A New Golden Age for Computer Architecture:

Processor Innovation to Enable Ubiquitous AI

David Patterson
UC Berkeley and Google

Lessons of last 50 years of Computer Architecture

1. *Raising the hardware/software interface creates opportunities for architecture innovation*
   - *e.g., C, Python, TensorFlow, PyTorch*
2. *Ultimately benchmarks and the marketplace settles architecture debates*
   - *e.g., SPEC, TPC, MLPerf, ...*

Instruction Set Architecture?

- Software talks to hardware using a **vocabulary**
  - Words called ***instructions***
  - Vocabulary called ***instruction set architecture (ISA)***

- Most important interface since determines software that can run on hardware
  - Software is distributed as instructions

IBM Compatibility Problem in Early 1960s

By early 1960's, *IBM had 4 incompatible lines of computers!*

701 ➡ 7094

650 ➡ 7074

702 ➡ 7080

1401 ➡ 7010

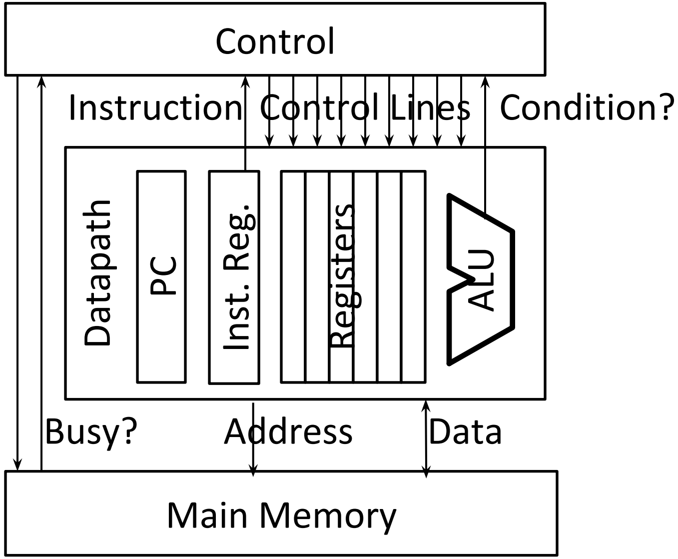
Each system had its own:

- Instruction set architecture (ISA)
- I/O system and Secondary Storage:
  magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche: business, scientific, real time, ...

IBM System/360 – one ISA to rule them all

Control versus Datapath

- Processor designs split between *datapath*, where numbers are stored and arithmetic operations computed, and *control*, which sequences operations on datapath
- Biggest challenge for computer designers was getting control correct

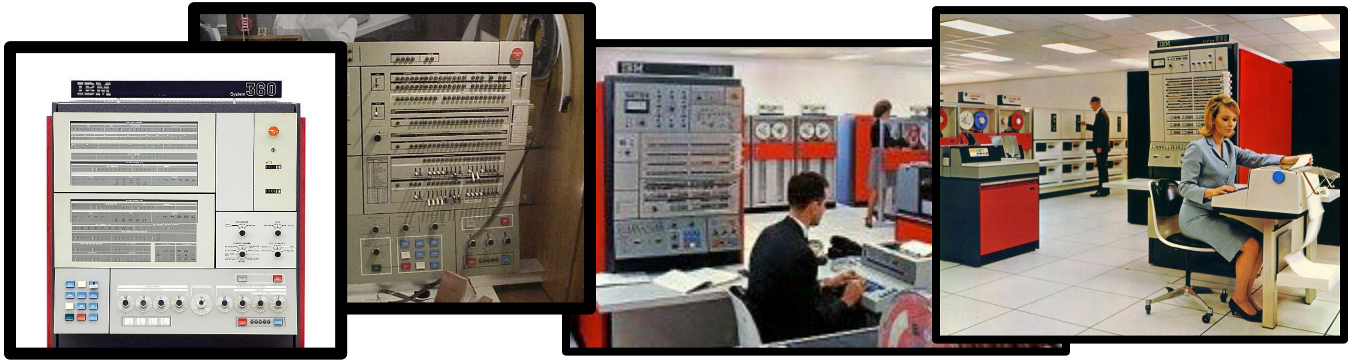


- **Maurice Wilkes** invented the idea of *microprogramming* to design the control unit of a processor*
  - Logic expensive vs. ROM or RAM
  - ROM cheaper and faster than RAM
  - *Control design now programming*

* "Micro-programming and the design of the control circuits in an electronic digital computer," M. Wilkes, and J. Stringer. *Mathematical Proc. of the Cambridge Philosophical Society,* Vol. 49, 1953.

Microprogramming in IBM 360

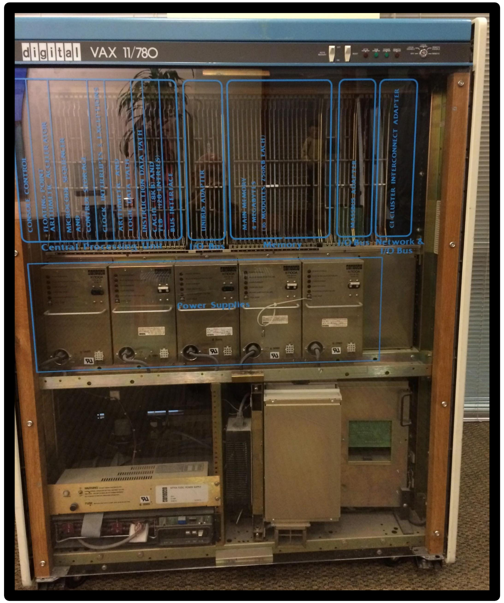
Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Microcode size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock cycle time (ROM)	750 ns	625 ns	500 ns	200 ns
Main memory cycle time	1500 ns	2500 ns	2000 ns	750 ns
Price (1964 $)	$192,000	$216,000	$460,000	$1,080,000
Price (2018 $)	$1,560,000	$1,760,000	$3,720,000	$8,720,000



Fred Brooks, Jr.

IC Technology, Microcode, and CISC

- Logic, RAM, ROM all implemented using same transistors
- Semiconductor RAM ≈ same speed as ROM
- With Moore's Law, memory for control store could grow
- Since RAM, easier to fix microcode bugs
- Allowed more complicated ISAs (CISC)
- Minicomputer (TTL server) example:
  - -Digital Equipment Corp. (DEC)
  - -VAX ISA in 1977
- 5K x 96b microcode

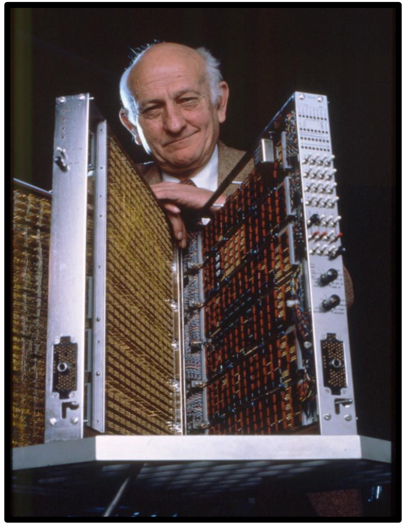


Microprocessor Evolution

- Rapid progress in 1970s, fueled by advances in MOS technology, imitated minicomputers and mainframe ISAs
- "Microprocessor Wars": compete by adding instructions (easy for microcode), justified given assembly language programming
- Intel iAPX 432: Most ambitious 1970s micro, started in 1975
  - 32-bit capability-based, object-oriented architecture, custom OS written in Ada
  - Severe performance, complexity (multiple chips), and usability problems; announced 1981
- Intel 8086 (1978, 8MHz, 29,000 transistors)
  - "Stopgap" 16-bit processor, 52 weeks to new chip
  - ISA architected in 3 weeks (10 person weeks) assembly-compatible with 8 bit 8080
- IBM PC 1981 picks Intel 8088 for 8-bit bus (and Motorola 68000 was late)
- Estimated PC sales: 250,000
- Actual PC sales: 100,000,000 ⇒ 8086 "overnight" success
- Binary compatibility of PC software ⇒ bright future for 8086

Analyzing Microcoded Machines 1980s

- *HW/SW interface rises from assembly to HLL programming*
  - Compilers now source of measurements
- John Cocke group at IBM
  - Worked on a simple pipelined processor, 801 minicomputer (ECL server), and advanced compilers inside IBM
  - Ported their compiler to IBM 370, only used simple register-register and load/store instructions (similar to 801)
  - Up to 3X faster than existing compilers that used full 370 ISA!
- Emer and Clark at DEC in early 1980s*
  - Found VAX 11/780 average clock cycles per instruction (CPI) = 10!
  - Found 20% of VAX ISA ⇒ 60% of microcode, but only 0.2% of execution time!



