Performance Modeling and Measurement

Jaynaryan T Tudu

Indian Institute of Technology Tirupati, India

21^{st} January, 2020
CS5202-Computer System Architecture

Computer Architecture: The Challenges

- Exercise in engineering trade-off analysis
  - Find the fastest/cheapest/power-efficient/etc. solution
  - Optimization problem with 100s of variables

Performance (better) | Cost (lesser) | Power (lesser)

- All the variables are changing
  - At non-uniform rates
  - With inflection points
- Two high-level effects:
  - Technology push
  - Application pull

Performance Growth: 1978 to 2018

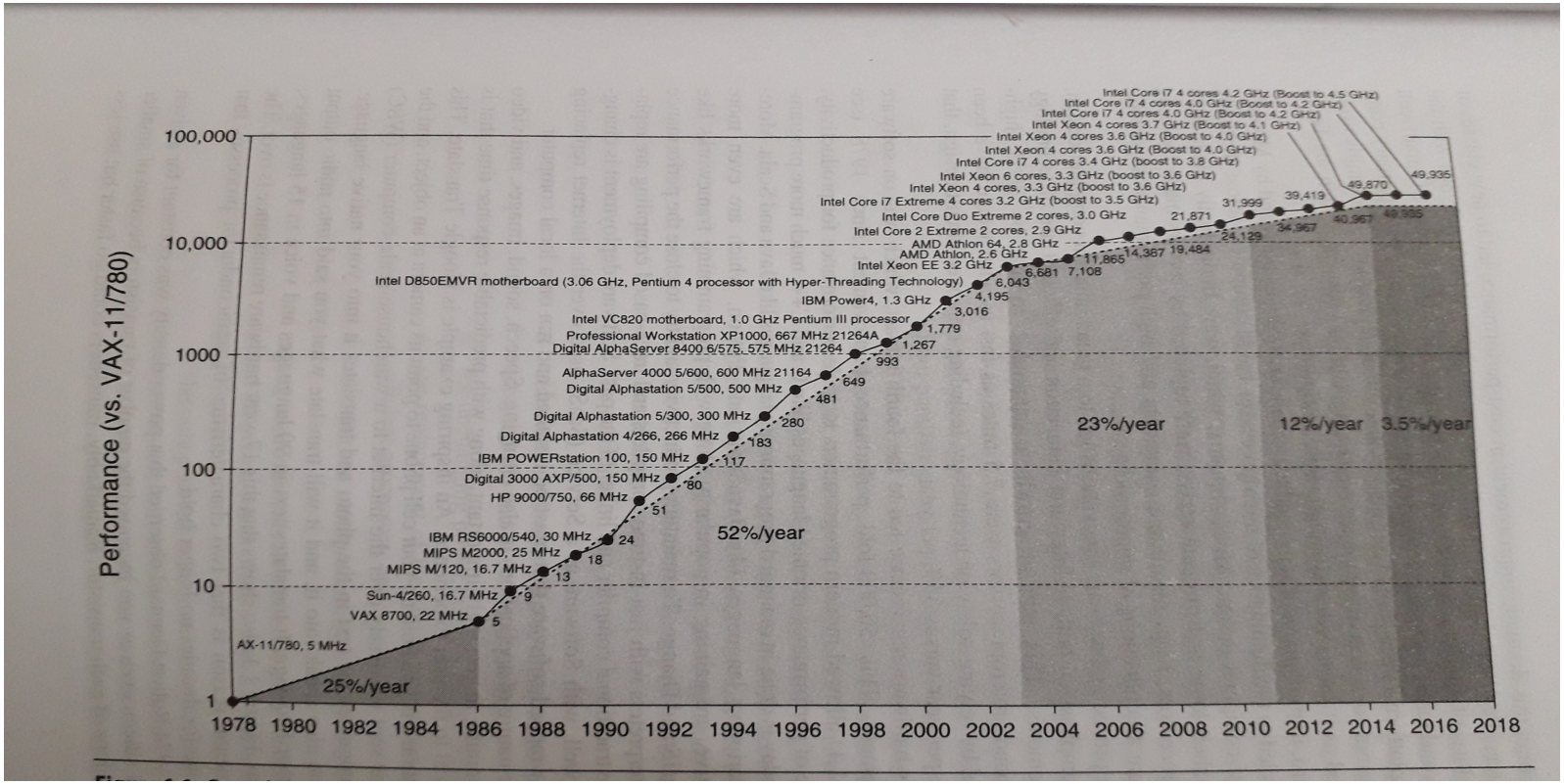

Figure : Growth of performance over last 40 years

- 1978 to 1986: Doubling every 3.5 years
- 1988 to 2002: Doubling every 2 years
- 2002 to 2010: Doubling every 4 years
- 2010 to 2014: Doubling every 8 years
- 2014 to 2018: Doubling every 20 years

Performance vs Design Time

-

Better performance need more design time
- Time-to-market is critically important

Example:

- Let a new design may take 3 years
- It will be 3 times faster
- But if technology improves 50%/year
- In 3 years $1.5^3 = 3.38$
- So the new design is worse! (unless it also employs new technology)

Performance and Cost

Why do a computer architecture/designer need cost analysis?
Ans: Whether to include the new feature for performance or not?

