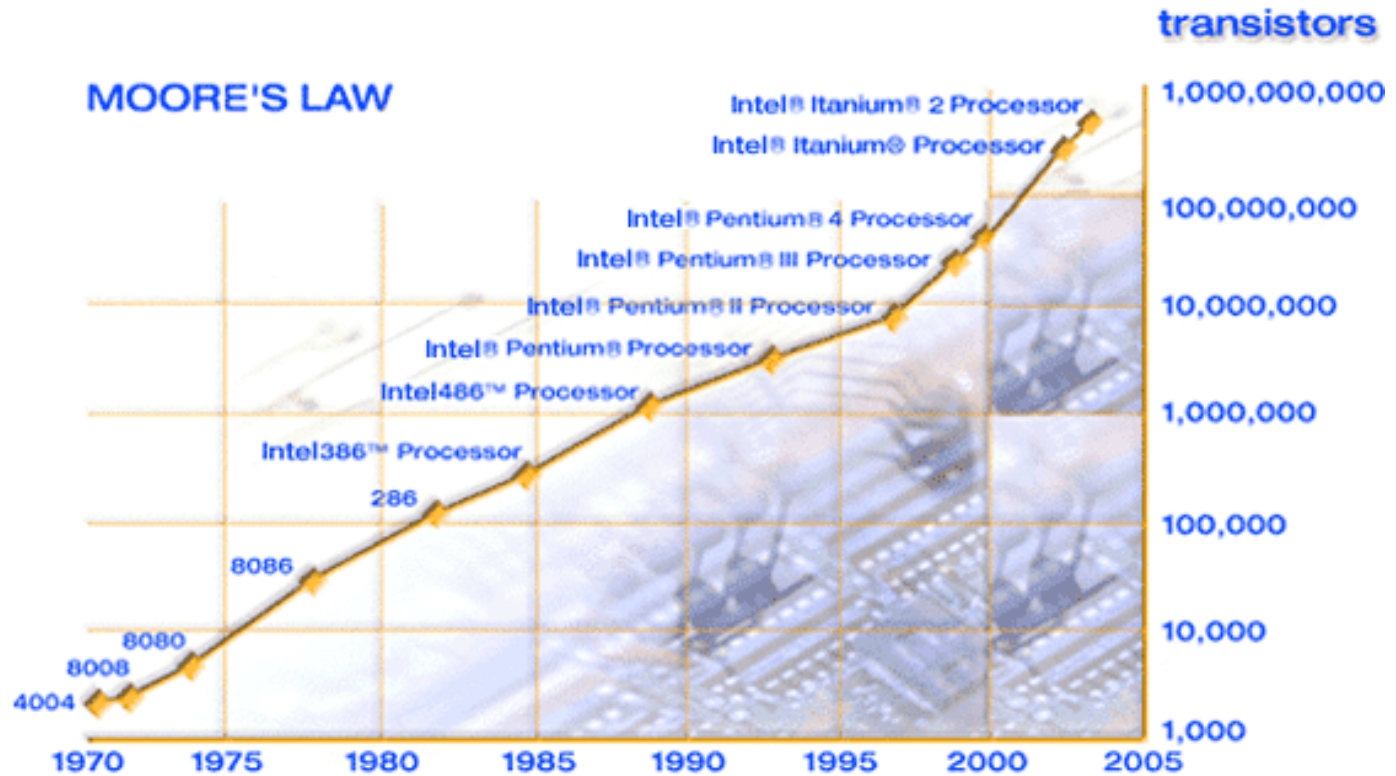


Performance of computer systems

- Many different factors among which:
 - Technology
 - Raw speed of the circuits (clock, switching time)
 - Process technology (how many transistors on a chip)
 - Organization
 - What type of processor (e.g., RISC vs. CISC)
 - What type of memory hierarchy
 - What types of I/O devices
 - How many processors in the system
 - Software
 - O.S., compilers, database drivers etc