John Cocke

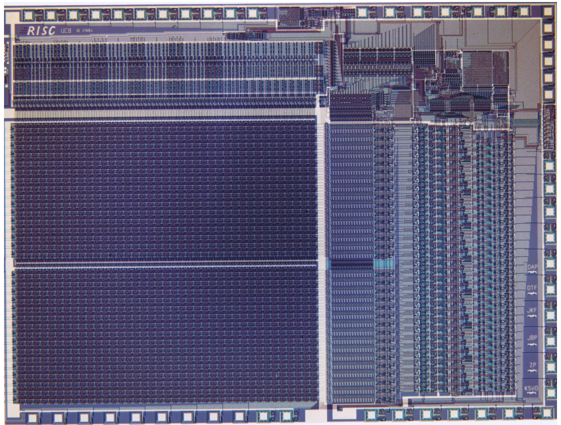
* "A Characterization of Processor Performance in the VAX-11/780," J. Emer and D.Clark, *ISCA*, 1984.

From CISC to RISC

- Use RAM for instruction *cache* of user-visible instructions
  - *Software concept: Compiler vs. Interpreter*
  - Contents of fast instruction memory change to what application needs now vs. ISA interpreter
- Use simple ISA
  - Instructions as simple as microinstructions, but not as wide
  - Enable pipelined implementations
  - Compiled code only used a few CISC instructions anyways
- Chaitin's register allocation scheme* benefits load-store ISAs

*Chaitin, Gregory J., et al. "Register allocation via coloring." *Computer languages* 6.1 (1981), 47-57.

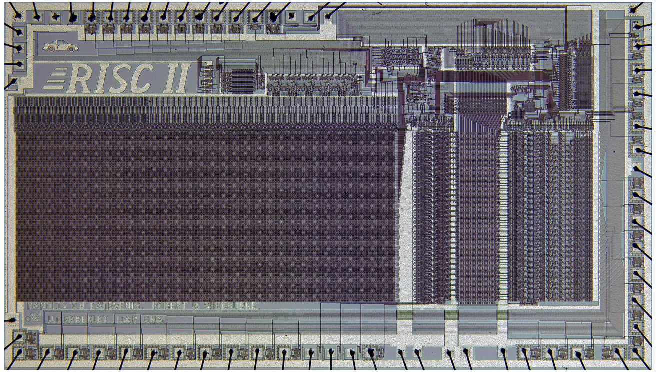
Berkeley and Stanford RISC Chips



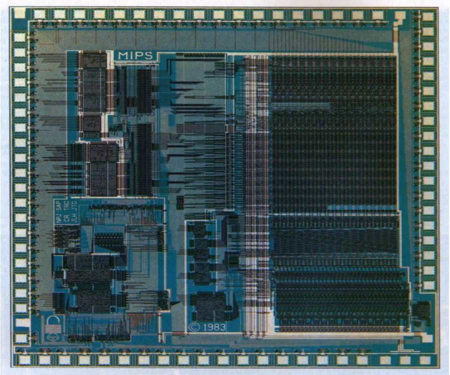
RISC-I (1982) Contains 44,420 transistors, fabbed in 5 µm NMOS, with a die area of 77 mm^2, ran at 1 MHz

Fitzpatrick, Daniel, John Foderaro, Manolis Katevenis, Howard Landman, David Patterson, James Peek, Zvi Peshkess, Carlo Séquin, Robert Sherburne, and Korbin Van Dyke. "A RISCy approach to VLSI." *ACM SIGARCH Computer Architecture News* 10, no. 1 (1982)

Hennessy, John, Norman Jouppi, Steven Przybylski, Christopher Rowen, Thomas Gross, Forest Baskett, and John Gill. "MIPS: A microprocessor architecture." In *ACM SIGMICRO Newsletter*, vol. 13, no. 4, (1982).



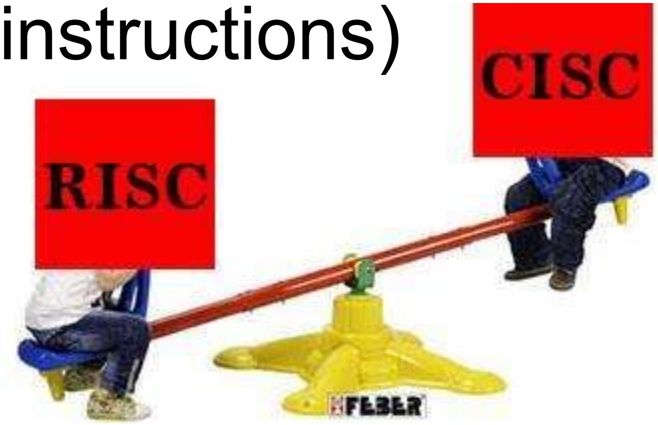
RISC-II (1983) contains 40,760 transistors, was fabbed in 3 µm NMOS, ran at 3 MHz, and the size is 60 mm^2



Stanford MIPS (1983) contains 25,000 transistors, was fabbed in 3 µm & 4 µm NMOS, ran at 4 MHz (3 µm), and size is 50 mm^2 (4 µm) (Microprocessor without Interlocked Pipeline Stages)

Reduced Instruction Set Computer?

- *Reduced Instruction Set Computer (RISC)* vocabulary uses simple words (instructions)
- RISC reads 25% more instructions since simple vs. Complex Instruction Set Computer (CISC) e.g., Intel 80x86
- But RISC reads them 5 times faster
- Net is 4 times faster

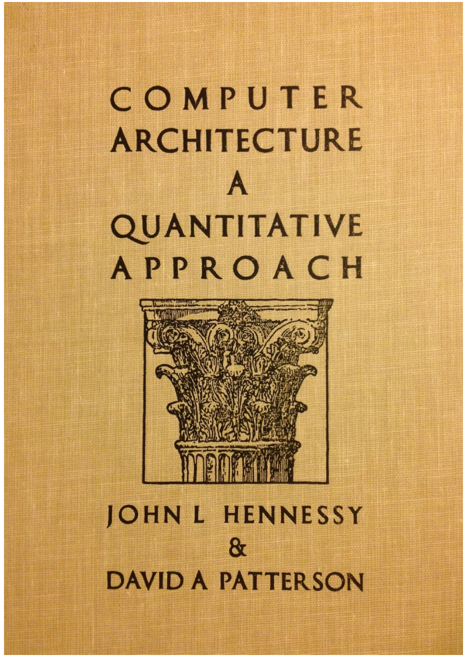


“Iron Law” of Processor Performance: How RISC can win

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Clock cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Clock cycle}}$$

- CISC executes fewer instructions / program (≈ 3/4X instructions) but many more clock cycles per instruction (≈ 6X CPI)
  ⇒ RISC ≈ 4X faster than CISC

“Performance from architecture: comparing a RISC and a CISC with similar hardware organization,” Dileep Bhandarkar and Douglas Clark, *Proc. Symposium, ASPLOS*, 1991.



