmm3,[edi+esi*8]
192 mulsd xmm1,xmm3
196 addsd xmm2,xmm1
200 add esi,0x1
203 jmp 146
```

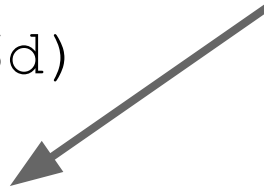
```
;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx
168 jnc 0x5b60a03c
174 movsd xmm1,[edx+esi*8]
179 cmp esi,eax
181 jnc 0x5b60a046
187 movsd xmm3,[edi+esi*8]
192 mulsd xmm1,xmm3
196 addsd xmm2,xmm1
200 add esi,0x1
203 jmp 146
```

esi contains i
ecx contains a.length



interruption mechanism
(e.g. for debugger)

```
;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx
168 jnc 0x5b60a03c
174 movsd xmm1,[edx+esi*8]
179 cmp esi,eax
181 jnc 0x5b60a046
187 movsd xmm3,[edi+esi*8]
192 mulsd xmm1,xmm3
196 addsd xmm2,xmm1
200 add esi,0x1
203 jmp 146
```



```
;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx bounds check for a[i]
168 jnc 0x5b60a03c
174 movsd xmm1,[edx+esi*8] load a[i] into xmm1
179 cmp esi,eax bounds check for b[i]
181 jnc 0x5b60a046
187 movsd xmm3,[edi+esi*8] load b[i] into xmm3
192 mulsd xmm1,xmm3
196 addsd xmm2,xmm1
200 add esi,0x1
203 jmp 146
```

```

;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx
168 jnc 0x5b60a03c
174 movsd xmm1,[edx+esi*8]
179 cmp esi,eax
181 jnc 0x5b60a046
187 movsd xmm3,[edi+esi*8]
192 mulsd xmm1,xmm3      a[i] * b[i]
196 addsd xmm2,xmm1     d += a[i] * b[i]
200 add esi,0x1
203 jmp 146

```

```
;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx
168 jnc 0x5b60a03c
174 movsd xmm1,[edx+esi*8]
179 cmp esi,eax
181 jnc 0x5b60a046
187 movsd xmm3,[edi+esi*8]
192 mulsd xmm1,xmm3
196 addsd xmm2,xmm1
200 add esi,0x1 i++
203 jmp 146
```

```
47  cmp  [eax-1], 0x2250cd81
54  jnz  0x2c60a014
60  mov  ecx, [eax+0x3]
63  mov  ecx, [ecx+0x7]
```

```
;;; B2 - LOOP entry
```

```
146  cmp  esi, ecx
148  jnl  205    (0x20d2bd6d)
154  cmp  esp, [0x89b5ad0]
160  jc   404    (0x20d2be34)
166  cmp  esi, ebx
168  jnc  0x5b60a03c
174  movsd xmm1, [edx+esi*8]
179  cmp  esi, eax
181  jnc  0x5b60a046
187  ...
```

```
47  cmp  [eax-1], 0x2250cd81
54  jnz  0x2c60a014
60  mov  ecx, [eax+0x3]
63  mov  ecx, [ecx+0x7]
```

```
;;; B2 - LOOP entry
```

```
146  cmp  esi, ecx
148  jnl  205    (0x20d2bd6d)
154  cmp  esp, [0x89b5ad0]
160  jc   404    (0x20d2be34)
166  cmp  esi, ebx
168  jnc  0x5b60a03c
174  movsd xmm1, [edx+esi*8]
179  cmp  esi, eax
181  jnc  0x5b60a046
```

```
187  mov  ecx, [ecx+0x7]
```



```
47  cmp  [eax-1], 0x2250cd81
54  jnz  0x2c60a014
60  mov  ecx, [eax+0x3]
63  mov  ecx, [ecx+0x7]
```

```
;;; B2 - LOOP entry
```

```
146  cmp  esi, ecx
148  jnl  205    (0x20d2bd6d)
154  cmp  esp, [0x89b5ad0]
160  jc   404    (0x20d2be34)
166  cmp  esi, ebx
168  jnc  0x5b60a03c
174  movsd xmm1, [edx+esi*8]
179  cmp  esi, eax
181  jnc  0x5b60a046
187  ...
```

How does V8 do that?

Crankshaft core in <1 minute

1. Compile function non-optimized. Every dynamic operation has an IC (inline cache).
2. Let non-optimized code run for some time
3. Once it's warm optimize.
4. Ask ICs for shapes and types. Build SSA HIR under optimistic assumptions.
5. For every instruction in HIR side-effects are known. Do GVN, LICM, representation/range inference.
6. Translate HIR to LIR. Allocate registers.
7. Generate Code

Crankshaft core in <1 minute

1.  (e).
2. e
- 3.
- 4.
5. are
ange
- 6.
7. Generate code

Crankshaft core in <1 minute

1. Compile function non-optimized. Every dynamic operation has an **IC (inline cache)**.

2. Instrument function to measure execution time

