Practical Global Merge Function with ThinLTO

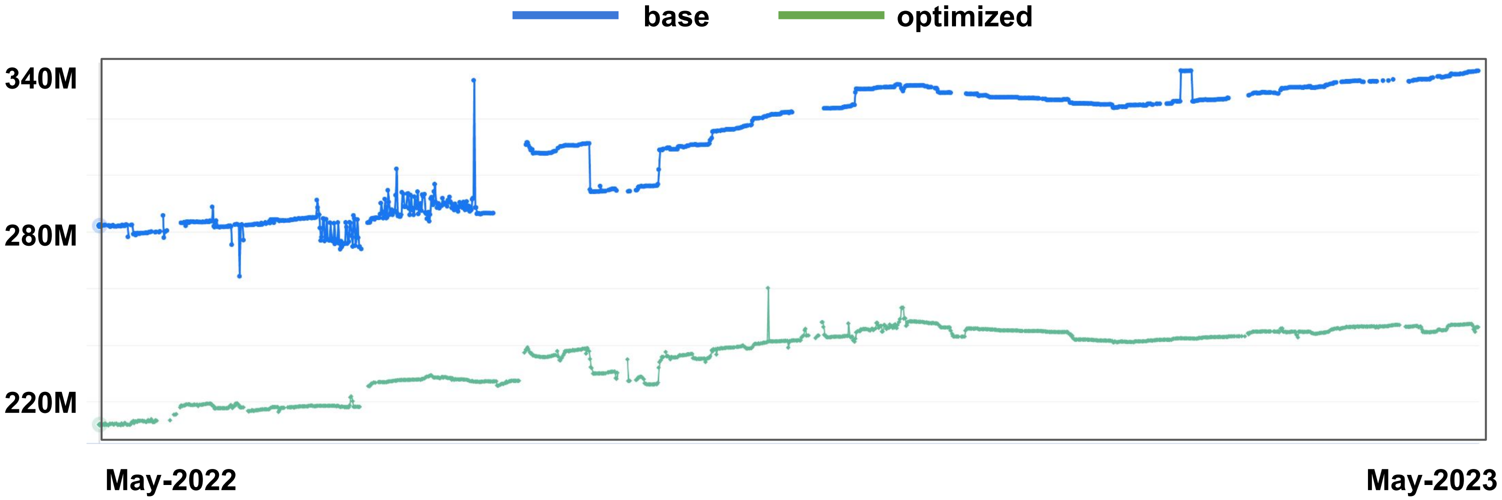
Kyungwoo Lee, Manman Ren, Sharon Xu, Ellis Hoag

Meta

App Size Continues to Grow

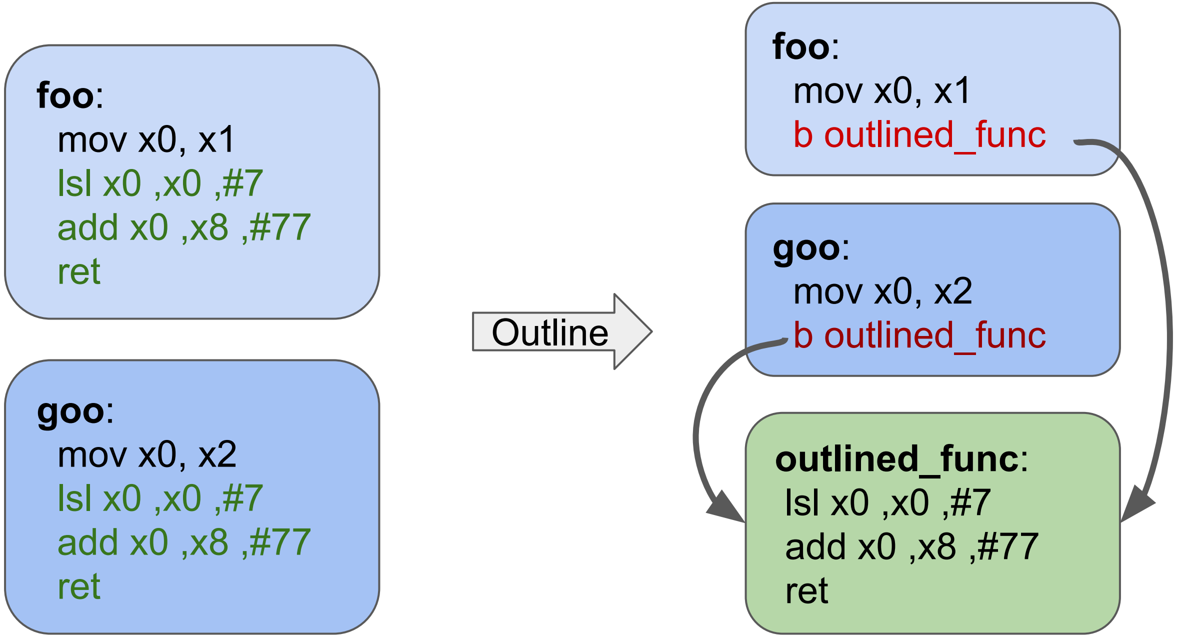
- Large and slow apps impact user experience and user retention
- Code size optimizations (e.g., outlining or merging) are critical!

