What does it take to make LLVM as performant as GCC?

James Molloy
ARM
Ana Pazos
Yin Ma
Qualcomm Innovation Center, Inc.

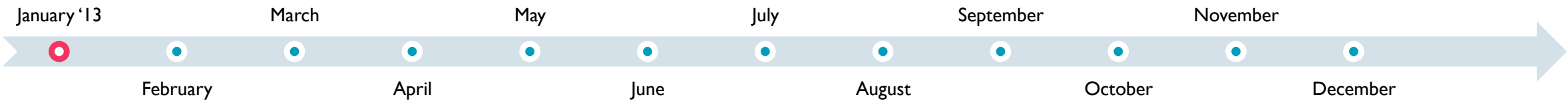
QUALCOMM INNOVATION CENTER, INC.
QuIC

ARM

Agenda

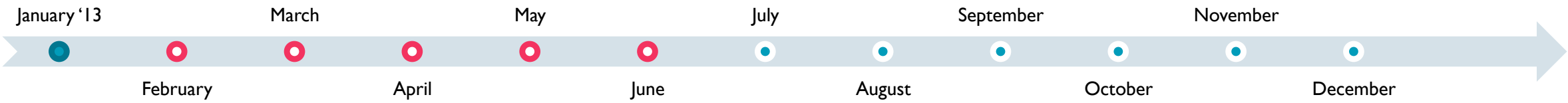
1. Background
2. Problems fixed
3. Current performance (vs GCC)
4. Current work
   - Induction variable selection
   - Addressing mode selection
   - Vectorizer
   - Inliner
5. Future work
6. Conclusions

Background



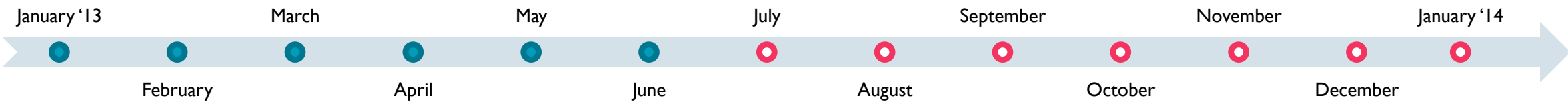
- January 2013 : AArch64 backend initial upstreaming

Background



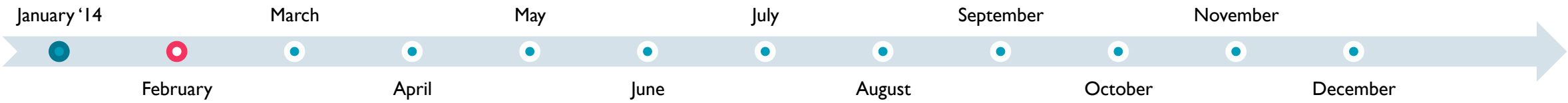
- January 2013 : AArch64 backend initial upstreaming
- February 2013 - June 2013 : conformance checking and fixes

Background

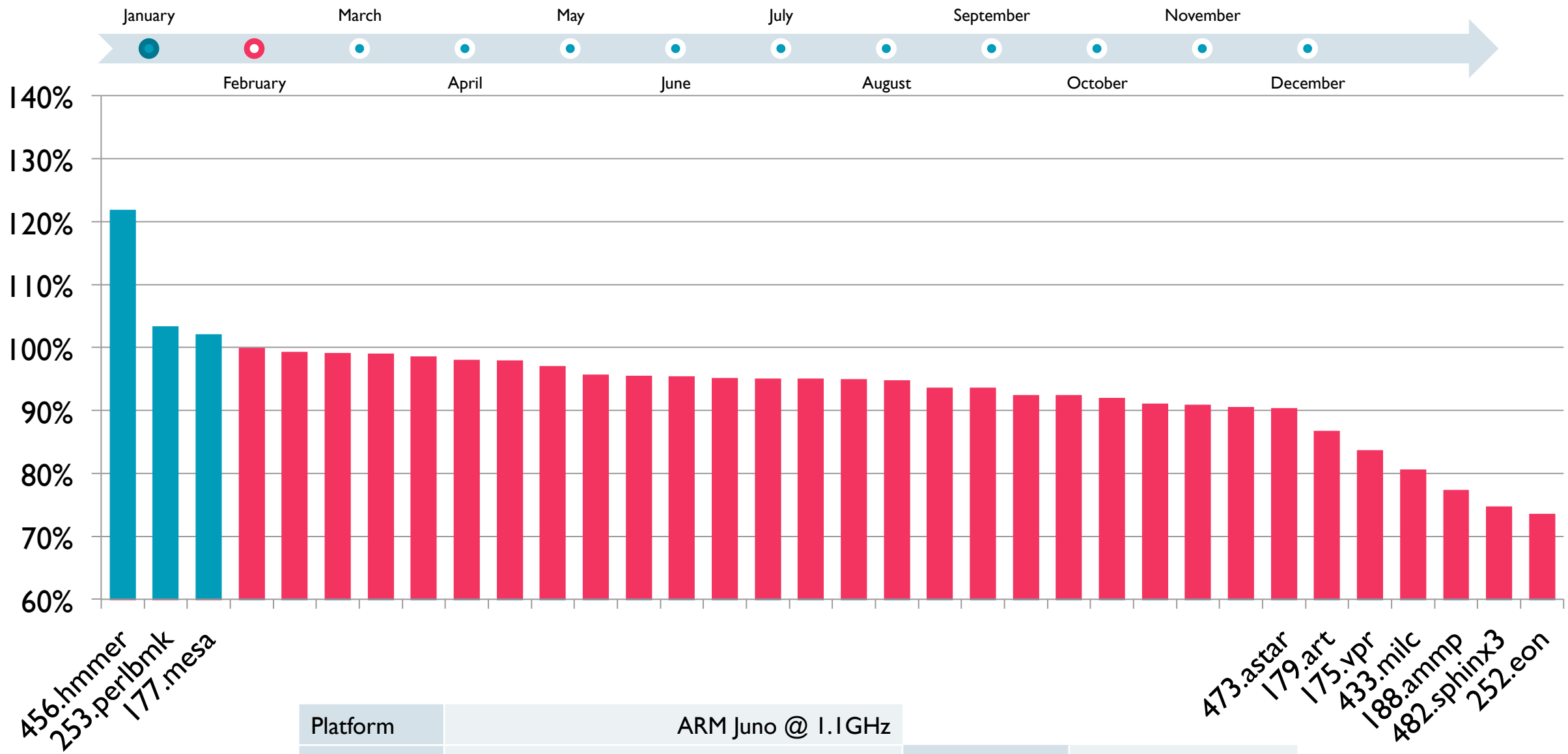


- January 2013 : AArch64 backend initial upstreaming
- February 2013 - June 2013 : conformance checking and fixes
- July 2013 - January 2014 : Implementation of NEON SIMD instructions

Methodology



- First target: SPEC2000 + SPEC2006 (INT+FP)
- GCC had at least half a year (multiple man-years) of tuning
- Start with a differential analysis
- Caveats:
  - Fast-math mode – best FP performance
  - No FORTRAN benchmarks – no FORTRAN frontend or libraries available
  - Initially comparison versus GCC 4.8, 4.9
    - Later, rolling comparison, trunk vs. trunk
  - Analysis done on Cortex-A53 and Cortex-A57, highlight results on Cortex-A57 results

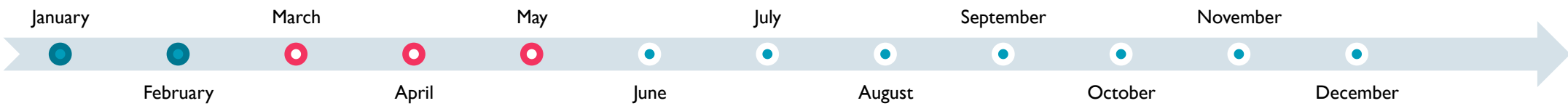


Platform	ARM Juno @ 1.1GHz		
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	Trunk r202557
GCC Flags	`-O3 -ffast-math -mcpu=cortex-a57 -ftree-vectorize`	GCC revision	FSF Trunk r210918

