Hot cold splitting in LLVM

Aditya Kumar
Facebook
Who called it the function outliner instead of cracking open a .cold.1 ...
“... 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

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
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

![Graph showing execution time comparison between With HCS and Base]
Code size

LLVM Testsuite
LLVM-testsuite (# of functions outlined)
LLVM testsuite (perf stat*)

Instructions and icache-misses (kimwitu++/kc)

* 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

```
entry:
  br i1 undef, label %if.then, label %if.else

  T  F

  if.then: ret void
  if.else: br label %if.then4

  if.then4:
  br i1 undef, label %if.then5, label %if.end

  T  F

  if.then5: br label %cleanup
  if.end: br label %cleanup

cleanup:
  %cleanup.dest.slot.0 = phi i32 [ 1, %if.then5 ], [ 0, %if.end ]
  unreachable
```

```
entry:
  br i1 undef, label %if.then, label %codeRep1

  T  F

  if.then: ret void
  codeRep1:
  call void @fun.cold.1() #1
  ret void

newFuncRoot:
  br label %if.else

  if.else:
  br label %if.then4

  if.then4:
  br i1 undef, label %if.then5, label %if.end

  T  F

  if.then5: br label %cleanup
  if.end: br label %cleanup

cleanup:
  %cleanup.dest.slot.0 = phi i32 [ 1, %if.then5 ], [ 0, %if.end ]
  unreachable
```
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
Acknowledgements

Vedant Kumar
Sebastian Pop
Teresa Johnson
Sergey Dmitriev
Krzysztof Parzyszek

References:

https://reviews.llvm.org/D50658
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