RECURSION INLINING IN LLVM

Pablo Barrio, ARM
Chandler Carruth, Google
James Molloy, ARM

Why?

- Performance improvements observed of up to ~20%.
- Potentially improve chances for other optimization passes:
  - Increase function size.
  - Increase knowledge of the context.
- Current status:
  - GCC inlines recursive function calls up to a certain depth.
  - LLVM marks recursive functions as *noinline.*
- Code size and register pressure must be controlled.

#1 Inline iteratively

```
f(x):
  if x == 0
    return 1
  a = g(x)
  b = f(a)
  return a + h(b)
```

```
f(x):
  if x == 0
    return 1
  a = g(x)

  if a == 0
    b = 1
  else:
    a1 = g(a)
    b1 = f(a1)
    b = a1 + h(b1)

  return a + h(b)
```

Level 1 inline

#2 Remove recursion with a stack

```
f(x):
  if x == 0
    return 1
  a = g(x)
  b = f(a)
  return a + h(b)
```

```
f(x):
  x1 = x

  while x1 != 0:
    a = g(x)
    x1 = a
    push(a)

  b = 1

  while S not empty:
    a = pop()
    b = a + h(b)

  return b
```

a is live

Fibonacci

Time relative to GCC base

	GCC	Clang
Base (-O3)	1.00	1.97
Stack	2.53	1.19
Iterative inlining	1.00	1.04

Object size (kB)

	GCC	Clang
Base (-O3)	3.50	2.54
Stack	2.67	2.71
Iterative inlining	3.50	2.74