上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Advanced Computer Architecture (X033503)

Quan Chen

Department of Computer Science and Engineering

Course Information

- Name: Quan Chen 陈全
- Title: Tenure-Track Associate Professor
- Office: SEIEE 3-522
- Email: chen-quan@cs.sjtu.edu.cn
- Homepage: www.cs.sjtu.edu.cn/~chen-quan
- Course site:
  - http://cc.sjtu.edu.cn/G2S/site/aca.html

Who am I?

Before Join SJTU as a Faculty

- 2014~2016, Postdoc @ Computer Science, *University of Michigan*, Ann Arbor, MI, USA
- 2013~2014, Research Assistant @ Computer Science, *Columbia University*, New York, NY, USA
- 2007~2014, Master + Ph.D @ Computer Science, *SJTU*

Research Interest: Computer Architecture, System

- Task Scheduling for Multi-core/Many-core
- Processing System for Big Data
- QoS-aware Scheduling in Datacenter
- Runtime Optimization for Various Hardware

Course Outline

- Basic Principles in Computer Architecture
- Multiprocessor, many-core Design
- Power-aware CPU Design
- Scheduling Techniques
- GPU-related Techniques
- Datacenter-related Techniques
- Neural Network Chips
- Design of TianHe Supercomputer
- …

Coping with This Course

- 4 paper-reading homeworks 40%
  - Read a paper on the latest computer architecture conference (ISCA, MICRO, ASPLOS, HPCA), and write a summary
- One paper critique 30%
  - Write a 3~4 page paper critique on a specific topic
  - A paper draft is acceptable as well
- One 15-min presentation 30%
  - Selecting a top-tier paper, and presenting it to your classmates
  - Presenting your architecture-related work (preferred)

Paper Critique

- *A critique is not a summary of the paper!* A critique requires judgement, synthesis, analysis, and opinion.
  - The paper review
  - The strength of the paper
  - The weakness of the paper
  - Your conclusion

Course Material

- *Main Textbook:*
  - (CAA) Computer Architecture: A Quantitative Approach Hennessey and Patterson, 5th Edition (2012)
  - M Dubois et al, Parallel Computer Organization and Design, 1st edition
- *Prerequisite:* Computer Organization and Design
  - *Reference book:* (COD) Computer Organization and Design: The Hardware/Software Interface 4th Edition. David A.Patterson，John L.Hennessy
- Other research papers we bring up in class or post online

Prerequisites

- Students with varied backgrounds
- Prerequisites – Basic Computer architecture
  - Basic computer organizaion (like in CS 209)
  - Basic architecture (like in CS 359)
  - Read and write in an assembly language, C, C++,..MIPS/ARM ISA preferred

Prerequisites

What you should know from CS 209 ?

Textbook: (COD) Computer Organization and Design: The Hardware/Software Interface 4th Edition. David A.Patterson，John L.Hennessy

- Basic machine structure
  - processor (data path, control, arithmetic), memory, I/O
- Instruction Sets
  - Types of instructions, instruction formats
- Read and write in an assembly language
  - MIPS preferred
- ¨Understand the concepts of pipelining and virtual memory

Prerequisites

What you should know from CS 359 ?

Textbook: (CAA) Computer Architecture: A Quantitative Approach
Hennessey and Patterson, 5th Edition (2012)

- Memory Design: Memory Hierarchy, Cache Memory (App. B, Ch.2)

- Processor Design: Pipelining (App. C, Ch.3)
  - processor (data path, control/branching schemes, arithmetic), Tomasulo, basic speculation, hazards (RAW, WAR, WAW)

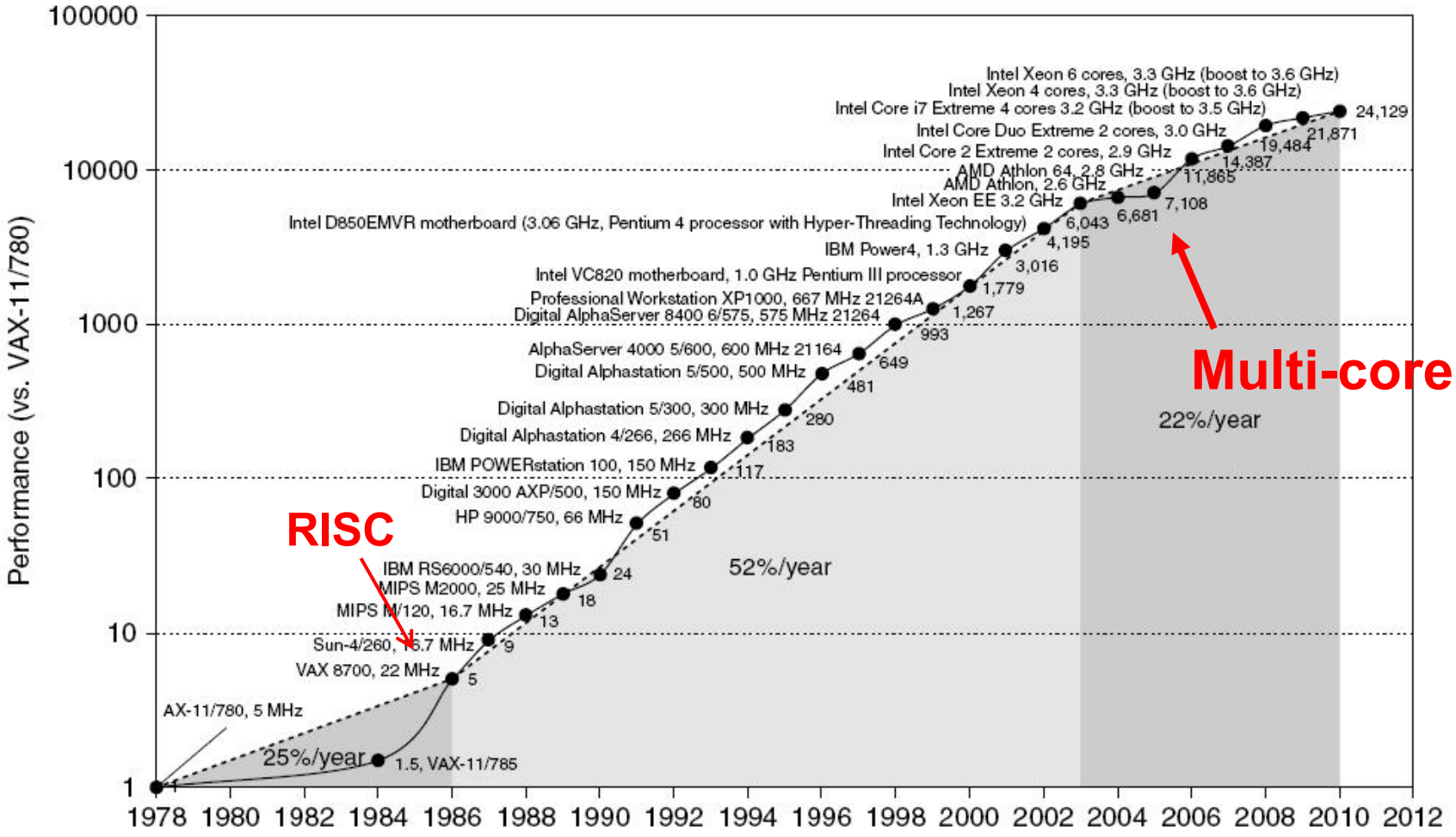
Fundamentals of Quantitative Design and Analysis

Copyright © 2012, Elsevier Inc. All rights reserved.

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

Processor Performance



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. start 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

- 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
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

What is "Computer Architecture"

Computer Architecture =

Von Neumann CPUs

Instruction Set Architecture +

Machine Organization

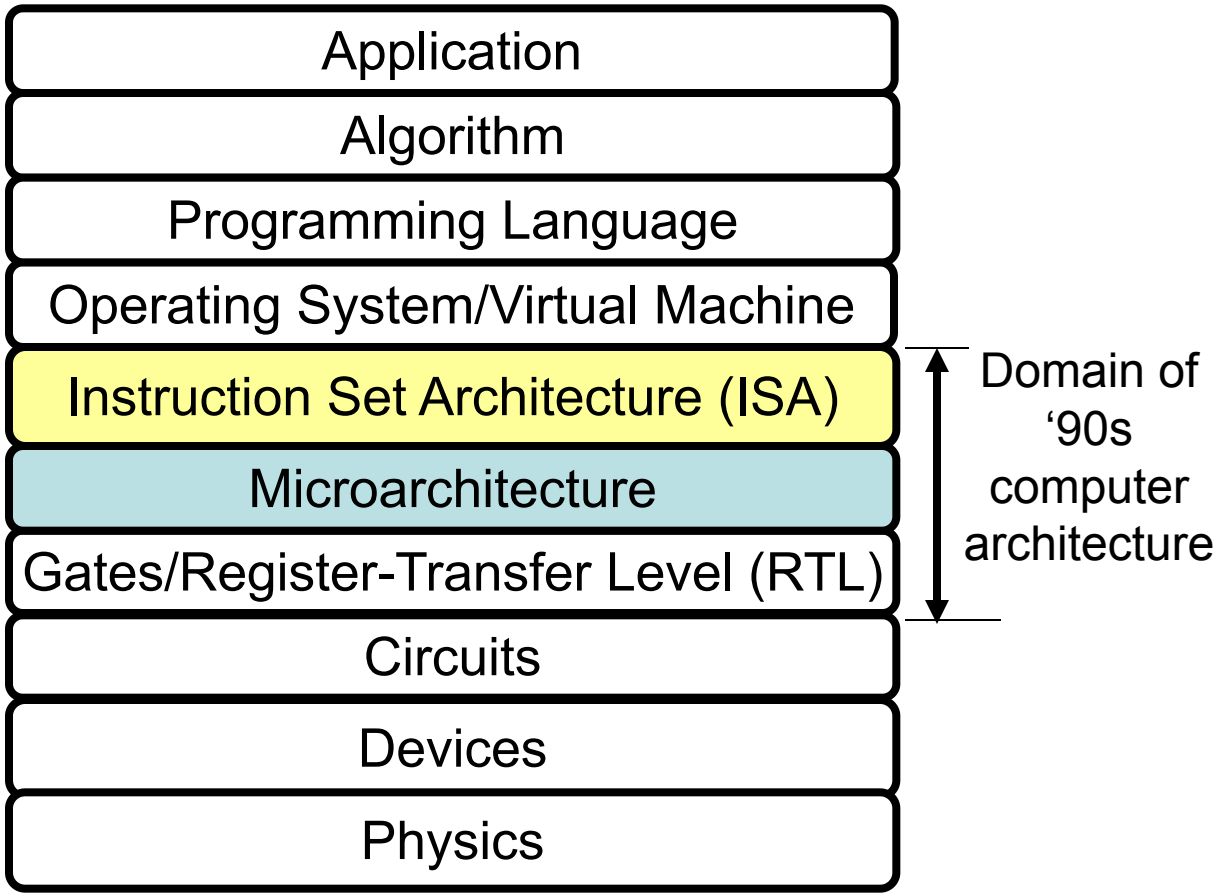
(e.g., Pipelining, Memory Hierarchy, Storage systems, etc)

