Benchmarking LLVM Using Embench:

What does it tell us about the compiler?

Jeremy Bennett

Embench 0.5

Benchmarking IoT Class Devices

History

Whetstone	Linpack	Dhrystone	1988 – 2018 30 YEARS spec®	CoreMark® An EEMBC Benchmark	MLPerf
1972	1977	1984	1989	2009	2018

History

Whetstone	Linpack	Dhrystone	spec (30 years, 1988 – 2018)	CoreMark® An EEMBC Benchmark	MLPerf
1972	1977	1984	1989	2009	2018

EMBC® EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM	MiBench	BEEBS	TACLe Timing Analysis on Code-Level
1997	2001	2013	2016

7 Lessons for Embench

1. Embench must be free
2. Embench must be easy to port and run
3. Embench must be a suite of *real* programs
4. Embench must have a supporting organization to maintain it
5. Embench must report a single summarizing score
6. Embench should summarize using geo mean and std. dev.
7. Embench must involve both academia and industry

The Plan

- **Jan - Jun 2019:** Small group created the initial version
  - Dave Patterson, Jeremy Bennett, Palmer Dabbelt, Cesare Garlati
  - mostly face-to-face
- **Jun 2019 – Feb 2020:** Wider group open to all
  - under FOSSi, with mailing list and monthly conference call
  - see www.embench.org
- **Feb 2020:** Launch at Embedded World

Current Status

- Set of 19 benchmarks for deeply embedded compute
  - up to 64KB ROM and 64kB RAM
  - need BlueTooth LE and ECDSA programs for competeness
- Early benchmark for context switching in RISC-V
  - also needs benchmark for interrupt latency
- Initial python build and benchmark scripts
  - so far mostly tested with simulators
  - so far mostly tested with RISC-V
- Need to widen to real hardware and other architectures

EMBECOSM®

Baseline Data

Name	*Comments*	*Orig Source*	*C LOC*	*code size*	*data size*	*time (ms)*	*branch*	*memory*	*compute*
aha-mont64	Montgomery multiplication	AHA	162	1,052	0	4,000	low	low	high
crc32	CRC error checking 32b	MiBench	101	230	1,024	4,013	high	med	low
cubic	Cubic root solver	MiBench	125	2,472	0	4,140	low	med	med
edn	More general filter	WCET	285	1,452	1,600	3,984	low	high	med
huffbench	Compress/Decompress	Scott Ladd	309	1,628	1,004	4,109	med	med	med
matmult-int	Integer matrix multiply	WCET	175	420	1,600	4,020	med	med	med
minver	Matrix inversion	WCET	187	1,076	144	4,003	high	low	med
nbody	Satellite N body, large data	CLBG	172	708	640	3,774	med	low	high
nettle-aes	Encrypt/decrypt	Nettle	1,018	2,880	10,566	3,988	med	high	low
nettle-sha256	Crytographic hash	Nettle	349	5,564	536	4,000	low	med	med
nsichneu	Large - Petri net	WCET	2,676	15,042	0	4,001	med	high	low
picojpeg	JPEG	MiBench2	2,182	8,036	1,196	3,748	med	med	high
qrduino	QR codes	Github	936	6,074	1,540	4,210	low	med	med
sglib-combined	Simple Generic Library for C	SGLIB	1,844	2,324	800	4,028	high	high	low
slre	Regex	SLRE	506	2,428	126	3,994	high	med	med
st	Statistics	WCET	117	880	0	4,151	med	low	high
statemate	State machine (car window)	C-LAB	1,301	3,692	64	4,000	high	high	low
ud	LUD composition Int	WCET	95	702	0	4,002	med	low	high
wikisort	Merge sort	Github	866	4,214	3236	4,226	med	med	med

EMBECOSM®

Embench and Clang/LLVM

The Top Level View

What Affects Embench Results?