```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}
```

Crankshaft core in <1 minute

1. Compile function non-optimized. Every dynamic operation has an IC (inline cache).
2. Let non-optimized code run for some time
3. Once it's warm optimize.
4. Ask ICs for shapes and types. Build SSA HIR under **optimistic assumptions**.
5. For every instruction in HIR side-effects are known. Do GVN, LICM, representation/range inference.
6. Translate HIR to LIR. Allocate registers.
7. Generate Code

```
;;; B2 - LOOP entry
146 cmp esi,ecx
148 jnl 205 (0x20d2bd6d)
154 cmp esp,[0x89b5ad0]
160 jc 404 (0x20d2be34)
166 cmp esi,ebx ; out-of-bounds load?
168 jnc 0x5b60a03c ; deoptimization
174 movsd xmm1,[edx+esi*8]
179 cmp esi,eax ; out-of-bounds load?
181 jnc 0x5b60a046 ; deoptimization
187 movsd xmm3,[edi+esi*8]
192 mulsd xmm1,xmm3
196 addsd xmm2,xmm1
200 add esi,0x1
203 jmp 146
```

```
Object.defineProperty(  
  Object.prototype,  
  "2",  
  {  
    get: function() {  
      /* some crazy side-effect */  
    }  
  }  
);  
  
new Float64Array(1)[2] /* oopsy */
```



```
Object.defineProperty(  
  Object.prototype,  
  "2",  
  {  
    get: function() {  
      /* some crazy side-effect */  
    }  
  }  
);  
  
new Float64Array(1)[2] /* oopsy */
```

V8 makes local assumptions

```
function dot (mul, a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += mul(a, b, i);  
  }  
  return d;  
}
```

```
dot(function mul(a, b, i) {  
  return a[i] * b[i];  
}, a, b);
```

**Can I defeat local assumptions
by a function call?**

```
;;; B2 - LOOP entry
179 cmp ebx,ecx
181 jnl 238
187 cmp esp,[0x94d6ad0]
193 jc 437 (0x35d2bfd5)
199 cmp ebx,edi
201 jnc 0x5930a046
207 movsd xmm1,[esi+ebx*8]
212 cmp ebx,eax
214 jnc 0x5930a050
220 movsd xmm3,[edx+ebx*8]
225 mulsd xmm1,xmm3
229 addsd xmm2,xmm1
233 add ebx,0x1
236 jmp 179 (0x35d2bed3)
```

Not with a simple one. It will be inlined.

```
89 mov esi, [ebp+0x10]
92 cmp esi, [0x43e0a59c] ; JSFunction mul
98 jnz 0x5930a01e
```

```
;; .....
```

```
;;; B2 - LOOP entry
179 cmp ebx, ecx
181 jnl 238
187 cmp esp, [0x94d6ad0]
193 jc 437 (0x35d2bfd5)
199 cmp ebx, edi
201 jnc 0x5930a046
207 movsd xmm1, [esi+ebx*8]
212 cmp ebx, eax
214 jnc 0x5930a050
```

Not with a simple
one. It will be inlined.

```
function dot (mul, a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += mul(a, b, i);  
  }  
  return d;  
}
```

```
dot(function mul(a, b, i) {  
  with (a) { };  
  return a[i] * b[i];  
}, a, b);
```

What if I *scare* away inlining?

