CS-4515
Computer Architecture

Professor Hugh C. Lauer
Professor Therese M. Smith

CS-4515, Computer Architecture

Instructors

Hugh C. Lauer

Therese M. Smith

Teaching Assistant — Menghai Pan

What is Computer Architecture?

- Class Discussion

- (Laptops closed, please!)

In the bad old days …

- …it used to be about Instruction Set design
- I.e., what operation codes, data types, and addressing modes were made available to the *assembly language* programmer

Not any more!

- ***No one*** **writes in assembly language or machine code any more**
  - See H & P, page 2, third paragraph
- **Instruction set is a *private contract* between hardware architect and compiler writer**
  - Very few exceptions
- **Device interfaces are private contract between hardware developer and OS**
  - … or device driver writer

Apple has changes the Macintosh instruction set twice in its lifetime …

… and the endian-ness once …

… without upsetting its customers or existing software!

Vendor-independent Operating Systems

- **Linux**
- **Some flavors of Unix**
- **Mac-OS**
    - M68000 *vs.* PPC *vs.* Pentium
- **Windows NT (!)**
- **Android, iOS, etc.**

What is Computer Architecture?

- **Much more than Instruction Set Design**
- **Addresses hurdles of implementation**
  - Speed
  - Power
  - Energy
  - Size
  - Appropriateness for applications
- **Highly quantitative approach**
  - Lots of simulations, lots of analysis

An Integrated Approach

- **Functioning of the complete system**
  - Hardware processor(s)
  - Memory, devices, long-term storage
  - Runtime system, compilers, operating system, etc.
  - Applications

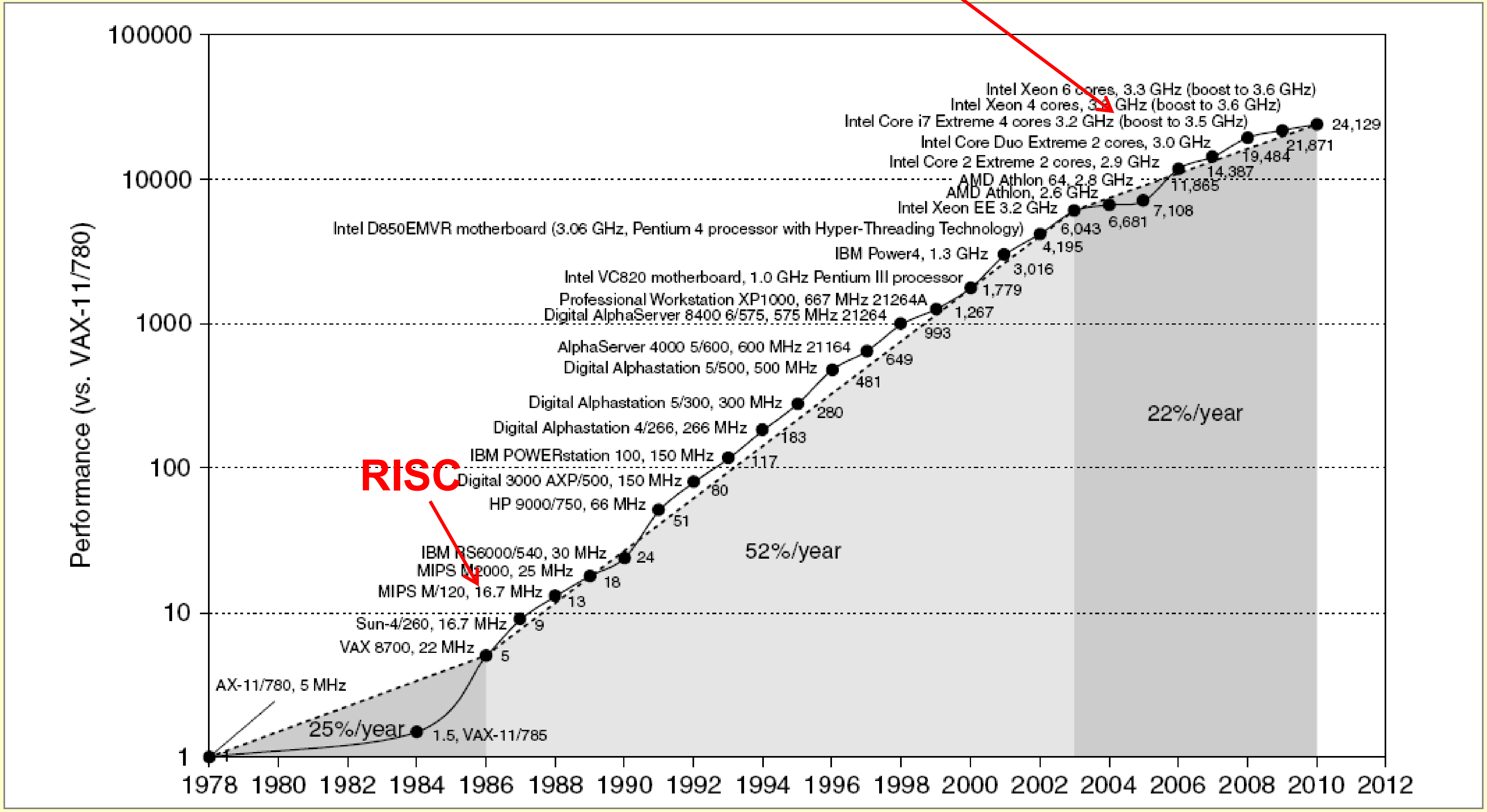
- **"Real" computer architecture:–**
  - Specific requirements of the target machine
  - Design to maximize performance within constraints: cost, power, and availability
  - Includes ISA, microarchitecture, hardware