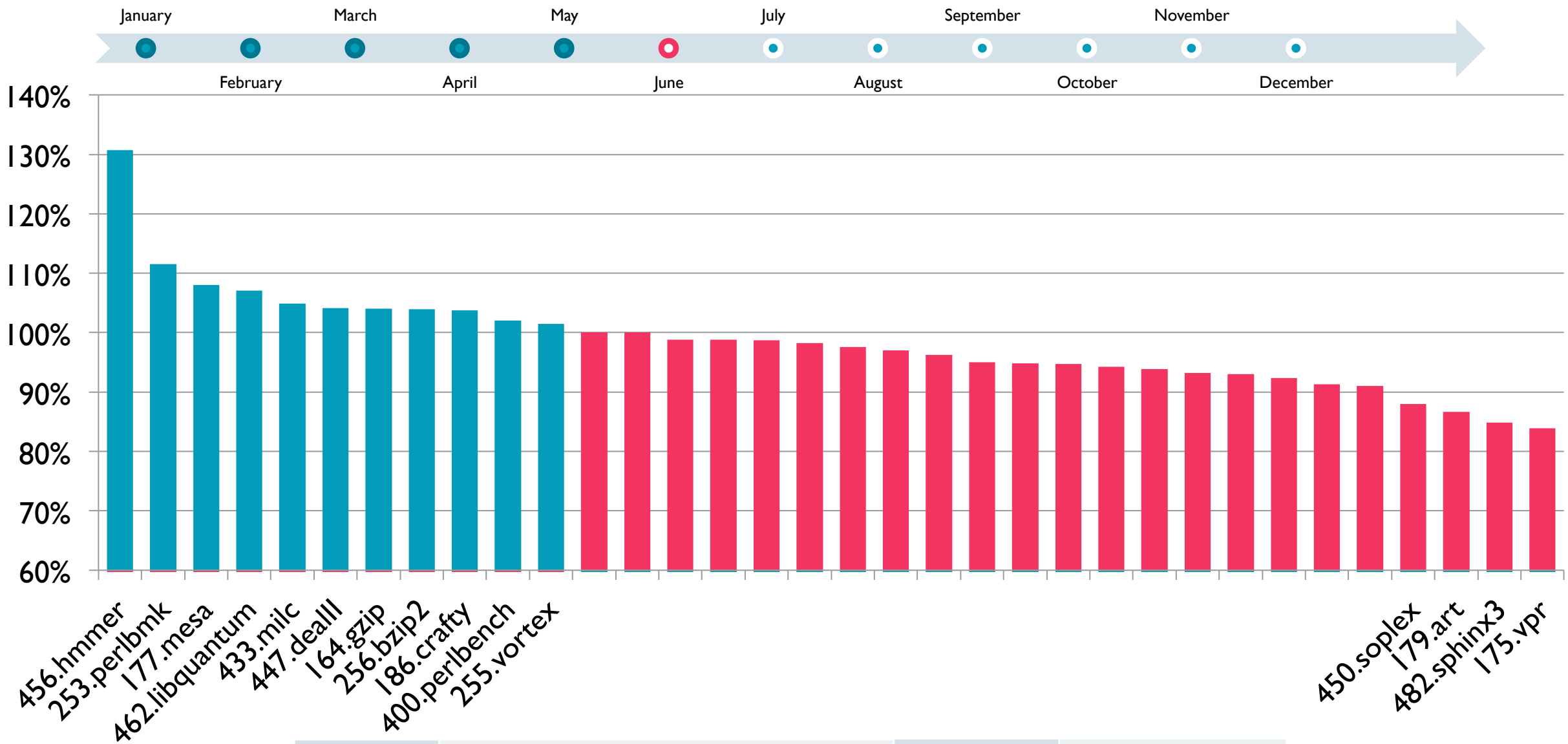
QUALCOMM INNOVATION CENTER, INC.
QuIC

ARM

ARM64




```
------------------------------------------------------------------------
r205090 | tnorthover | 2014-03-29 10:18:08 +0000 (Sat, 29 Mar 2014)

ARM64: initial backend import

This adds a second implementation of the AArch64 architecture to LLVM,
accessible in parallel via the "arm64" triple. The plan over the
coming weeks & months is to merge the two into a single backend,
during which time thorough code review should naturally occur.

Everything will be easier with the target in-tree though, hence this
commit.
------------------------------------------------------------------------
```



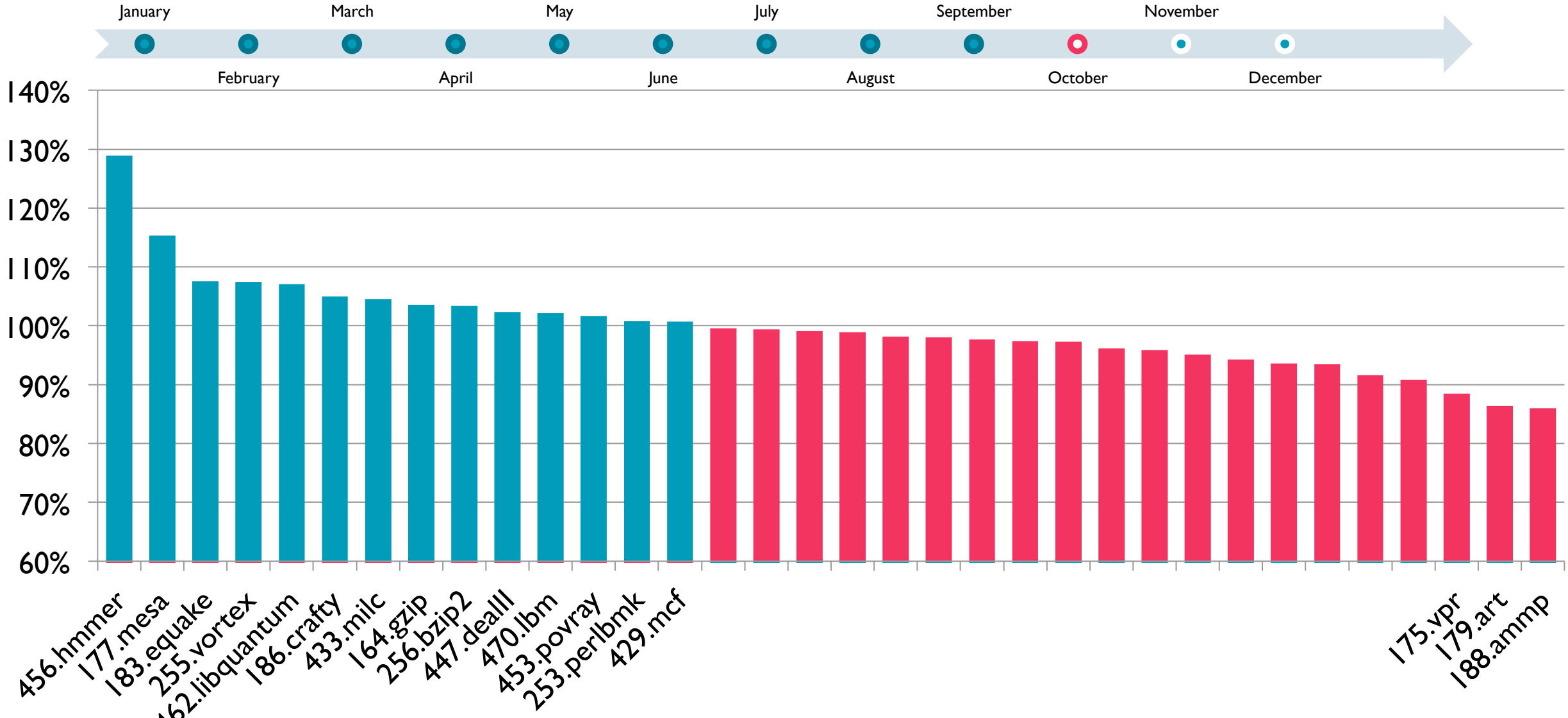
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	Trunk r209577
GCC Flags	`-O3 -ffast-math -mcpu=cortex-a57 -ftree-vectorize`	GCC revision	FSF Trunk r210918

QUALCOMM INNOVATION CENTER, INC. QuIC

ARM

Problems fixed

- Upped maximum interleave factor from 2x to 4x
  - Teach unroller that inner loops are riskier to unroll
- Swapped order of the SLP and Loop vectorizers
  - Don't let SLP mess up a loop for the Loop vectorizer!
- Implement fsub reductions in Loop vectorizer
- Improved floating point reassociation
  - Enabled reassociation in fast-math mode
- Reduced sign/zero extension and truncation operations.
  - Fixes in different areas (Legalize, IndVarSimp, etc.) improved CSE effectiveness.
- Added machine schedule models for Cortex-A53 and Cortex-A57 and tuned the models
- Wrote a pass to statically schedule FMADD/FMUL instructions – Cortex- A57 specific
- And more!



