# CE/CZ 3001: Advanced Computer Architecture

**Module 7: Data and Thread Level Parallelism**

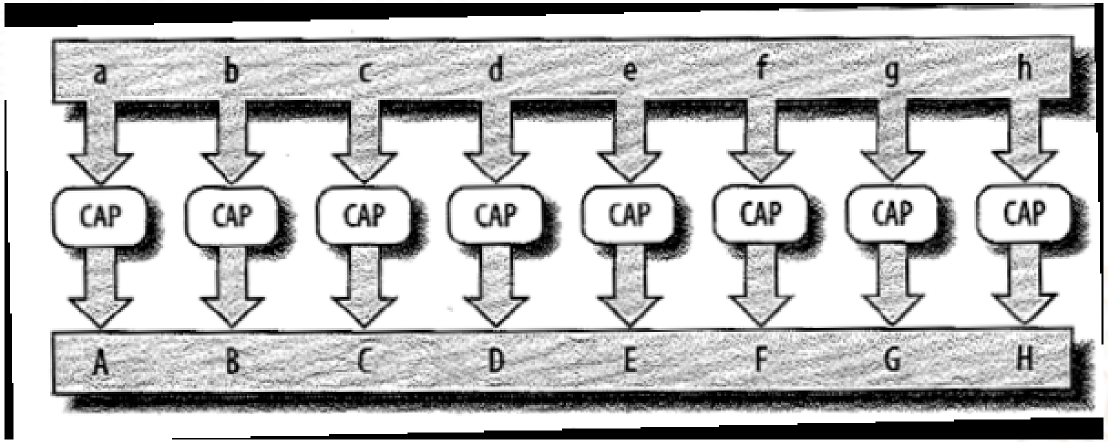
Asst Prof Liu Weichen
School of Computer Science and Engineering
Nanyang Technological University, Singapore

## Three levels of parallelism

- Instruction-Level Parallelism:
  - Multiple independent instructions are identified and grouped to be executed concurrently in different functional units in a single processor. Can reduce CPI to values less than 1. Examples: Superscalar and VLIW processors.
- Data-Level Parallelism:
  - The same operation is performed on multiple data values concurrently in multiple processing units. Can reduce the Instruction Count to enhance performance. Examples: Vector processors and array processors.
- Thread/Task Level Parallelism:
  - More than one independent threads/tasks are executed simultaneously. Can reduce the total execution time of multiple tasks. Examples: multi-core and multi-processor systems.

# **Data-level parallelism** (Part 1/2)

- Example: convert all characters in an array to upper-case
  - Can divide parts of the data between different tasks and perform the tasks in parallel.
  - Key: no dependencies between the tasks that cause their results to be ordered.



## **Data-level parallelism** (Part 2/2)

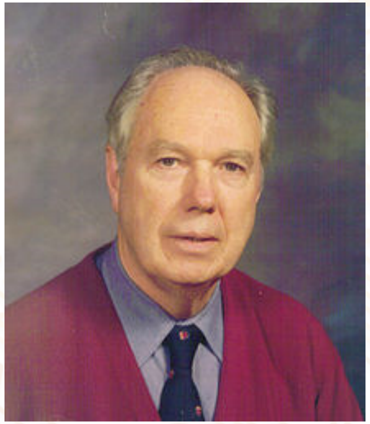
- Same set of operations are performed on different data elements.
    - Example: let **a**, **b**, and **c** be three vectors of length 100. All the 100 additions in the *for loop* can be executed concurrently.

```
for i ← 0 to 99 do
    a[i] ← b[i] + c[i]
endfor
```

- Such data parallel computation can be performed for matrix product, matrix addition, matrix-vector product, vector additions, and dot-product of vectors etc.
- Matrix computations are widely used in scientific computing. Similar data-level parallelism is also widely found in database operations, image, audio and video processing applications.
- Parallel computing systems need to be used for efficient use of data-level parallelism.

