Advanced Topics in Computer Architecture

Marenglen Biba
Department of Computer Science
University of New York Tirana

Computer Architecture: A Classic

- "What do the following have in common: Beatles' tunes, HP calculators, chocolate chip cookies, and *Computer Architecture*? They are all classics that have stood the test of time."

  Robert P. Colwell, Intel lead architect

Goals of the course

Understanding the design techniques, machine structures, technology factors, evaluation methods that will determine the form of computers in 21st Century

Learning outcomes

At the end of the course the student should be able to:

A. Understand and evaluate the hardware components of advanced architectures

B. Understand and analyze architectures performance and select among different ones for particular use scenarios.

C. Understand and analyze the most important parallel architectures in order to distinguish their main differences.

D. Understand and use simulation and evaluation tools for advanced computer architectures

Content

- Fundamentals of Computer Design
- Pipelining
- Instruction-Level Parallelism
- Multiprocessors and Thread-Level Parallelism
- Interconnection networks for parallel machines
- RISC Machines (throughout the course)

Assessment

Assessment Details:

Methods of Assessment	Please identify the LAST item of assessment that a student sits with a tick	Grading Mode	Weighting %	Minimum Pass Mark	Word Length	Outline Details
Coursework			40	40%	4000	a) Group project Case study Covering Learning Outcomes: D
Examination	√		60	40%		Covering Learning Outcomes: A,B,C

Text

ISBN Number	Author	Date	Title	Publisher
978-01237049 00	Hennessy & Patterson	2006	*Computer Architecture: A Quantitative Approach*, 4th Ed.	Morgan Kaufmann Publishers

Advanced Topics in Computer Architecture

Lecture 1
Fundamentals of Computer Design

Marenglen Biba
Department of Computer Science
University of New York Tirana

Outline

1.1 Introduction

1.2 Classes of Computers

1.3 Defining Computer Architecture

1.4 Trends in Technology

1.5 Trends in Power in Integrated Circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting, and Summarizing Performance

1.9 Quantitative Principles of Computer Design

1.10 Putting It All Together: Performance and Price-Performance

1.11 Fallacies and Pitfalls

Computer Technology

- Computer technology has made incredible progress in the roughly 60 years since the first general-purpose electronic computer was created.
- Today, less than $500 will purchase a personal computer that has more performance, more main memory, and more disk storage than a computer bought in 1985 for 1 million dollars.
- This rapid improvement has come both from advances in the:
  - technology used to build computers
  - innovation in computer design.

Microprocessor

- Although technological improvements have been fairly steady, progress arising from better computer architectures has been much less consistent.
- During the first 25 years of electronic computers, both forces made a major contribution, delivering performance improvement of about 25% per year.
- The late 1970s saw the emergence of the microprocessor.
- The ability of the microprocessor to ride the improvements in integrated circuit technology led to a higher rate of improvement—roughly 35% growth per year in performance.

Successful Architectures

- Two significant changes in the computer marketplace made it easier than ever before to be commercially successful with a new architecture.
  - First, the virtual elimination of assembly language programming reduced the need for object-code compatibility.
  - Second, the creation of standardized, vendor-independent operating systems, such as UNIX and its clone, Linux, lowered the cost and risk of bringing out a new architecture.

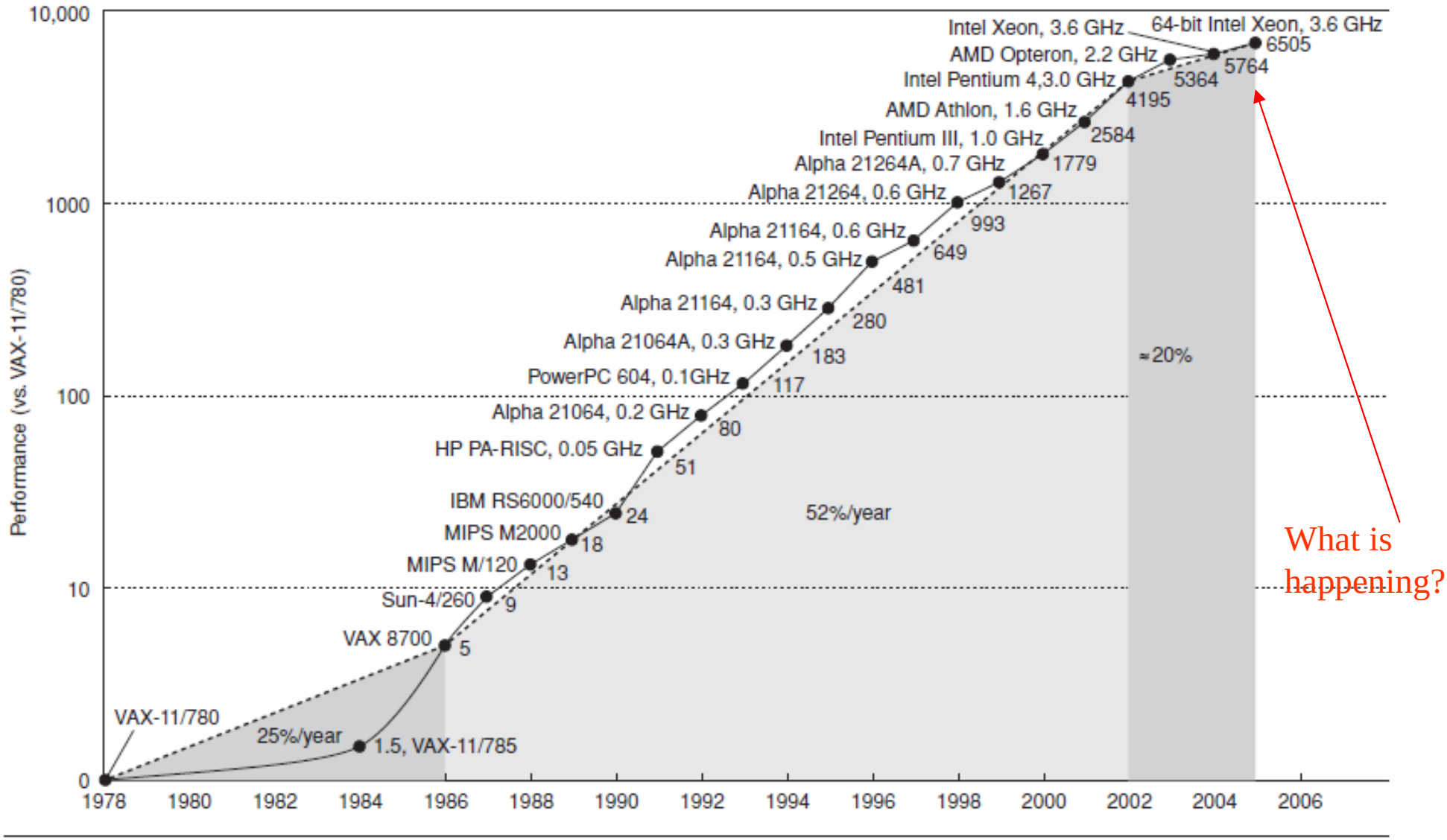
RISC Machines

- These changes made it possible to develop successfully a new set of architectures with simpler instructions, called RISC (Reduced Instruction Set Computer) architectures, in the early 1980s.
- The RISC-based machines focused the attention of designers on two critical performance techniques
  - *the exploitation of Instruction level parallelism initially through pipelining and later through multiple instruction issue*
  - *the use of caches (initially in simple forms and later using more sophisticated organizations and optimizations).*

RISC: Faster Machines

- The RISC-based computers raised the performance bar, forcing prior architectures to keep up or disappear.
  - The Digital Equipment Vax could not, and so it was replaced by a RISC architecture.
  - Intel rose to the challenge, primarily by translating x86 (or IA-32) instructions into RISC-like instructions internally, allowing it to adopt many of the innovations first pioneered in the RISC designs.
- As transistor counts soared in the late 1990s, the hardware overhead of translating the more complex x86 architecture became negligible.

Uniprocessor performance: 50% Annual Growth until 2002



The end of an era

- The 16-year renaissance is over.
- Since 2002, processor performance improvement has dropped to about 20% per year due to the triple hurdles of:
  - maximum power dissipation of air-cooled chips,
  - little instruction-level parallelism left to exploit efficiently, and
  - almost unchanged memory latency.
- Indeed, in 2004 Intel canceled its high-performance uniprocessor projects and joined IBM and Sun in declaring that the road to higher performance would be via multiple processors per chip rather than via faster uniprocessors.
- This signals a historic switch from relying solely on instruction level parallelism (ILP), to *thread-level parallelism* (TLP) and *data-level parallelism* (DLP).

Outline

1.1 Introduction

1.2 Classes of Computers

1.3 Defining Computer Architecture

1.4 Trends in Technology

1.5 Trends in Power in Integrated Circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting, and Summarizing Performance

1.9 Quantitative Principles of Computer Design

1.10 Putting It All Together: Performance and Price-Performance

1.11 Fallacies and Pitfalls

Classes of Computers

Feature	Desktop	Server	Embedded
Price of system	$500–$5000	$5000–$5,000,000	$10–$100,000 (including network routers at the high end)
Price of microprocessor module	$50–$500 (per processor)	$200–$10,000 (per processor)	$0.01–$100 (per processor)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Figure 1.2 A summary of the three mainstream computing classes and their system characteristics. Note the wide range in system price for servers and embedded systems. For servers, this range arises from the need for very large-scale multiprocessor systems for high-end transaction processing and Web server applications. The total number of embedded processors sold in 2005 is estimated to exceed 3 billion if you include 8-bit and 16-bit microprocessors. Perhaps 200 million desktop computers and 10 million servers were sold in 2005.

Desktop Computing

- The first, and still the largest market in dollar terms, is desktop computing.
- Desktop computing spans from low-end systems that sell for under $500 to high-end, heavily configured workstations that may sell for $5000.
- Throughout this range in price and capability, the desktop market tends to be driven to optimize *price-performance*.
- This combination of performance (measured primarily in terms of compute performance and graphics performance) and price of a system is what matters most to customers in this market, and hence to computer designers.
- As a result, the newest, highest-performance microprocessors and cost-reduced microprocessors often appear first in desktop systems

Servers

- As the shift to desktop computing occurred, the role of servers grew to provide larger-scale and more reliable file and computing services.
- The World Wide Web accelerated this trend because of the tremendous growth in the demand and sophistication of Web-based services.
- Such servers have become the backbone of large-scale enterprise computing, replacing the traditional mainframe.
- For servers, different characteristics are important. First, dependability is critical.
- Consider the servers running Google, taking orders for Cisco, or running auctions on eBay.
  - Failure of such server systems is far more catastrophic than failure of a single desktop, since these servers must operate seven days a week, 24 hours a day.
- To bring costs up-to-date, Amazon.com had $2.98 billion in sales in the fall quarter of 2005.
- As there were about 2200 hours in that quarter, the average revenue per hour was $1.35 million.

  During a peak hour for Christmas shopping, the potential loss would be many times higher.

Cost of downtime (Amazon data)

Application	Cost of downtime per hour (thousands of $)	Annual losses (millions of $) with downtime of		
		1% (87.6 hrs/yr)	0.5% (43.8 hrs/yr)	0.1% (8.8 hrs/yr)
Brokerage operations	$6450	$565	$283	$56.5
Credit card authorization	$2600	$228	$114	$22.8
Package shipping services	$150	$13	$6.6	$1.3
Home shopping channel	$113	$9.9	$4.9	$1.0
Catalog sales center	$90	$7.9	$3.9	$0.8
Airline reservation center	$89	$7.9	$3.9	$0.8
Cellular service activation	$41	$3.6	$1.8	$0.4
Online network fees	$25	$2.2	$1.1	$0.2
ATM service fees	$14	$1.2	$0.6	$0.1

Figure 1.3 The cost of an unavailable system is shown by analyzing the cost of downtime (in terms of immediately lost revenue), assuming three different levels of availability, and that downtime is distributed uniformly. These data are from Kembel [2000] and were collected and analyzed by Contingency Planning Research.

Servers: other features

- A second key feature of server systems is scalability.
  - Server systems often grow in response to an increasing demand for the services they support or an increase in functional requirements.
  - Thus, the ability to scale up the computing capacity, the memory, the storage, and the I/O bandwidth of a server is crucial.
- Lastly, servers are designed for efficient throughput.
  - That is, the overall performance of the server—in terms of transactions per minute or Web pages served per second—is what is crucial.

Supercomputers

- A related category is *supercomputers*.
- They are the most expensive computers, costing tens of millions of dollars, and they emphasize floating-point performance.
- Clusters of desktop computers, have largely overtaken this class of computer.
- As clusters grow in popularity, the number of conventional supercomputers is shrinking, as are the number of companies who make them.

Embedded Computers

- Embedded computers are the fastest growing portion of the computer market.

- These devices range from everyday machines—most microwaves, most washing machines, most printers, most networking switches, and all cars contain simple embedded microprocessors—to handheld digital devices, such as cell phones and smart cards, to video games and digital set-top boxes.

Embedded Computers

- Embedded computers have the widest spread of processing power and cost.
- They include 8-bit and 16-bit processors that may cost less than a dime, 32-bit microprocessors that execute 100 million instructions per second and cost under $5, and high-end processors for the newest video games or network switches that cost $100 and can execute a billion instructions per second.
- Although the range of computing power in the embedded computing market is very large, price is a key factor in the design of computers for this space.
- Performance requirements do exist, of course, but the primary goal is often meeting the performance need at minimum price, rather than achieving higher performance at a higher price.