LLVM Flags	-O3 -ffast-math -mcpu=cortex-a57	LLVM revision	Trunk r218131
GCC Flags	-O3 -ffast-math -mcpu=cortex-a57 -ftree-vectorize	GCC revision	FSF Trunk r215403

QUALCOMM INNOVATION CENTER, INC.
QuIC

ARM

Induction variable selection

```
void test_fun(int *b, int **c) {
  int i;
  for (i = 0; i < 100; i++)
    c[i] = &b[i];
}
```

- Poor choice of induction variable
- add cannot be folded into `str`
- Applicable to POWER (`stux`) too

- Patch in progress

```
test_fun:
        mov     x8, xzr
.LBB0_1:
        str     x0, [x1, x8]
        add     x8, x8, #8
        add     x0, x0, #4
        cmp      x8, #800
        b.ne    .LBB0_1

        ret
```

} `str x0, [x1], x8`

Addressing mode selection

```
struct s { int x, y, z; };

int f(struct s *b, int *c) {
  int a = 0, d;
  while (d = *c++) {
    if (d > 5)
      a += b[d].y;
    a += b[d].z;
  }
  return a;
}
```

```
if.then:
  %y    = getelementptr %struct.s* %b, i64 %idxprom, i32 1
  %2    = load i32* %y
  %add  = add nsw i32 %2, %a.011
  br label %if.end

if.end:
  %a.1  = phi i32 [ %add, %if.then ], [ %a.011, %while.body ]
  %z    = getelementptr %struct.s* %b, i64 %idxprom, i32 2
  %3    = load i32* %z, align 4
  %add3 = add nsw i32 %3, %a.1
  %4    = load i32* %incdec.ptr12
  %bool = icmp eq i32 %4, 0
  br i1 %bool, label %while.end.loopexit, label %while.body
```

Addressing mode selection

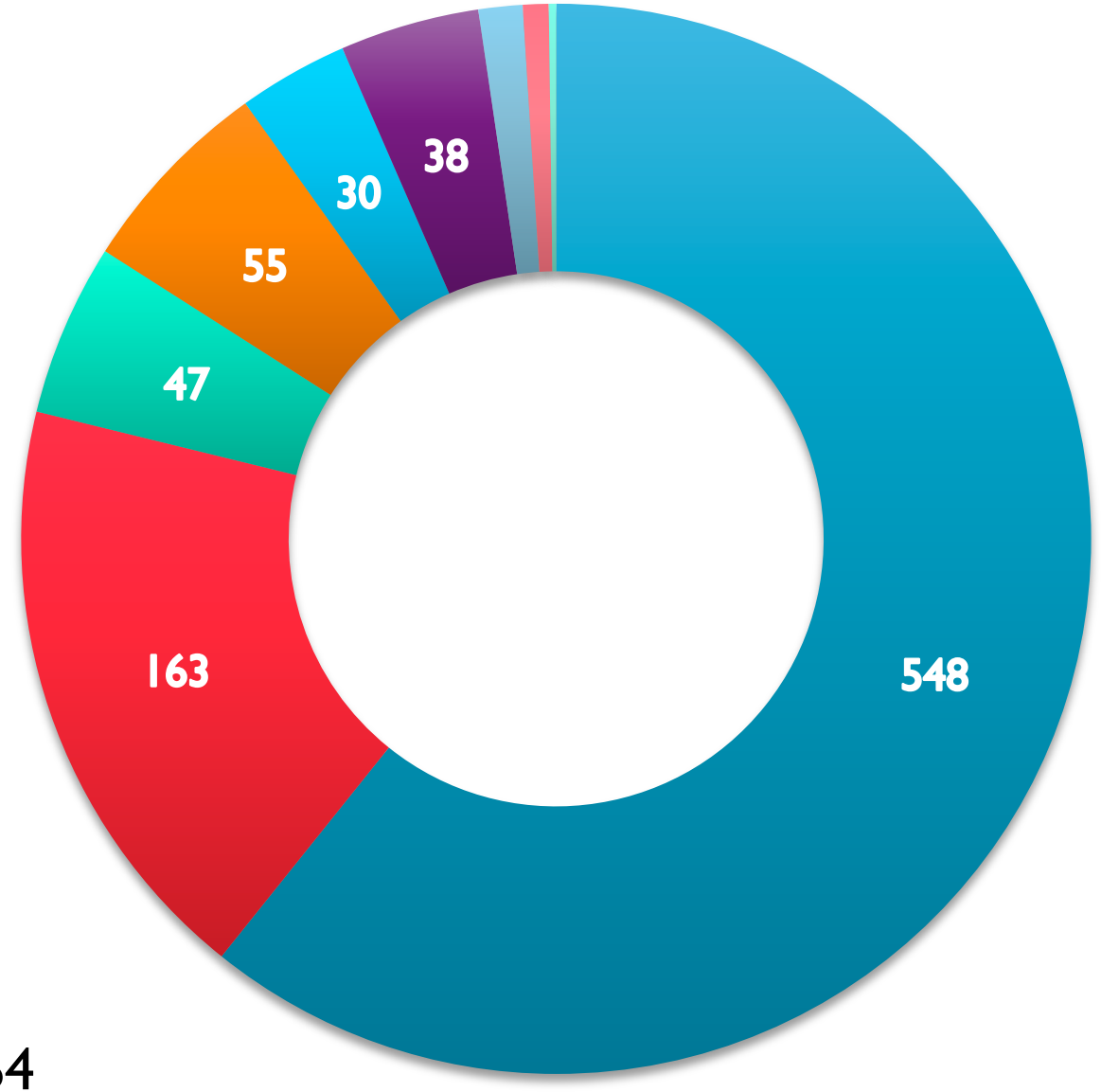
```
struct s { int x, y, z; };

int f(struct s *b, int *c) {
  int a = 0, d;
  while (d = *c++) {
    if (d > 5)
      a += b[d].y;
    a += b[d].z;
  }
  return a;
}
```

- Patch submitted (by Hao Liu)

```
.LBB0_2:
        ldrsh   x11, [x9]
        cmp     x11, #6
        b.lt    .LBB0_4

        madd    x12, x11, x10, x0
        ldr     w12, [x12, #4]
        add     w8, w12, w8
.LBB0_4:
        madd    x12, x11, x10, x0
        ldr     w12, [x12, #8]
        add     w8, w12, w8
        add     x9, x9, #4
        cbnz    w11, .LBB0_2
```

Vectorization

- Vectorized
- No information
- Not beneficial to vectorize
- Cannot identify array bounds
- Could not determine number of loop iterations
- Unsafe dependent memory operations in loop
- Cannot check memory dependencies at runtime
- Value used outside loop
- Control flow cannot be substituted for select



- Comparison versus GCC 4.9 for AArch64

Inlining

- GCC versus LLVM performance analysis reveals the LLVM inliner
  - Does not inline certain hot functions unless a high threshold is provided at –O3.
  - Produces larger and slower code at –Os.
- Identified use cases that should be considered in the inlining strategy.
- About the LLVM inliner
  - Traverses call graph in SCC order (i.e., bottom-up order).
  - Supports a deferred bottom-up inlining mode.
  - Cannot be modified to achieve a desired order of processing call sites due to its pass setup.

Inlining: Primary Use Case

- Use Case 1: A calls B calls C

```
A() { // Use Case 1
  call B(p1, p2, p3, p4, p5, p6)
}

B(p1, p2, p3, p4, p5, p6) {
  call C()
}
```

  - A bottom-up inliner always tries to inline C into B first.
  - But if C is inlined into B, B may be too big to be inlined into A.
  - There are cases it is more profitable to inline B into A.
  - LLVM inliner's solution: deferred bottom up inlining mode.
  - Desired behavior: Allow the inliner to decide which call site will be processed first.

Inlining: Other Use Cases

- Use Case 2
  - Desired behavior: Favor inlining call sites in loops.

```
A() { // Use Case 2
   call B()
   call C()
   call D()
   for (…) {
      call F()
   }
}
```

