Profiling-Based Global Machine Outlining

Gai Liu, Bo Hu, Zhuoli Li, Nian Sun, Luchuan Guo

ByteDance

Machine Outlining

- Replacing repeated sequences of instructions with calls to equivalent functions [Paquette, 2016]

```
ldr x8, [x19, #8]
str x0, [x19, #8]
mov x0, x8
bl  _objc_release
mov x0, x19
…
ldr x8, [x19, #8]
str x0, [x19, #8]
mov x0, x8
bl  _objc_release
…
```

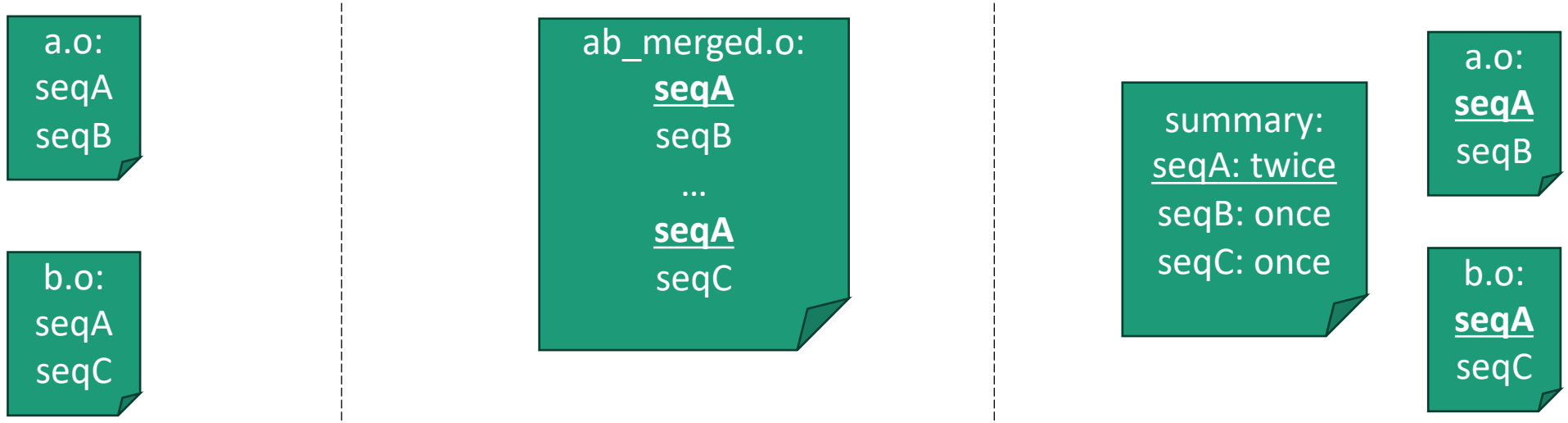
```
bl OUTLINED_FUNCTION_1
mov x0, x19
…
bl OUTLINED_FUNCTION_1
…
```

```
OUTLINED_FUNCTION_1:
  ldr x8, [x19, #8]
  str x0, [x19, #8]
  mov x0, x8
  b _objc_release
```


