Recent compiler optimizations

Reini Urban, YAPC::EU 2013

shootout nbody *(2012)*

Fortran	14.09s
C	20.72s
Go	32.11s
SBCL	42.75s
Javascript V8	44.78s
JRuby	8m
PHP	11m
Python 3	16m
Perl	**23m**
Ruby 1.9	26m
perlcc -O -O1	

nbody N=50.000.000

Fortran	14.09s
C	20.72s
Go	32.11s
SBCL	42.75s
Javascript V8	44.78s
perlcc -O -O1	**3m30s**
JRuby	8m
PHP	11m
Python 3	16m
Perl	**23m**
Ruby 1.9	26m

nbody N=50.000.000

Fortran	14.09s
C	20.72s
Go	32.11s
SBCL	42.75s
Javascript V8	44.78s
perlcc -O -O1	**3m30s**
JRuby	8m
PHP	11m
Python 3	16m
Perl	**23m**
Ruby 1.9	26m

6.5x

more perl and perlcc like implementations

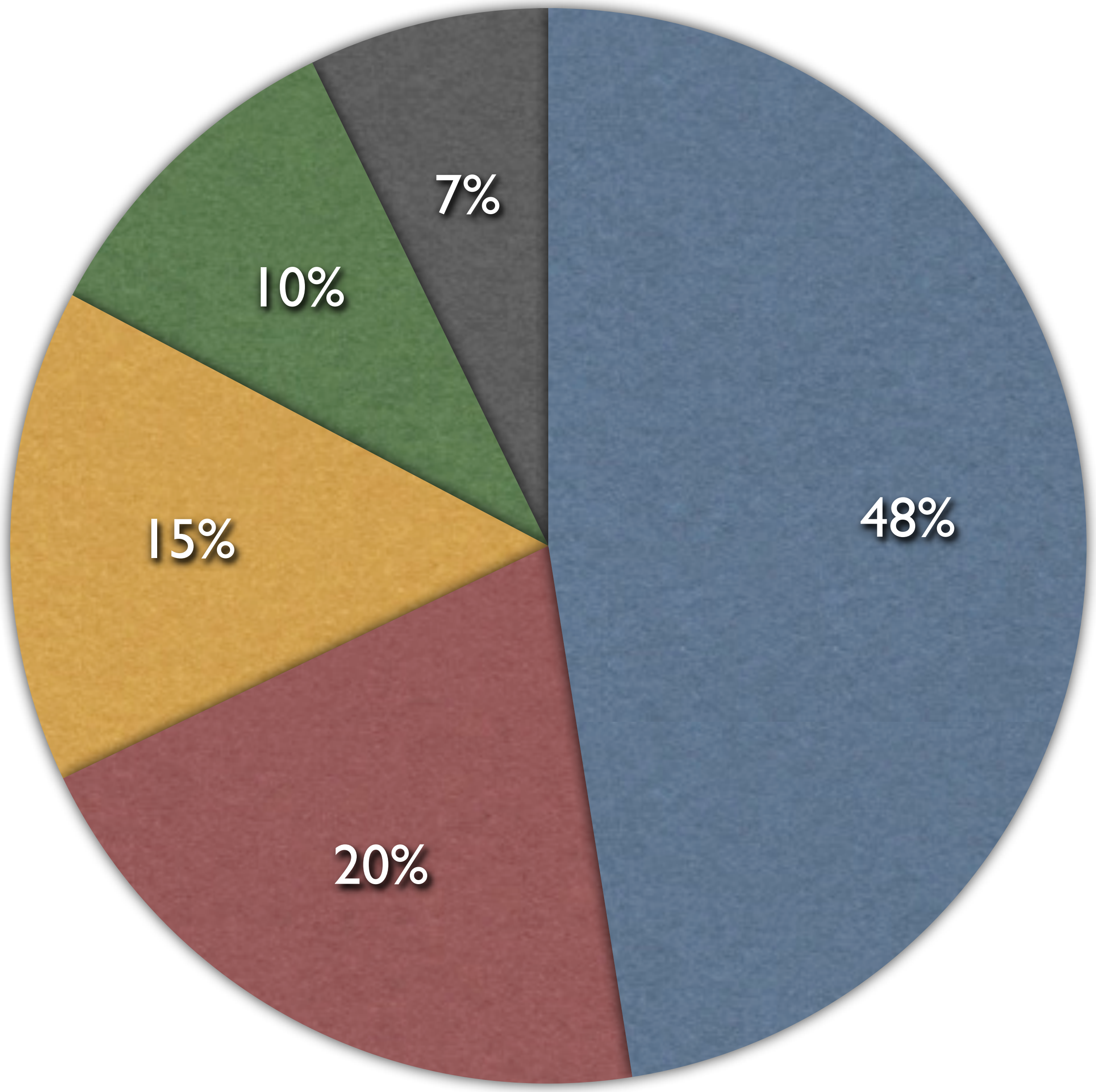
nbody N=50.000.000

C	20.72s
Java	22.52s
Go	32.11s
SBCL	42.75s
Javascript V8	44.78s
Dart	57.08s
pypy	74.38s
Erlang	119.84s
perlcc -O -O1	**3m30s**
Lua	7m
JRuby	8m
PHP	11m
Python 3	16m
Perl	**23m**

B::CC rundown

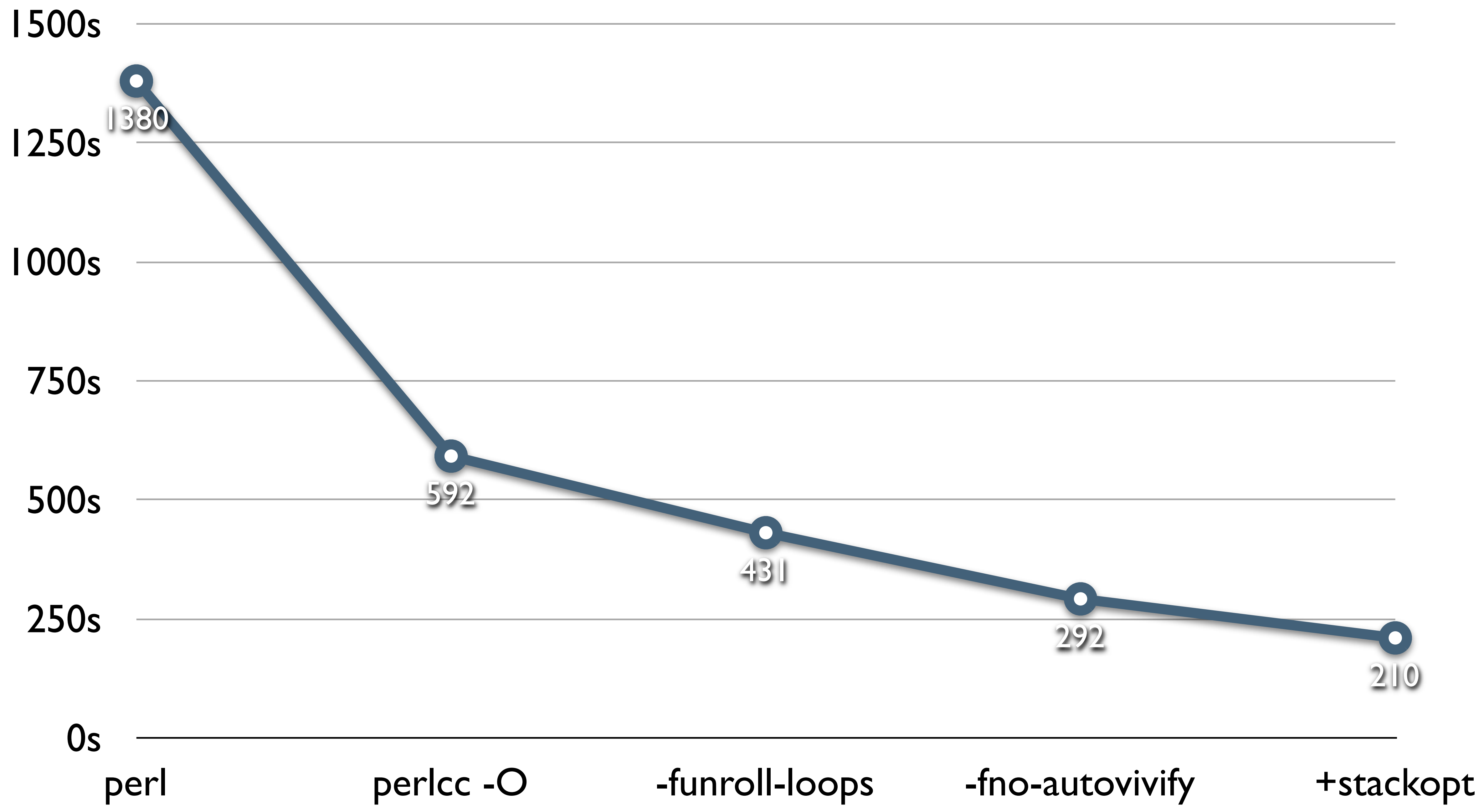
• perl	23m
• perlcc -O	9m52s
• perl -funroll-loops	14m13s
• perlcc -O -funroll-loops	7m11s
• perlcc -O -funroll-loops -fno-autovivify	4m52s
• perlcc -O -O1 (-fno-magic)	3m30s
• + *new aelem stackopt + -fno-cop*	*~2m36s*