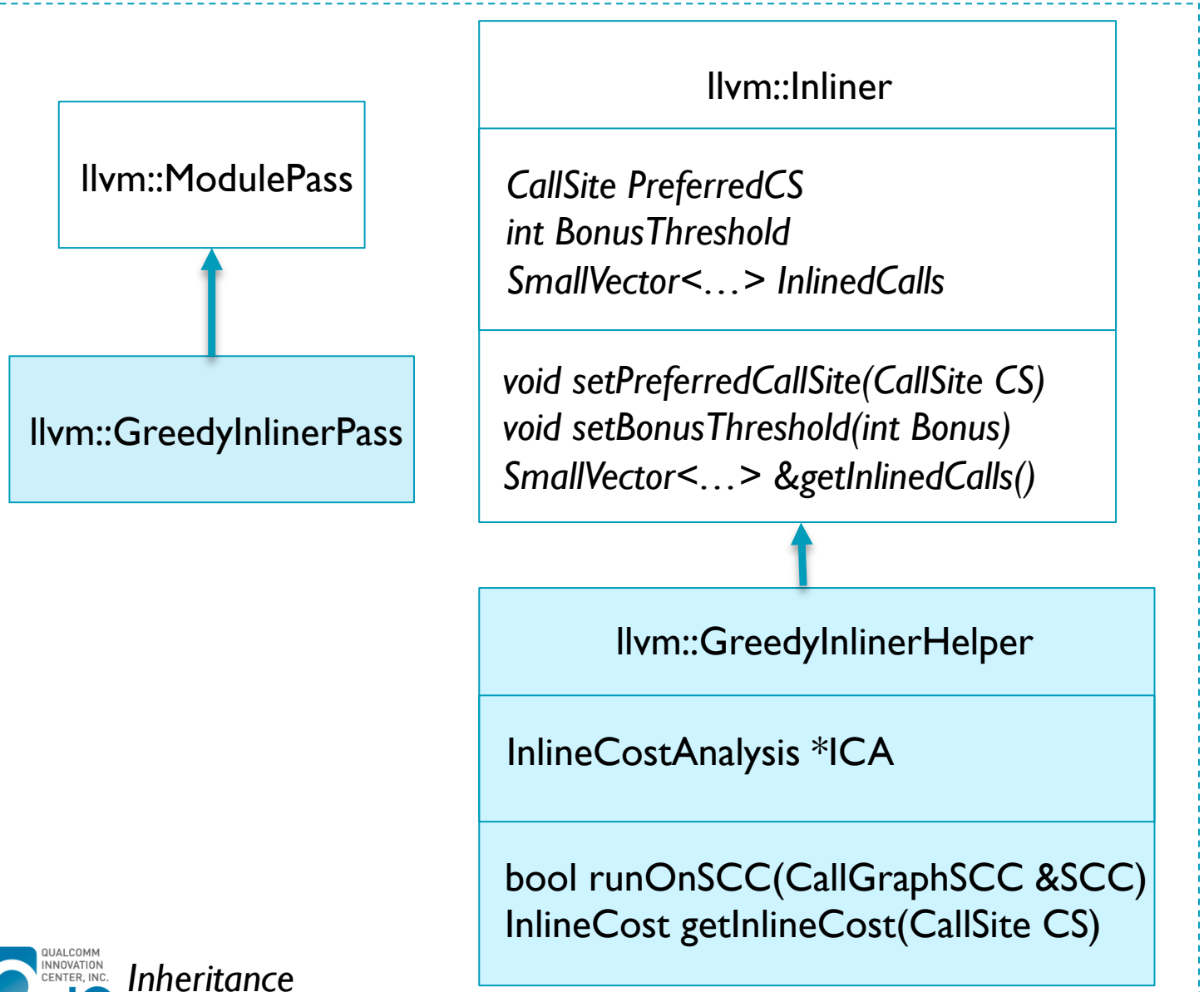
- Use Case 3
  - Desired behavior: Favor inlining call sites at root level which are more likely to be in the critical path.

```
A() { // Use Case 3
   call B()
   call C()
   call D()
   if (…)
      if (…)
         call F()
}
```

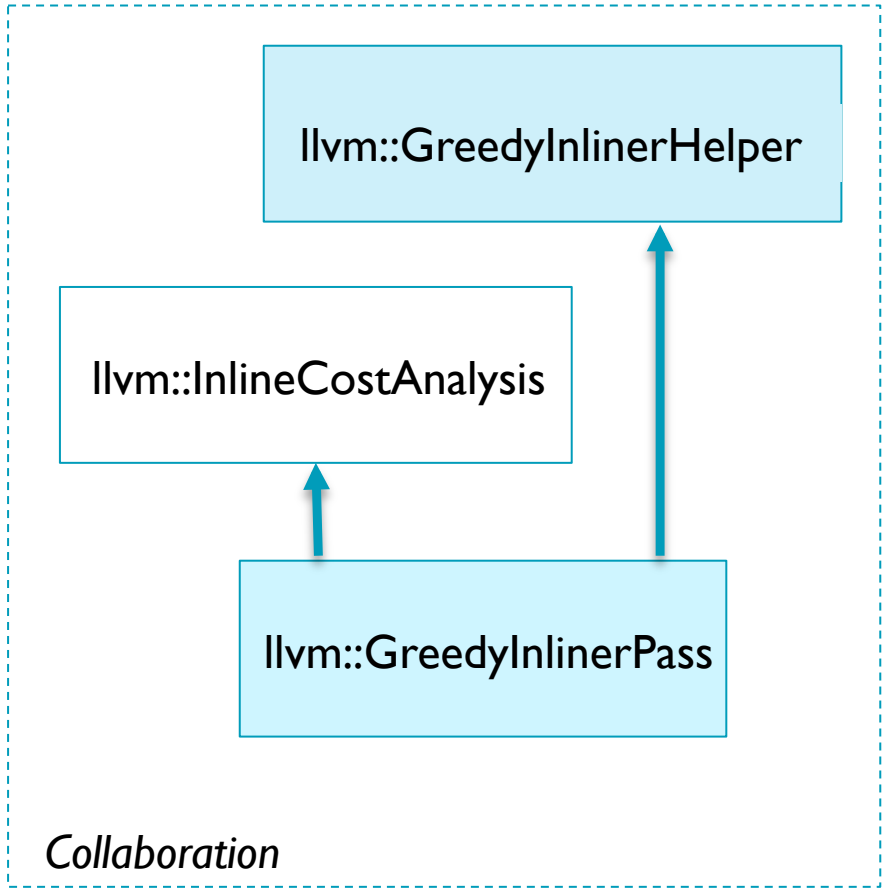
Inlining: Greedy Inliner Approach

- A module pass that builds upon the LLVM inliner and uses a different call site processing order.
  - LLVM inliner does the local decision and actual inlining work.
  - LLVM inliner special tunings are preserved.
- Uses a priority queue of call sites with computed weights.
  - The weight is computed based on size, use count, loop depth, branch level etc.
- Threshold for a call site can be further tuned with bonus policy to catch use cases.
- Patch with initial tuning for ARMv7 target up-streamed for code review and feedback. Experiments on AArch64 on going and indicate heuristics need tuning.
- Discussion to be continued at this year's BOF on "LLVM Inliner Improvements".