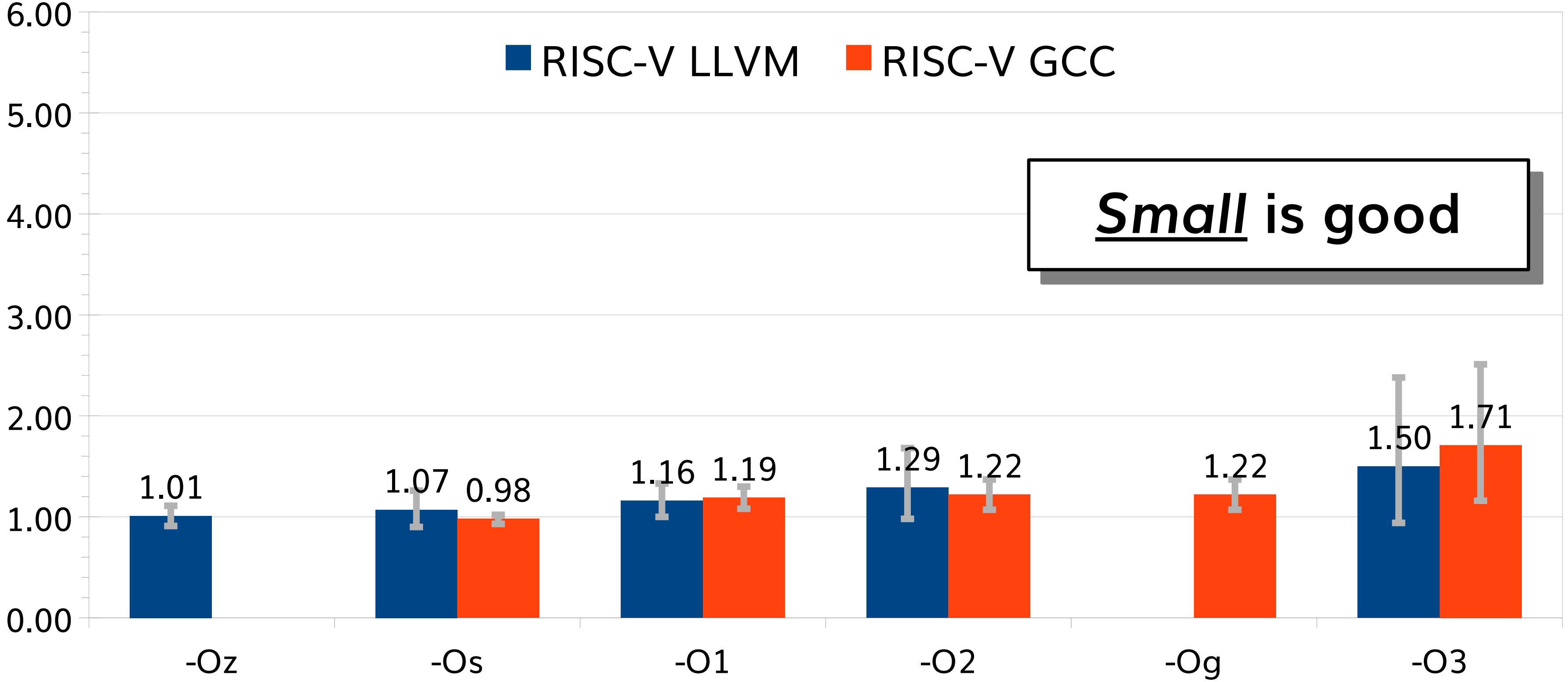
- Instruction Set Architecture: Arm, ARC, RISC-V, AVR, ...
  - extensions: ARM: v7, Thumb2, ..., RV32I, M, C, ...
- Compiler: open (Clang/LLVM, GCC) and proprietary (IAR, ...)
  - which optimizations included: Loop unrolling, inlining procedures, ...
  - older ISAs likely have more mature and better compilers?
- Libraries
  - open (GCC, LLVM) and proprietary (IAR, Sega, ...)
  - Embench excludes libraries when sizing
    - they can swamp code size for embedded benchmarks

