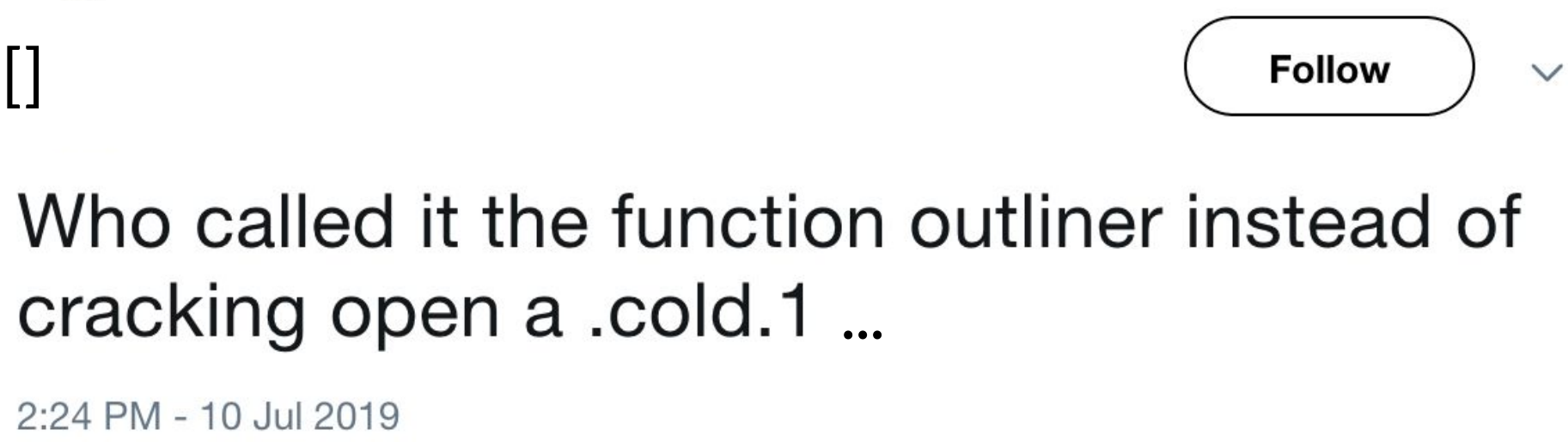
Hot cold splitting in LLVM

Aditya Kumar
Facebook



With apologies to the Tweeter...

"... but, yet, it's one of the most interesting things that happened in the LLVM optimizer this year."

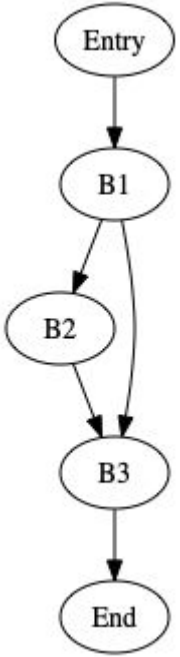
Anonymous Reviewer

Hot cold splitting

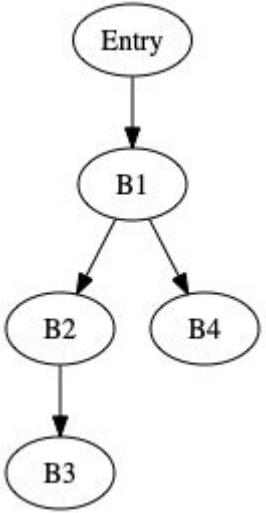
- Intro
- Regions
- Marking Edges
- Propagating Profile Info
- Extracting maximal region
- Experimental Results
- Opportunities for improvement

