

Initial Evaluation of Multiple RISC ISAs using the Embench™ Benchmark Suite

What is the Cost of Simplicity?



David Patterson (UC Berkeley), Jeremy Bennett (Embecosm),
Palmer Dabbelt (SiFive), Cesare Garlati (Hex Five Security),
and Ofer Shinaar (Western Digital)

Tuesday, December 10, 2019

State of Benchmarks for IoT/Embedded Computers

- Billions of Internet of Things (IoT) devices shipped soon
- Still no high quality, widely reported benchmark for embedded computers

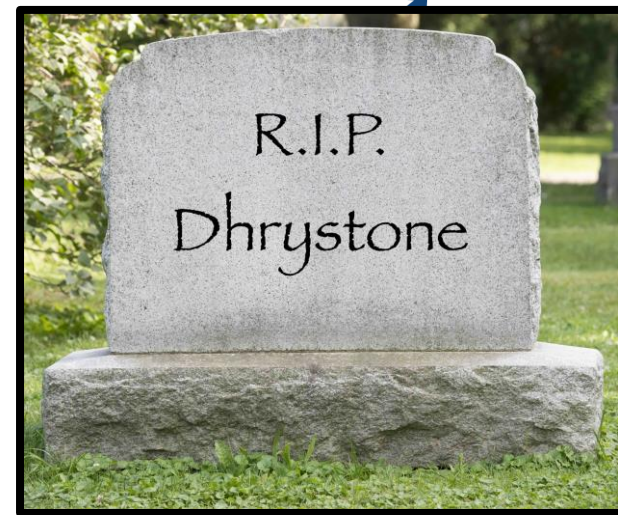


Yunsup Lee, SiFive CTO, Keynote address “Opportunities and Challenges of Building Silicon in the Cloud” 12/5/18 RISC-V Summit:
“... the benchmark scores are 4.9 CoreMarks/MHz and 2.5 DMIPS/MHz. I’m saying this in front of Dave [Patterson], who doesn’t really like Dhrystone or CoreMark as benchmarks. Sorry. This is the industry standard benchmark I learned.”

It’s past time to apologize

Embench

- Group from Academia and Industry Developing
 - We want your help to evolve Embench: info@embench.org
- Free
- Easy to Port
- Suite of ≈ 20 Real Programs (vs 1 Synthetic Program)
- Geometric Mean & Geometric Standard Deviation of Ratios to Reference Platform (PULP RI5CY core)
- Also Report Code Size, Context Switch Time, and Interrupt Latency
 - Necessary for embedded IoT devices yet novel part of formal benchmark
- Sustaining Organization involving Academia and Industry to Evolve over Time
 - FOSSi Foundation: The Free and Open Source Silicon Foundation
- Follows Agile Benchmark Development Philosophy: Versions 0.5, 0.6, ...
- Given current state of widely reported benchmarks for embedded computing, we believe Embench—even the 0.5 version—will be a big help to the IoT field



Embench 0.5 ($\leq 32\text{KB}$ ROM/Flash, $\leq 16\text{KB}$ RAM)




<i>Name</i>	<i>Comments</i>	<i>Original Source</i>	<i>C LOC</i>	<i>code size</i>	<i>data size</i>	<i>time (ms)</i>	<i>branch</i>	<i>memory</i>	<i>compute</i>
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	Cryptographic 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

















embench / **embench-iot** Watch 10 Star 21 Fork 14

Code Issues 2 Pull requests 0 Projects 0 Security Insights

15 commits 1 branch 0 packages 0 releases 5 contributors GPL-3.0

Branch: master New pull request Find file Clone or download

 **jeremybennett** Note that Embench is a trademark (#28) ... Latest commit 976679c 12 days ago

 baseline-data	Py build (#9)	3 months ago
 config	Py build (#9)	3 months ago
 doc	Note that Embench is a trademark (#28)	12 days ago
 pylib	Ensure we use at least Python 3.6. (#25)	26 days ago
 src	Use __int128 for 64 x 64 -> 128 bit multiplication if available (#19)	15 days ago
 support	Fix several errors in the places where floating point is used.	27 days ago
 .gitignore	Py build (#9)	3 months ago
 AUTHORS	Initial commit of the new repository.	6 months ago
 COPYING	Initial commit of the new repository.	6 months ago
 ChangeLog	Remove initialization of new empty dictionary. (#13)	27 days ago
 INSTALL	Update documentation and convert to Markdown (#27)	15 days ago
 NEWS	Clean up a couple of annoyances	6 months ago
 README.md	Note that Embench is a trademark (#28)	12 days ago
 benchmark_size.py	Ensure we use at least Python 3.6. (#25)	26 days ago
 benchmark_speed.py	Ensure we use at least Python 3.6. (#25)	26 days ago
 build_all.py	Ensure we use at least Python 3.6. (#25)	26 days ago