## Computing Models-Flynn's Classification

- Single instruction, single data stream – SISD
  - Conventional sequential processor
  - Not suitable to realize data-level parallelism
- Single instruction, multiple data stream – SIMD
  - Efficiently utilize the data-level parallelism
- Multiple instruction, single data stream – MISD
  - No commercial programmable system
- Multiple instruction, multiple data stream- MIMD
  - True multiprocessor system that issues multiple instructions
  - Can support data-level parallelism



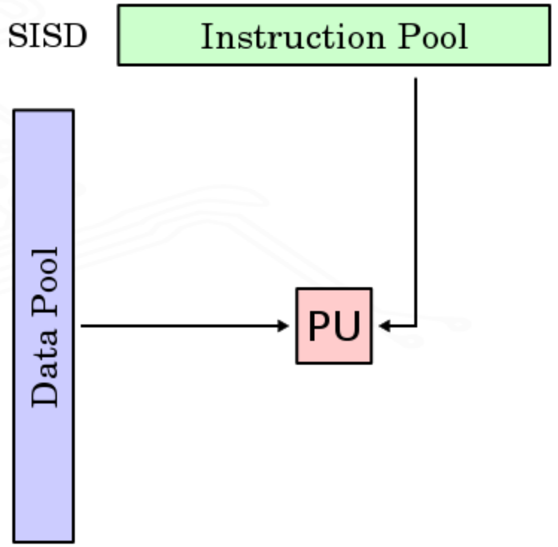
# Processor taxonomy (Part 1/4)

**processor architecture**

**single instruction single data stream (SISD)**

**uniprocessor system**

**pipeline**
**VLIW**
**superscalar/ multi-issue**



Ex: Intel Pentium 4

Conventional sequential processor
Not suitable to realize data-level parallelism

# Processor taxonomy (Part 2/4)

processor architecture

single instruction
single data stream
(SISD)

uniprocessor
system

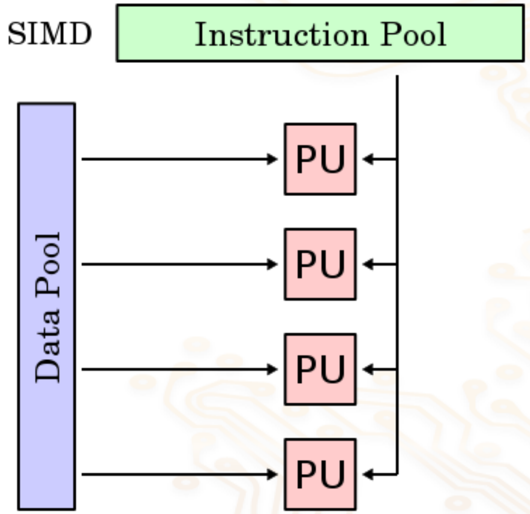
pipeline
VLIW
superscalar/ multi-
issue

Ex: Intel Pentium 4

single instruction
multiple data stream
(SIMD)

vector/array
processor

Ex: SSE instructions of x86
Efficiently uses data paralellism



# Processor taxonomy (Part 3/4)

**processor architecture**

**single instruction single data stream (SISD)**

**uniprocessor system**

**pipeline**
**VLIW**
**superscalar/ multi-issue**

Ex: Intel Pentium 4

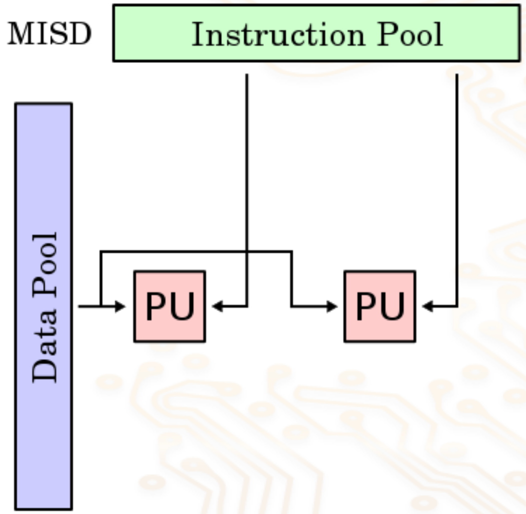
**single instruction multiple data stream (SIMD)**