Uncompressed App Size Growth



Function Outliner

- Outline the identical code sequence into a call
- IR Outliner (Opt) vs. **Machine Outliner (CodeGen)**



```
foo:
  mov x0, x1
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

```
goo:
  mov x0, x2
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

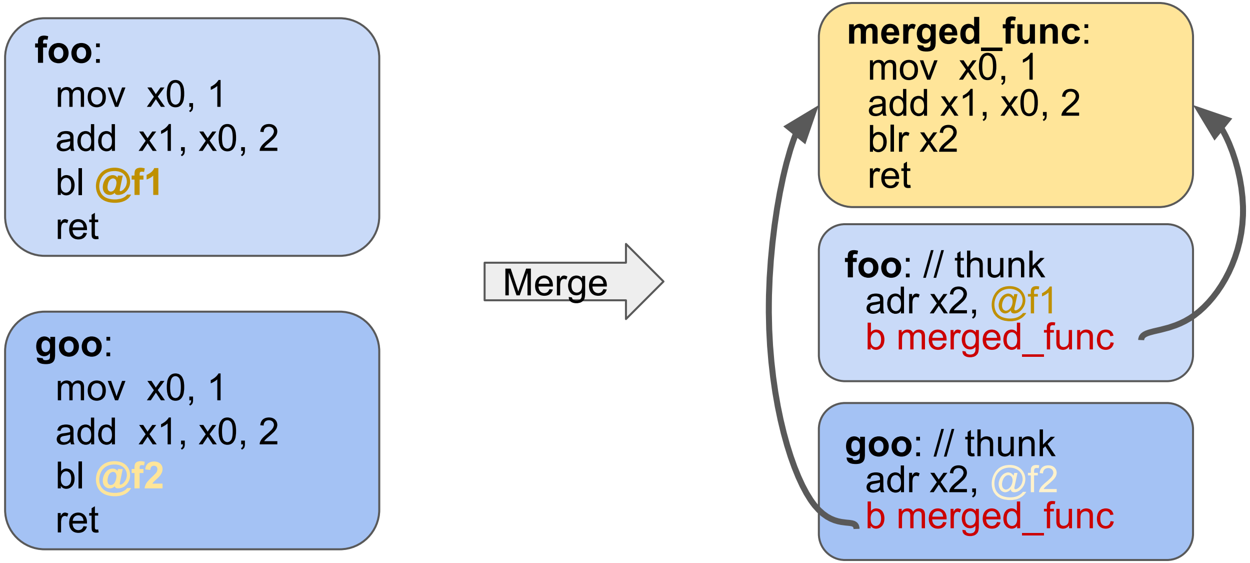
```
foo:
  mov x0, x1
  b outlined_func
```

```
goo:
  mov x0, x2
  b outlined_func
```

```
outlined_func:
 lsl x0 ,x0 ,#7
 add x0 ,x8 ,#77
 ret
```

Function Merger

- Merge identical functions, effectively similar to the linker’s ICF
- Can merge *similar* functions by parameterizing Constant [1]



```
foo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f1
  ret
```

```
goo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f2
  ret
```

```
merged_func:
  mov  x0, 1
  add x1, x0, 2
  blr x2
  ret
```

```
foo: // thunk
  adr x2, @f1
  b merged_func
```

```
goo: // thunk
  adr x2, @f2
  b merged_func
```

[1] Merge similar functions for swift, https://github.com/apple/swift/blob/main/lib/LLVMPasses/LLVMMergeFunctions.cpp

Function Merger vs. (Machine) Function Outliner in LLVM

	Function Merger	**Function Outliner**
Pass	Opt (IR)	Codegen (Machine IR)
Scope	Entire Function	Block or code sequence
Match	Identical or Similar	Identical
Call/Frame overhead	High	Low
Code size impact	Low - Medium	High
Debug/Metadata concern	High	Low
ThinLTO applicability	?	Yes [2]

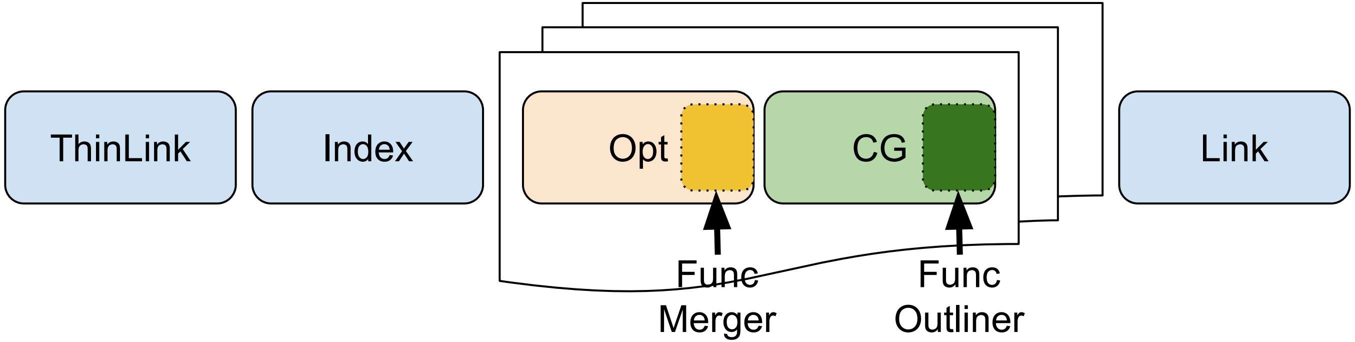
[2] Efficient Profile-Guided Size Optimization for Native Mobile Applications, CC2022. https://doi.org/10.1145/3497776.3517764

Our Design of Global Function Merger

- Effective in addition to (machine) function outliner and linker’s ICF
  - Function merger targets *similar* functions
  - Function outliner handles *dissimilar* functions while outlining identical blocks
  - *Identical* functions can be folded either from function merger or linker’s ICF
- Scalable with ThinLTO
  - Use a summary, *StableFunction* to track the *similarity* of functions
  - Create a *unique* merge instance within each module
  - Emit thunks without changing call-sites, to prevent invalidating summaries
- Practical for the production use
  - Sound in the presence of IR and summary mismatches
  - Maintain the integrity of merged function as possible to retain debug info or metadata.