Regions

1. SESE
2. SEME



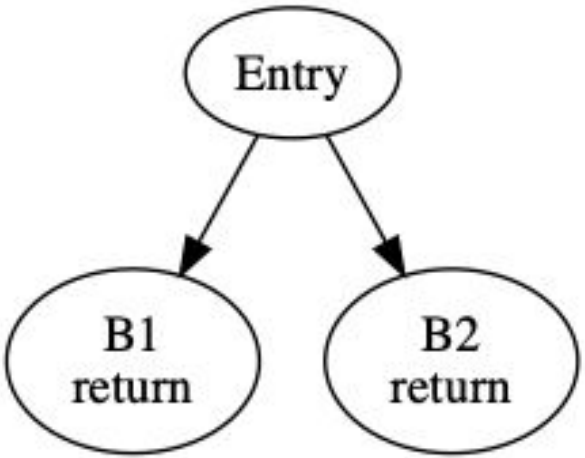
SESE

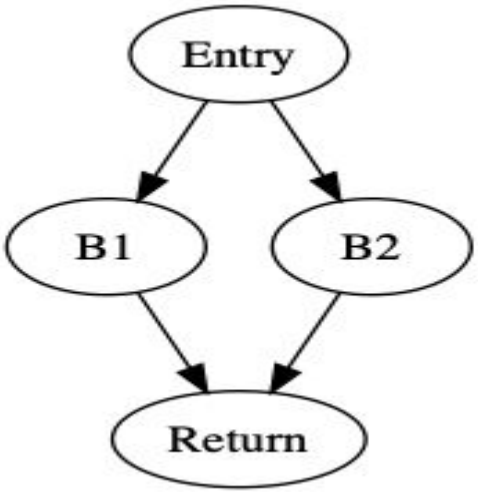


SEME

Image source: https://upload.wikimedia.org/wikipedia/commons/3/30/Some_types_of_control_flow_graphs.svg

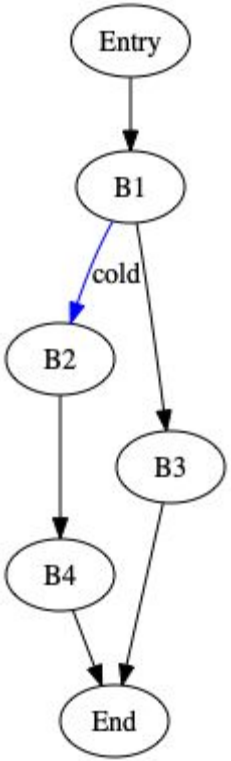
Converting SEME to SESE





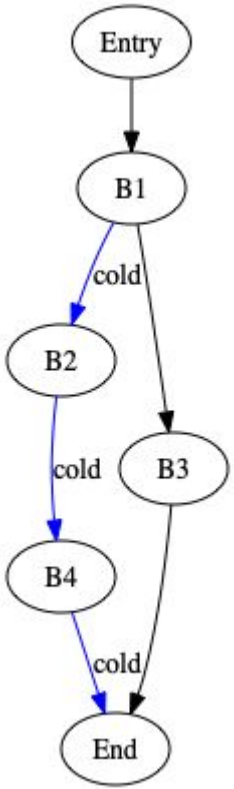
Marking Edges

- Using static analysis
    - e.g., __builtin_expect, assertions, non-returning functions, catch-block
- Using dynamic profile information



Propagating Profile Info

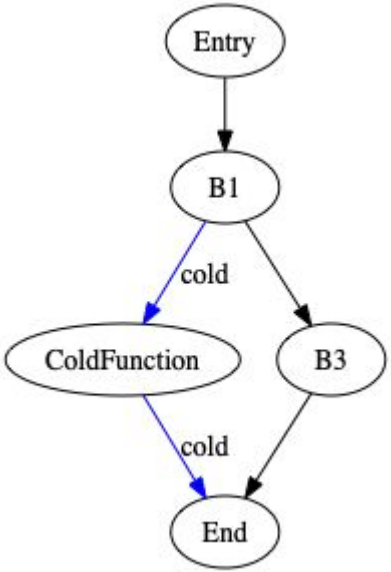
- Using dominance and post-dominance



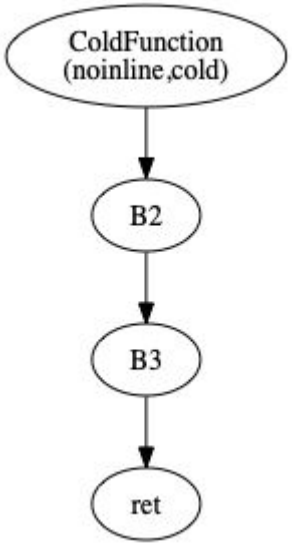
CFG of ‘`foo`’