Embedded Computers and real-time execution

- Often, the performance requirement in an embedded application is real-time execution.
- A *real-time performance* requirement is when a segment of the application has an absolute maximum execution time.
- For example, in a digital set-top box, the time to process each video frame is limited, since the processor must accept and process the next frame shortly.
- In some applications, a more nuanced requirement exists: the average time for a particular task is constrained as well as the number of instances when some maximum time is exceeded.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Defining Computer Architecture

- The task the computer designer faces is a complex one: *Determine what attributes are important for a new computer, then design a computer to maximize performance while staying within cost, power, and availability constraints.*
- This task has many aspects, including instruction set design, functional organization, logic design, and implementation.
- The implementation may encompass integrated circuit design, packaging, power, and cooling.
- Optimizing the design requires familiarity with a very wide range of technologies, from compilers and operating systems to logic design and packaging.

Defining Computer Architecture

- In the past, the term *computer architecture* often referred only to instruction set design.
- Other aspects of computer design were called *implementation*, often insinuating that implementation is uninteresting or less challenging.
- This view is incorrect.
- The architect's or designer's job is much more than instruction set design, and the technical hurdles in the other aspects of the project are likely more challenging than those encountered in instruction set design.
- We'll review instruction set architecture before describing the larger challenges for the computer architect.

Instruction Set Architecture

- We use the term *instruction set architecture* (ISA) to refer to the actual programmer visible instruction set.
- The ISA serves as the boundary between the software and hardware.
- In this quick review of ISA we will use examples from MIPS and 80x86 to illustrate the seven dimensions of an ISA

MIPS

- MIPS (originally an acronym for Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) developed by MIPS Computer Systems (now MIPS Technologies).
- The early MIPS architectures were 32-bit, and later versions were 64-bit. Multiple revisions of the MIPS instruction set exist, including MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, and MIPS64.
- The current revisions are MIPS32 (for 32-bit implementations) and MIPS64 (for 64-bit implementations).
- MIPS32 and MIPS64 define a control register set as well as the instruction set.

x86 family

- The term x86 refers to a family of instruction set architectures based on the Intel 8086.
- The term is derived from the fact that many early processors that are backward compatible with the 8086 also had names ending in "86".
- Many additions and extensions have been added to the x86 instruction set over the years, almost consistently with full backwards compatibility. The architecture has been implemented in processors from Intel, Cyrix, AMD, VIA and many others.
- As the x86 term became common after the introduction of the 80386, it usually implies binary compatibility with the 32-bit instruction set of the 80386.
- This may sometimes be emphasized as x86-32 to distinguish it either from the original 16-bit x86-16 or from the newer 64-bit x86-64 (also called x64).

Dimensions of ISAs

1. *Class of ISA*

- Nearly all ISAs today are classified as general-purpose register architectures, where the operands are either registers or memory locations.
- The 80x86 has 16 general-purpose registers and 16 that can hold floating-point data, while MIPS has 32 general-purpose and 32 floating-point registers.
- The two popular versions of this class are:
  - *register-memory* ISAs such as the 80x86, which can access memory as part of many instructions
  - *load-store* ISAs such as MIPS, which can access memory only with load or store instructions.
- All recent ISAs are load-store.

Dimensions of ISAs

2. Memory addressing

- Virtually all desktop and server computers, including the 80x86 and MIPS, use byte addressing to access memory operands.
- Some architectures, like MIPS, require that objects must be *aligned*.
  - An access to an object of size *s* bytes at byte address *A* is aligned if *A* mod *s* = 0.
- The 80x86 does not require alignment, but accesses are generally faster if operands are aligned.

Dimensions of ISAs

3. Addressing modes

- In addition to specifying registers and constant operands, addressing modes specify the address of a memory object.
- MIPS addressing modes are Register, Immediate (for constants), and Displacement, where a constant offset is added to a register to form the memory address.
- The 80x86 supports those three plus three variations of displacement:
  - no register (absolute),
  - two registers (based indexed with displacement),
  - two registers where one register is multiplied by the size of the operand in bytes (based with scaled index and displacement).

Dimensions of ISAs

4. Types and sizes of operands

- Like most ISAs, MIPS and 80x86 support operand sizes of 8-bit (ASCII character), 16-bit (Unicode character or half word), 32-bit (integer or word), 64-bit (double word or long integer), and IEEE 754 floating point in 32-bit (single precision) and 64-bit (double precision).
- The 80x86 also supports 80-bit floating point (extended double precision).

Dimensions of ISAs

5. Operations

- The general categories of operations are data transfer, arithmetic logical, control and floating point.
- MIPS is a simple and easy-to-pipeline instruction set architecture, and it is representative of the RISC architectures being used in 2006.
- The 80x86 has a much richer and larger set of operations and it is representative of the CISC machines.

MIPS instructions

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR*
LB, LBU, SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH, LHU, SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW, LWU, SW	Load word, load word unsigned, store word (to/from integer registers)
LD, SD	Load double word, store double word (to/from integer registers)
L.S, L.D, S.S, S.D	Load SP float, load DP float, store SP float, store DP float
MFC0, MTC0	Copy from/to GPR to/from a special register
MOV.S, MOV.D	Copy one SP or DP FP register to another FP register
MFC1, MTC1	Copy 32 bits to/from FP registers from/to integer registers
Arithmetic/logical	*Operations on integer or logical data in GPRs; signed arithmetic trap on overflow*
DADD, DADDI, DADDU, DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB, DSUBU	Subtract; signed and unsigned
DMUL, DMULU, DDIV, DDIVU, MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND, ANDI	And, and immediate
OR, ORI, XOR, XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV	Shifts: both immediate (DS__) and variable form (DS__V); shifts are shift left logical, right logical, right arithmetic
SLT, SLTI, SLTU, SLTIU	Set less than, set less than immediate; signed and unsigned

MIPS instructions

Control	*Conditional branches and jumps; PC-relative or through register*
BEQZ, BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ, BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T, BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN, MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J, JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL, JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode
Floating point	*FP operations on DP and SP formats*
ADD.D, ADD.S, ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D, SUB.S, SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D, MUL.S, MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D, MADD.S, MADD.PS	Multiply-add DP, SP numbers, and pairs of SP numbers
DIV.D, DIV.S, DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C.__.D, C.__.S	DP and SP compares: "__" = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

Figure 1.5 Subset of the instructions in MIPS64. SP = single precision; DP = double precision. Appendix B gives much more detail on MIPS64. For data, the most significant bit number is 0; least is 63.

Dimensions of ISAs

6. Control flow instructions

- Virtually all ISAs, including 80x86 and MIPS, support conditional branches, unconditional jumps, procedure calls, and returns.
- Both use PC-relative addressing, where the branch address is specified by an address field that is added to the PC.
- There are some small differences.
  - MIPS conditional branches (BE,BNE, etc.) test the contents of registers,
  - 80x86 branches (JE,JNE, etc.) test condition code bits as side effects of arithmetic/logic operations.

Dimensions of ISAs

7. Encoding an ISA

- There are two basic choices on encoding: *fixed length* and v*ariable length.*
  - All MIPS instructions are 32 bits long, which simplifies instruction decoding.
  - The 80x86 encoding is variable length, ranging from 1 to 18 bytes.
- Variable-length instructions can take less space than fixed-length instructions, so a program compiled for the 80x86 is usually smaller than the same program compiled for MIPS.
- Note that choices mentioned above will affect how the instructions are encoded into a binary representation.
  - For example, the number of registers and the number of addressing modes both have a significant impact on the size of instructions.

MIPS ISA formats

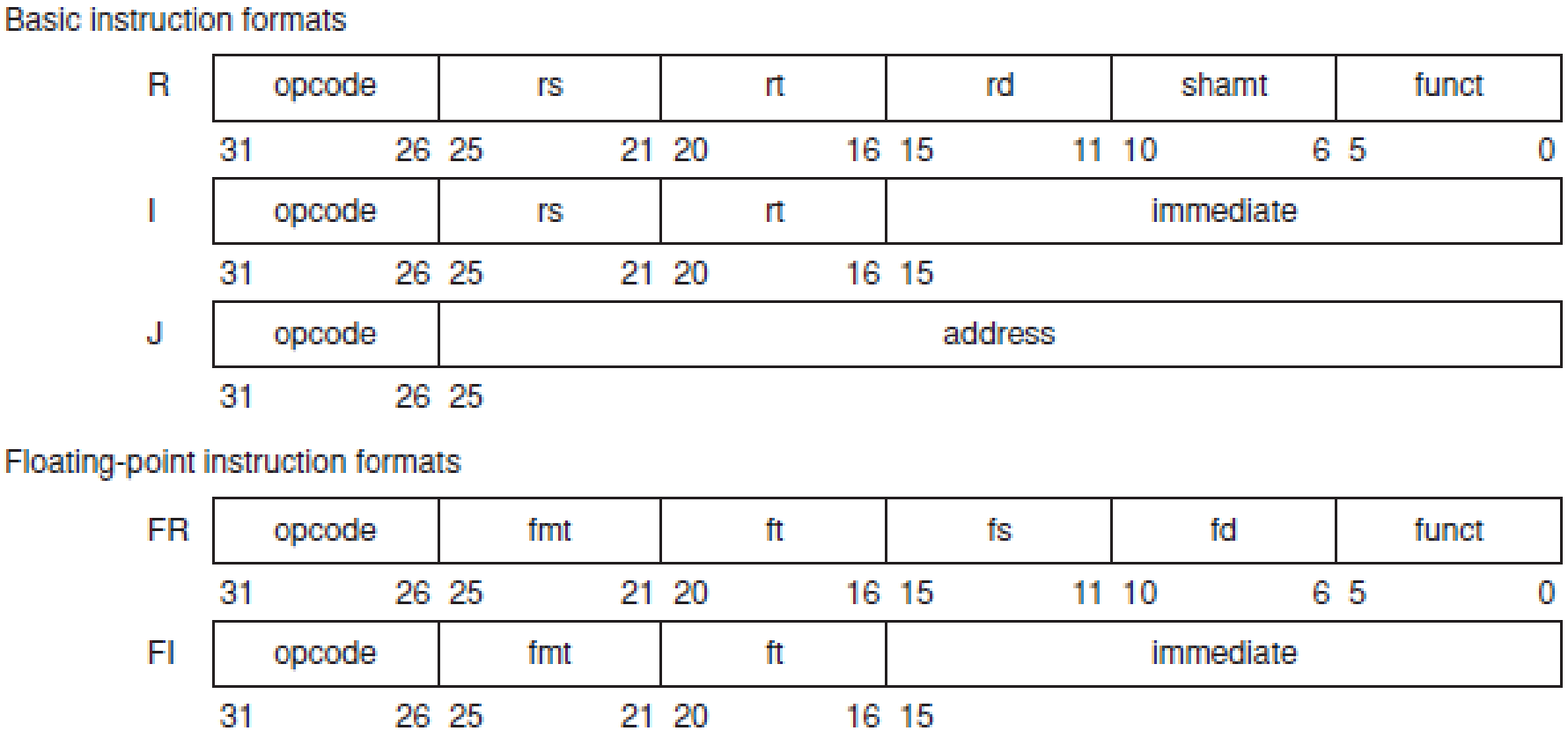


Figure 1.6 MIPS64 instruction set architecture formats. All instructions are 32 bits long. The R format is for integer register-to-register operations, such as DADDU, DSUBU, and so on. The I format is for data transfers, branches, and immediate instructions, such as LD, SD, BEQZ, and DADDIs. The J format is for jumps, the FR format for floating point operations, and the FI format for floating point branches.

Implementation: Organization and Hardware

- The implementation of a computer has two components: organization and hardware.
- The term *organization* includes the high-level aspects of a computer's design, such as the memory system, the memory interconnect, and the design of the internal processor or CPU (central processing unit—where arithmetic, logic, branching, and data transfer are implemented).
- For example, two processors with the same instruction set architectures but very different organizations are the AMD Opteron 64 and the Intel Pentium 4.
- Both processors implement the x86 instruction set, but they have very different pipeline and cache organizations.

Hardware

- *Hardware* refers to the specifics of a computer, including the detailed logic design and the packaging technology of the computer.
- Often a line of computers contains computers with identical instruction set architectures and nearly identical organizations, but they differ in the detailed hardware implementation.
- For example, the Pentium 4 and the Mobile Pentium 4 are nearly identical, but offer different clock rates and different memory systems, making the Mobile Pentium 4 more effective for low-end computers.

Defining Computer Architecture

- In this course, the word *architecture* covers all three aspects of computer design:
  - instruction set architecture
  - organization
  - hardware.

Computer Design Requirements

- Computer architects must design a computer to meet:
  - functional requirements as well as
  - price, power, performance, and availability goals.
- Often, architects also must determine what the functional requirements are, which can be a major task.
  - The requirements may be specific features inspired by the market.
  - Application software often drives the choice of certain functional requirements by determining how the computer will be used.
  - If a large body of software exists for a certain instruction set architecture, the architect may decide that a new computer should implement an existing instruction set.

