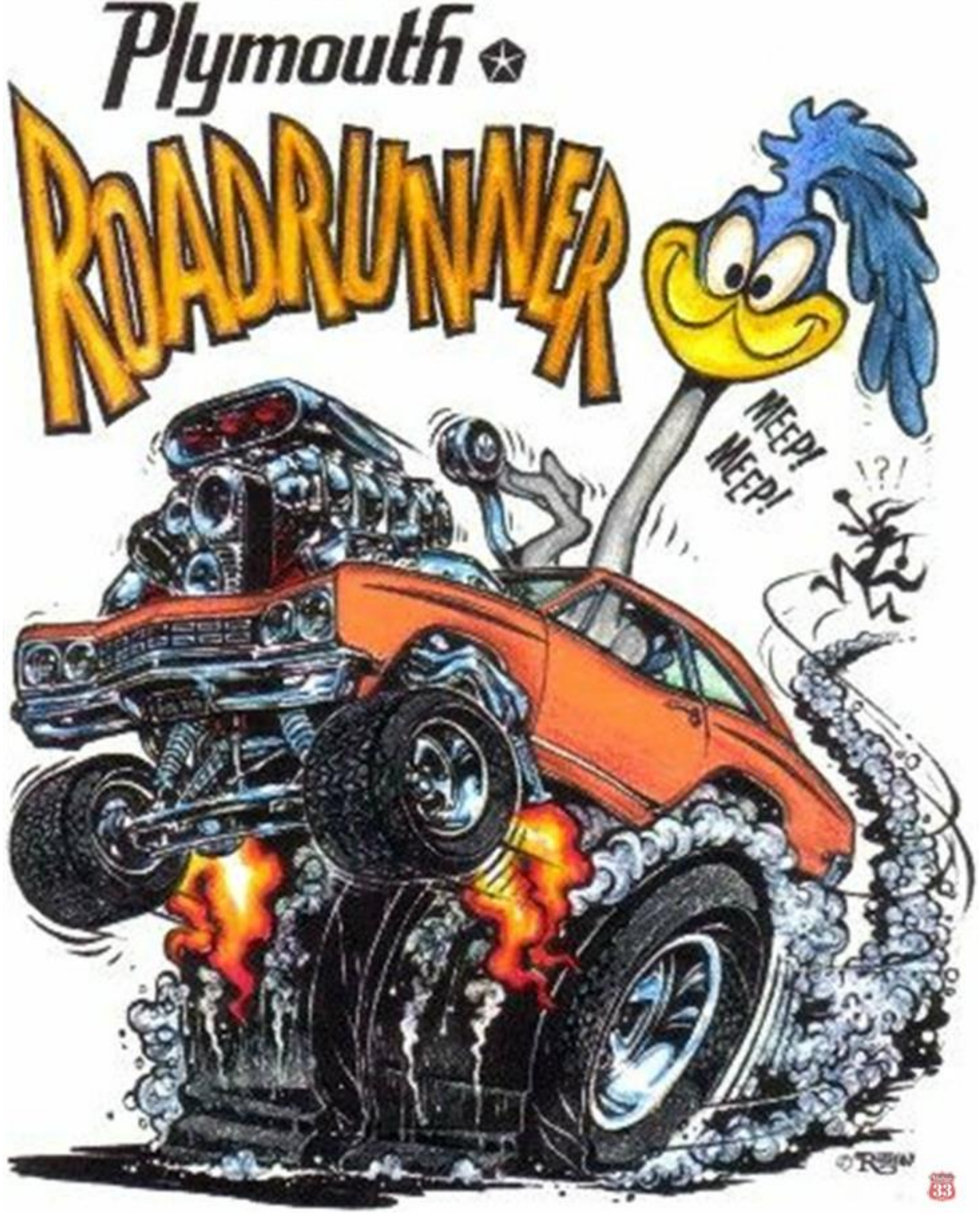