EMBECOSM®

Comparison Matrix

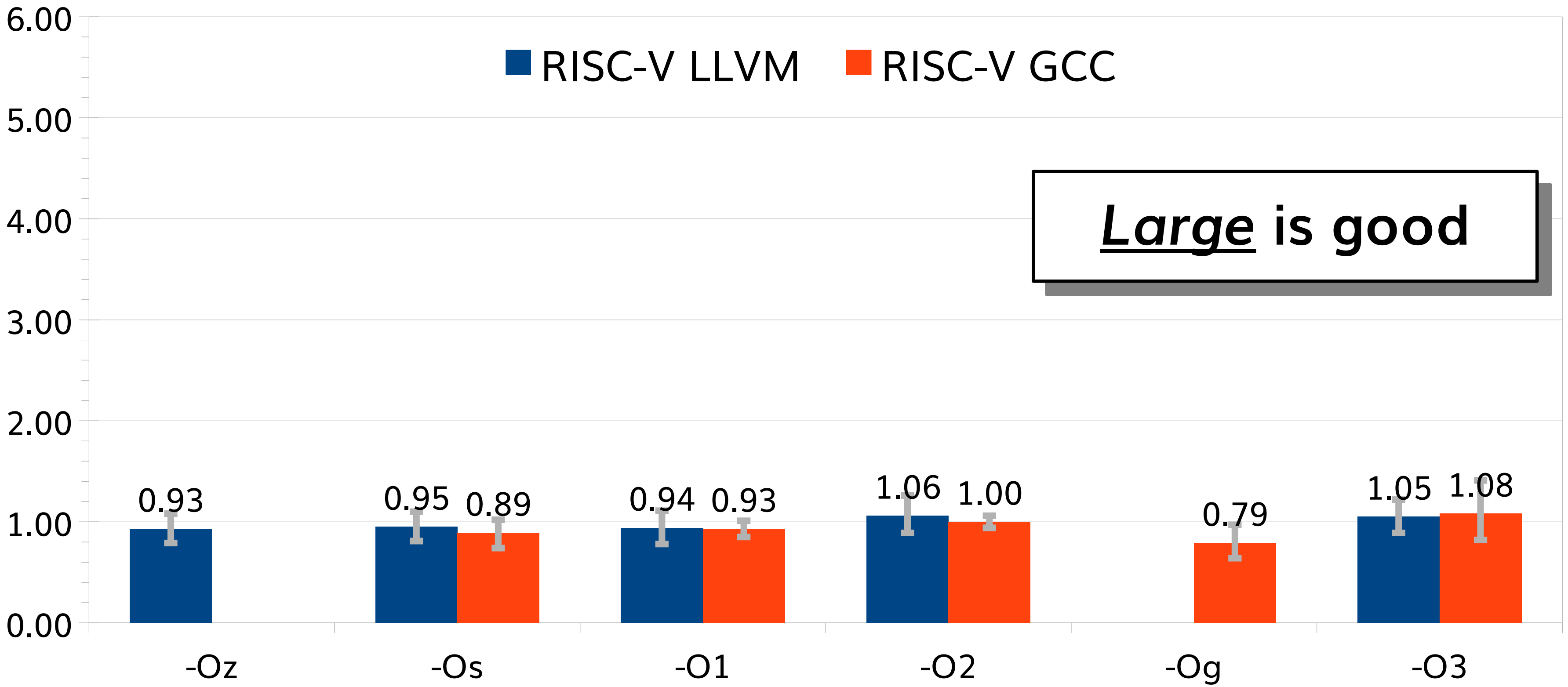
RISC-V	Clang/LLVM RISC-V RV32IMC	GCC RISC-V RV32IMC
arm	Clang/LLVM Arm Cortex M4	GCC Arm Cortex M4