B::CC rundown



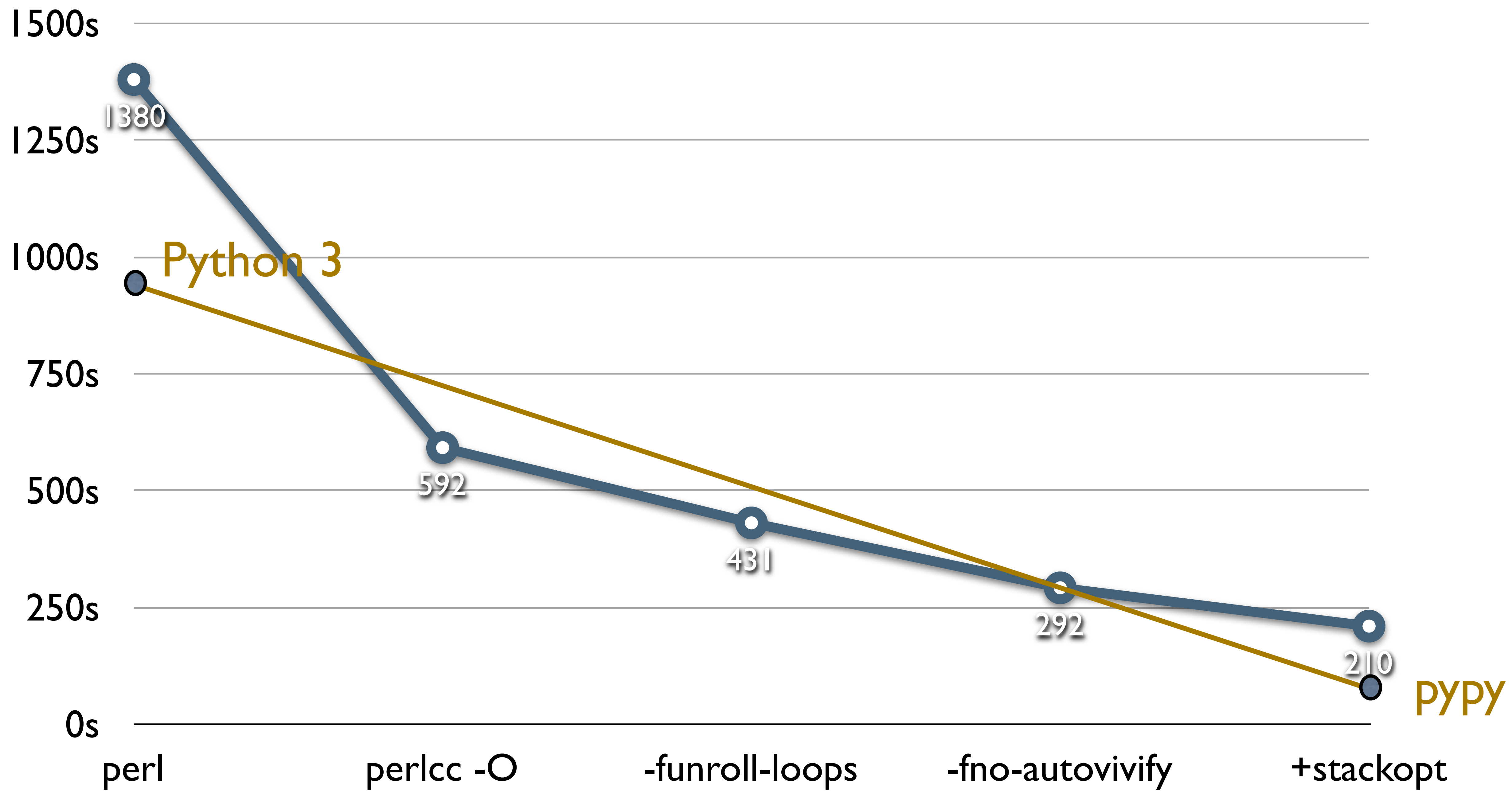
- ● perl
- ● perlcc -O
- ● -funroll-loops
- ● -fno-autovivify
- ● +stackopt

B::CC rundown

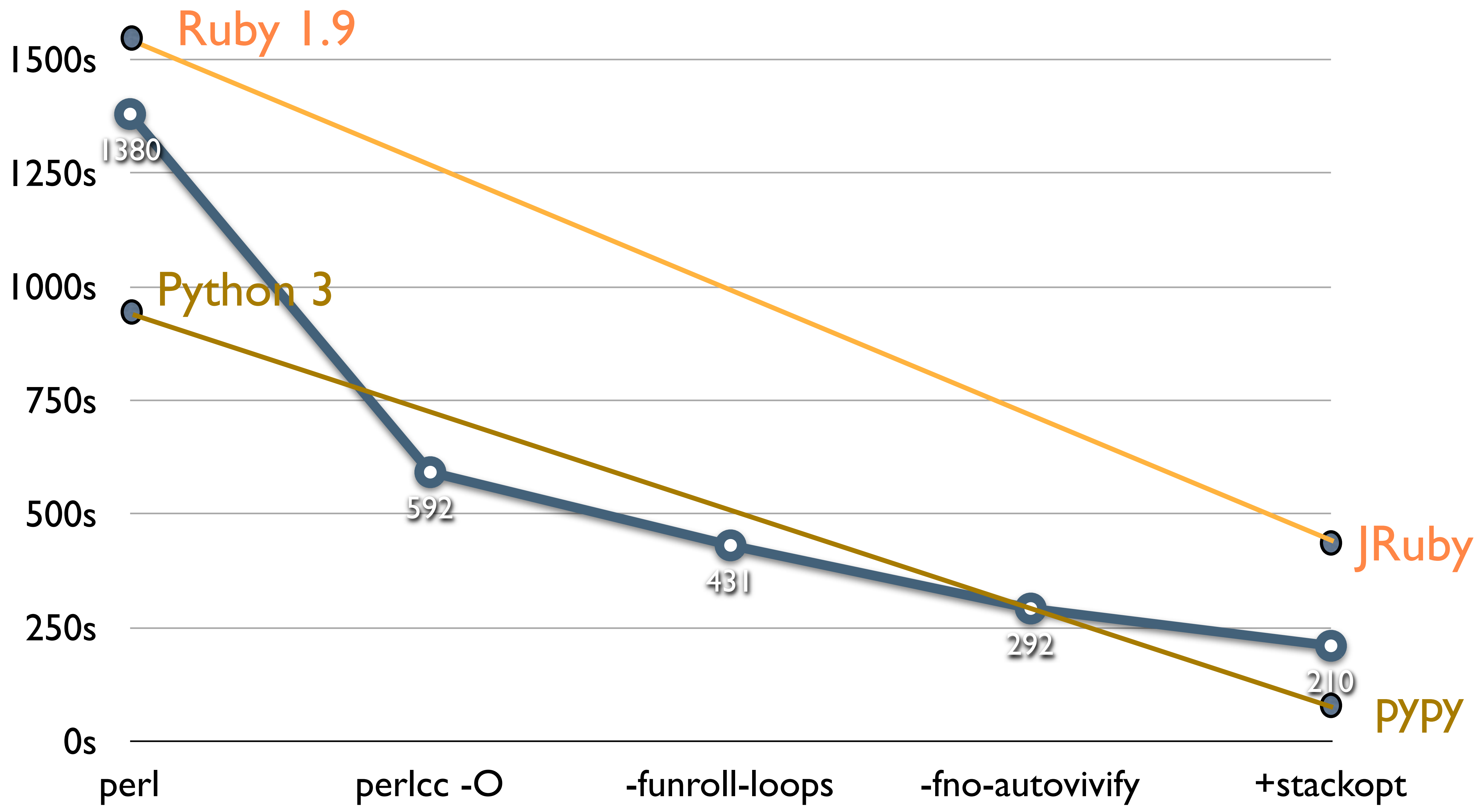


1500s
1250s
1000s
750s
500s
250s
0s
1380
592
431
292
210
perl
perlcc -O
-funroll-loops
-fno-autovivify
+stackopt

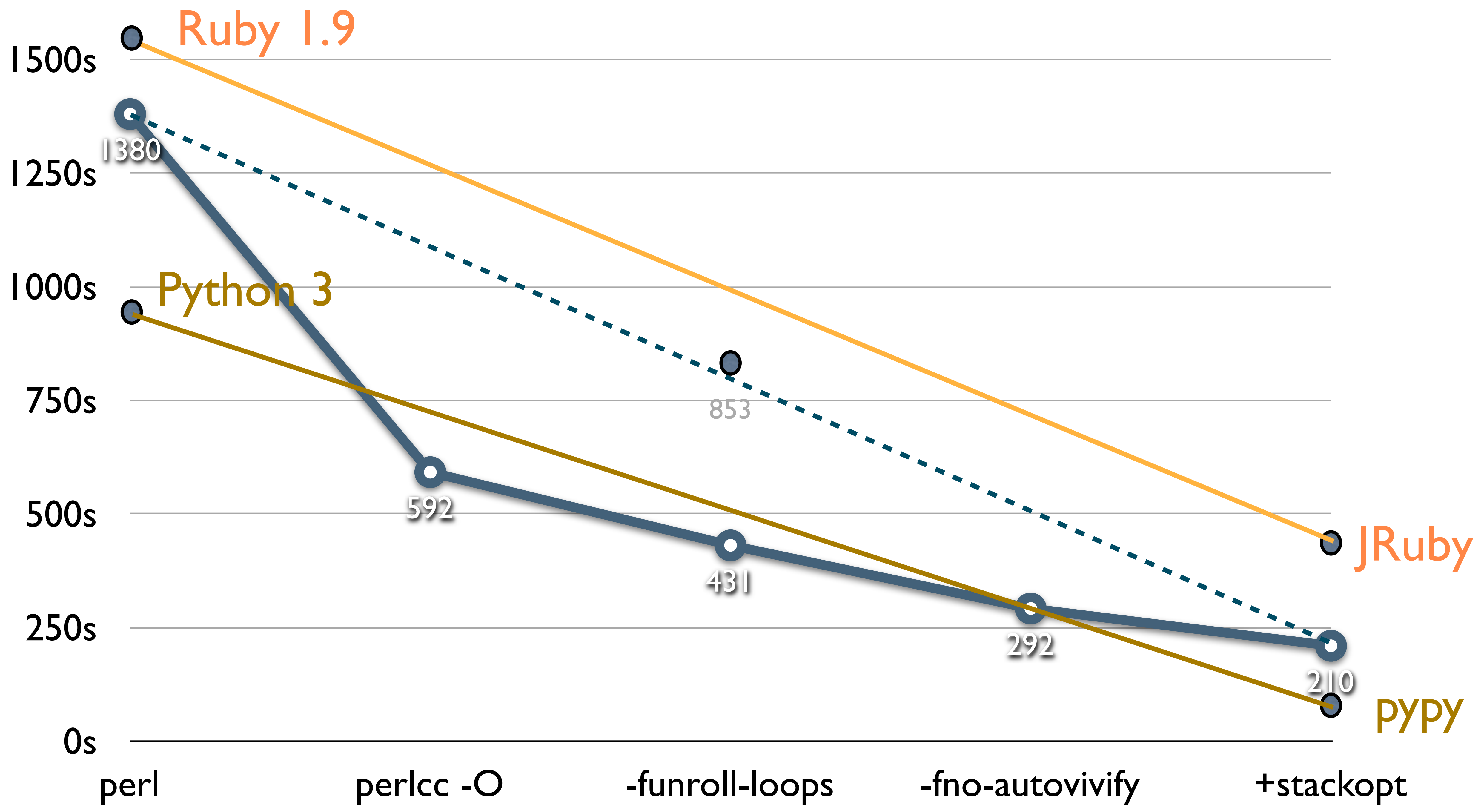
B::CC rundown


1500s
1250s
1000s
750s
500s
250s
0s
1380
592
431
292
210
Python 3
pypy
perl
perlcc -O
-funroll-loops
-fno-autovivify
+stackopt