Functional Requirements

Functional requirements	Typical features required or supported
Application area	*Target of computer*
General-purpose desktop	Balanced performance for a range of tasks, including interactive performance for graphics, video, and audio (Ch. 2, 3, 5, App. B)
Scientific desktops and servers	High-performance floating point and graphics (App. I)
Commercial servers	Support for databases and transaction processing; enhancements for reliability and availability; support for scalability (Ch. 4, App. B, E)
Embedded computing	Often requires special support for graphics or video (or other application-specific extension); power limitations and power control may be required (Ch. 2, 3, 5, App. B)
Level of software compatibility	*Determines amount of existing software for computer*
At programming language	Most flexible for designer; need new compiler (Ch. 4, App. B)
Object code or binary compatible	Instruction set architecture is completely defined—little flexibility—but no investment needed in software or porting programs
Operating system requirements	*Necessary features to support chosen OS (Ch. 5, App. E)*
Size of address space	Very important feature (Ch. 5); may limit applications
Memory management	Required for modern OS; may be paged or segmented (Ch. 5)
Protection	Different OS and application needs: page vs. segment; virtual machines (Ch. 5)
Standards	*Certain standards may be required by marketplace*
Floating point	Format and arithmetic: IEEE 754 standard (App. I), special arithmetic for graphics or signal processing
I/O interfaces	For I/O devices: Serial ATA, Serial Attach SCSI, PCI Express (Ch. 6, App. E)
Operating systems	UNIX, Windows, Linux, CISCO IOS
Networks	Support required for different networks: Ethernet, Infiniband (App. E)
Programming languages	Languages (ANSI C, C++, Java, FORTRAN) affect instruction set (App. B)

Figure 1.7 Summary of some of the most important functional requirements an architect faces. The left-hand column describes the class of requirement, while the right-hand column gives specific examples. The right-hand column also contains references to chapters and appendices that deal with the specific issues.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Trends in Technology

- If an instruction set architecture is to be successful, it must be designed to survive rapid changes in computer technology.
  - After all, a successful new instruction set architecture may last decades—for example, the core of the IBM mainframe has been in use for more than 40 years.
- An architect must plan for technology changes that can increase the lifetime of a successful computer.
  - To plan for the evolution of a computer, the designer must be aware of rapid changes in implementation technology.
- Four implementation technologies, which change at a dramatic pace, are critical to modern implementations:

1- Integrated circuit logic technology

- Transistor density increases by about 35% per year, quadrupling in somewhat over four years.
- Increases in die size are less predictable and slower, ranging from 10% to 20% per year.
- The combined effect is a growth rate in transistor count on a chip of about 40% to 55% per year.
- Device speed scales more slowly

2- SDRAM

- *Synchronous DRAM* (dynamic random-access memory)
  - Capacity increases by about 40% per year, doubling roughly every two years.

3- Magnetic disk technology

- Prior to 1990, density increased by about 30% per year, doubling in three years.
- It rose to 60% per year thereafter, and increased to 100% per year in 1996.
- Since 2004, it has dropped back to 30% per year.
- Despite this roller coaster of rates of improvement, disks are still 50–100 times cheaper per bit than DRAM.

4- *Network technology*

- Network performance depends both on the performance of switches and on the performance of the transmission system.
- Interconnections networks for parallel systems are essential in exploiting parallelism.

Performance Trends: Bandwidth over Latency

- *Bandwidth* or *throughput* is the total amount of work done in a given time, such as megabytes per second for a disk transfer.
- In contrast, *latency* or *response time* is the time between the start and the completion of an event, such as milliseconds for a disk access.
- Bandwidth improves much more rapidly than latency => next slide.

Bandwidth over Latency

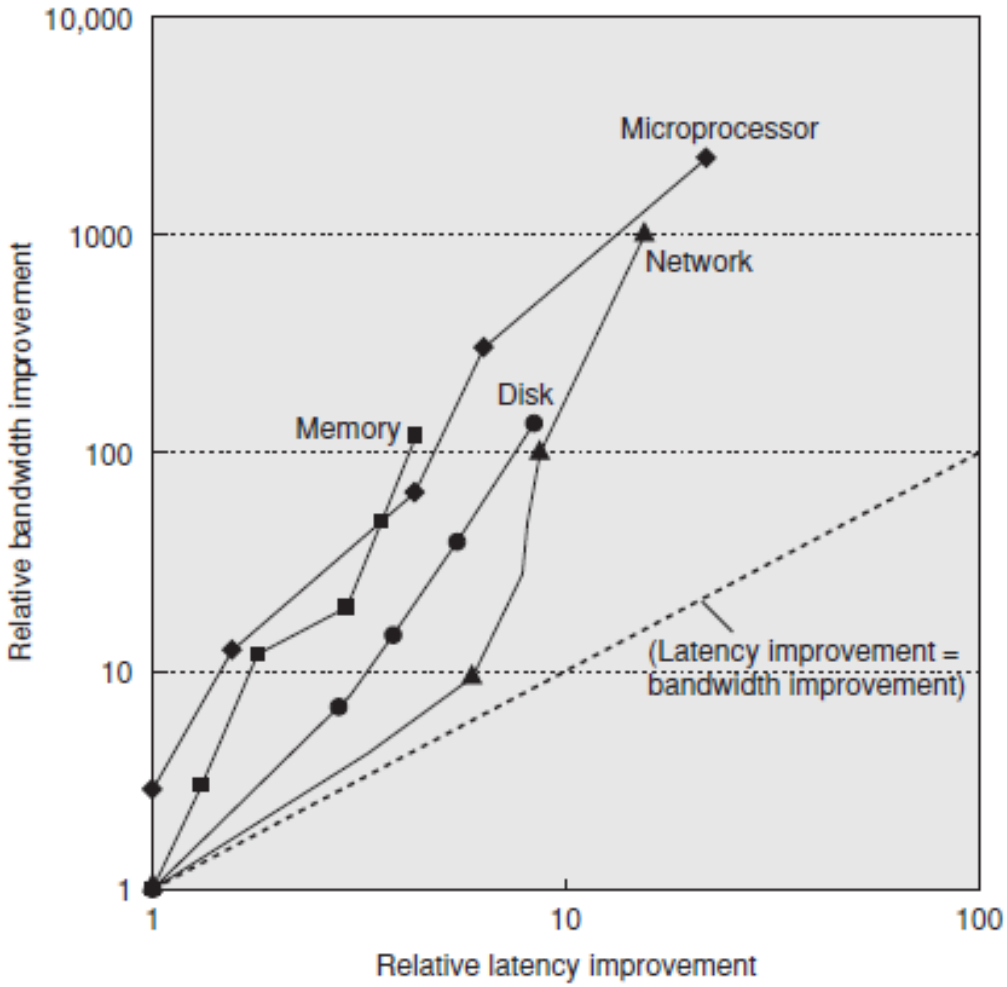


Figure 1.8 Log-log plot of bandwidth and latency milestones from Figure 1.9 relative to the first milestone. Note that latency improved about 10X while bandwidth improved about 100X to 1000X. From Patterson [2004].

Microprocessor	16-bit address/bus, microcoded	32-bit address.bus, microcoded	5-stage pipeline, on-chip I & D caches, FPU	2-way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip 1.2 cache
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4
Year	1982	1985	1989	1993	1997	2001
Die size (mm^2)	47	43	81	90	308	217
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000
Pins	68	132	168	273	387	423
Latency (clocks)	6	5	5	5	10	22
Bus width (bits)	16	32	32	64	64	64
Clock rate (MHz)	12.5	16	25	66	200	1500
Bandwidth (MIPS)	2	6	25	132	600	4500
Latency (ns)	320	313	200	76	50	15
Memory module	DRAM	Page mode DRAM	Fast page mode DRAM	Fast page mode DRAM	Synchronous DRAM	Double data rate SDRAM
Module width (bits)	16	16	32	64	64	64
Year	1980	1983	1986	1993	1997	2000
Mbits/DRAM chip	0.06	0.25	1	16	64	256
Die size (mm^2)	35	45	70	130	170	204
Pins/DRAM chip	16	16	18	20	54	66
Bandwidth (MBit/sec)	13	40	160	267	640	1600
Latency (ns)	225	170	125	75	62	52
Local area network	Ethernet	Fast Ethernet	Gigabit Ethernet	10 Gigabit Ethernet		
IEEE standard	802.3	803.3u	802.3ab	802.3ac		
Year	1978	1995	1999	2003		
Bandwidth (MBit/sec)	10	100	1000	10000		
Latency (μsec)	3000	500	340	190		
Hard disk	3600 RPM	5400 RPM	7200 RPM	10,000 RPM	15,000 RPM	
Product	CDC WrenI 94145-36	Seagate ST41600	Seagate ST15150	Seagate ST39102	Seagate ST373453	
Year	1983	1990	1994	1998	2003	
Capacity (GB)	0.03	1.4	4.3	9.1	73.4	
Disk form factor	5.25 inch	5.25 inch	3.5 inch	3.5 inch	3.5 inch	
Media diameter	5.25 inch	5.25 inch	3.5 inch	3.0 inch	2.5 inch	
Interface	ST-412	SCSI	SCSI	SCSI	SCSI	
Bandwidth (MBit/sec)	0.6	4	9	24	86	
Latency (ms)	48.3	17.1	12.7	8.8	5.7	

Figure 1.9 Performance milestones over 20 to 25 years for microprocessors, memory, networks, and disks. The

Performance Milestones

Scaling of Transistor Performance and Wires

- Integrated circuit processes are characterized by the *feature size*, which is the minimum size of a transistor or a wire in either the *x* or *y* dimension.
- Feature sizes have decreased from 10 microns in 1971 to 0.09 microns in 2006;
  - in fact, we have switched units, so production in 2006 was referred to as "90 nanometers," and 65 nanometer chips were underway.
- Since the transistor count per square millimeter of silicon is determined by the surface area of a transistor, the density of transistors increases quadratically with a linear decrease in feature size.

Transistor performance

- The increase in transistor performance, however, is more complex.
- As feature sizes shrink, devices shrink quadratically in the horizontal dimension and also shrink in the vertical dimension.
- The shrink in the vertical dimension requires a reduction in operating voltage to maintain correct operation and reliability of the transistors.
- This combination of scaling factors leads to a complex interrelationship between transistor performance and process feature size.
- To a first approximation, transistor performance improves linearly with decreasing feature size.

Wires performance

- Although transistors generally improve in performance with decreased feature size, wires in an integrated circuit do not.
- In particular, the signal delay for a wire increases in proportion to the product of its resistance and capacitance.
  - Of course, as feature size shrinks, wires get shorter, but the resistance and capacitance per unit length get worse.
- This relationship is complex, since both resistance and capacitance depend on:
  - detailed aspects of the process,
  - the geometry of a wire,
  - the loading on a wire, and
  - even the adjacency to other structures.

Wire Delay

- In general, wire delay scales poorly compared to transistor performance, creating additional challenges for the designer.
- In the past few years, wire delay has become a major design limitation for large integrated circuits and is often more critical than transistor switching delay.
- Larger and larger fractions of the clock cycle have been consumed by the propagation delay of signals on wires.
- In 2001, the Pentium 4 broke new ground by allocating 2 stages of its 20+-stage pipeline just for propagating signals across the chip.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Trends in Power in Integrated Circuits

- Power also provides challenges as devices are scaled.
- First, power must be brought in and distributed around the chip, and modern microprocessors use hundreds of pins and multiple interconnect layers for just power and ground.
- Second, power is dissipated as heat and must be removed.

Dynamic Power

- For CMOS chips, the traditional dominant energy consumption has been in switching transistors, also called *dynamic power*.
- The power required per transistor is proportional to the product of the load capacitance of the transistor, the square of the voltage, and the frequency of switching, with watts being the unit:

$$\text{Power}_{\text{dynamic}} = 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

- Mobile devices care about battery life more than power, so energy is the proper metric, measured in joules:

$$\text{Energy}_{\text{dynamic}} = \text{Capacitive load} \times \text{Voltage}^2$$

Dynamic Power

- Dynamic power and energy are greatly reduced by lowering the voltage, and so voltages have dropped from 5V to just over 1V in 20 years.
- The capacitive load is a function of the number of transistors connected to an output and the technology, which determines the capacitance of the wires and the transistors.
- For a fixed task, slowing clock rate reduces power, but not energy.

Heat dissipation

- As we move from one process to the next, the increase in the number of transistors switching, and the frequency with which they switch, dominates the decrease in load capacitance and voltage, leading to an overall growth in power consumption and energy.
- The first microprocessors consumed tenths of a watt, while a 3.2 GHz Pentium 4 Extreme Edition consumes 135 watts.
  - Given that this heat must be dissipated from a chip that is about 1 cm on a side, we are reaching the limits of what can be cooled by air ☺
- Several Intel microprocessors have temperature diodes to reduce activity automatically if the chip gets too hot.
  - For example, they may reduce voltage and clock frequency or the instruction issue rate.

Heat dissipation

- Distributing the power, removing the heat, and preventing hot spots have become increasingly difficult challenges.
- Power is now the major limitation to using transistors; in the past it was raw silicon area.
- As a result of this limitation, most microprocessors today turn off the clock of inactive modules to save energy and dynamic power.
  - For example, if no floating-point instructions are executing, the clock of the floating-point unit is disabled.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Trends in Cost

- Although there are computer designs where costs tend to be less important — specifically supercomputers — cost-sensitive designs are of growing significance.
- Indeed, in the past 20 years, the use of technology improvements to lower cost, as well as increase performance, has been a major theme in the computer industry.
- Textbooks often ignore the cost half of cost-performance because costs change, thereby dating books, and because the issues are subtle and differ across industry segments.
- Yet an understanding of cost and its factors is essential for designers to make intelligent decisions about whether or not a new feature should be included in designs where cost is an issue.

