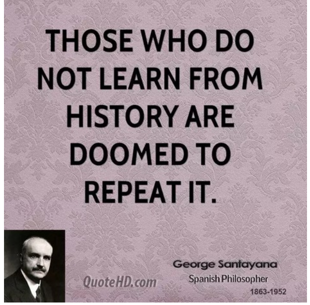
Towards Embench 0.6
A Free Benchmark Suite for IoT from an Academic-Industry Cooperative

David Patterson (UC Berkeley)
Jeremy Bennett, Paolo Savini (Embecosm)
Palmer Dabbelt (SiFive)
Cesare Garlati (Hex Five Security)

21 September 2020

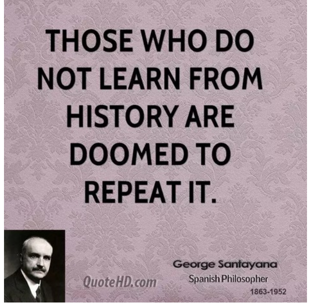
History (1)



- Incorporate good ideas and avoid mistakes of past benchmarks
- First learn from features of widely quoted benchmarks:

	Linpack	Dhrystone	SPEC CPU	CoreMark	MLPerf 0.5
Year	1977	1984	1989	2009	2018
Initial target	HPC	Sys. prog	Unix server	Embedded	ML server
Quality reputation	Low	Low	High	Low	Low
Free?	✓	✓	✗	✓	✓
Easy to port	✓	✓	✓	✓	✓
Revision freq.	None since 1991	None since 1998	3 years	Never	1 year (planned)
# programs	1	1	10-23	1	7
Organization	✓	✗	✓	✓	✓
Summary score	FLOPS	Speed ratio (S.R.)	Geomean S.R.	S.R	SR + Std. Dev.
Developer	Academia	Academia	Acad. + Industry	Academia	Acad. + Industry

History (2)



- Incorporate good ideas and avoid mistakes of past benchmarks
- First learn from features of *less* widely quoted benchmarks:

	EEMBC	*MiBench*	*BEEBS*	*TACLeBench*
Year	1997	2001	2013	2016
Initial target	Embedded	Embedded	Compiler	Worst Case Execution
Quality reputation	?	?	?	?
Free?	✘	✓	✓	✓
Easy to port	✘	✓	✓	✓
Revision freq.	Rare	Never	2 years	Never
# programs	41	36	80	52
Organization	✓	✘	✘	✘
Summary score	✘	✘	✘	✘
Developer	Industry	Academia	Acad. + Industry	Academia

Embench

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 geometric 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 Embench 0.5 at Embedded World
- **Present:** Working on Embench 0.6

Current Status

- Set of 19 benchmarks for MCU class processors
  - up to 64KB total memory
- Early benchmark for context switching in RISC-V
  - also needs benchmark for interrupt latency
- Python build and benchmark scripts
  - data available for Arm Cortex-M, RISC-V and ARC
  - need to widen to more architectures
- Ada version in progress
- Embench 0.6 will add floating point suite for up to 1MB machines

Baseline Data

Name	Comments	Orig Source	C LOC	code size	data size	time (ms)	branch	memory	compute
aha-mont64	Montgomery multiplication	AHA	162	1,072	0	4,004	low	low	high
crc32	CRC error checking 32b	MiBench	101	284	1,024	4,010	high	med	low
cubic	Cubic root solver	MiBench	125	1,584	0	3,831	low	med	med
edn	More general filter	WCET	285	1,324	1,600	4,010	low	high	med
huffbench	Compress/Decompress	Scott Ladd	309	1,242	1,001	4,120	med	med	med
matmult-int	Integer matrix multiply	WCET	175	492	1,600	3,985	med	med	med
minver	Matrix inversion	WCET	187	1,168	144	3,998	high	low	med
nbody	Satellite N body, large data	CLBG	172	950	640	2,808	med	low	high
nettle-aes	Encrypt/decrypt	Nettle	1,018	2,148	10,284	4,026	med	high	low
nettle-sha256	Crytographic hash	Nettle	349	3,396	412	3,997	low	med	med
nsichneu	Large - Petri net	WCET	2,676	11,968	8	4,001	med	high	low
picojpeg	JPEG	MiBench2	2,182	6,964	982	4,030	med	med	high
qrduino	QR codes	Github	936	5,814	1,505	4,253	low	med	med
sglib-combined	Simple Generic Library for C	SGLIB	1,844	2,272	800	3,981	high	high	low
slre	Regex	SLRE	506	2,200	121	4,010	high	med	med
st	Statistics	WCET	117	1,000	0	4,080	med	low	high
statemate	State machine (car window)	C-LAB	1,301	4,484	64	4,001	high	high	low
ud	LUD composition Int	WCET	95	720	0	3,999	med	low	high
wikisort	Merge sort	Github	866	4,296	3240	2,779	med	med	med

Compiler: arm-none-eabi-gcc 9.2.0, board: stm32f4-discovery
Options: -Os for code size, -O2 for code speed

Public Repository

The main Embench repository https://www.embench.org/

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

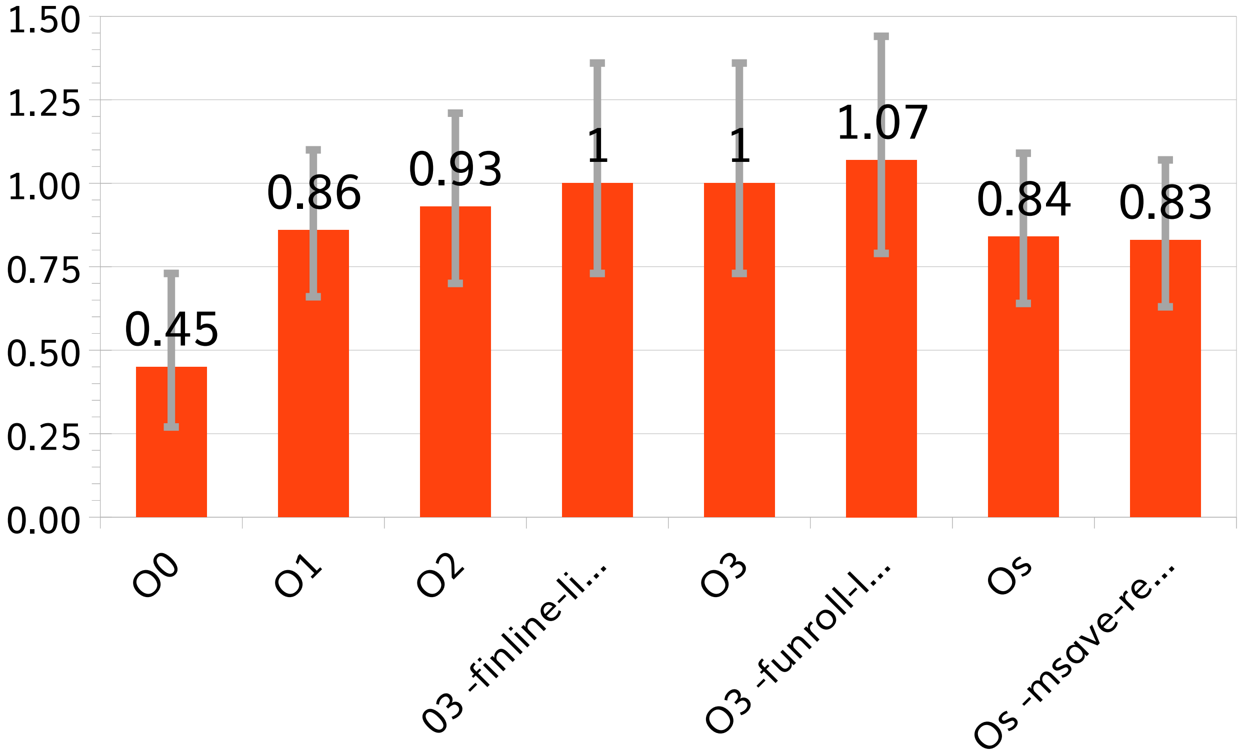
baseline-data	Py build (#9)	3 months ago
config	Py build (#9)	3 months ago
doc	Note that Embench is a trademark (#28)	12 days ago
pylib	Ensure we use at least Python 3.6. (#25)	26 days ago
src	Use __int128 for 64 x 64 -> 128 bit multiplication if available (#19)	15 days ago
support	Fix several errors 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

Embench

What Affects Embench Results?