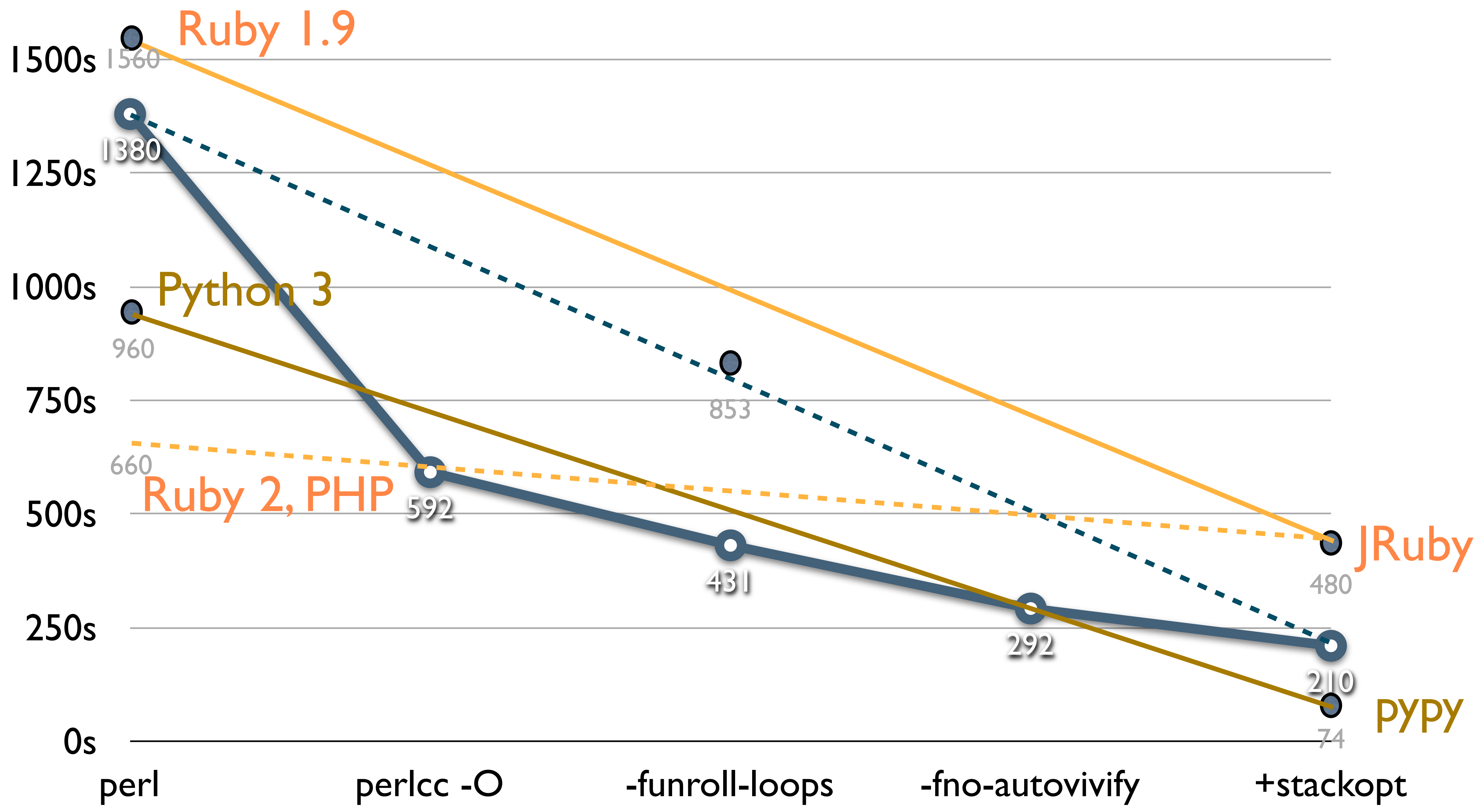
B::CC rundown


Ruby 1.9
Python 3
JRuby
pypy
1500s
1250s
1000s
750s
500s
250s
0s
1380
592
431
292
210
perl
perlcc -O
-funroll-loops
-fno-autovivify
+stackopt

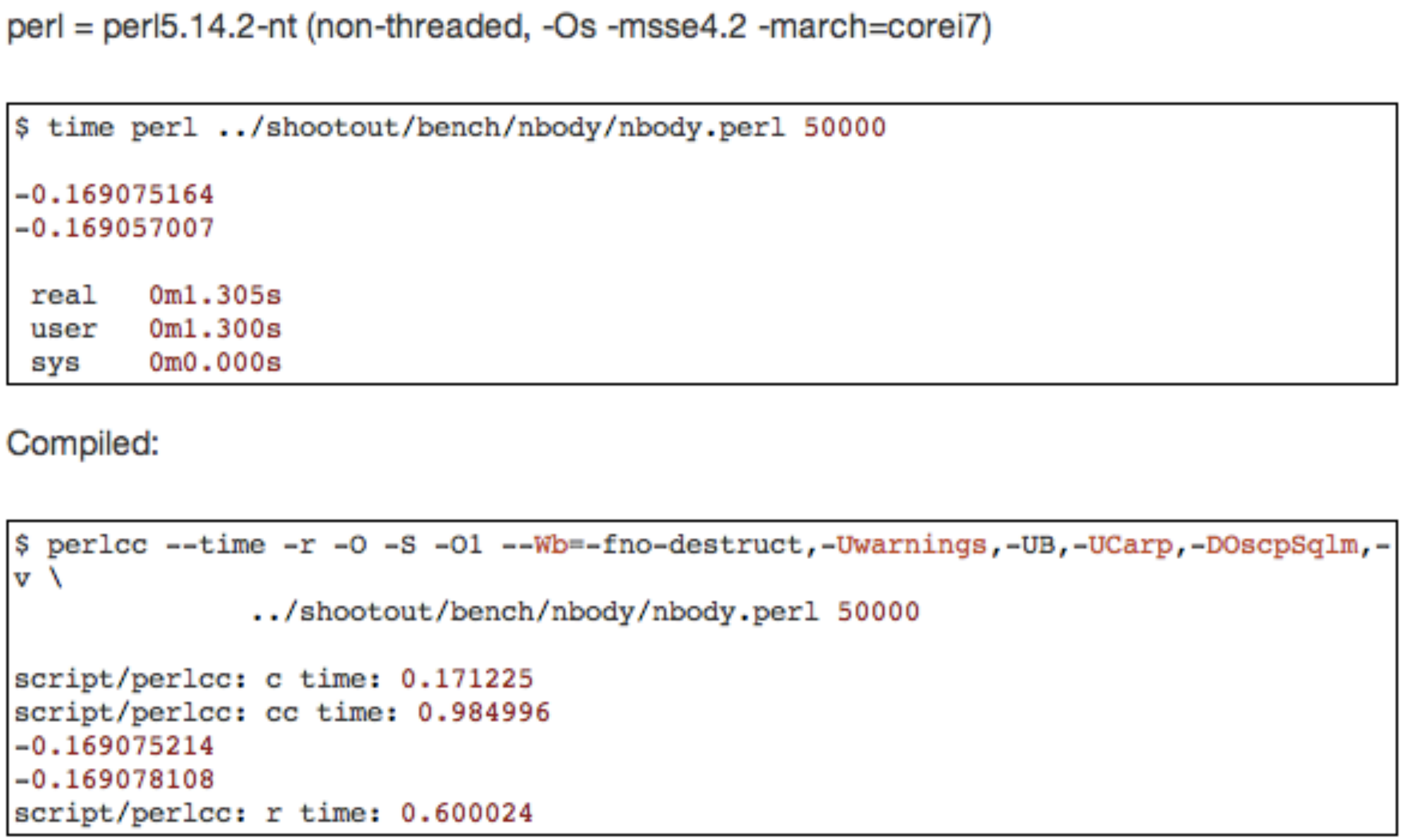
B::CC rundown


Ruby 1.9
Python 3
JRuby
pypy
1500s
1250s
1000s
750s
500s
250s
0s
1380
592
853
431
292
210
perl
perlcc -O
-funroll-loops
-fno-autovivify
+stackopt

B::CC rundown


Ruby 1.9
1500s
1560
1380
1250s
1000s
Python 3
960
750s
853
660
Ruby 2, PHP
592
500s
431
JRuby
480
250s
292
210
pypy
74
0s
perl
perlcc -O
-funroll-loops
-fno-autovivify
+stackopt

B::CC cmdline



perl = perl5.14.2-nt (non-threaded, -Os -msse4.2 -march=corei7)

```
$ time perl ../shootout/bench/nbody/nbody.perl 50000

-0.169075164
-0.169057007

 real    0m1.305s
 user    0m1.300s
 sys     0m0.000s
```