Extracting cold region

1. Find maximal region
2. Compute inputs outputs
3. Extract as function
4. Add attributes
    - noinline, minsize, cold



CFG of
‘foo’



CFG of
‘foo.cold.1’

Design decisions (implementing in the middle end)

Advantages

Focus on the optimization and tuning

Optimize cold functions for size

Take advantage of (thin)LTO

Helps all backend targets

Low maintenance overhead

Drawbacks

Architecture specific opportunities

Applications benefitting from HotColdSplitting

High icache misses

- Code with lots of branches
- Smaller page size

High premain time

- Reduce startup working set

Experiment Evaluation

Experimental setup

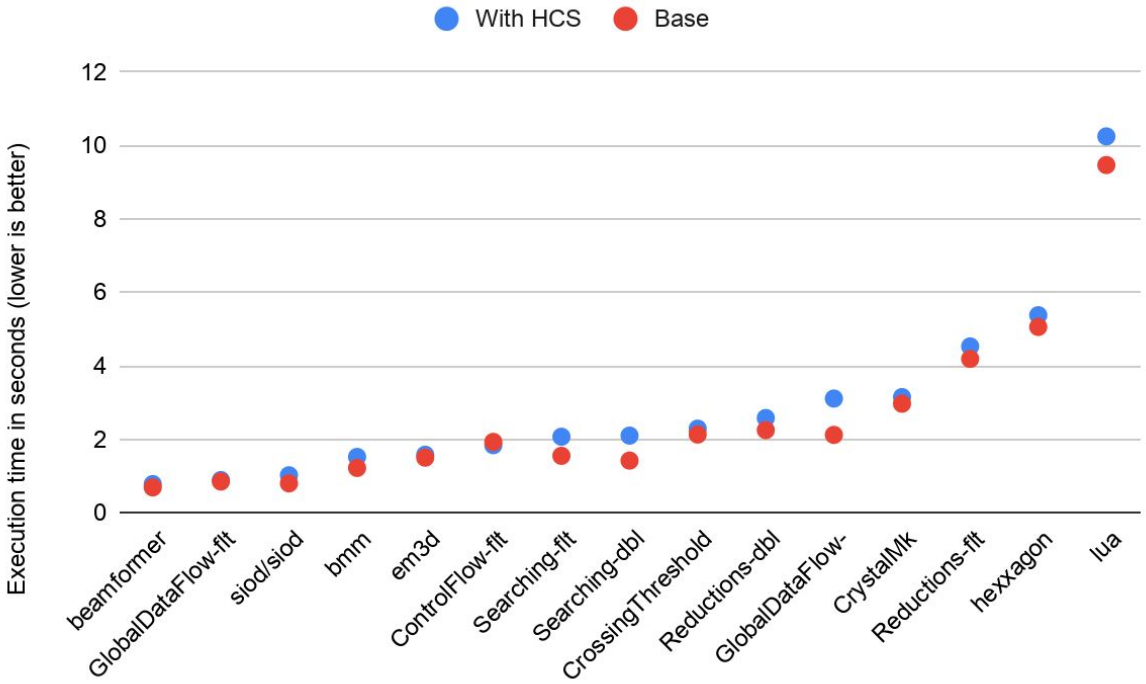
- 2 step build with PGO or AutoFDO

Measurements

- Measure pre-main metrics e.g., page faults
- iCache misses (`perf stat -e icache.misses`)
- Field data
- Code size