**vector/array processor**

Ex: SSE instructions of x86

**multiple instructions singe data stream (MISD)**

No commercial programmable system



# Processor taxonomy (Part 4/4)

**processor architecture**

**single instruction single data stream (SISD)**

**uniprocessor system**

**pipeline**
**VLIW**
**superscalar/ multi-issue**

Ex: Intel Pentium 4

**single instruction multiple data stream (SIMD)**

**vector/array processor**

Ex: SSE instructions of x86

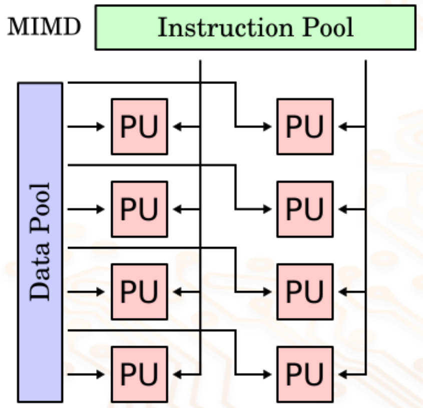
**multiple instructions singe data stream (MISD)**

No examples today

**multiple instruction multiple data streams (MIMD)**

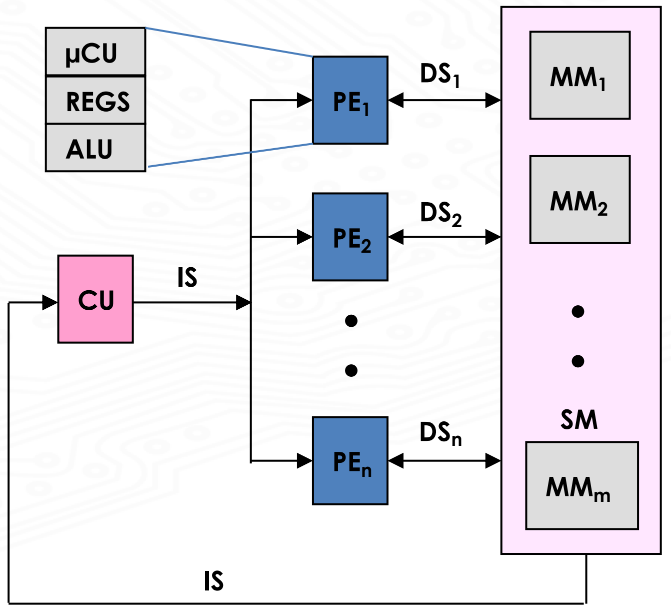
**conventional multiprocessor**

Ex: Intel Xeon e5345



# Single instruction multiple data stream – SIMD

SIMD architecture



**CU: Control Unit, PE: Processing Element, MM: Memory Module, SM: Shared Memory. DS: Data Stream, IS: Instruction Stream.**

- In each cycle only one machine instruction is issued: the **same instruction** is executed for **different data elements** by different processors.
- Central control unit issues instructions and controls simultaneous execution of instruction.
- Each processing element (PE) has its own data memory (MM) and also could interact via shared-memory (SM).
- All the PEs work synchronously.
- Applications: signal processing, matrix/scientific computation.
- Examples: vector and array processors

# Example of computing with SIMD system

- Consider a matrix-vector multiplication: **c= Ab**, where **A** is a 3x3matrix, and **b** is a vector of length 3. The product **c** is also a vector of length 3.