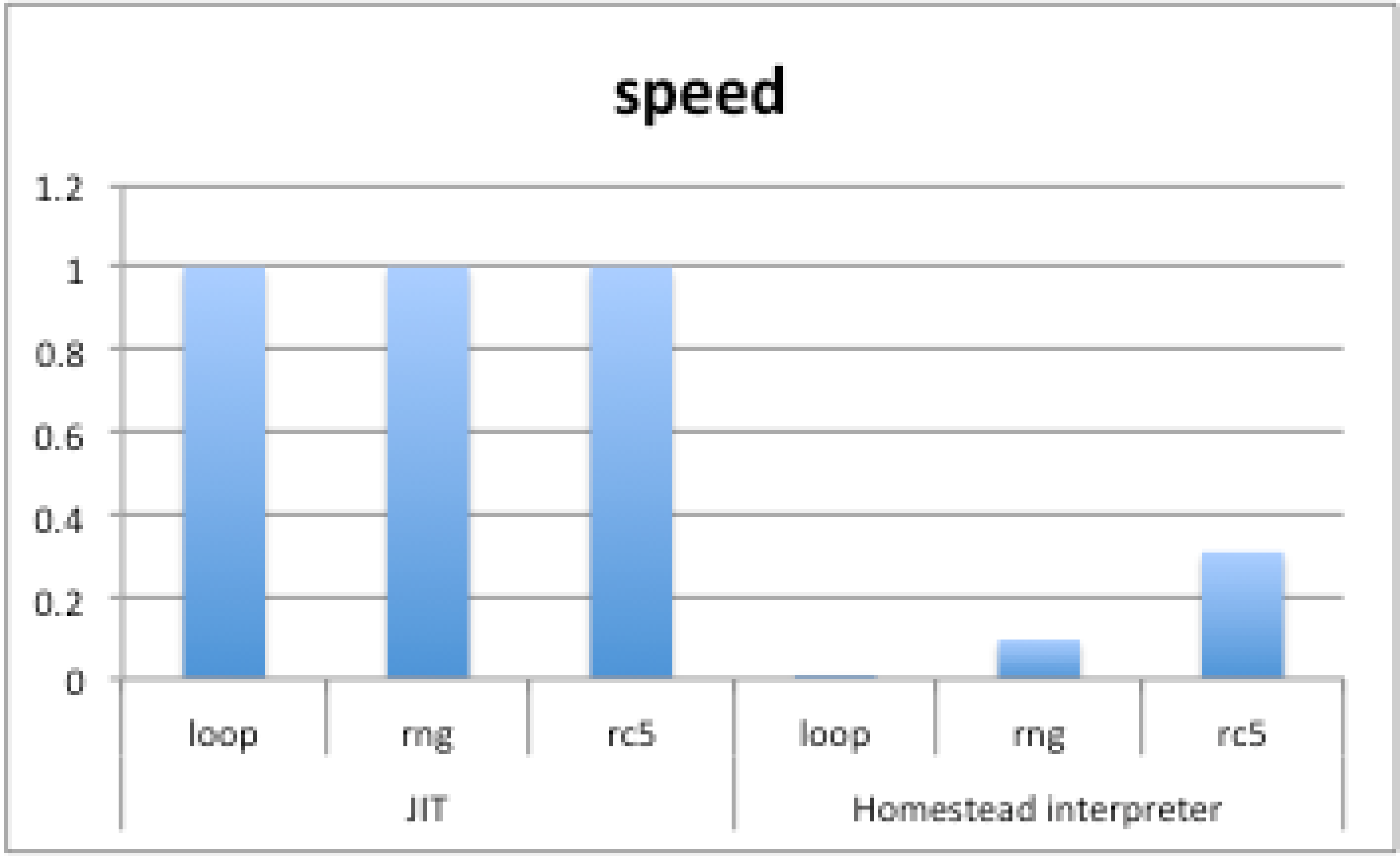
Making an EVM Interpreter Scream

