

CROWDSOURCING CODE TUNING WITH PEACHPY.IO

Marat Dukhan, Anirudh Gubba, Richard Vuduc
mdukhan3@gatech.edu, agubba3@gatech.edu, richie@cc.gatech.edu



INTRODUCTION

High-performance scientific libraries include a large portion of optimized assembly code. Maintaining processor-specific optimized assembly and keeping it up-to-date requires rare low-level optimization skills. In practice, a very small pool of potential contributors is further reduced by the lack of access to target hardware, and complicated software configurations required for scientific projects. We develop infrastructure to increase productivity of performance tuning experts:

- PeachPy, a Python-based DSL for assembly metaprogramming. PeachPy automates boring and error-prone parts of assembly programming, such as register allocation and adaption of code to different ABIs.
- WebPeachPy, a Python interpreter with PeachPy that runs inside a Web browser.
- WebRunner, a REST server that safely loads and executes compiled assembly kernels.
- PeachPy.io, an in-browser IDE for assembly code tuning and analysis.

PEACHPY EXAMPLE: SGEMM

```
ymm_c = [[YMMRegister() for n in range(nr)] for m in range(mr // 8)]
ymm_a = [YMMRegister() for m in range(mr // 8)]
ymm_b_n = YMMRegister()

VZEROALL()

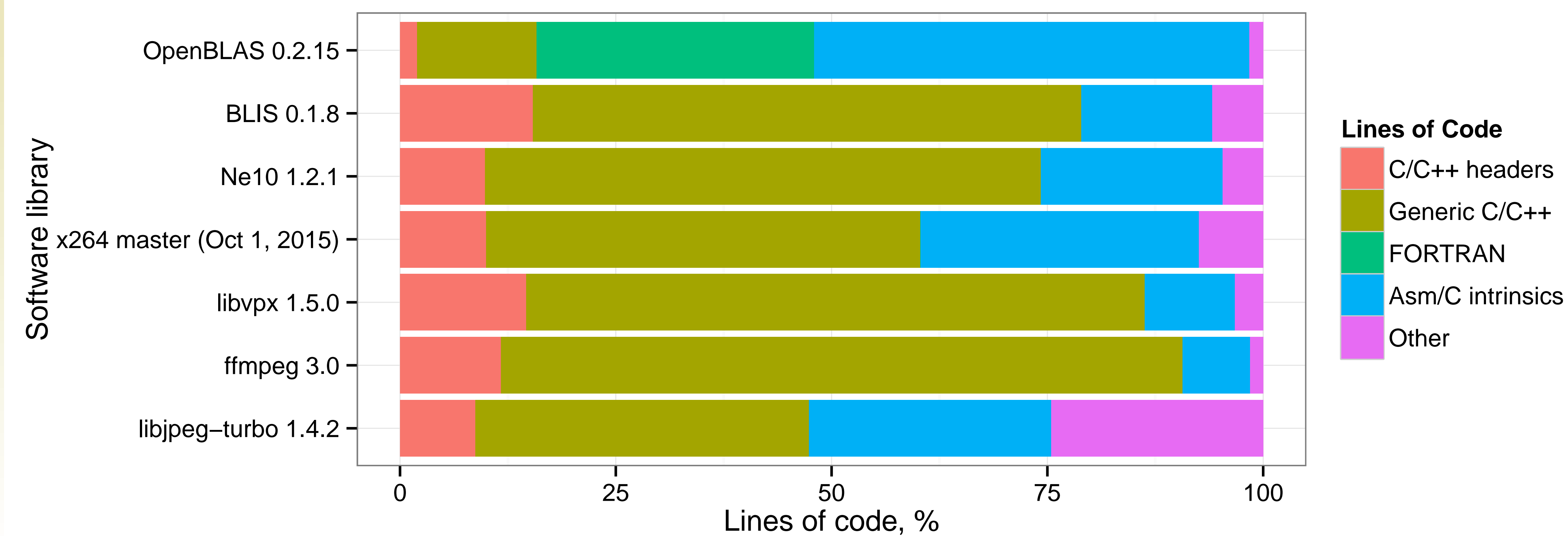
with Loop() as loop:
    for m in range(mr // 8):
        VMOVAPS(ymm_a[m], [reg_b + m * YMMRegister.size])

    for n in range(nr):
        VBROADCASTSS(ymm_b_n, [reg_b + n * float_.size])
        for m in range(mr // 8):
            VFMADD231PS(ymm_c[m][n], ymm_b_n, ymm_a[m])

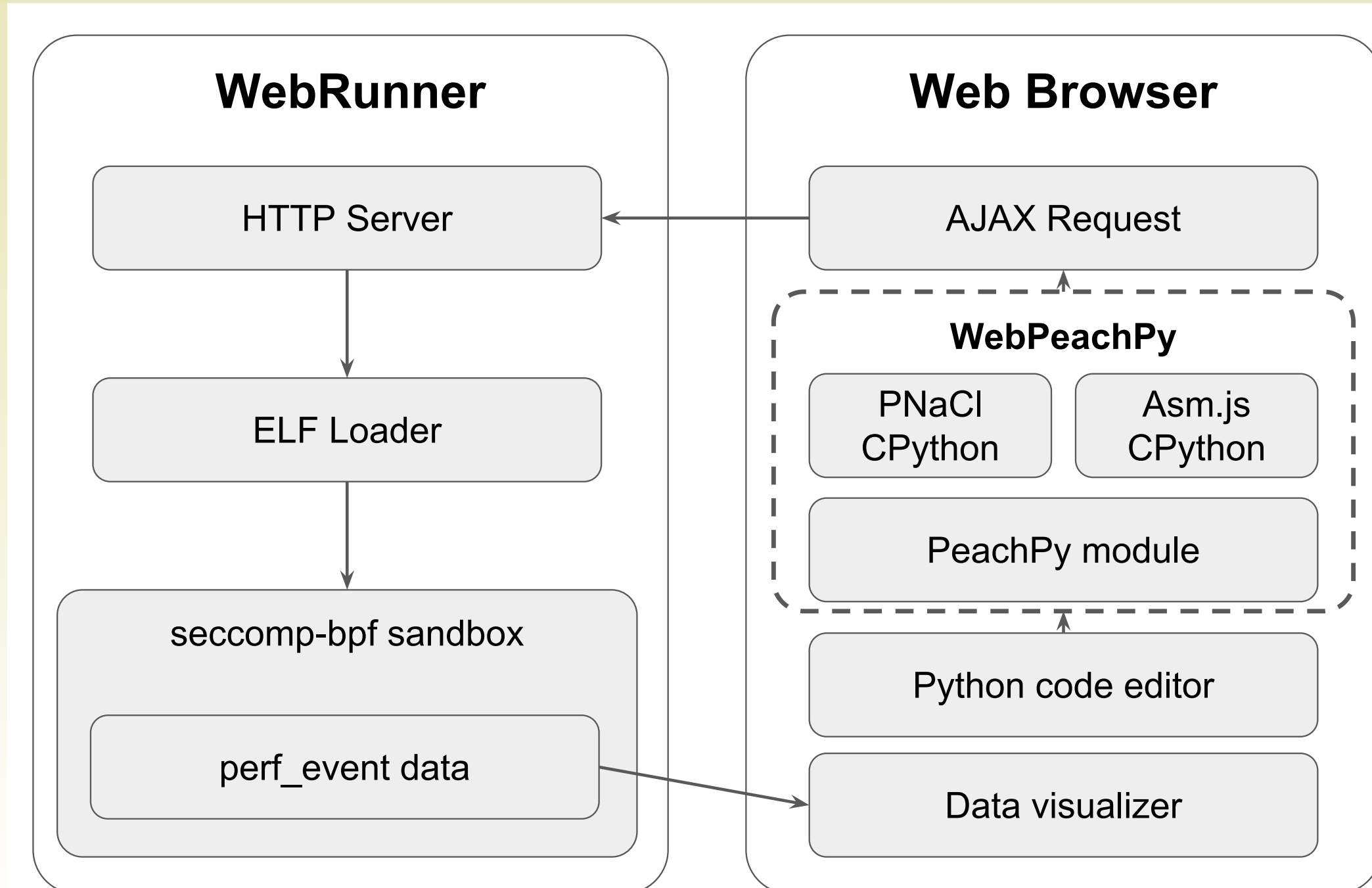
    ADD(reg_a, mr * float_.size)
    ADD(reg_b, nr * float_.size)

DEC(reg_k)
JNE(loop.begin)
```