Cost of computer manufacturing

- The cost of a manufactured computer component decreases over time even without major improvements in the basic implementation technology.
- The underlying principle that drives costs down is the *learning curve* — manufacturing costs decrease over time.
- The learning curve itself is best measured by change in *yield* — the percentage of manufactured devices that survives the testing procedure.
- Whether it is a chip, a board, or a system, designs that have twice the yield will have half the cost.

Learning curve

- Understanding how the learning curve improves yield is critical to projecting costs over a product's life.
- One example is that the price per megabyte of DRAM has dropped over the long term by 40% per year. Since DRAMs tend to be priced in close relationship to cost — with the exception of periods when there is a shortage or an oversupply — price and cost of DRAM track closely.
- Microprocessor prices also drop over time, but because they are less standardized than DRAMs, the relationship between price and cost is more complex.
- In a period of significant competition, price tends to track cost closely, although microprocessor vendors probably rarely sell at a loss.

Prices of Intel over the years

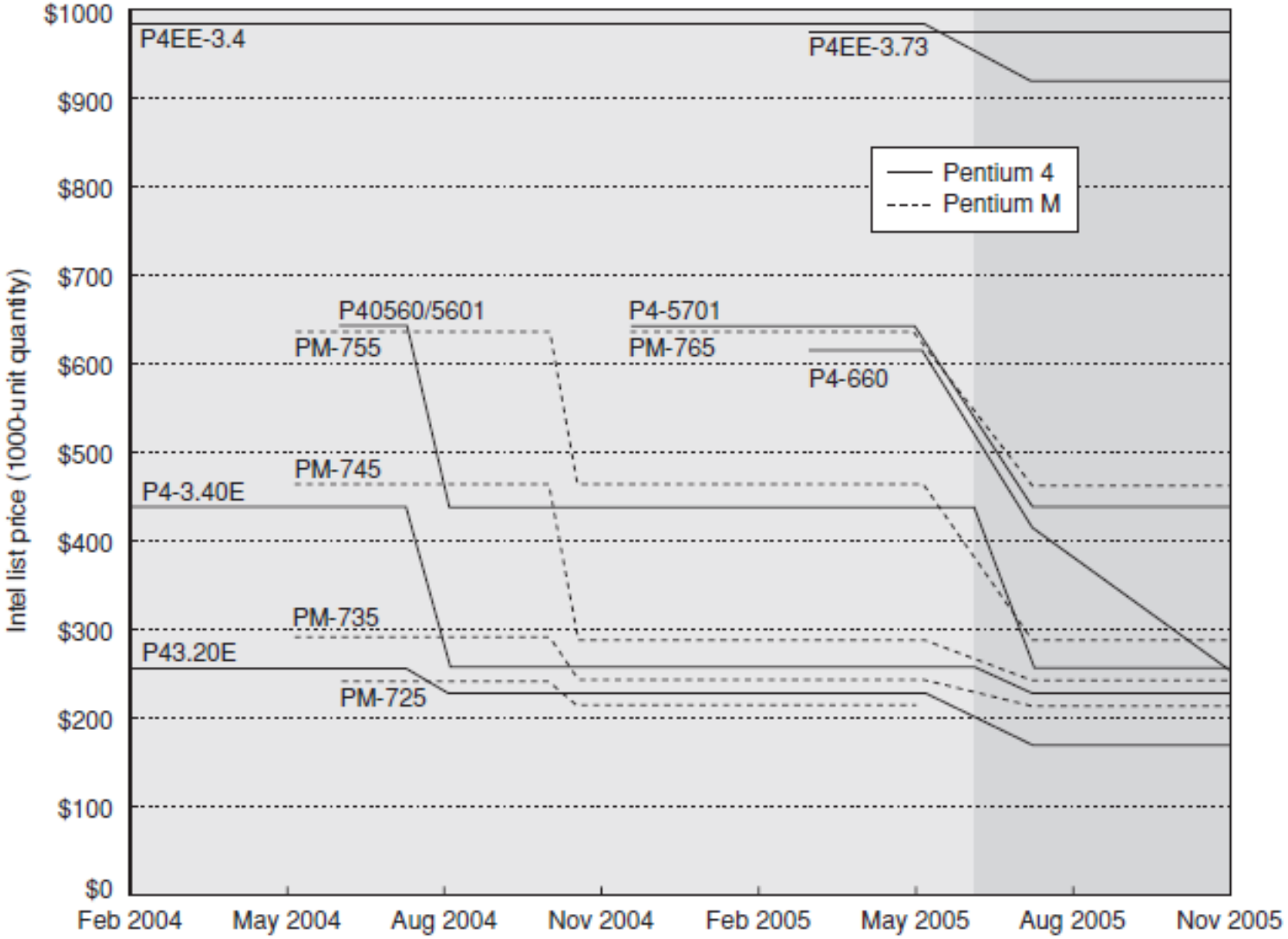


Figure 1.10 The price of an Intel Pentium 4 and Pentium M at a given frequency decreases over time as yield enhancements decrease the cost of a good die and competition forces price reductions. The most recent introductions will continue to decrease until they reach similar prices to the lowest-cost parts available today ($200). Such price decreases assume a competitive environment where price decreases track cost decreases closely. Data courtesy of *Microprocessor Report*, May 2005.

Cost of an Integrated Circuit

- In an increasingly competitive computer marketplace where standard parts — disks, DRAMs, and so on — are becoming a significant portion of any system's cost, integrated circuit costs are becoming a greater portion of the cost that varies between computers, especially in the high-volume, cost-sensitive portion of the market.

- Thus, computer designers must understand the costs of chips to understand the costs of current computers.

Die

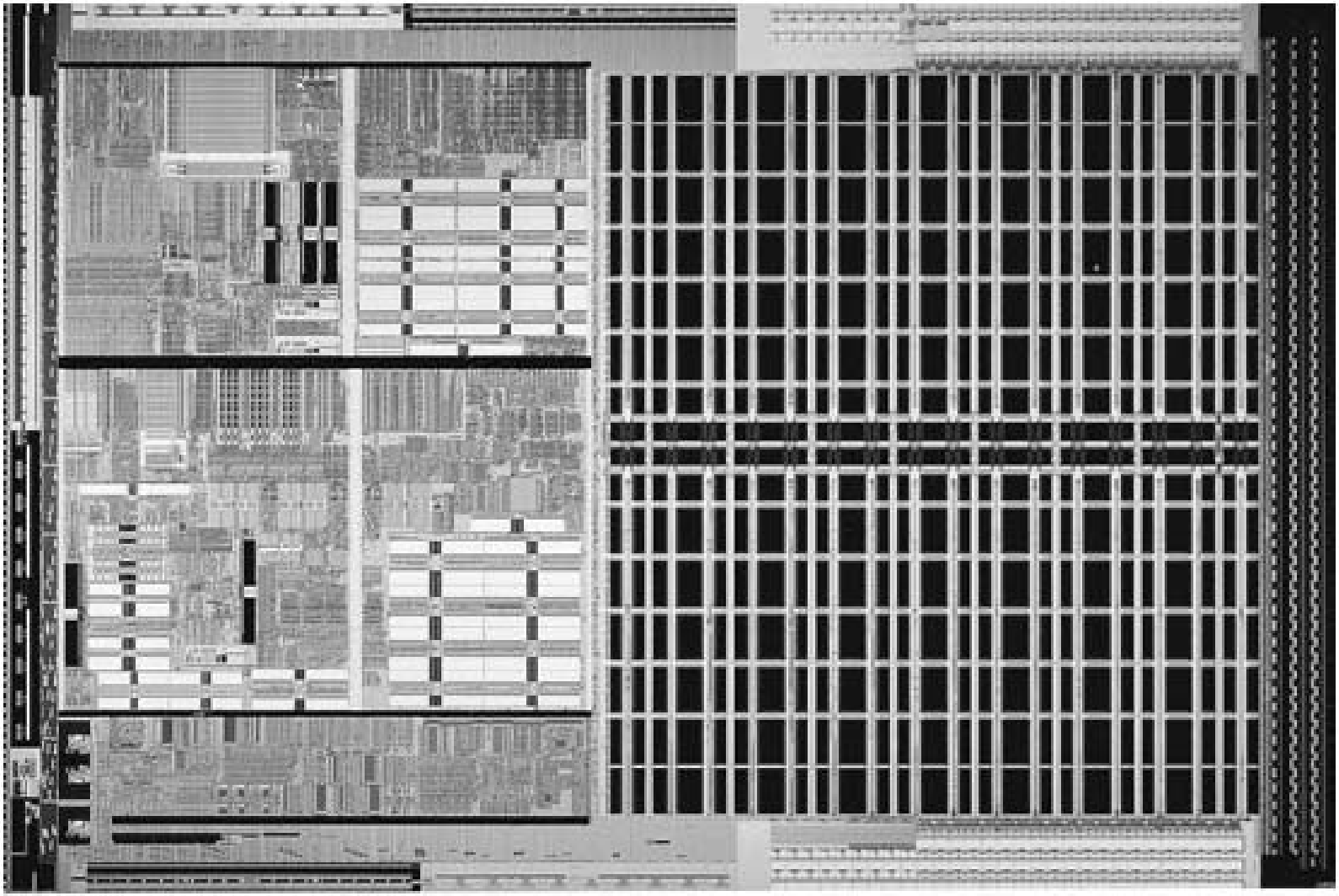


Figure 1.11 Photograph of an AMD Opteron microprocessor die. (Courtesy AMD.)

Wafer and dies

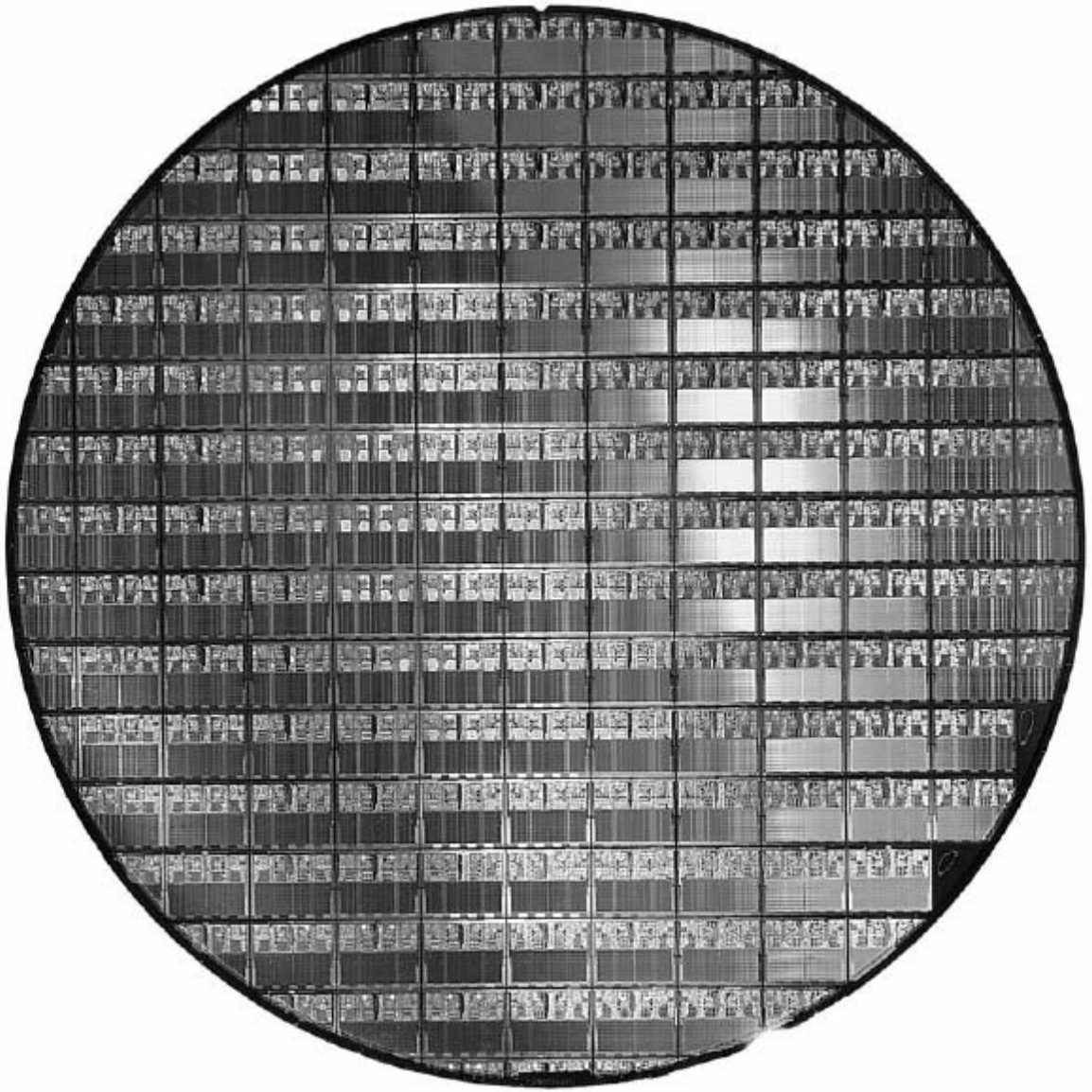


Figure 1.12 This 300mm wafer contains 117 AMD Opteron chips implemented in a 90 nm process. (Courtesy AMD.)