- Instruction Set Architecture: Arm, ARC, RISC-V, AVR, …
  - extensions: ARM: v7, Thumb2, …, RV32I, M, C, …
- Compiler: open (GCC, LLVM) and proprietary (IAR, …)
  - which optimizations included: Loop unrolling, inlining procedures, minimize code size, …
  - 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 benchmark

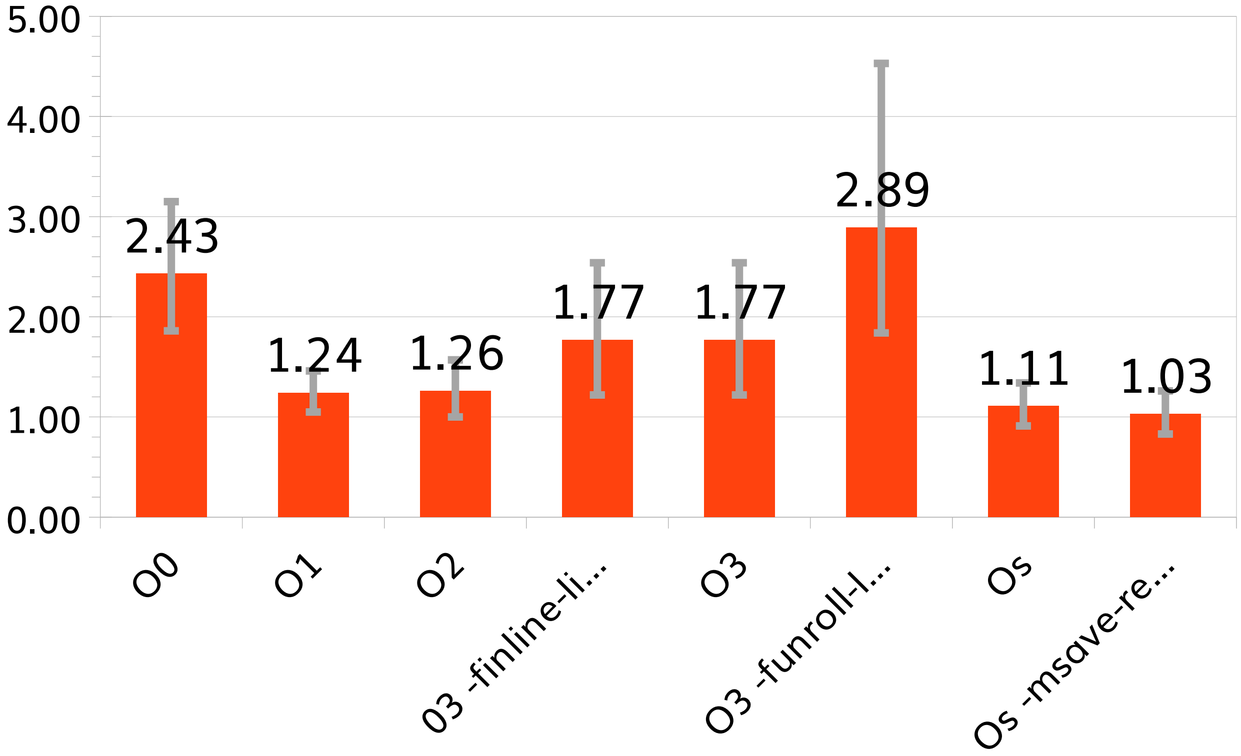
Impact of optimizations of GCC on RISC-V: Speed



PULP RI5CY RV32IMC GCC 10.2.0 (higher is faster)

- **`-msave-restore`** 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

Impact of optimizations of GCC on RISC-V: Size



PULP RI5CY RV32IMC GCC 10.2.0 (lower is smaller)

- **`-msave-restore`** 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

Instruction Set Observations

- `-msave-restore` 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 8%
- Reduces performance 1%

Benchmarking Lessons?

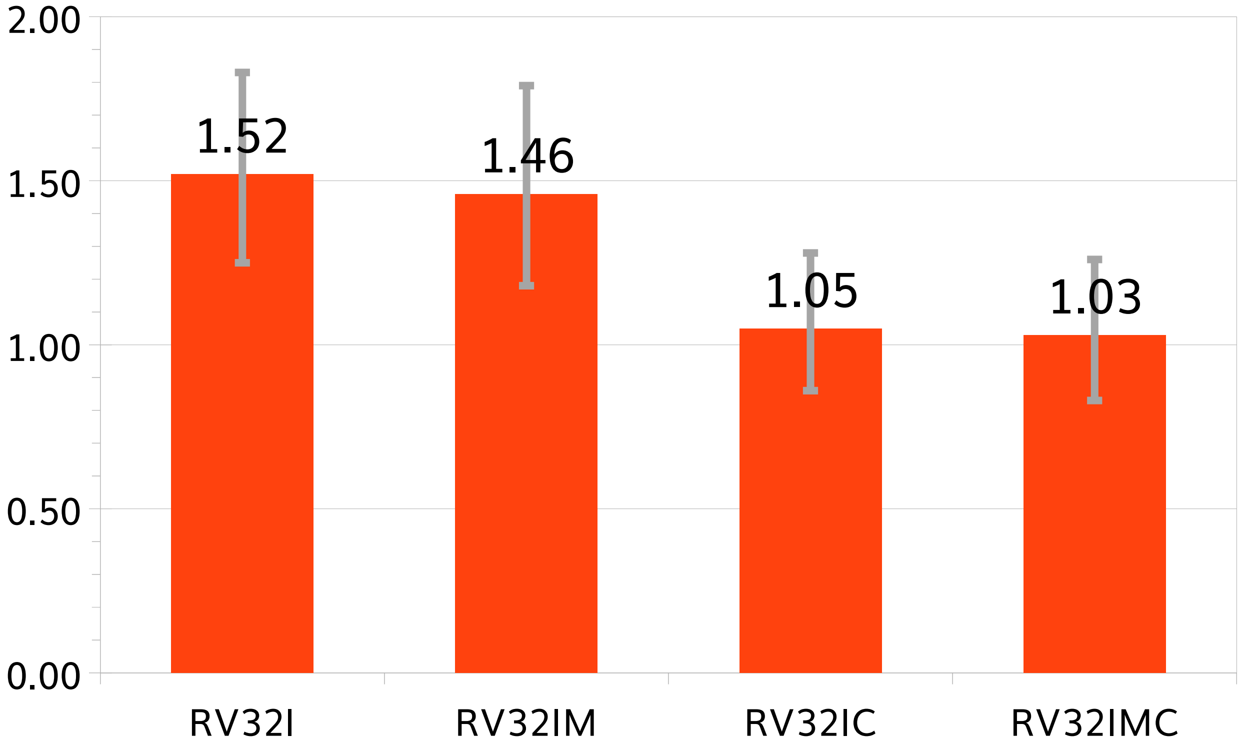
1) Must show code size with performance so as to get meaningful results

2) Importance of geometric standard deviation as well as geometric mean

   E.g., is **`-O3 -funroll-loops`** worthwhile compare to **`-Os -msave-restore`**?