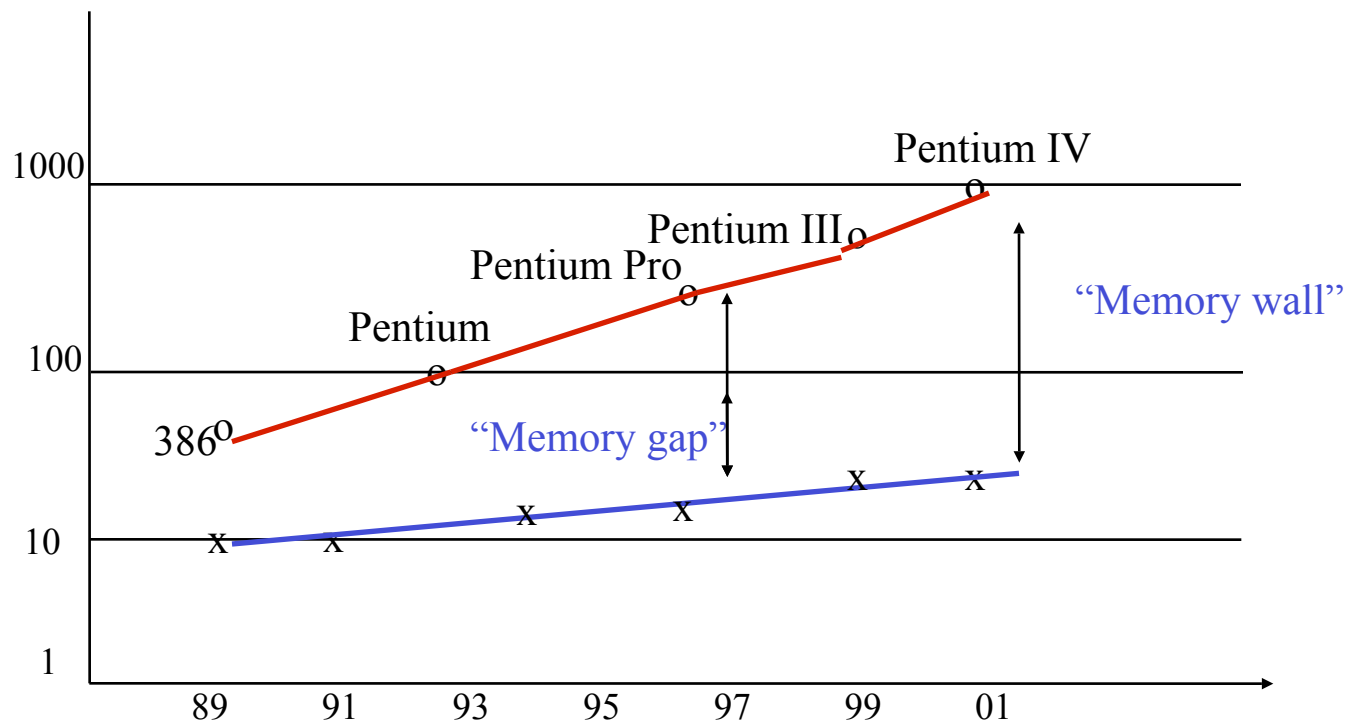
Moore's Law



Courtesy Intel Corp.

Processor-Memory Performance Gap

- x Memory latency decrease (10x over 8 years but densities have increased 100x over the same period)
- o x86 CPU speed (100x over 10 years)



What are some possible metrics

- Raw speed (peak performance = clock rate)
- **Execution time** (or **response time**): time to execute one (suite of) program from beginning to end.
 - Need benchmarks for integer dominated programs, scientific, graphical interfaces, multimedia tasks, desktop apps, utilities etc.
- **Throughput** (total amount of work in a given time)
 - measures utilization of resources (good metric when many users: e.g., large data base queries, Web servers)
- Quite often improving execution time will improve throughput and vice-versa

Execution time Metric

- Execution time: inverse of performance

$$Performance_A = 1 / (Execution_time_A)$$

- Processor A is faster than Processor B

$$Execution_time_A < Execution_time_B$$

$$Performance_A > Performance_B$$

- Relative performance

$$Performance_A / Performance_B = Execution_time_B / Execution_time_A$$

Measuring execution time

- Wall clock, response time, elapsed time
- Some systems have a “time” function
 - Unix 13.7u 23.6s 18:37 3% 2069+1821k 13+24io 62pf+0w
- Difficult to make comparisons from one system to another because of too many factors
- Remainder of this lecture: *CPU execution time*
 - Of interest to microprocessors vendors and designers

