Eight Key Ideas in Computer Architecture, from Eight Decades of Innovation

Behrooz Parhami
University of California, Santa Barbara
About This Presentation

This slide show was first developed as a keynote talk for remote delivery at CSICC-2016, Computer Society of Iran Computer Conference, held in Tehran on March 8-10. The talk was presented at a special session on March 9, 11:30 AM to 12:30 PM Tehran time (12:00-1:00 AM PST). All rights reserved for the author. ©2016 Behrooz Parhami

<table>
<thead>
<tr>
<th>Edition</th>
<th>Released</th>
<th>Revised</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>March 2016</td>
<td>June 2019</td>
</tr>
</tbody>
</table>
Some of the material in this talk come from, or will appear in updated versions of, my two computer architecture textbooks.
Eight Key Ideas in Computer Architecture, from Seven Decades of Innovation

Computer architecture became an established discipline when the stored-program concept was incorporated into bare-bones computers of the 1940s. Since then, the field has seen multiple minor and major innovations in each decade. I will present my pick of the most important innovation in each of the eight decades, from the 1940s to the 2010s, and show how these ideas, when connected to each other and allowed to interact and cross-fertilize, produced the phenomenal growth of computer performance, now approaching exa-op/s (billion billion operations / s), as well as to ultra-low-energy and single-chip systems. I will also offer predictions for what to expect in the 2020s and beyond.
Behrooz Parhami (PhD, UCLA 1973) is Professor of Electrical and Computer Engineering, and former Associate Dean for Academic Personnel, College of Engineering, at University of California, Santa Barbara, where he teaches and does research in the field of computer architecture: more specifically, in computer arithmetic, parallel processing, and dependable computing.

A Life Fellow of IEEE, a Fellow of IET and British Computer Society, and recipient of several other awards (including a most-cited paper award from *J. Parallel & Distributed Computing*), he has written six textbooks and more than 300 peer-reviewed technical papers. Professionally, he serves on journal editorial boards (including for 3 different *IEEE Transactions*) and conference program committees, and he is also active in technical consulting.
Background: 1820s-1930s

- Difference Engine
- Analytical Engine
- Punched Cards
- Program (Instructions)
- Data (Variable values)
Difference Engine: Fixed Program

Babbage’s Difference Engine 2

2nd-degree polynomial evaluation
Babbage used 7th-degree $f(x)$

<table>
<thead>
<tr>
<th>$D^{(2)}$</th>
<th>$D^{(1)}$</th>
<th>$f(x)$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>47</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>53</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>61</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>71</td>
<td>5</td>
</tr>
</tbody>
</table>

$f(x) = x^2 + x + 41$
Analytical Engine: Programmable

Ada Lovelace, world’s first programmer

Sample program >
Electromechanical and Plug-Programmable Computing Machines

Punched-card device

Zuse’s Z3

Turing’s Colossus

ENIAC
The Eight Key Ideas

- Stored program
- Microprogramming
- Parallel processing
- Cache memory
- Pipelining
- FPGAs
- GPUs
- Specialization

1940s: Stored program
1950s: Microprogramming
1960s: Parallel processing
1970s: Cache memory
1980s: Pipelining
1990s: FPGAs
2000s: GPUs
2010s: Specialization
1940s: Stored Program

Exactly who came up with the stored-program concept is unclear.

Legally, John Vincent Atanasoff is designated as inventor, but many others deserve to share the credit.

Babbage	Turing	Atanasoff

Eckert	Mauchly	von Neumann
First Stored-Program Computer

Manchester Small-Scale Experimental Machine
Ran a stored program on June 21, 1948
(Its successor, Manchester Mark 1, operational in April 1949)

EDSAC (Cambridge University; Wilkes et al.)
Fully operational on May 6, 1949

EDVAC (IAS, Princeton University; von Neumann et al.)
Conceived in 1945 but not delivered until August 1949

BINAC (Binary Automatic Computer, Eckert & Mauchly)
Delivered on August 22, 1949, but did not function correctly

von Neumann vs. Harvard Architecture

von Neumann architecture
(unified memory for code & data)

Programs can be modified like data
More efficient use of memory space

Harvard architecture
(separate memories for code & data)

Better protection of programs
Higher aggregate memory bandwidth
Memory optimization for access type
1950s: Microprogramming

Traditional control unit design (multicycle): Specify which control signals are to be asserted in each cycle and synthesize.