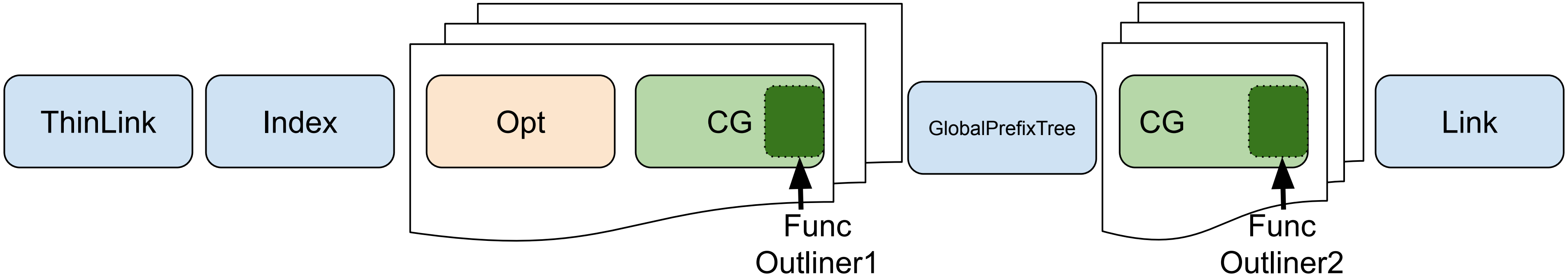
Overview of ThinLTO Pipeline

- Opt + Codegen (CG) for each module run in parallel
- Func Merger is at a late Opt (IR) pass
- Function Outliner is at a late Codgen (Machine IR) pass



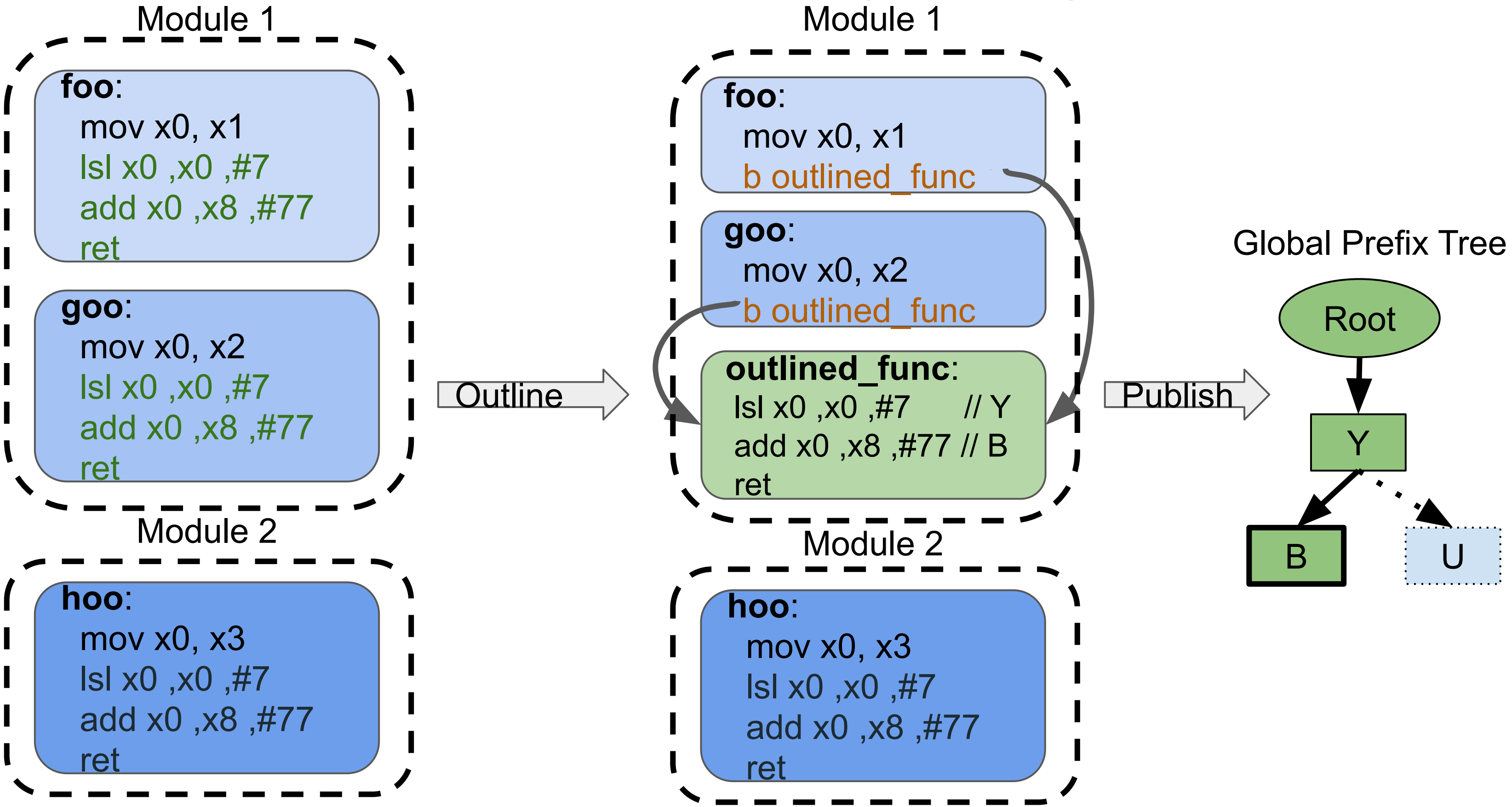
Overview of ThinLTO + Global Func Outliner [2]

- Run two-codegen (CG) rounds for (machine) function outliner
  - The 1st outlining runs **locally** and publishes stable hashes of local outlining instances
  - The 2nd outlings finds **cross-module** candidates matched in the global prefix tree
  - Linker folds identically outlined functions.