Code Size by Compiler

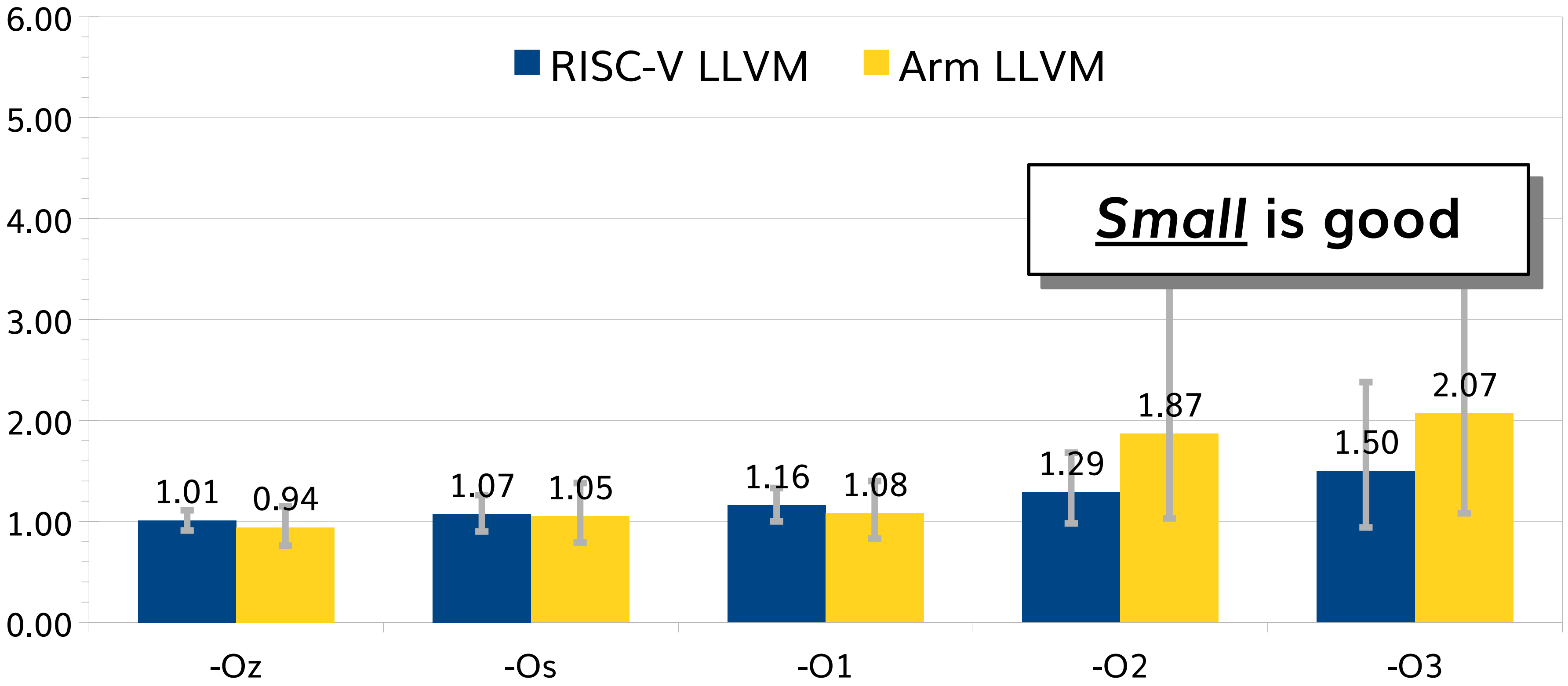


EMBECOSM®

Code Speed by Compiler



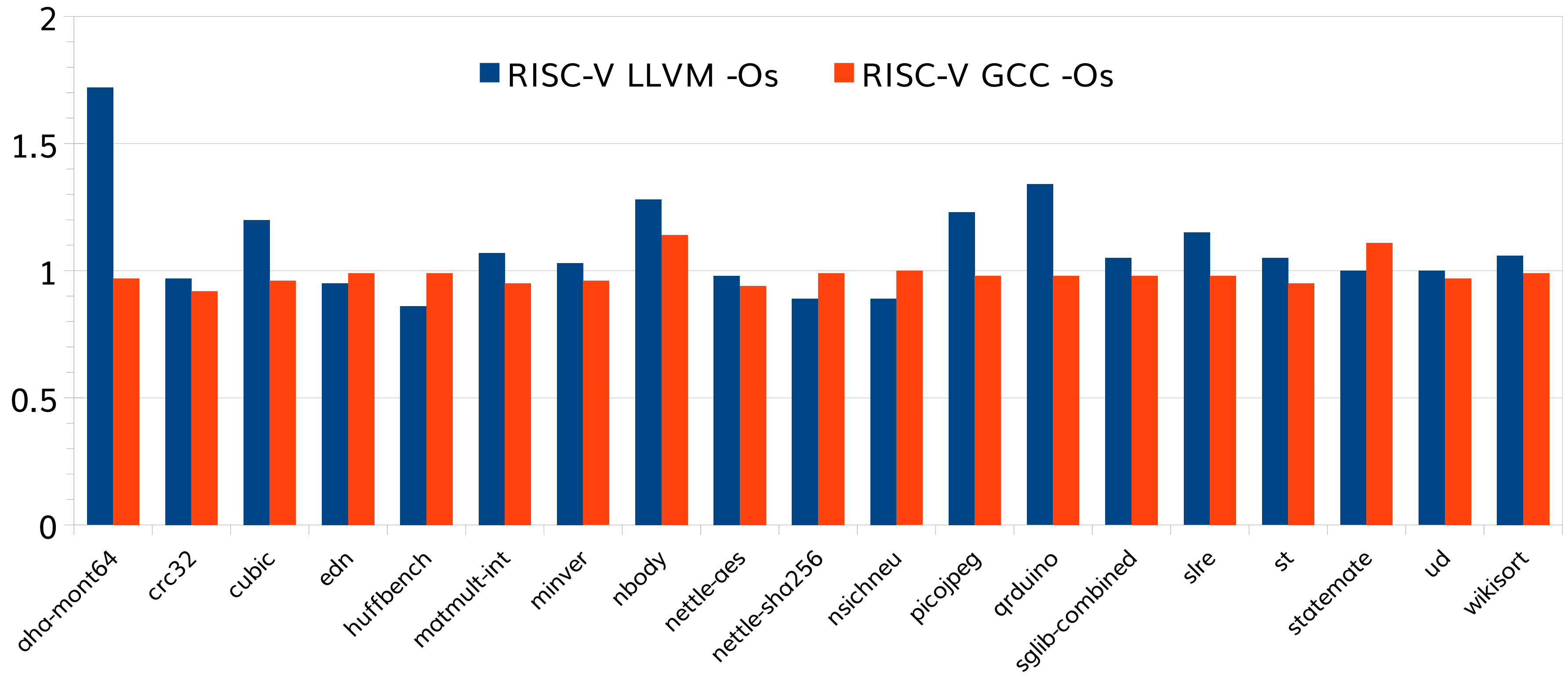
Code Size by Architecture



Embench and Clang/LLVM

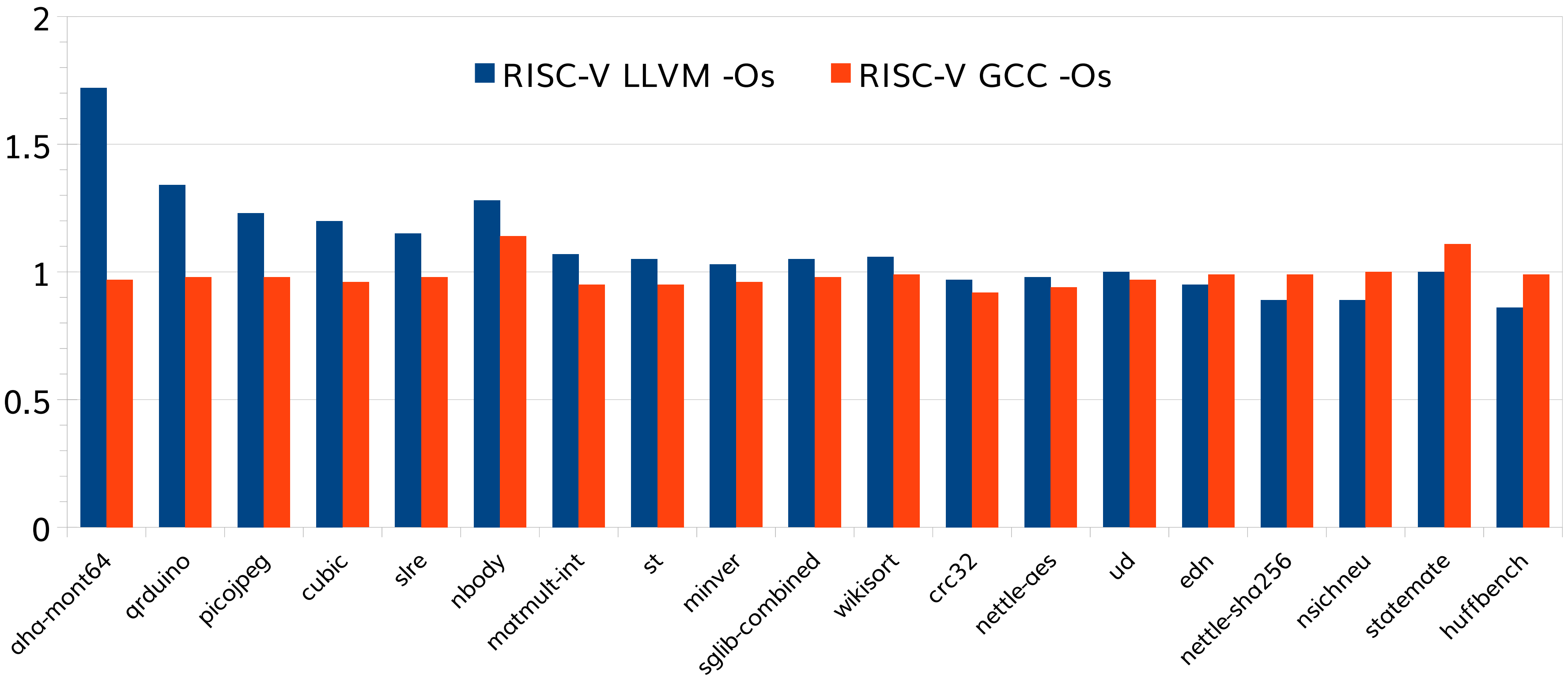
Individual Benchmark Results

LLVM v GCC Code Size with -Os



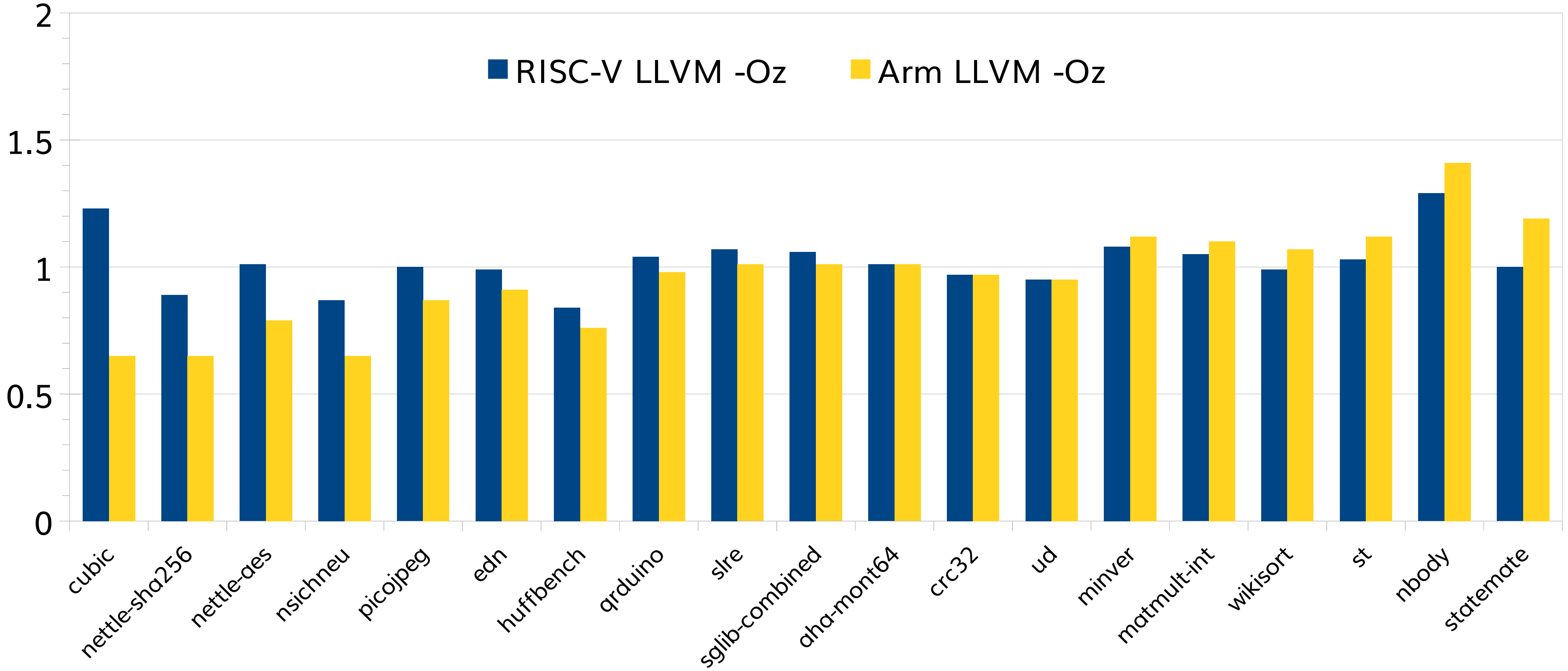
Small **is good**

LLVM v GCC Code Size with -Os (Sorted)



Small **is good**

RISC-V v Arm Code Size with -Oz (Sorted)



Small is good

Embench and Clang/LLVM

Going deep

aha-mont64 -Os

Clang/LLVM

```
$ nm --size-sort aha-mont64
00000002 T warm_caches
00000004 T initialise_board
00000004 T start_trigger
00000004 T stop_trigger
00000006 T benchmark
00000006 T _start
00000006 T verify_benchmark
