Lecture 2, Performance

Performance

(P&H: p.28-53)

- purchasing perspective
  - performance, cost
- design perspective
  - performance / cost and improvements
- require
  - method for calculation
  - basis for comparison
  - metric for evaluation
  - understanding of implications for architectural choices

Basis for comparison: Compare your product with what your competition would be when your product is released

There are multiple perspectives of performance. If we want to buy a computer, we would be concerned about the performance and costs of what are available to select the best. If we are designing a new computer, we would need to ensure that when it is ready, it would not only be better than existing computers, but would also be competitive against other computers available at that time. We require methods for calculating performance, a basis for comparing designs, metrics for evaluating performance, and appreciation of the impact of architectural choices.

Calculating performance

- CPI: average clock cycles per instruction
- number of cycles for program P = number of instr. for P x CPI
- execution time for P = clock cycle time x number of cycles for P
  = $\frac{1}{\text{clock speed}}$ x number of cycles for P
- average exe. time for P1, P2...Pn = $\frac{1}{n}$ (exe. time for P1 +···+ exe. time for Pn)
  (assume equal workload)

execution time equation:

$$\text{exe. time} = \text{instr. count} \times \text{CPI} \times \text{cycle time}$$

wl 2024 2.4

Clock cycles: Almost all computers have a clock that determines when events take place in the hardware. These discrete time intervals are called clock cycles. Designers refer to the length of a clock period as the time for a complete clock cycle (e.g. 250 picoseconds, or 250 ps), such that the clock rate (e.g. 4 gigahertz, or 4 GHz) is the reciprocal of the clock period. When we talk about clock or cycles, we refer to the clock cycle, and clock period is the same as cycle time. When we talk about frequency and clock frequency, we mean the clock rate. CPI: The term clock cycles per instruction, which is the average number of clock cycles each instruction takes to execute, is often abbreviated as CPI. Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in a program

CPI provides a way of comparing two different implementations of the same ISA, since the number of instructions executed for a program is the same. The execution time equation is simply the product of instruction count, the CPI, and the cycle time. The key idea is that performance should be compared on the basis of execution time, rather than other measures such as the number of instructions executed per second.

Example

- M1 and M2 implement the same instruction set with 2 classes of instructions: A and B
- CPI for M1 on class A instructions: A1
  B : B1
- CPI for M2 on class A instructions: A2
  B : B2
- clock speed for M1: C1 MHz
  M2: C2 MHz
- compare their peak and average performance of N instructions, half class A and half class B
- hints: find ratio of execution times
  check dimension of expression

wl 2024 2

This is a representative example of calculations concerning performance. There are many variables which make it look complex, but if you could keep a clear mind, you would find such calculations straight-forward.

Example (cont.)

- peak performance for N instructions ($P \downarrow Q$ = minimum of P,Q)
  - exe. time for M1 $= \frac{1}{C1} \times N \times$ minimum CPI for M1 $= \frac{N(A1 \downarrow B1)}{C1}$
  - compare M1 and M2 : $\frac{\text{exe. time for M1}}{\text{exe. time for M2}} = \frac{(A1 \downarrow B1)\, C2}{(A2 \downarrow B2)\, C1}$
- average performance given by execution time of N instructions, half class A and half class B
  - exe. time for M1 $= \frac{1}{C1} \times N \times$ average CPI for M1 $= \frac{N(A1 + B1)}{2C1}$
  - compare M1 and M2 : $\frac{\text{exe. time for M1}}{\text{exe. time for M2}} = \frac{(A1 + B1)\, C2}{(A2 + B2)\, C1}$
- CPI for N_A class A, N_B class B instructions? compare speed: number of instructions per sec?

Performance calculations are usually straightforward. To compare the performance of two designs, we just need to look at the ratio of execution times for the two designs.

