

# Stack operation optimization in Pen

November 13, 2022

[@raviqqe](#)

# Continuation Passing Style (CPS)

## Direct style

```
f = \ (x) {  
  y = g()  
  
  x + y  
}
```

## CPS

```
f = \ (k, x) {  
  g(\ [k, x] (y) {  
    k(x + y)  
  })  
}
```

# CPS for consecutive function calls

## Direct style

```
f = \ (x) {  
  y = f()  
  z = g()  
  
  x + y + z  
}
```

## CPS

```
f = \ (k, x) {  
  f(\ [k, x](y) {  
    f(\ [k, x, y](z) {  
      k(x + y + z)  
    })  
  })  
}
```

# CPS in Pen

- Pen uses CPS to make all functions suspendable.
  - i.e. every function is an *async* function.
  - Functions are suspended for I/O, synchronization, etc.
- Pen doesn't support the first-class continuations.

# CPS stack

- Pen uses two stacks at runtime.
- Machine stack
  - Nearly used other than reference count operations and foreign functions.
- Heap-allocated stack
  - Used to allocate "continuations" in CPS
  - No need for heap allocation of each continuation

# CPS transformation

1. Calculate environments of continuations.
2. Compile a direct-style IR to ANF.
  - Using the second-class continuations
3. Compile Pen-native function calls into CPS.

# Function entrypoints

```
fn function_entrypoint_2<A1, A2, T, F: Fn(A1, A2) -> T>(
    stack: &mut Stack,
    continuation: fn(stack: &mut Stack, result),
    closure: Arc<Closure<F>>,
    argument0: A0,
    argument1: A1,
) {
    // ...
}
```

- Continuations are raw function pointers of their entrypoints.
- Where are their environments?
  - In heap-allocated stacks!



# Function calls

1. Create a continuation or pass it down from a caller.
2. **Push environment of a continuation to a stack if necessary.**
3. Call a function entrypoint with the continuation entrypoint.

## Continuation entrypoint

1. **Pop environment of a continuation from a stack if necessary.**
2. Execute instructions.

What if those continuations' environments are the same or similar?

# Examples

- In a continuation,
  - We pop free variables of `a`, `b`, `c`.
- In its continuation,
  - We push free variables of `b`, `c`.

In this case, we don't need to push the environment at all if stack elements are properly ordered.

In general, if stack elements of continuations in a function are ordered properly, we can calculate diff of those and generate codes only to fill the diff.

# Stack operation optimization

- Stack elements are ordered by an ascending order of  $2^{\text{frequency}}$ 
  - **frequency** is a frequency at which free variables appear in continuations throughout a top-level function.
- When we push environments of continuations, we rather use partial push if applicable.
  - i. Pop **all** unused free variables.
  - ii. Push new free variables.

# Result

- 5% size reduce in module object files
- CPU time performance improvement was pretty minimal (~1%.)

# Questions

- Reinvention of the wheel?
  - Register coloring and active frame calculation?
  - `async` generator state machines in Rust
  - In CPS, we can extend and shrink active frames.
- Do we need minimum memory usage?
  - Tail call optimization + CPS = "stack GC"
  - Just use the maximum environment size throughout a function?
  - Is it more CPU-time friendly?

# Summary

Stack operations are now fully optimized!

# Appendix

# Stack operations in CPS

In `Add`,

```
504: 42 00 00 91      add     x2, x2, #0
508: fd 7b 45 a9      ldp    x29, x30, [sp, #80]
50c: 29 01 08 8b      add    x9, x9, x8
510: 08 81 00 91      add    x8, x8, #32      <-
514: f6 57 43 a9      ldp    x22, x21, [sp, #48]
518: 28 29 01 6d      stp    d8, d10, [x9, #16] <-
51c: 29 05 00 fd      str    d9, [x9, #8]     <-
520: 34 01 00 f9      str    x20, [x9]        <-
524: 68 06 00 f9      str    x8, [x19, #8]    <-
528: f4 4f 44 a9      ldp    x20, x19, [sp, #64]
```