Computer Technology

- **Performance improvements:–**
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by HLL compilers, UNIX
    - Lead to RISC architectures
  - Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages

Single Processor Performance

Move to multi-processor



Copyright © 2012, Elsevier Inc. All rights reserved.

Current Trends in Architecture

- **Cannot continue to leverage Instruction-Level parallelism (ILP)**
  - Single processor performance improvement ended in 2003
- **New models for performance:**
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- **These require explicit restructuring of the application**

Classes of Computers

- **Personal Mobile Device (PMD)**
  - e.g. smart phones, tablet computers
  - Emphasis on energy efficiency and real-time
- **Desktop Computing**
  - Emphasis on price-performance
- **Servers**
  - Emphasis on availability, scalability, throughput
- **Clusters / Warehouse Scale Computers**
  - Used for “Software as a Service (SaaS)”
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- **Embedded Computers**
  - Emphasis: price

Parallelism

- **I.e., doing more than one thing at a time**

- **Classes of parallelism in applications:**
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)

- **Classes of architectural parallelism:**
  - Instruction-Level Parallelism (ILP)
  - Vector architectures/Graphic Processor Units (GPUs)
  - Thread-Level Parallelism
  - Request-Level Parallelism

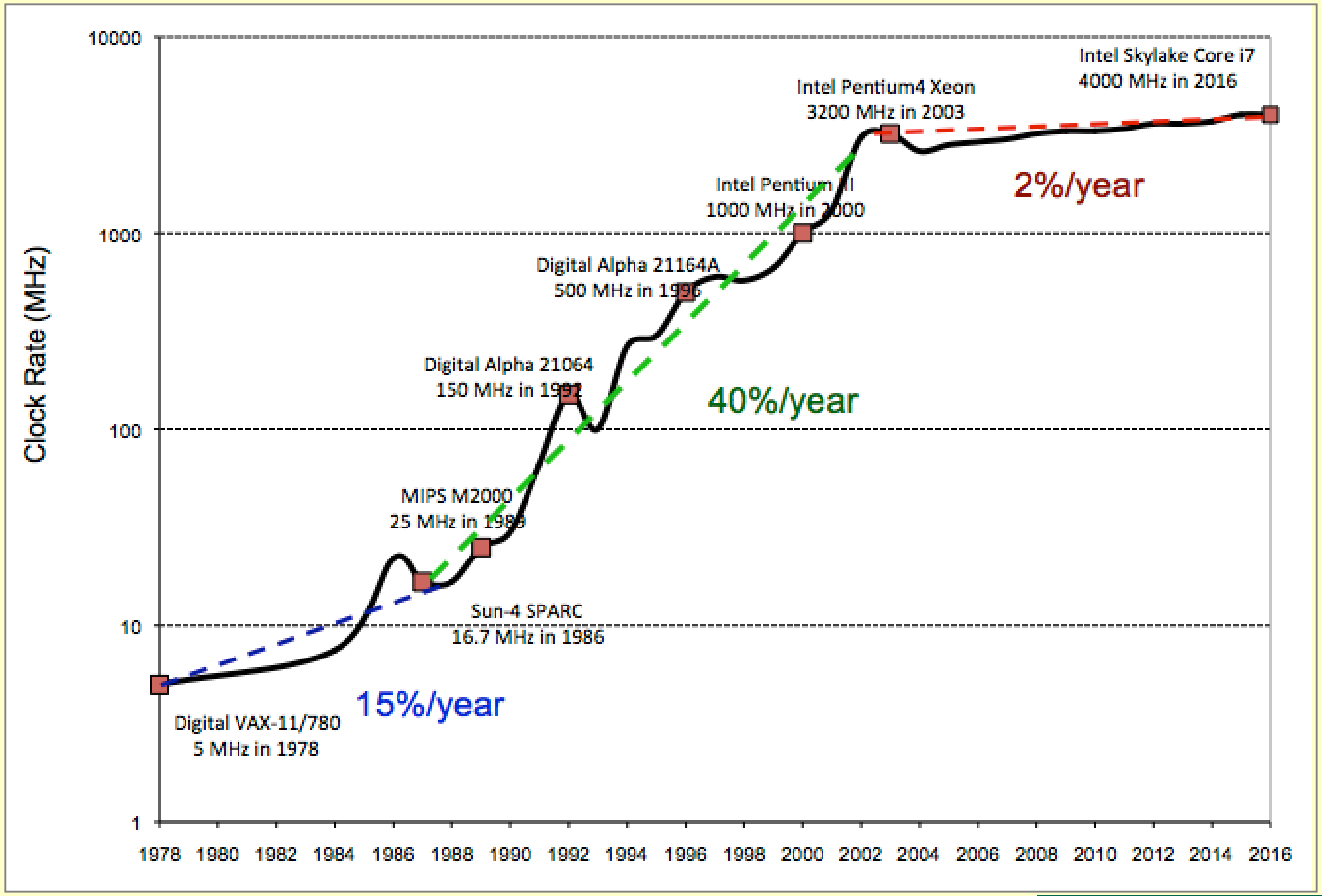
Flynn’s Taxonomy

- **Single instruction stream, single data stream (SISD)**
- **Single instruction stream, multiple data streams (SIMD)**
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- **Multiple instruction streams, single data stream (MISD)**
  - No commercial implementation
  - No one really knows what this means!
- **Multiple instruction streams, multiple data streams (MIMD)**
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

Trends in Technology

- **Integrated circuit technology**
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- **DRAM capacity: 25-40%/year (already slowing in 2015!)**
- **Flash capacity: 50-60%/year**
  - 15-20X cheaper/bit than DRAM
- **Magnetic disk technology: 40%/year**
  - 15-25X cheaper/bit then Flash
  - 300-500X cheaper/bit than DRAM

2017 Update — clock speeds



Confidential:–
From 6th edition
Not yet published

2017 Update

- **Dennard scaling ended ~2014**
  - Formerly, power density was constant as transistors got smaller ⇒
    - Linear dimension shrinks by factor of 2 *implies* voltage and current dropped by factor of two ⇒
    - … newer, faster chips used *less* power!
  - No longer!
    - Voltage and current cannot drop any farther without sacrificing the dependability of the integrated circuits
    - *Static power* (i.e., the power required just to keep the transistors working) becomes an increasing large fraction of overall power!