Compiled:

```
$ perlcc --time -r -O -S -O1 --Wb=-fno-destruct,-Uwarnings,-UB,-UCarp,-DOscpSqlm,-
v \
              ../shootout/bench/nbody/nbody.perl 50000

script/perlcc: c time: 0.171225
script/perlcc: cc time: 0.984996
-0.169075214
-0.169078108
script/perlcc: r time: 0.600024
```

1.305s => 0.600s   ~2x faster

```perl
# The Computer Language Shootout
# http://shootout.alioth.debian.org/
#
# contributed by Christoph Bauer
# converted into Perl by Márton Papp
# fixed and cleaned up by Danny Sauer
# optimized by Jesse Millikan

use constant PI            => 3.141592653589793;
use constant SOLAR_MASS    => (4 * PI * PI);
use constant DAYS_PER_YEAR => 365.24;

#  Globals for arrays... Oh well.
#  Almost every iteration is a range, so I keep the last index rather than a
count.
my (@xs, @ys, @zs, @vxs, @vys, @vzs, @mass, $last);

sub advance($)
{
  my ($dt) = @_;
  my ($mm, $mm2, $j, $dx, $dy, $dz, $distance, $mag);

  #  This is faster in the outer loop...
  for (0..$last) {
  #  But not in the inner loop. Strange.
    for ($j = $_ + 1; $j < $last + 1; $j++) {
      $dx = $xs[$_] - $xs[$j];
      $dy = $ys[$_] - $ys[$j];
      $dz = $zs[$_] - $zs[$j];
      $distance = sqrt($dx * $dx + $dy * $dy + $dz * $dz);
      $mag = $dt / ($distance * $distance * $distance);
      $mm = $mass[$_] * $mag;
      $mm2 = $mass[$j] * $mag;
      $vxs[$_] -= $dx * $mm2;
      $vxs[$j] += $dx * $mm;
      $vys[$_] -= $dy * $mm2;
      $vys[$j] += $dy * $mm;
      $vzs[$_] -= $dz * $mm2;
      $vzs[$j] += $dz * $mm;
    }
    # We're done with planet $_ at this point
    # This could be done in a seperate loop, but it's slower
    $xs[$_] += $dt * $vxs[$_];
    $ys[$_] += $dt * $vys[$_];
    $zs[$_] += $dt * $vzs[$_];
  }
}

sub energy
{
  my ($e, $i, $dx, $dy, $dz, $distance);

  $e = 0.0;
  for $i (0..$last) {
    $e += 0.5 * $mass[$i] *
          ($vxs[$i] * $vxs[$i] + $vys[$i] * $vys[$i] + $vzs[$i] * $vzs[$i]);
    for ($i + 1..$last) {
      $dx = $xs[$i] - $xs[$_];
      $dy = $ys[$i] - $ys[$_];
      $dz = $zs[$i] - $zs[$_];
      $distance = sqrt($dx * $dx + $dy * $dy + $dz * $dz);
      $e -= ($mass[$i] * $mass[$_]) / $distance;
    }
  }
  return $e;
}

sub offset_momentum
{
  my ($px, $py, $pz) = (0.0, 0.0, 0.0);

  for (0..$last) {
    $px += $vxs[$_] * $mass[$_];
    $py += $vys[$_] * $mass[$_];
    $pz += $vzs[$_] * $mass[$_];
  }
  $vxs[0] = - $px / SOLAR_MASS;
  $vys[0] = - $py / SOLAR_MASS;
  $vzs[0] = - $pz / SOLAR_MASS;
}
```

```
# @ns = ( sun, jupiter, saturn, uranus, neptune )
@xs = (0, 4.84143144246472090e+00, 8.34336671824457987e+00,
1.28943695621391310e+01, 1.53796971148509165e+01);
@ys = (0, -1.16032004402742839e+00, 4.12479856412430479e+00,
-1.51111514016986312e+01, -2.59193146099879641e+01);
@zs = (0, -1.03622044471123109e-01, -4.03523417114321381e-01,
-2.23307578892655734e-01, 1.79258772950371181e-01);
@vxs = map {$_ * DAYS_PER_YEAR}
  (0, 1.66007664274403694e-03, -2.76742510726862411e-03, 2.96460137564761618e-03,
2.68067772490389322e-03);
@vys = map {$_ * DAYS_PER_YEAR}
  (0, 7.69901118419740425e-03, 4.99852801234917238e-03, 2.37847173959480950e-03,
1.62824170038242295e-03);
@vzs = map {$_ * DAYS_PER_YEAR}
  (0, -6.90460016972063023e-05, 2.30417297573763929e-05, -2.96589568540237556e-05,
-9.51592254519715870e-05);
@mass = map {$_ * SOLAR_MASS}
  (1, 9.54791938424326609e-04, 2.85885980666130812e-04, 4.36624404335156298e-05,
5.15138902046611451e-05);

$last = @xs - 1;

offset_momentum();
printf ("%.9f\n", energy());

my $n = $ARGV[0];

# This does not, in fact, consume N*4 bytes of memory
for (1..$n){
  advance(0.01);
}

printf ("%.9f\n", energy());
```

```
static
CCPP(pp_sub_energy)
{
    double rnv0, lnv0, d1_e, d2_i, d3_dx, d4_dy, d5_dz, d6_distance, d11_tmp,
d13_tmp,
           d15_tmp, d16_tmp, d18_tmp, d19_tmp, d20_tmp, d22_tmp, d31_tmp, d32_tmp,
d33_tmp,
           d34_tmp, d35_tmp, d37_tmp, d38_tmp;
    SV *sv, *src, *dst, *left, *right;
    PERL_CONTEXT *cx;
    MAGIC *mg;
    I32 oldsave, gimme;
    dSP;
    /* init_pp: pp_sub_energy */
    /* load_pad: 39 names, 39 values */
    /* PL_curpad[1] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[1] iv=i1_e
nv=d1_e */
    /* PL_curpad[2] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[2] iv=i2_i
nv=d2_i */
    /* PL_curpad[3] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[3] iv=i3_dx
nv=d3_dx */
    /* PL_curpad[4] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[4] iv=i4_dy
nv=d4_dy */
    /* PL_curpad[5] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[5] iv=i5_dz
nv=d5_dz */
    /* PL_curpad[6] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[6]
iv=i6_distance nv=d6_distance */
    /* PL_curpad[7] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[7]
iv=i7_last nv=d7_last */
    /* PL_curpad[8] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
sv=PL_curpad[8] iv=i8_tmp nv=d8_tmp */
    /* PL_curpad[9] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
sv=PL_curpad[9] iv=i9_tmp nv=d9_tmp */
    /* PL_curpad[10] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[10]
iv=i10_tmp nv=d10_tmp */
    /* PL_curpad[11] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
sv=PL_curpad[11] iv=i11_tmp nv=d11_tmp */
    /* PL_curpad[12] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[12]
iv=i12_tmp nv=d12_tmp */
    /* PL_curpad[13] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
sv=PL_curpad[13] iv=i13_tmp nv=d13_tmp */
    /* PL_curpad[14] = Padsv type=T_UNKNOWN flags=VALID_SV sv=PL_curpad[14]
iv=i14_tmp nv=d14_tmp */
    /* PL_curpad[15] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
sv=PL_curpad[15] iv=i15_tmp nv=d15_tmp */
    /* PL_curpad[16] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
    /* PL_curpad[39] = Padsv type=T_UNKNOWN flags=VALID_SV|REGISTER|TEMPORARY
sv=PL_curpad[39] iv=i39_tmp nv=d39_tmp */
  lab_1fd4ba0:  /* nextstate */
    /* stack =  */
    /* COP (0x1fd4ba0) nextstate [0] */
    /* ../shootout/bench/nbody/nbody.perl:51 */
    TAINT_NOT;
    sp = PL_stack_base + cxstack[cxstack_ix].blk_oldsp;
    FREETMPS;
    /* write_back_stack() 0 called from B::CC::compile_bblock */
  lab_1fd4a10:  /* pushmark */
    /* stack =  */
    /* OP (0x1fd4a10) pushmark [0] */
    /* write_back_stack() 0 called from B::CC::pp_pushmark */
    PUSHMARK(sp);
    /* stack =  */
    /* OP (0x1fd4960) padsv [1] */
    SAVECLEARSV(PL_curpad[1]);
    /* stack = PL_curpad[1] */
    /* OP (0x1fd49c0) padsv [2] */
    SAVECLEARSV(PL_curpad[2]);
    /* stack = PL_curpad[1] PL_curpad[2] */
    /* OP (0x1fd4a40) padsv [3] */
    SAVECLEARSV(PL_curpad[3]);
    /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] */
    /* OP (0x1fd4a90) padsv [4] */
    SAVECLEARSV(PL_curpad[4]);
    /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] PL_curpad[4] */
    /* OP (0x1fd4990) padsv [5] */
    SAVECLEARSV(PL_curpad[5]);
    /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] PL_curpad[4] PL_curpad[5] */
    /* OP (0x1fd4930) padsv [6] */
    SAVECLEARSV(PL_curpad[6]);
    /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] PL_curpad[4] PL_curpad[5]
PL_curpad[6] */
    /* LISTOP (0x1e99820) list [0] */
    /* list */
    /* write_back_stack() 6 called from B::CC::pp_list */
    EXTEND(sp, 6);
    PUSHs((SV*)PL_curpad[1]);
    PUSHs((SV*)PL_curpad[2]);
    PUSHs((SV*)PL_curpad[3]);
    PUSHs((SV*)PL_curpad[4]);
    PUSHs((SV*)PL_curpad[5]);
    PUSHs((SV*)PL_curpad[6]);
    PP_LIST(1);
```