[2] Efficient Profile-Guided Size Optimization for Native Mobile Applications, CC2022. https://doi.org/10.1145/3497776.3517764

ThinLTO + Global Func Outliner (1st CG)



Module 1

```
foo:
  mov x0, x1
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

```
goo:
  mov x0, x2
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

Module 2

```
hoo:
  mov x0, x3
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

Module 1

```
foo:
  mov x0, x1
  b outlined_func
```

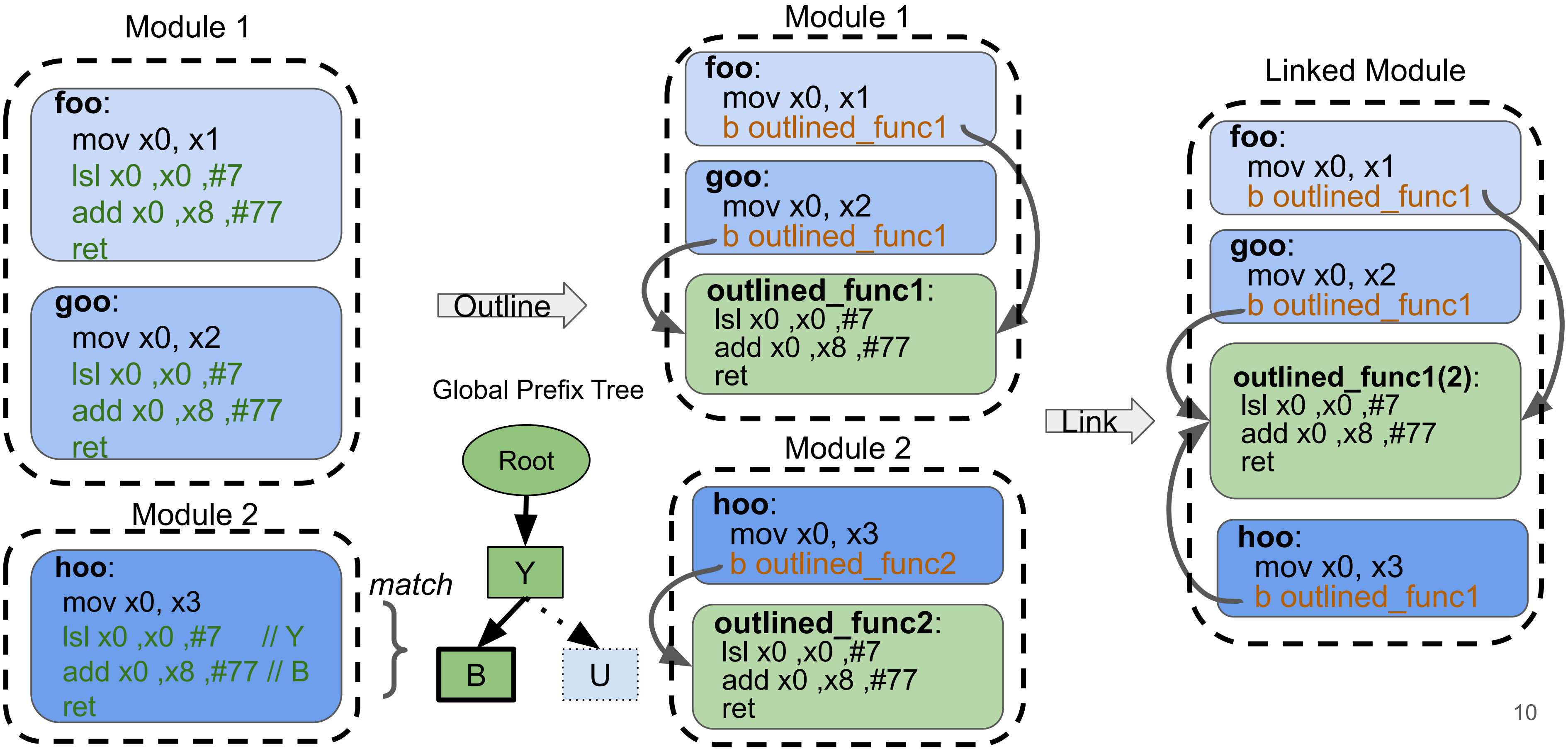
```
goo:
  mov x0, x2
  b outlined_func
```

```
outlined_func:
 lsl x0 ,x0 ,#7      // Y
 add x0 ,x8 ,#77 // B
 ret
```

Module 2

```
hoo:
  mov x0, x3
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

ThinLTO + Global Func Outliner (2nd CG)



Module 1

```
foo:
  mov x0, x1
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret

goo:
  mov x0, x2
  lsl x0 ,x0 ,#7
  add x0 ,x8 ,#77
  ret
```

Module 2

```
hoo:
 mov x0, x3
 lsl x0 ,x0 ,#7      // Y
 add x0 ,x8 ,#77 // B
 ret
```

Module 1

```
foo:
  mov x0, x1
  b outlined_func1

goo:
  mov x0, x2
  b outlined_func1

outlined_func1:
 lsl x0 ,x0 ,#7
 add x0 ,x8 ,#77
 ret
```

Module 2

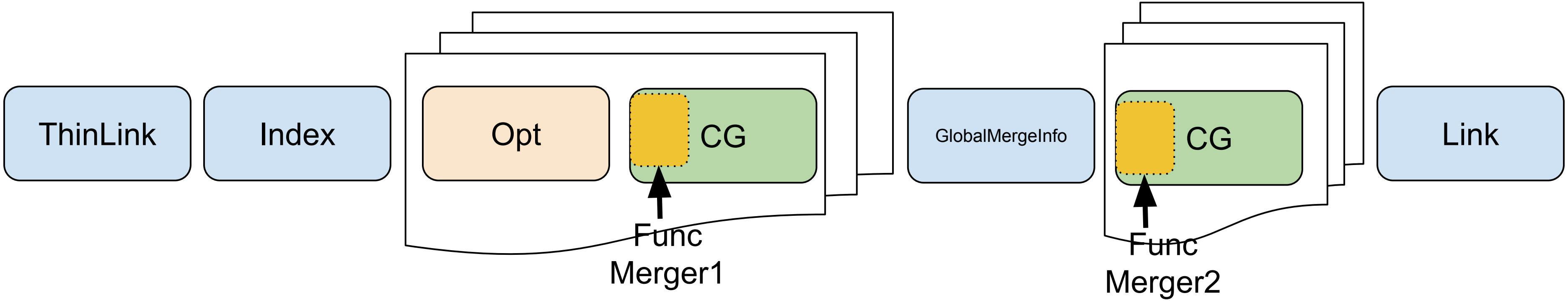
```
hoo:
  mov x0, x3
  b outlined_func2

outlined_func2:
 lsl x0 ,x0 ,#7
 add x0 ,x8 ,#77
 ret
```