How to Measure Performance?

- Instruction rate (MIPS, millions of instructions per second)
  - + Easy to understand, bigger is better
  - − But can’t compare different ISAs, higher MIPS can be slower
- Time to run toy program (puzzle)
  - + Can compare different ISAs, shorter time always faster
  - − But not representative of real programs
- Synthetic programs (Whetstone, Dhrystone)
  - + Tries to match characteristics of real programs
  - − Compilers can remove most code, less realistic over time
- Benchmark suite relative to reference computer (SPEC)
  - + Real programs, bigger is better, geometric mean fair
  - − Must update every 2-3 years to stay uptodate ⇒ organization

CISC vs. RISC Today

PC Era

- Hardware translates x86 instructions into internal RISC instructions *(Compiler vs Interpreter)*
- Then use any RISC technique inside MPU
- > 350M / year !
- x86 ISA eventually dominates servers as well as desktops

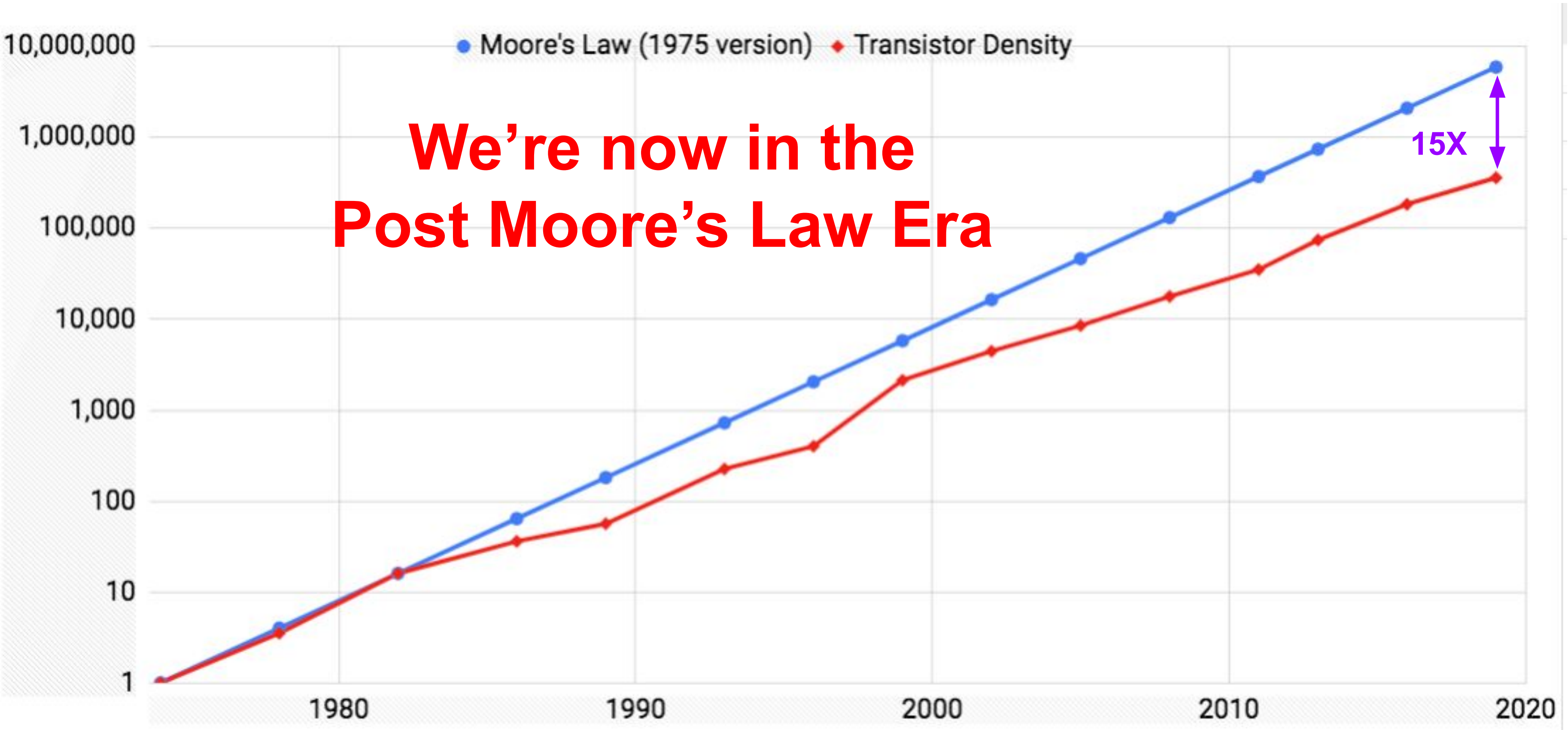
PostPC Era: Client/Cloud

- IP in SoC vs. MPU
- Value die area, energy as much as performance
- > 20B total / year in 2017
- 99% Processors today are RISC
- *Marketplace settles debate*

Lessons from RISC vs CISC

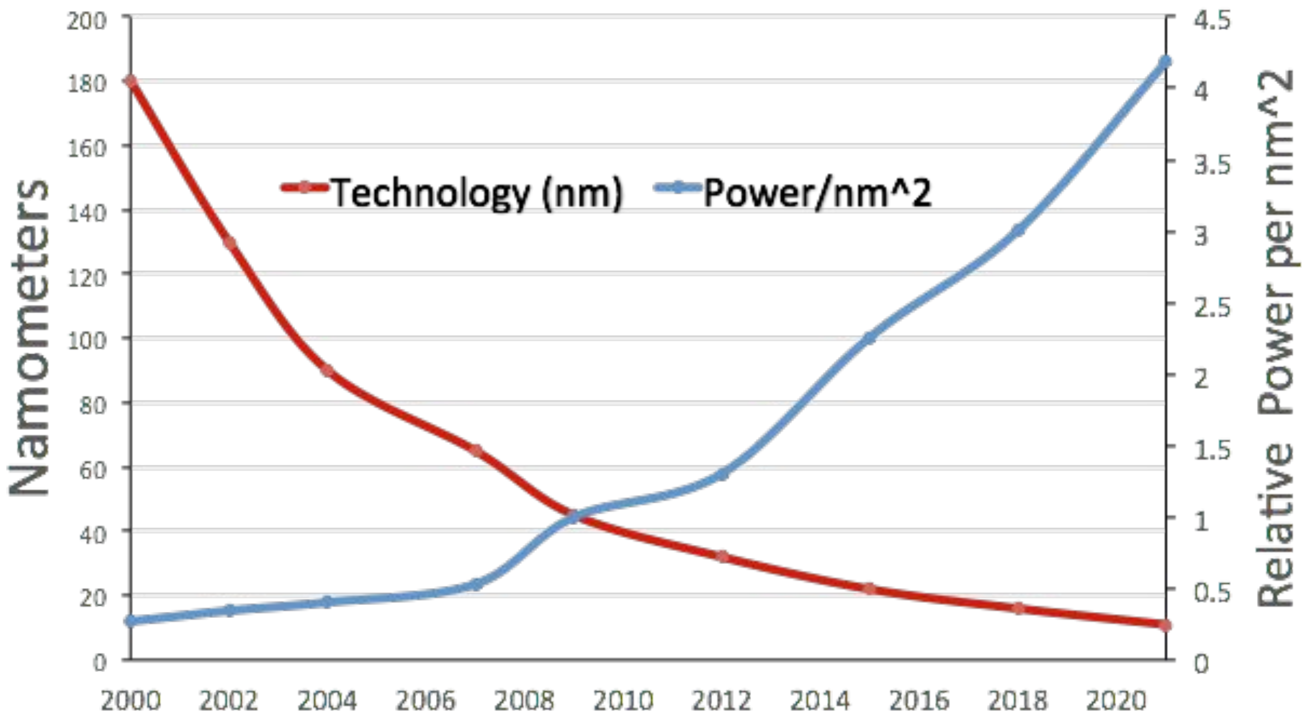
- Less is More
  - It’s harder to come up with simple solutions, but they accelerate progress
- Importance of the software stack vs the hardware
  - If compiler can’t generate it, who cares?
- Importance of good benchmarks
  - Hard to make progress if you can’t measure it
  - For better or for worse, benchmarks shape a field
- Take the time for a quantitative approach vs rely on intuition to start quickly