00000012 T initialise_benchmark
00000022 T main
0000011e T montmul
000005c6 t benchmark_body
```

GCC

```
$ nm --size-sort aha-mont64
00000002 T warm_caches
00000004 T initialise_board
00000004 T start_trigger
00000004 T stop_trigger
00000006 T benchmark
00000006 T _start
00000006 T verify_benchmark
00000022 T main
00000034 T initialise_benchmark
00000052 T mulul64
0000006a T modul64
000000a6 T xbinGCD
000000ee T montmul
0000016e t benchmark_body
```


EMBECOSM®

aha-mont64 -Os

Clang/LLVM

```
$ nm --size-sort aha-mont64
00000002 T warm_caches
00000004 T initialise_board
00000004 T start_trigger
00000004 T stop_trigger
00000006 T benchmark
00000006 T _start
00000006 T verify_benchmark
00000012 T initialise_benchmark
00000022 T main
0000011e T montmul
000005c6 t benchmark_body
```

GCC

```
$ nm --size-sort aha-mont64
00000002 T warm_caches
00000004 T initialise_board
00000004 T start_trigger
00000004 T stop_trigger
00000006 T benchmark
00000006 T _start
00000006 T verify_benchmark
00000022 T main
00000034 T initialise_benchmark
00000052 T mulul64
0000006a T modul64
000000a6 T xbinGCD
000000ee T montmul
0000016e t benchmark_body
```

aha-mont64 -Os

Clang/LLVM

```
$ nm --size-sort aha-mont64
00000002 T warm_caches
00000004 T initialise_board
00000004 T start_trigger
00000004 T stop_trigger
00000006 T benchmark
00000006 T _start
00000006 T verify_benchmark
00000012 T initialise_benchmark
00000022 T main
0000011e T montmul
000005c6 t benchmark_body
```