Cost of an Integrated Circuit

- Although the costs of integrated circuits have dropped exponentially, the basic process of silicon manufacture is unchanged: A *wafer* is still tested and chopped into *dies* that are packaged. Thus the cost of a packaged integrated circuit is:

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

Cost of Dies

- Learning how to predict the number of good chips per wafer requires first learning how many dies fit on a wafer and then learning how to predict the percentage of those that will work. From there it is simple to predict cost:

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

- The most interesting feature of this first term of the chip cost equation is its sensitivity to die size, shown below:
- The number of dies per wafer is approximately the area of the wafer divided by the area of the die. It can be more accurately estimated by:

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

Cost of Dies

- Processing of a 300 mm (12-inch) diameter wafer in a leading-edge technology costs between $5000 and $6000 in 2006.
- Assuming a processed wafer cost of $5500, the cost of the 1.00 cm^2 die would be around $13, but the cost per die of the 2.25 cm^2 die would be about $46, or almost four times the cost for a die that is a little over twice as large.

What should a computer designer remember about chip costs?

- The manufacturing process dictates the wafer cost, wafer yield, and defects per unit area, so the sole control of the designer is die area.
- In practice, because the number of defects per unit area is small, the number of good dies per wafer, and hence the cost per die, grows roughly as the square of the die area.
- The computer designer affects die size, and hence cost, both by what functions are included on or excluded from the die and by the number of I/O pins.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Dependability

- Historically, integrated circuits were one of the most reliable components of a computer.
- Although their pins may be vulnerable, and faults may occur over communication channels, the error rate inside the chip was very low.
- That conventional wisdom is changing as we head to feature sizes of 65 nm and smaller, as both transient faults and permanent faults will become more commonplace, so architects must design systems to cope with these challenges.

Service Level Agreements

- One difficult question is deciding when a system is operating properly.
- This philosophical point became concrete with the popularity of Internet services.
- Infrastructure providers started offering *Service Level Agreements* (SLA) or *Service Level Objectives* (SLO) to guarantee that their networking or power service would be dependable.
- For example, they would pay the customer a penalty if they did not meet an agreement more than some hours per month.
- Thus, an SLA could be used to decide whether the system was up or down.

Measures of dependability

- Systems alternate between two states of service with respect to an SLA:
  1. *Service accomplishment*, where the service is delivered as specified
  2. *Service interruption*, where the delivered service is different from the SLA
- Transitions between these two states are caused by *failures* (from state 1 to state 2) or *restorations* (2 to 1).
- Quantifying these transitions leads to the two main measures of dependability:
  - *Module reliability*
  - *Module availability*

Module reliability

- *Module reliability* is a measure of the continuous service accomplishment (or, equivalently, of the time to failure) from a reference initial instant.
- Hence, the *mean time to failure* (MTTF) is a reliability measure.
- The reciprocal of MTTF is a rate of failures, generally reported as failures per billion hours of operation, or *FIT* (for *failures in time)*.
- Thus, an MTTF of 1,000,000 hours equals $10^9 / 10^6$ or 1000 FIT.
- Service interruption is measured as *mean time to repair* (MTTR).
- *Mean time between failures* (MTBF) is simply the sum of MTTF + MTTR.

Module availability

- *Module availability* is a measure of the service accomplishment with respect to the alternation between the two states of accomplishment and interruption.
- For nonredundant systems with repair, module availability is:

$$\text{Module availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$$

- Note that reliability and availability are now quantifiable metrics, rather than synonyms for dependability.
- From these definitions, we can estimate reliability of a system quantitatively if we make some assumptions about the reliability of components and that failures are independent.

Example

- Assume a disk subsystem with the following components and MTTF:
  - 10 disks, each rated at 1,000,000-hour MTTF
  - 1 SCSI controller, 500,000-hour MTTF
  - 1 power supply, 200,000-hour MTTF
  - 1 fan, 200,000-hour MTTF
  - 1 SCSI cable, 1,000,000-hour MTTF
- Using the simplifying assumptions that the lifetimes are exponentially distributed and that failures are independent, compute the MTTF of the system as a whole.

Example

Answer The sum of the failure rates is

$$\text{Failure rate}_{\text{system}} = 10 \times \frac{1}{1{,}000{,}000} + \frac{1}{500{,}000} + \frac{1}{200{,}000} + \frac{1}{200{,}000} + \frac{1}{1{,}000{,}000}$$

$$= \frac{10 + 2 + 5 + 5 + 1}{1{,}000{,}000 \text{ hours}} = \frac{23}{1{,}000{,}000} = \frac{23{,}000}{1{,}000{,}000{,}000 \text{ hours}}$$

or 23,000 FIT. The MTTF for the system is just the inverse of the failure rate:

$$\text{MTTF}_{\text{system}} = \frac{1}{\text{Failure rate}_{\text{system}}} = \frac{1{,}000{,}000{,}000 \text{ hours}}{23{,}000} = 43{,}500 \text{ hours}$$

or just under 5 years.

Quantifying performance

- Having quantified the cost, power, and dependability of computer technology, we are ready to quantify performance.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Measuring Performance

- When we say one computer is faster than another, what do we mean?
- The user of a desktop computer may say a computer is faster when a program runs in less time, while an Amazon.com administrator may say a computer is faster when it completes more transactions per hour.
- The computer user is interested in reducing *response time*

  — the time between the start and the completion of an event also referred to as *execution time*.
- The administrator of a large data processing center may be interested in increasing *throughput* — the total amount of work done in a given time.

Comparing Alternatives

- In comparing design alternatives, we often want to relate the performance of two different computers, say, X and Y.
- The phrase “X is faster than Y” is used here to mean that the response time or execution time is lower on X than on Y for the given task. In particular, “X is *n* times faster than Y” will mean:

$$\frac{\text{Execution time}_{\text{Y}}}{\text{Execution time}_{\text{X}}} = n$$

Comparing Alternatives

- Since execution time is the reciprocal of performance, the following relationship holds:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

- The phrase “the throughput of X is 1.3 times higher than Y” signifies here that the number of tasks completed per unit time on computer X is 1.3 times the number completed on Y.

Measuring performance

- Unfortunately, time is not always the metric quoted in comparing the performance of computers.
- Hennessy & Patterson's position is that the only consistent and reliable measure of performance is the execution time of real programs, and that all proposed alternatives to time as the metric or to real programs as the items measured have eventually led to misleading claims or even mistakes in computer design.

Execution time

- Even execution time can be defined in different ways depending on what we count.
- The most straightforward definition of time is called *wall-clock time, response time,* or *elapsed time*, which is the latency to complete a task, including disk accesses, memory accesses, input/output activities, operating system overhead— everything.
- With multiprogramming, the processor works on another program while waiting for I/O and may not necessarily minimize the elapsed time of one program.
- Hence, we need a term to consider this activity. *CPU time* recognizes this distinction and means the time the processor is computing, *not* including the time waiting for I/O or running other programs. (Clearly, the response time seen by the user is the elapsed time of the program, not the CPU time.)

Benchmarks

- The best choice of benchmarks to measure performance are real applications, such as a compiler.
- Attempts at running programs that are much simpler than a real application have led to performance pitfalls.
- Examples include
  - *kernels*, which are small, key pieces of real applications;
  - *toy programs*, which are 100-line programs from beginning programming assignments, such as quicksort; and
  - *synthetic benchmarks*, which are fake programs invented to try to match the profile and behavior of real applications, such as Dhrystone.

Benchmark-specific flags

- Another issue is the conditions under which the benchmarks are run.
- One way to improve the performance of a benchmark has been with benchmark-specific flags;
  - these flags often caused transformations that would be illegal on many programs or would slow down performance on others.
- To restrict this process and increase the significance of the results, benchmark developers often require the vendor to use one compiler and one set of flags for all the programs in the same language (C or FORTRAN).

Benchmark suites

- Collections of benchmark applications, called *benchmark suites*, are a popular measure of performance of processors with a variety of applications.
- Of course, such suites are only as good as the constituent individual benchmarks.
- Nonetheless, a key advantage of such suites is that the weakness of any one benchmark is lessened by the presence of the other benchmarks.
- The goal of a benchmark suite is that it will characterize the relative performance of two computers, particularly for programs not in the suite that customers are likely to run.

EEMBC

- The EDN Embedded Microprocessor Benchmark Consortium (or EEMBC, pronounced “embassy”) is a set of 41 kernels used to predict performance of different embedded applications: automotive/industrial, consumer, networking, office automation, and telecommunications. EEMBC reports unmodified performance and “full fury” performance, where almost anything goes.
- Because they use kernels, and because of the reporting options, EEMBC does not have the reputation of being a good predictor of relative performance of different embedded computers in the field.

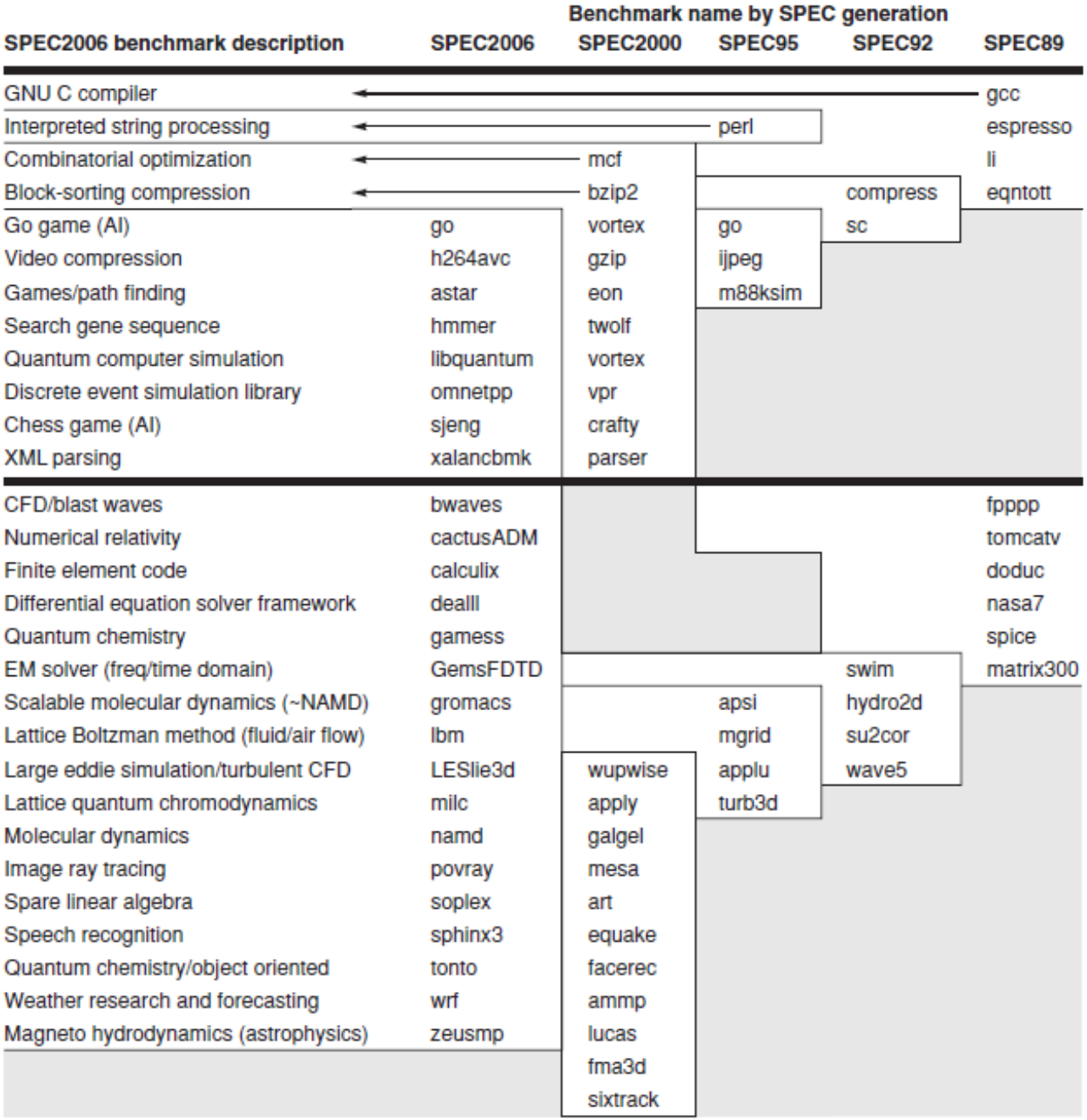
SPEC

- One of the most successful attempts to create standardized benchmark application suites has been the SPEC (Standard Performance Evaluation Corporation), which had its roots in the late 1980s efforts to deliver better benchmarks for workstations.
- Just as the computer industry has evolved over time, so has the need for different benchmark suites, and there are now SPEC benchmarks to cover different application classes.
- All the SPEC benchmark suites and their reported results are found at *www.spec.org.*
- Although we focus our discussion on the SPEC benchmarks, there are also many benchmarks developed for PCs running the Windows operating system.