Inlining: Greedy Inliner Inheritance and Collaboration Diagrams

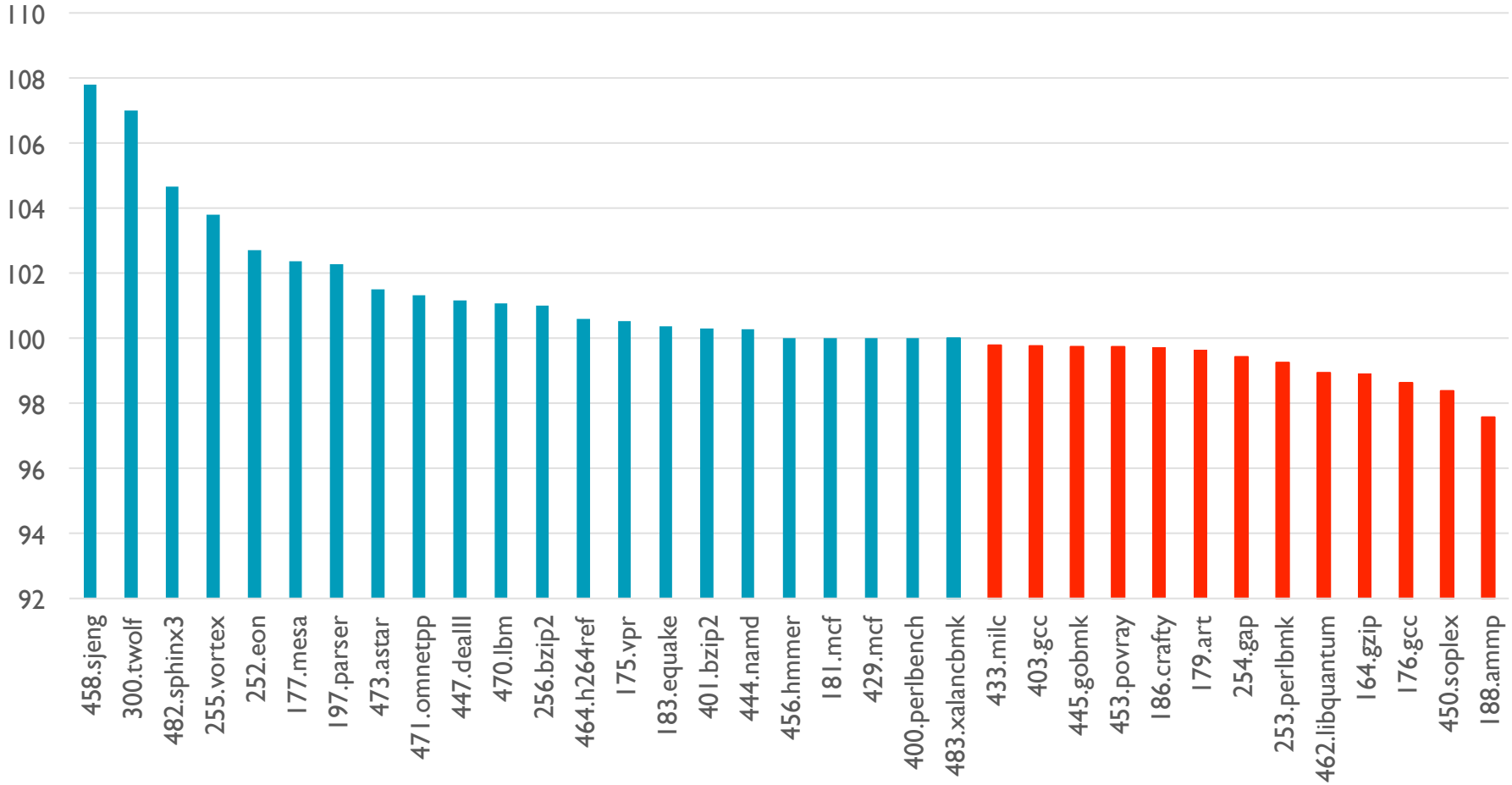


Inheritance



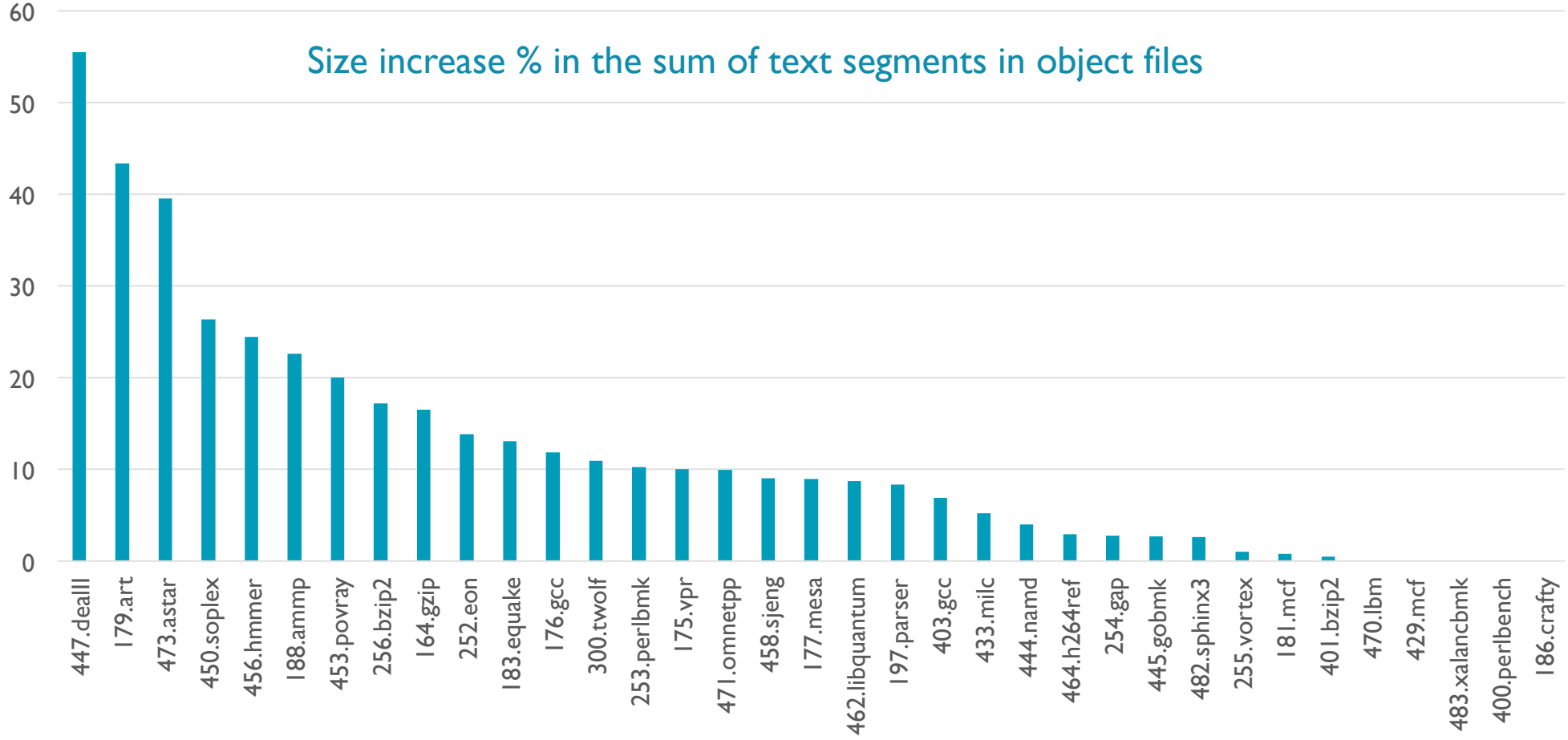
Collaboration

Inlining: Greedy Inliner Speedup (-O3) on ARMv7



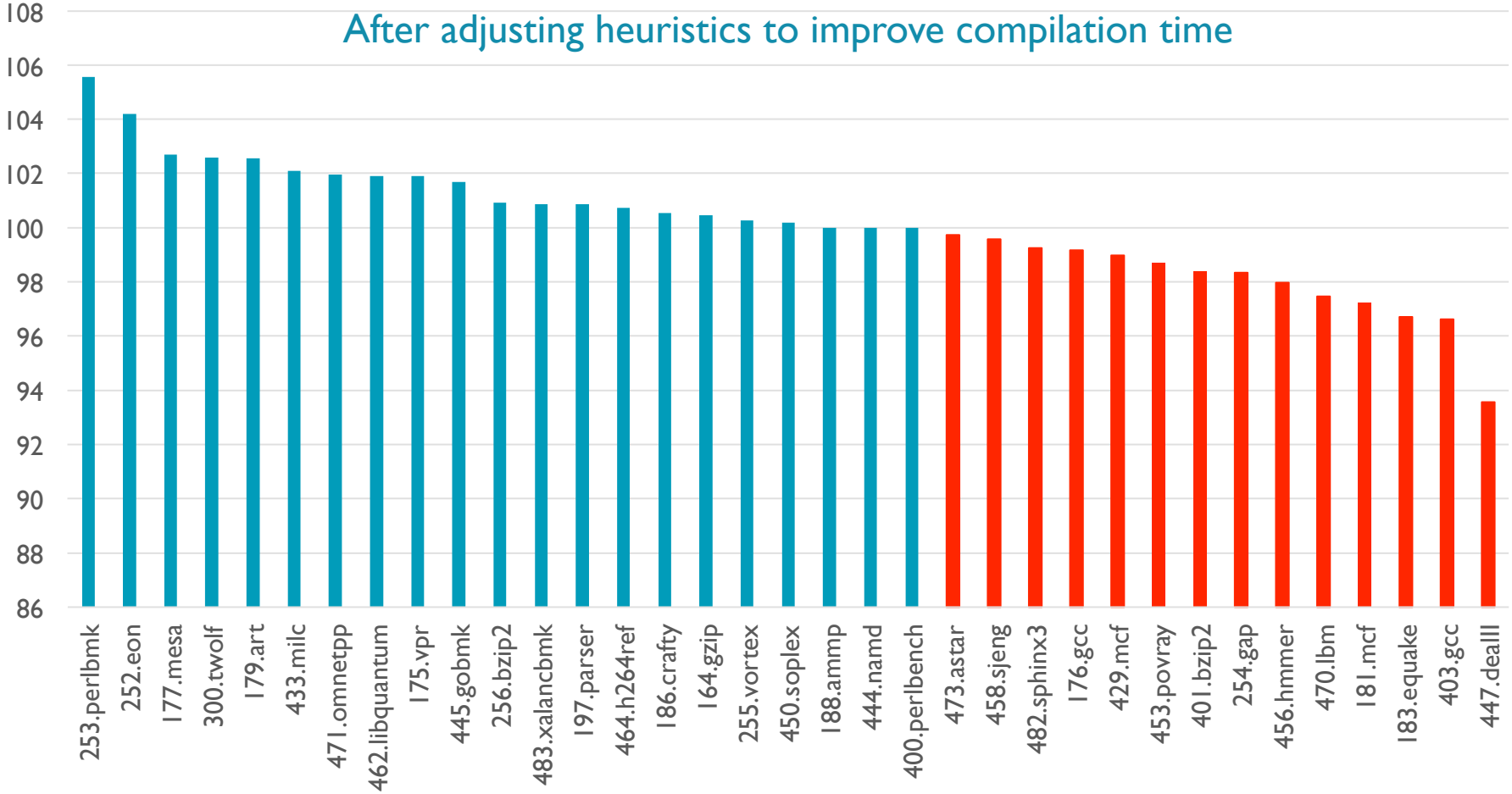
Platform	Nexus 4 device		
LLVM baseline Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	internal branch
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57 -mllvm -greedy-inliner=true`		

Inlining: Greedy Inliner Size Increase (-O3) on ARMv7



Platform	Nexus 4 device		
LLVM baseline Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	internal branch
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57 -mllvm -greedy-inliner=true`		