```
function dot (mul, a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += mul(a, b, i);  
  }  
  return d;  
}
```

```
dot(function mul(a, b, i) {  
  with (a) { };  
  return a[i] * b[i];  
}, a, b);
```

Resulting native code is scary.
Lets look into HIR

B2:
i20 = Phi [i57, i50]
d21 = Phi [d56, d48]
CheckMaps t5 [0x4750cd81]
t28 = LoadNamedField t5 @8
i60 = Change t28 t to i
CompareIDAndBranch LT i20 i60 goto (B4, B5)

B4:
PushArgument t40
PushArgument t5
PushArgument t6
t58 = Change i20 i to t
PushArgument t58
t46 = InvokeFunction t7 t4 #4 changes[*]
d61 = Change t46 t to d
d48 = Add d21 d61
i50 = Add i20 i55
Goto B2

```
B2:  
i20 = Phi [i57, i50]  
d21 = Phi [d56, d48]  
CheckMaps t5 [0x4750cd81]  
t28 = LoadNamedField t5 @8  
i60 = Change t28 t to i  
CompareIDAndBranch LT i20 i60 goto (B4, B5)
```

```
B4:  
PushArgument t40  
PushArgument t5  
PushArgument t6  
t58 = Change i20 i to t  
PushArgument t58  
t46 = InvokeFunction t7 t4 #4 changes [*]  
d61 = Change t46 t to d  
d48 = Add d21 d61  
i50 = Add i20 i55  
Goto B2
```



```
B2:  
i20 = Phi [i57, i50]  
d21 = Phi [d56, d48]  
CheckMaps t5 [0x4750cd81]  
t28 = LoadNamedField t5 @8  
i60 = Change t28 t to i  
CompareIDAndBranch LT i20 i60 goto (B4, B5)
```

```
B4:  
PushArgument t40  
PushArgument t5  
PushArgument t6  
t58 = Change i20 i to t  
PushArgument t58  
t46 = InvokeFunction t7 t4 #4 changes[*]  
d61 = Change t46 t to d  
d48 = Add d21 d61  
i50 = Add i20 i55  
Goto B2
```

```
B2:  
i20 = Phi [i57, i50]  
d21 = Phi [d56, d48]  
CheckMaps t5 [0x4750cd81]  
t28 = LoadNamedField t5 @8  
i60 = Change t28 t to i  
CompareIDAndBranch LT i20 i60 goto (B4, B5)
```

```
B4:  
PushArgument t40  
PushArgument t5  
PushArgument t6  
t58 = Change i20 i to t  
PushArgument t58  
t46 = InvokeFunction t7 t4 #4 changes[*]  
d61 = Change t46 t to d  
d48 = Add d21 d61  
i50 = Add i20 i55  
Goto B2
```

```
function dot (a, b) {  
  var d = 0;  
  for (var i = 0; i < a.length; i++) {  
    d += a[i] * b[i];  
  }  
  return d;  
}
```

```
var a = [1.1, 2.2, 3.3];  
var b = [0.1, 0.2, 0.3];
```

```
while (true) dot(a, b);
```

```
201  cmp  eax,edx
203  jnc  0x3440a050
209  cmp  [edi+eax*8+0xb],0x7fffffff
217  jz   0x3440a05a
223  movsd xmm1,[edi+eax*8+0x7]
229  cmp  eax,esi
231  jnc  0x3440a064
237  cmp  [ebx+eax*8+0xb],0x7fffffff
245  jz   0x3440a06e
251  movsd xmm3,[ebx+eax*8+0x7]
```

```
201  cmp  eax,edx
203  jnc  0x3440a050
209  cmp  [edi+eax*8+0xb],0x7fffffff
217  jz   0x3440a05a
223  movsd xmm1,[edi+eax*8+0x7]
229  cmp  eax,esi
231  jnc  0x3440a064
237  cmp  [ebx+eax*8+0xb],0x7fffffff
245  jz   0x3440a06e
251  movsd xmm3,[ebx+eax*8+0x7]
```

```
201  cmp  eax,edx
203  jnc  0x3440a050
209  cmp  [edi+eax*8+0xb],0x7fffffff
217  jz   0x3440a05a
223  movsd xmm1,[edi+eax*8+0x7]
229  cmp  eax,esi
231  jnc  0x3440a064
237  cmp  [ebx+eax*8+0xb],0x7fffffff
245  jz   0x3440a06e
251  movsd xmm3,[ebx+eax*8+0x7]
```

check for array "hole"
stored as special NaN



```
201  cmp  eax,edx
203  jnc  0x3440a050
209  cmp  [edi+eax*8+0xb],0x7fffffff
217  jz   0x3440a05a
223  movsd xmm1,[edi+eax*8+0x7]
229  cmp  eax,esi
231  jnc  0x3440a064
237  cmp  [ebx+eax*8+0xb],0x7fffffff
245  jz   0x3440a06e
251  movsd xmm3,[ebx+eax*8+0x7]
```

check for array "hole"
stored as special NaN



**Here one can suggest to track
also denseness of an array.**