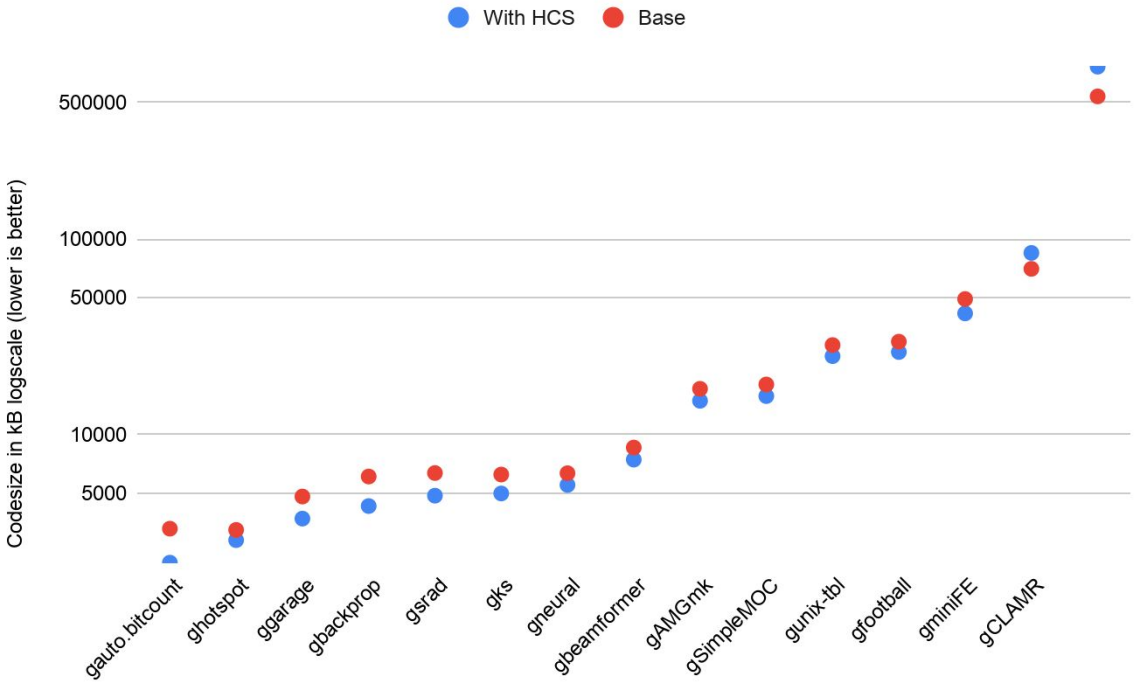
Execution time

LLVM Testsuite



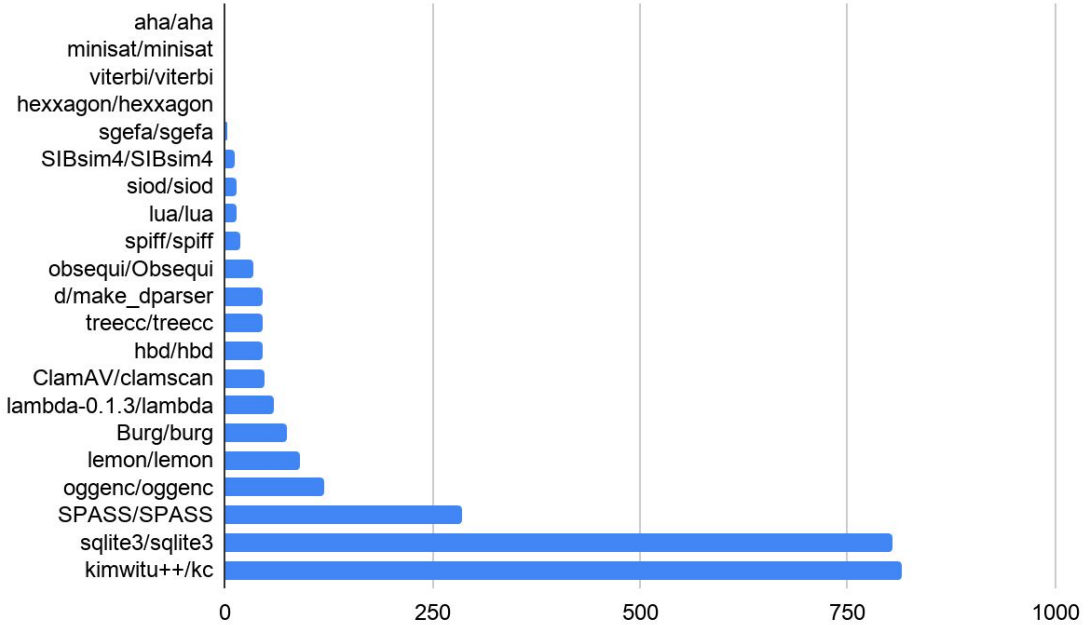