Linked Module

```
foo:
  mov x0, x1
  b outlined_func1

goo:
  mov x0, x2
  b outlined_func1

outlined_func1(2):
 lsl x0 ,x0 ,#7
 add x0 ,x8 ,#77
 ret

hoo:
  mov x0, x3
  b outlined_func1
```

ThinLTO + Global Func Merger

- Push down function merger from Opt to the pre-CG hook
- Run two-codegen (CG) rounds for function merger
  - The 1st CG just **analyzes** functions to compute *StableFunction*
  - The 2nd CG actually **merges** functions using *GlobalMergeInfo*



ThinLTO + Global Func Merger (1st CG)

- Compute *StableFunction* which is independent of IR
- Publish it to a global state, *GlobalMergeInfo*

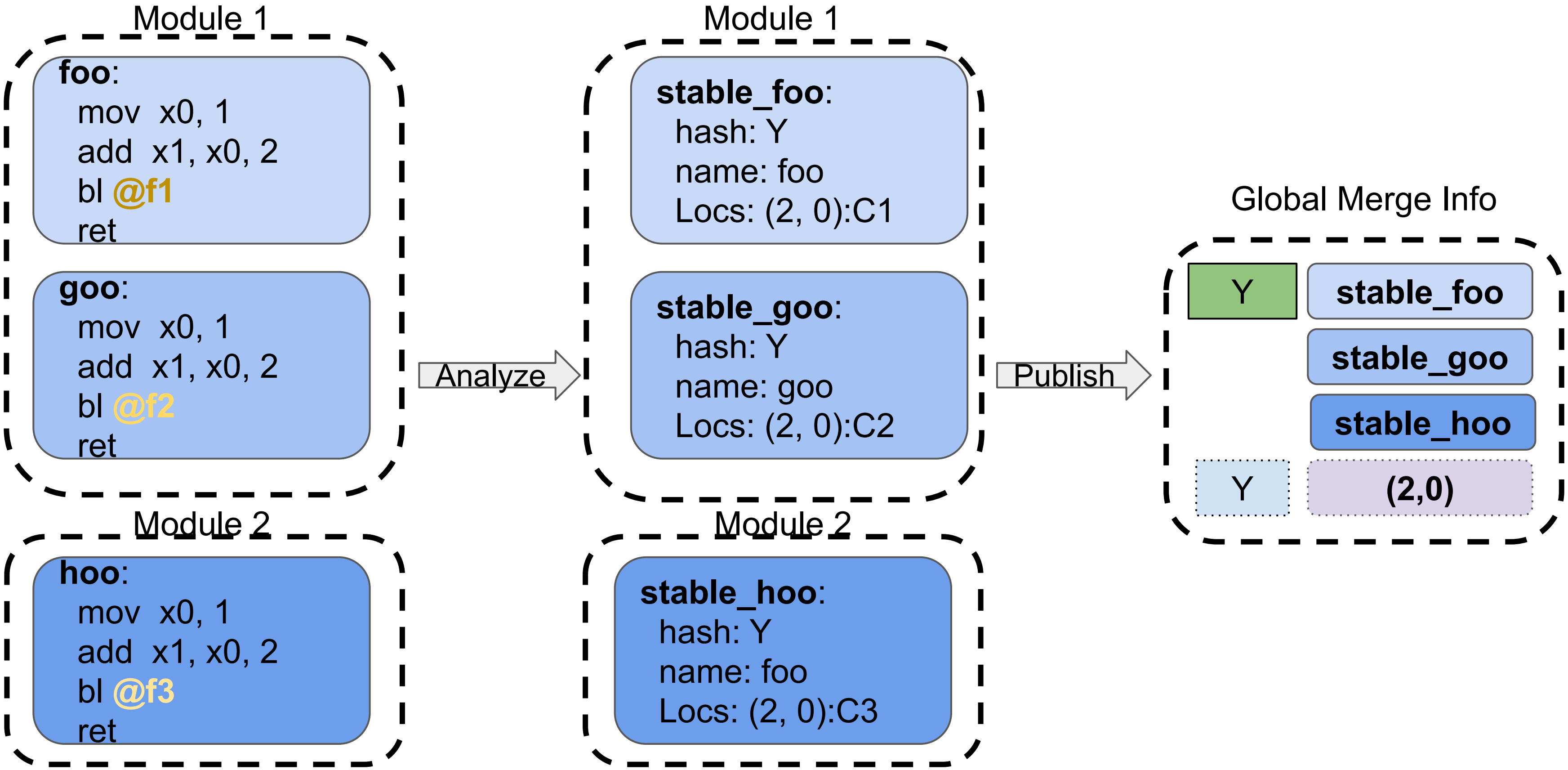
```
struct StableFunction {
 /// Stable hash ignoring Const for eligible ops.
 uint64_t StableHash = 0;

 /// Function name
 std::string Name;

 /// Module identifier
 std::string ModuleIdentifier;

 /// Map of (inst, opnd) indices to the Const hash
 InstOpndIdConstHashMapTy InstOpndIndexToConstHash;
...
};
```

ThinLTO + Global Func Merger (GlobalMergeInfo)

- All stable functions are registered to *StableHashToStableFuncs*
- Once joined, determine *StableHashParams* that will supply original Constants

```
struct GlobalMergeInfo {
 /// A map from stable function hash to stable functions.
 StableHashToStableFuncsTy StableHashToStableFuncs;

 /// A map from stable function hash to parameters pointing
to the pair of (instruction, operand) indices.
 StableHashParamsTy StableHashParams;

 /// mutex when updating the global merge function info.
 std::mutex MergeMutex;
...
};
```

ThinLTO + Global Func Merger (1st CG)



Module 1

```
foo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f1
  ret
```

```
goo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f2
  ret
```