   - 24% faster programs but
   - 186% bigger programs

Impact of ISA of GCC on RISC-V: Size



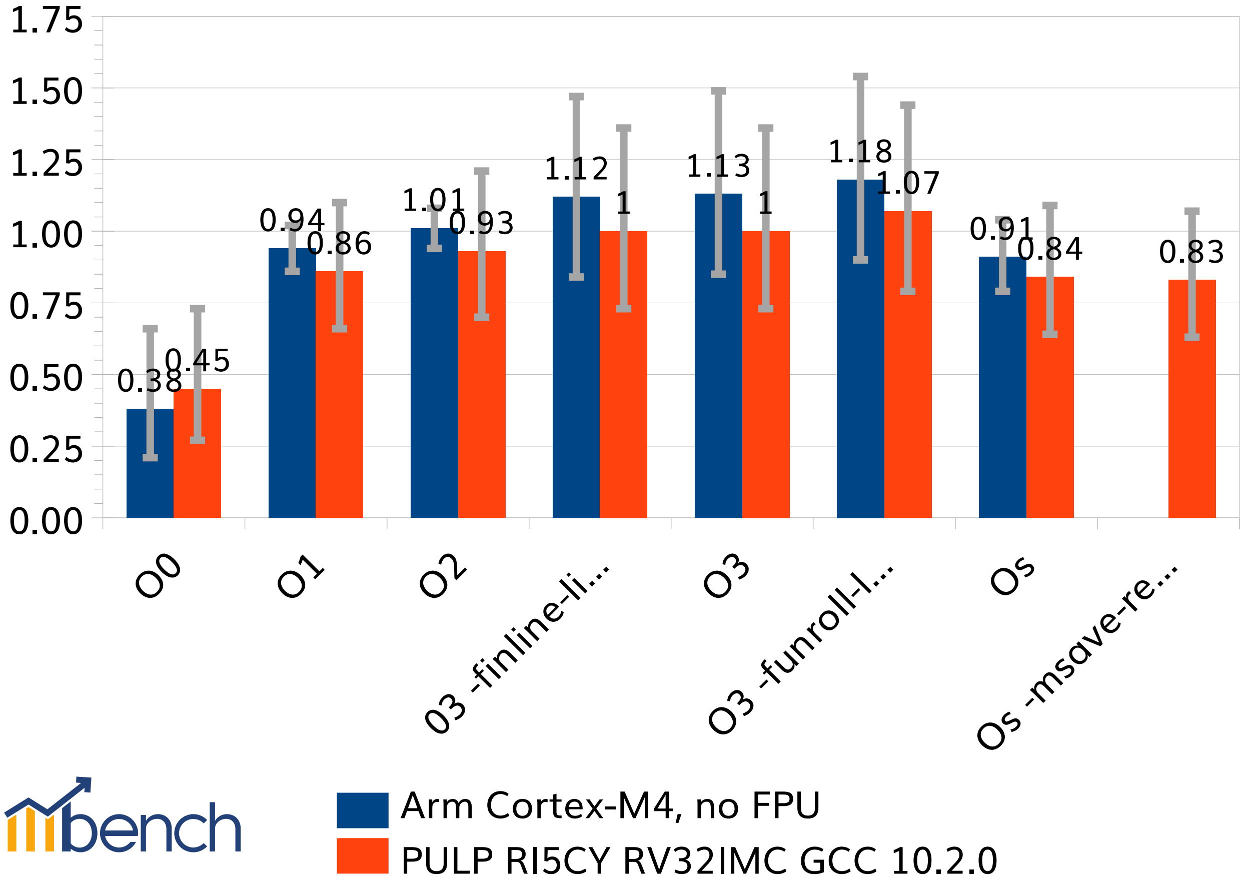
PULP RI5CY RV32 GCC 10.2.0 (lower is smaller)

- Add
  - M (multiply/divide)
  - C (compress)
- **`-O2`** for speed
- **`-Os -msave-restore`** for size

Benchmarking/RISC-V Lessons?

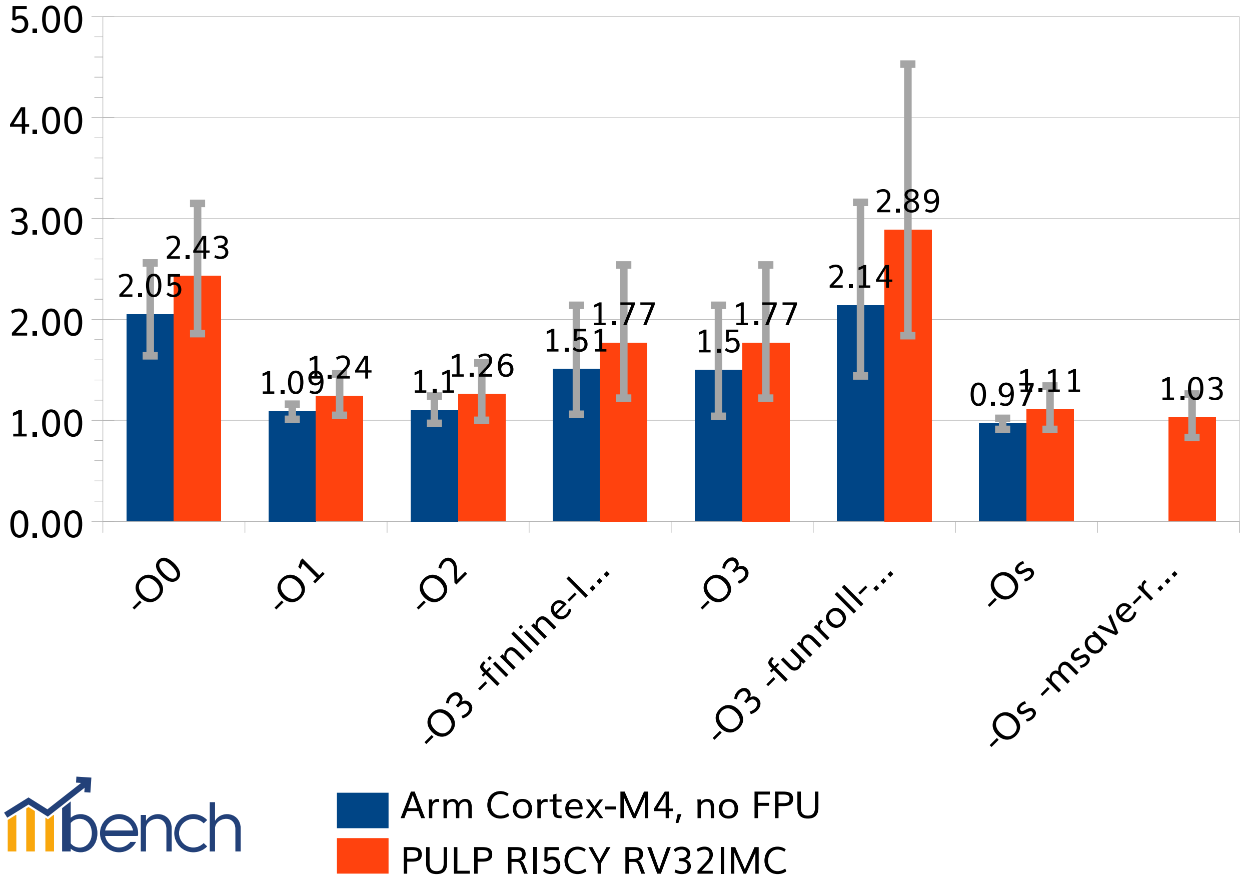
1) Multiply/divide (RV32IM) improves performance 1.5-1.7x over integer baseline (RV32I) and reduces code size 3-6%
2) Compress has no impact on performance, reduces code size 1.4-1.5x
3) Without hardware multiplication widely varying performance (geometric std dev 2.4-2.8)