Definition of CPU execution time

CPU execution_time = CPU clock_cycles * clock cycle_time

- *CPU clock_cycles* is program dependent thus *CPU execution_time* is program dependent
- *clock cycle_time* (nanoseconds, *ns*) depends on the particular processor
- *clock cycle_time = 1 / clock cycle_rate* (rate in *MHz*)
 - clock cycle_time = 1 μ s, clock cycle_rate = 1 MHz
 - clock cycle_time = 1ns, clock cycle_rate = 1 GHz
- Alternate definition
CPU execution_time = CPU clock_cycles / clock

CPI -- Cycles per instruction

- Definition: **CPI** average number of clock cycles per instr.
 $CPU\ clock_cycles = Number\ of\ instr. * CPI$
 $CPU\ exec_time = Number\ of\ instr. * CPI * clock\ cycle_time$
- Computer architects try to **minimize CPI**
 - or **maximize** its inverse **IPC** : number of instructions per cycle
- CPI in isolation is not a measure of performance
 - program dependent, compiler dependent
 - but good for assessing architectural enhancements (experiments with same programs and compilers)
- In an ideal pipelined processor (to be seen soon) $CPI = 1$
 - but... not ideal so $CPI > 1$
 - could have $CPI < 1$ if several instructions execute in parallel (superscalar processors)

Classes of instructions

- Some classes of instr. take longer to execute than others
 - e.g., floating-point operations take longer than integer operations
- Assign CPI's per classes of inst., say CPI_i
*CPU exec_time = $\sum (CPI_i * C_i) * \text{clock cycle_time}$*
where C_i is the number of insts. of class i that have been executed
- Note that minimizing the number of instructions does not necessarily improve execution time
- Improving one part of the architecture can improve the CPI of one class of instructions

How to measure the average CPI

A given of the processor

Elapsed time: wall clock

$$\text{CPU exec_time} = \text{Number of instr.} * \text{CPI} * \text{clock cycle_time}$$

- Count instructions executed in each class
- Needs a simulator
 - interprets every instruction and counts their number
- or a profiler
 - discover the most often used parts of the program and instruments only those
 - or use sampling
- Use of programmable hardware counters
 - modern microprocessors have this feature but it's limited

Other popular performance measures: MIPS

- MIPS (Millions of instructions per second)
 - MIPS = Instruction count / (Exec.time * 10⁶)
 - MIPS = (Instr. count * clock rate)/(Instr. count *CPI * 10⁶)
 - MIPS = clock rate /(CPI * 10⁶)
- MIPS is a rate: the higher the better
- MIPS in isolation no better than CPI in isolation
 - Program and/or compiler dependent
 - Does not take the instruction set into account
 - can give “wrong” comparative results

Other metric: MFLOPS

- Similar to MIPS in spirit
- Used for scientific programs/machines
- MFLOPS: million of floating-point ops/second

Benchmarks

- Benchmark: workload representative of what a **system** will be used for
- Industry benchmarks
 - SPECint and SPECfp industry benchmarks updated every few years, Currently SPEC CPU2000
 - Linpack (Lapack), NASA kernel: scientific benchmarks
 - TPC-A, TPC-B, TPC-C and TPC-D used for databases and data mining
 - Other specialized benchmarks (Olden for list processing, Specweb, SPEC JVM98 etc...)
 - Benchmarks for desktop applications, web applications are not as standard
 - Beware! Compilers are super optimized for the benchmarks

How to report (benchmark) performance

- If you measure **execution times** use **arithmetic mean**
 - e.g., for n benchmarks
$$(\sum exec_time_j) / n$$
- If you measure **rates** use **harmonic mean**
$$n / (\sum 1/rate_j) = 1 / (\text{arithmetic mean})$$

Computer design: Make the common case fast

- Amdahl's law (speedup)
- $\text{Speedup} = (\text{performance with enhancement}) / (\text{performance base case})$
Or equivalently,
 $\text{Speedup} = (\text{exec.time base case}) / (\text{exec.time with enhancement})$
- For example, application to parallel processing
 - s fraction of program that is sequential
 - Speedup S is at most $1/s$
 - That is if 20% of your program is sequential the maximum speedup with an infinite number of processors is at most 5