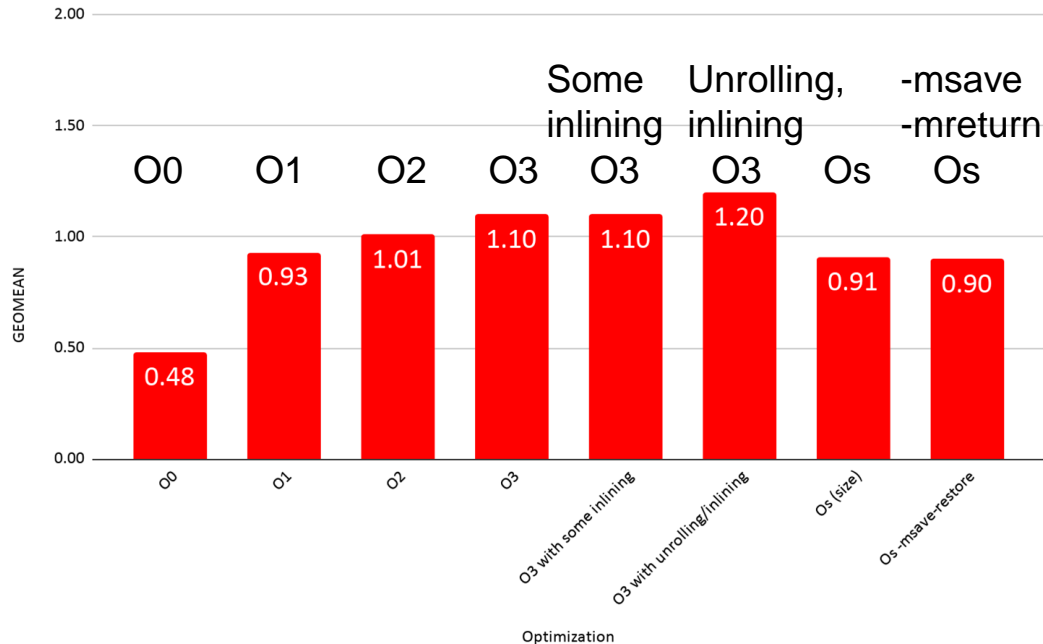
What affects code size for embedded apps?

- Instruction Set Architecture
 - ARM vs ARC vs RISC-V vs AVR vs ...
 - Which extensions included:
 - ARM: v7, Thumb2, ...
 - RISC-V: RV32, RV64, M (Multiple/divide), C (compress), ...
- Compiler
 - Open (GCC, LLVM) and Proprietary (Embecosm, IAR, ...)
 - Which optimizations included: Loop unrolling, inlining procedures, minimize code size, ...
 - How fast are compilers improving?
 - Older ISAs likely have more mature and better compilers?
- Libraries
 - Open (GCC, LLVM) and Proprietary (IAR, Sega, ...)
- **Embench excludes libraries as they can swamp code size for embedded benchmark**

Impact of optimizations of GCC on RISC-V: Speed

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Higher is faster)

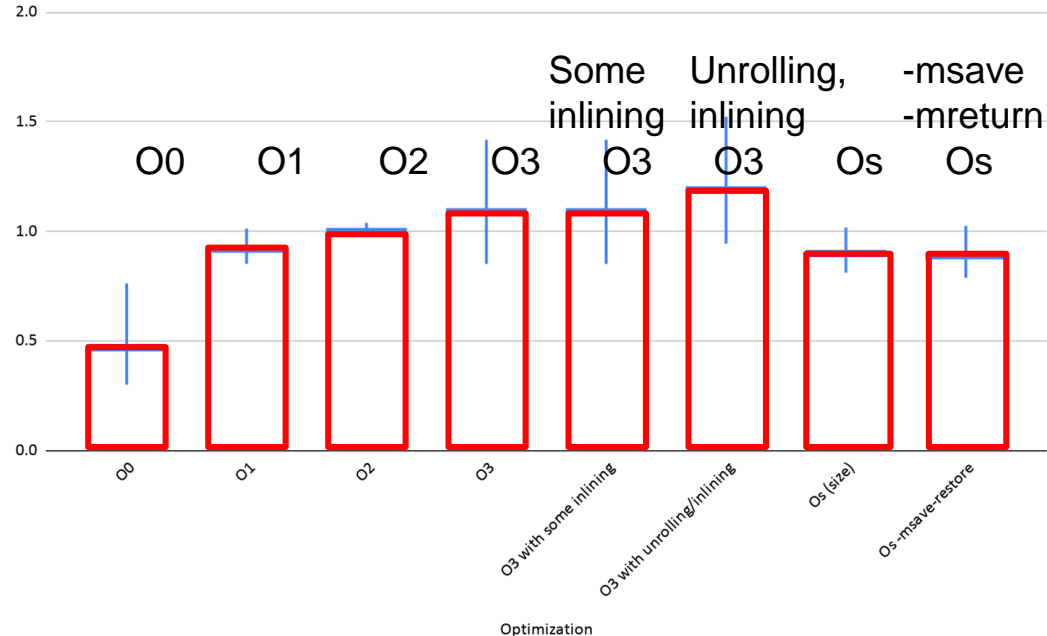
GEOMEAN vs. Optimization: Speed



Impact of optimizations: Speed with Geo Std Dev

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Higher is faster)