- **Cycle 1**: Start
  - State 0: Inst'Data = 0, MemRead = 1, MemWrite = 1, ALUSrcX = 0, ALUSrcY = 0, ALUFunc = '+', PCWrite = 1

- **Cycle 2**: Jump/Branch
  - State 1: lw/sw
  - State 5: Inst'Data = 1, MemWrite = 1, ALUSrcX = 1, ALUSrcY = 1, ALUFunc = '+', JumpAddr = %, PCSrc = %, PCWrite = 1

- **Cycle 3**: ALU-type
  - State 2: Inst'Data = 1, MemRead = 1, ALUSrcX = 1, ALUSrcY = 2, ALUFunc = '+', RegDst = 0, RegInSrc = 0, RegWrite = 1
  - State 6: Inst'Data = 1, MemWrite = 1, RegDst = 0, RegInSrc = 0, RegWrite = 1

- **Cycle 4**: Discussion on states and logic
  - State 3: Inst'Data = 1, MemRead = 1, ALUSrcX = 1, ALUSrcY = 2, ALUFunc = '−', JumpAddr = %, PCSrc = %, PCWrite = 1

- **Cycle 5**: Analysis of states and logic
  - State 4: Inst'Data = 1, MemWrite = 1, RegDst = 0, RegInSrc = 0, RegWrite = 1

Notes for State 5:
- % 0 for j or jal, 1 for syscall, don't care for other instr
- @ 0 for j, jal, and syscall, 1 for jr, 2 for branches
- # 1 for j, jr, jal, and syscall, ALUZero ('') for beq (bne), bit 31 of ALUout for bltz
- For jal, RegDst = 2, RegInSrc = 1, RegWrite = 1

Notes for State 7:
- ALUFunc is determined based on the op and fn fields

Gives rise to random logic
Error-prone and inflexible
Hardware bugs hard to fix after deployment
Design from scratch for each system
The Birth of Microprogramming

The control state machine resembles a program (microprogram) comprised of instructions (microinstructions) and sequencing.

Every microinstruction contains a branch field.

Maurice V. Wilkes (1913-2010)
Microprogramming Implementation

Each microinstruction controls the data path for one clock cycle
1960s: Parallel Processing

Associative (content-addressed) memories and other forms of parallelism (compute-I/O overlap, functional parallelism) had been in existence since the 1940s.

Highly parallel machine, proposed by Daniel Slotnick in 1964, later morphed into ILLIAC IV in 1968 (operational in 1975).

Michael J. Flynn devised his now-famous 4-way taxonomy (SISD, SIMD, MISD, MIMD) in 1966 and Amdahl formulated his speed-up law and rules for system balance in 1967.
The ILLIAC IV Concept: SIMD Parallelism

Common control unit fetches and decodes instructions, broadcasting the control signals to all PEs.

Each PE executes or ignores the instruction based on local, data-dependent conditions.

The interprocessor routing network is only partially shown.

June 2019
Various Forms of MIMD Parallelism

**Global shared memory**
- Memory latency
- Memory bandwidth
- Cache coherence

**Distributed shared memory or message-passing architecture**
- Scalable network performance
- Flexible and more robust
- Memory consistency model
Warehouse-Sized Data Centers

Cooling: High-efficiency water-based cooling systems—less energy-intensive than traditional chillers—circulate cold water through the containers to remove heat, eliminating the need for air-conditioned rooms.

Structure: A 24,000-square-meter facility houses 400 containers. Delivered by trucks, the containers attach to a spine infrastructure that feeds network connectivity, power, and water. The data center has no conventional raised floors.

Power: Two power substations feed a total of 300 megawatts to the data center, with 200 MW used for computing equipment and 100 MW for cooling and electrical losses. Batteries and generators provide backup power.

Container: Each 67.5-cubic-meter container houses 2500 servers, about 10 times as many as conventional data centers pack in the same space. Each container integrates computing, networking, power, and cooling systems.