Cost

Three main component of cost analysis:

- Cost of manufacturing
- Cost of purchasing (Price)
- Cost of operation

Cost of Manufacturing:

$$Cost\ of\ an\ IC = \frac{Cost\ of\ die + Cost\ of\ die\ test + Cost\ of\ Packaging\ and\ Manufacturing\ test}{Final\ test\ yield}$$

Performance vs Cost

- Corollary to Moore's Law (the transistor density double every 18 months):
  - Cost halves every two years
- Computers cost-effective for
  - National security : weapons design
  - Enterprise computing : banking
  - Departmental computing : computer-aided design
  - Personal computer : spreadsheets,email, web
  - Pervasive computing : prescription drug labels

Performance Measurement

- Which computer is fastest?
- The answer is not so simple
  - Scientific simulation - FP performance
  - Program development - Integer performance
  - Database workload - Memory, I/O

Performance Measurement of Computer

- Want to buy the fastest computer for what you want to do?
  - Workload is all-important
  - Correct measurement and analysis
- Want to design the fastest computer for what the customer wants to pay?
  - Cost is an important criterion

Defining Performance

What is important to whom?

- Computer System User:
  - Minimize elapsed time for program = time_end - time_start
  - This is called: response time
- Computer Center Manager:
  - Maximize completion rate = $\frac{\#jobs}{second}$
  - Called throughput

Defining Performance for Computer Architect

- CPU time = time spent running a program
- Intuitively, bigger should be faster, therefore: Performance = $\frac{1}{Xtime}$, where X is response time or CPU Execution time etc.
- Elapsed time = CPU time + I/O waiting

Our focus will be on CPU time.

To Improve Performance

1. Response time
2. Throughput

- Faster CPU:
  Improves both: response time and throughput
- Add more CPUs:
  Improves throughput and (perhaps response time due to less queueing)

Performance Comparison

- Machine A is n times faster than machine B iff:
  perf(A)/perf(B) = time(B)/time(A) = n
- Machine A is x% faster than machine B iff:
  perf(A)/perf(B) = time(B)/time(A) = 1 + x/100

Example

time(A) = 10s, time(B) = 15s
15/10 = 1.5 $\Rightarrow$ A is 1.5 times faster than B
15/10 = 1.5 $\Rightarrow$ A is 50% faster than B

Performance Metrics

Other than CPU time and throughput there are two important metrics to quantify the performance.

- MIPS: Millions of Instruction per Second
- MFLOPS: Millions of Floating Point Operation per Second
- MIPS = instruction count/(execution time x 10^6)
  = clock rate/(CPI x 10^6)
  Execution time = $\frac{instruction\ \ count\ \ x\ \ CPI}{Clock\ \ rate}$
- But MIPS has serious shortcomings

Limitations of MIPS: Example

- E.g. without FP hardware, an FP op may take 50 single-cycle instructions (It uses Floating point subroutine of simpler instructions)
- With FP hardware, only one 2-cycle instruction
- Considering clock-rate = 1 MHz
- Thus, adding FP Hardware:
  - CPI increases: without FP: 50/50 $\Rightarrow$ with FP: 2/1
  - Instruction/program decreases:
    without FP: 50/1 $\Rightarrow$ with FP: 1/1
  - Total execution time decreases: without FP: 50 $\Rightarrow$ with FP: 2
  - But, MIPS gets worse:
    without FP: 1 MIPS $\Rightarrow$ with FP: 0.5 MIPS

MIPS = clock rate/(CPI x 10^6)

Limitations of MIPS: Example

Give a program having following set of instructions:

Operation	Frequency	CPI
ALU Operations	43%	1
Loads	21%	2
Stores	12%	2
branches	24%	2

- Compiler 1: un-optimized with 100% instruction count
- Compiler 2: optimized with 50% reduction in ALU instruction count
- Considering clock-rate = 50 MHz (therefore: cycle time = 20ns)
- Compute the MIPS for Compiler 1 and Compiler 2?