Desktop Benchmark

- Desktop benchmarks divide into two broad classes:
  - processor-intensive benchmarks and
  - graphics-intensive benchmarks, although many graphics benchmarks include intensive processor activity.
- SPEC originally created a benchmark set focusing on processor performance (initially called SPEC89), which has evolved into its fifth generation: SPEC CPU2006, which follows SPEC2000, SPEC95, SPEC92, and SPEC89.
- SPEC CPU2006 consists of a set of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).

SPEC benchmarks



SPEC2006 programs and the evolution of the SPEC benchmarks over time, with integer programs above the line and floating-point programs below the line. Of the 12 SPEC2006 integer programs, 9 are written in C, and the rest in C++. For the floating-point programs the split is 6 in FORTRAN, 4 in C++, 3 in C, and 4 in mixed C and Fortran.

Server Benchmarks - 1

- Just as servers have multiple functions, so there are multiple types of benchmarks.
- The simplest benchmark is perhaps a processor throughput-oriented benchmark.
- SPEC CPU2000 uses the SPEC CPU benchmarks to construct a simple throughput benchmark where the processing rate of a multiprocessor can be measured by running multiple copies (usually as many as there are processors) of each SPEC CPU benchmark and converting the CPU time into a rate.
- This leads to a measurement called the SPECrate.

Server Benchmarks - 2

- Most server applications and benchmarks have significant I/O activity arising from either disk or network traffic, including benchmarks for file server systems, for Web servers, and for database and transaction processing systems.
- SPEC offers both a file server benchmark (SPECSFS) and a Web server benchmark (SPECWeb).
- SPECSFS is a benchmark for measuring NFS (Network File System) performance using a script of file server requests; it tests the performance of the I/O system (both disk and network I/O) as well as the processor.
- SPECSFS is a throughput-oriented benchmark but with important response time requirements.
- SPECWeb is a Web server benchmark that simulates multiple clients requesting both static and dynamic pages from a server, as well as clients posting data to the server.

Server Benchmarks - 3

- Transaction-processing (TP) benchmarks measure the ability of a system to handle transactions, which consist of database accesses and updates.
- Airline reservation systems and bank ATM systems are typical simple examples of TP;
  - more sophisticated TP systems involve complex databases and decision-making.
- In the mid-1980s, a group of concerned engineers formed the vendor-independent Transaction Processing Council (TPC) to try to create realistic and fair benchmarks for TP.
- The TPC benchmarks are described at *www.tpc.org*.

Reporting Performance Results

- The guiding principle of reporting performance measurements should be *reproducibility* — list everything another experimenter would need to duplicate the results.
- A SPEC benchmark report requires an extensive description of the computer and the compiler flags, as well as the publication of both the baseline and optimized results.
- In addition to hardware, software, and baseline tuning parameter descriptions, a SPEC report contains the actual performance times, shown both in tabular form and as a graph.
- A TPC benchmark report is even more complete, since it must include results of a benchmarking audit and cost information.
- These reports are excellent sources for finding the real cost of computing systems, since manufacturers compete on high performance and cost-performance.

Summarizing Performance Results

- In practical computer design, you must evaluate myriads of design choices for their relative quantitative benefits across a suite of benchmarks believed to be relevant.
- Likewise, consumers trying to choose a computer will rely on performance measurements from benchmarks, which hopefully are similar to the user's applications.
- In both cases, it is useful to have measurements for a suite of benchmarks so that the performance of important applications is similar to that of one or more benchmarks in the suite and that variability in performance can be understood.

Choosing summarizing measures

- Once we have chosen to measure performance with a benchmark suite, we would like to be able to summarize the performance results of the suite in a single number.
- A straightforward approach to computing a summary result would be to compare the arithmetic means of the execution times of the programs in the suite.
  - But some SPEC programs take four times longer than others, so those programs would be much more important if the arithmetic mean were the single number used to summarize performance.
- An alternative would be to add a weighting factor to each benchmark and use the weighted arithmetic mean as the single number to summarize performance.
- The problem would be then how to pick weights; since SPEC is a consortium of competing companies, each company might have their own favorite set of weights, which would make it hard to reach consensus.

Reference Computer

- Rather than pick weights, we could normalize execution times to a reference computer by dividing the time on the reference computer by the time on the computer being rated, yielding a ratio proportional to performance.
  - SPEC uses this approach, calling the ratio the SPECRatio.

- For example, suppose that the SPECRatio of computer A on a benchmark was 1.25 times higher than computer B; then you would know

$$1.25 = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\dfrac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_A}}{\dfrac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_B}} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

- Notice that the execution times on the reference computer drop out and the choice of the reference computer is irrelevant when the comparisons are made as a ratio, which is the approach we consistently use.

Geometric Mean

- Because a SPECRatio is a ratio rather than an absolute execution time, the mean must be computed using the *geometric* mean. (Since SPECRatios have no units, comparing SPECRatios arithmetically is meaningless.) The formula is:

$$\text{Geometric mean} = \sqrt[n]{\prod_{i=1}^{n} sample_i}$$

- In the case of SPEC, $sample_i$ is the SPECRatio for program *i*. Using the geometric mean ensures two important properties:
  1. The geometric mean of the ratios is the same as the ratio of the geometric means.
  2. The ratio of the geometric means is equal to the geometric mean of the performance ratios, which implies that the choice of the reference computer is irrelevant.
- Hence, the motivations to use the geometric mean are substantial, especially when we use performance ratios to make comparisons.

Relative performance in SPEC

Benchmarks	Ultra 5 Time (sec)	Opteron Time (sec)	SPECRatio	Itanium 2 Time (sec)	SPECRatio	Opteron/Itanium Times (sec)	Itanium/Opteron SPECRatios
wupwise	1600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16	2.16
art	2600	92.4	28.13	21.0	123.67	4.40	4.40
equake	1300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric mean			20.86		27.12	1.30	1.30

Figure 1.14 SPECfp2000 execution times (in seconds) for the Sun Ultra 5—the reference computer of SPEC2000—and execution times and SPECRatios for the AMD Opteron and Intel Itanium 2. (SPEC2000 multiplies the ratio of execution times by 100 to remove the decimal point from the result, so 20.86 is reported as 2086.) The final two columns show the ratios of execution times and SPECratios. This figure demonstrates the irrelevance of the reference computer in relative performance. The ratio of the execution times is identical to the ratio of the SPECRatios, and the ratio of the geometric means (27.12/20.86 = 1.30) is identical to the geometric mean of the ratios (1.30).

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Quantitative Principles of Computer Design

- Now that we have seen how to define, measure, and summarize performance, cost, dependability, and power, we can explore guidelines and principles that are useful in the design and analysis of computers.
- This section introduces important observations about design, as well as two equations to evaluate alternatives.

Take Advantage of Parallelism: System Level

- Taking advantage of parallelism is one of the most important methods for improving performance..
- The first example is the use of parallelism at the system level.
- To improve the throughput performance on a typical server benchmark, such as SPECWeb or TPC-C, multiple processors and multiple disks can be used.
- The workload of handling requests can then be spread among the processors and disks, resulting in improved throughput.
- Being able to expand memory and the number of processors and disks is called *scalability*, and it is a valuable asset for servers.

Processor Level

- At the level of an individual processor, taking advantage of parallelism among instructions is critical to achieving high performance.
- One of the simplest ways to do this is through pipelining.
  - The basic idea behind pipelining, is to overlap instruction execution to reduce the total time to complete an instruction sequence.
- A key insight that allows pipelining to work is that not every instruction depends on its immediate predecessor, and thus, executing the instructions completely or partially in parallel may be possible.

Principle of Locality

- Important fundamental observations have come from properties of programs.
- The most important program property that we regularly exploit is the *principle of locality:* Programs tend to reuse data and instructions they have used recently.
- A widely held rule of thumb is that a program spends 90% of its execution time in only 10% of the code.
- An implication of locality is that we can predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past.
- The principle of locality also applies to data accesses, though not as strongly as to code accesses.

Types of Locality

- Two different types of locality have been observed.
  - *Temporal locality* states that recently accessed items are likely to be accessed in the near future.
  - *Spatial locality* says that items whose addresses are near one another tend to be referenced close together in time.

Focus on the Common Case

- Perhaps the most important and pervasive principle of computer design is to focus on the common case:
  - In making a design trade-off, favor the frequent case over the infrequent case.
- This principle applies when determining how to spend resources, since the impact of the improvement is higher if the occurrence is frequent.
- In addition, the frequent case is often simpler and can be done faster than the infrequent case.
  - For example, when adding two numbers in the processor, we can expect overflow to be a rare circumstance and can therefore improve performance by optimizing the more common case of no overflow.
  - This may slow down the case when overflow occurs, but if that is rare, then overall performance will be improved by optimizing for the normal case.

Amdahl's Law

- The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's Law.
- Amdahl's Law states that:

  the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

- Amdahl's Law defines the *speedup* that can be gained by using a particular feature.

Amdahl's Law

- What is speedup? Suppose that we can make an enhancement to a computer that will improve performance when it is used. Speedup is the ratio:

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

Or alternatively:

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

- Speedup tells us how much faster a task will run using the computer with the enhancement as opposed to the original computer.

Amdahl's Law

- Amdahl's Law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

1. *The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement* — For example, if 20 seconds of the execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is 20/60. This value, which we will call $\text{Fraction}_{\text{enhanced}}$, is always less than or equal to 1.

Amdahl's Law

- *The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program.*
- This value is the time of the original mode over the time of the enhanced mode. If the enhanced mode takes, say, 2 seconds for a portion of the program, while it is 5 seconds in the original mode, the improvement is 5/2. We will call this value, which is always greater than 1, $Speedup_{enhanced}$.

Amdahl's Law

- The execution time using the original computer with the enhanced mode will be the time spent using the unenhanced portion of the computer plus the time spent using the enhancement:

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

- The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Amdahl's Law: Example 1

- Suppose that we want to enhance the processor used for Web serving.
- The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

$$\text{Fraction}_{\text{enhanced}} = 0.4, \text{Speedup}_{\text{enhanced}} = 10, \text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

Amdahl's Law: Example 2

- A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics.
- Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark.
  - One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10.
  - The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application.
- The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root.
- Compare these two design alternatives!!!

Amdahl's Law: Example 2

- We can compare these two alternatives by comparing the speedups:

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

- Improving the performance of the FP operations overall is slightly better because of the higher frequency.

The Processor Performance Equation

- Essentially all computers are constructed using a clock running at a constant rate.
- These discrete time events are called *ticks*, *clock ticks*, *clock periods*, *clocks*, *cycles*, or *clock cycles*.
- Computer designers refer to the time of a clock period by its duration (e.g., 1 ns) or by its rate, cycles per second, (e.g., 1 GHz).
- CPU time for a program can then be expressed two ways:

$$\textit{CPU time} = \textit{CPU clock cycles for a program} \times \textit{Clock cycle time}$$

Or alternatively:

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Clock cycles per instruction

- In addition to the number of clock cycles needed to execute a program, we can also count the number of instructions executed — the *instruction path length* or *instruction count* (IC).
- If we know the number of clock cycles and the instruction count, we can calculate the average number of *clock cycles per instruction* (CPI).
- Because it is easier to work with we use CPI.
- Designers sometimes also use *instructions per clock* (IPC), which is the inverse of CPI.
- CPI is computed as:

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

CPU Time

- By transposing instruction count in the above formula, clock cycles can be defined as IC × CPI. This allows us to use CPI in the execution time formula:

CPU time = Instruction count × Cycles per instruction × Clock cycle time

- Expanding the first formula into the units of measurement shows how the pieces fit together:

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

- As this formula demonstrates, processor performance is dependent upon three characteristics: clock cycle (or rate), clock cycles per instruction, and instruction count.
- Furthermore, CPU time is *equally dependent* on these three characteristics:
  - A 10% improvement in any one of them leads to a 10% improvement in CPU time.

Improving CPU performance

- Unfortunately, it is difficult to change one parameter in complete isolation from others because the basic technologies involved in changing each characteristic are interdependent:
  - *Clock cycle time* — Hardware technology and organization
  - *CPI* — Organization and instruction set architecture
  - *Instruction count* — Instruction set architecture and compiler technology
- Luckily, many potential performance improvement techniques primarily improve one component of processor performance with small or predictable impacts on the other two.

Total CPU cycles

- Sometimes it is useful in designing the processor to calculate the number of total processor clock cycles as:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

where IC_i represents number of times instruction *i* is executed in a program and CPI_i represents the average number of clocks per instruction for instruction *i*.

- This form can be used to express CPU time as

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

and overall CPI as

$$\text{CPI} = \frac{\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i}{\text{Instruction count}} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i$$

Processor Performance Equation: Example

- Consider again the example of the FPSQR.
- Suppose we have made the following measurements:
  - Frequency of FP operations = 25%
  - Average CPI of FP operations = 4.0
  - Average CPI of other instructions = 1.33
  - Frequency of FPSQR= 2%
  - CPI of FPSQR = 20
- Assume that the two design alternatives are to decrease the CPI of FPSQR to 2 or to decrease the average CPI of all FP operations to 2.5.
- Compare these two design alternatives using the processor performance equation.