and also Unconventional Organization

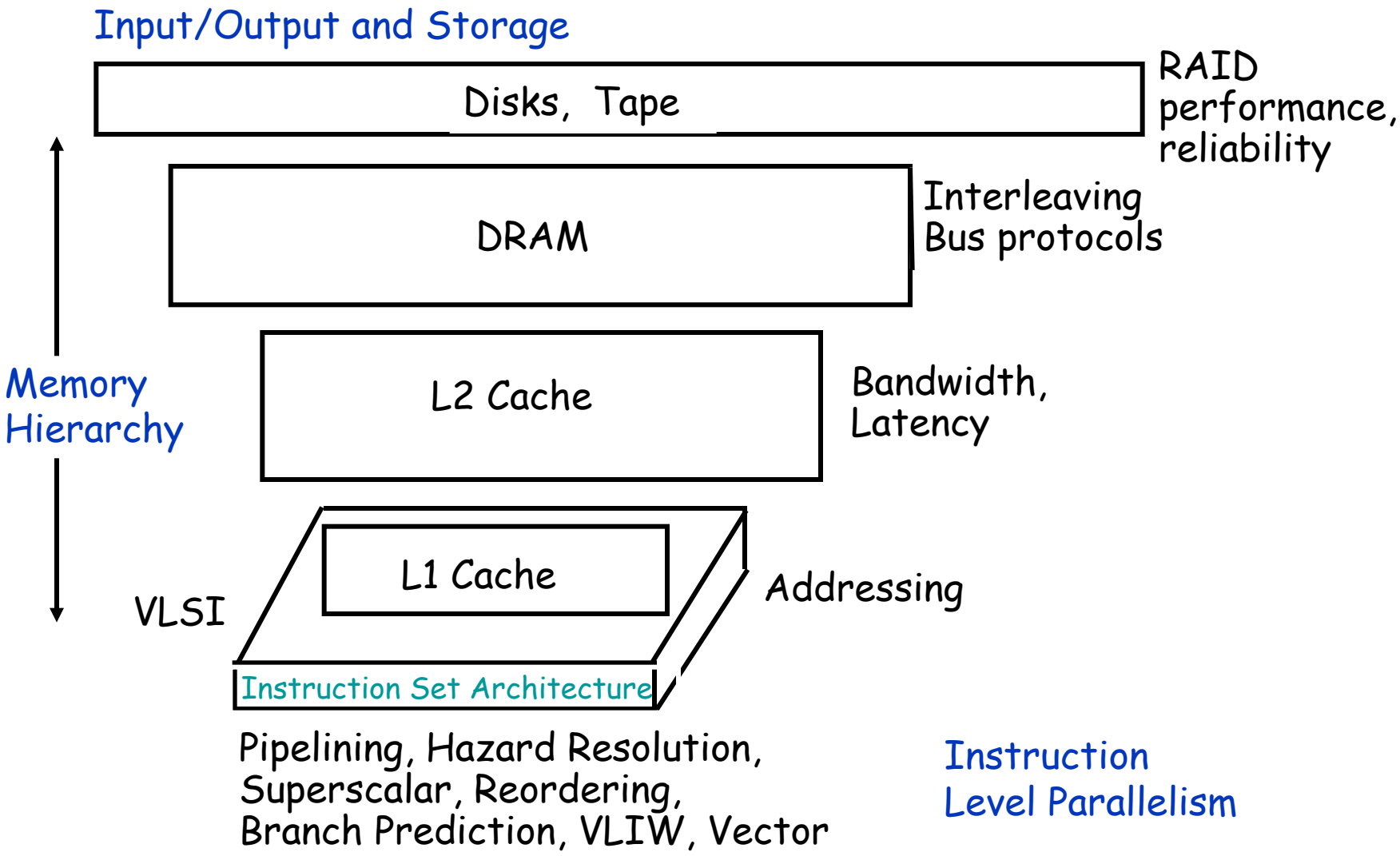
IBM 360 (minicomputer, mainframe, supercomputer)

Intel X86 vs. ARM vs. Neuromorphic vs. Nanoprocessors

Abstraction Layers in Modern Systems



Computer Architecture Topics - Processors



Compatibility Problem at IBM

- By early 1960’s, IBM had 4 incompatible lines of computers !
- Each system had its own:
  - Instruction set
  - I/O system and Secondary Storage:magnetic tapes, drums and disks
  - assemblers, compilers, libraries,...
  - market niche business, scientific, real time, ...

- Milestone: The first true ISA designed as portable hardware-software interface! -- IBM360 , Integrate the 4 incompatible lines.

IBM 360: A General- Purpose Register (GPR) Machine

- Processor State
  - 16 General Purpose 32 bit Registers
  - 4 Floating Point 64-bit Registers
  - A Program Status Word(PSW)
    - PC, Condition codes, Control flags
- A 32 bit machine with 24-bit addresses
  - But no instruction contains a 24-bit address!
- Data Formats
  - 8-bit bytes, 16-bit half-words, 32—bit words, 64-bit double-words

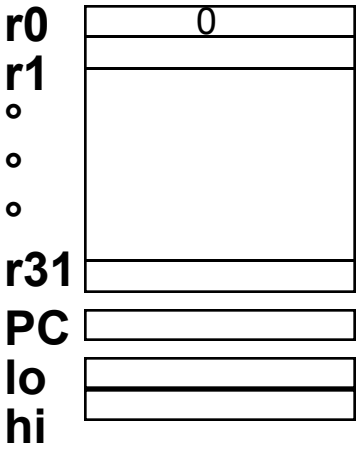
The IBM 360 is why bytes are 8- bits long today!

IBM 360: Initial Implementations

- IBM 360 instruction set architecture (ISA) completely hide the underlying technological differences between various models.

	Model 30	Model 70
Storage	8K - 64 KB	256K - 512 KB
Data path	8-bit	64-bit
Circuit Delay	30 nsec/level	5 nsec/level
Local Store	Main Store	Transistor Registers
Control Store	Read only 1 μ sec	Conventional circuits

Example: MIPS R3000



Programmable storage

- 2^32 x bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- HI, LO, PC

Data types ?

Format ?

Addressing Modes?

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

32-bit instructions on word boundary

Control

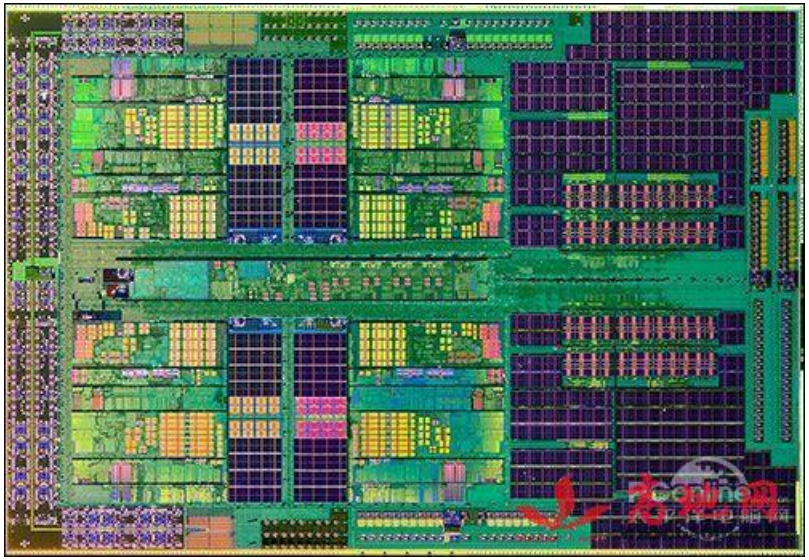
J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Same ISA
Different Microarchitecture

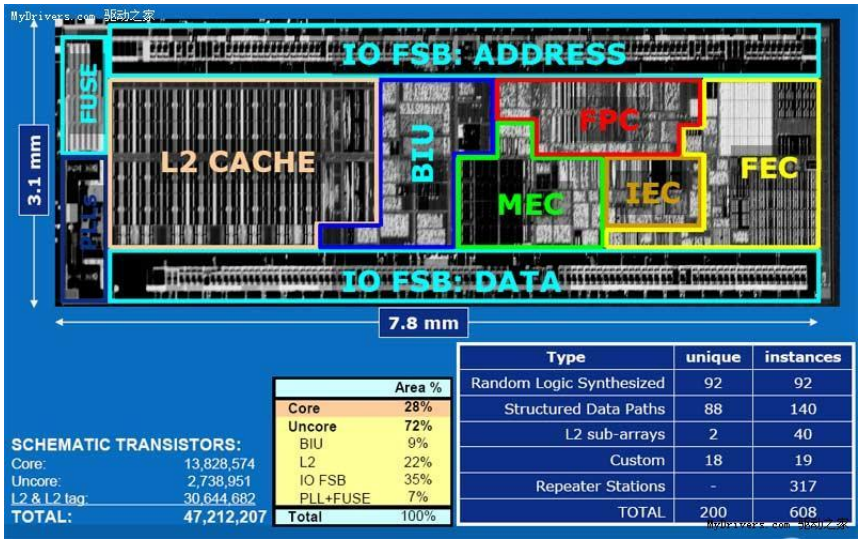
AMD Phenom II X4 945

- X86 Instruction Set
- Quad Core, 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out of order
- 2.6GHz