MIPS = clock rate/(CPI x 10^6)

Limitations of MIPS: Three points

1. MIPS is dependent on instruction set. (Difficult to compare across platform)
2. MIPS varies between program on same machine (compiler effect)
3. MIPS can vary inversely to performance (the FP hardware example)

- MIPS is used to measure the peak performance and not the overall performance
- MIPS is fine on same compiler on same ISA (on two different machines)
- Example: AMD compared with Intel
- Reason is: the instruction/program remain constant (what differ is CPI and clock-rate)

Limitations of MIPS: Three points

- MFLOPS = FP ops in program/(execution time x 10^6)
- Assuming FP ops independent of compiler and ISA
  - Often safe for numeric codes: matrix size determines # of FP ops/program
  - However, not always safe:
  - Missing instructions (e.g. FP divide)
  - Optimizing compilers
- Relative MIPS and normalized MFLOPS adds to confusion

Therefor, the preferred is CPU time! The "Iron Law" of processor performance.

The Iron Law Example

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which is faster and how much?
  - Time/Program = instr/program x cycles/instr x sec/cycle
  - Time(A) = N x 2.0 x 1 = 2N
  - Time(B) = N x 1.2 x 2 = 2.4N
  - Compare: Time(B)/Time(A) = 2.4N/2N = 1.2
- So, Machine A is 20% faster than Machine B for this program

The Central Question is Which Program

- Execution time of what program?
- Best case: Execute the same set of programs on different machines
- Use of benchmarks:
  - Programs chosen to measure performance
  - Predict performance of actual workload
  - Saves effort and money
  - Representative? Honest?

The SPEC Benchmark

SPEC

SPEC: Standard Performance Evaluation Corporation

- Formed in 80s to standardise the evaluation process
- SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006, and SPEC2017 (the most recent one)

The SPEC2017

- SPEC CPU2017 has 43 benchmarks, organized into 4 suites:
  - SPECrate 2017 Integer
  - SPECspeed 2017 Integer
  - SPECrate 2017 Floating Point
  - SPECspeed 2017 Floating Point

Difference between rate and speed: compile flags; workload sizes; and run rules

Example: Compiler parallelization allowed for SPECspeed but not allowed for SPECrate

The SPEC2017: SPECrate 2017 Int

SPECrate 2017 Int	Language	KLOC	Application Area
500.perlbench_r	C	362	Perl interpreter
502.gcc_r	C	1,304	GNU C compiler
505.mcf_r	C	3	Route planning
520.omnetpp_r	C++	134	Discrete Event simulation - com-net
523.xalancbmk_r	C++	520	XML to HTML conversion via XSLT
525.x264_r	C	96	Video compression
531.deepsjeng_r	C++	10	AI: alpha-beta tree search (Chess)
541.leela_r	C++	21	AI: Monte Carlo tree search (Go)
548.exchange2_r	Fortran	1	AI: recursive solution generator (Sudoku)
557.xz_r	C	33	General data compression

Source: https://www.spec.org/cpu2017/Docs/index.html#benchmarks

The SPEC2017: SPECspeed 2017 Int

SPECspeed 2017 Int	Language	KLOC	Application Area
600.perlbench_s	C	362	Perl interpreter
602.gcc_s	C	1,304	GNU C compiler
605.mcf_s	C	3	Route planning
620.omnetpp_s	C++	134	Discrete Event simulation - com-net
623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
625.x264_s	C	96	Video compression
631.deepsjeng_s	C++	10	AI: alpha-beta tree search (Chess)
641.leela_s	C++	21	AI: Monte Carlo tree search (Go)
648.exchange2_s	Fortran	1	AI: recursive solution generator (Sudoku
657.xz_s	C	33	General data compression

Source: https://www.spec.org/cpu2017/Docs/index.html#benchmarks

The SPEC2017: SPECrate 2017 FP

SPECrate 2017 FP	Language	KLOC	Application Area
503.bwaves_r	Fortran	1	Explosion modeling
507.cactuBSSN_r	C++, C, Fortran	257	Physics: relativity
508.namd_r	C++	8	Molecular dynamics
510.parest_r	C++	427	Biomedical imaging: OT with FE
511.povray_r	C++, C	170	Ray tracing
519.lbm_r	C	1	Fluid dynamics
521.wrf_r	Fortran, C	991	Weather forecasting
526.blender_r	C++, C	1,577	3D rendering and animation
527.cam4_r	Fortran, C	407	Atmosphere modeling
538.imagick_r	C	259	Image manipulation
544.nab_r	C	24	Molecular dynamics
549.fotonik3d_r	Fortran	14	Computational Electromagnetics
554.roms_r	Fortran	210	Regional ocean modeling