Processor Performance Equation: Example

- First, observe that only the CPI changes; the clock rate and instruction count remain identical.
- We start by finding the original CPI with neither enhancement:

$$\text{CPI}_{\text{original}} = \sum_{i=1}^{n} \text{CPI}_i \times \left(\frac{\text{IC}_i}{\text{Instruction count}} \right)$$
$$= (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

- We can compute the CPI for the enhanced FPSQR by subtracting the cycles saved from the original CPI:

$$\text{CPI}_{\text{with new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \times (\text{CPI}_{\text{old FPSQR}} - \text{CPI}_{\text{of new FPSQR only}})$$
$$= 2.0 - 2\% \times (20 - 2) = 1.64$$

- We can compute the CPI for the enhancement of all FP instructions the same way or by summing the FP and non-FP CPIs. Using the latter gives us

$$\text{CPI}_{\text{new FP}} = (75\% \times 1.33) + (25\% \times 2.5) = 1.62$$

Processor Performance Equation: Example

- Since the CPI of the overall FP enhancement is slightly lower, its performance will be marginally better.
- Specifically, the speedup for the overall FP enhancement is:

$$\text{Speedup}_{\text{new FP}} = \frac{\text{CPU time}_{\text{original}}}{\text{CPU time}_{\text{new FP}}} = \frac{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{original}}}{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{new FP}}}$$

$$= \frac{\text{CPI}_{\text{original}}}{\text{CPI}_{\text{new FP}}} = \frac{2.00}{1.625} = 1.23$$

- We obtained this same speedup using Amdahl's Law.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Performance and Price-Performance for Desktop and Rack-Mountable Systems

- Although there are many benchmark suites for desktop systems, a majority of them are OS or architecture specific.
- In this section we examine the processor performance and price-performance of a variety of desktop systems using the SPEC CPU2000 integer and floating-point suites.
- SPEC CPU2000 summarizes processor performance using a geometric mean normalized to a Sun Ultra 5, with larger numbers indicating higher performance.

Systems

Vendor/model	Processor	Clock rate	L2 cache	Type	Price
Dell Precision Workstation 380	Intel Pentium 4 Xeon	3.8 GHz	2 MB	Desk	$3346
HP ProLiant BL25p	AMD Opteron 252	2.6 GHz	1 MB	Rack	$3099
HP ProLiant ML350 G4	Intel Pentium 4 Xeon	3.4 GHz	1 MB	Desk	$2907
HP Integrity rx2620-2	Itanium 2	1.6 GHz	3 MB	Rack	$5201
Sun Java Workstation W1100z	AMD Opteron 150	2.4 GHz	1 MB	Desk	$2145

Figure 1.15 Five different desktop and rack-mountable systems from three vendors using three different microprocessors showing the processor, its clock rate, L2 cache size, and the selling price. Figure 1.16 plots absolute performance and price performance. All these systems are configured with 1 GB of ECC SDRAM and approximately 80 GB of disk. (If software costs were not included, we added them.) Many factors are responsible for the wide variation in price despite these common elements. First, the systems offer different levels of expandability (with the Sun Java Workstation being the least expandable, the Dell systems being moderately expandable, and the HP BL25p blade server being the most expandable). Second, the cost of the processor varies by at least a factor of 2, with much of the reason for the higher costs being the size of the L2 cache and the larger die. In 2005, the Opteron sold for about $500 to $800 and Pentium 4 Xeon sold for about $400 to $700, depending on clock rates and cache size. The Itanium 2 die size is much larger than the others, so it's probably at least twice the cost. Third, software differences (Linux or a Microsoft OS versus a vendor-specific OS) probably affect the final price. These prices were as of August 2005.

Comparing Performance

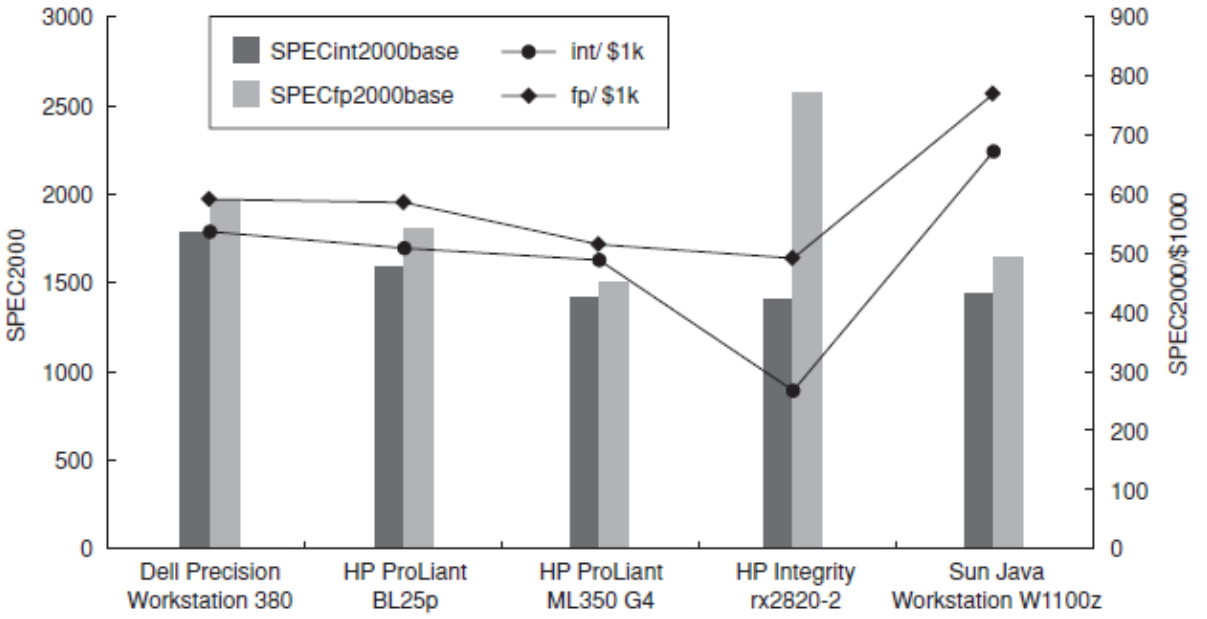


Figure 1.16 Performance and price-performance for five systems in Figure 1.15 measured using SPEC CINT2000 and CFP2000 as the benchmark. Price-performance is plotted as CINT2000 and CFP2000 performance per $1000 in system cost. These performance numbers were collected in January 2006 and prices were as of August 2005. The measurements are available online at *www.spec.org*.

The Itanium 2–based design has the highest floating-point performance but also the highest cost, and hence has the lowest performance per thousand dollars, being off a factor of 1.1–1.6 in floating-point and 1.8–2.5 in integer performance.

While the Dell based on the 3.8 GHz Intel Xeon with a 2 MB L2 cache has the high performance for CINT and second highest for CFP, it also has a much higher cost than the Sun product based on the 2.4 GHz AMD Opteron with a 1 MB L2 cache, making the latter the price-performance leader for CINT and CFP.

Performance and Price-Performance for Transaction-Processing Servers

- One of the largest server markets is online transaction processing (OLTP).
  - The standard industry benchmark for OLTP is TPC-C, which relies on a database system to perform queries and updates. Five factors make the performance of TPC-C particularly interesting.
- First, TPC-C is a reasonable approximation to a real OLTP application. Although this is complex and time-consuming, it makes the results reasonably indicative of real performance for OLTP.
- Second, TPC-C measures total system performance, including the hardware, the operating system, the I/O system, and the database system, making the benchmark more predictive of real performance.
- Third, the rules for running the benchmark and reporting execution time are very complete, resulting in numbers that are more comparable.
- Fourth, because of the importance of the benchmark, computer system vendors devote significant effort to making TPC-C run well.
- Fifth, vendors are required to report both performance and price-performance, enabling us to examine both.
- For TPC-C, performance is measured in transactions per minute (TPM), while price-performance is measured in dollars per TPM.

Systems

Vendor and system	Processors	Memory	Storage	Database/OS	Price
IBM eServer p5 595	64 IBM POWER 5 @1.9 GHz, 36 MB L3	64 cards, 2048 GB	6548 disks 243,236 GB	IBM DB2 UDB 8.2/ IBM AIX 5L V5.3	$16,669,230
IBM eServer p5 595	32 IBM POWER 5 @1.9 GHz, 36 MB L3	32 cards, 1024 GB	3298 disks 112,885 GB	Orcale 10g EE/ IBM AIX 5L V5.3	$8,428,470
HP Integrity rx5670 Cluster	64 Intel Itanium 2 @ 1.5 GHz, 6 MB L3	768 dimms, 768 GB	2195 disks, 93,184 GB	Orcale 10g EE/ Red Hat E Linux AS 3	$6,541,770
HP Integrity Superdome	64 Intel Itanium 2 @ 1.6 GHz, 9 MB L3	512 dimms, 1024 GB	1740 disks, 53,743 GB	MS SQL Server 2005 EE/MS Windows DE 64b	$5,820,285
IBM eServer pSeries 690	32 IBM POWER4+ @ 1.9 GHz, 128 MB L3	4 cards, 1024 GB	1995 disks, 74,098 GB	IBM DB2 UDB 8.1/ IBM AIX 5L V5.2	$5,571,349
Dell PowerEdge 2800	1 Intel Xeon @ 3.4 GHz, 2MB L2	2 dimms, 2.5 GB	76 disks, 2585 GB	MS SQL Server 2000 WE/ MS Windows 2003	$39,340
Dell PowerEdge 2850	1 Intel Xeon @ 3.4 GHz, 1MB L2	2 dimms, 2.5 GB	76 disks, 1400 GB	MS SQL Server 2000 SE/ MS Windows 2003	$40,170
HP ProLiant ML350	1 Intel Xeon @ 3.1 GHz, 0.5MB L2	3 dimms, 2.5 GB	34 disks, 696 GB	MS SQL Server 2000 SE/ MS Windows 2003 SE	$27,827
HP ProLiant ML350	1 Intel Xeon @ 3.1 GHz, 0.5MB L2	4 dimms, 4 GB	35 disks, 692 GB	IBM DB2 UDB EE V8.1/ SUSE Linux ES 9	$29,990
HP ProLiant ML350	1 Intel Xeon @ 2.8 GHz, 0.5MB L2	4 dimms, 3.25 GB	35 disks, 692 GB	IBM DB2 UDB EE V8.1/ MS Windows 2003 SE	$30,600

Figure 1.17 The characteristics of 10 OLTP systems, using TPC-C as the benchmark, with either high total performance (top half of the table, measured in transactions per minute) or superior price-performance (bottom half of the table, measured in U.S. dollars per transactions per minute). Figure 1.18 plots absolute performance and price performance, and Figure 1.19 splits the price between processors, memory, storage, and software.

Performance and Price-performance

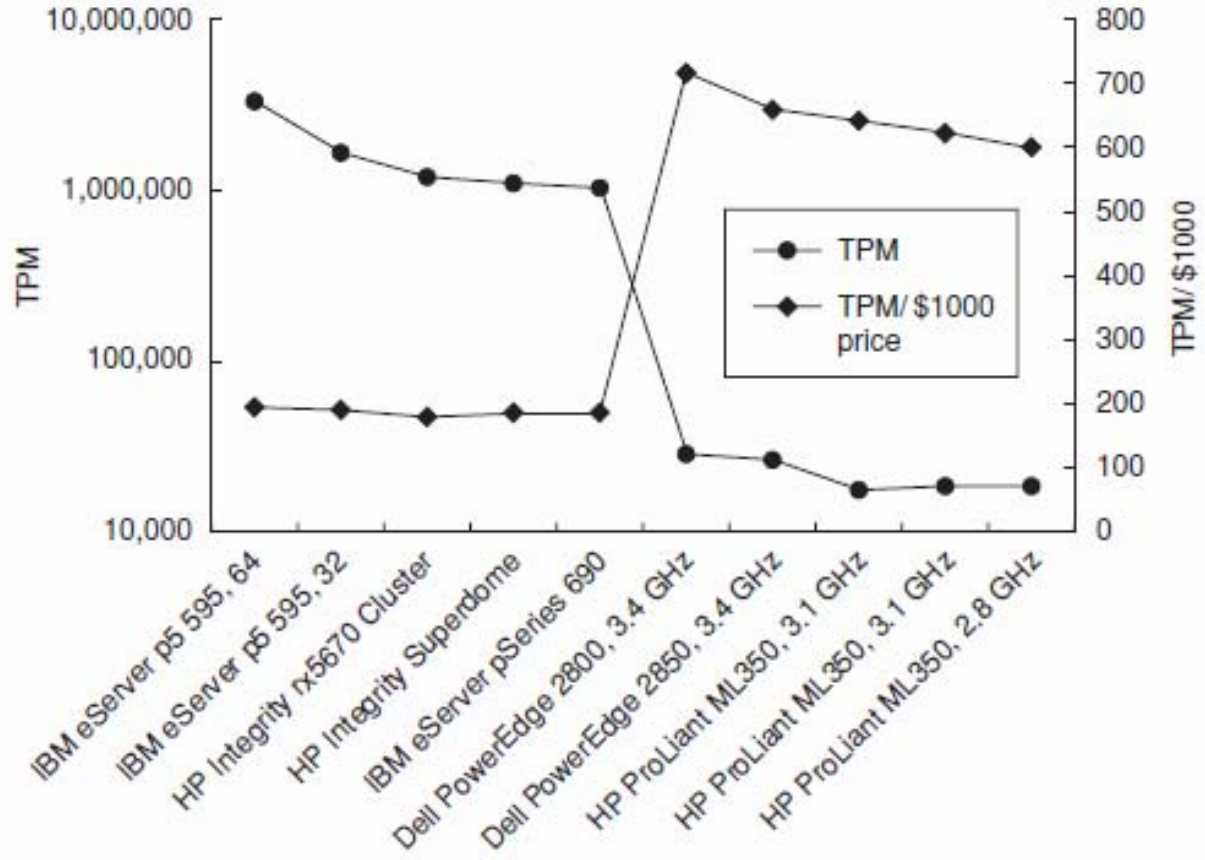


Figure 1.18 Performance and price-performance for the 10 systems in Figure 1.17 using TPC-C as the benchmark. Price-performance is plotted as TPM per $1000 in system cost, although the conventional TPC-C measure is $/TPM (715 TPM/$1000 = $1.40 $/TPM). These performance numbers and prices were as of July 2005. The measurements are available online at *www.tpc.org*.