$$Ab= \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

| INSTRUCTION-1 | INSTRUCTION-2 | INSTRUCTION-3 |
|---|---|---|
| $t_{00}= a_{00}\, b_0$<br>$t_{10}= a_{10}\, b_0$<br>$t_{20}= a_{20}\, b_0$ | $t_{01}= a_{01}\, b_1$<br>$t_{11}= a_{11}\, b_1$<br>$t_{21}= a_{21}\, b_1$ | $t_{02}= a_{02}\, b_2$<br>$t_{12}= a_{12}\, b_2$<br>$t_{22}= a_{22}\, b_2$ |
| INSTRUCTION-4 | Initialize | $c_0 =0$ , $c_1 =0$ and $c_2 =0$ |
| INSTRUCTION-5 | INSTRUCTION-6 | INSTRUCTION-7 |
| $c_0 = c_0 + t_{00}$<br>$c_1 = c_1 + t_{10}$<br>$c_2 = c_2 + t_{20}$ | $c_0 = c_0 + t_{01}$<br>$c_1 = c_1 + t_{11}$<br>$c_2 = c_2 + t_{21}$ | $c_0 = c_0 + t_{02}$<br>$c_1 = c_1 + t_{12}$<br>$c_2 = c_2 + t_{22}$ |

- Product of a matrix of size *N* x *N* with a vector of length *N* can be completed by (2*N*+1) instructions.
- Sequential processing require nearly ($2N^2$) instructions.

  **SIMD array processors: Operations are performed on multiple data elements at the same time by multiple processing elements.**

## Advantages of SIMD over MIMD (Part 1/2)

- SIMD architectures can make use of data-level parallelism efficiently. If there are ‘*N*’ processing units, *N* operations can be performed in one clock cycle => *high performance*.

- Several operations on different data can be realized by one instruction => the *instruction bandwidth*, (i.e., the number of instruction to be used for a task) is reduced by a factor of *N*.

- Loop-overhead instructions (address-increment and branch condition check) also will be reduced by a factor of *N*.

## Advantages of SIMD over MIMD (Part 2/2)

- Energy used in instruction fetch will be reduced. SIMD is more energy efficient than MIMD. It makes SIMD attractive for personal mobile devices.

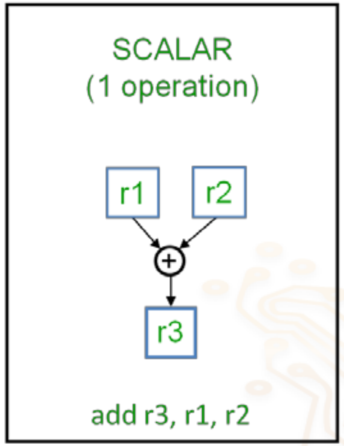
- Biggest advantage of SIMD versus MIMD is that the programmer continues to think sequentially yet achieves parallel speedup by having parallel data operation.

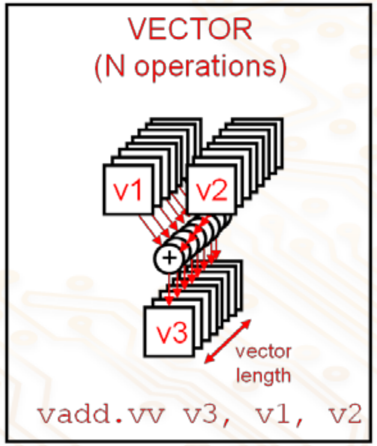
## Types of SIMD computing systems

- Vector Processors
  - Pipelined execution of many data operations.
  - Operations on multiple data elements are performed in consecutive time steps (clock cycles) in pipelined form.
- SIMD array processors
  - Operations are performed on multiple data elements at the same time by multiple processing elements.
- Multimedia SIMD instruction set extensions
  - MMX (Multimedia Extensions) in 1996
  - SSE (Streaming SIMD Extensions) versions in 2000
  - AVX (Advanced Vector Extensions) till today

## Vector Processor

- A kind of SIMD processor. The instructions of a vector processor operate on 1-dimensional arrays of data called vectors.
- Arrays are primitive data elements. Need to perform multiple arithmetic and logic operations in parallel. Consists of multiple functional units. *Vector-registers* and heavily interleaved memory.
- Typical vector operations are
  - **add** two vectors to produce a third
  - **subtract** two vectors to produce a third
  - **multiply** two vectors to produce a third
  - **divide** two vectors to produce a third
  - **load** a vector from memory
  - **store** a vector to memory