OPTIMIZED LIBRARIES: CODE COMPOSITION



PEACHPY.IO INFRASTRUCTURE



PEACHPY EXAMPLE: x264 MACRO

```
def abs2(dst1, dst2, src1, src2): ; dst1, dst2, src1, src2, tmp, tmp
    if function target_has_sse3: ; if cpuflag(sse3)
        PABSW(dst1, src1) ; pabsw %1, %3
        PABSW(dst2, src2) ; pabsw %2, %4
    elif dst1 == src1: ; elif idn %1, %3
        tmp1, tmp2 = \
            MMXRegister(), MMXRegister() ; pxor %5, %5
        PXOR(tmp1, tmp1) ; pxor %6, %6
        PXOR(tmp2, tmp2) ; psubw %5, %1
        PSUBW(tmp1, dst1) ; psubw %6, %2
        PSUBW(tmp2, dst2) ; pmaxsw %1, %5
        PMAXSW(dst1, tmp1) ; pmaxsw %2, %6
    else: ; else
        PXOR(dst1, dst1) ; pxor %1, %1
        PXOR(dst2, dst2) ; pxor %2, %2
        PSUBW(dst1, src1) ; psubw %1, %3
        PSUBW(dst2, src2) ; psubw %2, %4
        PMAXSW(dst1, src1) ; pmaxsw %1, %3
        PMAXSW(dst2, src2) ; pmaxsw %2, %4
    ; endif
; endmacro
```