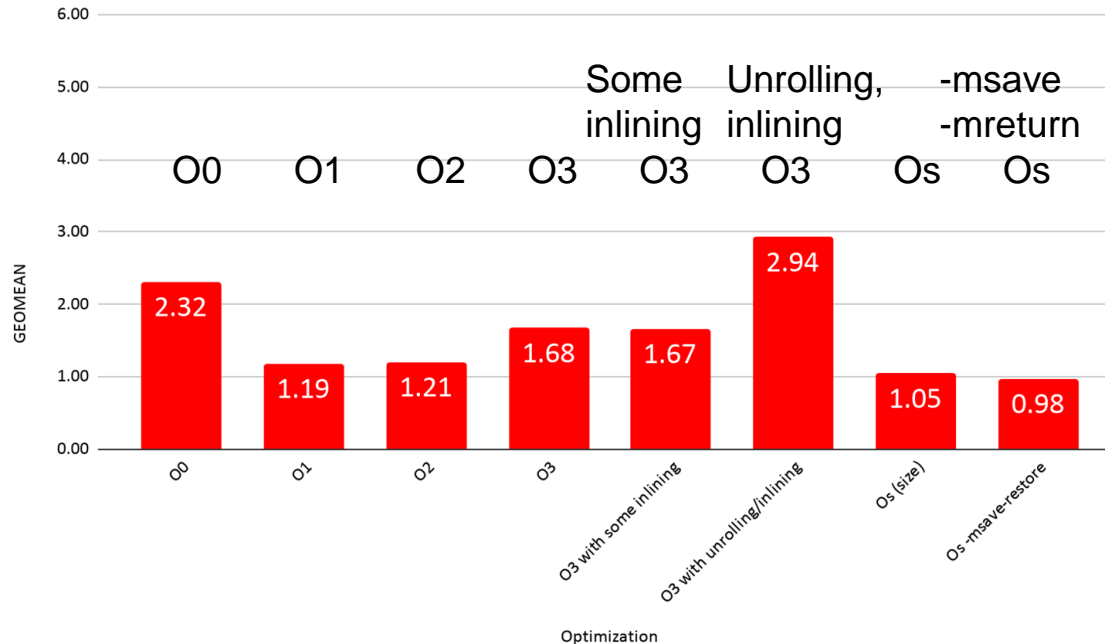
GEOMEAN Speed vs GCC Optimization



Impact of optimizations of GCC on RISC-V: Code Size

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Lower is smaller)

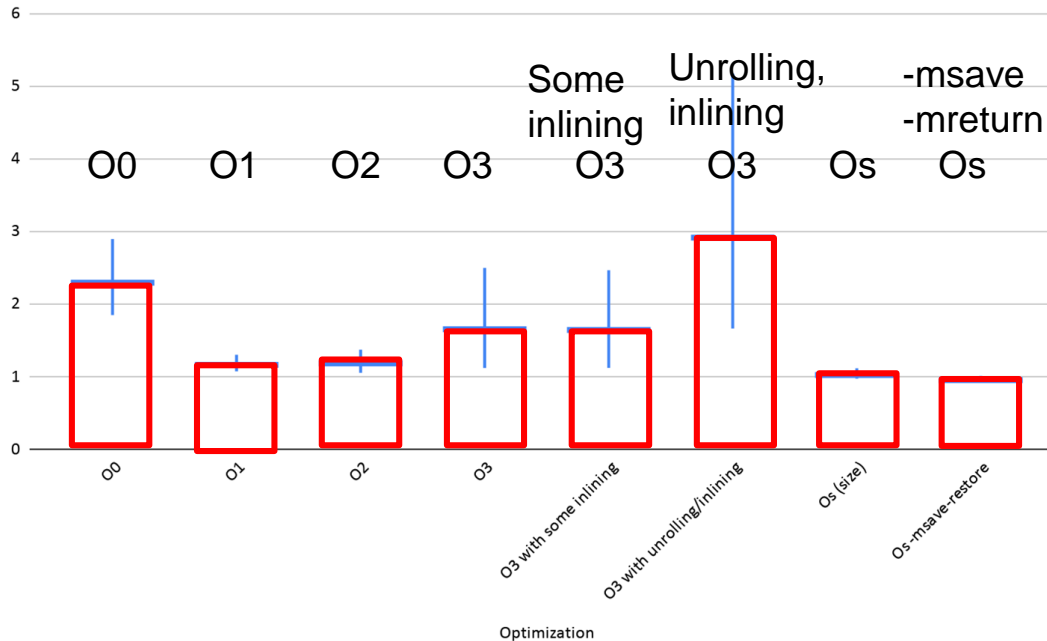
GEOMEAN vs. Optimization: Code Size



Impact of optimizations: Code Size with Geo Std Dev

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Lower is smaller)

GEOMEAN Code Size vs GCC Optimization



Instruction Set Observations

- -msave-mrestore invokes functions to save and restore registers at procedure entry and exit instead of inline code of stores and loads
 - ISA Alternative would be Store Multiple instruction and Load Multiple instruction
- Reduces code size another 7%
- But also reduces performance 10%

Benchmarking Lessons?