## Properties of Vector Processor (Part 1/2)

- Multiple parallel operations: One vector instruction can perform *N* computations, where *N* is the vector length.
    - involves less number of instructions
    - lower instruction count => less execution time => higher performance
- Computation in a given clock cycle is independent of previous result: Dependence check not required => Simpler design
    - allows deep pipeline of functional unit => high clock rate

## Properties of Vector Processor (Part 2/2)

- Fewer Branches: less branch overhead instructions
- Memory access pattern per vector instruction known
  - parallel memory access helps reducing memory latency
  - can use a high-bandwidth memory system
  - data caches need not be used

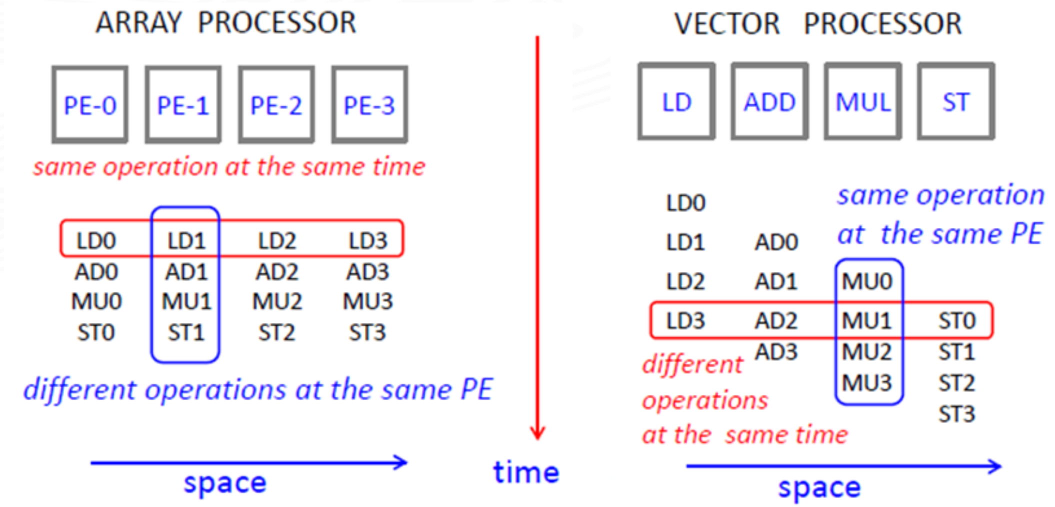
## Classes of Vector Processor

- **Memory-memory Vector Processors:**
  - Vector operands are fetched from memory and results are also stored in the memory.
- **Vector-register Processors:** Vector equivalent of load-store architectures
  - Except load and store instructions, for all instructions, operations are between registers.
  - Memory operands are made available in the registers and then operations are performed on register operands.