PEACHPY.IO CODE EDITOR

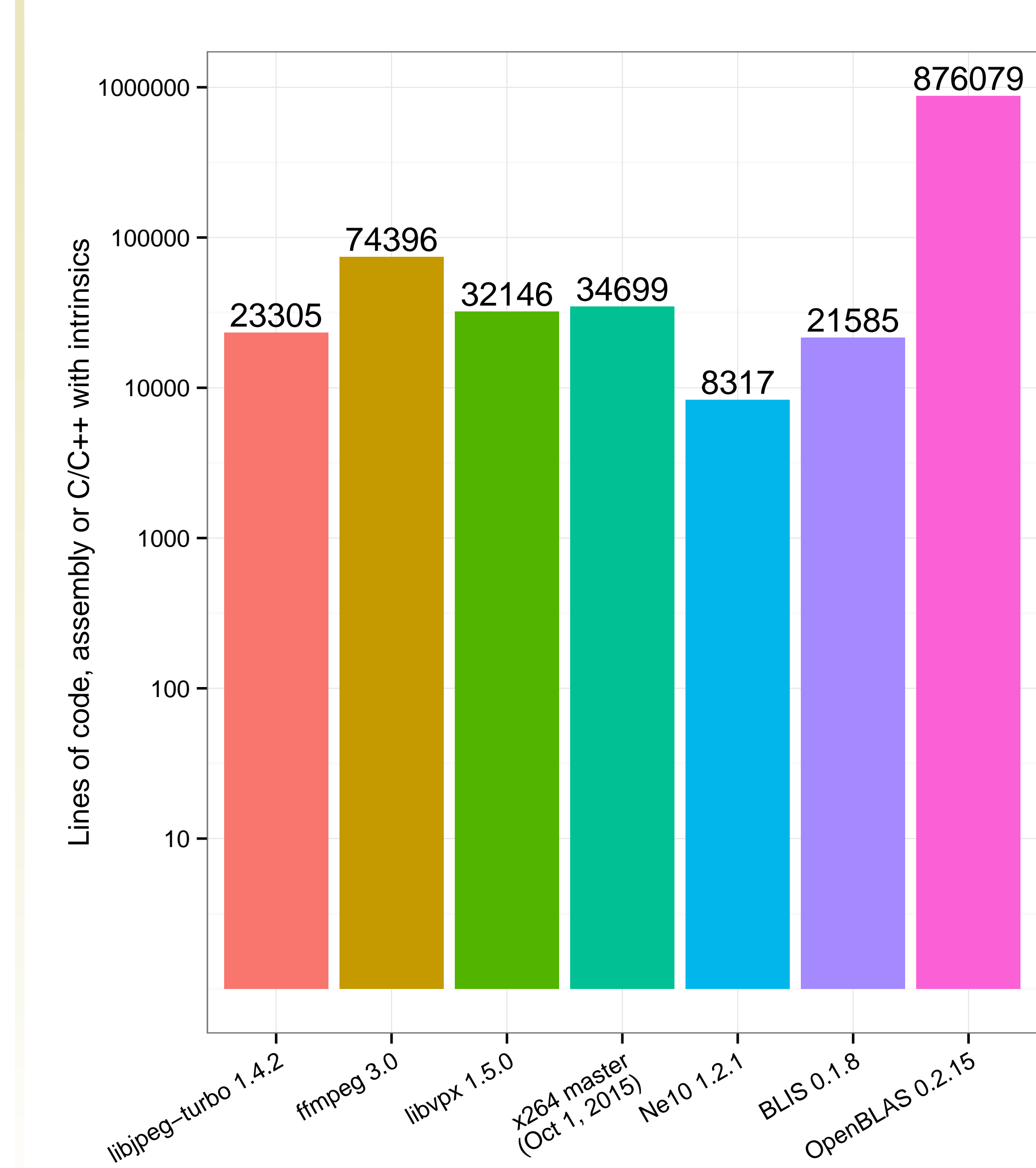
```
Target: Intel Skylake
Quick Run Show Log About PeachPy

39
40 with BlockO as unit_stride:
41     unroll_registers = 5
42     unroll_elements = unroll_registers * XMMRegister.size / float_.size
43
44     vector_loop = Loop()
45     scalar_loop = Loop()
46
47     xmm_accs = [XMMRegister() for _ in range(unroll_registers - 1)]
48     for xmm_acc in xmm_accs[1:]:
49         XORPS(xmm_acc, xmm_acc)
50
51     SUB(reg_n, unroll_elements)
52     JB(vector_loop.end)
53
54     with vector_loop:
55         xmm_xs = [XMMRegister() for _ in range(unroll_registers)]
56         xmm_ys = [XMMRegister() for _ in range(unroll_registers)]
57
58         for i, (xmm_x, xmm_y) in enumerate(zip(xmm_xs, xmm_ys)):
59             MOVUPS(xmm_x, [reg_x + XMMRegister.size * i])
60             MOVUPS(xmm_y, [reg_y + XMMRegister.size * i])
61
62             ADD(reg_x, unroll_registers * XMMRegister.size)
63             ADD(reg_y, unroll_registers * XMMRegister.size)
64
65             for xmm_x, xmm_y, xmm_acc in zip(xmm_xs, xmm_ys, xmm_accs):
66                 MULPS(xmm_x, xmm_y)
67                 ADDPS(xmm_acc, xmm_x)
68
69             SUB(reg_n, unroll_elements)
70             JAE(vector_loop.begin)
71
72     # Reduction of multiple XMM registers into into a single XMM register
73     REDUCE(ADDPS, xmm_accs)
74
75     ADD(reg_n, unroll_elements)
76     JZ(scalar_loop.end)
77
78     with scalar_loop:
79         ...
80
81 Intel Skylake: 394 cycles (3.31 IPC)
```