Inlining: Greedy Inliner Speedup (-O3) on AArch64

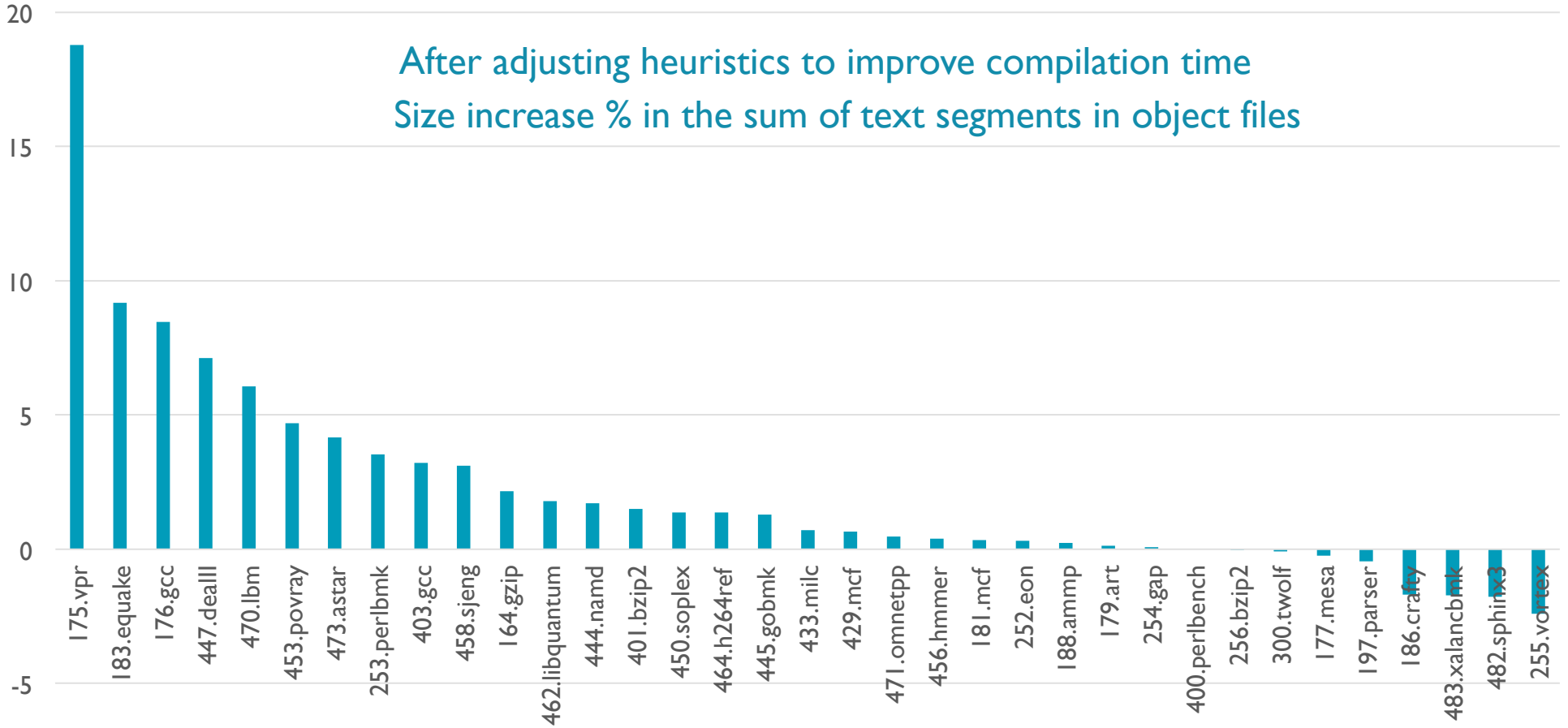


Platform	Qualcomm cortex-a57 core		
LLVM baseline Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	Trunk r218131
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57 -mllvm -greedy-inliner=true`		

QUALCOMM INNOVATION CENTER, INC. QuIC

ARM

Inlining: Greedy Inliner Size Increase (-O3) on AArch64



Platform	Qualcomm cortex-a57 core		
LLVM baseline Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	Trunk r218131
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57 -mllvm -greedy-inliner=true`		

QUALCOMM INNOVATION CENTER, INC. QuIC

ARM

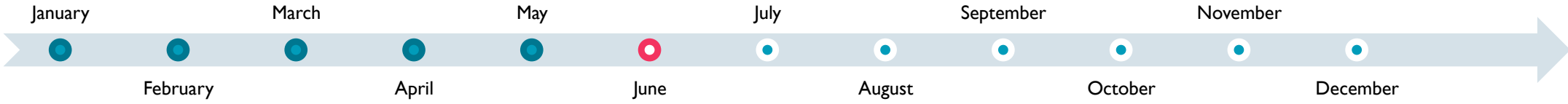
Future Work

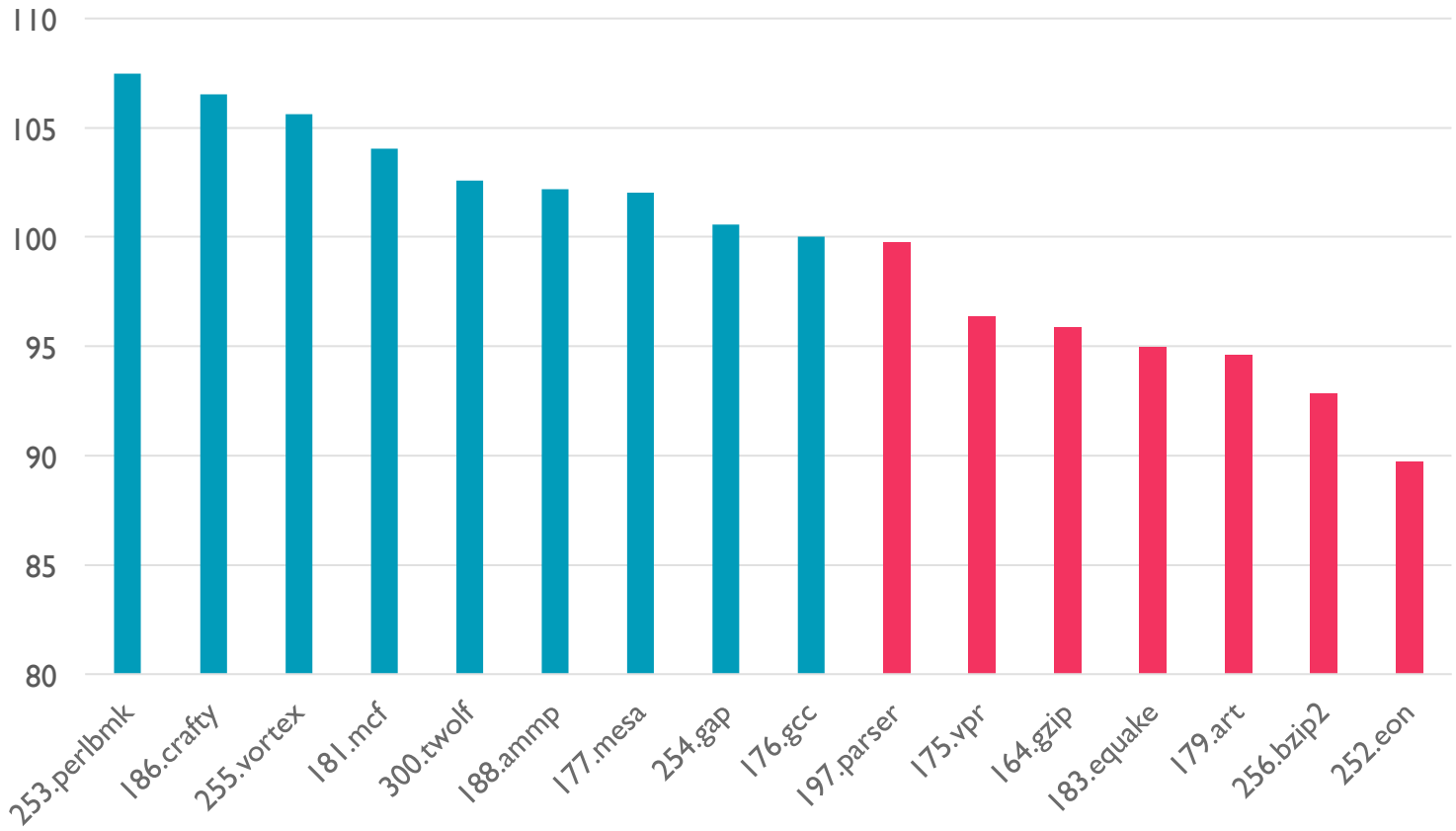
- BOF discussion on LLVM inliner to set goals and how to achieve them.
- Detected some issues that can be resolved with alias analysis improvements.
  - Remove redundant load, e.g. PR20074.
  - Hoist/sink loads/stores out of loops, e.g., PR20585 and PR21229.
  - Will LLVM's strict aliasing rules allow aggressive optimizations like in GCC?
- Continue performance analysis
  - Enabling other optimizations for high performance, e.g., LTO, PGO.
  - Diversifying workload.
- How to raise geomean even higher? Thoughts? Come see us!