Intel Atom

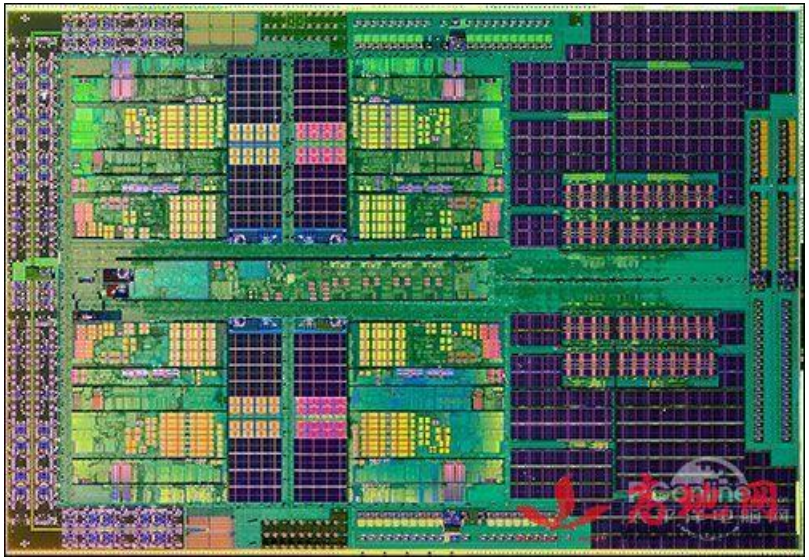
- X86 Instruction Set
- Single Core, •2W
- Decode 2 instructions/Cycle/Core
- 32KB L1 I Cache, 24KB L1 D Cache
- 512KB L2 Cache
- In order
- 1.6GHz



Different ISA
Different Microarchitecture

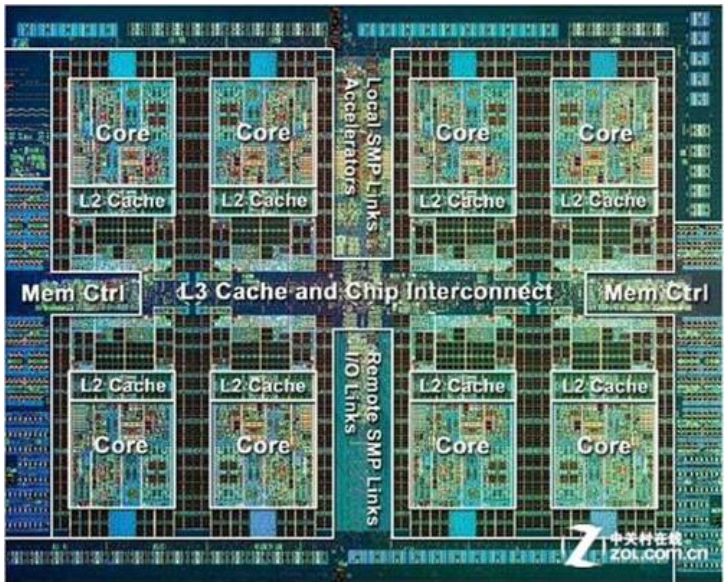
AMD Phenom II X4 945

- X86 Instruction Set
- Quad Core, 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out of order
- 2.6GHz

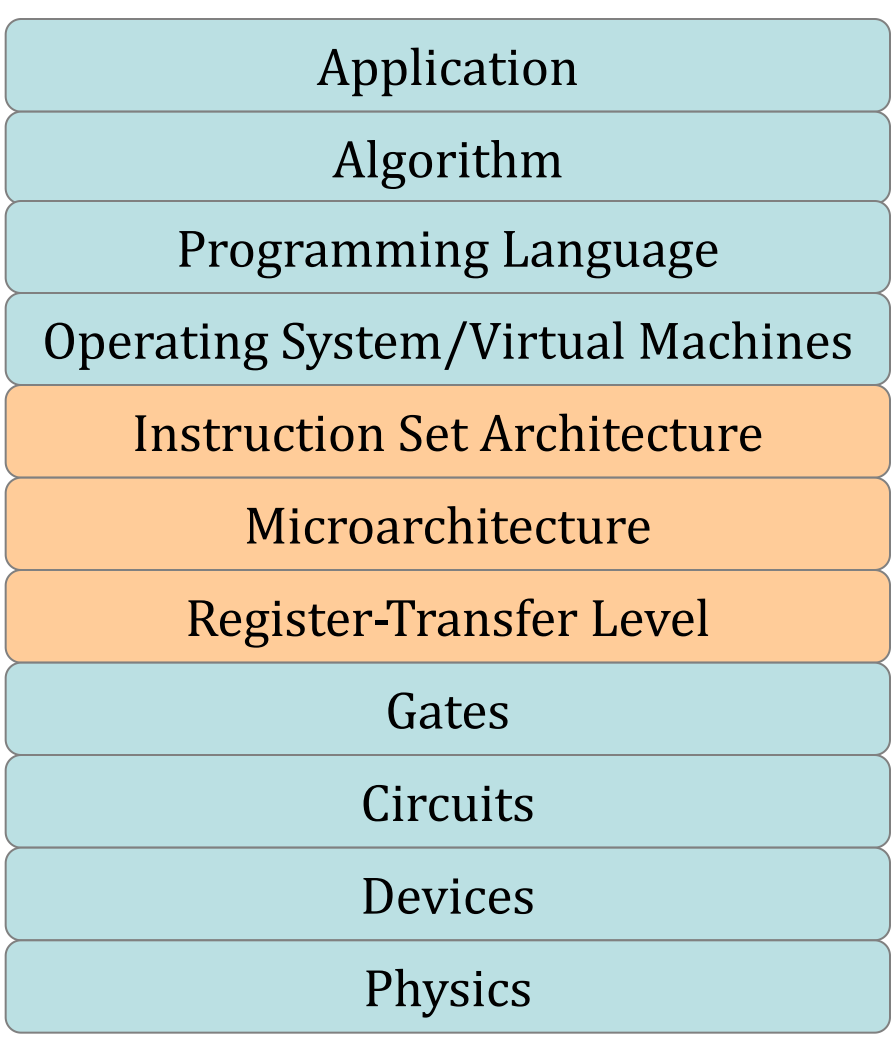


IBM POWER7

- Power Instruction Set
- Eight Core, 200W
- Decode 6 Instructions/Cycle/Core
- 32KB L1 I Cache, 32KB L1 D Cache
- 256KB L2 Cache
- Out of order
- 4.25GHz



Computing Architecture is Changing



Application Requirements:
Suggest how to improve architecture

Architecture provides feedback to guide application and technology research directions

Technology Constraints:
- Restrict what can be done efficiently
- New technologies make new arch possible

ISA vs. Computer Architecture

- Old definition of computer architecture = instruction set design(ISA)
- Our view is computer architecture >> ISA

- Computer architecture is not just about transistors, individual instructions, or particular implementations
  - E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions
- Think across all hardware/software boundaries
  - New technology ⇒ New Capabilities ⇒ New Architectures ⇒ New Tradeoffs
  - Delicate balance between backward compatibility and efficiency

Trends in Technology

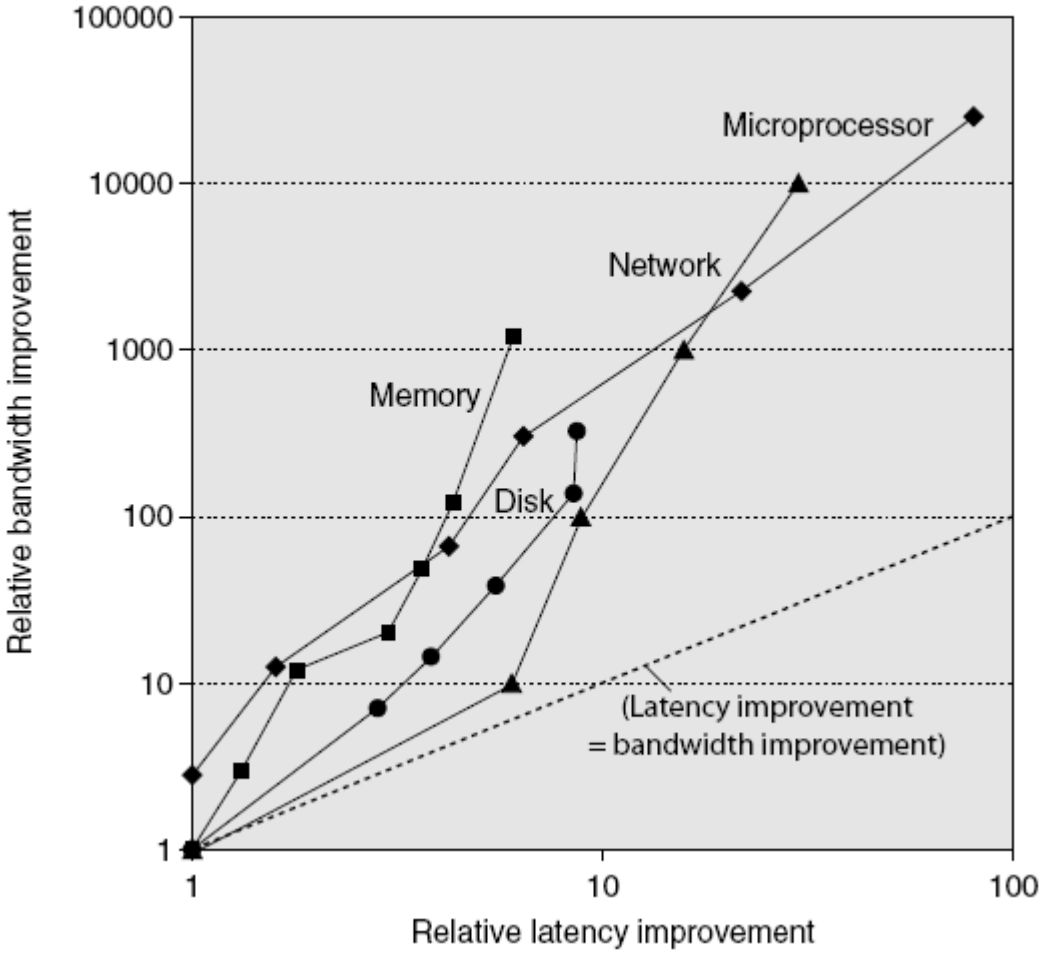
- Integrated circuit （集成电路） technology
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year （Moore ‘s Law)
  - Transistor performance scales linearly（线性增长）
    - Wire delay does not improve with feature size!
- DRAM capacity: 25-40%/year (slowing)
- Flash capacity: 50-60%/year
  - 15-20X cheaper/bit than DRAM
- Magnetic disk technology: 40%/year
  - 15-25X cheaper/bit than Flash
  - 300-500X cheaper/bit than DRAM