Comparing Architectures with GCC: Speed



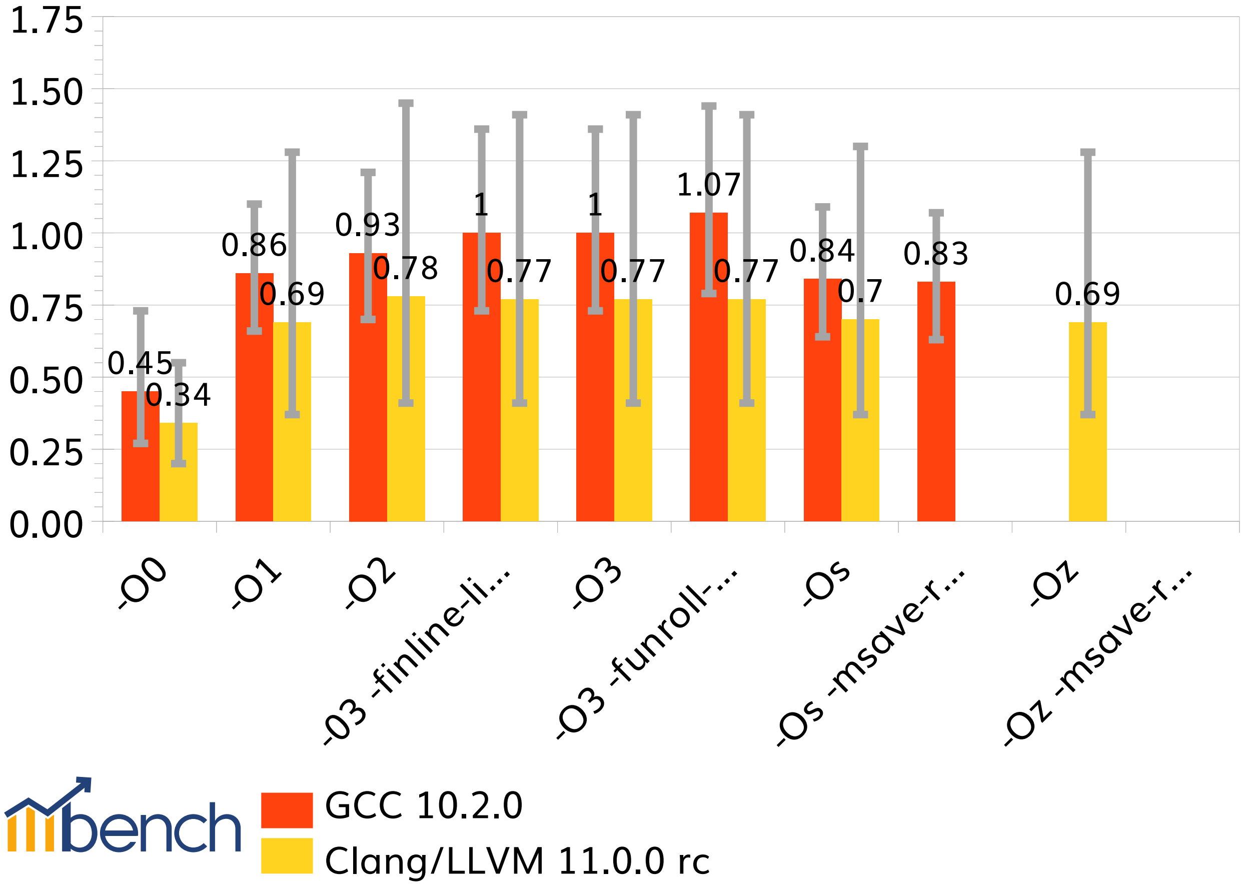
- GCC 10.2.0
  - higher is faster

Comparing Architectures with GCC: Size



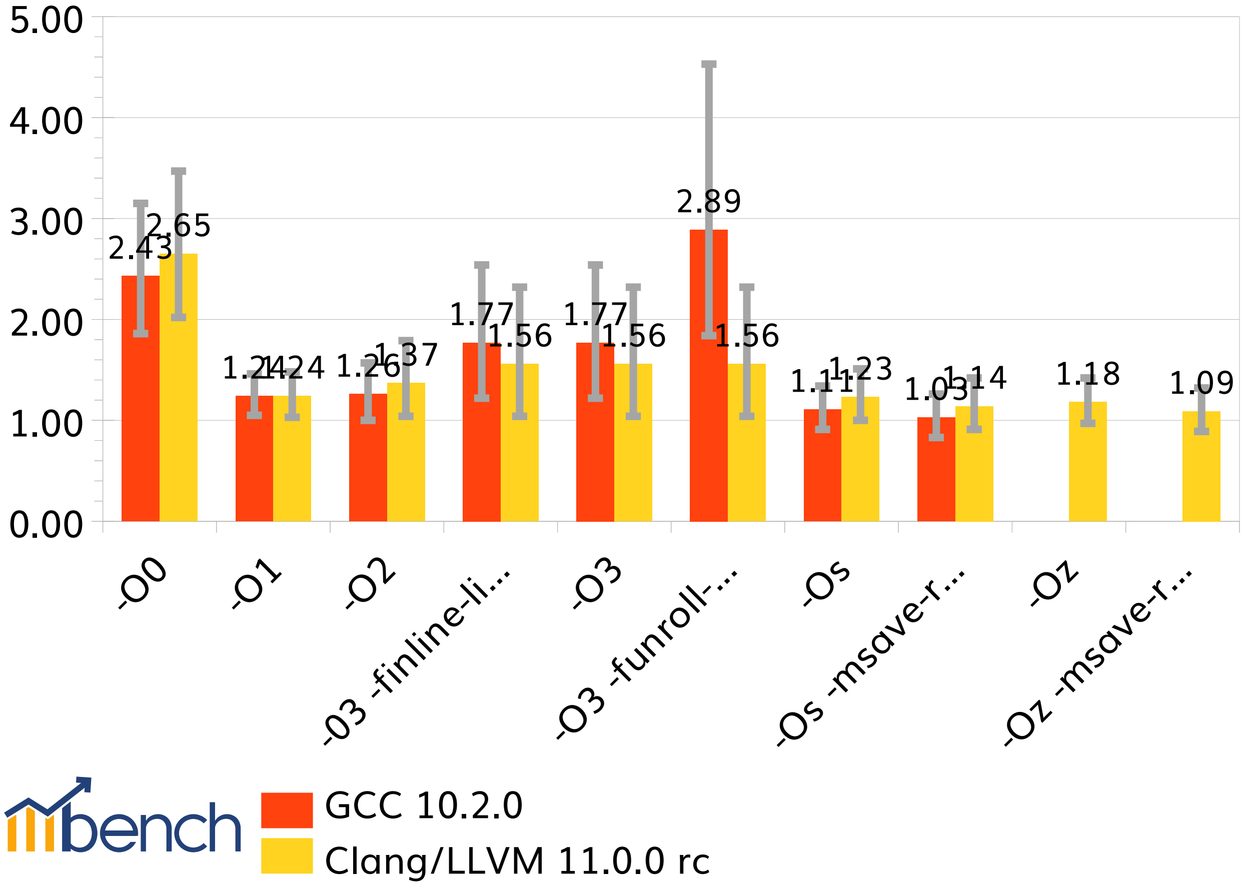
- GCC 10.2.0
  - lower is smaller

Comparing Compilers GCC v LLVM: Speed



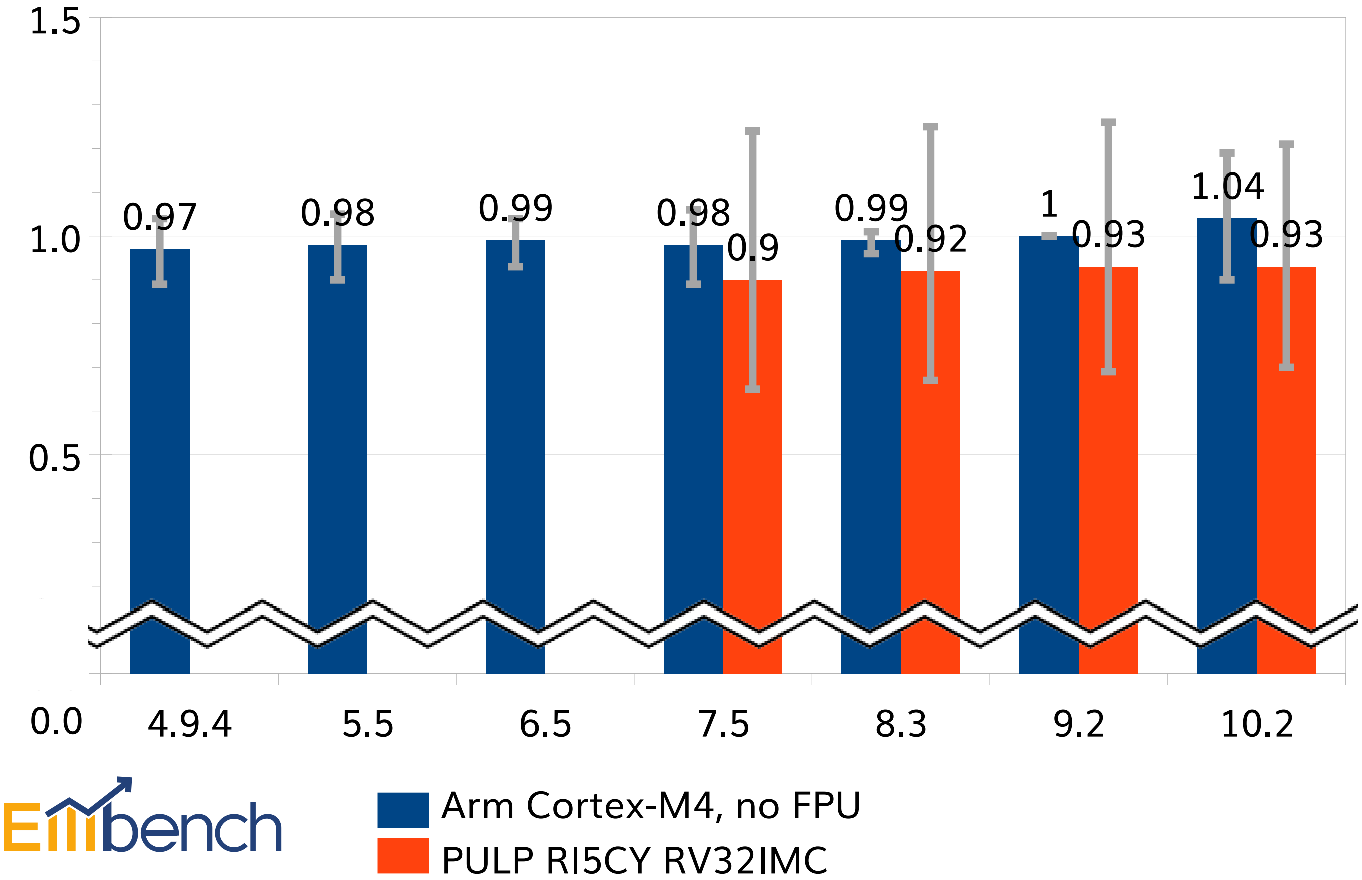
- PULP RI5CY RV32IMC
  - higher is faster
- Clang/LLVM variations
  - **`-msave-restore`** enabled by default with **`-Os`**
  - **`-Oz`** for further code size optimization