GCC

```
$ nm --size-sort aha-mont64
00000002 T warm_caches
00000004 T initialise_board
00000004 T start_trigger
00000004 T stop_trigger
00000006 T benchmark
00000006 T _start
00000006 T verify_benchmark
00000022 T main
00000034 T initialise_benchmark
00000052 T mulul64
0000006a T modul64
000000a6 T xbinGCD
000000ee T montmul
0000016e t benchmark_body
```

mont64.c

```
static int __attribute__ ((noinline))
benchmark_body (int rpt)
{
      ...
      mulul64 (a, b, &p1hi, &p1lo);
      p1 = modul64 (p1hi, p1lo, m);
      mulul64 (p1, p1, &p1hi, &p1lo);
      p1 = modul64 (p1hi, p1lo, m);
      mulul64 (p1, p1, &p1hi, &p1lo);
      p1 = modul64 (p1hi, p1lo, m);
      ...
      mulul64 (p, rinv, &phi, &plo);
      ...
```

```
uint64
montmul (uint64 abar, uint64 bbar,
         uint64 m, uint64 mprime)
{
  ...
  mulul64 (abar, bbar, &thi, &tlo);
  ...
  mulul64 (tm, m, &tmmhi, &tmmlo);
  ...
```


EMBECOSM®

Disassemble `benchmark_body`

LLVM

```
101b4 <benchmark_body>:
        ...
1023a:  mulhu   a0,s11,a5
1023e:  mulhu   a1,s6,a5
10242:  mul     a2,s6,a5
10246:  add     a2,a2,a0
        ...
10310:  mulhu   a1,a3,a3
10314:  mulhu   a4,a3,a2
10318:  mul     a5,a3,a2
1031c:  add     s1,a1,a5
        ...
103d6:  mulhu   a1,a3,a3
103da:  mulhu   a4,a3,a2
        ...
```

GCC

```
102c6 <benchmark_body>:
        ...
10400:  mul     a5,s3,s0
10404:  mul     s1,s1,s2
10408:  mul     a0,s2,s0
1040c:  add     s1,s1,a5
1040e:  mulhu   s0,s2,s0
        ...
```

Instances of `DW_TAG_inlined_subroutine`

Benchmark	LLVM	GCC
aha-mont64	13	0
crc32	1	0
cubic	0	0
edn	3	0
huffbench	1	0
matmult_int	6	2
minver	2	4
nbody	0	0
nettle-aes	7	2
nettle-sha256	3	1

Benchmark	LLVM	GCC
nsichneu	0	0
picojpeg	180	40
qrduino	45	8
sglib-combined	41	14
slre	13	12
statemate	1	4
st	5	4
ud	0	0
wikisort	23	24

cubic -Os

Clang/LLVM

```
$ nm --size-sort cubic
00000002 T initialise_benchmark
...
00000012 T __multf3
00000012 T __subtf3
00000022 T main
000000f8 T verify_benchmark
000001d6 t benchmark_body
000008ba T SolveCubic
```

GCC

```
$ nm --size-sort cubic
00000002 T initialise_benchmark
...
00000012 T __multf3
00000012 T __subtf3
00000030 T main
000000da T verify_benchmark
0000021a t benchmark_body
0000063e T SolveCubic
```

cubic -Os: Stack Usage

```
10094 <SolveCubic>:
10094:  addi    sp,sp,-1424
        ...
```

```
10094 <SolveCubic>:
10094:  addi    sp,sp,-304
        ...
```

cubic/basicmath_small.c:33

LLVM

```
0cb51563     bne    x10,x11,10a30
6549         c.lui  x10,0x12
d2852603     lw     x12,-728(x10)
d2850413     addi   x8,x10,-728
4054         c.lw   x13,4(x8)
400005b7     lui    x11,0x40000
4501         c.li   x10,0
2eb1         c.jal  10cd8
80000637     lui    x12,0x80000
fff60a13     addi   x20,x12,-1
0145f5b3     and    x11,x11,x20
68497637     lui    x12,0x68497
68260913     addi   x18,x12,1666
3d3c2637     lui    x12,0x3d3c2
5c260993     addi   x19,x12,1474
864a         c.mv   x12,x18
86ce         c.mv   x13,x19
2e29         c.jal  10cb8
```

GCC

```
8d218493     addi x9,x3,-1838
4090         c.lw x12,0(x9)
40d4         c.lw x13,4(x9)
8a21a503     lw   x10,-1886(x3)  # __SDATA_BEGIN__+0xa0
8a61a583     lw   x11,-1882(x3)  # __SDATA_BEGIN__+0xa4
80000437     lui  x8,0x80000
fff44413     xori x8,x8,-1
2a05         c.jal10a40
8aa1aa03     lw   x20,-1878(x3)  # __SDATA_BEGIN__+0xa8
8ae1aa83     lw   x21,-1874(x3)  # __SDATA_BEGIN__+0xac
872a         c.mv x14,x10
0085f7b3     and  x15,x11,x8
8652         c.mv x12,x20
86d6         c.mv x13,x21
853a         c.mv x10,x14
85be         c.mv x11,x15
28e5         c.jal10a20
```