Bandwidth and Latency

- Bandwidth（带宽） or throughput（吞吐率）
  - Total work done in a given time
  - 10,000-25,000X improvement for processors
  - 300-1200X improvement for memory and disks

- Latency （延迟）or response time（响应时间）
  - Time between start and completion of an event
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Power and Energy

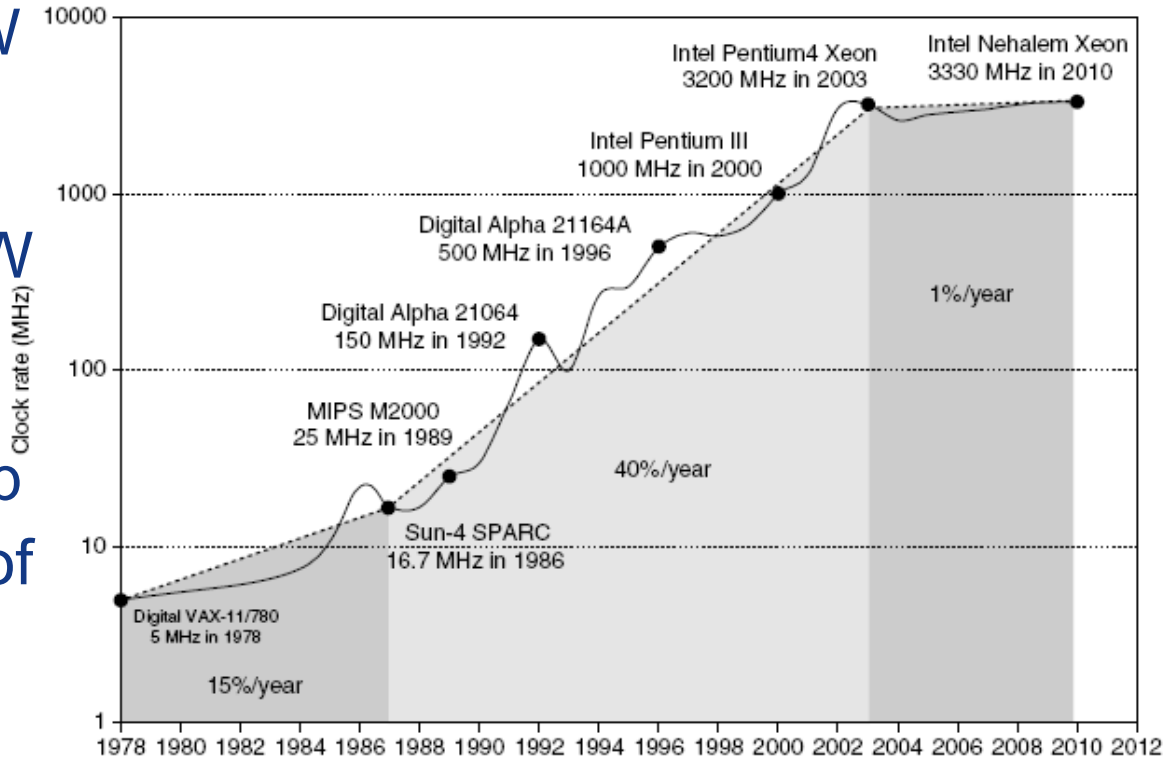
- Problem: Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

Dynamic Energy and Power

- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0
  - ½ x Capacitive load x Voltage^2
- Dynamic power
  - ½ x Capacitive load x Voltage^2 x Frequency switched
- Reducing clock rate reduces power, not energy

Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air



Reducing Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling
  - Low power state for DRAM, disks
  - Overclocking, turning off cores

Static Power

- Static power consumption
  - $\text{Current}_{\text{static}}$ x Voltage
  - Scales with number of transistors
  - To reduce: power gating

Changes in Technology lead to Changes in Architecture

1970s

- Multi-Chip CPUs
- Memory very expensive
- Complex instruction sets(good density)
- Microcoded control

1980s

- 5k- 500k transistiors
- Single-chip, pipelined CPU
- simple hardwire control
- simple instruction sets
- Small on-chip caches

1990s

- 1M-64M transistiors
- Complex control to exploit instruction level parallelism
- Deep pipeline
- Multi-level caches

2000s

- 100M- 5B transistiors
- Slow wires, power consumption, design complexity, memory latency..
- Multi-core, heterogeneous systems
- Support for parallelism

Example:

SIMD Architecture

- Vector architectures (cray-1 80's)
- SIMD extensions
  - For x86 processors (MMX_{1996}->SSE_{1999}->$SSE4_{2007}$->AVX_{2010})

    Expect two additional cores per chip per year

    SIMD width to double every four years

    Potential speedup from SIMD to be twice that from MIMD!

- Graphics Processor Units (GPUs)

```
for (i=0; i<=999; i=i+1)
        x[i] = x[i] + y[i];
```

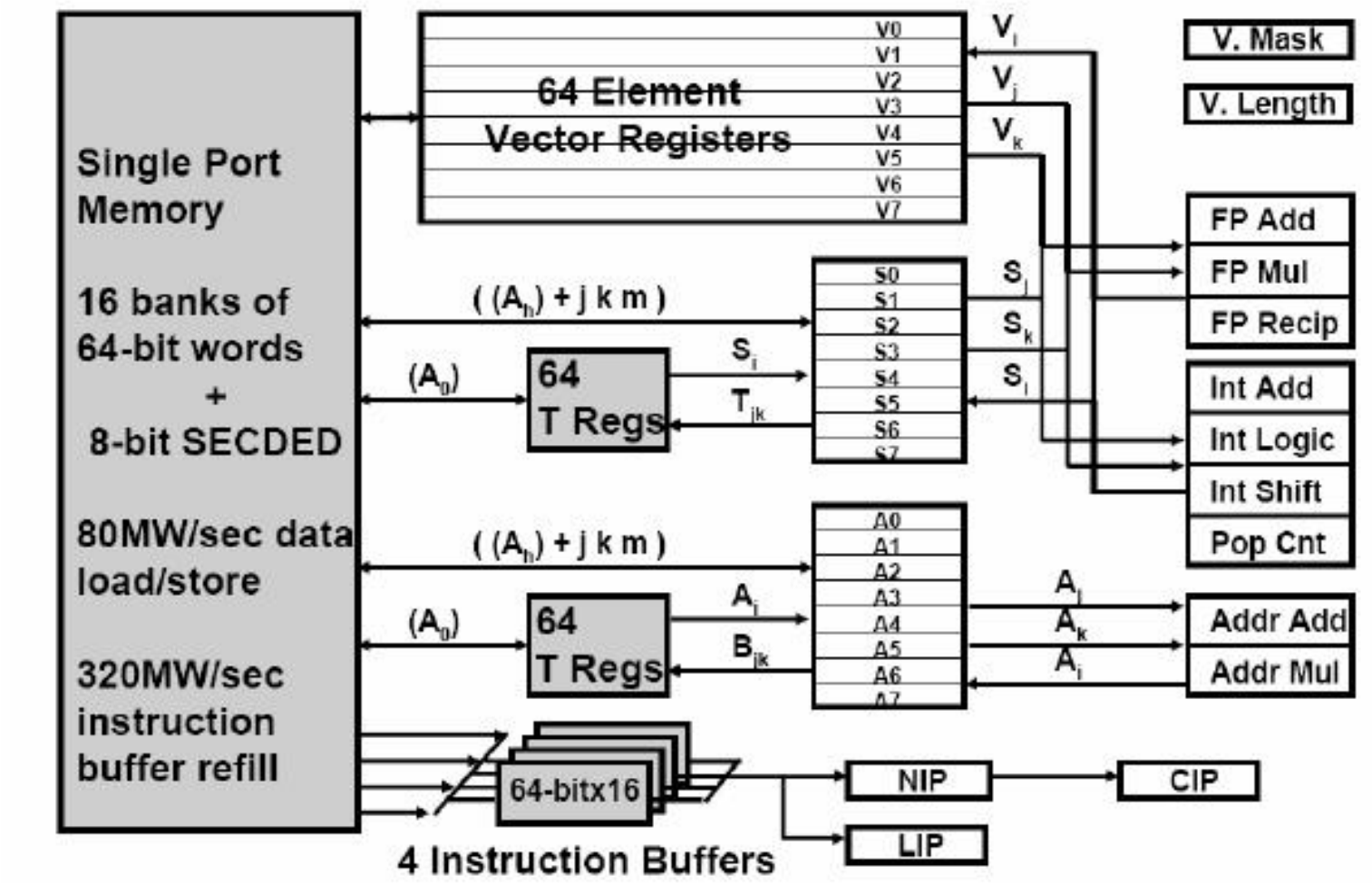
Vector Code

```
# C code
for (i=0; i<64; i++)
  C[i] = A[i] * B[i];
```

```
# Scalar Assembly Code
    LI R4, 64
loop:
    L.D F0, 0(R1)
    L.D F2, 0(R2)
    MUL.D F4, F2, F0
    S.D F4, 0(R3)
    DADDIU R1, 8
    DADDIU R2, 8
    DADDIU R3, 8
    DSUBIU R4, 1
    BNEZ R4, loop
```

```
# Vector Assembly Code
    LI VLR, 64
    LV V1, R1
    LV V2, R2
    MULVV.D V3, V1, V2
    SV V3, R3
```