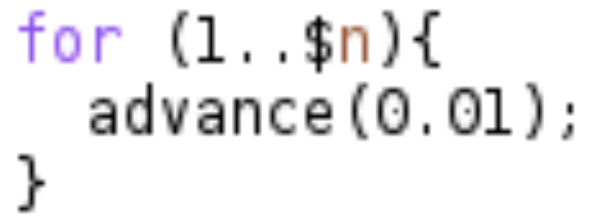
```
lab_1fffd30:    /* nextstate */
/* ../shootout/bench/nbody/nbody.perl:61 */
TAINT_NOT;
sp = PL_stack_base + cxstack[cxstack_ix].blk_oldsp;
FREETMPS;
/* stack =  */
/* OP (0x1fd5260) padsv [3] */
/* stack = PL_curpad[3] */
/* OP (0x1fd5290) padsv [3] */
/* stack = PL_curpad[3] PL_curpad[3] */
/* BINOP (0x1fd51c0) multiply [31] */
d3_dx = SvNV(PL_curpad[3]);
rnv0 = d3_dx; lnv0 = d3_dx; /* multiply */
d31_tmp = lnv0 * rnv0;
/* stack = d31_tmp */
/* OP (0x1fffaf0) padsv [4] */
/* stack = d31_tmp PL_curpad[4] */
/* OP (0x1fffb20) padsv [4] */
/* stack = d31_tmp PL_curpad[4] PL_curpad[4] */
/* BINOP (0x1fffb50) multiply [32] */
d4_dy = SvNV(PL_curpad[4]);
rnv0 = d4_dy; lnv0 = d4_dy; /* multiply */
d32_tmp = lnv0 * rnv0;
/* stack = d31_tmp d32_tmp */
/* BINOP (0x1fffb90) add [33] */
rnv0 = d32_tmp; lnv0 = d31_tmp; /* add */
d33_tmp = lnv0 + rnv0;
/* stack = d33_tmp */
/* OP (0x1fffbd0) padsv [5] */
/* stack = d33_tmp d5_dz */
/* OP (0x1fffc00) padsv [5] */
/* stack = d33_tmp d5_dz d5_dz */
/* BINOP (0x1fffc30) multiply [34] */
rnv0 = d5_dz; lnv0 = d5_dz; /* multiply */
d34_tmp = lnv0 * rnv0;
/* stack = d33_tmp d34_tmp */
/* BINOP (0x1fffc70) add [35] */
rnv0 = d34_tmp; lnv0 = d33_tmp; /* add */
d35_tmp = lnv0 + rnv0;
/* stack = d35_tmp */
/* UNOP (0x1fffcb0) sqrt [6] */
/* write_back_lexicals(0) called from B::CC::default_pp */
sv_setnv(PL_curpad[5], d5_dz);
sv_setnv(PL_curpad[31], d31_tmp);
sv_setnv(PL_curpad[32], d32_tmp);
```

```
/* stack = d31_tmp PL_curpad[4] */
/* OP (0x1fffb20) padsv [4] */
/* stack = d31_tmp PL_curpad[4] PL_curpad[4] */
/* BINOP (0x1fffb50) multiply [32] */
d4_dy = SvNV(PL_curpad[4]);
rnv0 = d4_dy; lnv0 = d4_dy; /* multiply */
d32_tmp = lnv0 * rnv0;
/* stack = d31_tmp d32_tmp */
/* BINOP (0x1fffb90) add [33] */
rnv0 = d32_tmp; lnv0 = d31_tmp; /* add */
d33_tmp = lnv0 + rnv0;
/* stack = d33_tmp */
/* OP (0x1fffbd0) padsv [5] */
/* stack = d33_tmp d5_dz */
/* OP (0x1fffc00) padsv [5] */
/* stack = d33_tmp d5_dz d5_dz */
/* BINOP (0x1fffc30) multiply [34] */
rnv0 = d5_dz; lnv0 = d5_dz; /* multiply */
d34_tmp = lnv0 * rnv0;
/* stack = d33_tmp d34_tmp */
/* BINOP (0x1fffc70) add [35] */
rnv0 = d34_tmp; lnv0 = d33_tmp; /* add */
d35_tmp = lnv0 + rnv0;
/* stack = d35_tmp */
/* UNOP (0x1fffcb0) sqrt [6] */
/* write_back_lexicals(0) called from B::CC::default_pp */
sv_setnv(PL_curpad[5], d5_dz);
sv_setnv(PL_curpad[31], d31_tmp);
sv_setnv(PL_curpad[32], d32_tmp);
sv_setnv(PL_curpad[33], d33_tmp);
sv_setnv(PL_curpad[34], d34_tmp);
sv_setnv(PL_curpad[35], d35_tmp);
/* write_back_stack() 1 called from B::CC::default_pp */
EXTEND(sp, 1);
PUSHs((SV*)PL_curpad[35]);
PL_op = (OP*)&unop_list[31];
DOOP(PL_ppaddr[OP_SQRT]);
/* invalidate_lexicals(0) called from B::CC::default_pp */
/* stack =  */
```

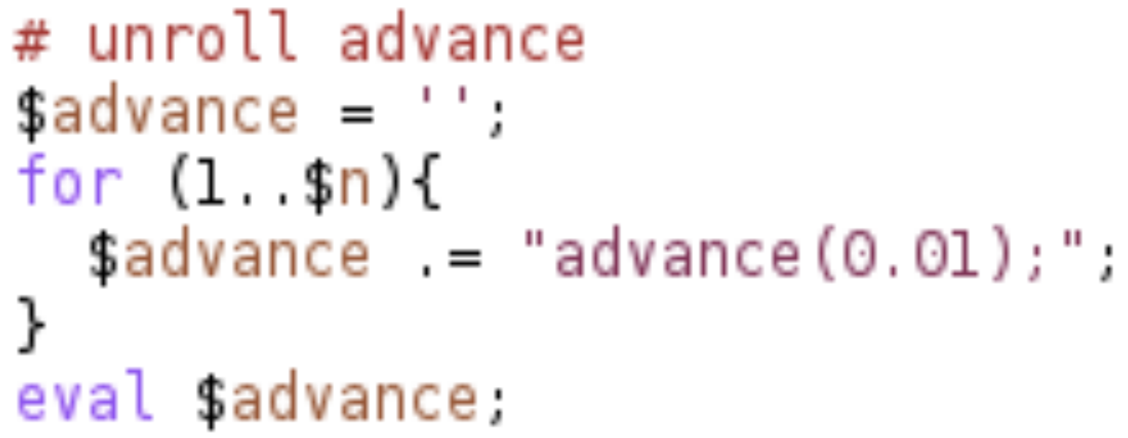
```
 $distance = sqrt($dx * $dx + $dy * $dy + $dz * $dz);
```

Idea 1: Unroll loop / Inline Functions

unroll loop:



```
for (1..$n){
  advance(0.01);
}
```



```
# unroll advance
$advance = '';
for (1..$n){
  $advance .= "advance(0.01);";
}
eval $advance;
```

22m13.754s => 21m48.015s (25s,1.9%)

Idea 1: Unroll loop / Inline Functions

unroll loop:

```
for (1..$n){
  advance(0.01);
}
```

```
# unroll advance
$advance = '';
for (1..$n){
  $advance .= "advance(0.01);";
}
eval $advance;
```

22m13.754s => 21m48.015s ~~(25s, 1.9%)~~

inline function: 3.4%

2: Unroll AELEM to AELEMFAST

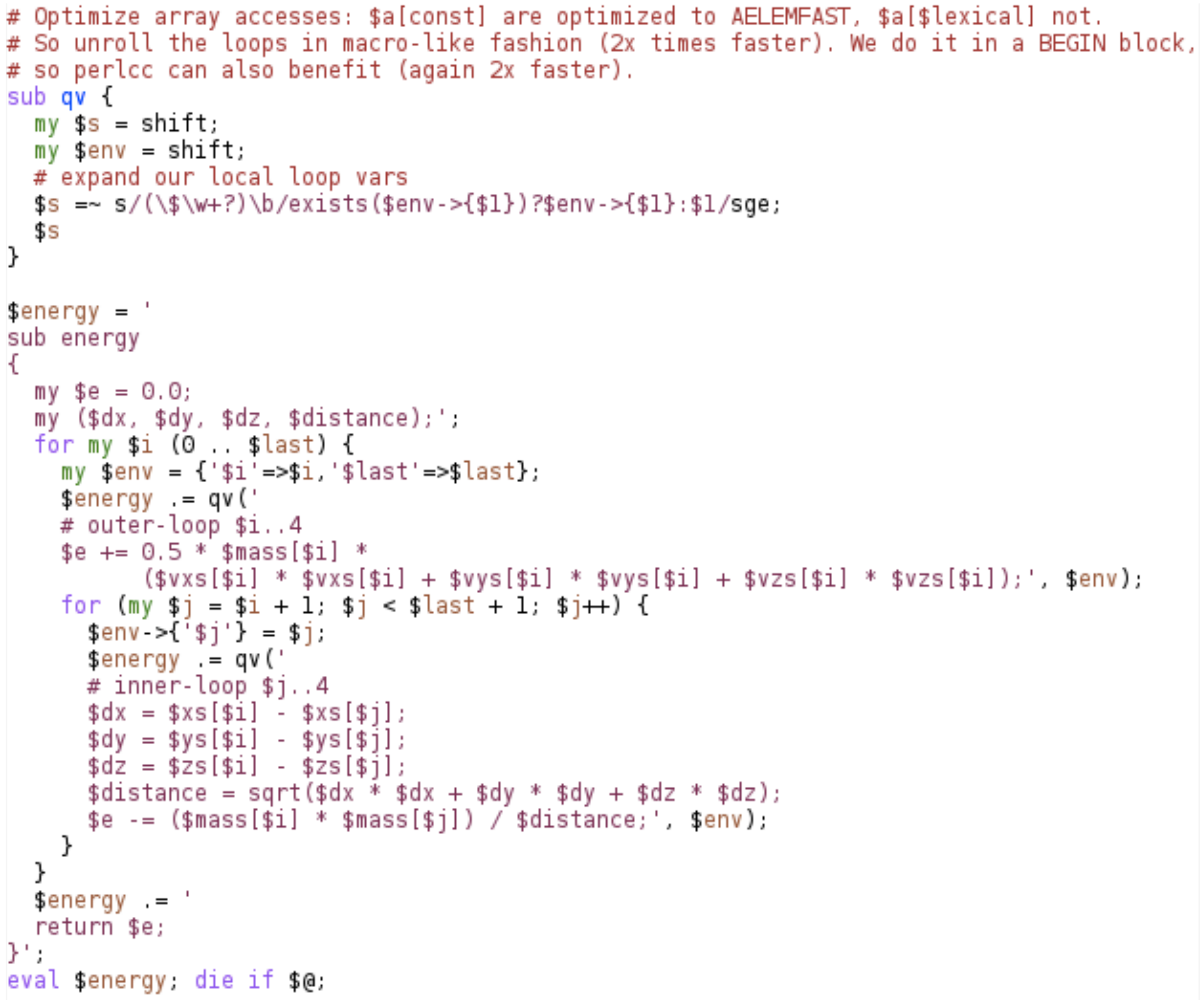
```
for (my $j = $i + 1; $j < $last + 1; $j++) {
  # inner-loop $j..4
  $dx = $xs[$i] - $xs[$j];
  $dy = $ys[$i] - $ys[$j];
  $dz = $zs[$i] - $zs[$j];
  ...
```

```
# Optimize array accesses: $a[const] are optimized to AELEMFAST, $a[$lexical] not.
# So unroll the loops in macro-like fashion (2x times faster). We do it in a BEGIN block
# so perlcc can also benefit (again 2x faster).
$energy = '
sub energy
{
  my $e = 0.0;
  my ($dx, $dy, $dz, $distance);';
for my $i (0 .. $last) {
  $energy .= "
# loop $i..4
    \$e += 0.5 * \$mass[$i] *
          (\$vxs[$i] * \$vxs[$i] + \$vys[$i] * \$vys[$i] + \$vzs[$i] * \$vzs[$i]);
";
  for (my $j = $i + 1; $j < $last + 1; $j++) {
    $energy .= "
    # inner-loop $j..4
    \$dx = \$xs[$i] - \$xs[$j];
    \$dy = \$ys[$i] - \$ys[$j];
    \$dz = \$zs[$i] - \$zs[$j];
    \$distance = sqrt(\$dx * \$dx + \$dy * \$dy + \$dz * \$dz);
    \$e -= (\$mass[$i] * \$mass[$j]) / \$distance;";
  }
}
$energy .= '
  return $e;
}';
eval $energy; die if $@;
```