Module 2

```
hoo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f3
  ret
```

Analyze

Module 1

stable_foo:
hash: Y
name: foo
Locs: (2, 0):C1

stable_goo:
hash: Y
name: goo
Locs: (2, 0):C2

Module 2

stable_hoo:
hash: Y
name: foo
Locs: (2, 0):C3

Publish

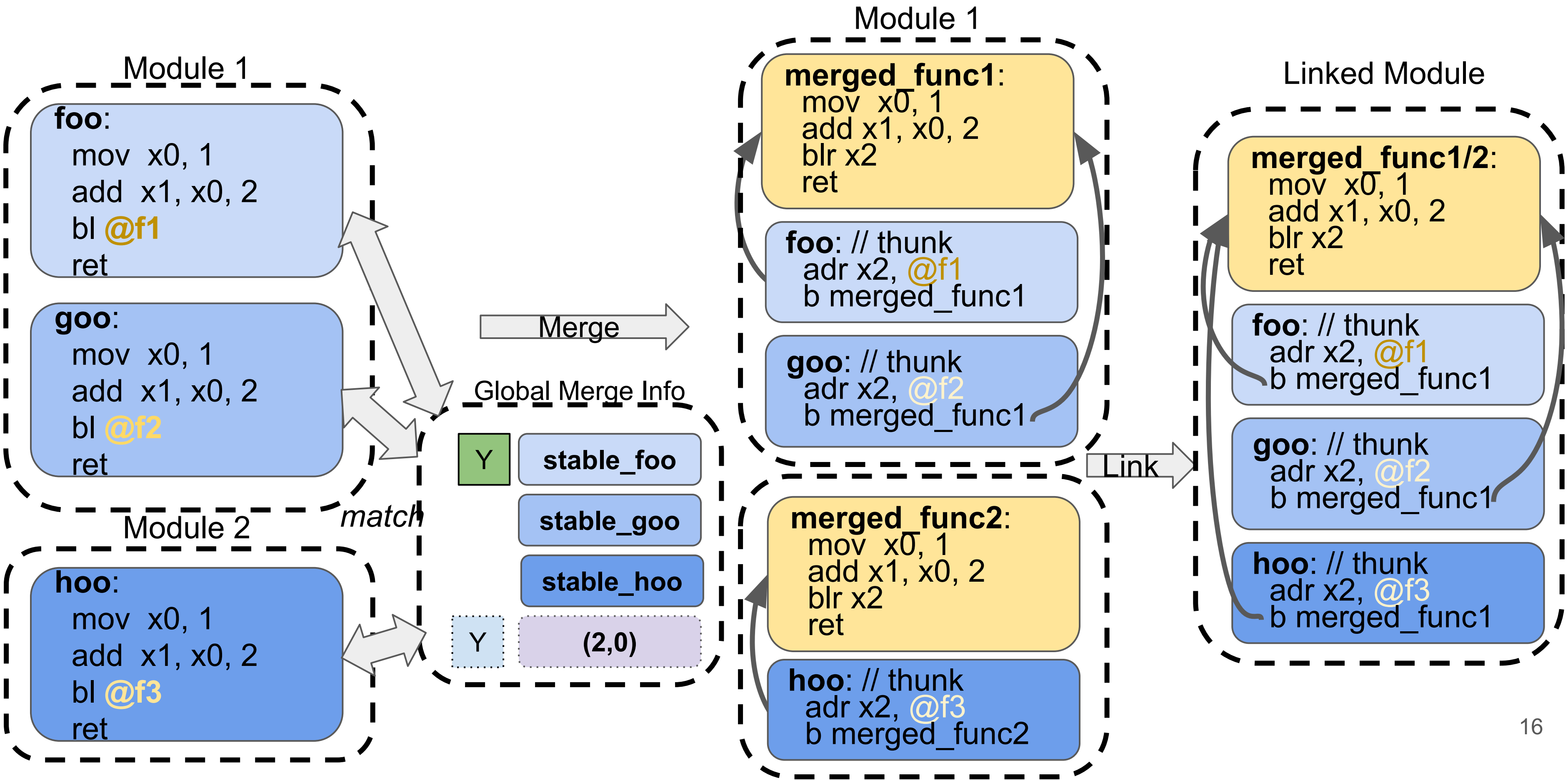
Global Merge Info

Y	**stable_foo**
	stable_goo
	stable_hoo
Y	**(2,0)**

ThinLTO + Global Func Merger (2nd CG)

- Optimistically merge functions using *GlobalMergeInfo*
  - Find a set of *StableFunctions* matched in the current module
  - Ensure those functions are mergeable with IRs by a local merge function (LMF)
  - The first function supplies the body of a (local) *merged_function* while the original functions become thunks.
- Linker folds identically *merged_functions* via ICF (deduplication).

ThinLTO + Global Func Merger (2nd CG)



Module 1

```
foo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f1
  ret
```

```
goo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f2
  ret
```

Module 2

```
hoo:
  mov  x0, 1
  add  x1, x0, 2
  bl @f3
  ret
```

Global Merge Info

Y	stable_foo
	stable_goo
	stable_hoo
Y	(2,0)

Module 1

```
merged_func1:
  mov  x0, 1
  add x1, x0, 2
  blr x2
  ret
```

```
foo: // thunk
  adr x2, @f1
  b merged_func1
```

```
goo: // thunk
  adr x2, @f2
  b merged_func1
```

```
merged_func2:
  mov  x0, 1
  add x1, x0, 2
  blr x2
  ret
```

```
hoo: // thunk
  adr x2, @f3
  b merged_func2
```