1. Must show code size with performance so as to get meaningful results
2. Importance of Geo Standard Deviation as well as Geo Mean

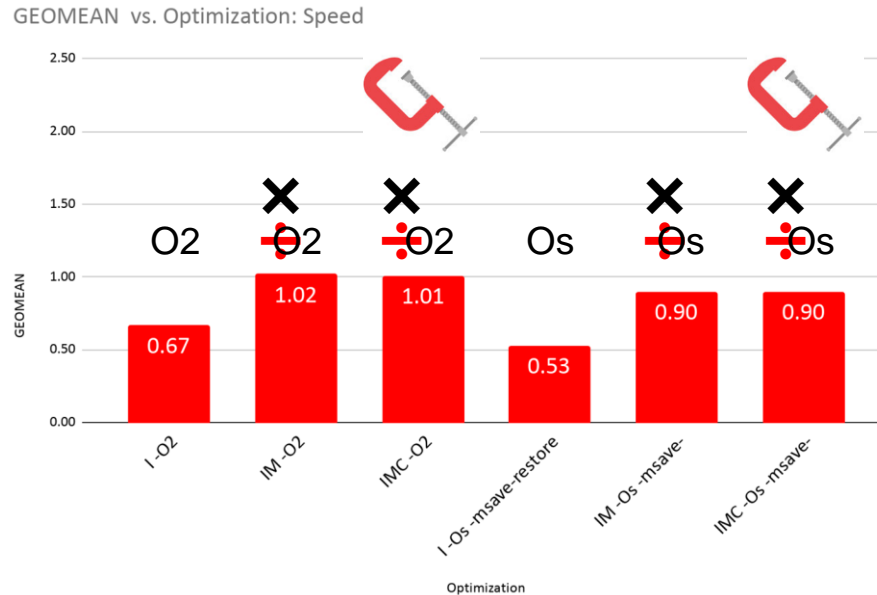
E.g., O3 with loop unrolling and procedure inlining worthwhile?

1X to 1.5X (1.2X Geo Mean) faster programs but

2X to 5X (3X Geo Mean) bigger programs

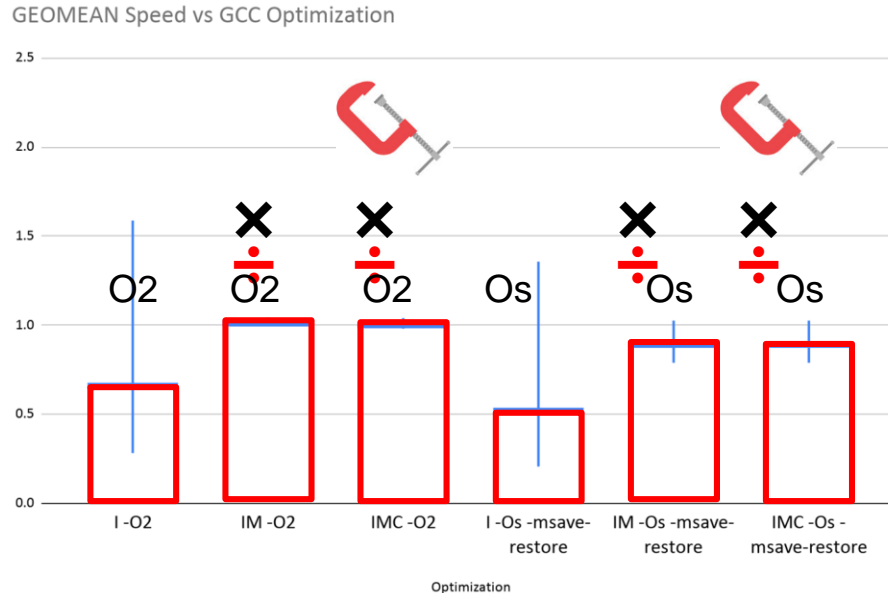
Impact of ISA of GCC on RISC-V: Speed

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave-mrestore
- RI5CY RV32I GCC 10.0.0 (Higher is faster)