Image from IEEE Spectrum, June 2009
June 2019: Top 2 computers are US-based
China has 219 machines in the top 500
The Shrinking Supercomputer

Sandia Lab's ASCI Red, 1997
150 sq. meters, 800 kw

Sony Playstation, 2006
0.08 sq. meter, < 0.2 kw

Both perform at ~2 TFLOPS
1970s: Cache Memory

First paper on “buffer” memory: Maurice Wilkes, 1965

First implementation of a general cache memory: IBM 360 Model 85 (J. S. Liptay; IBM Systems J., 1968)

Broad understanding, varied implementations, and studies of optimization and performance issues in the 1970s

Modern cache implementations

- Harvard arch for L1 cashes
- von Neumann arch higher up
- Many other caches in system besides processor cashes
Hierarchical memory provides the illusion that high speed and large size are achieved simultaneously.
Hit/Miss Rate, and Effective Cycle Time

One level of cache with hit rate $h$

$$C_{\text{eff}} = hC_{\text{fast}} + (1 - h)(C_{\text{slow}} + C_{\text{fast}}) = C_{\text{fast}} + (1 - h)C_{\text{slow}}$$

Cache is transparent to user; transfers occur automatically

Data is in the cache fraction $h$ of the time (say, hit rate of 98%)

Go to main $1 - h$ of the time (say, cache miss rate of 2%)
The Locality Principle

Temporal:
Accesses to the same address are typically clustered in time

Spatial:
When a location is accessed, nearby locations tend to be accessed also

Illustration of temporal and spatial localities
Summary of Memory Hierarchy

Cache memory: provides illusion of very high speed

Main memory: reasonable cost, but slow & small

Virtual memory: provides illusion of very large size

Locality makes the illusions work
Translation Lookaside Buffer

Virtual-to-physical address translation by a TLB and how the resulting physical address is used to access the cache memory.

Program page in virtual memory

```
...  
lw   $t0,0($s1)     
addi  $t1,$zero,0   
L:    add  $t1,$t1,1
     beq  $t1,$s2,D  
     add  $t2,$t1,$t1
     add  $t2,$t2,$t2
     add  $t2,$t2,$s1
     lw   $t3,0($t2)  
     slt  $t4,$t0,$t3
     beq  $t4,$zero,L
     addi $t0,$t3,0   
     j    L           
D:    ...           
```

All instructions on this page have the same virtual page address and thus entail the same translation.
Disk Caching and Other Applications

1. Head movement from current position to desired cylinder:
   **Seek time** (0-10s ms)

2. Disk rotation until the desired sector arrives under the head:
   **Rotational latency** (0-10s ms)

3. Disk rotation until sector has passed under the head:
   **Data transfer time** (< 1 ms)

Web caching
- Client-side caching
- Caching within the cloud
- Server-side caching
1980s: Pipelining

An important form of parallelism that is given its own name

Used from early days of digital circuits in various forms

Latch positions in a four-stage pipeline

Pipelining period

Latency

Output available
Vector Processor Implementation

From scalar registers

- Load unit A
- Load unit B
- Store unit

Vector register file

Function unit 1 pipeline

Function unit 2 pipeline

Function unit 3 pipeline

Forwarding muxes
Overlapped Load/Store and Computation

Vector processing via segmented load/store of vectors in registers in a double-buffering scheme. Solid (dashed) lines show data flow in the current (next) segment.
Simple Instruction-Execution Pipeline

- Cycle 1: Instruction cache → Register file
- Cycle 2: Register file → ALU
- Cycle 3: ALU → Data cache
- Cycle 4: Data cache → Register file
- Cycle 5: Register file → ALU
- Cycle 6: ALU → Data cache
- Cycle 7: Data cache → Register file
- Cycle 8: Register file → ALU
- Cycle 9: ALU → Data cache

Task dimension: Instr 1, Instr 2, Instr 3, Instr 4, Instr 5
Pipeline Stalls or Bubbles

Data dependency and its possible resolution via forwarding

$5 = 6 + 7$
$8 = 8 + 6$
$9 = 8 + 2$
sw $9, 0(3)$

Data forwarding
Problems Arising from Deeper Pipelines

Forwarding more complex and not always workable Interlocking/stalling mechanisms needed to prevent errors
Branching and Other Complex Pipelines

Front end:
- In-order or out-of-order

Instr. issue:
- In-order or out-of-order

Write-back:
- In-order or out-of-order

Commit:
- In-order or out-of-order

The more OoO stages, the higher the complexity
1990s: FPGAs

Programmable logic arrays were developed in the 1970s.

PLAs provided cost-effective and flexible replacements for random logic or ROM/PROM.

The related programmable array logic devices came later.