Aspects of processor performance

$$\frac{\text{processor}}{\text{time}} = \frac{\text{sec}}{\text{prog}} = \frac{\text{intsr}}{\text{prog}} \times \frac{\text{cycles}}{\text{instr}} \times \frac{\text{sec}}{\text{cycle}}$$

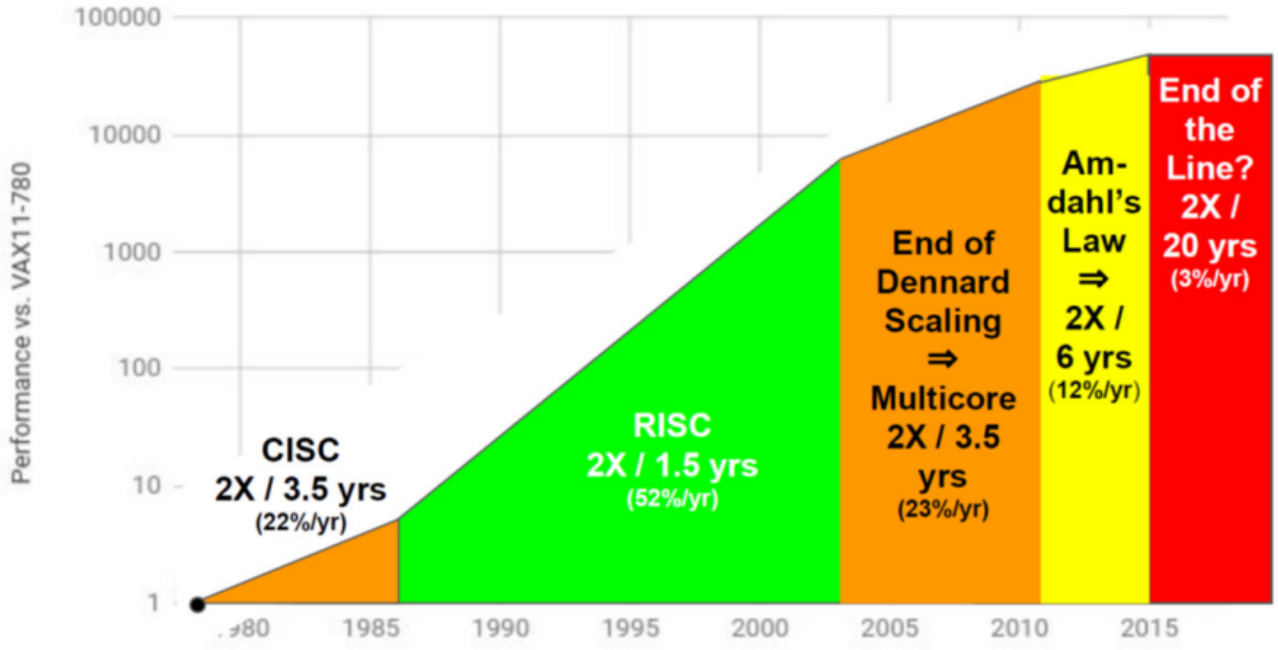
	Instruction Count	CPI	Clock Rate
Algorithm	√	√	
Language	√	√	
Compiler	√	√	
Instruction Set	√	√	√
Organization			√
Technology			√

wl 2024 2.

The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI. The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions. The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways. The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the clock rate of the processor. In contrast, the machine organization and the transistor technology would only affect the clock rate.

Changing the Algorithm affects IC, CPI but not CR, etc.

Challenges facing computer architects!



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

wl 2024 2.8

There was a golden age of RISC that led to rapid performance improvement, but that began to slow down since about 2003. Can we repeat this golden age?

Performance example

- minimise: $\text{exe. time} = \text{instr. count} \times \text{CPI} \times \text{cycle time}$

-

	SUN 68000	SUN RISC
Instruction count ratio	1.0	1.25
Cycle time	40 ns	60 ns
CPI	5.0 – 7.0	1.3 – 1.7
Execution time ratio	2	1
Price ratio	1	1.1 – 1.2

Lets use the execution time equation to compare the performance of two computers from SUN. The old one is based on the 68000 processor which has a CISC architecture, while the new one is based on a RISC architecture called SPARC. Although the cycle time and the instruction count of the RISC are worse than those of the 68000, its lower CPI enables higher performance. The pricing for the RISC-based design is also competitive: although costing only up to 20% more, the RISC would double the performance of the 68000!

Doubles performance, but only slightly higher price.