Greg Colvin


Plymouth
ROADRUNNER
MEEP! MEEP!
!?!


LUCAS OIL
014
DODGE
MICHIGAN
HEMICAR
1968 Dodge Super Bee 426 Hemi
Only 128 Hemi Super Bee's Built in 1968
RoadTestTV

Belvedere 4-door Sedan



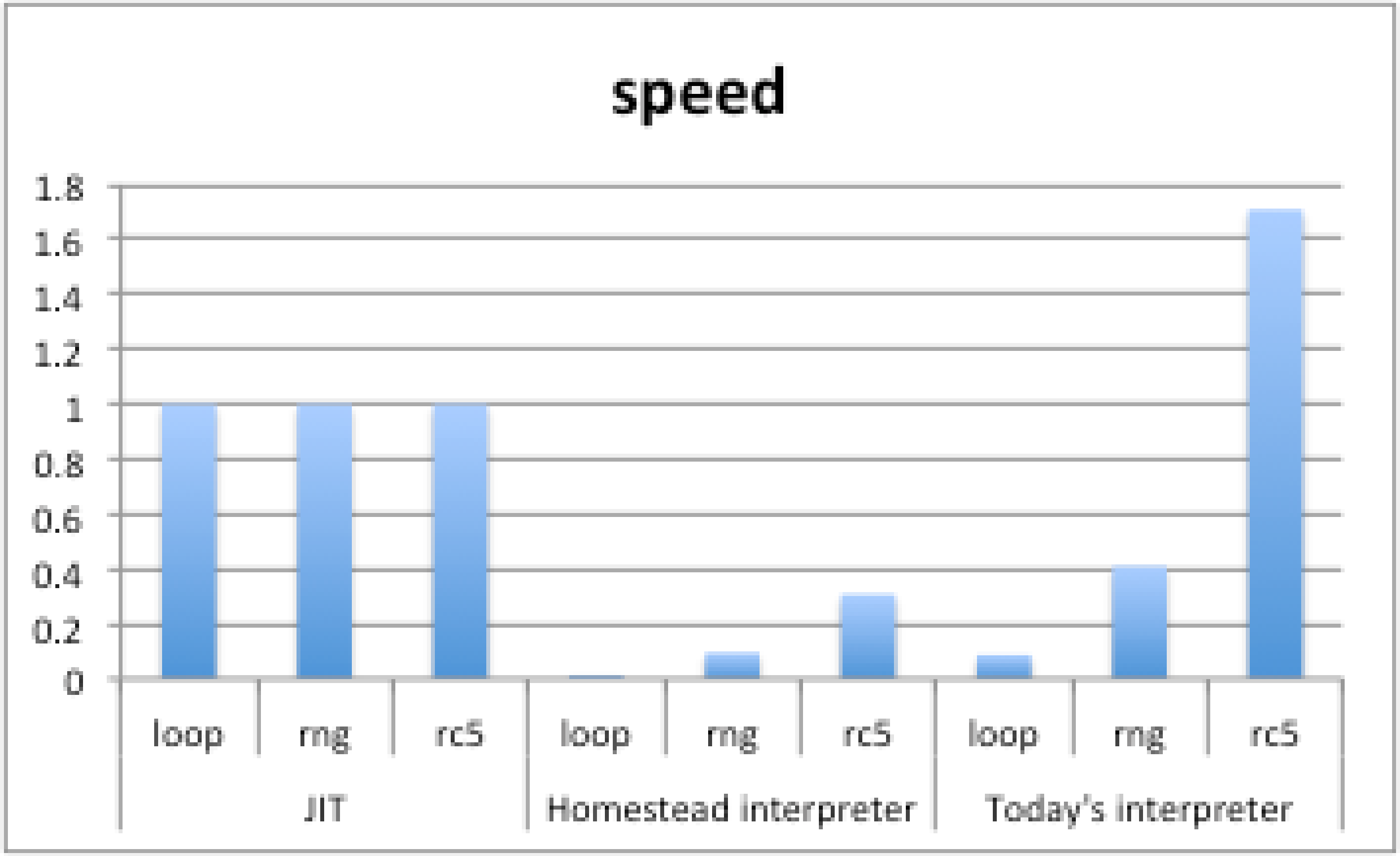
speed
1.2
1
0.8
0.6
0.4
0.2
0
loop
rng
rc5
loop
rng
rc5
JIT
Homestead interpreter

General approach

- Correctness preserving transformations
- Small testable changes
- No structural changes

Specific changes

- Operations: From infinite precision to 512 bit
- Gas calculations: From 256 bit to 64 bit
- No more gas and memory calculations than necessary for each operation