At the beginning of `Add`'s first continuation,

```
2a70: ff 03 01 d1      sub     sp, sp, #64
2a74: 08 04 40 f9      ldr     x8, [x0, #8]          <-
2a78: 09 00 00 90      adrp   x9, 0x2000 <__k_1a+0x8>
2a7c: e9 23 01 6d      stp    d9, d8, [sp, #16]
2a80: 08 40 60 1e      fmov   d8, d0
2a84: f4 4f 02 a9      stp    x20, x19, [sp, #32]
2a88: f3 03 00 aa      mov    x19, x0
2a8c: 08 81 00 d1      sub    x8, x8, #32           <-
2a90: fd 7b 03 a9      stp    x29, x30, [sp, #48]
2a94: 08 04 00 f9      str    x8, [x0, #8]         <-
2a98: 23 01 40 f9      ldr    x3, [x9]
```

At the end of `Add`'s first continuation,

```
2ae4: 42 00 00 91      add    x2, x2, #0
2ae8: fd 7b 43 a9      ldp    x29, x30, [sp, #48]
2aec: 08 81 00 91      add    x8, x8, #32      <-
2af0: 68 06 00 f9      str    x8, [x19, #8]    <-
2af4: 28 69 28 fc      str    d8, [x9, x8]     <-
2af8: 68 06 40 f9      ldr    x8, [x19, #8]    <-
2afc: e9 23 41 6d      ldp    d9, d8, [sp, #16]
2b00: 08 21 00 91      add    x8, x8, #8      <-
2b04: 68 06 00 f9      str    x8, [x19, #8]    <-
2b08: f4 4f 42 a9      ldp    x20, x19, [sp, #32]
```

```

    add    x10, x10, #8
    str    x8, [x0, #8]
Lloh21:
    ldr    x21, [x9, "_Foo.pen:f"@PAGEOFF+8]
    ldr    x9, [x0, #16]
    ldr    x0, [x0]
    cmp    x10, x9
    b.ls   LBB6_2
; %bb.1:
    lsl    x20, x9, #1
    mov    x1, x20
    bl     __pen_realloc
    ldr    x8, [x19, #8]
    str    x0, [x19]
    str    x20, [x19, #16]
LBB6_2:
    add    x9, x8, #32
    add    x8, x0, x8
Lloh22:
    adrp   x1, __k_15@PAGE
Lloh23:
    add    x1, x1, __k_15@PAGEOFF
Lloh24:
    adrp   x2, "_Foo.pen:f"@PAGE+8
Lloh25:
    add    x2, x2, "_Foo.pen:f"@PAGEOFF+8
    str    x9, [x19, #8]
    mov    x0, x19
    str    d8, [x8, #32]
    ldr    x8, [x19, #8]
    add    x8, x8, #8
    str    x8, [x19, #8]
    blr    x21
    ldp    x29, x30, [sp, #48]
    ldp    x20, x19, [sp, #32]
    ldp    x22, x21, [sp, #16]
    ldp    d9, d8, [sp], #64
    ret
.loh AdrpLdr    Lloh20, Lloh21
.loh AdrpAdd    Lloh24, Lloh25
.loh AdrpAdd    Lloh22, Lloh23
; -- End function

```

## After:

```

    .p2align    2
    __k_1a:
; %bb.0:
    stp    d9, d8, [sp, #-64]!
    stp    x22, x21, [sp, #16]
    ldr    x22, [x0, #8]
Lloh20:
    adrp   x8, "_Foo.pen:f"@PAGE+8
    stp    x20, x19, [sp, #32]
    stp    x29, x30, [sp, #48]
    fmov   d8, d0
    sub    x9, x22, #32
    mov    x19, x0
    str    x9, [x0, #8]
    add    x9, x22, #8
Lloh21:
    ldr    x21, [x8, "_Foo.pen:f"@PAGEOFF+8]
    ldr    x8, [x0, #16]
    ldr    x0, [x0]
    cmp    x9, x8
    blr    x21

```