Linked Module

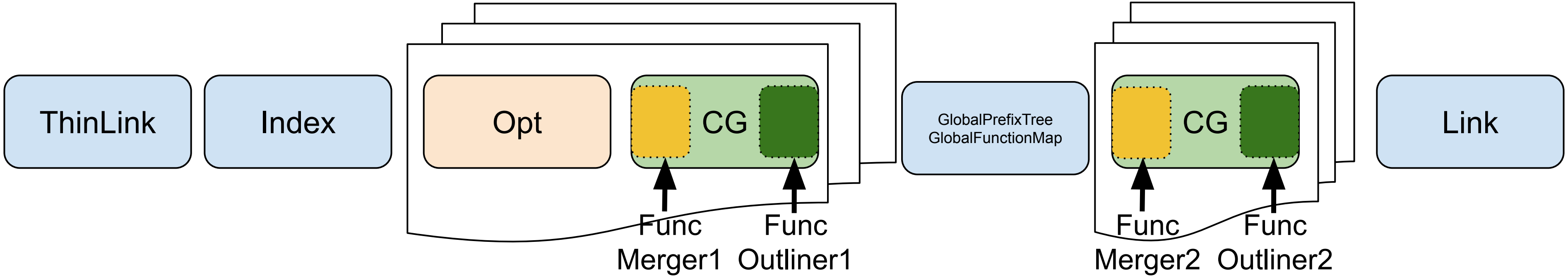
```
merged_func1/2:
  mov  x0, 1
  add x1, x0, 2
  blr x2
  ret
```

```
foo: // thunk
  adr x2, @f1
  b merged_func1
```

```
goo: // thunk
  adr x2, @f2
  b merged_func1
```

```
hoo: // thunk
  adr x2, @f3
  b merged_func1
```

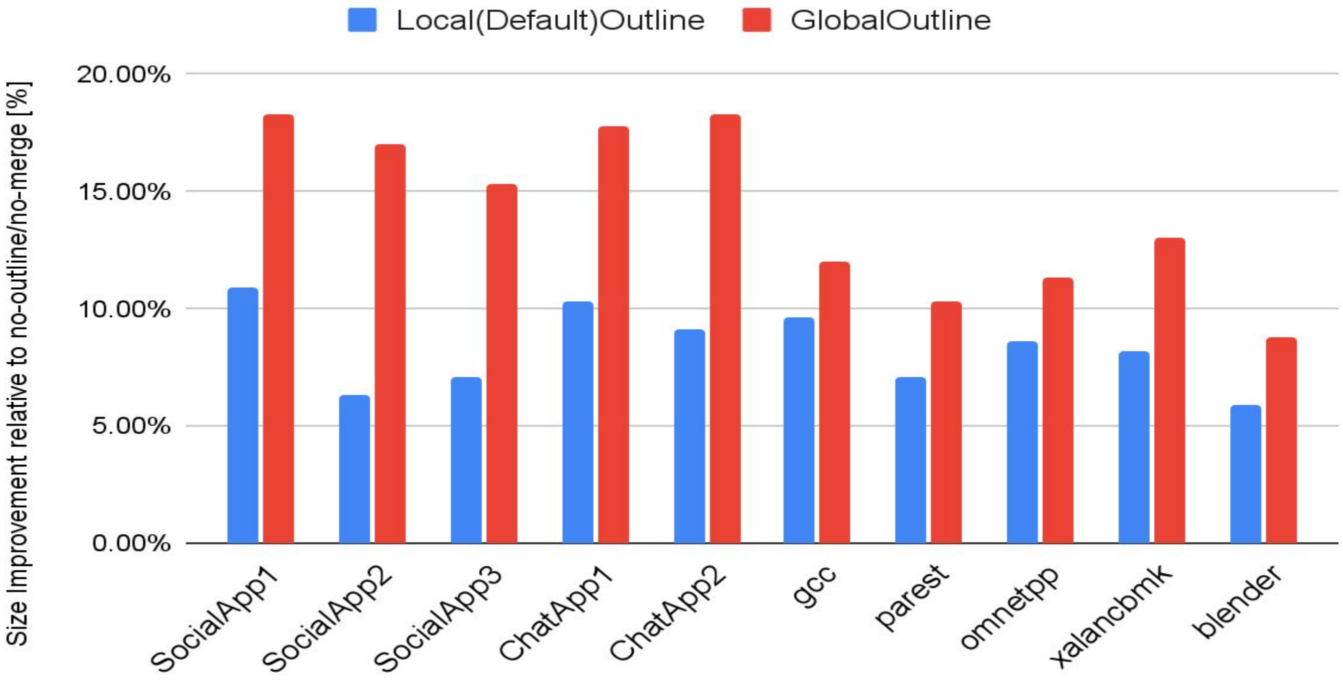
ThinLTO + Global Func Merger + Global Func Outliner