Moore's Law Slowdown in Intel Processors



Moore, Gordon E. "No exponential is forever: but 'Forever' can be delayed!" *Solid-State Circuits Conference, 2003.*

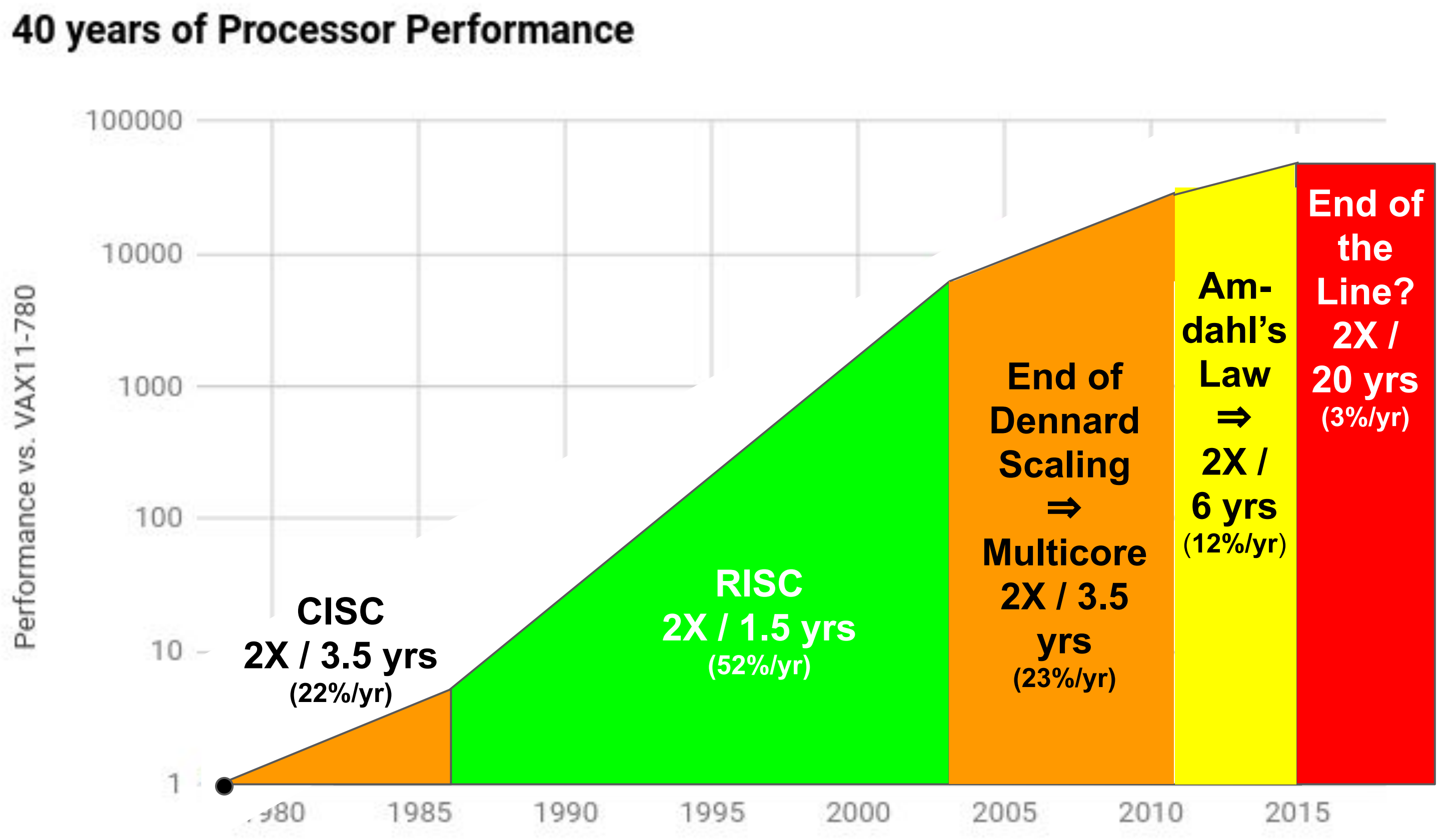
Technology & Power: Dennard Scaling



Power consumption based on models in "Dark Silicon and the End of Multicore Scaling," Hadi Esmaelizadeh, *ISCA*, 2011

Energy scaling for fixed task is better, since more and faster transistors

End of Growth of Single Program Speed?



Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

Domain Specific Architectures (DSAs)

- Achieve higher efficiency by tailoring the architecture to characteristics of the domain

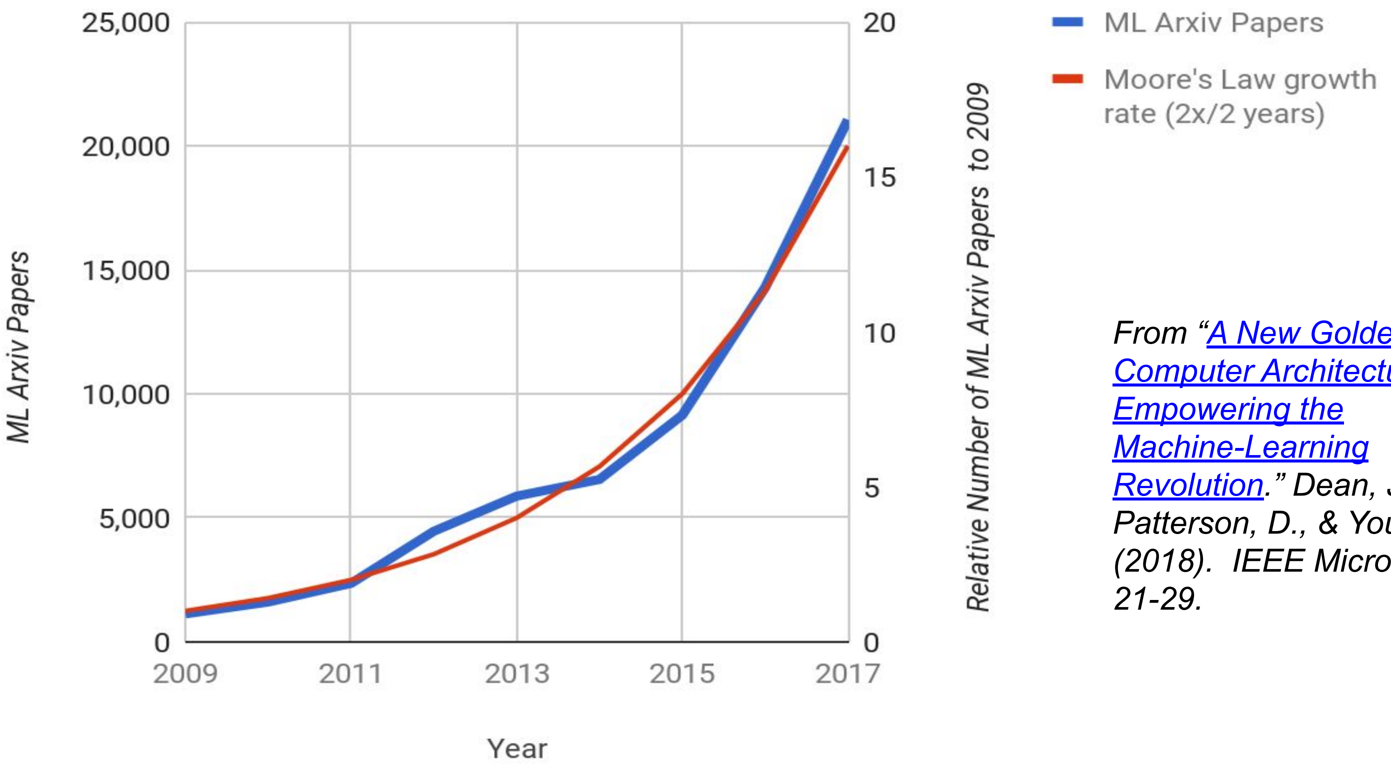
- Not one application, but a domain of applications