RISC Instruction set design principles

- make the common case fast
  - reduce CPI
- simple and regular format, small number of general-purpose registers
  - reduce cycle time
  - simplify implementation: more adaptable to new technology
  - include useful facilities on chip, e.g. memory management
  - easier to use, fewer alternatives: manually and by compilers
  - greater confidence in hardware correctness
  - smaller chip size: higher yield, reduce production cost
- require compromise: increased code size
  need better compilers

wl 2024 2.1(

So we could see that the secret of RISC is the low CPI compared with CISC. It reflects the simple and regular instruction format, which reduces the number of cycles needed for each instruction to complete. But RISC has drawbacks, such as large code size compared with CISC.

Smaller chip size: less likely dust will affect chip

Improving performance

- fast, local store
  - e.g. on-chip caches
- concurrent execution of instructions
  - multiple function units : super scalar
  - "production line" arrangement : pipeline
  - multiple instruction streams : multi-threading
- direct hardware implementation, domain-specific optim.
  - reconfigurable hardware: billions of programmable gates
  - no fetch / decode, customise at compile time and at run time
  - see http://www.doc.ic.ac.uk/~wl/papers/federico_faggin.htm
- other technologies
  - manycore processors: e.g. Graphics Processing Unit (GPU)
  - quantum computing: based on quantum circuits… qubits

wl 2024 2.11

Programmers want memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory today is often the majority of computer cost. Architects have found that they can address these conflicting demands with a hierarchy of memories, with the fastest, smallest, and the most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom. As you'll learn in the second part of this module, caches give the programmer the illusion that main memory is almost as fast as the top of the hierarchy and nearly as big and as cheap as the bottom of the hierarchy. In other words, the closer to the top of the hierarchy next to the CPU, the faster and more expensive per bit of the memory; the wider the base of the layer, the larger the memory.

Performance evaluation

- guide design decisions
- compare different architectures
- compare implementations of a given architecture
- compare compilers for a given implementation
- depend on particular programs and data
- reporting performance measurements: reproducibility
- speed sometimes in: million instructions per second
  - larger MIPS = faster program?
- benchmark examples
  - SPEC: System Performance Evaluation Co-operative
  - EEMBC: Embedded Microprocessor Benchmark Consortium

Performance can guide design decisions, help compare architectures and their implementations. Benchmarking is a systematic way to evaluate performance.

Basis of evaluation

method	pros	cons
actual target workload	• representative	• specific, not portable • difficult to run/measure • hard to identify problem
full application benchmarks	• portable • wide spread • improve. useful	• less representative
small kernel benchmarks	• easy to use • early in design cycle • identify peak capability	• peak performance far away from typical performance

Example: SPEC CPU2000

- 26 component-level benchmarks with inputs
  for processor, memory, compiler; not I/O, network, graphics
- 12 integer (11 in C, 1 in C++)
  gcc, compress, FPGA place and route, chess, ray tracing
- 14 floating-point intensive
  qcd, multi-grid solver, finite-element, face recognition
- reference machine: normalise performance metrics
  Sun Ultra5_10 with 300 MHz processor (see *opensparc.net*)
- more recent: SPEC CPU2006 (12 integer, 17 float. point)
  criteria: compute-bound, portable, latest applications
  see *http://www.spec.org/osg/*
- other SPEC benchmarks: Java, GPU, HPC, power workload
  non-SPEC benchmarks: EEMBC, GroundHog

wl 2024 2.14

A well-known benchmark is SPEC CPU2000. There are 26 programs in this benchmark, covering representative computations. SPEC CPU2006 is a more recent benchmark.