2017 update (continued)

- **Multiprocessing is no longer a "silver bullet"**
  - Amdahl's Law is alive and well!
- **Moore's Law is (pretty much) done!**
- **…**

Digression: Making a chip

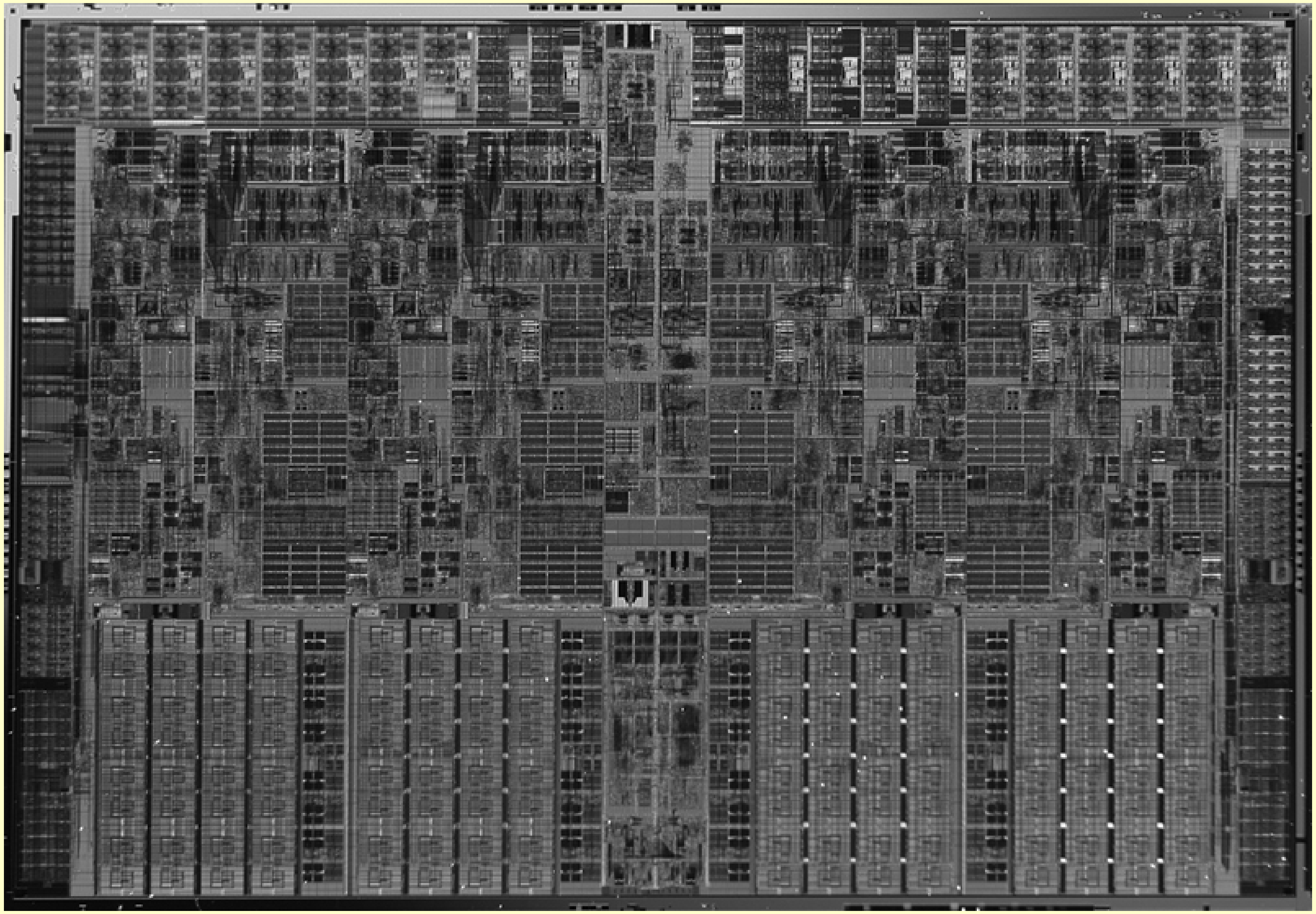


Figure 1.13 Photograph of an Intel Core i7 microprocessor die, which is evaluated in Chapters 2 through 5. The dimensions are 18.9 mm by 13.6 mm (257 mm2) in a 45 nm process. (Courtesy Intel.)