PALs were less flexible than PLAs, but more cost-effective.
Why FPGA Represents a Paradigm Shift

Modern FPGAs can implement any functionality
Initially used only for prototyping
Even a complete CPU needs a small fraction of an FPGA's resources
FPGAs come with multipliers and IP cores (CPUs/SPs)
FPGAs Are Everywhere

Applications are found in virtually all industry segments:

- Aerospace and defense
- Medical electronics
- Automotive control
- Software-defined radio
- Encoding and decoding
Example: Bit-Serial 2nd-Order Digital Filter

LUTs, registers, and an adder are all we need for linear expression evaluation: 
\[ y^{(i)} = ax^{(i)} + bx^{(i-1)} + cx^{(i-2)} + dy^{(i-1)} + ey^{(i-2)} \]
2000s: GPUs

Simple graphics and signal processing units were used since the 1970s.

In the early 2000s, the two major players, ATI and Nvidia, produced powerful chips to improve the speed of shading.

In the late 2000s, GPGPUs (extended stream processors) emerged and were used in lieu of, or in conjunction with, CPUs in high-performance supercomputers.

GPUs are faster and more power-efficient than CPUs.

GPUs use a mixture of parallel processing and functional specialization to achieve super-high performance.
CPU vs. GPU Organization

Small number of powerful cores versus
Very large number of simple stream processors

Demo (analogy for MPP): https://www.youtube.com/watch?v=fKK933KK6Gg
CPU vs. GPU Performance

Peak performance (GFLOPS) and peak data rate (GB/s)
General-Purpose Computing on GPUs

Suitable for numerically intensive matrix computations

First application to run faster on a GPU was LU factorization

Users can ignore GPU features and focus on problem solving
- Nvidia CUDA Programming System
- Matlab Parallel Computing Toolbox
- C++ Accelerated Massive Parallelism

Many vendors now give users direct access to GPU features

Example system (Titan): Cray XK7 at DOE’s Oak Ridge Nat’l Lab used more than ¼ M Nvidia K20x cores to accelerate computations (energy-efficient: 2+ gigaflops/W)
2010s: Specialization

This decade has not ended yet and its new ideas have a short track record; hence what I say is subject to revision.

Processors targeted for mobile applications (ARM) and emergence of many different specialized chips.

Logic-in-memory designs (getting over the memory wall).

Tensor processing units for speeding up specific functions such as those needed for neural-network computations.

Cloud allows the utilization of most-appropriate resources.

Computer architecture came of age: David Patterson and John Hennessey honored with ACM Turing Award in 2017.
Specialized Processors and Chips

Examples of specialized chips:
12 Taptic engine
20 Power management, Apple
21 Apps processor, Apple
22 Battery charger, TI
23 Audio codec, Apple
24 Power management, Apple
26 LTE transceiver, Qualcomm
27 Wi-Fi/Bluetooth, Apple
28 LTE modem, Qualcomm
30 NFC controller, NXP Semi

Same story in automotive and other systems

Apple iPhone X teardown
(E&T magazine, Jan. 2018)
Logic-in-Memory Architectures

Old idea (aka processing-in-memory) from the 1970s, now economically feasible at very-large scale

von Neumann bottleneck

Data movement, comparison, and very simple processing, done in parallel, taking advantage of high internal memory bandwidth

Leveraging 3D stacked DRAM to do much of the required processing without moving data out of the memory (Ghose et al., 2018)
Tensor Processing Unit

Google’s TPU has a systolic matrix multiply unit, a unified buffer (24-MB register file), and hardwired activation unit.

The heart of TPU:
Systolic array

Multiplying an input matrix by a weight matrix
The Eight Key Ideas

- Stored program
- Parallel processing
- Cache memory
- Microprogramming
- Pipelining
- FPGAs
- GPUs
- Specialization

Design advances
Performance boosts
## Innovations for Improved Performance

*(Parhami: *Computer Architecture*, 2005)*

Computer performance grew by a factor of about 10000 between 1980 and 2000. 100 due to faster technology and 100 due to better architecture.

### Available computing power ca. 2000:
- GFLOPS on desktop
- TFLOPS in supercomputer center
- PFLOPS on drawing board

### Architectural method

<table>
<thead>
<tr>
<th>Architectural method</th>
<th>Improvement factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Pipelining (and superpipelining)</td>
<td>3-8 √</td>
</tr>
<tr>
<td>2. Cache memory, 2-3 levels</td>
<td>2-5 √</td>
</tr>
<tr>
<td>3. RISC and related ideas</td>
<td>2-3 √</td>
</tr>
<tr>
<td>4. Multiple instruction issue (superscalar)</td>
<td>2-3 √</td>
</tr>
<tr>
<td>5. ISA extensions (e.g., for multimedia)</td>
<td>1-3 √</td>
</tr>
<tr>
<td>6. Multithreading (super-, hyper-)</td>
<td>2-5 ?</td>
</tr>
<tr>
<td>7. Speculation and value prediction</td>
<td>2-3 ?</td>
</tr>
<tr>
<td>8. Hardware acceleration [e.g., GPU]</td>
<td>2-10 ?</td>
</tr>
<tr>
<td>9. Vector and array processing</td>
<td>2-10 ?</td>
</tr>
<tr>
<td>10. Parallel/distributed computing</td>
<td>2-1000s ?</td>
</tr>
</tbody>
</table>