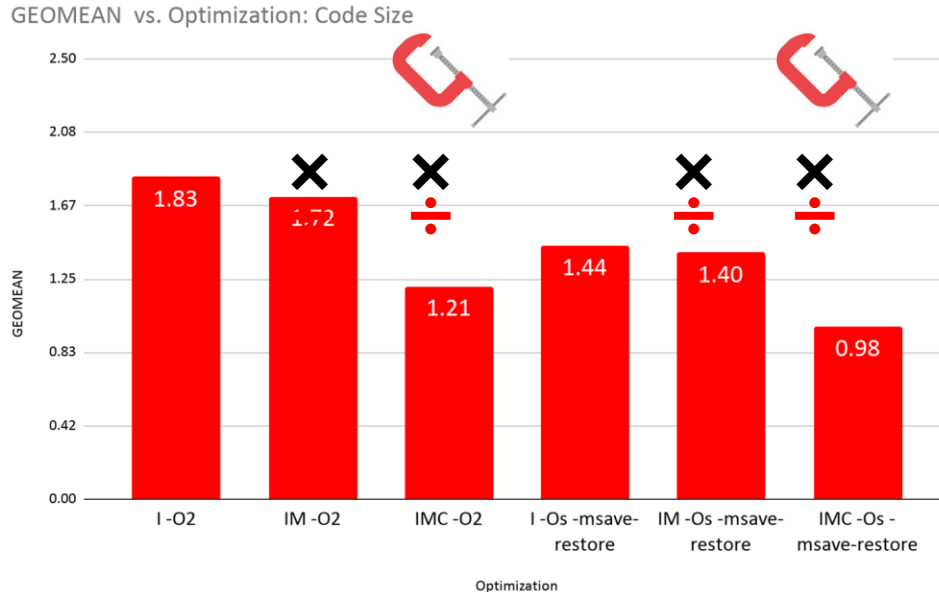
Impact of optimizations: Speed with Geo Std Dev

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave -mrestore
- RI5CY RV32I GCC 10.0.0 18-Nov-19 (Higher is faster)



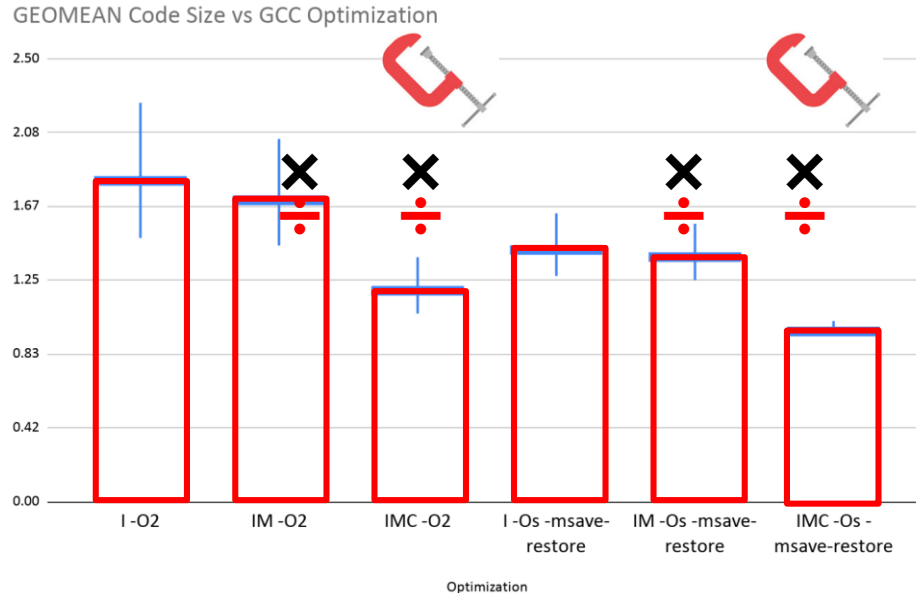
Impact of ISA of GCC on RISC-V: Code Size

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave-mrestore
- RI5CY RV32 GCC 10.0.0 (Lower is smaller)



Impact of optimizations: Code Size with Geo Std Dev

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave -mrestore
- RI5CY RV32I GCC 10.0.0 18-Nov-19 (Lower is smaller)