ByteDance

Global Machine Outlining

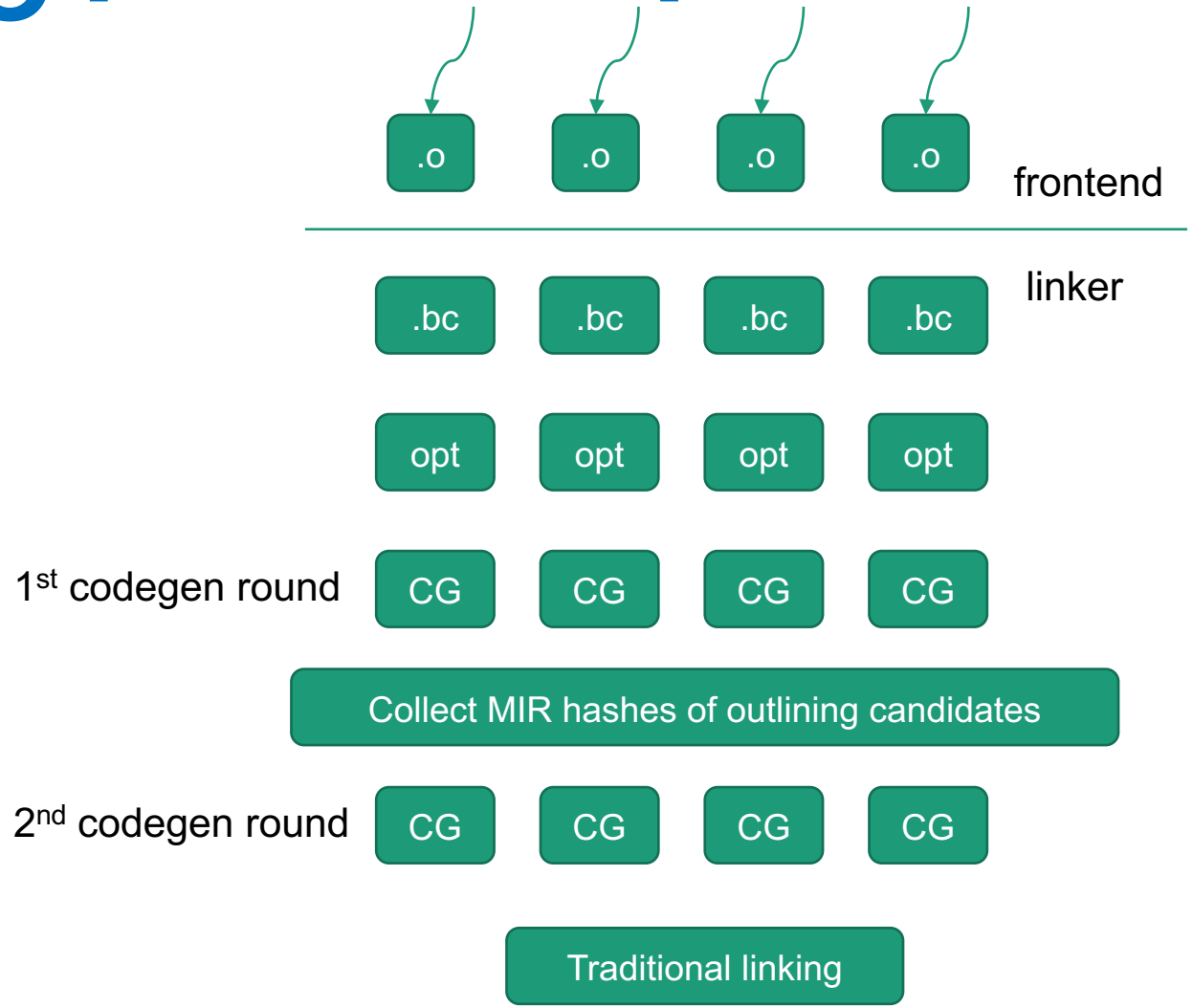
- Goal: outline common sequences across compilation units
  - Use LTO-based techniques
- Example: correctly deciding to outline seqA from different CUs



ByteDance

Two-Round ThinLTO [Lee, et al., 2020]