2: Unroll AELEM to AELEMFAST

```
$ perl -MO=Concise,energy nbody.perl
  ...
  <2> add[t19] sKP/2 ->16
     <2> add[t16] sK/2 ->y
        <2> multiply[t13] sK/2 ->q
           <2> aelem sK/2 ->m
              <0> padav[@vxs:FAKE:] sR ->k
              <0> padsv[$i:111,116] s ->l
           <2> aelem sK/2 ->p
              <0> padav[@vxs:FAKE:] sR ->n
              <0> padsv[$i:111,116] s ->o
vs
$ perl -MO=Concise,energy nbody.perl-2.perl
  ...
  <2> add[t15] sKP/2 ->n
     <2> add[t12] sK/2 ->j
        <2> multiply[t9] sK/2 ->f
           <1> ex-aelem sK/2 ->d
              <0> aelemfast_lex[@vxs:FAKE:] sR ->d
              <0> ex-const s ->-
           <1> ex-aelem sK/2 ->e
              <0> aelemfast_lex[@vxs:FAKE:] sR ->e
              <0> ex-const s ->-
```

2: Unroll AELEM to AELEMFAST



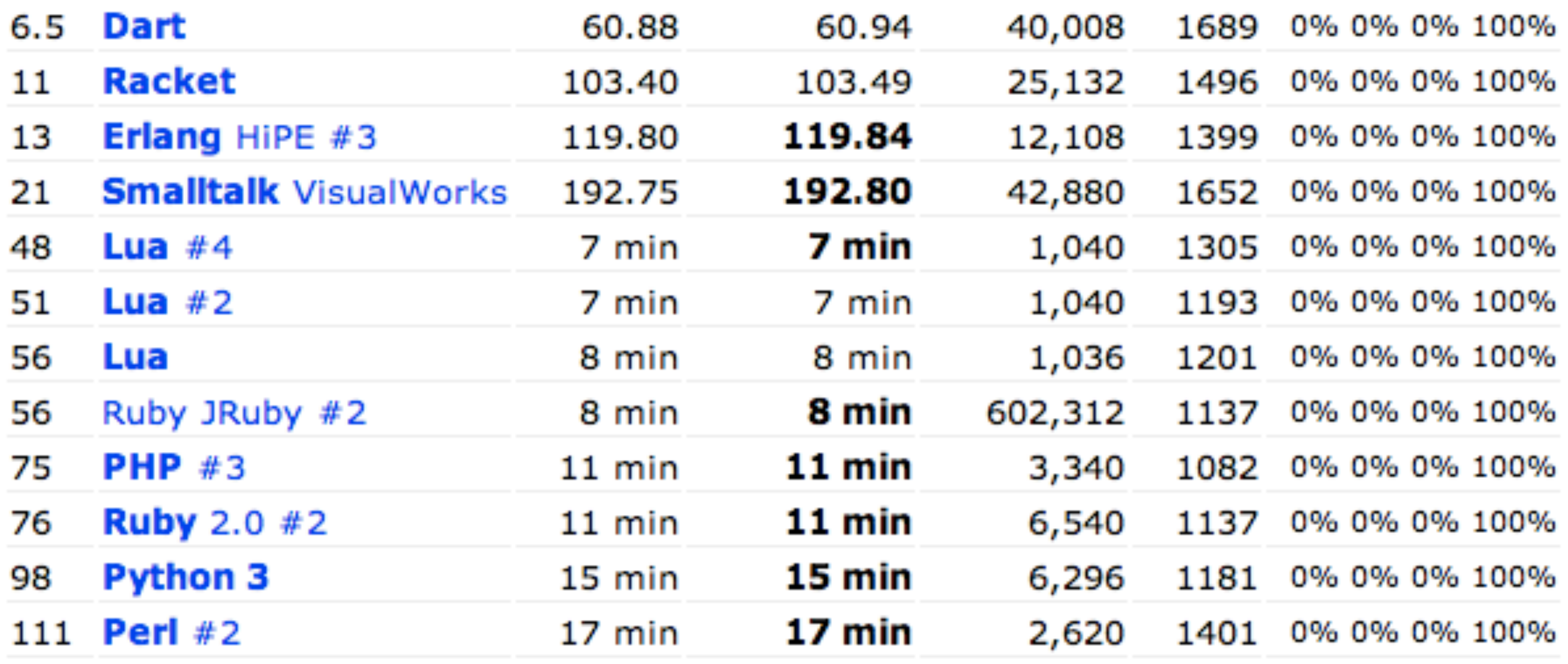
```perl
# Optimize array accesses: $a[const] are optimized to AELEMFAST, $a[$lexical] not.
# So unroll the loops in macro-like fashion (2x times faster). We do it in a BEGIN block,
# so perlcc can also benefit (again 2x faster).
sub qv {
  my $s = shift;
  my $env = shift;
  # expand our local loop vars
  $s =~ s/(\$\w+?)\b/exists($env->{$1})?$env->{$1}:$1/sge;
  $s
}

$energy = '
sub energy
{
  my $e = 0.0;
  my ($dx, $dy, $dz, $distance);';
  for my $i (0 .. $last) {
    my $env = {'$i'=>$i,'$last'=>$last};
    $energy .= qv('
    # outer-loop $i..4
    $e += 0.5 * $mass[$i] *
          ($vxs[$i] * $vxs[$i] + $vys[$i] * $vys[$i] + $vzs[$i] * $vzs[$i]);', $env);
    for (my $j = $i + 1; $j < $last + 1; $j++) {
      $env->{'$j'} = $j;
      $energy .= qv('
      # inner-loop $j..4
      $dx = $xs[$i] - $xs[$j];
      $dy = $ys[$i] - $ys[$j];
      $dz = $zs[$i] - $zs[$j];
      $distance = sqrt($dx * $dx + $dy * $dy + $dz * $dz);
      $e -= ($mass[$i] * $mass[$j]) / $distance;', $env);
    }
  }
  $energy .= '
  return $e;
}';
eval $energy; die if $@;
```

shootout

- pure perl solution
- unroll-loops variant as nbody.perl-2.perl #2
- from 23m to 14m13s *(non-threaded)* *(62%)*

shootout nbody (2013)



6.5	**Dart**	60.88	60.94	40,008	1689	0% 0% 0% 100%
11	**Racket**	103.40	103.49	25,132	1496	0% 0% 0% 100%
13	**Erlang** HiPE #3	119.80	**119.84**	12,108	1399	0% 0% 0% 100%
21	**Smalltalk** VisualWorks	192.75	**192.80**	42,880	1652	0% 0% 0% 100%
48	**Lua** #4	7 min	**7 min**	1,040	1305	0% 0% 0% 100%
51	**Lua** #2	7 min	7 min	1,040	1193	0% 0% 0% 100%
56	**Lua**	8 min	8 min	1,036	1201	0% 0% 0% 100%
56	Ruby JRuby #2	8 min	**8 min**	602,312	1137	0% 0% 0% 100%
75	**PHP** #3	11 min	**11 min**	3,340	1082	0% 0% 0% 100%
76	**Ruby** 2.0 #2	11 min	**11 min**	6,540	1137	0% 0% 0% 100%
98	**Python 3**	15 min	**15 min**	6,296	1181	0% 0% 0% 100%
111	**Perl** #2	17 min	**17 min**	2,620	1401	0% 0% 0% 100%

-funroll-loops (60-27%)

```
git checkout unroll-loops
perldoc lib/B/CC.pm
```

-funroll-loops

Perform loop unrolling when iteration count is known. Changes AELEM to AELEMFAST with known indices when:

