Efficient clustering of case statements for indirect branch prediction

Evandro Menezes, Aditya Kumar, Sebastian Pop
Samsung Austin R&D Center
{e.menezes, aditya.k7, s.pop}@samsung.com

February 4, 2017
Austin, TX
Branch Prediction

- In super-pipelined processors, there is a significant lag between the beginning of the execution of an instruction and when its result becomes available.
Branch Prediction

• This trait also applies to conditional direct branches
  - However, waiting for the result of the instructions affecting a conditional branch is wasteful.
  - Therefore, the processor makes an educated guess as to where the conditional branch will lead to.
  - If the guess is wrong, it all goes to waste.

• Conditional direct branch prediction techniques aim at increasing the frequency when the processor makes the right guesses.
Indirect Branch Prediction

• Like conditional direct branches, indirect branches may lead to more than one target.

• Unlike conditional direct branches, which may lead to just two targets, indirect branches may lead to multiple targets.

• Therefore, indirect branch prediction techniques have traditionally been less efficient than for conditional direct branch.

• Moreover, because of multiple possible targets, predicting indirect branches needs more resources than predicting conditional direct branches, leading to more implementation compromises and limitations, depending on the transistor budget.
Jump Tables

• In the presence of a large number of conditional cases, it is more efficient to use a table of targets together with indirect branches.
  - Advantages: simple execution, compact code size.
  - Disadvantages: reliance on efficient indirect branch prediction.

• Moreover, with the increased popularity of interpreted languages and object oriented languages, many programs rely heavily on jump tables.
Jump Tables in LLVM

• The previous algorithm used to generate jump tables in LLVM, `SelectionDAGBuilder::findJumpTables()`, by Hans Wennborg, was based on the work by Kannan & Proebsting.
  - Minimizes the number of partitions of clusters of cases by maximizing their sizes.
  - $O(n^2)$

• However, the resulting jump tables have virtually unlimited size, which may overwhelm the processor resources dedicated to indirect branch prediction.
The algorithm in `SelectionDAGBuilder::findJumpTables()` was modified to optionally limit the size of partitions of clusters.

- Suiting them to the limitations of the indirect branch predictor in the target.
- The default, unlimited size, yields the same results as before.

Moreover, since conditional direct prediction tends to be more accurate than indirect branch prediction, it favors conditional direct branches over sparse clusters with few cases.

- Maximizes the number of partitions of clusters by limiting their sizes.
- Maximizes the density of clusters of cases.
  - Fewer sparse partitions, falling back to conditional direct branches.
  - $O(n \log n)$
Jump Tables in LLVM

declare void @g(i32)

define void @test(i32 %x) {
  entry:
    switch i32 %x, label %return [
      i32 1, label %bb1
      i32 2, label %bb2
      i32 3, label %bb3
      i32 4, label %bb4
      i32 14, label %bb14
      i32 15, label %bb15
    ]
  return: ret void
  bb1: tail call void @g(i32 1) br label %return
  bb2: tail call void @g(i32 2) br label %return
  bb3: tail call void @g(i32 3) br label %return
  bb4: tail call void @g(i32 4) br label %return
  bb14: tail call void @g(i32 5) br label %return
  bb15: tail call void @g(i32 6) br label %return
}

-max-jump-table-size=0
Jump Tables in LLVM

declare void @g(i32)

define void @test(i32 %x) {
  entry:
    switch i32 %x, label %return [
      i32 1, label %bb1
      i32 2, label %bb2
      i32 3, label %bb3
      i32 4, label %bb4

      i32 14, label %bb14
      i32 15, label %bb15
    ]
  return: ret void
  bb1: tail call void @g(i32 1) br label %return
  bb2: tail call void @g(i32 2) br label %return
  bb3: tail call void @g(i32 3) br label %return
  bb4: tail call void @g(i32 4) br label %return
  bb14: tail call void @g(i32 5) br label %return
  bb15: tail call void @g(i32 6) br label %return
}
Results: Samsung Exynos M1

SPECint2000

sub-title
Results: Samsung Exynos M1

Other Benchmarks

- Lua
- JPEG
- LZMA

Bar chart showing performance metrics for Lua, JPEG, and LZMA benchmarks.
Results: ARM Cortex A57

SPECint2000

sub-title

94% 96% 98% 100% 102% 104% 106%

∞ 8 16 32 64

164.gzip 175.vpr 176.gcc 181.mcf 197.parser 253.perlbmk 254.gap 255.vortex 256.bzip2 300.twolf
Results: ARM Cortex A57

Other Benchmarks

Lua

JPEG

LZMA

∞ 8 16 32 64
Results: Intel i7 Skylake

SPECint2000

sub-title

---

Minus 175.vpr and 252.eon
Conclusion

• The results depend largely on the implementation of the indirect branch predictor in the processor and, of course, on the workload.

• In particular, though a jump table may have a large number of entries, only a few of them may be used in a given workload, possibly within the limitations of a particular indirect branch predictor.

Give it a try!
-mlir -min-jump-table-entries=<entries>
-mlir -max-jump-table-size=<entries>
References

• `llvm::SelectionDAGBuilder`

• https://reviews.llvm.org/D21940

• https://reviews.llvm.org/D25212

• Kannan; Proebsting. “Correction to ‘producing good code for the case statement’”, 1994.


• Kim; João; Mutlu; Lee; Patt; Cohn. “VPC Prediction: Reducing the Cost of Indirect Branches via Hardware-Based Dynamic Devirtualization”, 2007.