### Established methods
- Previously discussed
- Covered in Part VII

### Newer methods
- Available computing power ca. 2000:
  - GFLOPS on desktop
  - TFLOPS in supercomputer center
  - PFLOPS on drawing board

Computer performance grew by a factor of about 10000 between 1980 and 2000. 100 due to faster technology and 100 due to better architecture.
Shares of Technology and Architecture in Processor Performance Improvement

Overall Performance Improvement (SPECINT, relative to 386)

Gate Speed Improvement (FO4, relative to 386)

Feature Size (\(\mu\text{m}\))

~1985

--------- 1995-2000 ---------

Much of arch. improvements already achieved

~2005

~2010

Source: “CPU DB: Recording Microprocessor History,” CACM, April 2012.
Continuing Challenge in Architecture

Preserving and expressing parallelism from the application domain all the way to the hardware implementation

Source: T. Nowatzki et al., CACM, June 2019
2020s and Beyond: Looking Ahead

Design improvements
- Adaptation and self-optimization (learning)
- Security (hardware-implemented)
- Reliability via redundancy and self-repair
- Mixed analog/digital design style
- Virtualization: Mapping or provisioning
- Open-source hardware (just like software)

Performance improvements
- Revolutionary new technologies: Atomic-scale
- New computational paradigms
- Brain-inspired and biological computing
- Speculation and value prediction
- Better performance per watt (power wall)
We Need More than Sheer Performance

Environmentally responsible design
Reusable designs, parts, and material

Power efficiency and proportionality
Starting publication in 2016: IEEE Transactions on Sustainable Computing
Questions or Comments?

parhami@ece.ucsb.edu
http://www.ece.ucsb.edu/~parhami/
Trends in Processor Chip Density, Performance, Clock Speed, Power, and Number of Cores

NRC Report (2011): The Future of Computing Performance: Game Over or Next Level?
Peak Performance of Supercomputers

The Quest for Higher Performance

Top Three Supercomputers in November 2012 (http://www.top500.org)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Supercomputer</th>
<th>Location</th>
<th>Architecture</th>
<th>Cores, Memory, OS</th>
<th>Interconnect</th>
<th>Performance (PFLOPS)</th>
<th>Power (MW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cray Titan</td>
<td>ORNL, Tennessee</td>
<td>XK7 architecture</td>
<td>560,640 cores, 710 TB, Cray Linux</td>
<td>Cray Gemini interconn’</td>
<td>17.6/27.1</td>
<td>8.2 MW</td>
</tr>
<tr>
<td>2</td>
<td>IBM Sequoia</td>
<td>LLNL, California</td>
<td>Blue Gene/Q arch</td>
<td>1,572,864 cores, 1573 TB, Linux</td>
<td>Custom interconnect</td>
<td>16.3/20.1</td>
<td>7.9 MW</td>
</tr>
<tr>
<td>3</td>
<td>Fujitsu K Computer</td>
<td>RIKEN AICS, Japan</td>
<td>RIKEN architecture</td>
<td>705,024 cores, 1410 TB, Linux</td>
<td>Tofu interconnect</td>
<td>10.5/11.3</td>
<td>12.7 MW</td>
</tr>
</tbody>
</table>

* max/peak performance

In the top 10, IBM also holds ranks 4-7 and 9-10. Dell and NUDT (China) hold ranks 7-8.
The Flynn/Johnson Classification

- **SISD** (Single Instruction, Single Data stream)
  - **Uniprocessors**
- **SIMD** (Single Instruction, Multiple Data stream)
  - **Array or vector processors**
- **MISD** (Multiple Instruction, Single Data stream)
  - **Rarely used**
- **MIMD** (Multiple Instruction, Multiple Data stream)
  - **Multiproc’s or multicomputers**

**Flynn’s categories**

- **Shared variables**
- **Message passing**
- **Global memory**
- **Distributed memory**