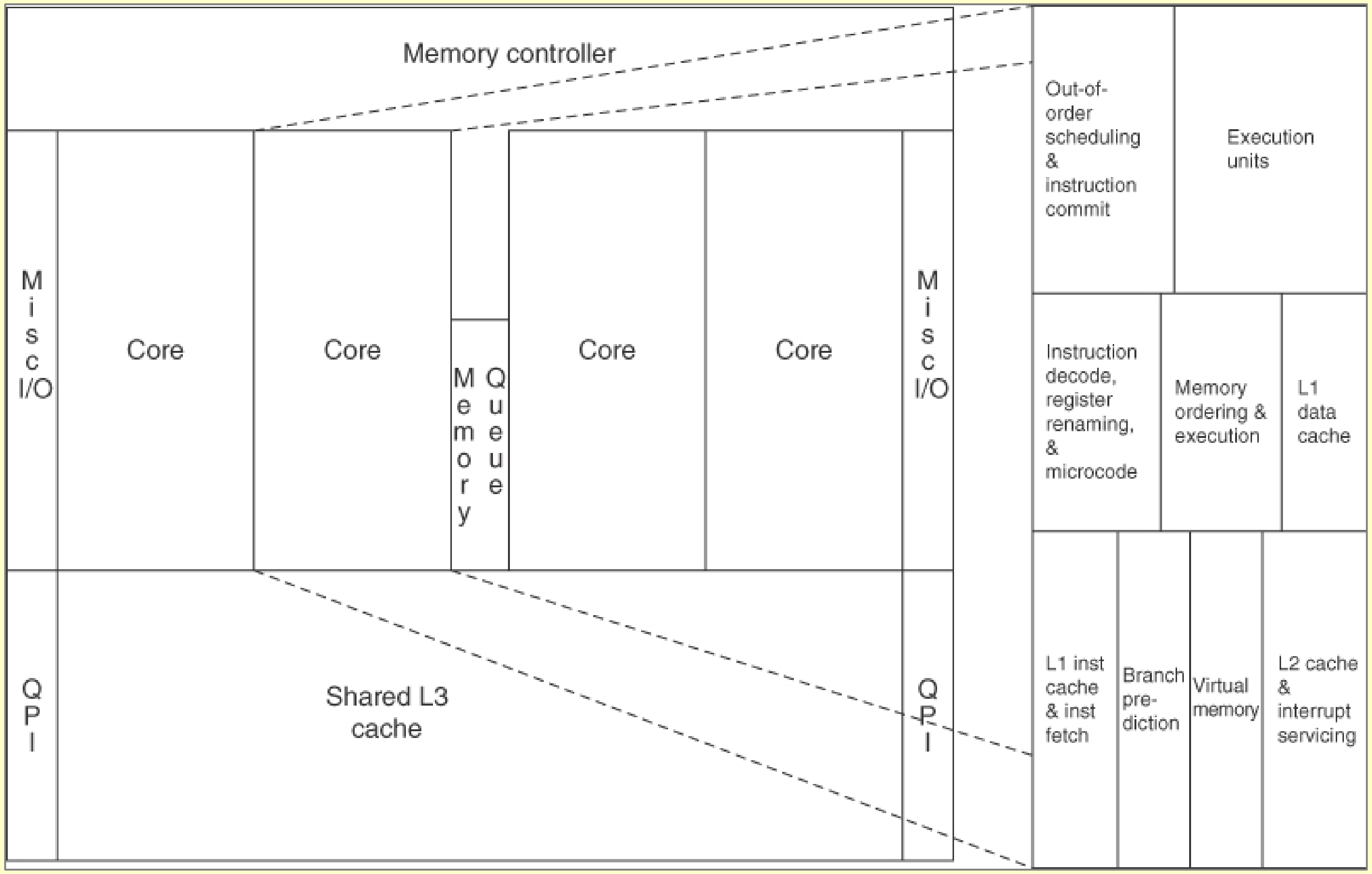

Memory controller
Misc I/O
Core
Core
Memory
Queue
Core
Core
Misc I/O
QPI
Shared L3 cache
QPI
Out-of-order scheduling & instruction commit
Execution units
Instruction decode, register renaming, & microcode
Memory ordering & execution
L1 data cache
L1 inst cache & inst fetch
Branch pre-diction
Virtual memory
L2 cache & interrupt servicing