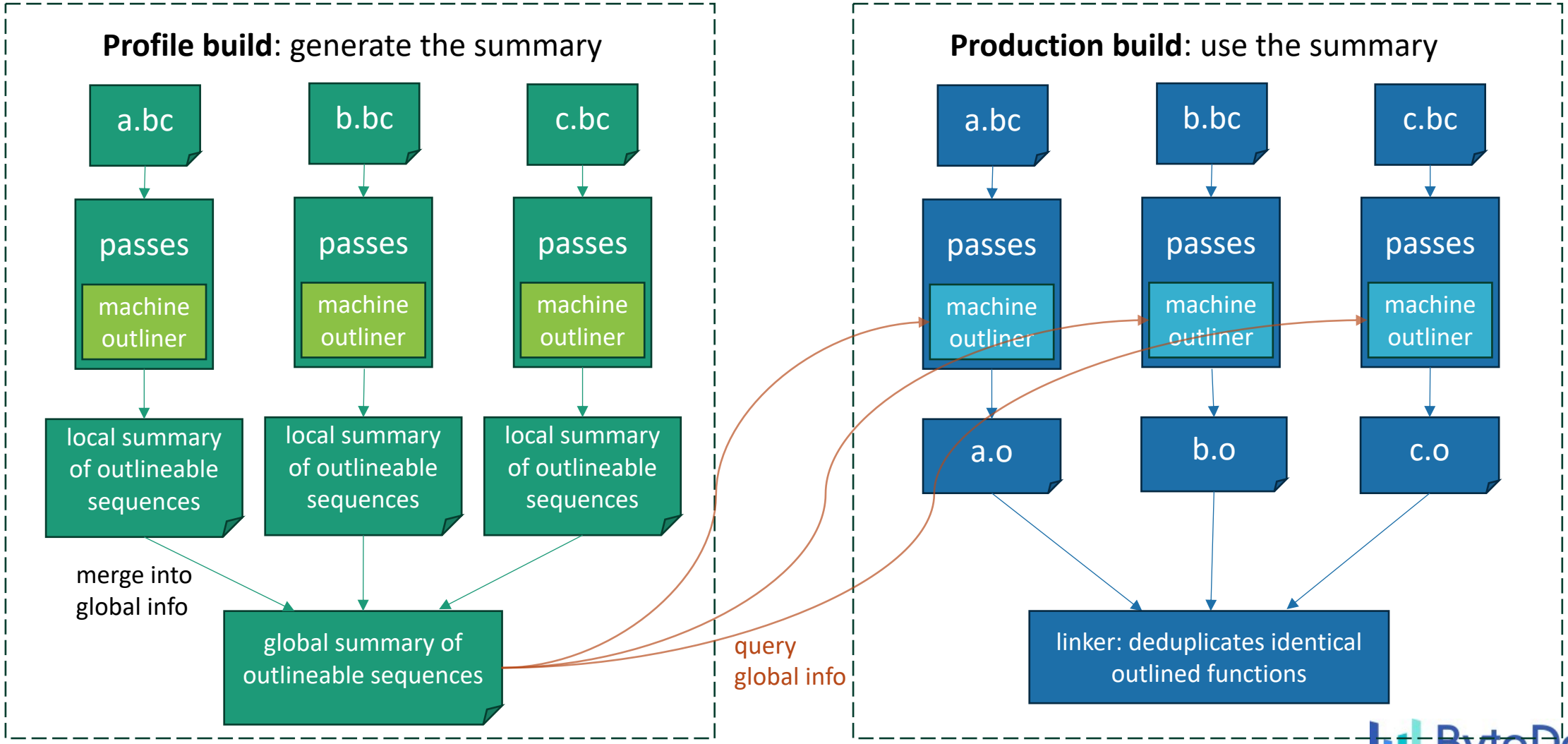
- *1st round*: gather MIR hashes (summary) of outlined functions
- *2nd round*: outline more candidates that match MIR hashes
- compile time increased by ~50%



Our Proposal

- Observation
  - Production apps evolve slowly
  - Summaries change little in consecutive builds
- Idea
  - Decouple summary generation and summary consumption
    - Like profile-guided optimization, but “profiling” done during compilation
  - Summary generation done offline during profile builds
  - Production builds are now faster since they directly read the summary