Conclusions

- Example of productive cooperation among ARM, QuIC, Apple, LLVM and Clang community.
- Performance is very important for LLVM AArch64 compiler to be competitive.
- Acknowledgments
  - ARM Ltd.: Jiangning Liu, Hao Liu, Kevin Qin.
  - QuIC Inc.: Dave Estes, Yin Ma, Balaram Makam, Chad Rosier, Sanjin Sijaric, Weiming Zhao, Zhaoshi Zheng.
  - Apple: Tim Northover, Andy Trick
  - LLVM and Clang community reviewers.

Back-up Slides

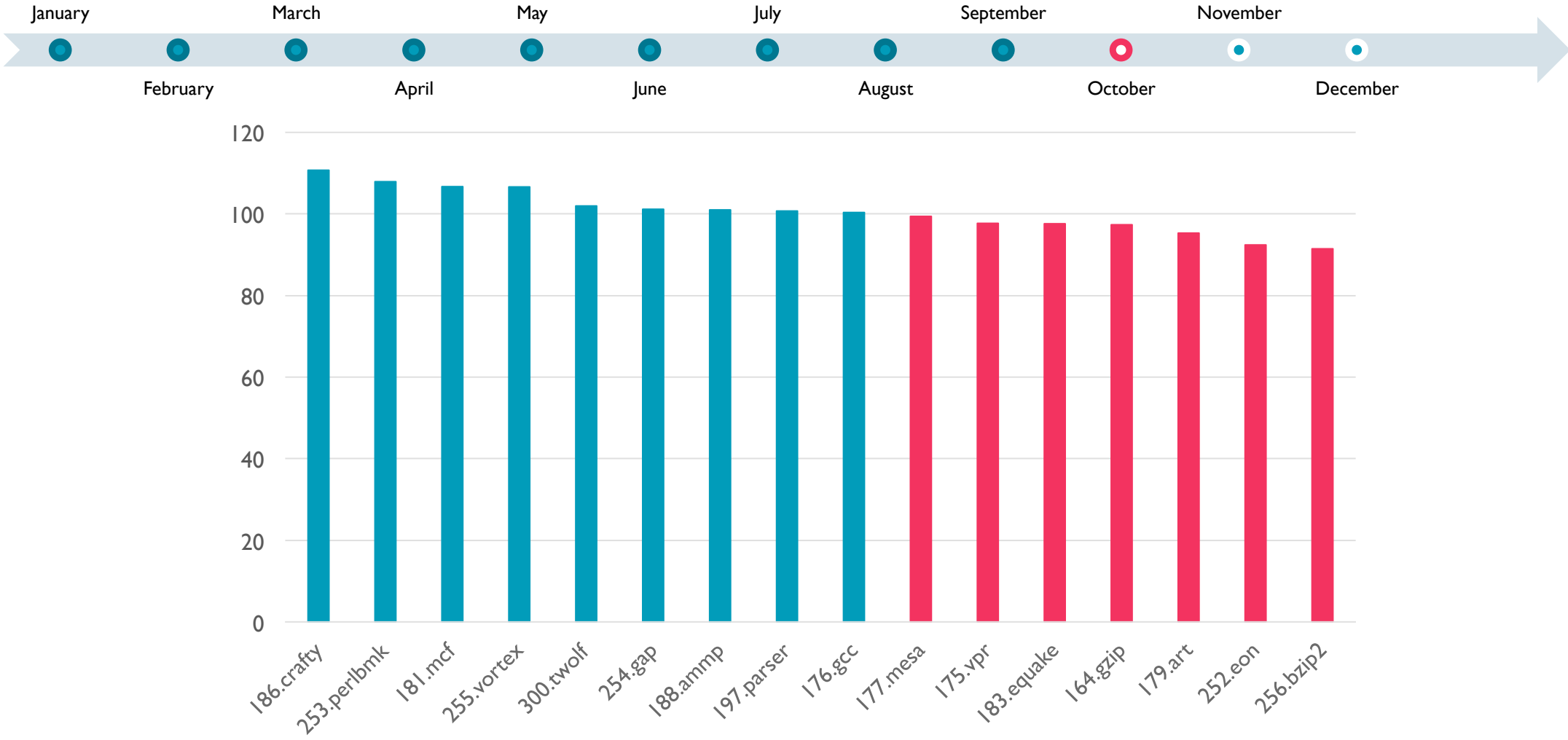




Platform	Qualcomm cortex-a53 core		
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	Trunk r209577
GCC Flags	`-O3 -ffast-math -mcpu=cortex-a57 -ftree-vectorize`	GCC revision	4.9

QUALCOMM INNOVATION CENTER, INC. QuIC

ARM



Platform	Qualcomm cortex-a53 core		
LLVM Flags	`-O3 -ffast-math -mcpu=cortex-a57`	LLVM revision	Trunk r218131
GCC Flags	`-O3 -ffast-math -mcpu=cortex-a57 -ftree-vectorize`	GCC revision	4.9

QUALCOMM INNOVATION CENTER, INC. QuIC

ARM

Problems fixed

- Removed a single redundant load…
  - PRLE resolves the issue but it is slow; Improve GVN? Down our priority list!
- Reduced spilling from Q registers
  - 128-bit Q registers are not callee-saved and this cost needs to be taken into account in optimizations.
- Loop unroller
  - Use a loop to simplify the runtime unrolling prologue.
- Improved rematerialization
  - Identified arithmetic and logical instructions that are as cheap as move instructions on AArch64.
- DAG transformations to allow more efficient machine idioms to be generated.
  - Generate TBZ, TBNZ, CMN, CINC, UBFX; lower SDIV by power of 2 using ADD+SELECT+SHIFT; convert MUL by (power of 2 +-1) to SHIFT+ADD/SUB.

Problems fixed

- Disabled conditional select instruction generation for predicted branches on A57.
- MI scheduler: enabled Post-RA and enable/improved AA during machine scheduling
- Machine model for A57 details
  - Modeled instruction latency, micro-op details, forwarding for MAC instructions and hazards for SQRT/DIV instructions.
  - Experimented with how to model the compiler look-ahead capability
    - Issue width reduced to 3 so that the scheduler can better accommodate the narrower decode and dispatch width.

Inlining: Greedy Inliner Main Algorithm

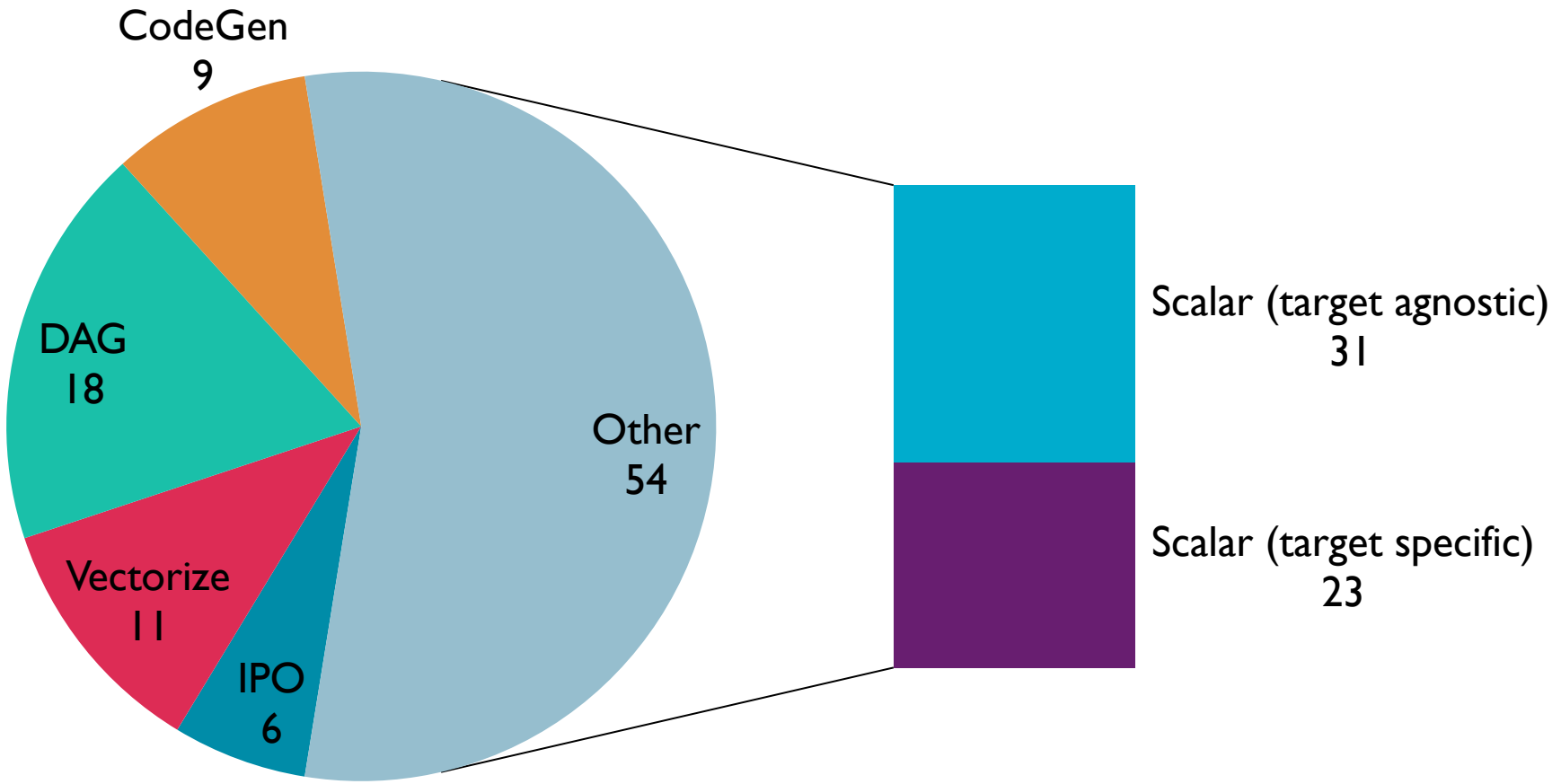
```
for each function in Module
   CallSites += collectFunctionCallSites()
computeCallSitesWeight(CallSites)

FuncInliner = createGreedyInlinerHelperPass()

do
   CS = getBestCallSite(CallSites)
   BonusThreshold = ComputeBonusThreshold(CS)

   setBonusThreshold(FuncInliner, BonusThreshold)
   setPreferredCallSite(FuncInliner, CS)
   Change = run(FuncInliner)

   if no Change continue

   CallSites += getInlinedCalls(FuncInliner)
   computeCallSitesWeight(CallSites)
while CallSites not empty
```