Code size

LLVM Testsuite

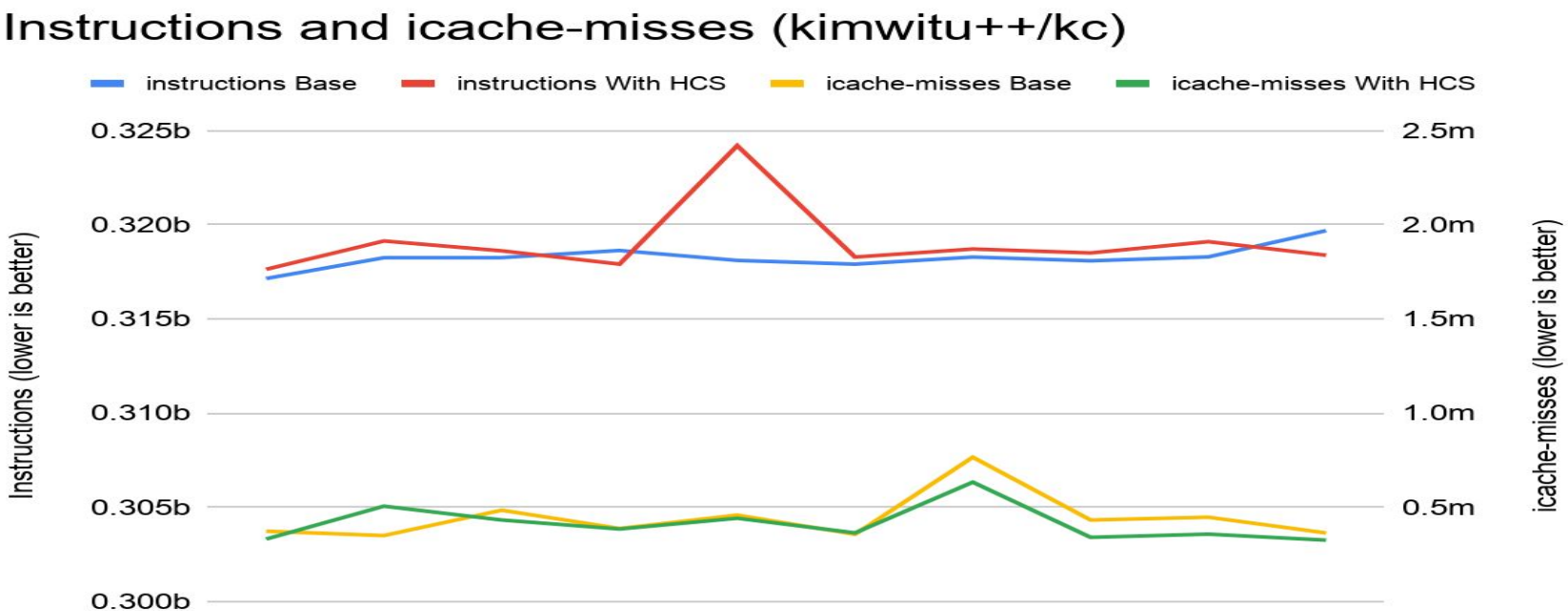


LLVM-testsuite (# of functions outlined)