Performance and Price-performance

- The highest-performing system is a 64-node shared-memory multiprocessor from IBM, costing a whopping $17 million.
  - It is about twice as expensive and twice as fast as the same model half its size, and almost three times faster than the third-place cluster from HP.
- The computers with the best price-performance are all uniprocessors based on Pentium 4 Xeon processors, although the L2 cache size varies.
- Notice that these systems have about three to four times better price-performance than the high-performance systems.
- Although these five computers also average 35–50 disks per processor, they only use 2.5–3 GB of DRAM per processor.

Outline

1.1 Introduction
1.2 Classes of Computers
1.3 Defining Computer Architecture
1.4 Trends in Technology
1.5 Trends in Power in Integrated Circuits
1.6 Trends in Cost
1.7 Dependability
1.8 Measuring, Reporting, and Summarizing Performance
1.9 Quantitative Principles of Computer Design
1.10 Putting It All Together: Performance and Price-Performance
1.11 Fallacies and Pitfalls

Pitfall and Fallacies

- The purpose of this section, is to explain some commonly held misbeliefs or misconceptions that you should avoid.
- We call such misbeliefs *fallacies*.
- When discussing a fallacy, we try to give a counterexample.
- We also discuss *pitfalls* — easily made mistakes.
- Often pitfalls are generalizations of principles that are true in a limited context.
- The purpose of these sections is to help to avoid making these errors in computers design.

Pitfall: Falling prey to Amdahl's Law

- Virtually every practicing computer architect knows Amdahl's Law.
- Despite this, we almost all occasionally expend tremendous effort optimizing some feature before we measure its usage.
- Only when the overall speedup is disappointing do we recall that we should have measured first before we spent so much effort enhancing it!

Pitfall: A single point of failure.

- The calculations of reliability improvement using Amdahl's Law show that dependability is no stronger than the weakest link in a chain.
- No matter how much more dependable we make the power supplies, as we did in our example, the single fan will limit the reliability of the disk subsystem.
- This Amdahl's Law observation led to a rule of thumb for fault-tolerant systems to make sure that every component was redundant so that no single component failure could bring down the whole system.

Fallacy: The cost of the processor dominates the cost of the system.

- Computer science is processor centric, perhaps because processors seem more intellectually interesting than memories or disks and perhaps because algorithms are traditionally measured in number of processor operations.
- This fascination leads us to think that processor utilization is the most important figure of merit.
- Indeed, the high-performance computing community often evaluates algorithms and architectures by what fraction of peak processor performance is achieved.
- This would make sense if most of the cost were in the processors.
  - This is not true!

Breakdown of costs

	Processor + cabinetry	Memory	Storage	Software
IBM eServer p5 595	28%	16%	51%	6%
IBM eServer p5 595	13%	31%	52%	4%
HP Integrity rx5670 Cluster	11%	22%	35%	33%
HP Integrity Superdome	33%	32%	15%	20%
IBM eServer pSeries 690	21%	24%	48%	7%
Median of high-performance computers	21%	24%	48%	7%
Dell PowerEdge 2800	6%	3%	80%	11%
Dell PowerEdge 2850	7%	3%	76%	14%
HP ProLiant ML350	5%	4%	70%	21%
HP ProLiant ML350	9%	8%	65%	19%
HP ProLiant ML350	8%	6%	65%	21%
Median of price-performance computers	7%	4%	70%	19%

Figure 1.19 Cost of purchase split between processor, memory, storage, and software for the top computers running the TPC-C benchmark in Figure 1.17. Memory is just the cost of the DRAM modules, so all the power and cooling for the computer is credited to the processor. TPC-C includes the cost of the clients to drive the TPC-C benchmark and the three-year cost of maintenance, which are not included here. Maintenance would add about 10% to the numbers here, with differences in software maintenance costs making the range be 5% to 22%. Including client hardware would add about 2% to the price of the high-performance servers and 7% to the PC servers.

Fallacy: Benchmarks remain valid indefinitely

- Several factors influence the usefulness of a benchmark as a predictor of real performance, and some change over time.
- A big factor influencing the usefulness of a benchmark is its ability to resist “cracking,” also known as “benchmark engineering” or “benchmarksmanship.”
- Once a benchmark becomes standardized and popular, there is tremendous pressure to improve performance by targeted optimizations or by aggressive interpretation of the rules for running the benchmark.
- Small kernels or programs that spend their time in a very small number of lines of code are particularly vulnerable.
- Example: next slide

Benchmarksmanship

- For example, despite the best intentions, the initial SPEC89 benchmark suite included a small kernel, called matrix300, which consisted of eight different 300 × 300 matrix multiplications.
- In this kernel, 99% of the execution time was in a single line (see SPEC [1989]).
- When an IBM compiler optimized this inner loop (using an idea called blocking), performance improved by a factor of 9 over a prior version of the compiler!
- This benchmark tested compiler tuning and was not, of course, a good indication of overall performance, nor of the typical value of this particular optimization.

Fallacy: The rated mean time to failure of disks is 1,200,000 hours or almost 140 years, so disks practically never fail.

- The current marketing practices of disk manufacturers can mislead users.
- How is such an MTTF calculated? Early in the process, manufacturers will put thousands of disks in a room, run them for a few months, and count the number that fail.
- They compute MTTF as the total number of hours that the disks worked cumulatively divided by the number that failed.
- One problem is that this number far exceeds the lifetime of a disk, which is commonly assumed to be 5 years or 43,800 hours.
- For this large MTTF to make some sense, disk manufacturers argue that the model corresponds to a user who buys a disk, and then keeps replacing the disk every 5 years — the planned lifetime of the disk.
  - The claim is that if many customers (and their great grandchildren) did this for the next century, on average they would replace a disk 27 times before a failure, or about 140 years.

Fallacy: The rated mean time to failure of disks is 1,200,000 hours or almost 140 years, so disks practically never fail.

- A more useful measure would be percentage of disks that fail.
- Assume 1000 disks with a 1,000,000-hour MTTF and that the disks are used 24 hours a day.
- If you replaced failed disks with a new one having the same reliability characteristics, the number that would fail in a year (8760 hours) is:

$$\text{Failed disks} = \frac{\text{Number of disks} \times \text{Time period}}{\text{MTTF}} = \frac{1000 \text{ disks} \times 8760 \text{ hours/drive}}{1{,}000{,}000 \text{ hours/failure}} = 9$$

- Stated alternatively, 0.9% would fail per year, or 4.4% over a 5-year lifetime.

Fallacy: The rated mean time to failure of disks is 1,200,000 hours or almost 140 years, so disks practically never fail.

- Moreover, high numbers are quoted assuming limited ranges of temperature and vibration; if they are exceeded, then all bets are off.
- A recent survey of disk drives in real environments [Gray and van Ingen 2005] claims about 3–6% of SCSI drives fail per year, or an MTTF of about 150,000–300,000 hours, and about 3–7% of ATA drives fail per year, or an MTTF of about 125,000–300,000 hours.
- The quoted MTTF of ATA disks is usually 500,000–600,000 hours.
- Hence, according to this report, real-world MTTF is about 2–4 times worse than manufacturer's MTTF for ATA disks and 4–8 times worse for SCSI disks.

Fallacy: Peak performance tracks observed performance

- The only universally true definition of peak performance is "*the performance level a computer is guaranteed not to exceed.*"
- Next slide shows the percentage of peak performance for four programs on four multiprocessors.
- It varies from 5% to 58%.
- Since the gap is so large and can vary significantly by benchmark, peak performance is not generally useful in predicting observed performance.

Fallacy: Peak performance tracks observed performance

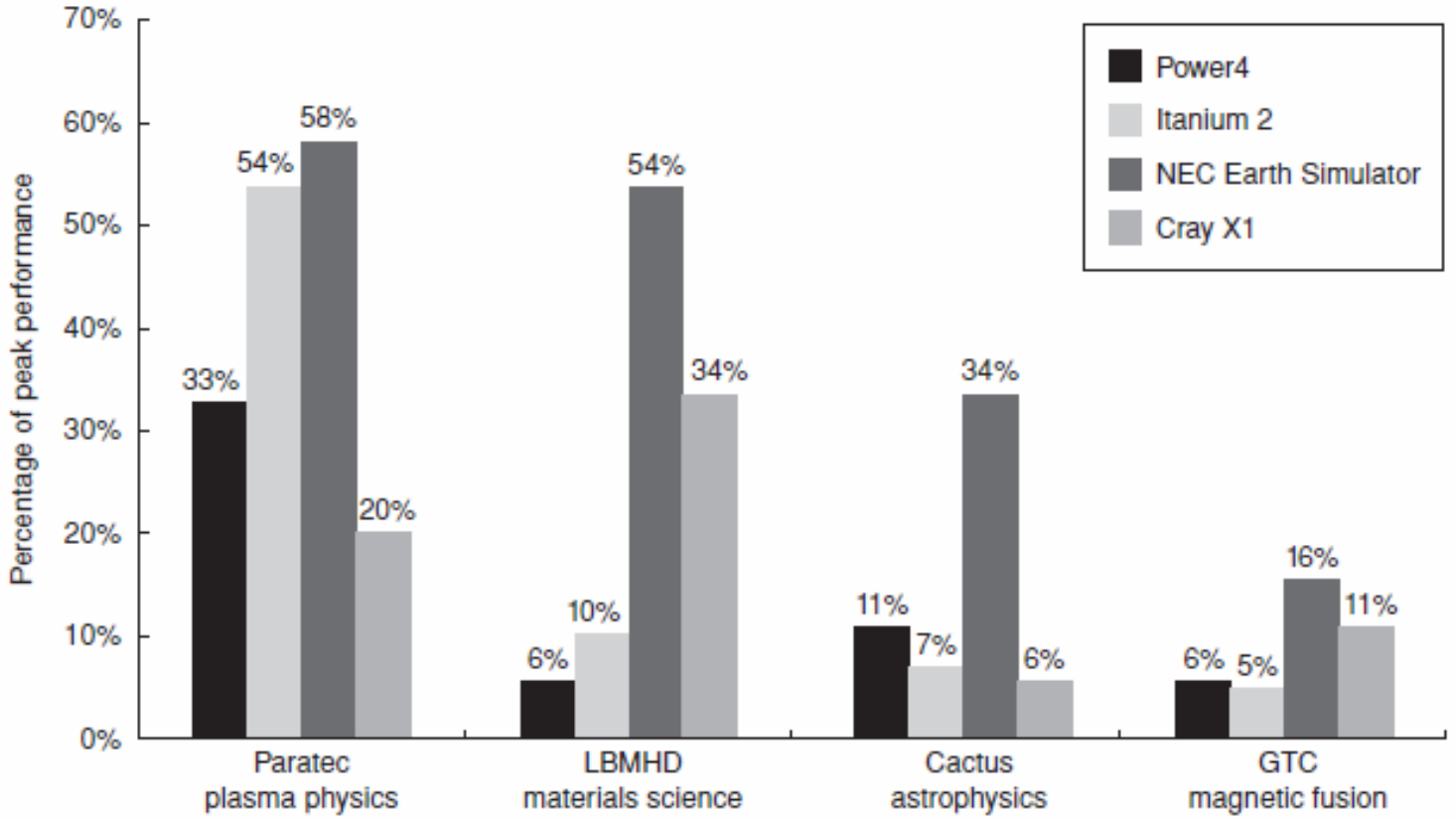


Figure 1.20 Percentage of peak performance for four programs on four multiprocessors scaled to 64 processors. The Earth Simulator and X1 are vector processors. (See Appendix F.) Not only did they deliver a higher fraction of peak performance, they had the highest peak performance and the lowest clock rates. Except for the Paratec program, the Power 4 and Itanium 2 systems deliver between 5% and 10% of their peak. From Oliker et al. [2004].

Pitfall: Fault detection can lower availability

- This apparently ironic pitfall is because computer hardware has a fair amount of state that may not always be critical to proper operation.
- In processors that try to aggressively exploit instruction-level parallelism, not all the operations are needed for correct execution of the program. Mukherjee et al. [2003] found that less than 30% of the operations were potentially on the critical path for the SPEC2000 benchmarks running on an Itanium 2.
- The same observation is true about programs.
- If a register is “dead” in a program — that is, the program will write it before it is read again — then errors do not matter.

  If you were to crash the program upon detection of a transient fault in a dead register, it would lower availability unnecessarily.
- The pitfall is in detecting faults without providing a mechanism to correct them.

End of Lecture 1

- Readings
  - Book: Chapter 1