- Different from strict ASIC since still runs software

Why DSAs Can Win (no magic) Tailor the Architecture to the Domain

- More effective parallelism for a specific domain:
  - SIMD vs. MIMD
  - VLIW vs. Speculative, out-of-order
- More effective use of memory bandwidth
  - User controlled versus caches
- Eliminate unneeded accuracy
  - IEEE replaced by lower precision FP
  - 32-64 bit integers to 8-16 bit integers
- Domain specific programming language provides path for software

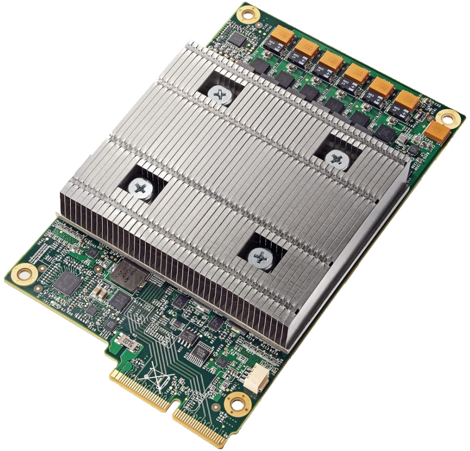
Deep learning is causing a machine learning revolution



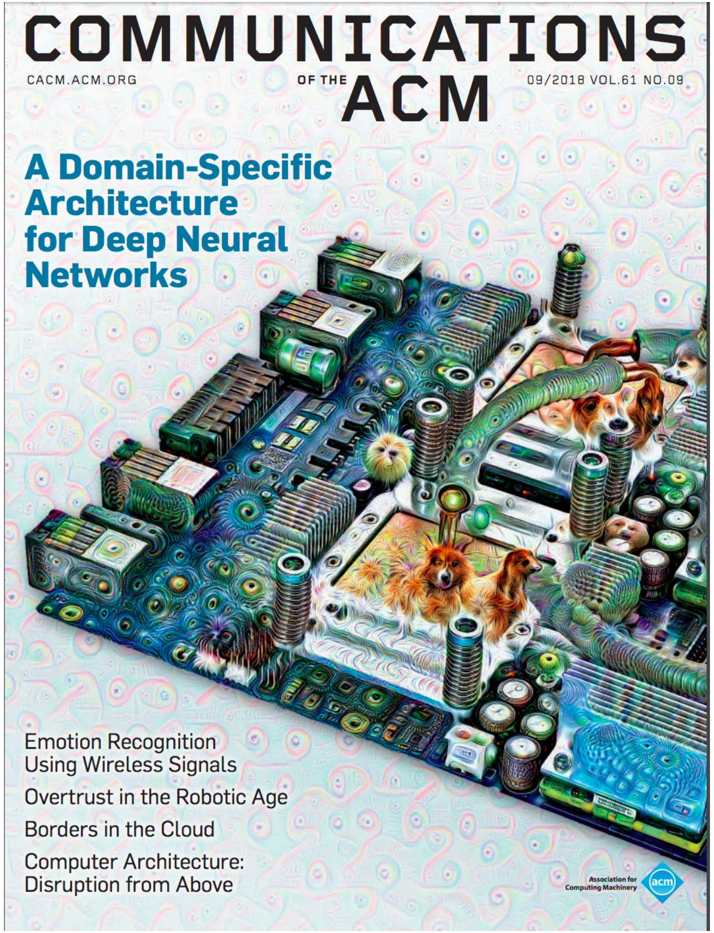
From "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution." Dean, J., Patterson, D., & Young, C. (2018). IEEE Micro, 38(2), 21-29.

Tensor Processing Unit v1 (Announced May 2016)

Google-designed chip for neural net **inference**



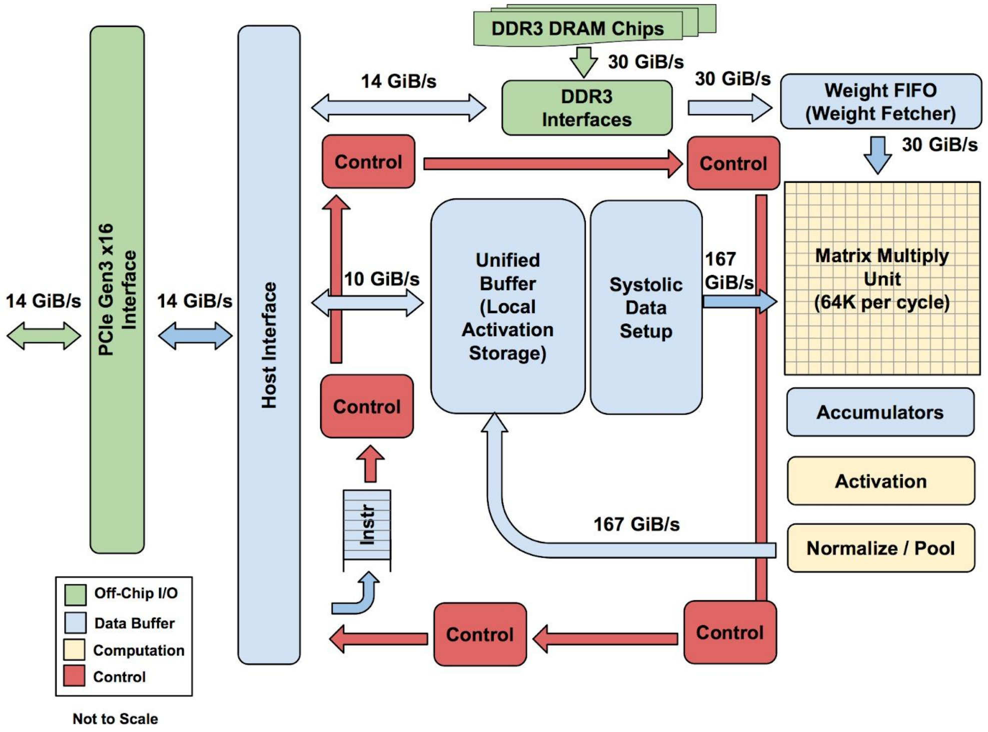
In production use for 3 years: used by billions on search queries, for neural machine translation, for AlphaGo match, …



