Profile Guided Function Layout in LLVM and LLD

Michael Spencer (Sony Interactive Entertainment)
LLVM Developers’ Meeting, October 2018
```c
void APInt::tcExtract(WordType *dst, unsigned dstCount, const WordType *src, unsigned srcBits, unsigned srcLSB) {
    unsigned n = dstParts * APINT_BITS_PER_WORD - shift;
    if (n < srcBits) {
    } else if (n > srcBits) {
        if (srcBits % APINT_BITS_PER_WORD)
    }
    while (dstParts < dstCount)
        dst[dstParts++] = 0;
}

void APInt::tcShiftRight(WordType *Dst, unsigned Words, unsigned Count) {
    if (!Count)
        return;
    if (BitShift == 0)
        return;
    else {
        for (unsigned i = 0; i != WordsToMove; ++i)
            if (i + 1 != WordsToMove)
            ...
    }
}
```
Why Function Layout?

- Instruction cache
- Instruction Translation Lookaside Buffer (ITLB)
Design and Implementation

LLVM

Profile

Extract

Layout

compiler-rt/llvm/perf/…

LLD
Profiling

- Standard llvm PGO workflow
Extract

- Need to inform the linker about the profile in a way it can understand
- New LLVM pass CGProfile (Call Graph Profile)
  - Only works with the new Pass Manager
- Extract the weighted call-graph into module metadata
- Runs late in the pipeline
Example

declare void @b()
define void @a() !prof !1 {
    call void @b()
    ret void
}

@foo = common global i32 ()* null, align 8
declare i32 @func1()
...

define void @freq(i1 %cond) !prof !1 {
    %tmp = load i32 ()*, i32 ()** @foo, align 8
call i32 %tmp(), !prof !3
    br i1 %cond, label %A, label %B, !prof !2
A:
    call void @a();
    ret void
B:
    call void @b();
    ret void
}

!llvm.module.flags = !{!0}

!0 = !{i32 5, !"CG Profile", !1}
!1 = !{!2, !3, !4, !5, !6, !7, !8}
!2 = !{void ()* @a, void ()* @b, i64 32}
!3 = !{void (i1)* @freq, i32 ()* @func4, i64 1030}
!4 = !{void (i1)* @freq, i32 ()* @func2, i64 410}
!5 = !{void (i1)* @freq, i32 ()* @func3, i64 150}
!6 = !{void (i1)* @freq, i32 ()* @func1, i64 10}
!7 = !{void (i1)* @freq, void ()* @a, i64 11}
!8 = !{void (i1)* @freq, void ()* @b, i64 21}

!1 = !{"function_entry_count", i64 32}
!2 = !{"branch_weights", i32 5, i32 10}
!3 = !{"VP", i32 0, i64 1600, i64 7651369219802541373, i64 1030, ...}
• How to inform the linker about the profile in a way it understands?
  - MC writes the metadata into the object file
  - New ELF section type SHT_LLVM_CALL_GRAPH_PROFILE
  - List of weighted edges between symbols
  - Normal symbol resolution and merging resolve to the section containing the code
Example – ELF Representation

Module Metadata

```llvm
!llvm.module.flags = !{!0}

!0 = ![i32 5, !"CG Profile", !1]
!1 = ![i2, i3, i4, i5, i6, i7, i8]
!2 = ![void () @a, void () @b, i64 32]
!3 = ![void (i1) @freq, i32 () @func4, i64 1030]
!4 = ![void (i1) @freq, i32 () @func2, i64 410]
!5 = ![void (i1) @freq, i32 () @func3, i64 150]
!6 = ![void (i1) @freq, i32 () @func1, i64 10]
!7 = ![void (i1) @freq, void () @a, i64 11]
!8 = ![void (i1) @freq, void () @b, i64 21]
```

ELF Assembly

```assembly
.cg_profile a, b, 32
.cg_profile freq, func4, 1030
.cg_profile freq, func2, 410
.cg_profile freq, func3, 150
.cg_profile freq, func1, 10
.cg_profile freq, a, 11
.cg_profile freq, b, 21
```
Optimizing Function Placement for Large-Scale Data-Center Applications

Guilherme Ottoni  Bertrand Maher
Facebook, Inc., USA
{ottoni,bertrand}@fb.com

Abstract
Modern data-center applications often comprise a large amount of code, with substantial working sets, making them good candidates for code-layout optimizations. Although

While the large size and performance criticality of such applications make them good candidates for profile-guided code-layout optimizations, these characteristics also impose scalability challenges to optimize these applications.
Call Chain Clustering (aka C³)

• Find a good ordering for functions
  ▪ Minimize the average distance between hot calls and their targets
  ▪ While obeying constraints:
    ✫ Don’t create huge clusters
    ✫ Don’t reduce the density (hotness / size) of the cluster too much

• Single pass over clusters starting at the highest density
  ▪ Merge “good” hot callees

• Sort clusters by density

• Fast:
  ▪ 40ms for a graph of 35,000 functions and 115,000 edges
How To Use

● Profile
  ▪ $ clang -c -fprofile-generate -fexperimental-new-pass-manager ...
  ▪ $ ld ...
  ▪ $ prog
  ▪ $ llvm-profdata merge default.profraw -o default.profdata

● Optimize and Extract
  ▪ $ clang -c -fprofile-use=default.profdata -fexperimental-new-pass-manager ...

● Layout
  ▪ $ llld ...
Results – PlayStation®4 Game A

Frame Time Decrease over no PGO (Higher is Better)

- **CPU Heavy**
  - PGO: 5%
  - PGO + Function Layout: 6%
- **CPU Heavy (Scene Type)**
  - PGO: 13%
  - PGO + Function Layout: Not applicable
- **GPU Heavy**
  - PGO: 1%
  - PGO + Function Layout: 5%
Why the improvement?

- Why
  - Reduction in ITLB miss rate
Results – PlayStation®4 Game ITLB

Control  PGO Without Layout  PGO With Layout
Why the improvement?

- Why
  - Reduction in ITLB miss rate
  - Reduction in instruction cache miss rate
Results – PlayStation®4 Game I$
Future Work

- Improved clustering heuristic
  - Max cluster size
  - Cluster quality degradation
- Function Slicing
  - Separate out cold blocks
  - Separate out hot blocks with different callees
Questions?