Cray－1（1976）



内存体读写：50ns； 处理器周期：12.5ns（80MHz）

SIMD Extensions

- Implementations:
  - Intel MMX (1996)

    Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)

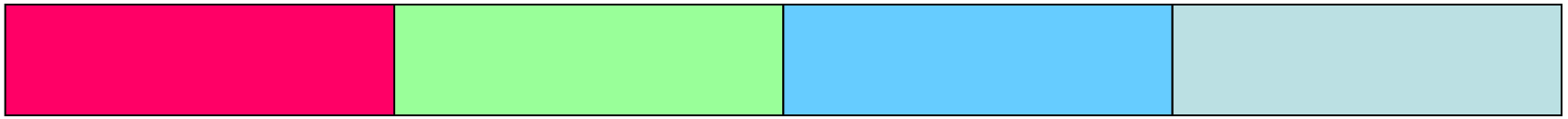
    Eight 16-bit integer ops

    Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector Extensions (2010)

    Four 64-bit integer/fp ops

Packed data

- Packed byte
  - 8bytes
- Packed short word
  - 4*16bits
- Packed word
  - 2*32bits
- Packed double word
  - 1*64bits

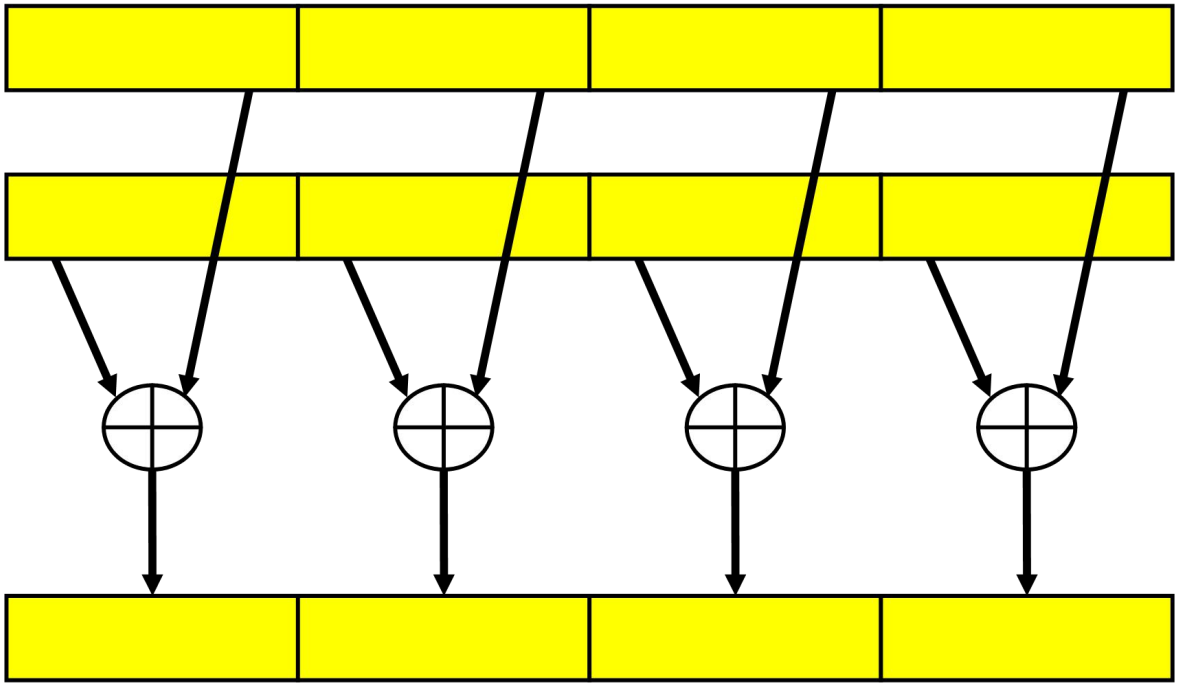


Graphic data operations

- Video decoding 视频解码
  - 加减乘除 *arithmetic* operations
  - 比较Compare
  - 对准align
  - 扩展expand
  - 包装pack
  - 合并merge
- DSP (digital signal processor)
  - 乘加(multiply and add)