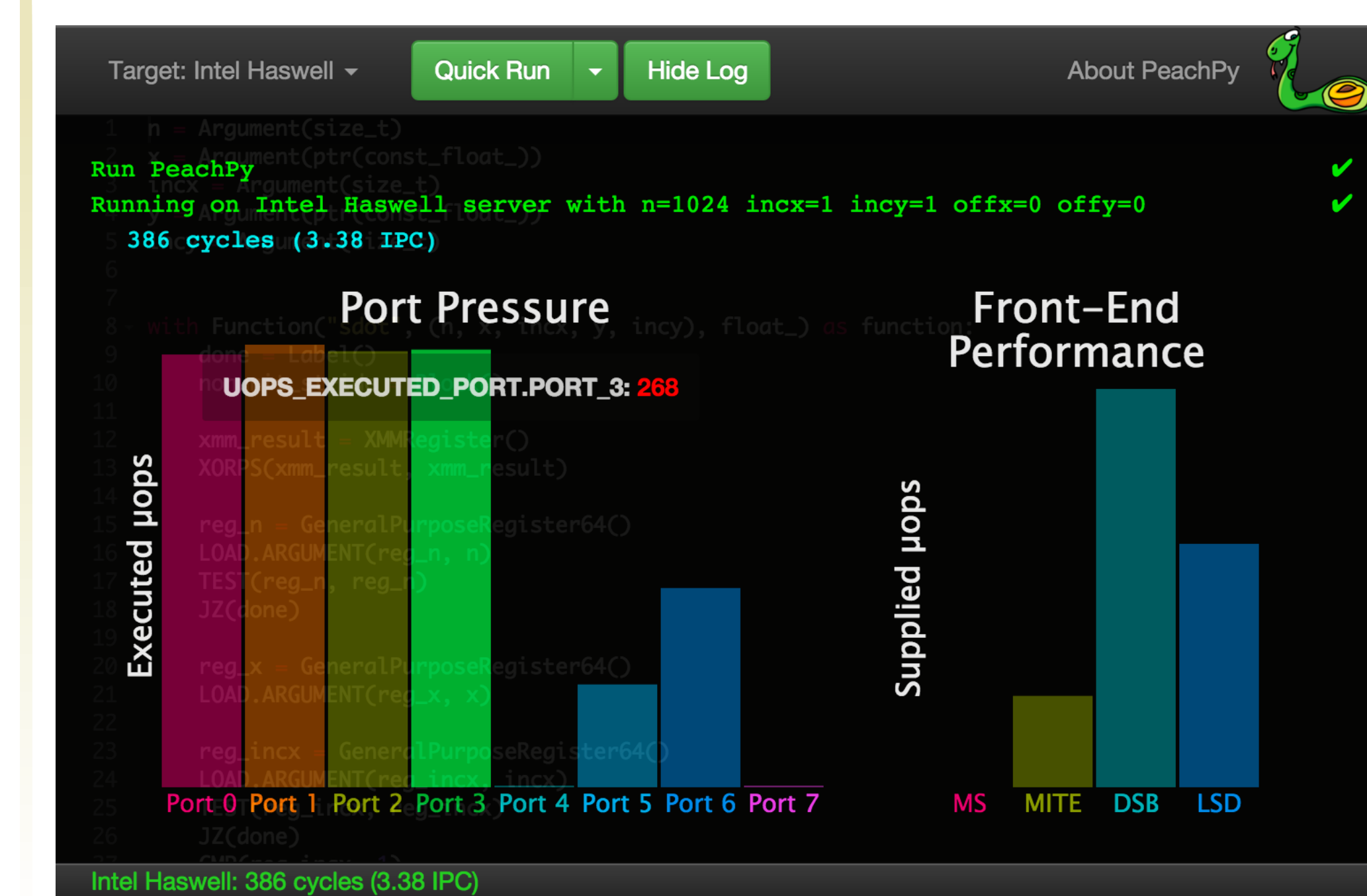
WEBRUNNER SPECIFICATION

```
<?xml version='1.0' encoding='utf-8'?>
<kernel name="sdot" namespace="blis">
    <query>
        <parameter name="n" type="uint64" default="1024" />
        <parameter name="incx" type="uint32" default="1" min="1" />
        <parameter name="incy" type="uint32" default="1" min="1" />
        <parameter name="offx" type="uint32" default="0" max="63" />
        <parameter name="offy" type="uint32" default="0" max="63" />
    </query>
    <call>
        <argument name="n" type="size_t" />
        <argument name="x" type="const float*" />
        <argument name="incx" type="size_t" />
        <argument name="y" type="const float*" />
        <argument name="incy" type="size_t" />
    </call>
</kernel>
```

TARGET-SPECIFIC CODES



PEACHPY.IO CODE ANALYSIS



ACKNOWLEDGEMENTS

This material is based upon work supported by the U.S. National Science Foundation (NSF) Award Number 1339745. We appreciate a hardware donation from AMD to support PeachPy.io Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF or AMD.