Comparing Compilers GCC v LLVM: Size



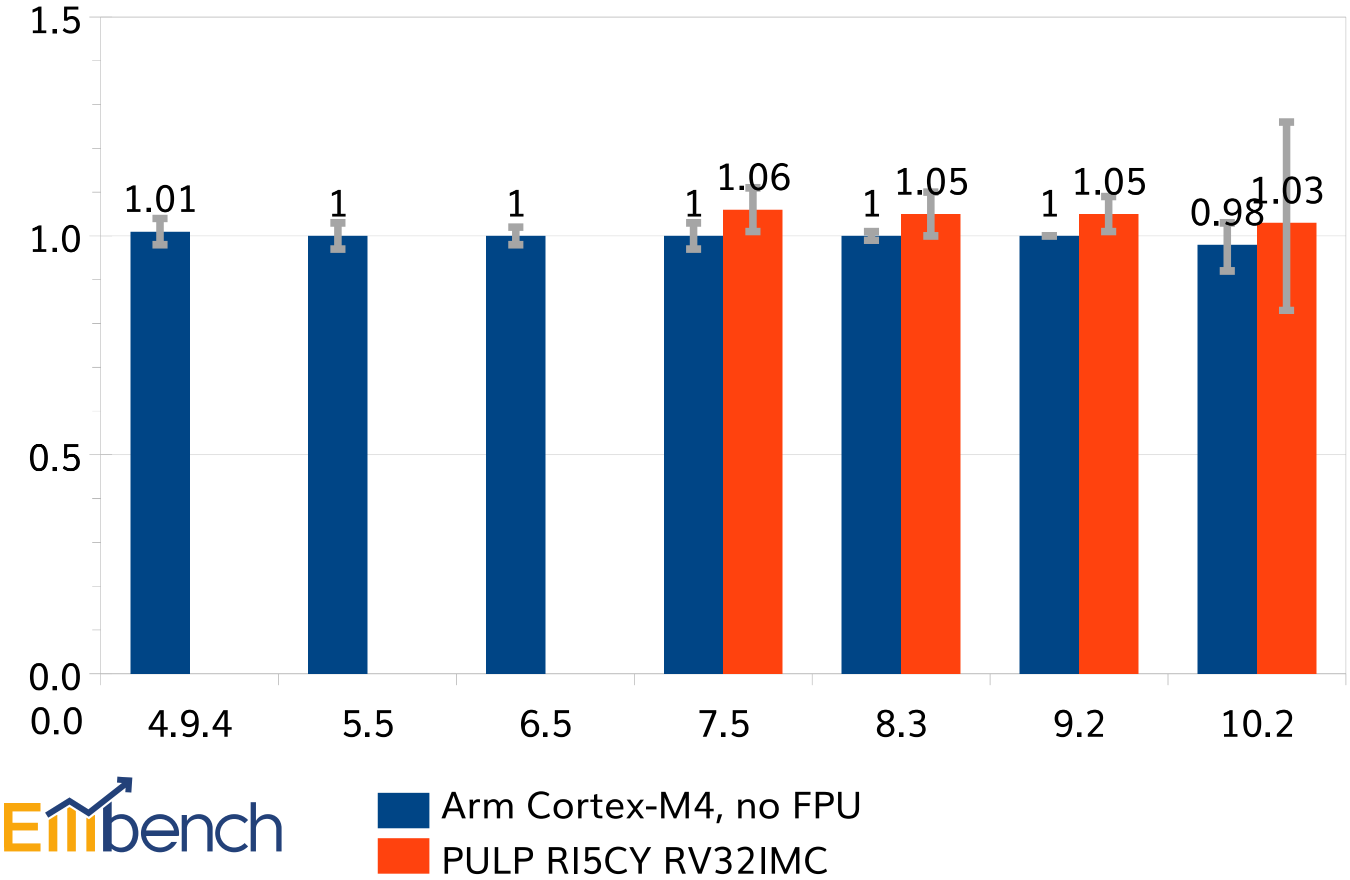
- PULP RI5CY RV32IMC
  - lower is smaller
- Clang/LLVM variations
  - **`-msave-restore`** enabled by default with **`-Os`**
  - **`-Oz`** for further code size optimization

Code Speed over GCC versions



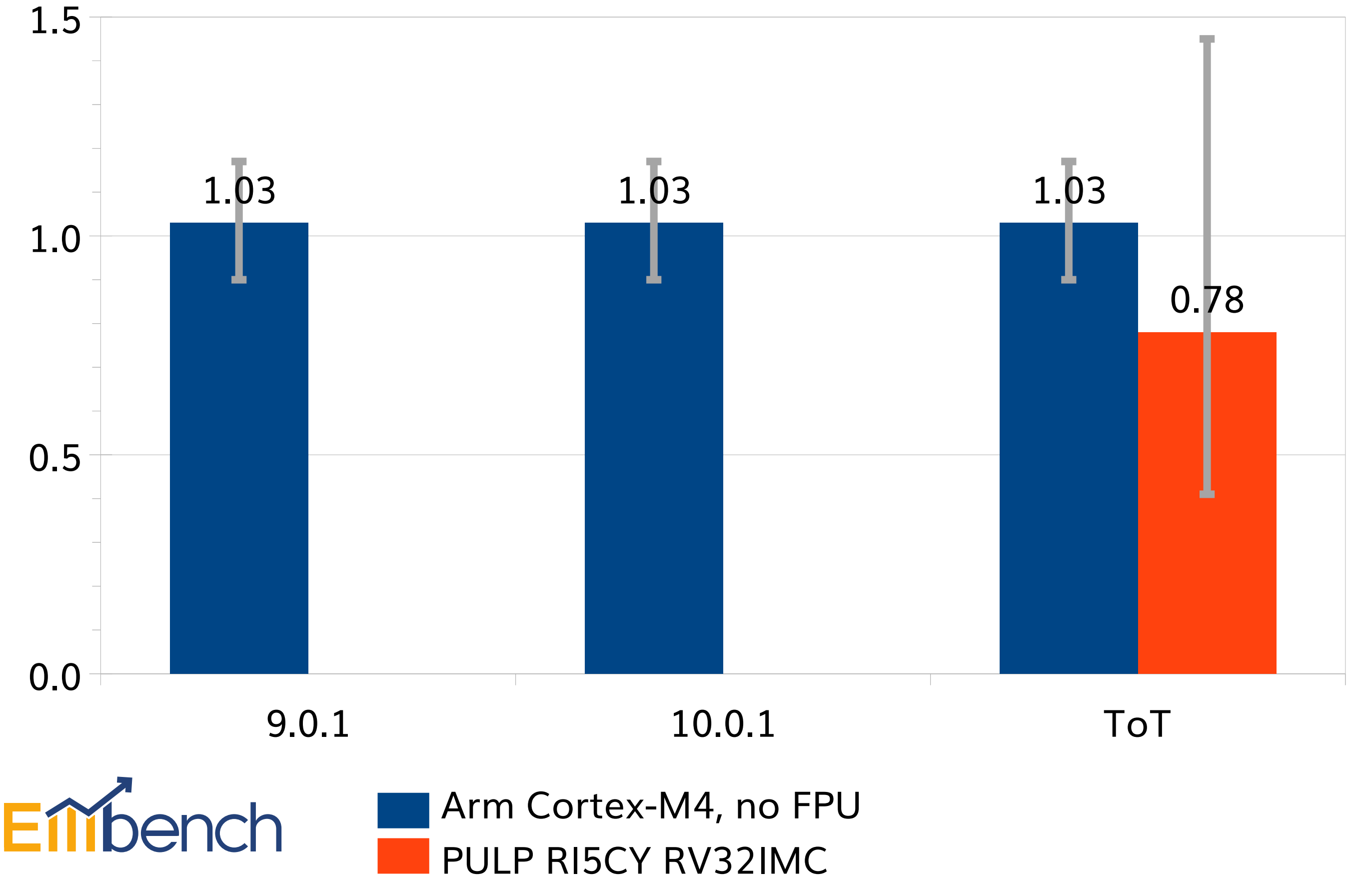
- GCC 4.9.4 - 10.2
  - higher is faster
  - **-O2**