SPEC CINT2006 for Opteron X4 2356

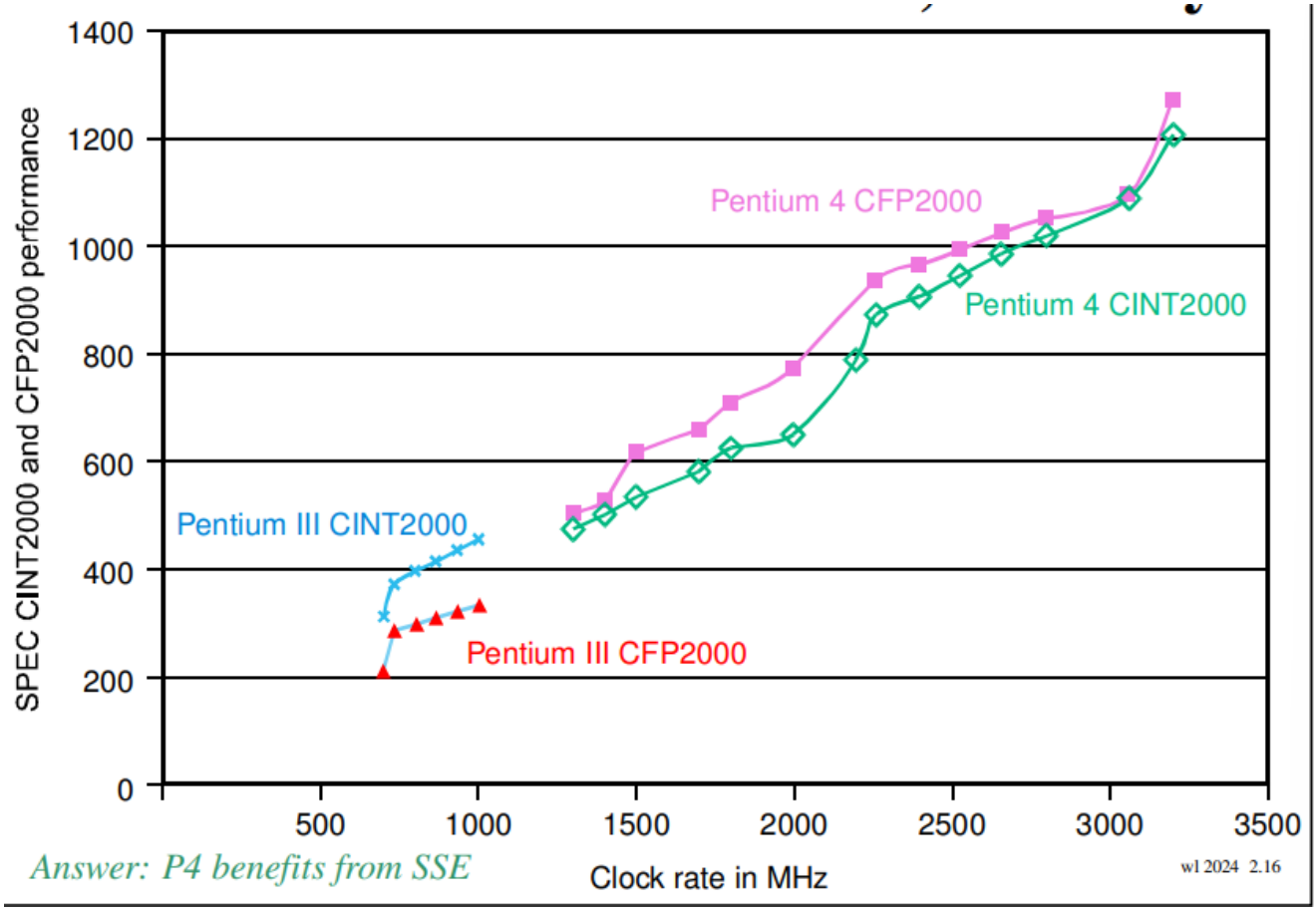
Name	Description	IC×10⁹	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

Geometric mean: $\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$

Here is an example of performance evaluation for the Opteron X4 2356 processor using the SPEC CINT2006. The SPECratio normalises performance results by comparing the performance relative to a reference machine. When comparing two computers using SPECratios, use the geometric mean so that it gives the same relative answer no matter what computer is used to normalise the results. If we averaged the normalized execution time values with an arithmetic mean, the results would vary depending on the computer we choose as the reference

PIII vs P4: what different, and why?



Finally, lets look at the integer (CINT2000) and floating-point (CFP2000) benchmarks for two microprocessors: Pentium III and Pentium 4. One could see that Pentium III's integer performance is better than its floating-point performance, while it is the other way round for Pentium 4. Why is that the case? The Pentium 4 provides a set of new instructions called the Streaming SIMD Extensions 2 that provide a significant boost for floating-point performance. Thus, both the instruction count and the CPI for the Pentium 4 will differ from those of the Pentium 3, producing improved performance.