* The iteration count is known at compile-time,

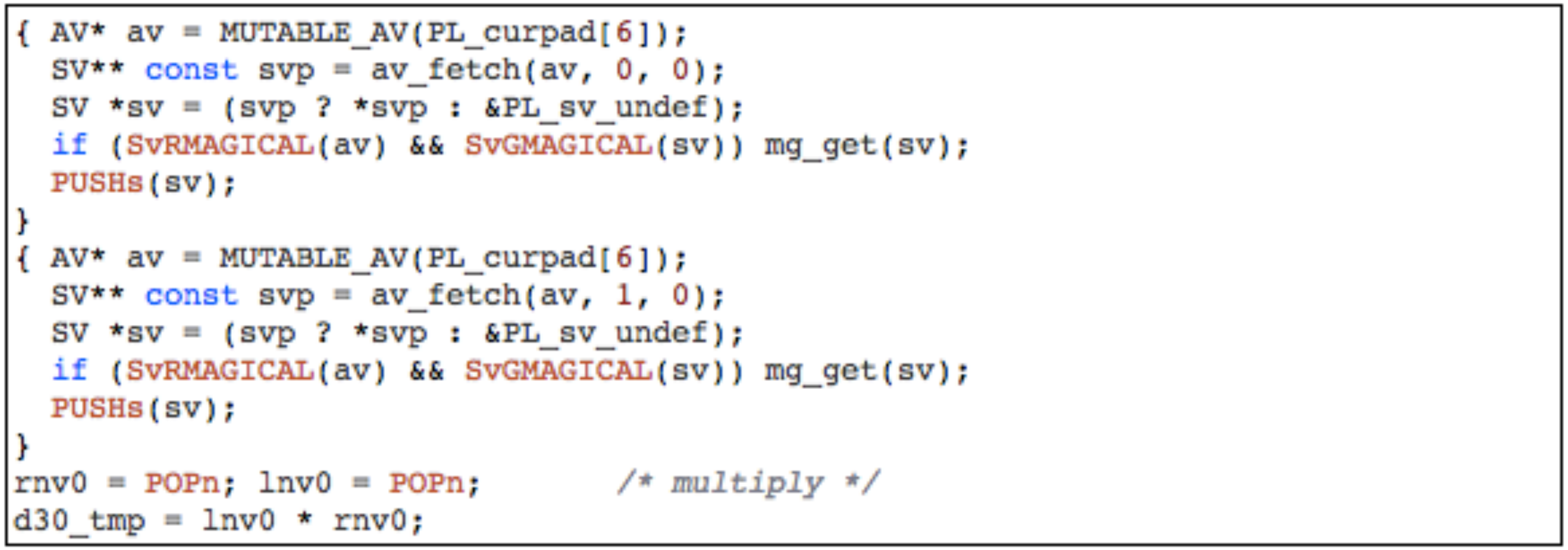
* The maximum iteration count is lower than 256,

* AELEM accesses are detected inside the loop, and the benefit of AELEMFAST outweighs the cost of the unrolling.

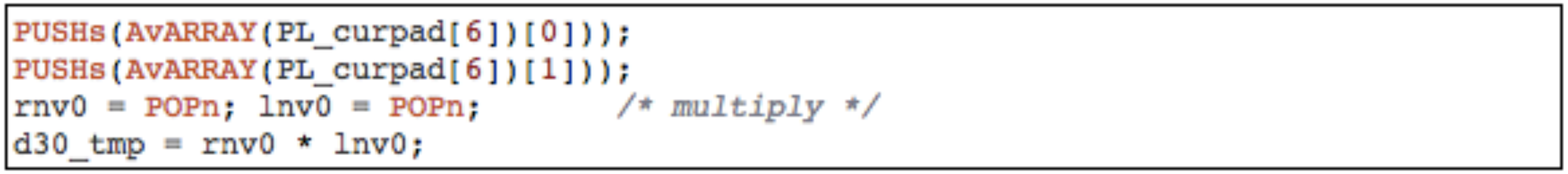
Enabled with -O1.

Idea 3: -fno-autovivify array elems

```
$px += $vxs[$_] * $mass[$_];
```



```
{ AV* av = MUTABLE_AV(PL_curpad[6]);
  SV** const svp = av_fetch(av, 0, 0);
  SV *sv = (svp ? *svp : &PL_sv_undef);
  if (SvRMAGICAL(av) && SvGMAGICAL(sv)) mg_get(sv);
  PUSHs(sv);
}
{ AV* av = MUTABLE_AV(PL_curpad[6]);
  SV** const svp = av_fetch(av, 1, 0);
  SV *sv = (svp ? *svp : &PL_sv_undef);
  if (SvRMAGICAL(av) && SvGMAGICAL(sv)) mg_get(sv);
  PUSHs(sv);
}
rnv0 = POPn; lnv0 = POPn;        /* multiply */
d30_tmp = lnv0 * rnv0;
```



```
PUSHs(AvARRAY(PL_curpad[6])[0]));
PUSHs(AvARRAY(PL_curpad[6])[1]));
rnv0 = POPn; lnv0 = POPn;        /* multiply */
d30_tmp = rnv0 * lnv0;
```

Idea 3: -fno-autovivify array elems

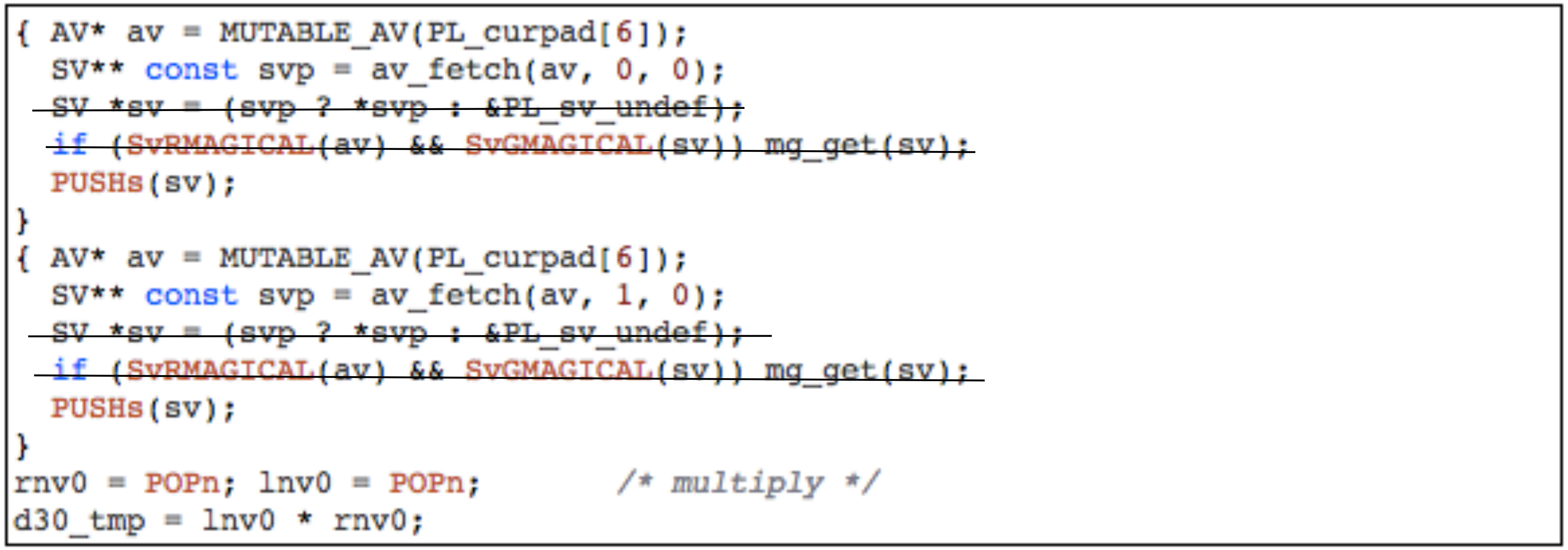
```
$px += $vxs[$_] * $mass[$_];
```

```
{ AV* av = MUTABLE_AV(PL_curpad[6]);
  SV** const svp = av_fetch(av, 0, 0);
  ~~SV *sv = (svp ? *svp : &PL_sv_undef);~~
  if (SvRMAGICAL(av) && SvGMAGICAL(sv)) mg_get(sv);
  PUSHs(sv);
}
{ AV* av = MUTABLE_AV(PL_curpad[6]);
  SV** const svp = av_fetch(av, 1, 0);
  ~~SV *sv = (svp ? *svp : &PL_sv_undef);~~
  if (SvRMAGICAL(av) && SvGMAGICAL(sv)) mg_get(sv);
  PUSHs(sv);
}
rnv0 = POPn; lnv0 = POPn;        /* multiply */
d30_tmp = lnv0 * rnv0;
```

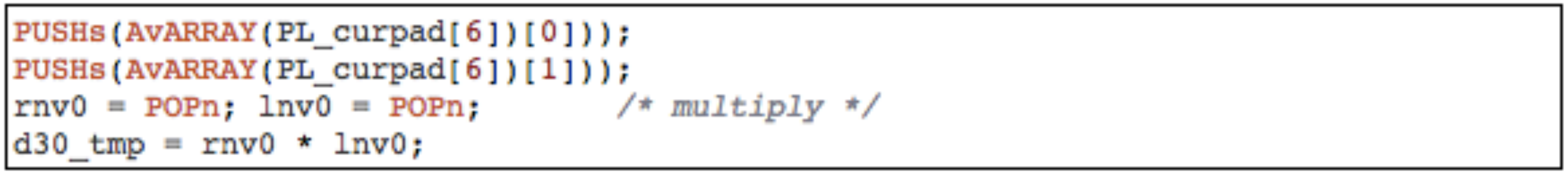
```
PUSHs(AvARRAY(PL_curpad[6])[0]));
PUSHs(AvARRAY(PL_curpad[6])[1]));
rnv0 = POPn; lnv0 = POPn;        /* multiply */
d30_tmp = rnv0 * lnv0;
```

Idea 3: -fno-autovivify, -fno-magic

```perl
$px += $vxs[$_] * $mass[$_];
```



```c
{ AV* av = MUTABLE_AV(PL_curpad[6]);
  SV** const svp = av_fetch(av, 0, 0);
~~SV *sv = (svp ? *svp : &PL_sv_undef);~~
  ~~if (SvRMAGICAL(av) && SvGMAGICAL(sv)) mg_get(sv);~~
  PUSHs(sv);
}
{ AV* av = MUTABLE_AV(PL_curpad[6]);
  SV** const svp = av_fetch(av, 1, 0);
~~SV *sv = (svp ? *svp : &PL_sv_undef);~~
~~if (SvRMAGICAL(av) && SvGMAGICAL(sv)) mg_get(sv);~~
  PUSHs(sv);
}
rnv0 = POPn; lnv0 = POPn;        /* multiply */
d30_tmp = lnv0 * rnv0;
```



```c
PUSHs(AvARRAY(PL_curpad[6])[0]));
PUSHs(AvARRAY(PL_curpad[6])[1]));
rnv0 = POPn; lnv0 = POPn;        /* multiply */
d30_tmp = rnv0 * lnv0;
```

if at compile-time:
  index >=0 && < declared size (autovivify)
  no SVs_RMG magic attached (and -fno-magic asserts not added at run-time)
  no autovivification pragma or perlcc flag

-O1 -fno-autovivify (77%)

-fno-autovivify

Do not vivify array (and soon also hash elements) when accessing them. Beware: Vivified elements default to undef, unvivified elements are invalid.