**Johnson’s expansion**

- **GMSV** (Global memory, Shared variables)
  - **Shared-memory multiprocessors**
  - **Rarely used**
- **GMMP** (Global memory, Message passing)
  - **Distributed-memory multicomputers**
- **DMSV** (Distributed memory, Shared variables)
  - **Distributed shared memory**
- **DMMP** (Distributed memory, Message passing)
  - **Distrib-memory multicomputers**
Shared-Control Systems

(a) Shared-control array processor, SIMD

(b) Multiple shared controls, MSIMD

(c) Separate controls, MIMD

From completely shared control to totally separate controls.
MIMD Architectures

Control parallelism: executing several instruction streams in parallel

GMSV: Shared global memory – symmetric multiprocessors
DMSV: Shared distributed memory – asymmetric multiprocessors
DMMP: Message passing – multicomputers
Past and Current Performance Trends

Intel 4004: The first μp (1971)

0.06 MIPS (4-bit processor)

- 8008
- 8080
- 8084

8-bit

16-bit

- 8086
- 8088
- 80186
- 80188
- 80286

Intel Pentium 4, circa 2005

10,000 MIPS (32-bit processor)

- 80386
- 80486
- Pentium, MMX
- Pentium Pro, II
- Pentium III, M
- Celeron

32-bit
Energy Consumption is Getting out of Hand
Amdahl’s Law

\[ s = \frac{1}{f + (1-f)/p} \leq \min(p, 1/f) \]

- \( f \) = fraction unaffected
- \( p \) = speedup of the rest

Graph shows speedup (\( s \)) vs. enhancement factor (\( p \)) with different values of \( f \): 0, 0.01, 0.02, 0.05, 0.1.
### Amdahl’s System Balance Rules of Thumb

The need for high-capacity, high-throughput secondary (disk) memory

<table>
<thead>
<tr>
<th>Processor speed</th>
<th>RAM size</th>
<th>Disk I/O rate</th>
<th>Number of disks</th>
<th>Disk capacity</th>
<th>Number of disks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GIPS</td>
<td>1 GB</td>
<td>100 MB/s</td>
<td>1</td>
<td>100 GB</td>
<td>1</td>
</tr>
<tr>
<td>1 TIPS</td>
<td>1 TB</td>
<td>100 GB/s</td>
<td>1000</td>
<td>100 TB</td>
<td>100</td>
</tr>
<tr>
<td>1 PIPS</td>
<td>1 PB</td>
<td>100 TB/s</td>
<td>1 Million</td>
<td>100 PB</td>
<td>100 000</td>
</tr>
<tr>
<td>1 EIPS</td>
<td>1 EB</td>
<td>100 PB/s</td>
<td>1 Billion</td>
<td>100 EB</td>
<td>100 Million</td>
</tr>
</tbody>
</table>

1 RAM byte for each IPS  
1 I/O bit per sec for each IPS  
100 disk bytes for each RAM byte
Design Space for Superscalar Pipelines

Front end: In-order or out-of-order
Instr. issue: In-order or out-of-order
Writeback: In-order or out-of-order
Commit: In-order or out-of-order

The more OoO stages, the higher the complexity

Example of complexity due to out-of-order processing:
MIPS R10000

Available instruction-level parallelism and the speedup due to multiple instruction issue in superscalar processors [John91].
Speculative Loads

Examples of software speculation in IA-64.

(a) Control speculation

(b) Data speculation
Value prediction for multiplication or division via a memo table.
Implementing Symmetric Multiprocessors

Structure of a generic bus-based symmetric multiprocessor.

Computing nodes (typically, 1-4 CPUs and caches per node)

Interleaved memory

I/O modules

Very wide, high-bandwidth bus
Interconnection Networks

(a) Direct network

(b) Indirect network

Examples of direct and indirect interconnection networks.
Direct Interconnection Networks

(a) 2D torus
(b) 4D hypercube
(c) Chordal ring
(d) Ring of rings

A sampling of common direct interconnection networks. Only routers are shown; a computing node is implicit for each router.
Graphic Processors, Network Processors, …

Simplified block diagram of Toaster2, Cisco Systems’ network processor.
Computing in the Cloud

Computational resources, both hardware and software, are provided by, and managed within, the cloud.

Users pay a fee for access.

Managing / upgrading is much more efficient in large, centralized facilities (warehouse-sized data centers or server farms).

This is a natural continuation of the outsourcing trend for special services, so that companies can focus their energies on their main business.

Image from Wikipedia