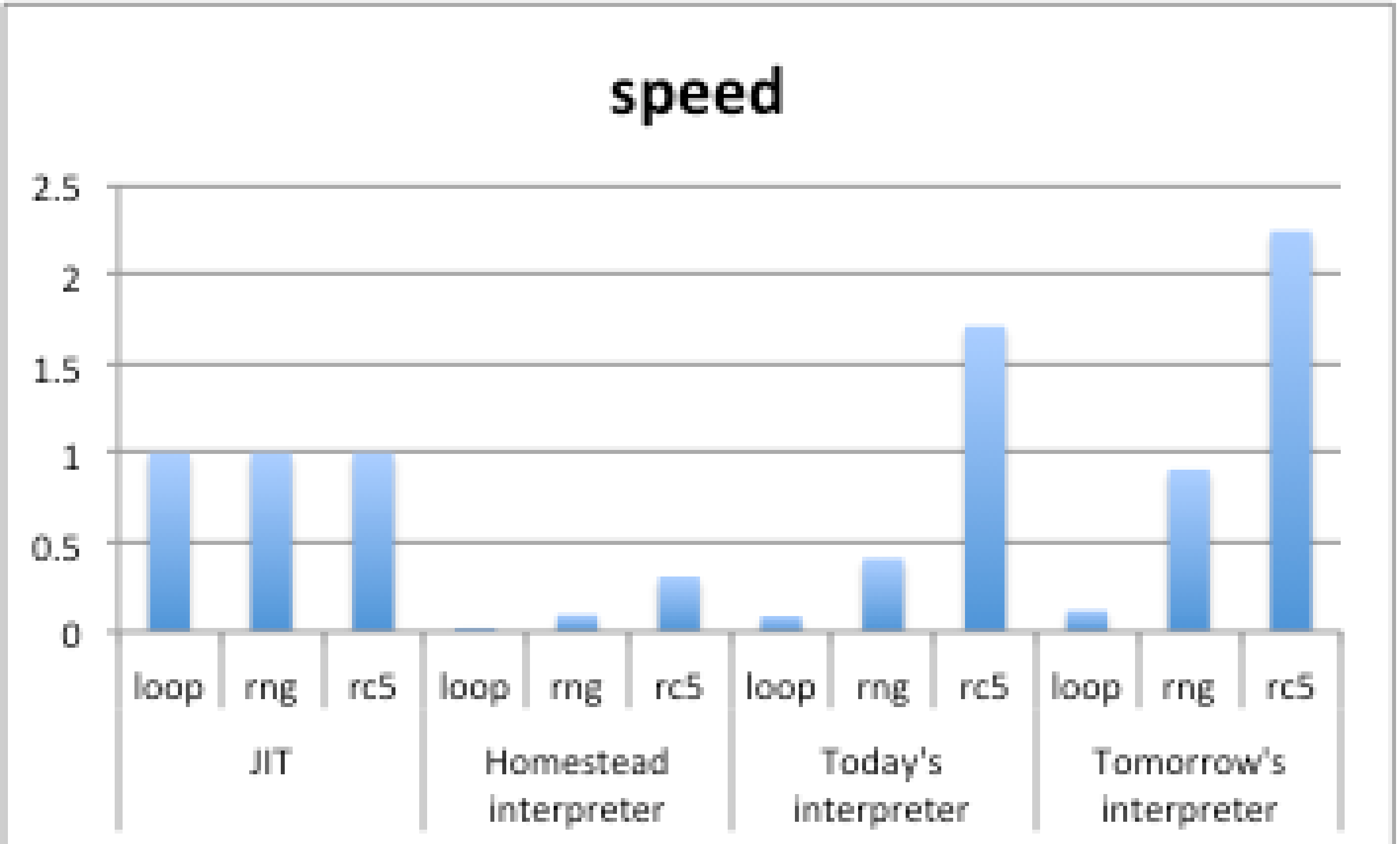

speed
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
loop
rng
rc5
loop
rng
rc5
loop
rng
rc5
JIT
Homestead interpreter
Today's interpreter

General approach

- New, faster opcodes.
- Constant pool – 256 native-format constants.
- *Indirect threading*

Specific changes

- PUSHC. Assign from const pool to stack.
- Use index into pool after opcode, not constant
- JUMPV, JUMPVI. Verify target of jump at load time rather than run time


speed
2.5
2
1.5
1
0.5
0
loop
rng
rc5
loop
rng
rc5
loop
rng
rc5
loop
rng
rc5
JIT
Homestead interpreter
Today's interpreter
Tomorrow's interpreter

Edelbrock
MID MICHIGAN MOTORPLEX
MID MICHIGAN MOTORPLEX
MID MICHIGAN MOTORPLEX
Race 1
RoadTestTV


TEST
526

Fine tuning

- Better 256-bit arithmetic library. (GMP vs. Boost)
- Opcodes for 64-bit arithmetic
- Opcodes for SIMD arithmetic

Structural changes

- Fast conversion of stack code to register code
- 256 registers indexed by single byte
- Code becomes 3-address code


Groutaone
walls
walls
Free
Estimates
Ph: 668-1016
BATH FITTER
CHRIS DERKSEN

loop.sol

```
for (int i = 0; i < 10000000; ++i) {}
```

rng.sol

```
for (int i = 0; i < 1000000; ++i) {
   rand1 = 0xffe7649d5eca8417 * rand1 + 0xf47ed85c4b9a6379;
   rand2 = 0xf8e5dd9a5c994bba * rand2 + 0xf91d87e4b8b74e55;
   rand3 = 0xff97f6f3b29cda52 * rand3 + 0xf393ada8dd75c938;
   rand4 = 0xfe8d437c45bb3735 * rand3 + 0xf47d9a7b5428ffec;
}
```

rc5.sol

```
function rotate_left(uint v, uint n) returns (uint) {
   return v << n | v >> (256 - n);
}

function rotate_right(uint v, uint n) returns (uint) {
   return v >> n | v << (256 - n);
}

function encrypt(uint[] message) {
   for (uint i = 0; i < message.length; i += 2) {
      uint A = message[i];
      uint B = message[i+1];
      A += Key[0];
      B += Key[1];
      for (uint j = 1; j < 16; ++j) {
         A = rotate_left((A ^ B), B) + Key[2 * j];
         B = rotate_left((B ^ A), A) + Key[2 * j + 1];
      }
      message[i] = A;
      message[i+1] = B;
   }
}
```