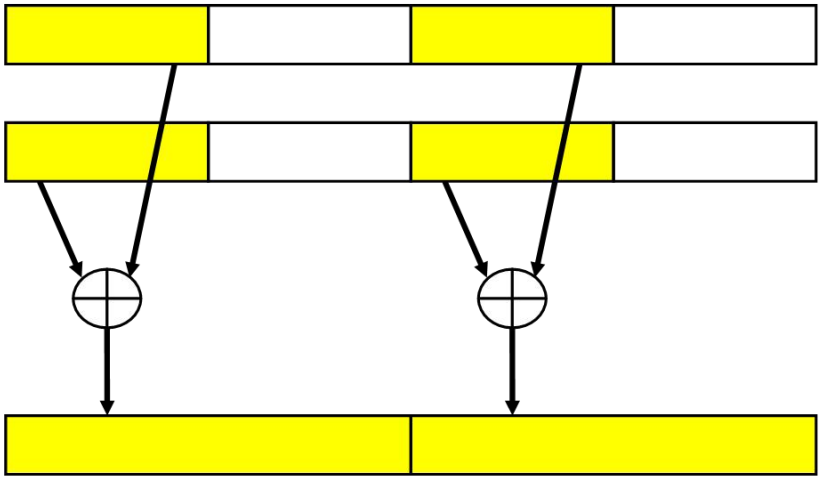
Graphic data operations

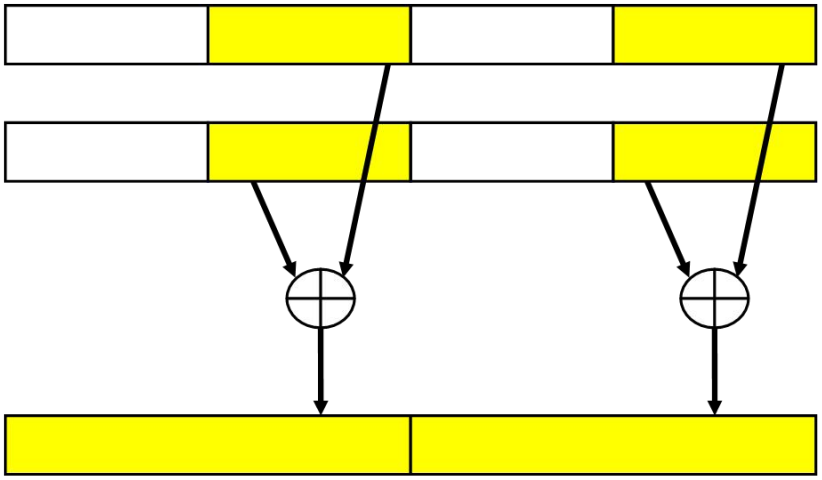
Packed word addition



Graphic data operations

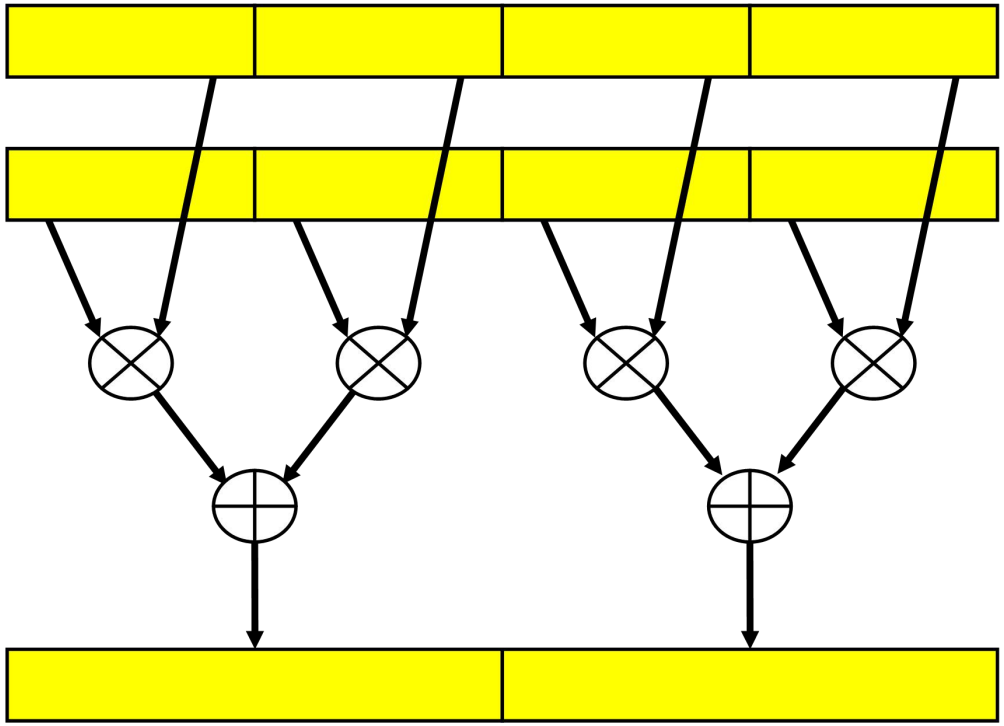
- Packed word multiplication





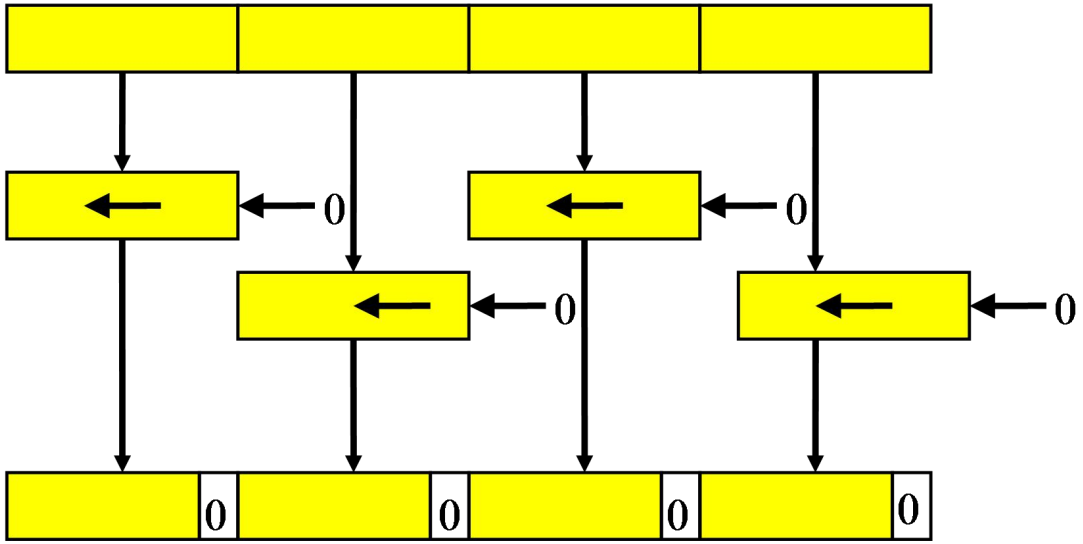
Graphic data operations

- Packed word multiplication and addition

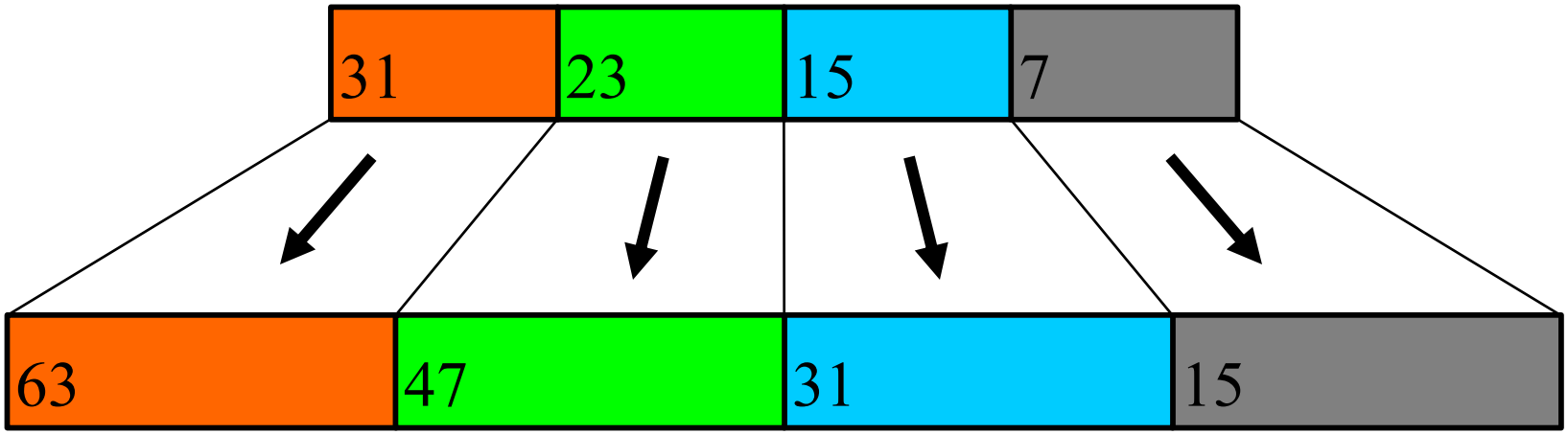


Graphic data operations

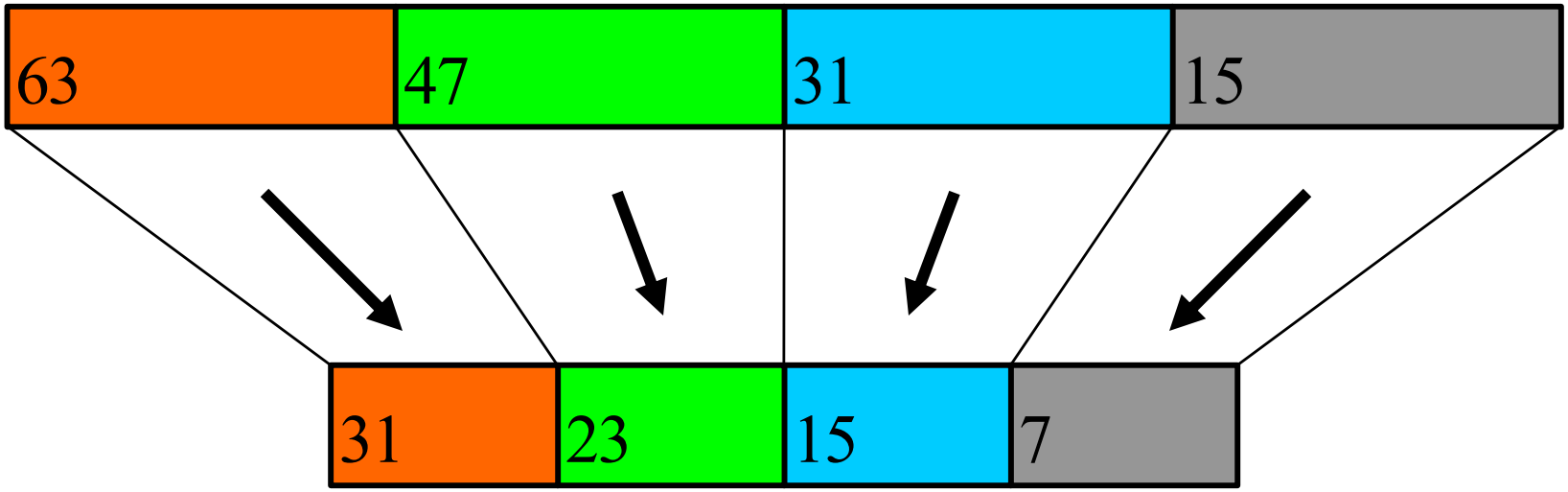
- shift



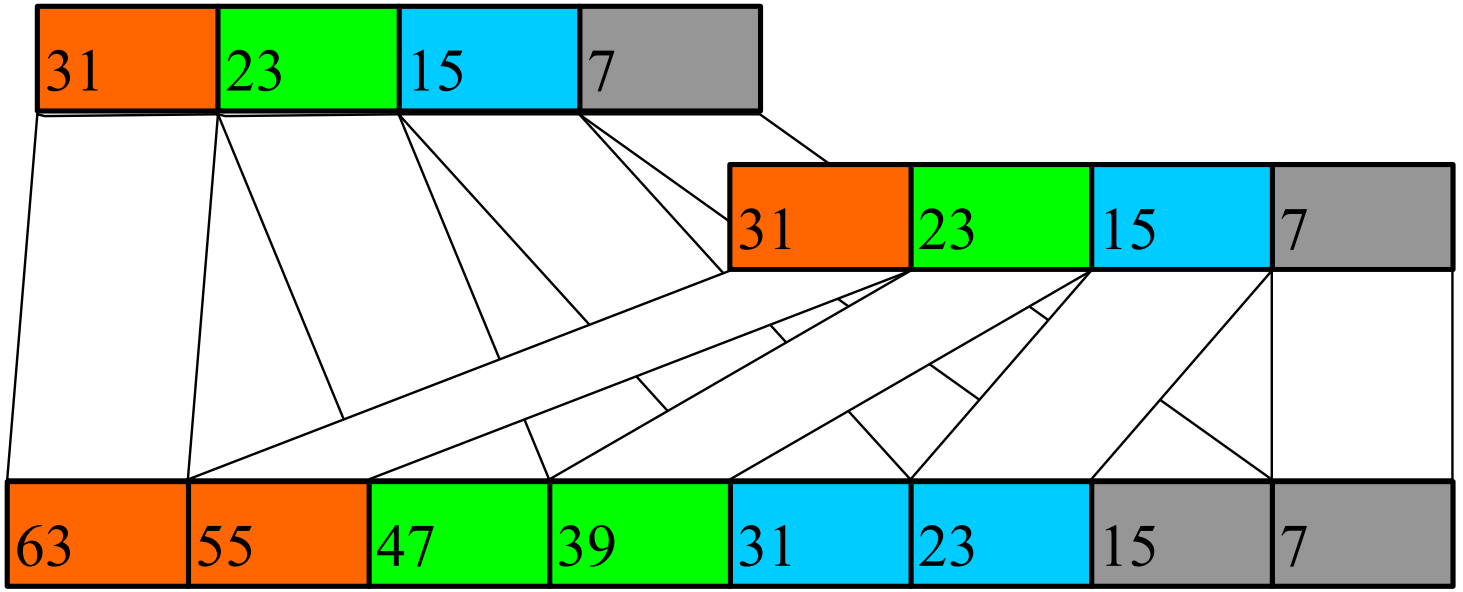
扩展操作expand



包装操作pack



合并操作merge



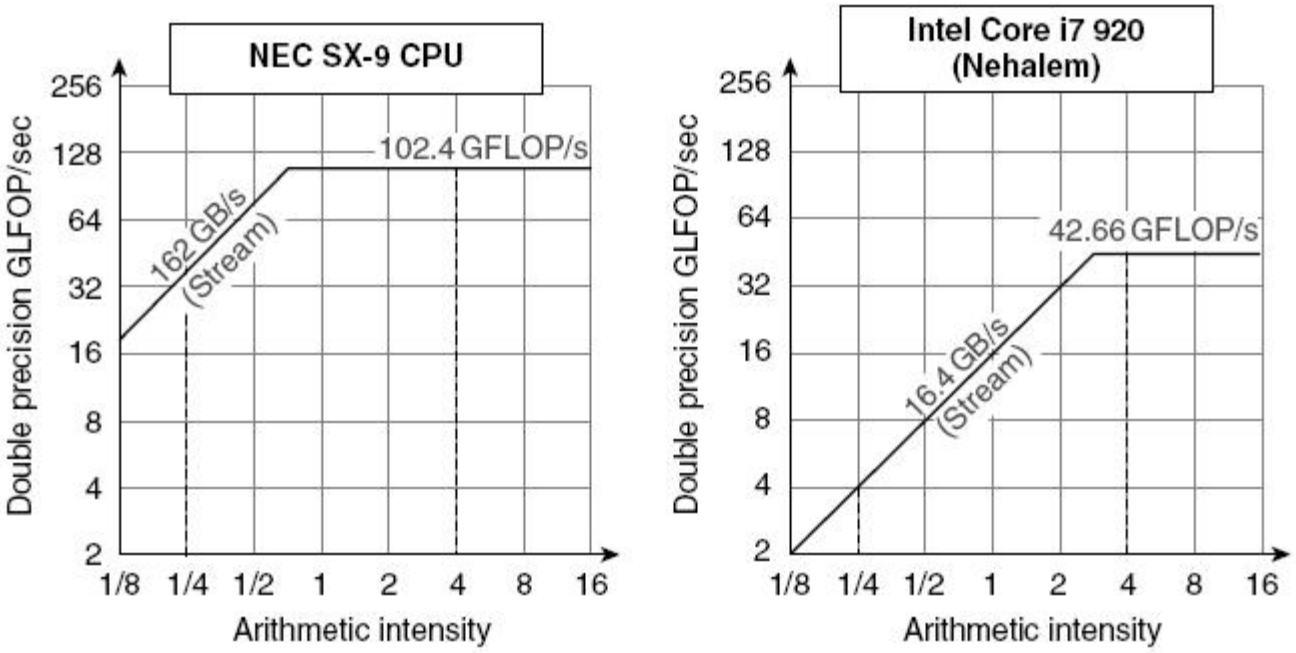
Example SIMD Code

- Example DXPY:

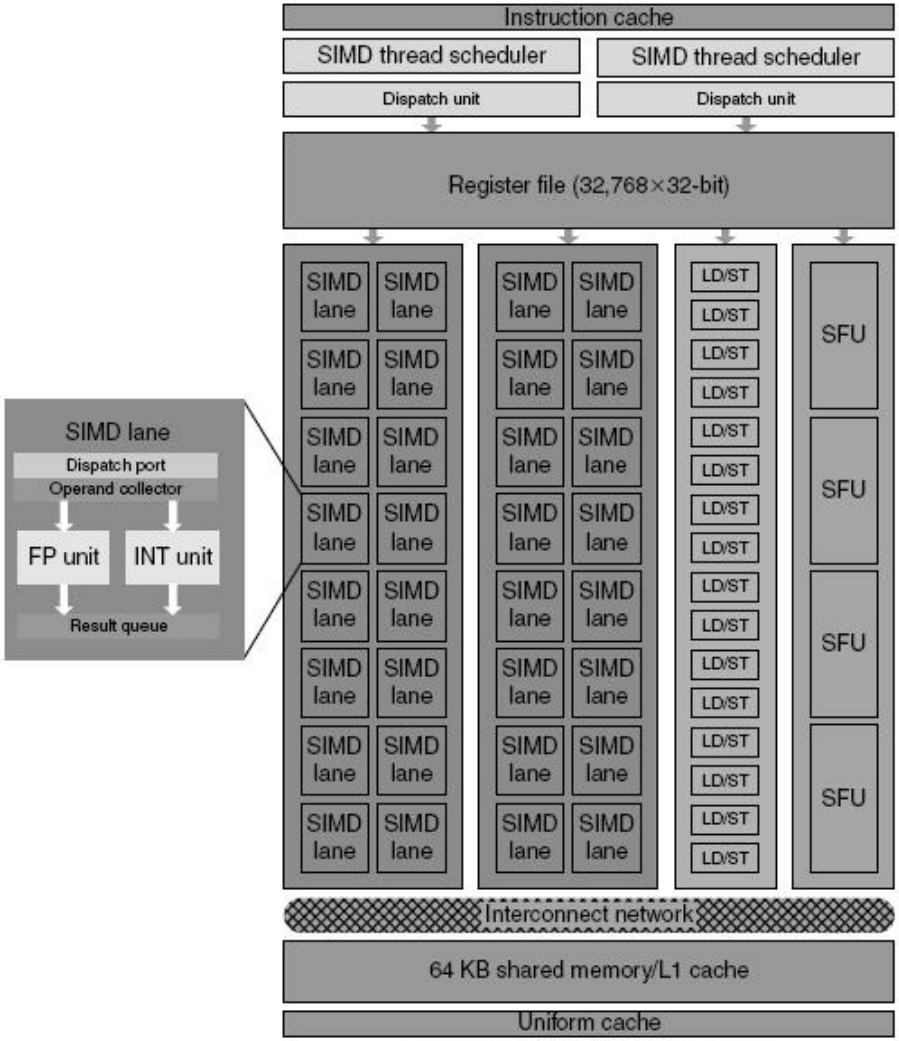
```
        L.D       F0,a          ;load scalar a
        MOV       F1, F0        ;copy a into F1 for SIMD MUL
        MOV       F2, F0        ;copy a into F2 for SIMD MUL
        MOV       F3, F0        ;copy a into F3 for SIMD MUL
        DADDIU    R4,Rx,#512    ;last address to load
Loop:   L.4D      F4,0[Rx]      ;load X[i], X[i+1], X[i+2], X[i+3]
        MUL.4D    F4,F4,F0      ;a×X[i],a×X[i+1],a×X[i+2],a×X[i+3]
        L.4D      F8,0[Ry]      ;load Y[i], Y[i+1], Y[i+2], Y[i+3]
        ADD.4D    F8,F8,F4      ;a×X[i]+Y[i], ..., a×X[i+3]+Y[i+3]
        S.4D      0[Ry],F8      ;store into Y[i], Y[i+1], Y[i+2], Y[i+3]
        DADDIU    Rx,Rx,#32     ;increment index to X
        DADDIU    Ry,Ry,#32     ;increment index to Y
        DSUBU     R20,R4,Rx     ;compute bound
        BNEZ      R20,Loop      ;check if done
```

Performance

- Attainable GFLOPs/sec Min = (Peak Memory BW × Arithmetic Intensity, Peak Floating Point Perf.)



Fermi Multithreaded SIMD Proc.



Fermi streaming multiprocessor (SM)

How to measure performance?

- Time to run the task
  - Execution time, response time, latency
  - Performance may be defined as 1 / Ex_Time
- Tasks per day, hour, minute, ...
  - Throughput, bandwidth
- Individual application vs. System management

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)

Speedup

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

"Y is *n* times faster than X" means

$$n = \text{speedup} = \frac{\text{Execution\_time(old / brand x)}}{\text{Execution\_time(new / brand y)}} = 1 + w/100$$

Speedup must be greater than 1; speedup of w%

Tx/Ty = 3/2 = 1.5 but not Ty/Tx = 2/3 = 0.67

Evaluating Metrics

MIPS and MFLOPS

- MIPS (Million Instructions Per Second)
  - Can we compare two different CPUs using MIPS?
- MFLOPS (Million Floating-point operations Per Sec.)
  - Application dependent (e.g., compiler)
- How do we compare two computer systems?
- Use benchmarks
  - Standard representative programs
- Benchmarks: e.g., SPEC CPU 2000: 26 applications (with inputs)
  - SPECint: Twelve integer, e.g., gcc, gzip, perl
  - SPECfp: Fourteen floating-point intensive, e.g., equake

SPEC CPU Benchmark

SPECint2000

Benchmark	Language	Category
164.gzip	C	Compression
175.vpr	C	FPGA Circuit Place& Route
176.gcc	C	C Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Prog Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

www.spec.org

SPECfp2000

Benchmark	Language	Category
168.wupwise	Fortran77	Quantum Chromodynamics
171.swim	Fortran77	Shallow Water Modeling
172.mgrid	Fortran77	Multi-grid Solver
173.applu	Fortran77	Partial Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran90	Fluid Dynamics
179.art	C	Image Recognition /Neural Nets
183.equake	C	Seismic Wave Propagation
187.facerec	Fortran 90	Face Recognition
188.ammp	C	Computational Chemistry
189.lucas	Fortran90	Primality Testing
191.fma3d	Fortran90	Finite-element Crash - Nuclear Physics
200.sixtrack	Fortran77	Accelerator Design
301.apsi	Fortran77	Meteorology: Pollutant Distribution

Other Benchmarks

Source: L. John, "Performance Evaluation: Techniques, Tools and Benchmarks," The Computer Engineering Handbook, CRC Press, 2001.

lca.ece.utexas.edu/pubs/john_perfeval.pdf

Workload Category	Example Benchmark Suite
CPU Benchmarks - Uniprocessor	SPEC CPU 2000 Java Grande Forum Benchmarks SciMark, ASCI
CPU - Parallel Processor	SPLASH, NASPAR
Multimedia	MediaBench
Embedded	EEMBC benchmarks
Digital Signal Processing	BDTI benchmarks
Java - Client side	SPECjvm98, CaffeineMark
Java - Server side	SPECjBB2000, VolanoMark
Java - Scientific	Java Grande Forum Benchmarks SciMark
Transaction Processing On-Line Transaction Processing	TPC-C, TPC-W
Transaction Processing Decision Support Systems	TPC-H, TP-R
Web Server	SPEC web99, TPC-W, VolanoMark
Electronic commerce	TPC-W, SPECjBB2000
Mail-server	SPECmail2000
Network File System	SPEC SFS 2.0
Personal Computer	SYSMARK, WinBench, DMarkMAX99

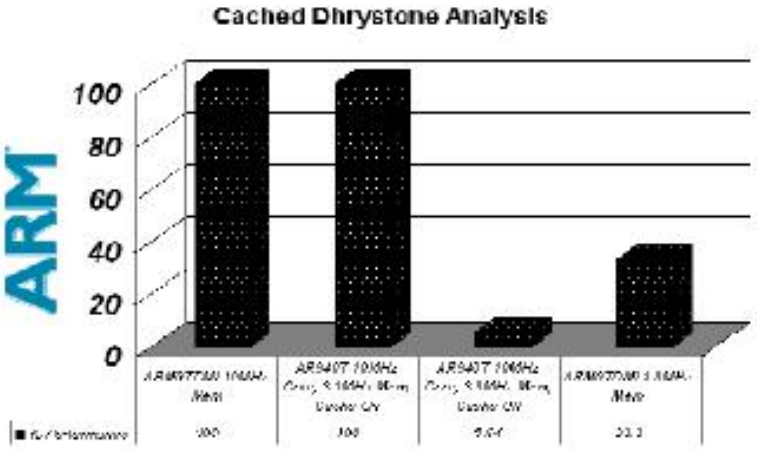
Synthetic Benchmarks

Whetstone Benchmark (C/C++)