Figure 1.14 Floor-plan of Core i7 die in Figure 1.13 on left with close-up of floor-plan of second core on right.

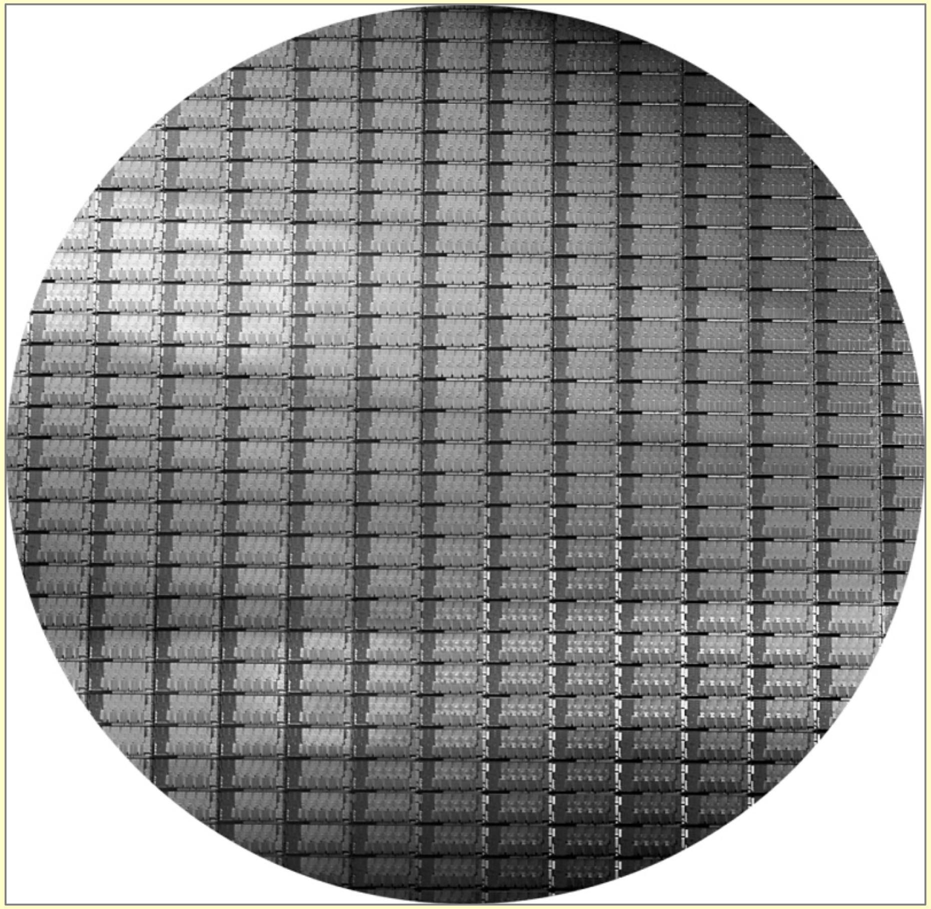


Figure 1.15 This 300 mm wafer contains 280 full Sandy Bridge dies, each 20.7 by 10.5 mm in a 32 nm process. (Sandy Bridge is Intel's successor to Nehalem used in the Core i7.) At 216 mm2, the formula for dies per wafer estimates 282. (Courtesy Intel.)

Cost of Integrated Circuit

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

$5000–6000

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

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

Accounts for partial dies at edge of wafer

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

Bose-Einstein formula (empirically derived)

Based on semi-conductor process

Questions?