A Domain-Specific Architecture for Deep Neural Networks, Jouppi, Young, Patil, Patterson, *Communications of the ACM*, September 2018

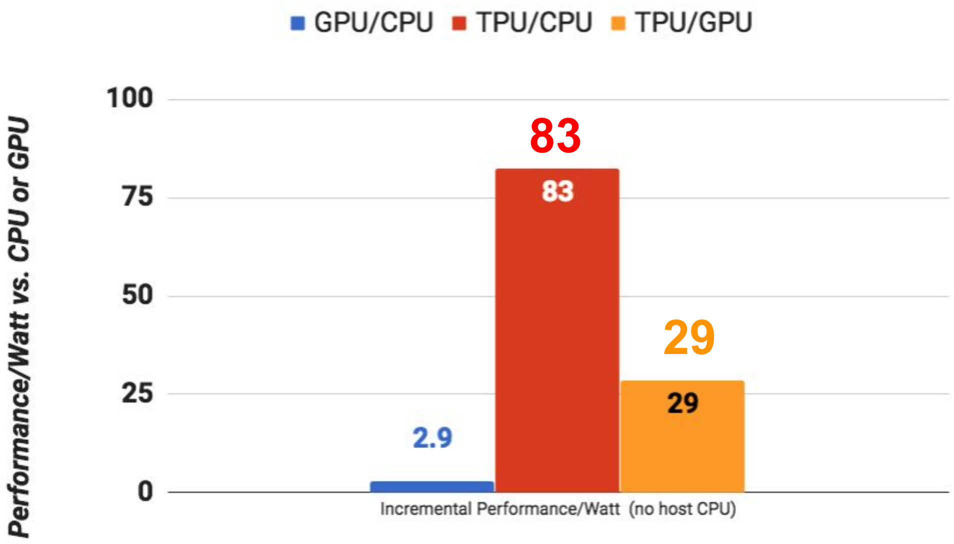
TPU: High-level Chip Architecture

- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - 65,536 * 2 * 700M
- >25X as many MACs vs GPU
- >100X as many MACs vs CPU
- 4 MiB of on-chip Accumulator memory + 24 MiB of on-chip Unified Buffer (activation memory)
- 3.5X as much on-chip memory vs GPU
- 8 GiB of off-chip weight DRAM memory



Perf/Watt TPU vs CPU & GPU

Using production applications vs contemporary CPU and GPU



Reasons for TPUv1 Success

Two dimensional arithmetic unit with 64,000 multiplier/accumulators (256x256)

⇒ faster matrix multiplies for neural networks

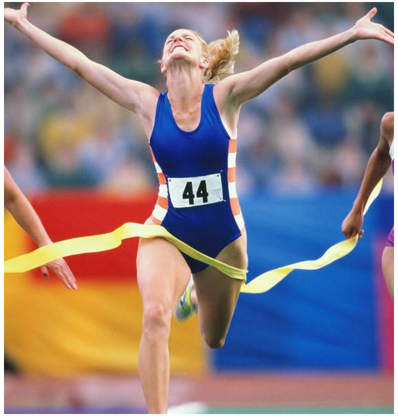
8-bit Integer data vs 32-bit Floating-Point data

⇒ more efficient computation & memory

TPUv1 drops general purpose CPU/GPU features (e.g., caches, branch predictors)

⇒ saves area & energy

⇒ reuse transistors for domain-specific hardware



The Launching of “1000 Chips”

- Intel acquires DSA chip companies
  - Nervana: ($0.4B) August 2016
  - Movidius: ($0.4B) September 2016
  - MobilEye: ($15.3B) March 2017
  - Habana: ($2.0B) December 2019
- Alibaba, Amazon inference chips
- >100 startups ($2B) launch on own bets
  - Dataflow architecture: Graphcore, ...
  - Asynchronous logic: Wave Computing, ...
  - Analog computing: Mythic, …
  - Wafer Scale computer: Cerebras
  - Coarse-Grained Reconfigurable Arch: SambaNova, ...

Helen of Troy by Evelyn De Morgan

How to Measure ML Performance?

- Operation rate (GOPS, billions of operations per second)
  - Easy to understand, bigger is better
  - But peak rates not for same program
  - Operations can vary between DSAs (FP vs int, 4b/8b/16b/32b)
- Time to run old DNN (MNIST, AlexNet)
  - Can compare different ISAs, shorter time always faster
  - But not representative of today's DNNs
- Benchmark suite relative to reference computer (MLPerf)
  - Real programs, bigger is better, same DNN model, same data set, geometric mean fair comparison, batch size ranges set
  - Must update every 1-2 years to stay uptodate ⇒ organization

Embedded Computing and ML

- ML becoming one of the most important workloads
- But lots of applications don’t need highest performance
  - For many, just enough at low cost
- Microcontrollers most popular processors
  - Cheap, Low Power, fast enough for many apps
- Despite importance, no good microprocessor benchmarks
  - Still quote synthetic programs: Dhrystone, CoreMarks
- Decided to try to fix
- EmBench: better for all embedded, includes ML benchmarks also

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

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

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 ▾

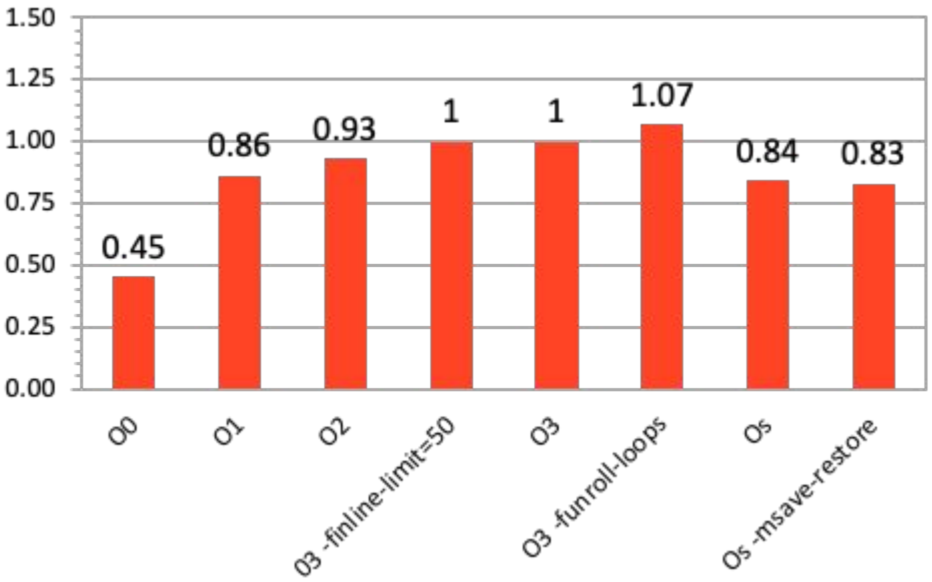
jeremybennett Note that Embench is a trademark (#28) ··· Latest commit 976679c 12 days ago