EMBECOSM®

cubic/basicmath_small.c:33

LLVM

```
0cb51563    bne   x10,x11,10a30
6549        c.lui x10,0x12
d2852603    lw    x12,-728(x10)
d2850413    addi  x8,x10,-728
4054        c.lw  x13,4(x8)
400005b7    lui   x11,0x40000
4501        c.li  x10,0
2eb1        c.jal 10cd8
80000637    lui   x12,0x80000
fff60a13    addi  x20,x12,-1
0145f5b3    and   x11,x11,x20
68497637    lui   x12,0x68497
68260913    addi  x18,x12,1666
3d3c2637    lui   x12,0x3d3c2
5c260993    addi  x19,x12,1474
864a        c.mv  x12,x18
86ce        c.mv  x13,x19
2e29        c.jal 10cb8
```

GCC

```
8d218493    addi x9,x3,-1838
4090        c.lw x12,0(x9)
40d4        c.lw x13,4(x9)
8a21a503    lw   x10,-1886(x3)  # __SDATA_BEGIN__+0xa0
8a61a583    lw   x11,-1882(x3)  # __SDATA_BEGIN__+0xa4
80000437    lui  x8,0x80000
fff44413    xori x8,x8,-1
2a05        c.jal10a40
8aa1aa03    lw   x20,-1878(x3)  # __SDATA_BEGIN__+0xa8
8ae1aa83    lw   x21,-1874(x3)  # __SDATA_BEGIN__+0xac
872a        c.mv x14,x10
0085f7b3    and  x15,x11,x8
8652        c.mv x12,x20
86d6        c.mv x13,x21
853a        c.mv x10,x14
85be        c.mv x11,x15
28e5        c.jal10a20
```


EMBECOSM®

cubic/basicmath_small.c:33

LLVM

```
0cb51563    bne    x10,x11,10a30
6549        c.lui  x10,0x12
d2852603    lw     x12,-728(x10)
d2850413    addi   x8,x10,-728
4054        c.lw   x13,4(x8)
400005b7    lui    x11,0x40000
4501        c.li   x10,0
2eb1        c.jal  10cd8
80000637    lui    x12,0x80000
fff60a13    addi   x20,x12,-1
0145f5b3    and    x11,x11,x20
68497637    lui    x12,0x68497
68260913    addi   x18,x12,1666
3d3c2637    lui    x12,0x3d3c2
5c260993    addi   x19,x12,1474
864a        c.mv   x12,x18
86ce        c.mv   x13,x19
2e29        c.jal  10cb8
```

GCC

```
8d218493    addi  x9,x3,-1838
4090        c.lw  x12,0(x9)
40d4        c.lw  x13,4(x9)
8a21a503    lw    x10,-1886(x3)   # __SDATA_BEGIN__+0xa0
8a61a583    lw    x11,-1882(x3)   # __SDATA_BEGIN__+0xa4
80000437    lui   x8,0x80000
fff44413    xori  x8,x8,-1
2a05        c.jal 10a40
8aa1aa03    lw    x20,-1878(x3)   # __SDATA_BEGIN__+0xa8
8ae1aa83    lw    x21,-1874(x3)   # __SDATA_BEGIN__+0xac
872a        c.mv  x14,x10
0085f7b3    and   x15,x11,x8
8652        c.mv  x12,x20
86d6        c.mv  x13,x21
853a        c.mv  x10,x14
85be        c.mv  x11,x15
28e5        c.jal 10a20
```


EMBECOSM®

nettle-aes: Arm v RISC-V

Arm

```
ea82 607c    eor.w   r0, r2, ip, ror #25
```

- Heavy use of constant pools at ends of functions and short loads of global constants via other registers.
- Conditional instructions

RISC-V

```
013e9693f    slli    x13,x29,0x13
00ded793     srli    x15,x29,0xd
8edd         c.or    x13,x15
8db5         c.xor   x11,x13
```

- Many global loads/stores (32-bit)
- Explicit loops

Summary

Summary

- Standard benchmarks provide a useful comparison
- Comparison can identify optimization possibilities
  - by comparing between compilers
  - by comparing between architectures
- Some problems can't be fixed by the compiler!
- Works for any benchmark set – for example
  - https://github.com/westerndigitalcorporation/riscv32-Code-density-test-bench

EMBECOSM®

Thank You

www.embecosm.com

www.embench.org

Jeremy Bennett

jeremy.bennett@embecosm.com