The SPEC2017: SPECspeed 2017 FP

SPECspeed 2017 FP	Language	KLOC	Application Area
603.bwaves_s	Fortran	1	Explosion modeling
607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
608.namd_s	C++	8	Molecular dynamics
610.parest_s	C++	427	Biomedical imaging: OT with F
611.povray_s	C++, C	170	Ray tracing
619.lbm_s	C	1	Fluid dynamics
621.wrf_s	Fortran, C	991	Weather forecasting
626.blender_s	C++, C	1,577	3D rendering and animation
627.cam4_s	Fortran, C	407	Atmosphere modeling
628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (clin
638.imagick_s	C	259	Image manipulation
644.nab_s	C	24	Molecular dynamics
649.fotonik3d_s	Fortran	14	Computational Electromagnetics
654.roms_s	Fortran	210	Regional ocean modeling

SPEC benchmark: How it has evolved

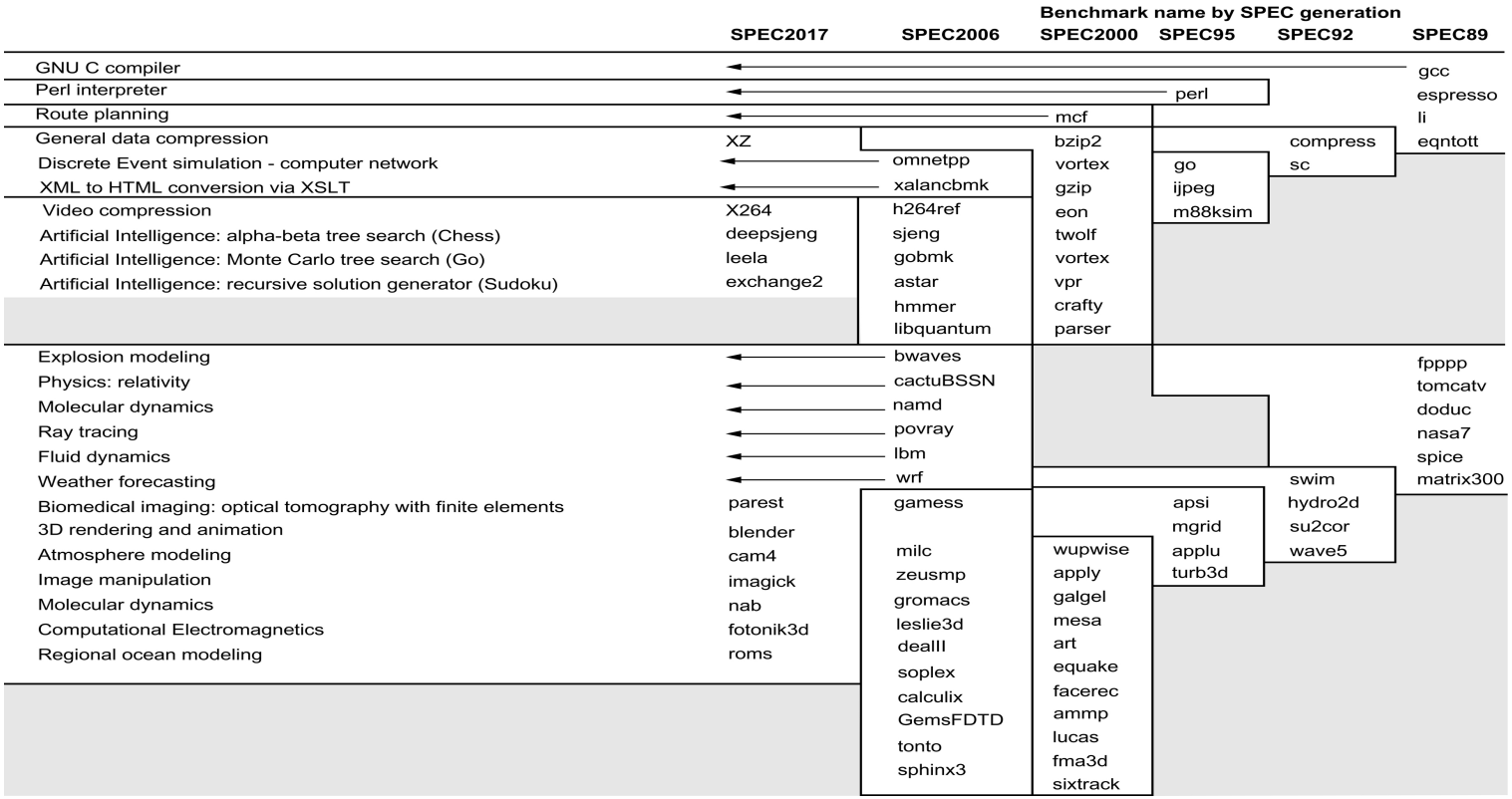

Figure : Evolution of SPEC benchmark since 1989 to 2017.