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

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

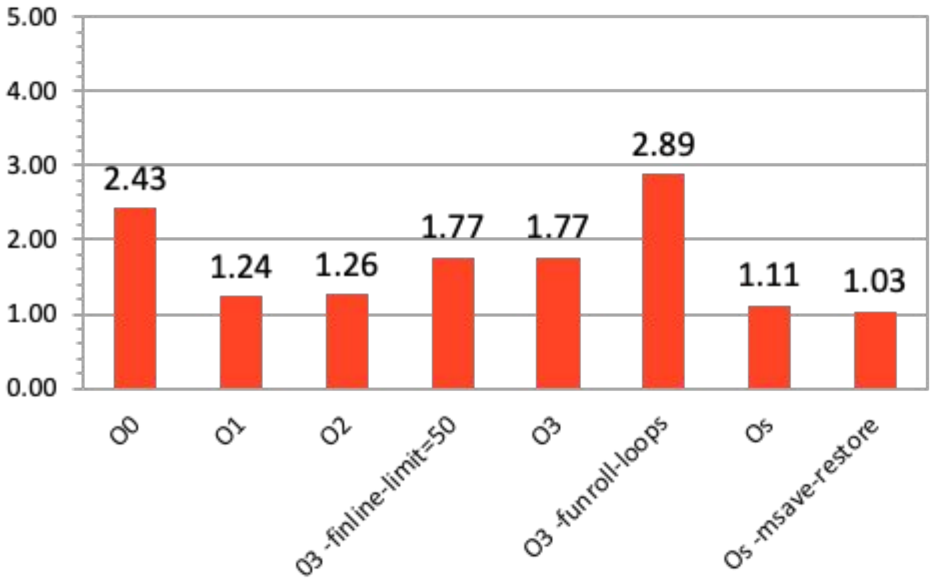
Impact of optimizations of GCC on RISC-V: Speed



PULP RI5CY RV32IMC GCC 10.1.0 (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

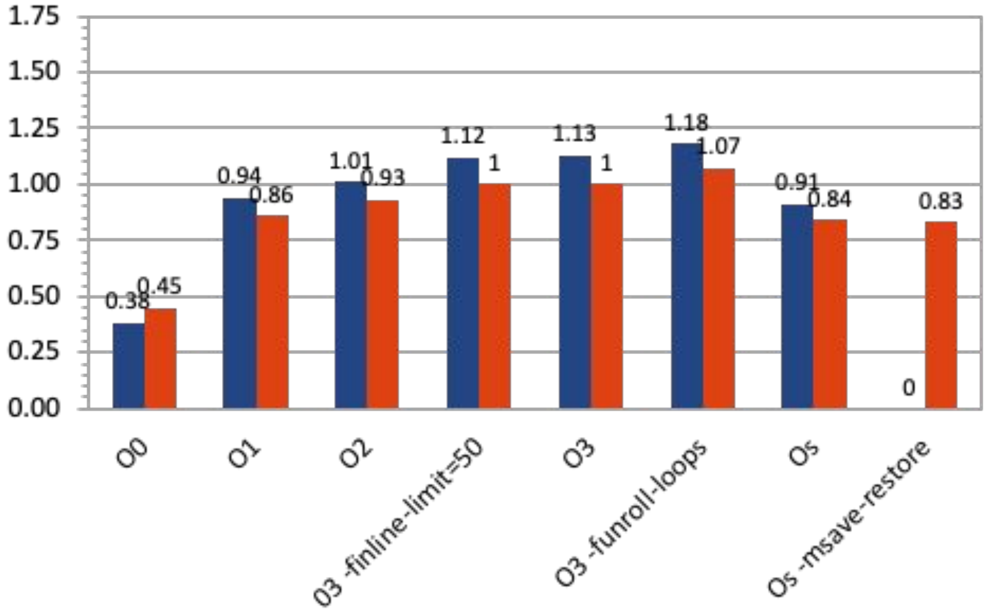
Impact of optimizations of GCC on RISC-V: Size



PULP RI5CY RV32IMC GCC 10.1.0 (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

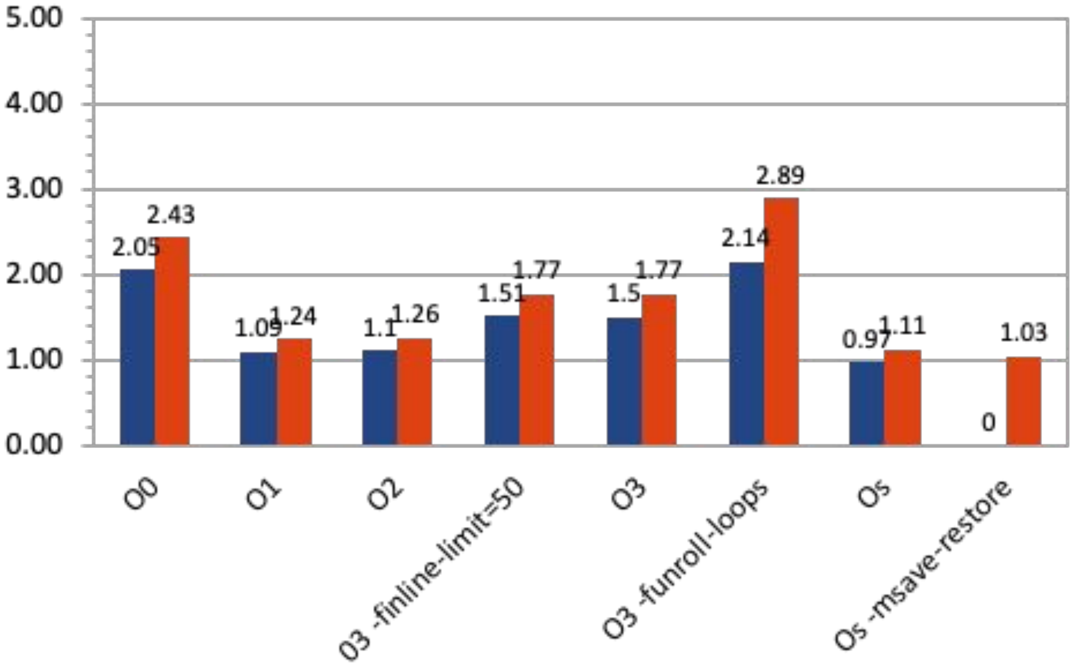
Comparing Architectures with GCC: Speed



- GCC 10.2.0
  - higher is faster

Arm Cortex-M4, no FPU
PULP RI5CY RV32IMC GCC 10.2.0 (soft core in FPGA

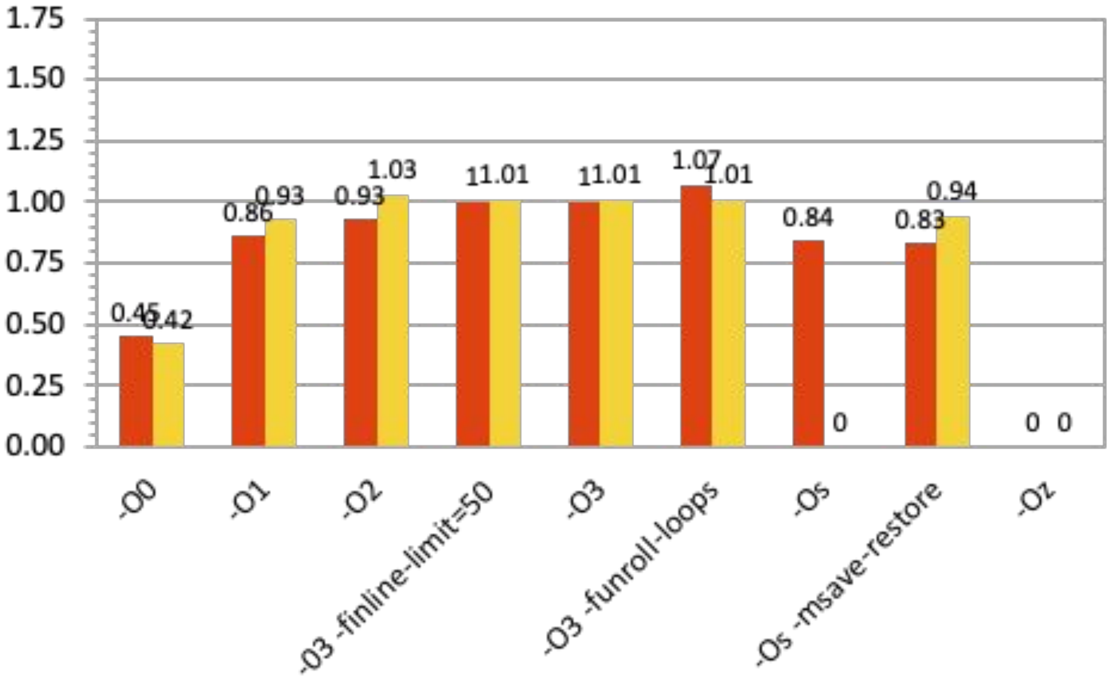
Comparing Architectures with GCC: Size



- GCC 10.2.0
  - lower is smaller

Arm Cortex-M4, no FPU
PULP RI5CY RV32IMC GCC 10.2.0 (soft core in FPGA)

