Inlining for Size

Kyungwoo Lee
Ellis Hoag
Nathan Lanza
Mobile Apps Optimized for Size

- Mobile apps are mainly optimized for size (-Oz).
  - Demand more features on constrained devices.
  - Size and performance can be tuned with PGO [1].
- A (Full) link-time optimization (LTO) can minimize app size.
  - Practically, ThinLTO [2], a scalable LTO, is used for large apps.
- Inlining has been primarily considered a speed optimization.
  - Inlining is also critical for app size.
  - However, inlining for size with ThinLTO is suboptimal.

02 MOTIVATION

NoLTO - No inlining occurs across modules

M1

```c
int main(B *b) {
    return [b foo:10];
}
```

M2

```c
- (int) foo:(int) i
    __attribute__((objc_direct)) {
        // Direct (small) callee
        return [C goo:i];
    }
```

M3

```c
+ (int) goo:(int) j {
    // Dynamic callee
    return hoo(j) * zoo(j);
}
```

M4

```c
int hoo(int i) {
    // Large callee
}
```

M5

```c
int zoo(int i) {
    // Small callee
}
```
NoLTO - No inlining occurs across modules

(M1) int main(B *b) {
  return [b foo:10];
}

(M2) - (int) foo:(int) i
    __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
    }

(M3) + (int) goo:(int) j {
    // Dynamic callee
    return hoo(j) * zoo(j);
    }

(Full)LTO - All (direct) callees are inlined

After Inlining

M1+M2+M3+M4+M5

main
foo

...............goo
hoo
zoo
The baseline inliner aggressively inlines a local function called from a single call-site.

- Or, only tiny functions (<= OptMinSizeThreshold(5)) become inline candidates with -Oz.

Our size-inliner uses the simple cost model, to find a candidate where $C_{\text{before}} > C_{\text{after}}$

\[ C_{\text{before}} = C_{\text{callee}} + N \times C_{\text{call}} \]
\[ C_{\text{after}} = N \times C_{\text{callee}} \]

where $N$ is # of call sites, $C_{\text{callee}}$ is the callee size, and $C_{\text{call}}$ is the call overhead.
02 MOTIVATION

(Full)LTO - Limit Study

- The baseline inliner (-Oz) is close to our size inliner with # of Call Site, N = 1, which is the majority of size-win.
  - Our size-inliner can improve the size win further by 0.4% w/ up to # of Call Site, N = 3.
ThinLTO - Only small (eligible) callee inlined

Before Inlining

M1
int main(B *b) {
    return [b foo:10];
}

M2
- (int) foo:(int) i
  __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
  }

M3
+ (int) goo:(int) j {
  // Dynamic callee
  return hoo(j) * zoo(j);
}

M4
int hoo(int i) {
  // Large callee
}

M5
int zoo(int i) {
  // Small callee
}
ThinLTO - Callee is not eligible to import.

02 MOTIVATION

Before Inlining

M1

int main(B *b) {
    return [b foo:10];
}

foo is not eligible to import due to private references to metadata C.

M2

- (int) foo:(int) i
  __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
  }

M3

+ (int) goo:(int) j {
  // Dynamic callee
  return hoo(j) * zoo(j);
}

hoo is large and ThinLTO cannot determine its unique call-site.

M4

int hoo(int i) {
  // Large callee
}

M5

int zoo(int i) {
  // Small callee
}
ThinLTO - Only small (eligible) callee inlined

Before Inlining

```c
int main(B *b) {
    return [b foo:10];
}
```

- (int) foo:(int) i
  __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
  }

```
int hoo(int i) {
    // Large callee
    return hoo(j) * zoo(j);
}
```  

After Inlining

- `foo` is not eligible to import due to private references to metadata C.
Our Contribution

Thin-Link

IR Optimizer
& Codegen
Our Contribution

- Size-Inlining Analyzer + Pre-Inliner
  - Extend bitcode summary that reflects call-site counts (within each module)
  - Propagate summaries, and determine inline candidates
  - Force to import & inline those candidates to realize the size win ahead.
Our Contribution

- **Size-Inlining Analyzer + Pre-Inliner**
  - Extend bitcode summary that reflects call-site counts (within each module)
  - Propagate summaries, and determine inline candidates
  - Force to import & inline those candidates to realize the size win ahead.

- **Size-Inlining Analyzer + Pre-Merger**
  - Find the inline candidates *that are not eligible to import*.
  - Merge their parent bitcode modules to remove inline restrictions.
ThinLTO /w Size Inliner

Before Inlining

```
int main(B *b) {
    return [b foo:10];
}

-(int) foo:(int) i
    __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
}

int hoo(int i) {
    // Large callee
}

int zoo(int i) {
    // Small callee
}

+(int) goo:(int) j {
    // Dynamic callee
    return hoo(j) * zoo(j);
}
```
ThinLTO /w Size Inliner

Before Inlining

M1

```c
int main(B *b) {
    return [b foo:10];
}
```

M2

```c
- (int) foo:(int) i
    __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
    }
```

M3

```c
+ (int) goo:(int) j {
    // Dynamic callee
    return hoo(j) * zoo(j);
    }
```

M4

```c
int hoo(int i) {
    // Large callee
    }
```

M5

```c
int zoo(int i) {
    // Small callee
    }
```

After Inlining

M1+M2

```c
main foo
```
ThinLTO /w Size Inliner

Before Inlining

M1

int main(B *b) {
    return [b foo:10];
}

M2

- (int) foo:(int) i
  __attribute__((objc_direct)) {
    // Direct (small) callee
    return [C goo:i];
  }

M3

+ (int) goo:(int) j {
    // Dynamic callee
    return hoo(j) * zoo(j);
}

M4

int hoo(int i) {
    // Large callee
}

M5

int zoo(int i) {
    // Small callee
}

After Inlining

M1 + M2

main foo

goo hoo zoo
ThinLTO /w Size Inliner

Before Inlining

M1
int main(B *b) {
    return [b foo:10];
}

M2
- (int) foo:(int) i
    __attribute__((objc_direct)) {
        // Direct (small) callee
        return [C goo:i];
    }

M3
+ (int) goo:(int) j {
    // Dynamic callee
    return hoo(j) * zoo(j);
}

M4
int hoo(int i) {
    // Large callee
}

M5
int zoo(int i) {
    // Small callee
}

After Inlining

M1+M2
main

M3
goo
hoo
zoo

M4
hoo

M5
zoo
Code Size Impact

- SocialApp
  - A large app written in Objective-C/Swift
- ChatApp
  - A medium size app written in Objective-C/C++
- Clang
  - A compiler benchmark
  - Pre-merger has no impact
Conclusion

- The size inliner for (Full)LTO can still improve the size by 0.4% for Clang.
- The size inliner for ThinLTO improved [1]:
  - The code size, 2.8% for SocialApp, and 4.0% for ChatApp.
  - Clang became 3% smaller and 6.1% faster.

https://dl.acm.org/doi/10.1145/3519941.3535074