Machine	Freq (GHz)	Mflop (1)	MWIPS
Celeron	1.3	464	1298
Atom mobile	1.6	374	833
Athlon XP	2.1	621	2133
Athlon 64	2.2	655	2313
Core 2 Duo	2.4	852	2403
Pentium 4E	3.0	607	1505
Core i7 CP	2.8-3.07	1065	3206
Phenom II	3.0	900	3257
Core i7	3.7-3.9	1237	3982

Dhrystone Benchmark

Core	DMIPS /MHz.	Freq. (MHz)	DMIPS
4Kc™	1.3	300	390
4KEc™	1.35	300	405
5Kc™	1.4	350	490
5Kf™	1.4	320	448
20Kc™	1.7	600	1020



Speedup and Amdahl's Law

- Speedup = $CPUtime_{old}$ / $CPUtime_{new}$
- Given an optimization x that accelerates fraction f_x of program by a factor of S_x, how much is the overall speedup?

$$Speedup = \frac{CPUTime_{old}}{CPUTime_{new}} = \frac{CPUTime_{old}}{CPUTime_{old}[(1-f_x)+\frac{f_x}{S_x}]} = \frac{1}{(1-f_x)+\frac{f_x}{S_x}}$$

- Lesson's from Amdhal's law
  - Make common cases fast: as $f_x \rightarrow 1$, speedup$\rightarrow S_x$
  - But don't overoptimize common case: as $S_x \rightarrow \infty$, speedup$\rightarrow 1 / (1-f_x)$
    - Speedup is limited by the fraction of the code that can be accelerated
    - Uncommon case will eventually become the common one
  - Amdhal's law applies on other metrics too: cost, power consumption, …

Example 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?

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

Amdahl's law can serve as a guide to how much an enhancement will improve performance and how to distributed resource to improve cost-performance.

Amdahl's law is useful for comparing the overall system performance of two alternatives, but it can also be applied to compare two processor design alternatives.

Amdahl's Law

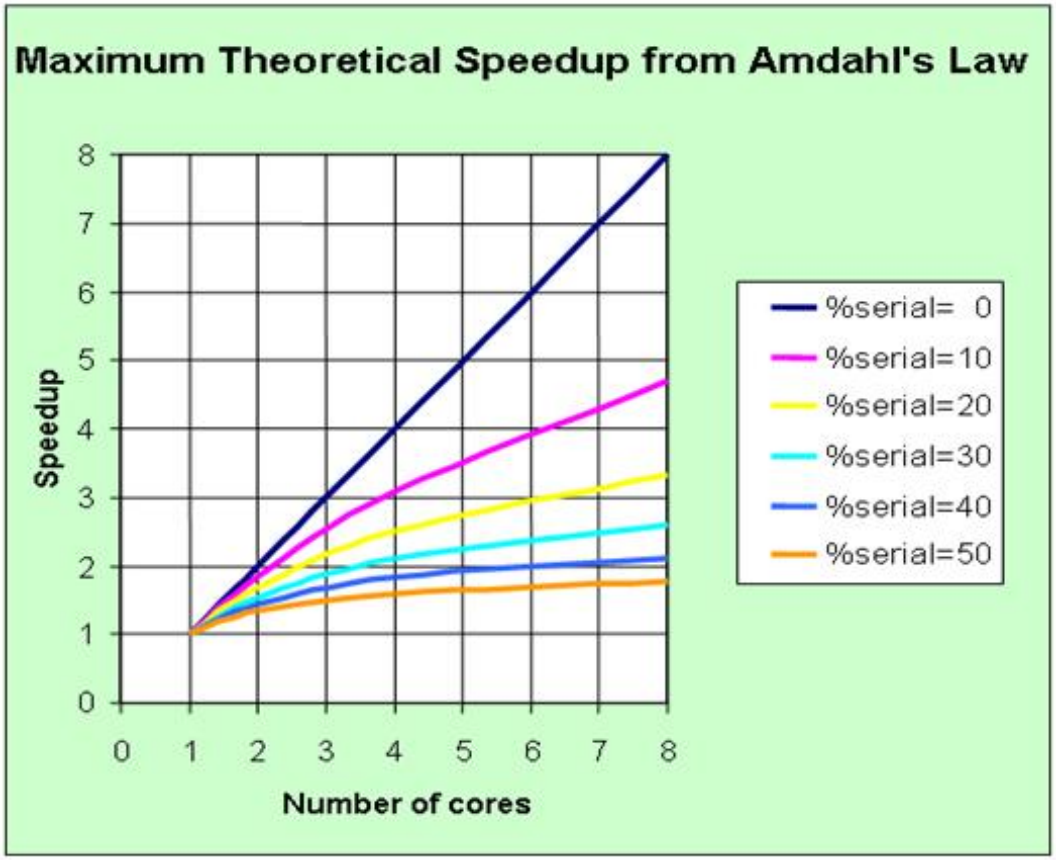
- All parallel programs contain:
  - parallel sections (we hope!)
  - serial sections (we despair!)
- Serial sections limit the parallel effectiveness
- Amdahl's Law states this formally
  - Effect of multiple processors on speed up

$$Speedup(p) = \frac{1}{s + (1-s)/p}$$

- where
  - s = serial fraction of code
  - (1-s) = parallel fraction of code
  - p = number of processors

Example:
s = 0.5, 1-s = 0.5, p = 2
Sp, max = 1 / (0.5 + 0.25) = 1.333

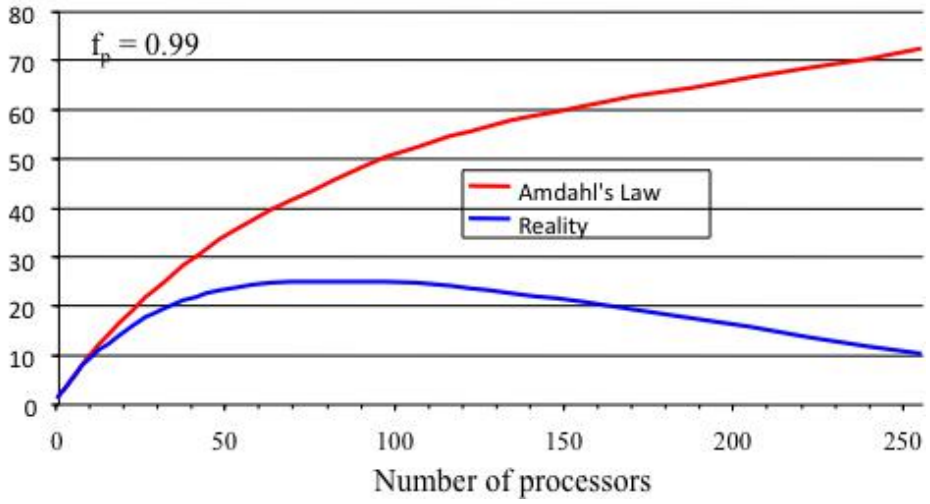
Amdahl's Law with infinitely parallel execution resources



One conclusion made by Amdahl is that "... the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude."

Practical Limits: Amdahl's Law vs. Reality

- In reality, the situation is even worse than predicted by Amdahl's Law due to:
  - Load balancing (waiting)
  - Scheduling (shared processors or memory)
  - Cost of Communications
  - I/O



Gustafson's Law

- In 1988, John Gustafson refined Amdahl's model by adjusting some of its underlying assumptions:
  - There exists workloads that are not in fixed sizes in nature: When provided with more compute power, they expand to consume the newly provided power.
  - When the problem size is increased, the parallel portion expands faster than the serial portion.

Lemma1: Problem size

- Graphics. If I give you more compute power, you will just run your frames at a higher resolution or with more details.
- Numerical analysis such as computing pi. if I give you more compute power, you will just compute more digits of pi.
- Weather Prediction. If I give you more compute power, you will just run your software longer to get even more accurate predictions.

Lemma 2

- When the problem size is increased, the parallel portion expands faster than the serial portion.
  - For example, Martix-Matrix-Multiply (MMM). The setup of MMM, ie.. initializing the matrices increases linearly with the size of the matrix. However, the actual compute is O(n^3).

Gustafson's Law

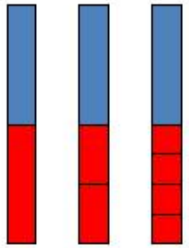
- Effect of multiple processors on run time of a problem with a fixed amount of parallel work per processor.

$$Speedup(p) = p + (1 - p)s$$

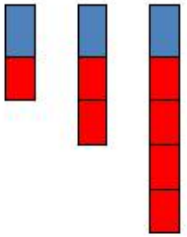
- s is the fraction of non-parallelized code where the parallel work per processor is fixed (not the same as 1-s from Amdahl's)
- p is the number of processors

Comparison of Amdahl and Gustafson

Amdahl : fixed work



Gustafson : fixed work per processor



$$Speedup(p) = \frac{1}{s + (1-s)/p}$$

$$Speedup(p) = p + (1-p)s$$

s=0.5
$Speedup_2$=1/0.5+0.5/2=1.33
$Speedup_4$= 1/0.5+0.5/4=1.6

s=0.5
$Speedup_2$=2+(1-2)*0.5=1.5
$Speedup_4$= 4+(1-4)*0.5=2.5

Scaling: Strong vs. Weak

- We want to know how quickly we can complete analysis on a particular data set by increasing the PE count
  - Amdahl's Law
  - Known as "strong scaling"

- We want to know if we can analyze more data in approximately the same amount of time by increasing the PE count
  - Gustafson's Law
  - Known as "weak scaling"

Principles of Computer Design

- Take Advantage of Parallelism
  - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units
- Principle of Locality
  - Reuse of data and instructions
- Focus on the Common Case
  - Amdahl's Law

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

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

Principles of Computer Design

The Processor Performance Equation

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

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

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

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

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

Principles of Computer Design

- Different instruction types having different CPIs

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

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

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Acknowledgements

- These slides contain material developed and copyright by:
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - David Wentzlaff (Princeton)
  - Christos Kozyrakis (Standford)