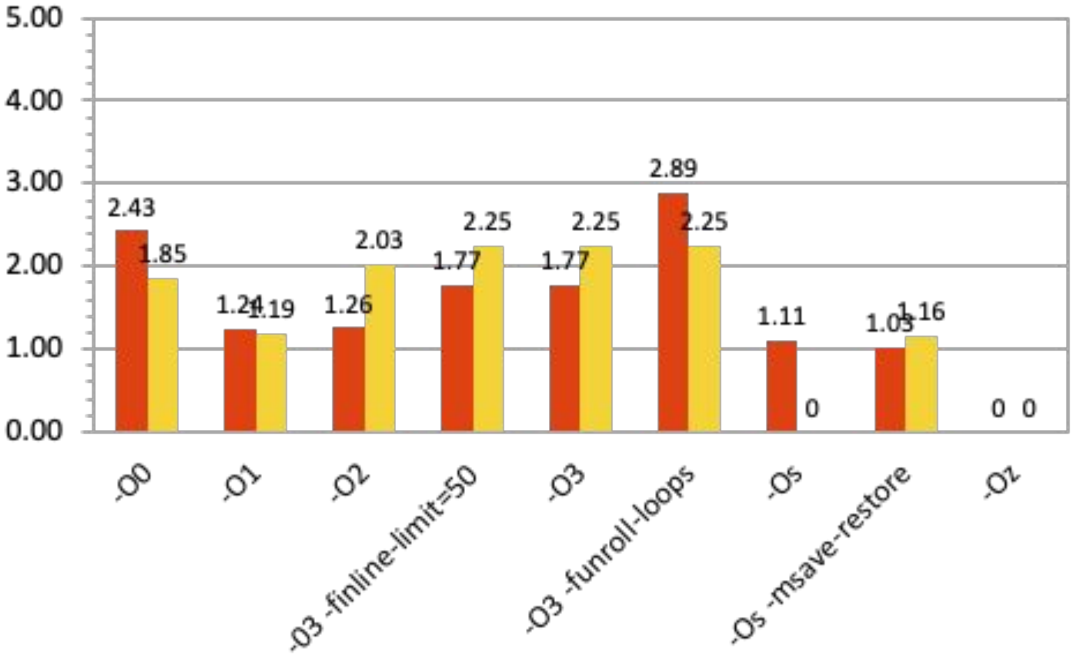
Comparing Compilers GCC v LLVM: Speed



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

GCC 10.2.0
Clang/LLVM 11.0.0 rc

Comparing Compilers GCC v LLVM: Size

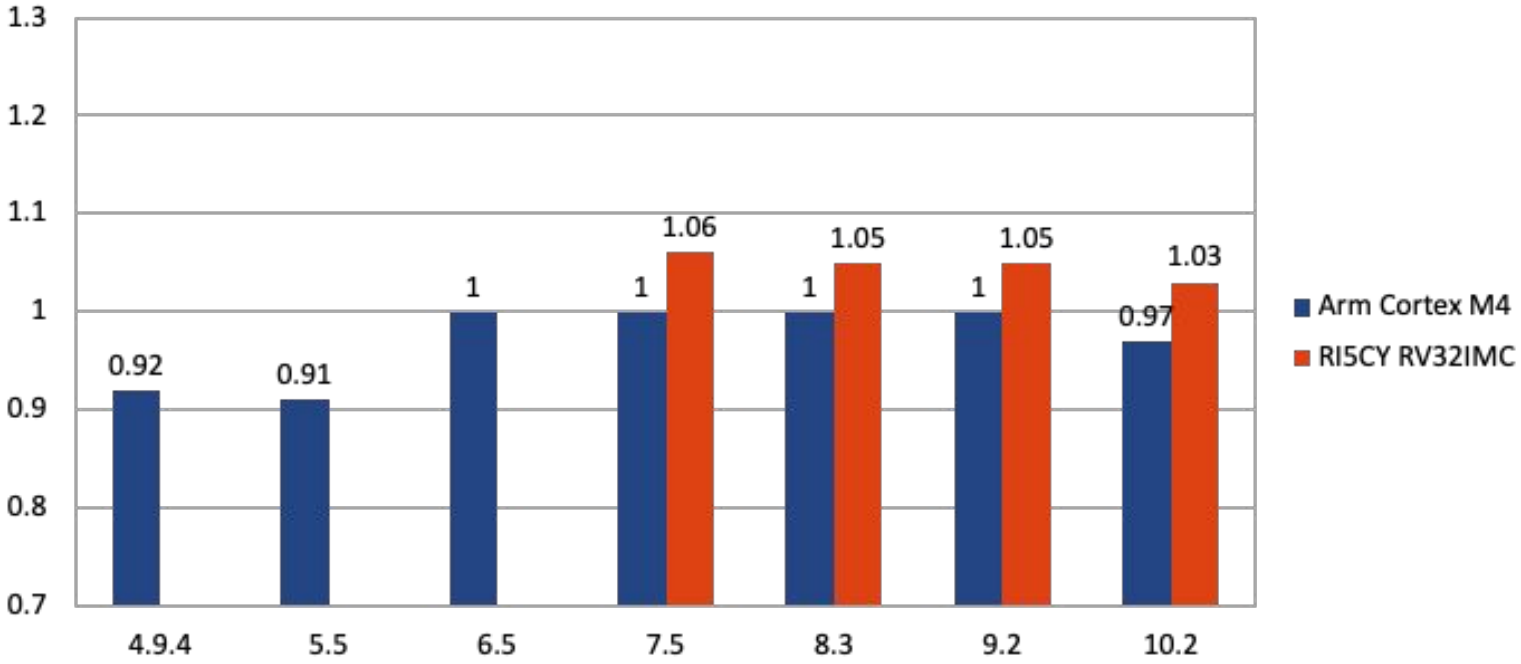


- PULP RI5CY RV32IMC
  - lower is smaller
- Clang/LLVM variations
  - **-msave-restore** enabled by default with **-Os**
  - **-Oz** for further code size optimization

Embench

GCC 10.2.0
Clang/LLVM 11.0.0 rc

Code Size over GCC versions



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

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
3) More mature architecture have more mature compilers

Conclusions

- End of Dennard Scaling, slowing of Moore's Law ⇒ DSA
- ML DSAs need HW/SW codesign
- To measure progress, need good benchmarks,
- MLPerf for data center and high end edge
- For microcontrollers, Embench 0.5 suite is already better than synthetic programs Dhrystone and CoreMark, and will get better
  - Many more studies: more ISAs, more compilers, more cores,
- Let us know if you'd like to help: Email info@embench.org