LLVM Testsuite



LLVM testsuite (perf stat*)



** perf stat -e instructions,icache.misses (try `perf list` to find out other metrics of interest)*

Impact

1. Enabled in Xcode, swift-llvm
2. ios-13 shipped with hot cold splitting enabled
    - All core libraries e.g., libc++, libSystem, dyld, CoreFoundation, UIKit, SSL

Opportunities for improvement

1. Concepts of hot-cold
2. Outlining maximal regions
3. Improving static analysis
4. Improving Code Extractor
5. Tuning cost model for code-size
6. Merge Similar Function meets Hot Cold Splitting
7. Outlining regions post-dominated by non-returning function calls (D69257)

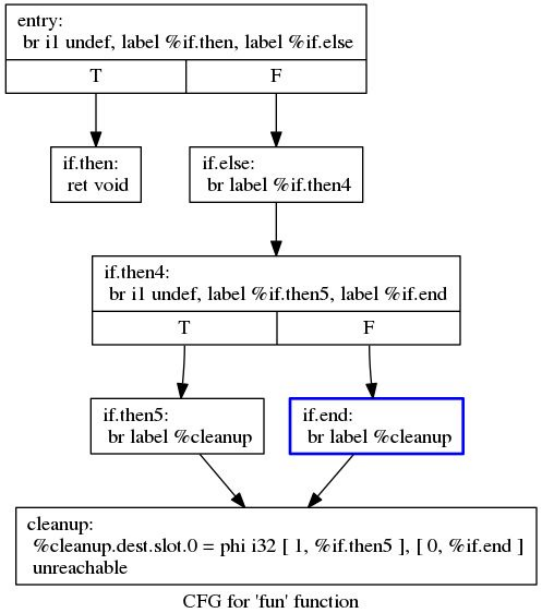
Concepts of hot-cold partitioning

Hot = interesting

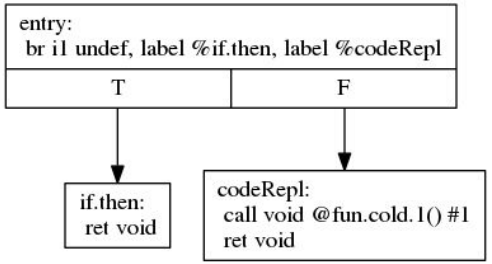
Cold = not interesting

- Randomly outlining code
  - https://reviews.llvm.org/D65376
- Hard coding custom sub-graphs
  - Or pass as compiler flags