Definition – RISC

- ***R**educe **I**nstruction **S**et **C**omputer*
  - *vs.* CISC – *Complex Instruction Set Computer*

- **In the bad old days …**
  - Instruction sets included features to simplify coding of applications in assembly language
  - Complex, many cycles, difficult to interrupt, virtualize, etc.
- **RISC machines are much simpler**

CISC Examples

- **BLT – block transfer**
  - Move *n* bytes from one location to another
  - Multiple cycles
  - Special cases when not aligned, etc.
  - Restart issues after page faults, etc.
- **IBM 360**
  - Translate
  - Translate and Test
  - Edit and Mark
    - Similar functionality to `strtok()` in C
- **Variable length instructions**

RISC Principles

- **Focus on common case**
  - Let software handle the special cases
- **Each instruction is exactly one word**
- **Each instruction does exactly one thing**
  - E.g., Load, Store, Add, Subtract, Jump or Branch
- **Simple addressing modes**
  - Register + offset
- **Each instruction takes exactly one cycle**
  - Except floating point arithmetic

RISC Principles

Simpler design & higher performance

- **Focus on common case**
  - Let software handle the special cases
- **Each instruction is exactly one word**
- **Each instruction does exactly one thing**
  - E.g., Load, Store, Add, Subtract, Jump or Branch
- **Simple addressing modes**
  - Register + offset
- **Each instruction takes exactly one cycle**
  - Except floating point arithmetic

Intel Processors — x86, x86_64, etc

- **RISC machines in CISC clothing!**
- **Core (internal) instructions are RISC in nature**
  - With very simple addressing modes
- **Complex x86 instructions ⇒ subroutines of core instructions**
  - Stored and invoked in hardware
  - Interruptible, page faults, can be restarted, etc.

Notable RISC Machines

- **SPARC**
  - Directly derived from Patterson's Berkeley RISC
- **MIPS**
  - Hennessy's team at Stanford; SGI processors
  - Primarily sold as embedded μprocessors for other chips
- **Power PC**
  - IBM-Motorola-Apple
- **HP PA-RISC**
  - Incorporated Apollo DN-10000 after merger
- **DEC Alpha**
- **ARM**
  - Widely used embedded μprocessor for cell phones, PDAs, etc.
- **…**

Why RISC?

Performance

Performance

Performance

Taking advantage of parallelism

(Almost) everything in this course is about parallelism

(Almost) everything in the textbook is about parallelism

Amdahl's Law

- Let *P* be ratio of time in parallelizable portion of algorithm to total time of algorithm
- I.e.,

$$P = \frac{\text{Exec time of parallelizable part}}{\text{Total execution time}}$$

Amdahl's Law (continued)

Very Important!