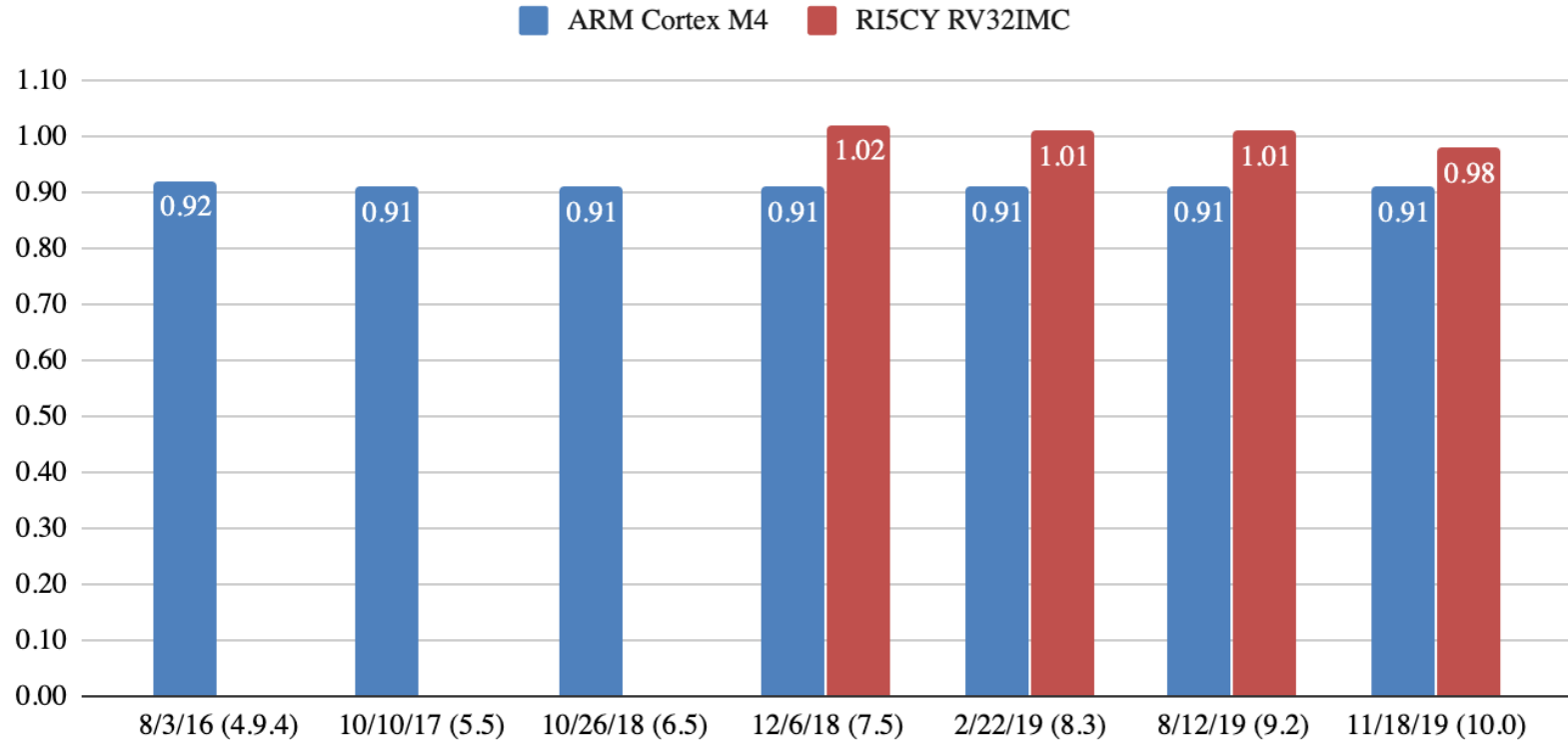


Benchmarking/RISC-V Lessons?

1. Multiply/Divide (RV32IM) improves performance 1.5-1.7X over integer baseline (RV32I) and reduces code size 3% to 6%
2. Compress has no impact on performance, reduces code size 1.4-1.5X
3. -msave-mrestore reduce performance 10%, code size 1.25X over GCC -O2
 - As opposed to 10% code size over GCC -Os
4. Integer only has widely varying performance (Geo Std Dev 2.5)

Code Size over GCC Versions (ARM M4, RV32IMC)

GCC -Os (-msave-mrestore) dates and versions

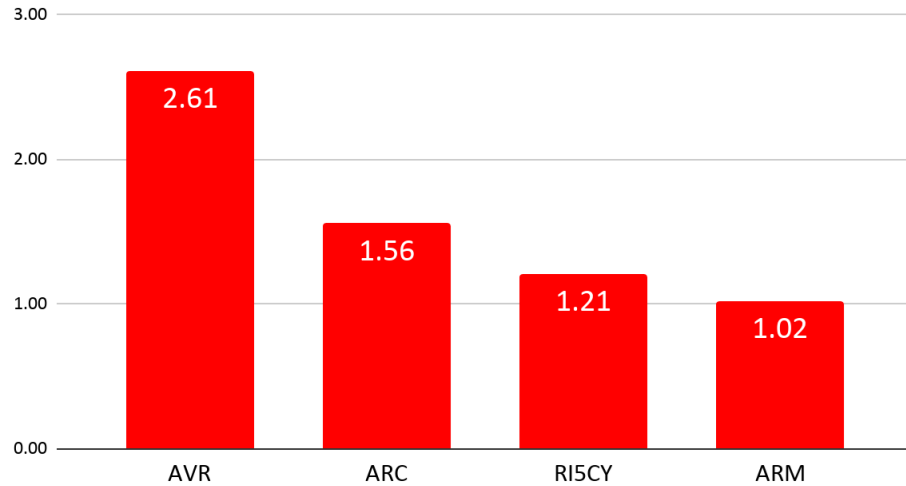


Code Size by Architecture: GCC 10.0.0 -O2 & -Os

- Atmel AVR ATmega64 (8-bit RISC microcontroller)
 - Licensed core AVR ISA from Atmel (acquired by Microchip Technology in 2016)
- ARC EM (32-bit RISC style ISA) not including compressed instructions
 - Licensed core from Synopsys
- RI5CY RV32IMC (32-bit RISC-V with compressed instructions)
 - Popular open source core from ETH Zurich: low power targeting high energy efficiencies
- ARM Cortex M4 (32-bit ARM with Thumb2 compressed instructions)
 - Popular licensed core from ARM