Code Size over GCC versions



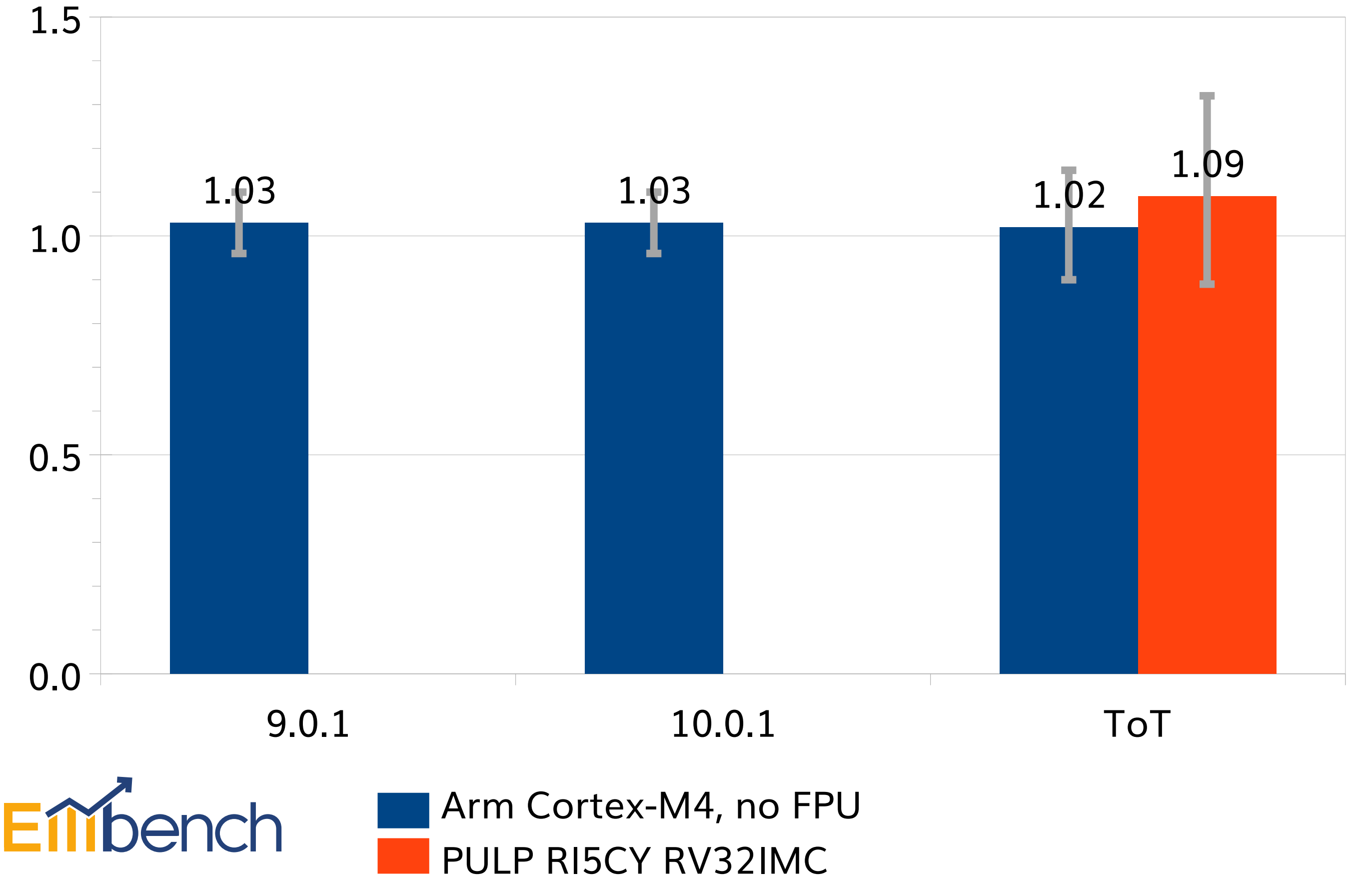
- GCC 4.9.4 - 10.2
  - lower is smaller
  - **`-Os -msave-restore`**

Code Speed over Clang/LLVM versions



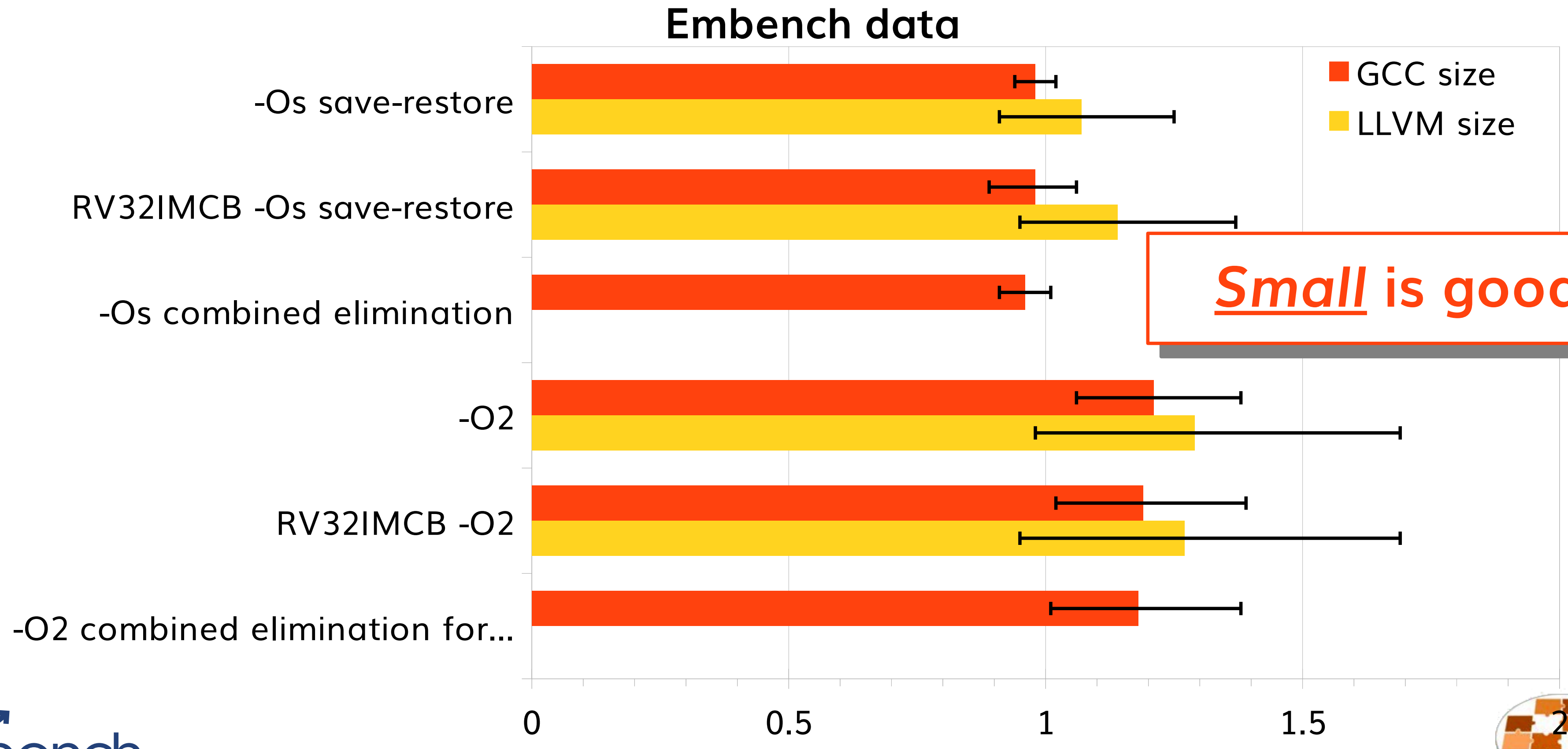
- Clang 9.0.1 - ToT
  - higher is faster
  - ToT 53ffeea6d59
  - **-O2**