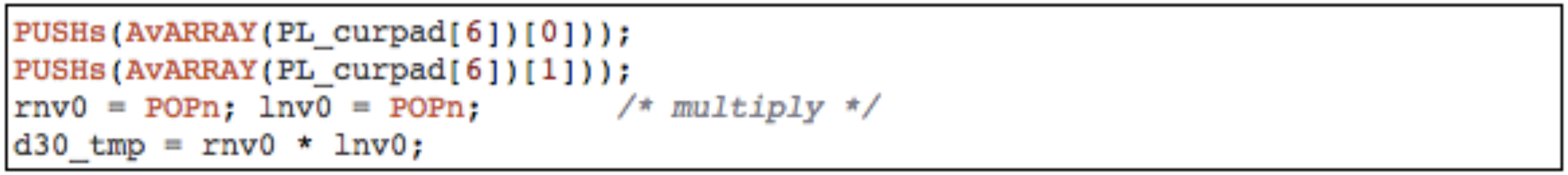
This is similar to the pragma "no autovivification" and allows very fast array accesses, 4-6 times faster, without the overhead of autovivification.pm

-fno-magic (40%)

-fno-magic

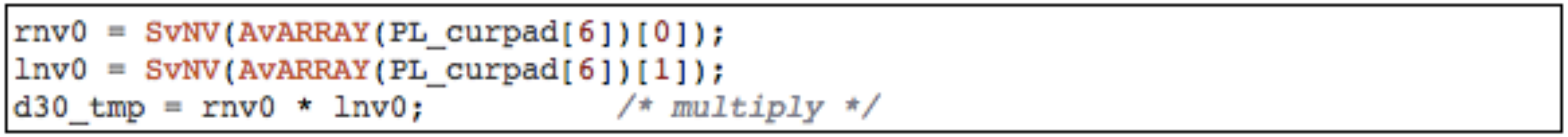
Assume certain data being optimized is never tied at run-time or is holding other magic. This mainly holds for arrays being optimized, but in the future hashes also.

Stack optimizations (20%)



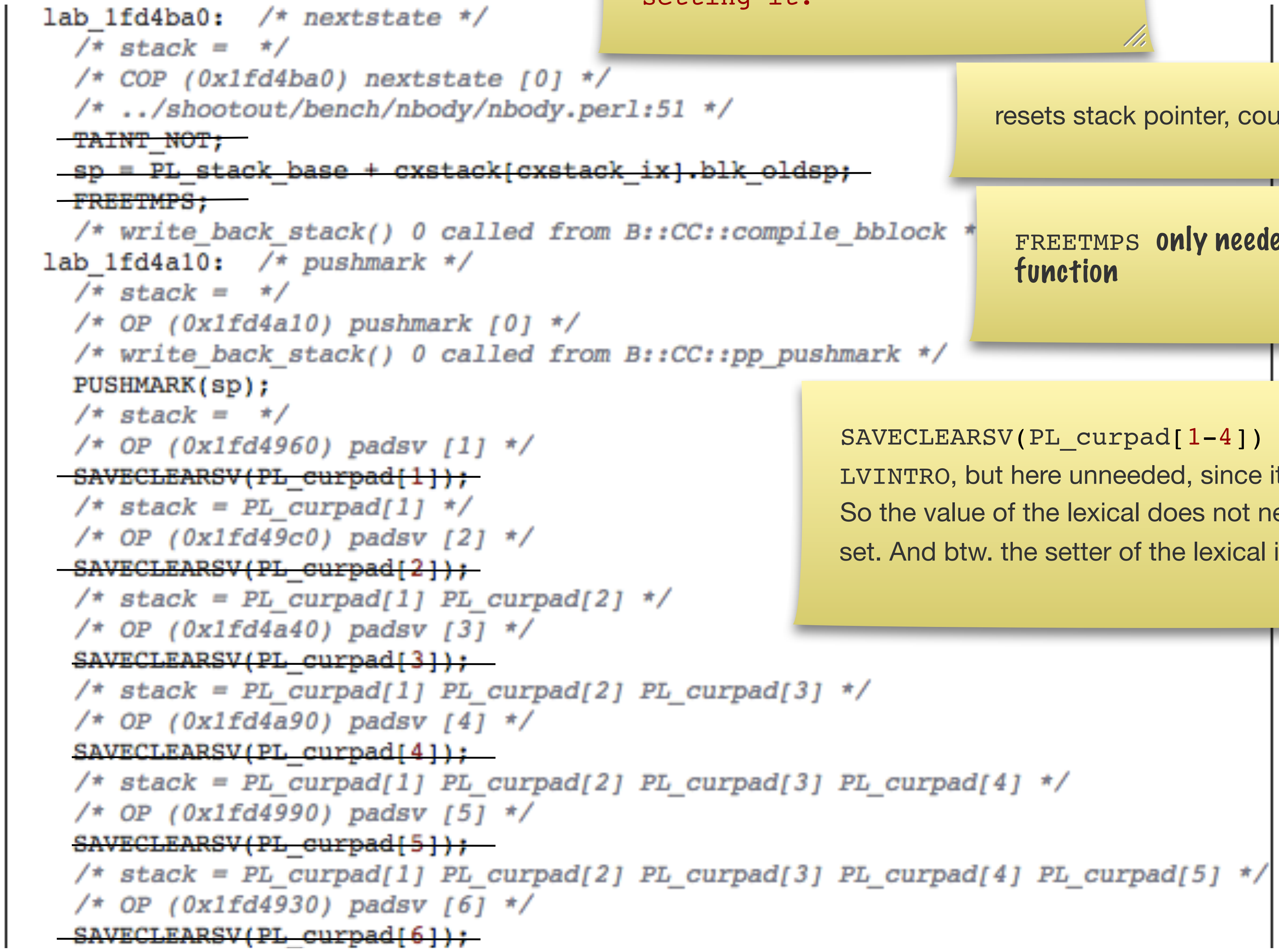
```
PUSHs(AvARRAY(PL_curpad[6])[0]));
PUSHs(AvARRAY(PL_curpad[6])[1]));
rnv0 = POPn; lnv0 = POPn;       /* multiply */
d30_tmp = rnv0 * lnv0;
```

B::Stackobj::Aelem *(ongoing work)*



```
rnv0 = SvNV(AvARRAY(PL_curpad[6])[0]);
lnv0 = SvNV(AvARRAY(PL_curpad[6])[1]);
d30_tmp = rnv0 * lnv0;          /* multiply */
```

Minor optimizations (<10%)



```
lab_1fd4ba0:  /* nextstate */
  /* stack =  */
  /* COP (0x1fd4ba0) nextstate [0] */
  /* ../shootout/bench/nbody/nbody.perl:51 */
~~TAINT_NOT;~~
~~sp = PL_stack_base + cxstack[cxstack_ix].blk_oldsp;~~
~~FREETMPS;~~
  /* write_back_stack() 0 called from B::CC::compile_bblock *
lab_1fd4a10:  /* pushmark */
  /* stack =  */
  /* OP (0x1fd4a10) pushmark [0] */
  /* write_back_stack() 0 called from B::CC::pp_pushmark */
  PUSHMARK(sp);
  /* stack =  */
  /* OP (0x1fd4960) padsv [1] */
~~SAVECLEARSV(PL_curpad[1]);~~
  /* stack = PL_curpad[1] */
  /* OP (0x1fd49c0) padsv [2] */
~~SAVECLEARSV(PL_curpad[2]);~~
  /* stack = PL_curpad[1] PL_curpad[2] */
  /* OP (0x1fd4a40) padsv [3] */
~~SAVECLEARSV(PL_curpad[3]);~~
  /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] */
  /* OP (0x1fd4a90) padsv [4] */
~~SAVECLEARSV(PL_curpad[4]);~~
  /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] PL_curpad[4] */
  /* OP (0x1fd4990) padsv [5] */
~~SAVECLEARSV(PL_curpad[5]);~~
  /* stack = PL_curpad[1] PL_curpad[2] PL_curpad[3] PL_curpad[4] PL_curpad[5] */
  /* OP (0x1fd4930) padsv [6] */
~~SAVECLEARSV(PL_curpad[6]);~~
```

`PL_tainted = 0`, nobody is setting it.

resets stack pointer, could be better handled by ourselves

`FREETMPS` only needed if locals are used in the function

`SAVECLEARSV(PL_curpad[1-4])` is part of `padsv /` `LVINTRO`, but here unneeded, since it is in the context of sassign. So the value of the lexical does not need to be cleared before it is set. And btw. the setter of the lexical is already optimized to a

- http://blogs.perl.org/users/rurban/2012/09/optimizing-compiler-benchmarks-part-1.html - part-4
- B::CC

Overview

• perl	23m
• perl -funroll-loops	14m13s
• perlcc -O	9m52s
• perlcc -O -funroll-loops	7m11s
• perlcc -O -funroll-loops -fno-autovivify	4m52s
• perlcc -O -O1 (-fno-magic)	3m30s
• + *new aelem stackopt + -fno-cop*	*~2m36s*