- If T_S is execution time in serial environment, then

$$T_P = T_S \times \left((1-P) + \frac{P}{N} \right)$$

is execution time on *N* processors

- I.e., *speedup* factor is

$$S = \frac{T_S}{T_P} = \frac{1}{(1-P) + \frac{P}{N}}$$

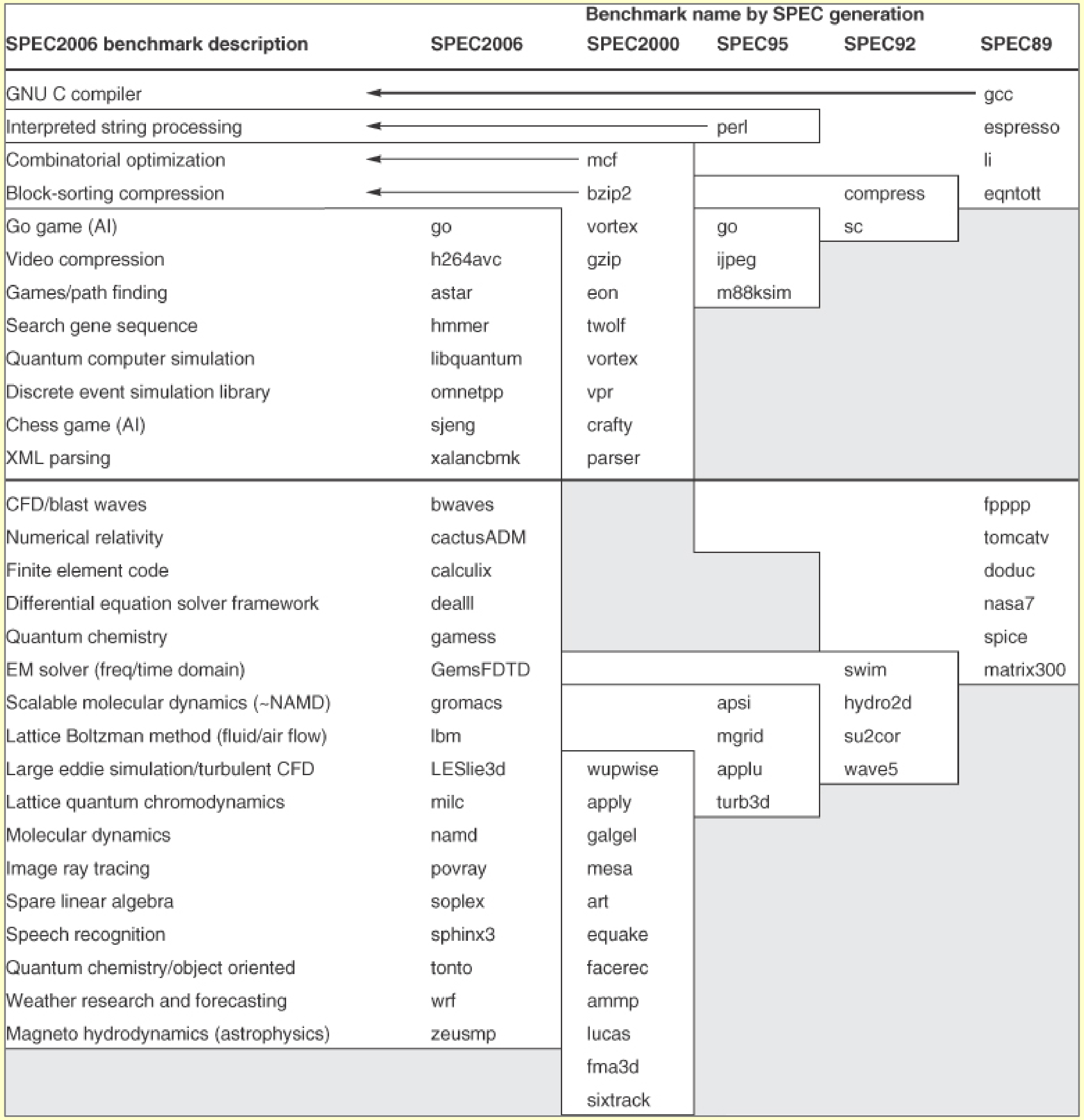
Diminishing returns in N

Described (in next edition of H&P) **as**

Amdahl's Disheartening Law!

Measuring Performance

- **Typical performance metrics:**
  - Response time
  - Throughput
- **Speedup of X relative to Y**
  - Execution time$_Y$ / Execution time$_X$
- **Execution time**
  - Wall clock time: includes all system overheads
  - CPU time: only computation time
- **Benchmarks**
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)



		Benchmark name by SPEC generation			
SPEC2006 benchmark description	**SPEC2006**	**SPEC2000**	**SPEC95**	**SPEC92**	**SPEC89**
GNU C compiler	←	←	←	←	gcc
Interpreted string processing	←	←	perl		espresso
Combinatorial optimization	←	mcf			li
Block-sorting compression	←	bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

Fig. 1.16

Processor Performance Equation

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

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

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

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

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

Page 49

Principles of Computer Design

- **Different instruction types having different CPIs**

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

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

Processor Performance Equation

- **Needed for Homework #1**
- **See §1.9**

Homework Assignment #1

- Do *one of* the following from Hennessy & Patterson
  - 1.2
  - 1.5
  - 1.12
  - 1.15
- Due *Thursday, March 23, 6:00 PM*

- *Submit via Canvas*
  - This is Homework #1

Questions?