```
var klass = Klass({
  init: function (x, y, z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
});
```

```
function test () {
  return new klass(1, 2, 3);
}
```

```
while (true) test();
```



```
t10 = LoadGlobalCell [0x5f60a4d1]
t14 = CheckFunction t10 0x52716959 "ctor"
t15 = AllocateObject
EnterInlined ctor
t20 = LoadNamedGeneric t15.init changes[*]
t22 = CheckNonSmi t20
CheckMaps t20 [0x3ab0b4a1]
CheckPrototypeMaps t20
PushArgument t15
PushArgument (Constant 1)
PushArgument (Constant 2)
PushArgument (Constant 3)
t30 = InvokeFunction t16 t20 #4 changes[*]
LeaveInlined
```

```
t10 = LoadGlobalCell [0x5f60a4d1]
t14 = CheckFunction t10 0x52716959 "ctor"
t15 = AllocateObject
EnterInlined ctor
t20 = LoadNamedGeneric t15.init changes[*]
t22 = CheckNonSmi t20
CheckMaps t20 [0x3ab0b4a1]
CheckPrototypeMaps t20
PushArgument t15
PushArgument (Constant 1)
PushArgument (Constant 2)
PushArgument (Constant 3)
t30 = InvokeFunction t16 t20 #4 changes[*]
LeaveInlined
```

```
t10 = LoadGlobalCell [0x5f60a4d1]
t14 = CheckFunction t10 0x52716959 "ctor"
t15 = AllocateObject
EnterInlined ctor
t20 = LoadNamedGeneric t15.init changes[*]
t22 = CheckNonSmi t20
CheckMaps t20 [0x3ab0b4a1]
CheckPrototypeMaps t20
PushArgument t15
PushArgument (Constant 1)
PushArgument (Constant 2)
PushArgument (Constant 3)
t30 = InvokeFunction t16 t20 #4 changes[*]
LeaveInlined
```

```
t10 = LoadGlobalCell [0x5f60a4d1]
t14 = CheckFunction t10 0x52716959 "ctor"
t15 = AllocateObject
EnterInlined ctor
t20 = LoadNamedGeneric t15.init changes[*]
t22 = CheckNonSmi t20
CheckMaps t20 [0x3ab0b4a1]
CheckPrototypeMaps t20
PushArgument t15
PushArgument (Constant 1)
PushArgument (Constant 2)
PushArgument (Constant 3)
t30 = InvokeFunction t16 t20 #4 changes[*]
LeaveInlined
```

Yes, we can! `apply` is completely gone!

```
var len = [1,2,3].map(function (i) {  
    return i * i;  
}).reduce(function (a, b) {  
    return a + b;  
}, 0);
```

VS

```
var arr = [1,2,3];  
var len = 0;  
for (var i = 0; i < arr.length; i++) {  
    len += arr[i] * arr[i];  
}
```

```
var len = [1,2,3].map(function (i) {  
    return i * i;  
}).reduce(function (a, b) {  
    return a + b;  
}, 0);
```

VS

```
var arr = [1,2,3];  
var len = 0;  
for (var i = 0; i < arr.length; i++) {  
    len += arr[i] * arr[i];  
}
```

Unfortunately not now :-)

```
var len = [1,2,3].map(function (i) {  
  return i * i;  
}).reduce(function (a, b) {  
  return a + b;  
}, 0);
```

VS

```
var arr = [1,2,3];  
var len = 0;  
for (var i = 0; i < arr.length; i++) {  
  len += arr[i] * arr[i];  
}
```

Requires some plumbing

```
var len = [1,2,3].map(function (i) {  
    return i * i;  
}).reduce(function (a, b) {  
    return a + b;  
}, 0);
```

VS

```
var arr = [1,2,3];  
var len = 0;  
for (var i = 0; i < arr.length; i++) {  
    len += arr[i] * arr[i];  
}
```

We accept patches :-)

Dealing with unexpected slowdowns

- Have reduction/microbench => file a bug at <http://v8.googlecode.com>
- Don't have reduction? Ensure code is optimized (`--trace-opt`, `--trace-deopt`). Look at HIR/LIR/generated code. Try to spot inefficiencies:
 - Not hoisted loop invariants.
 - Generic loads and stores.
 - Redundant checks
 - Excessive conversions
 - Not inlined functions

Thank you!

Q & A