High Level Design



ByteDance

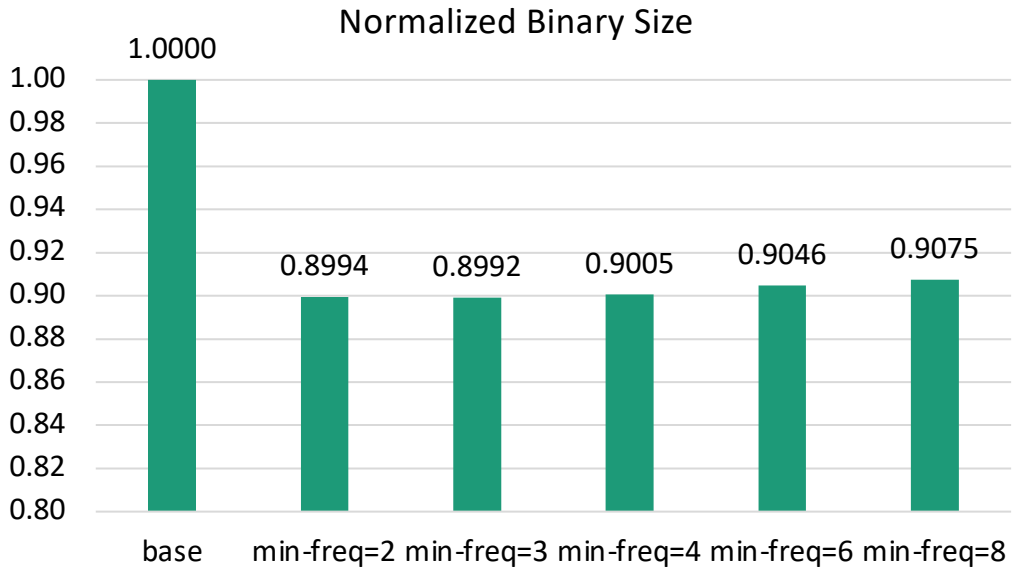
Implementation Details

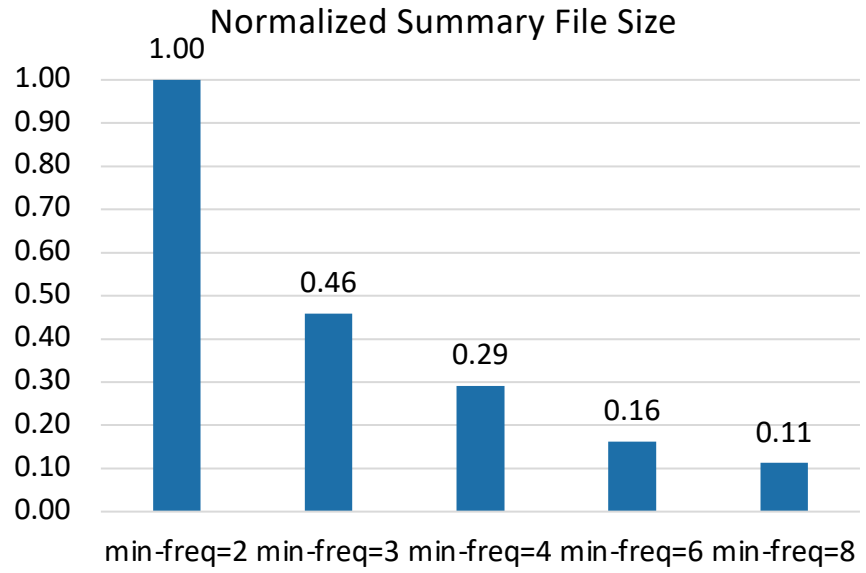
- Each entry in the summary is a pair of *(sequence hash, frequency)*
- Summary reading speed is key to reduce overall compile time overhead
  - Small summary file: add option *–min-seq-freq* controls the minimum occurrence of sequences in the summary
  - Fast lookup: sort entries based on hash values to speed up lookup during production runs
- In production build, machine outliner considers every sequence within certain sizes:
  - $OutliningCost = GlobalOccurenceCount * PerCallOverhead + SequenceSize + FrameOverhead$
  - $NotOutliningCost = GlobalOccurenceCount * SequenceSize$
  - Beneficial if $OutliningCost < NotOutliningCost$

ByteDance

Experiment

- Tested on a demo Swift app (~10MB binary size)
- *min-freq* controls the minimum frequency of sequences in the summary
  - Sequences with occurrence < min-freq are dropped
- Compile time increase is barely measurable due to small app size
  - <10% compile time increase in our internal large apps with *min-freq* >= 4





ByteDance

Conclusions

- Compilation time for production builds is an important metric in our build flow
- We proposed a profiling-based technique to speed up global machine outlining without reducing its effectiveness
- The proposed technique could potentially be applied to other ThinLTO-like optimizations

ByteDance