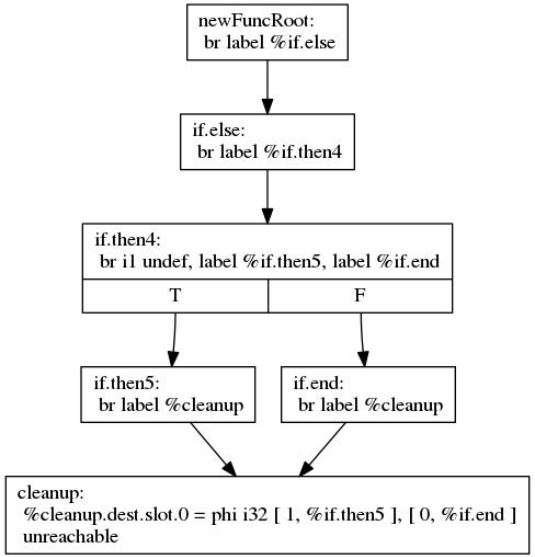
Outlining maximal regions



CFG for 'fun' function



CFG for 'fun' function

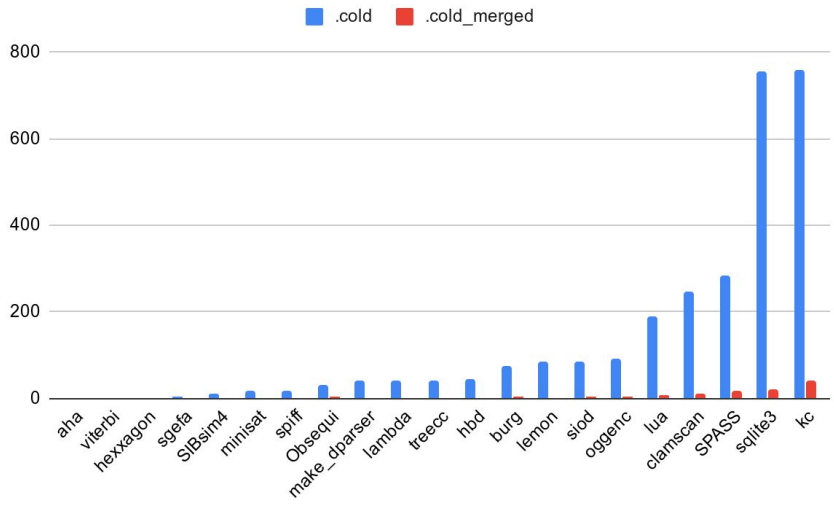


CFG for 'fun.cold.1' function

Merge Similar Function + Hot Cold Splitting

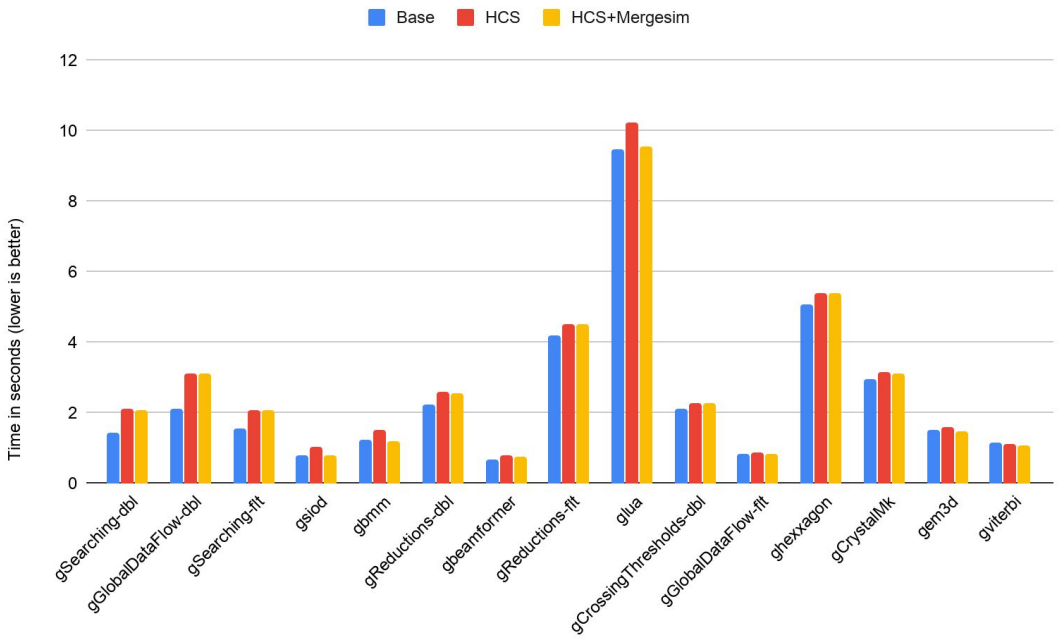
Schedule MergeSim after HotColdSplit

- May improve code-size with appropriate cost model

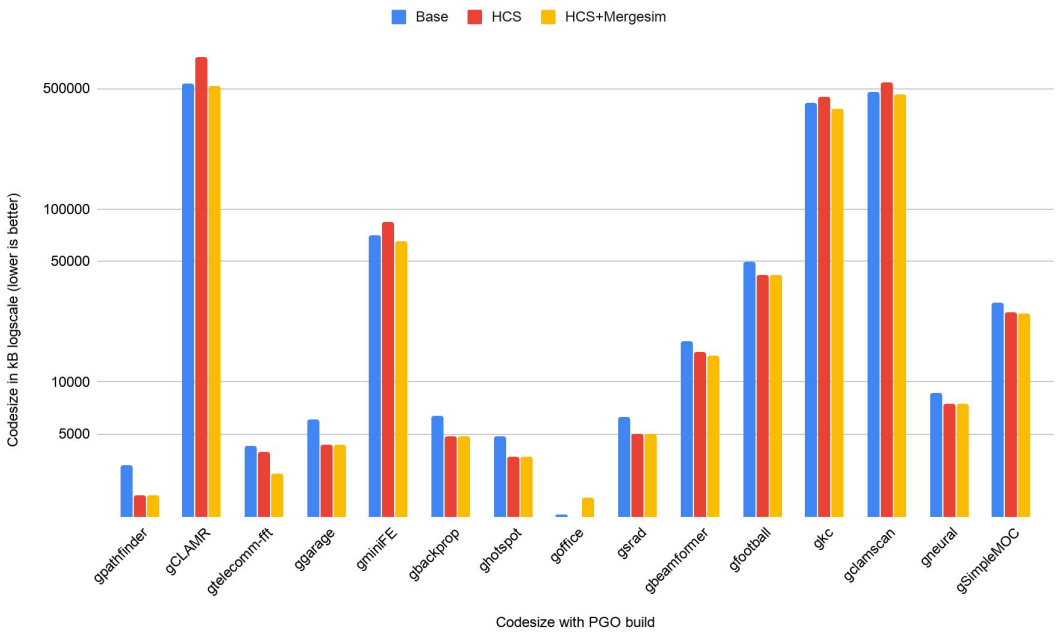


*Repaired the port of merge-similar-functions (MergeSim) to thinLTO https://reviews.llvm.org/D52896

Performance



Codesize


Base
HCS
HCS+Mergesim
500000
100000
50000
10000
5000
Codesize in kB logscale (lower is better)
gpathfinder
gCLAMR
gtelecomm-fft
ggarage
gminiFE
gbackprop
ghotspot
goffice
gsrad
gbeamformer
gfootball
gkc
gclamscan
gneural
gSimpleMOC
Codesize with PGO build

Acknowledgements

Vedant Kumar
Sebastian Pop
Teresa Johnson
Sergey Dmitriev
Krzysztof Parzyszek

```
$ c++filt __Z3fooi
foo(int)
$ c++filt __Z3fooi.cold.1
foo(int) (.cold.1)
$ c++filt __Z3fooi_cold
__Z3fooi_cold
```

References:

https://reviews.llvm.org/D50658
http://lists.llvm.org/pipermail/llvm-dev/2019-January/129606.html

Possible questions

- How does Hot Cold splitting perform in absence of profile information, i.e. using only static analysis?
  - Depends on programmer annotations and programming-language features
  - Only 280 functions outlined in llvm without profile information.
- Is this optimization now mature enough to be ON by default with PGO?
  - Issues with AssumptionCache, and CodeExtractor: PR40710, PR43424
- Difference in performance for C vs C++ applications?
  - Try-catch blocks
- Interaction with code layout optimization which reorder hot/warm BBs to reduce instruction cache misses
  - Reordering doesn't change dominance
- Debuginfo support for this optimization
  - Reasonable?
- How to reduce code-size growth
  - Tune the number of function arguments to be created while splitting