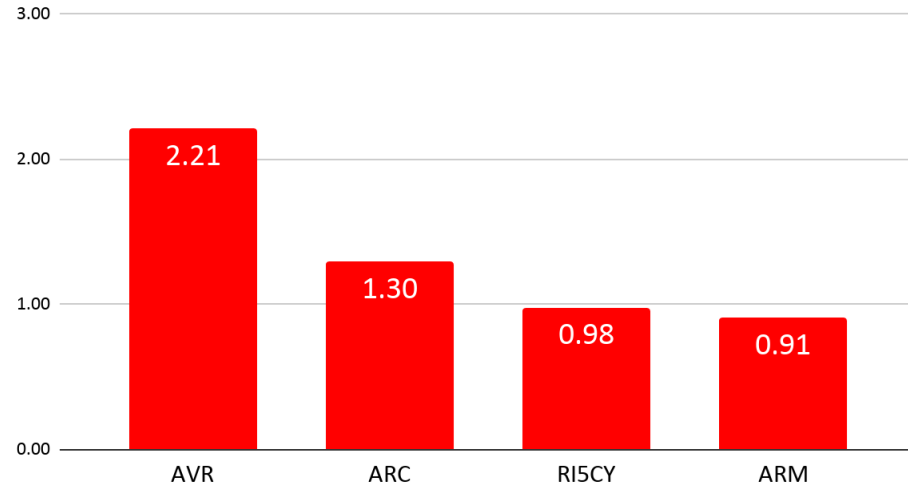
Code Size by Architecture: GCC 10.0.0 -O2 & -Os

Code Size GCC -O2 (10.0.0) Lower is smaller code



O2

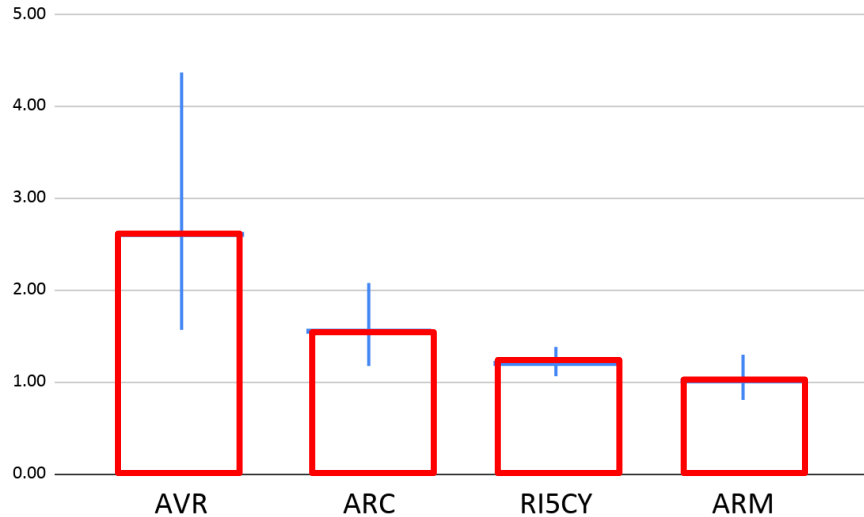
Code Size GCC -Os (10.0.0) Lower is smaller code



Os
(-msave-mrestore for RISCY)

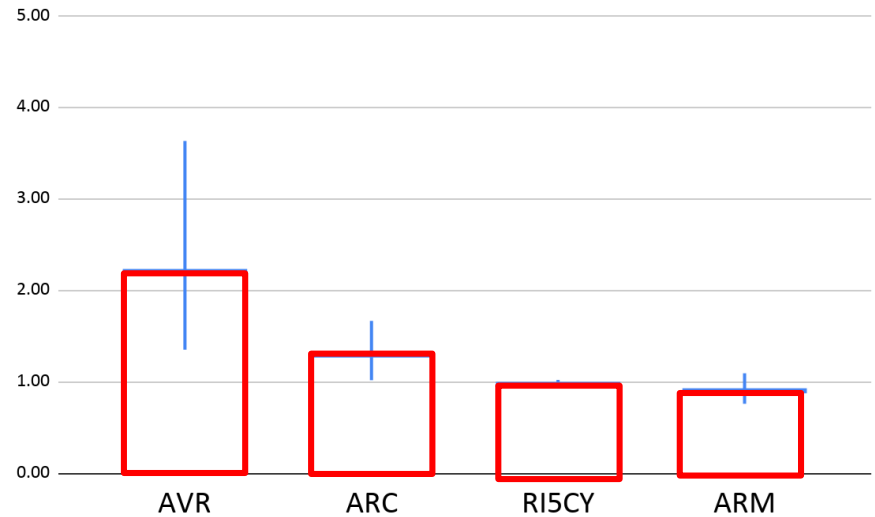
Code Size by Architecture: GCC 10.0.0 -O2 & -Os