- GCC the oldest to survive till date.
- There are total of 89 benchmarks so far in SPEC.

Summarising the Performance Results

Example:
The total execution time.

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

The one answer with unique number for execution time:
How much faster is B: Total(A)/Total(B) = 9.1 time

Summerising the Performance: Arithmetic Mean

- arith_mean(A) $= 1001/2 = 500.5$
- arith_mean(B) $= 110/2 = 55$
- arith_mean(A) / arith_mean(B) $= 9.1$

- The General formula: $\sum_{i=1}^{n} exectime(i)/n$
- If the programs are executed nonuniformly the weighted AM could be applied:

$$\frac{\sum_{i=1}^{n} weight(i) * exectime(i)}{\sum weight(i)}$$

where n is total programs.

Summerising the Performance: Harmonic Mean

- Harmonic mean:

  $1/\mathrm{H} = \frac{\sum_{i=1}^{n} 1/exectime(i)}{n}$

  $\mathrm{H} = n/\sum_{i=1}^{n} 1/exectime(i)$

- The use is useful where the sample points are in rates or performance. (Example: reporting MIPS or MFLOPS)
  - Rate has time denominator $(1/t_i)$

Summerising the Performance: Dealing with Ratios

Example:

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

Lets take ration with respect to A (normalise with respect to A)

	Machine A	Machine B
Program 1	1	10
Program 2	1	0.1
Average	1	5.05

Summerising the Performance: Dealing with Ratios

Example:

	Machine A	Machine B
Program 1	1	10
Program 2	1000	100
Total	1001	110

Lets take ration with respect to B (normalise with respect to B)

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1
Average	5.05	1

The first calculation tells us: A is 5.05x better than B

The second calculation tells us: B is 5.05x better than A

Don't use arithmetic mean on ratios!

Summerising the Performance: Geometric Mean

- Use geometric mean for ratios
- Geometric mean:

  $\sqrt[n]{\prod_{i=1}^{n} ratio_i}$
- In the Example: GM of Machine A is 1, and GM of Machine B is also 1
- A is as good as B
- Corollary: GM of ratios is not proportional to total time
- Corollary: GM of ratios is equal to the ratio of GMs
- Independent of reference machine: ratio of GM is equal to the geometric mean of the performance ratios.

Example on use of Ratio and Geometric mean

Benchmarks	Sun Ultra Enterprise 2 time (seconds)	AMD A10-6800K time (seconds)	SPEC 2006Cint ratio	Intel Xeon E5-2690 time (seconds)	SPEC 2006Cint ratio	AMD/Intel times (seconds)	Intel/AMD SPEC ratios
perlbench	9770	401	24.36	261	37.43	1.54	1.54
bzip2	9650	505	19.11	422	22.87	1.20	1.20
gcc	8050	490	16.43	227	35.46	2.16	2.16
mcf	9120	249	36.63	153	59.61	1.63	1.63
gobmk	10,490	418	25.10	382	27.46	1.09	1.09
hmmer	9330	182	51.26	120	77.75	1.52	1.52
sjeng	12,100	517	23.40	383	31.59	1.35	1.35
libquantum	20,720	84	246.08	3	7295.77	29.65	29.65
h264ref	22,130	611	36.22	425	52.07	1.44	1.44
omnetpp	6250	313	19.97	153	40.85	2.05	2.05
astar	7020	303	23.17	209	33.59	1.45	1.45
xalancbmk	6900	215	32.09	98	70.41	2.19	2.19
Geometric mean			31.91		63.72	2.00	2.00

Figure : The ratio of AMD and Intel coputer with reference to Sun Ultra

Summary: AM, GM, and HM

- Use AM for times
- Use HM if forced to use rates
- Use GM if forced to use ratios

Reading Meterials

- John L Henessy and D Patternson, Computer Architecture: A Quantitative Approach, 5th Edition, pp. 36-58 (Chapter 1).
- J.E. Smith, Characterizing Computer Performance with a Single Number, CACM Volume 31, Issue 10 (October 1988), pp.1202-1206.