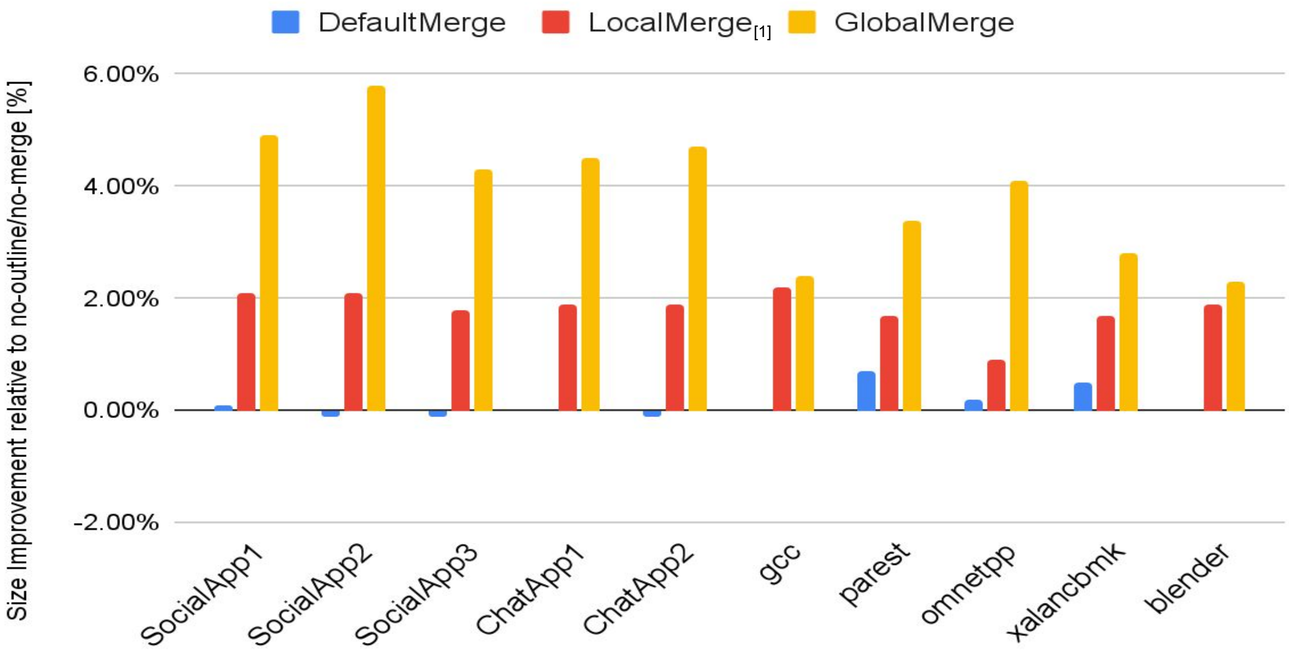
Evaluation

- Benchmarks compiled with -Oz + ThinLTO
  - Mobile Apps (iOS)
    - Objective-C/Swift
    - Code size ranges from 50M to 200M
  - Spec CPU@2017 (MacOS)
    - C/C++
    - Code size range is < 10M
- Size Saving from Function Outlining and/or Function Merge
- ThinLTO time increase for two codegen rounds: 6 ~ 40%
  - Much less than (*full*)LTO time 200% ~ 300%
  - Can avoid two codegen rounds by getting codegen artifacts from the prior builds.

Function Outlining Saving (/w-o Function Merge)

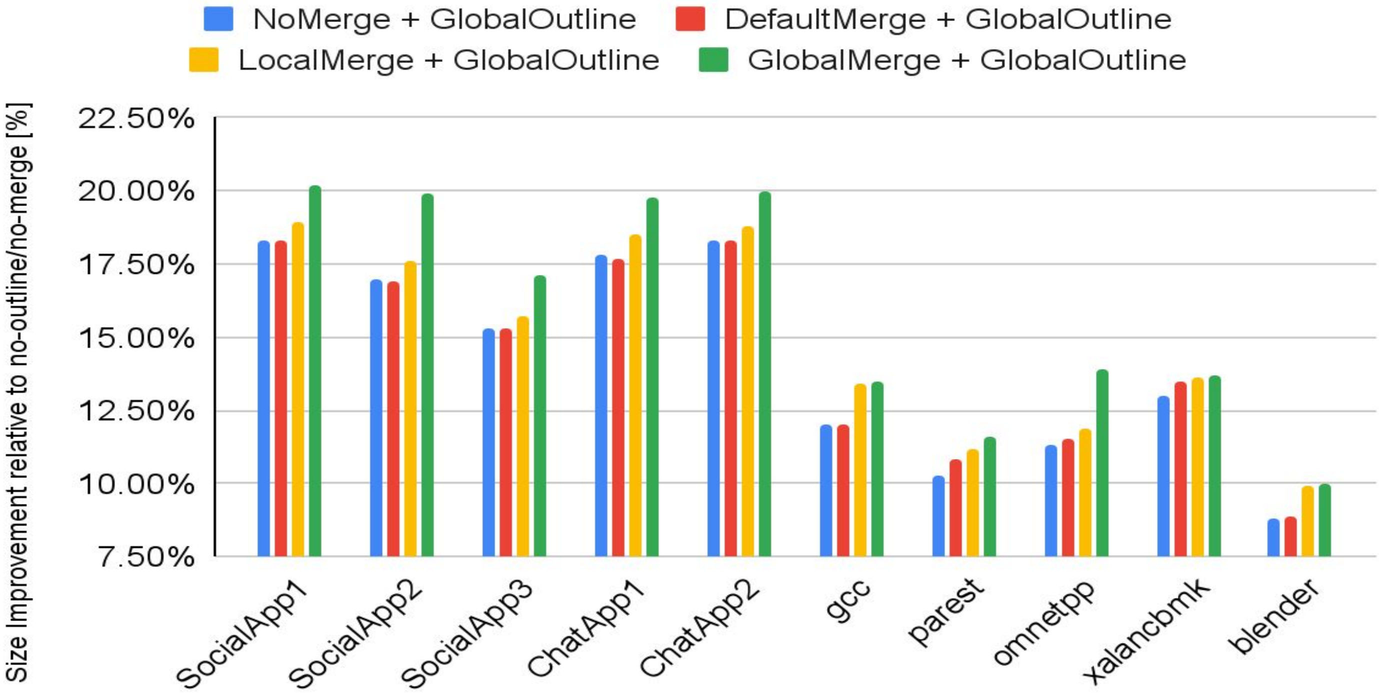


Function Merge Saving (/w-o Function Outline)



[1] Merge similar functions for swift, https://github.com/apple/swift/blob/main/lib/LLVMPasses/LLVMMergeFunctions.cpp

Function Merge Saving (/w Global Function Outline)



Summary

- On top of the state-of-the-art outliners [2,3] with ThinLTO + -Oz, evaluated code size reduction on mobile apps:
  - Built-in LLVM merge function: +/- 0.1%
  - Local merge function: 0.4% ~ 1.2%
  - Global merge function: 2.1% ~ 3.5%
- Ongoing/Future work
  - Serialize codegen artifacts for single codegen
  - LD64 vs. LLD integration for Darwin
  - Upstream

[2] Efficient Profile-Guided Size Optimization for Native Mobile Applications, CC2022. https://dl.acm.org/doi/10.1145/3497776.3517764

[3] An experience with code-size optimization for production iOS mobile applications, https://dl.acm.org/doi/10.1109/CGO51591.2021.9370306