Code Size GCC -O2 (10.0.0) Lower is smaller code



O2

Code Size GCC -Os (10.0.0) Lower is smaller code



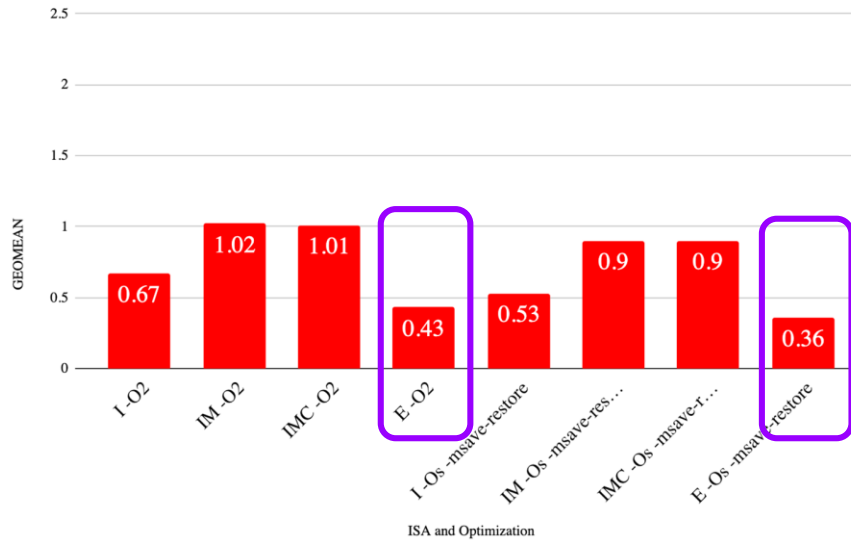
Os

(-msave-mrestore for RI5CY)

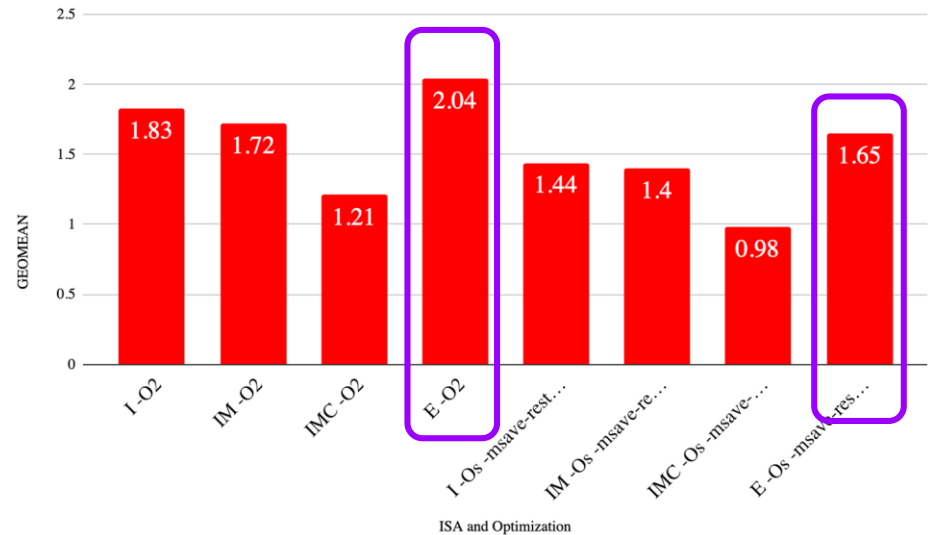
RISC-V E option (for embedded)

- Only 16 registers for tiniest processors: 0.4X perf, 1.7-2.0X larger program

GEOMEAN vs. Optimization: Speed



GEOMEAN vs. Optimization: Code Size



Lots more to explore with Embench

- More compilers: LLVM, IAR, ...
 - And more optimizations
- More architectures: MIPS, Tensilica, ARMv8, RV64I, ...
 - And more instruction extensions: bit manipulation, vector, floating point, ...
- More processors: ARM M7, M33, M24, RISC-V Rocket, BOOM, ...
- Context switch times
- In later versions of Embench: Interrupt Latency, Floating Point programs
- Collect and publish results on [Embench.org](https://embench.org) web site
- Want to help? Email info@embench.org

Related Talks by Embench Members next 2 sessions

- **GCC Compiler: Code Size Density**
 - Nidal Faour and Ofer Shinaar, Western Digital
 - 1:20pm - 1:40pm today, Grand Ballroom 220-C (We're in 220-A now)
 - Using small test cases derived from real scenarios when comparing the RISC-V to other Cores
- **Open Source Compiler Tool Chains for RISC-V:
Past, Present and Future**
 - Jeremy Bennett, Embecosm
 - 1:50 pm - 2:10pm today, Grand Ballroom 220-C (We're in 220-A now)
 - Using Embench to explore more features and compilers (e.g., LLVM, RISC-V Bit extension)

Conclusions

- Code size and performance should be linked for embedded benchmarks
 - Loop unrolling and procedure inlining can triple code size
- RISC-V M extension improves performance 1.5-1.7X and code size 3%-6%
- Using GCC and Embench, RV32IMC code much smaller than AVR
- ARM Thumb2 smaller than RV32IMC, but within one standard deviation
- In past year RISC-V GCC getting better at code size, ARM GCC stable/mature
- For GCC 10.0 compiler, adding Load Multiple/Store Multiple might add 10% performance with same code size over `-msave-mrestore` optimizations at cost of more complex ISA implementation
- We believe Embench 0.5 suite is already an improvement over single synthetic programs Dhrystone and CoreMark
- Let us know if you'd like to help: Email info@embench.org