Code Size over Clang/LLVM versions



- Clang 9.0.1 - ToT
  - higher is faster
  - ToT 53ffeea6d59
  - **`-Oz -msave-restore`**

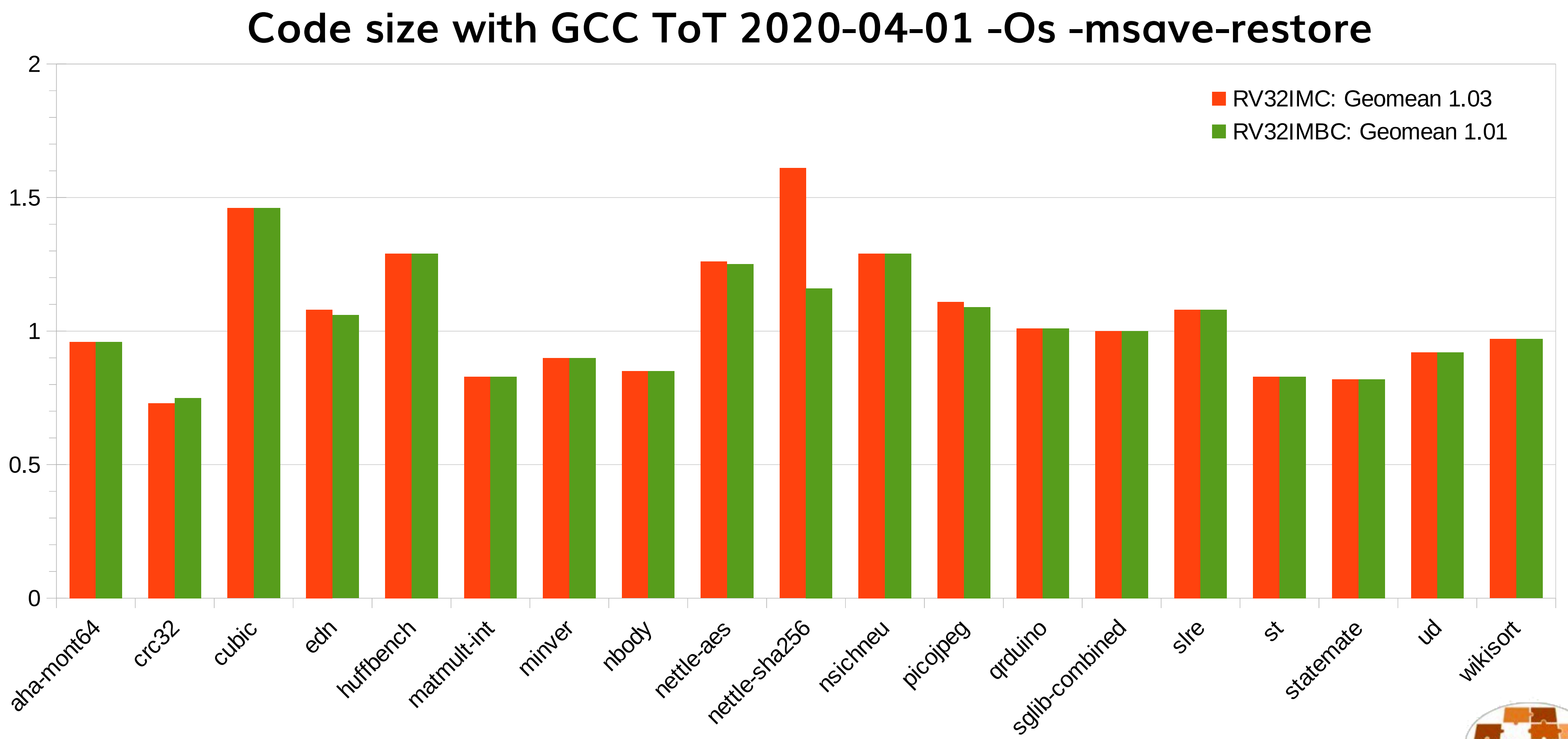
Future Size Optimization Techniques



Small is good

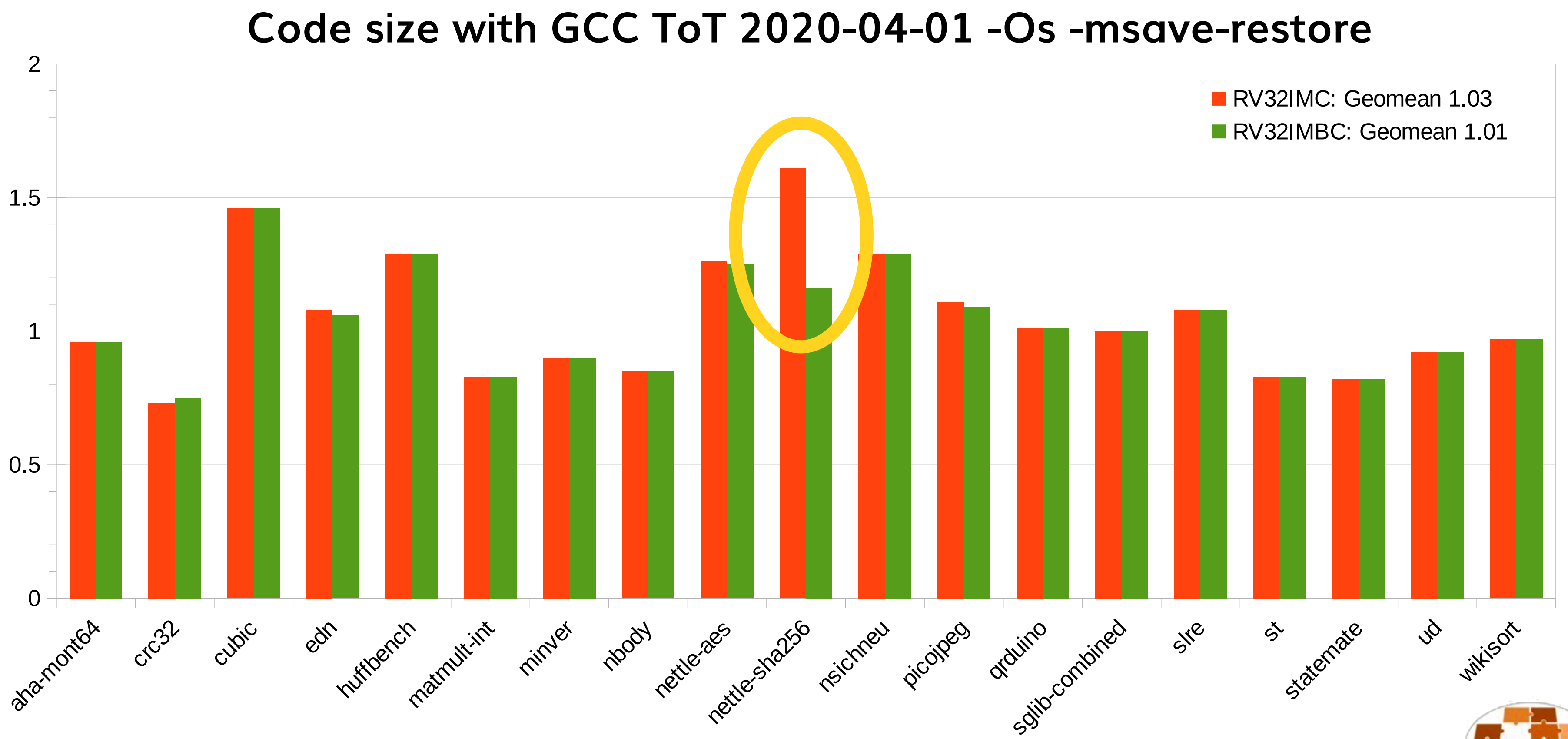
Embench

The Detail Matters: GCC Bit Manipulation



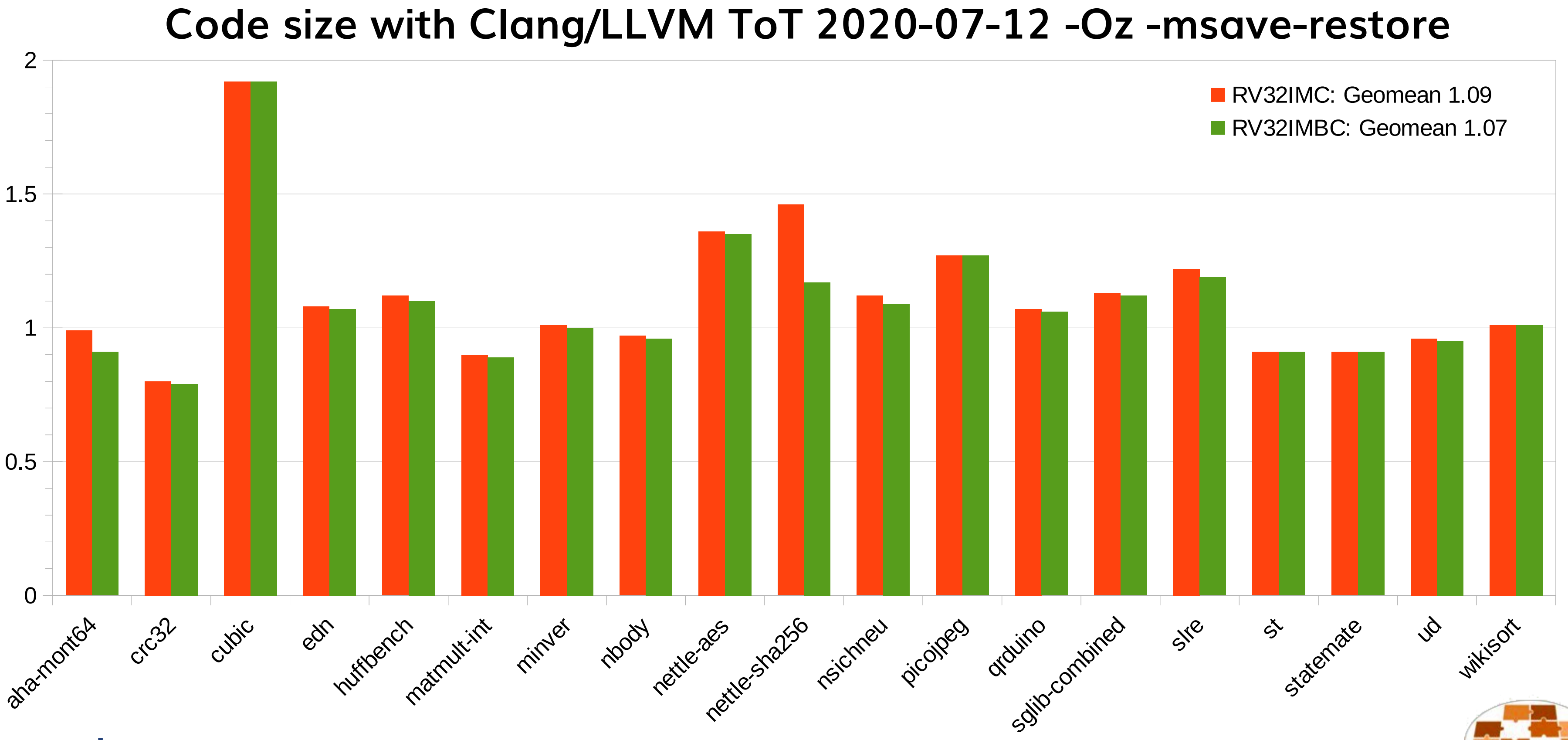
Embench

The Detail Matters: GCC Bit Manipulation

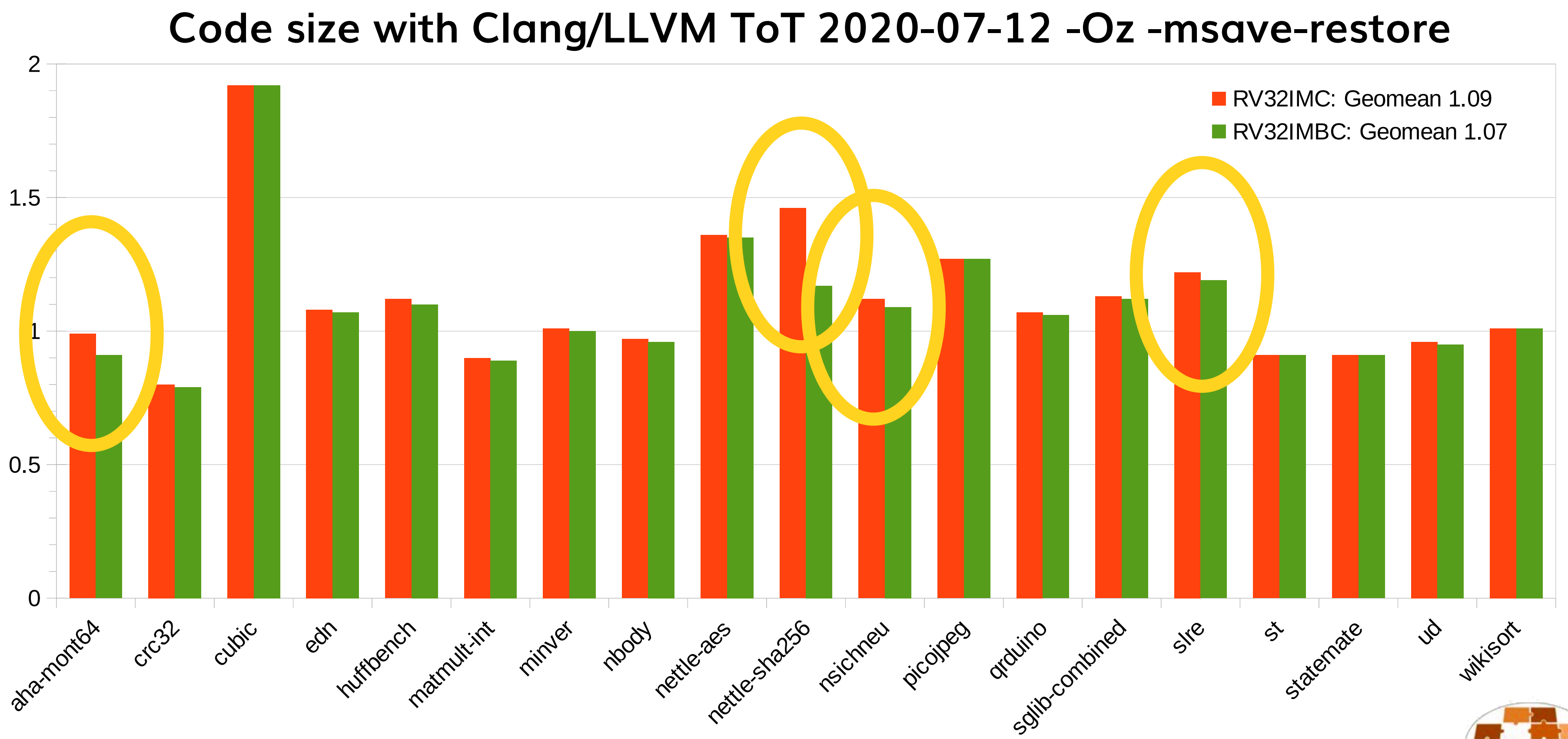


Highlighted > 2% benefit, best win is 28%

The Detail Matters: Clang/LLVM Bit Manipulation



The Detail Matters: Clang/LLVM Bit Manipulation



Embench

Highlighted > 2% benefit, best win is 20%

Bit Manipulation Lessons?

1) GCC only wins on one benchmark (but wins big)
2) Clang/LLVM wins on four benchmarks, but none as big as GCC
3) GCC needs to match more optimization patterns
4) Clang/LLVM could do more where it does match

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 for larger machines in Embench 0.6
- Published results in embench-iot-results repository
- Want to help? Email info@embench.org
-

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%
- ARM Thumb2 smaller than RV32IMC, but within one standard deviation
- RISC-V GCC improved to shrink code size by 4% in last 12 months
- Many more studies: more ISAs, more compilers, more processors, ...
- We believe Embench 0.5 suite is already an improvement over single synthetic programs Dhrystone and CoreMark, and will get better
- Let us know if you'd like to help: Email info@embench.org
-