Inlining: Greedy Inliner Call Site Weight Computation

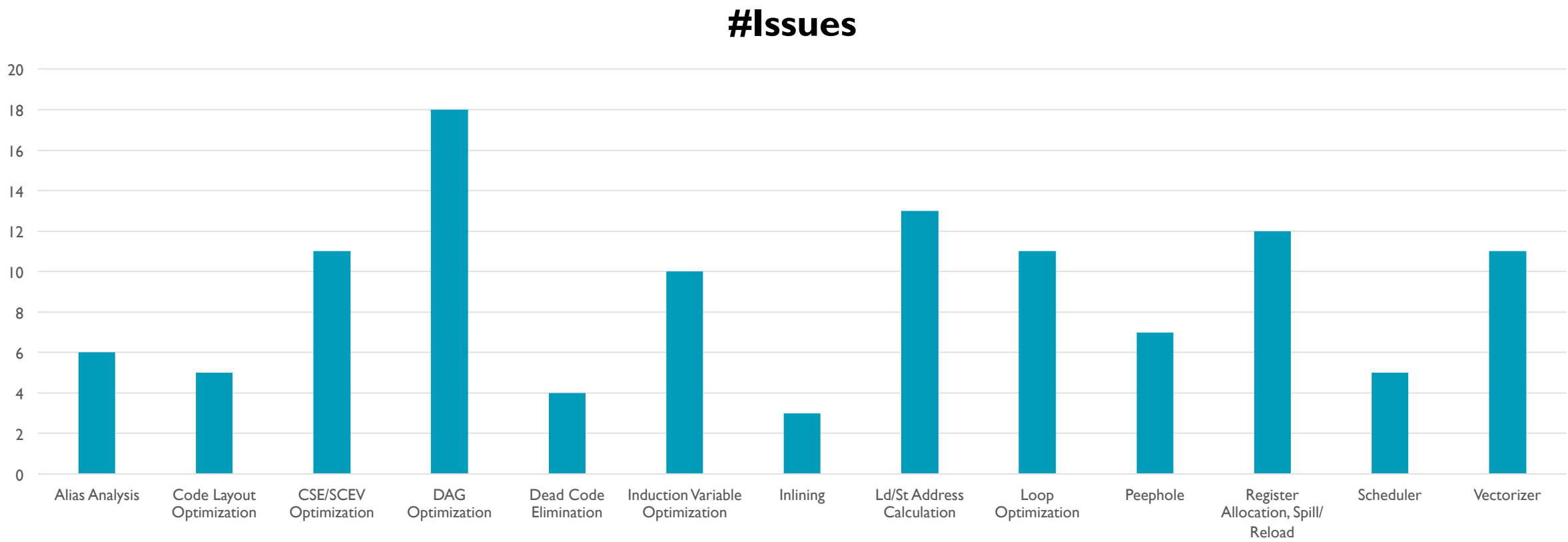
- B - Benefit Point if inlining (larger is better, 0 is no special benefit)
  - Catch Special Need
- L - Loop depth of this call site (larger is better)
- S - Size of the callee (smaller is better)
  - Based on instruction count and basic block count
- U - Use Bonus Factor, initialized to 1
  - Call site with one or two uses get some bonus.
- BL - Branch Level
  - Call site in branch will have lower priority in a function.
- C – How many calls to this callee.
- S – Scale up to make threshold works better
- Weight = B * L * U * S / (C * SQRT(S) * BL)

Performance Analysis Details

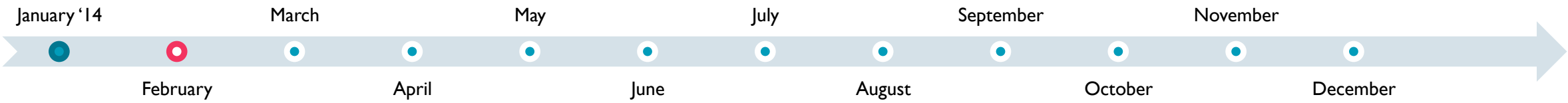
- ~100 issues found in several compiler areas.



Performance Analysis Details

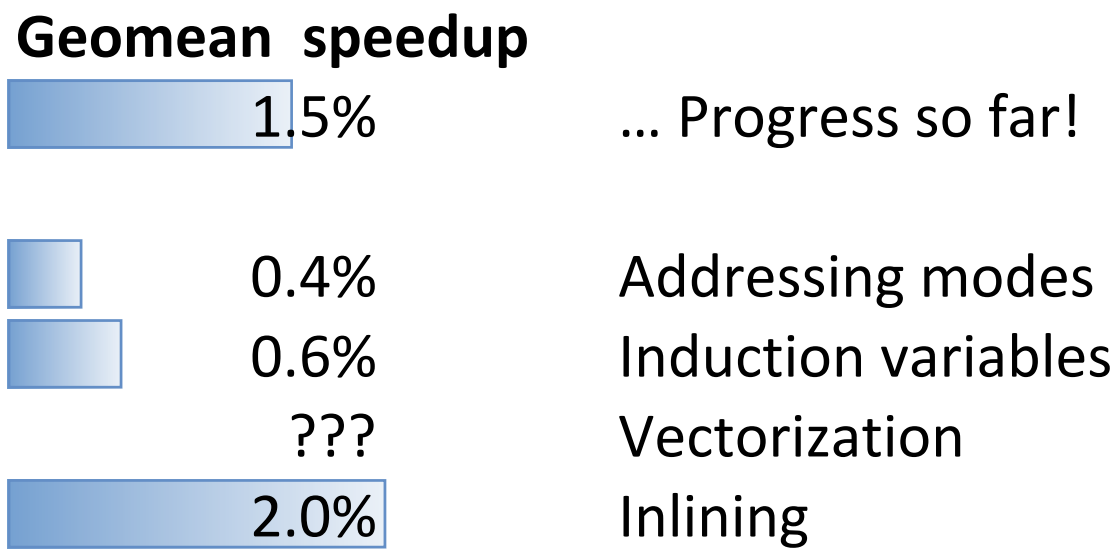


Methodology



- Benchmarks as a proxy for performance
- Standard set of benchmarks
  - SPEC2000, SPEC2006
  - EEMBC
  - Geekbench
  - Dhrystone
  - Coremark
- First target: SPEC (INT+FP)

Current work



Addressing mode selection

0.4%

- Complex addressing mode calculation
- Represented as GEPs
- Calculation not split up before ISel

```
.LBB0_2:
        ldrsh   x11, [x9]
        cmp     x11, #6
        b.lt    .LBB0_4

        madd    x12, x11, x10, x0
        ldr     w12, [x12, #4]
        add     w8, w12, w8
.LBB0_4:
        madd    x12, x12, x10, x0
        ldr     w12, [x12, #8]
        add     w8, w12, w8
        add     x9, x9, #4
        cbnz    w11, .LBB0_2
```

- Patch submitted (by Hao Liu)