# SIMD: Array Processor vs Vector Processor

Instruction stream

| | | |
|---|---|---|
| LD | VR | ← A[3:0] |
| ADD | VR | ← VR, 10 |
| MUL | VR | ← VR, 20 |
| ST | A[3:0] | ← VR |



# SIMD instruction set extensions for multimedia (Part 1/2)

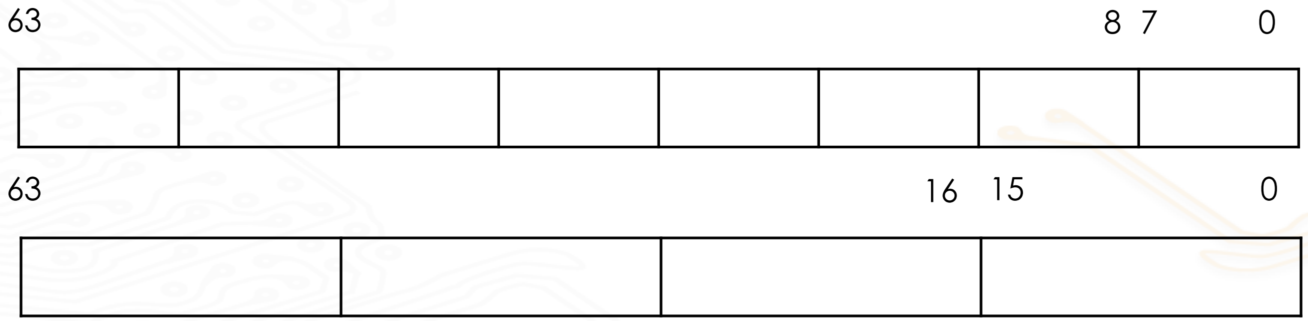
- Multimedia applications are popular and we need an instruction set architecture to match that properly.
  - Media applications exhibit high data-level parallelism.
  - Each instruction can operate on a group of data elements to reduce the instruction bandwidth.
- Most media applications operate on narrower data types.
  - Many graphics systems used 8 bits to represent each of the three primary colours plus 8 bits for transparency.
  - Image pixels are 8-bit integer and audio samples are usually represented by 8 or 16 bits, while bit-width of processor is usually 32 or 64-bit processors.

## SIMD instruction set extensions for multimedia (Part 2/2)

- Storing one data element in a word inefficiently utilizes the processor resources.
  - Can pack 4 or 8 data elements in a 32-bit or 64-bit word respectively?
  - Can read four elements of an array with one 32-bit load, and similarly for stores?

# SIMD extension

- Data elements are packed in a word like short vectors.



- Up to 8 operations can be performed in parallel in a 64bit processor.
  - Potential performance enhancement for image processing applications is 8x.
  - In practice the improvement is less than 8x but still good.

## MMX instruction set (Part 1/2)

- 57 new instructions comprised of
  - Integer arithmetic operations like **packed** add, sub, multiply, and multiply-add, packed logical operations.
  - Shift and rotate, Compare.
  - Move: from/to memory and from/to registers
  - Pack/Unpack (Conversion between packed data types)
- Like vector instructions, an SIMD instruction specifies the vector operations on vectors of data.

## MMX instruction set (Part 2/2)

- SIMD extension instructions specify fewer operands and hence use smaller register files compared with the vector processors with large register files such as the VMIPS.

- Does not require much modification over the existing architecture
  - small additional cost to enhance the standard ALU and easy to implement: increase in implementation complexity is marginal.
  - co-exists with the existing processor to implement general applications efficiently.

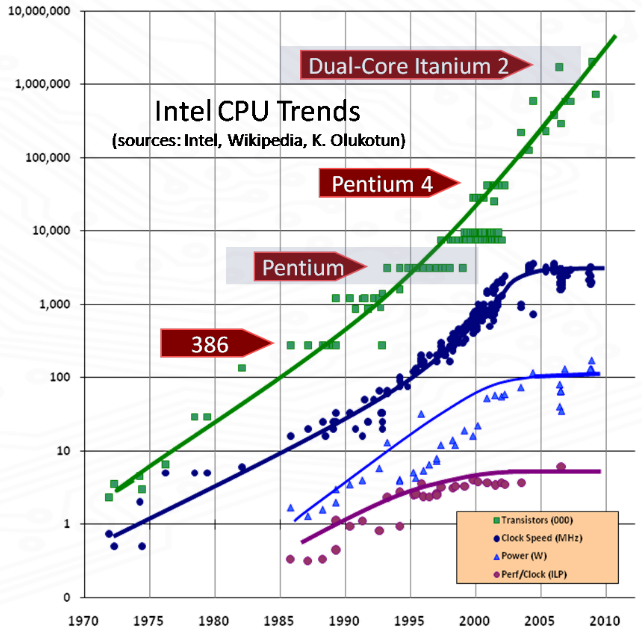
## Summary

- Data level parallelism
- Single instruction multiple data (SIMD)
  - Vector processor
  - Array processor
  - Multimedia SIMD instruction set extensions

## Part 2: Thread Level Parallelism

- Motivation for multicore systems
- Moore’s law in multicore
- Examples and applications for multicore systems
- Challenges in efficient multicore system design

## Motivation for multicore



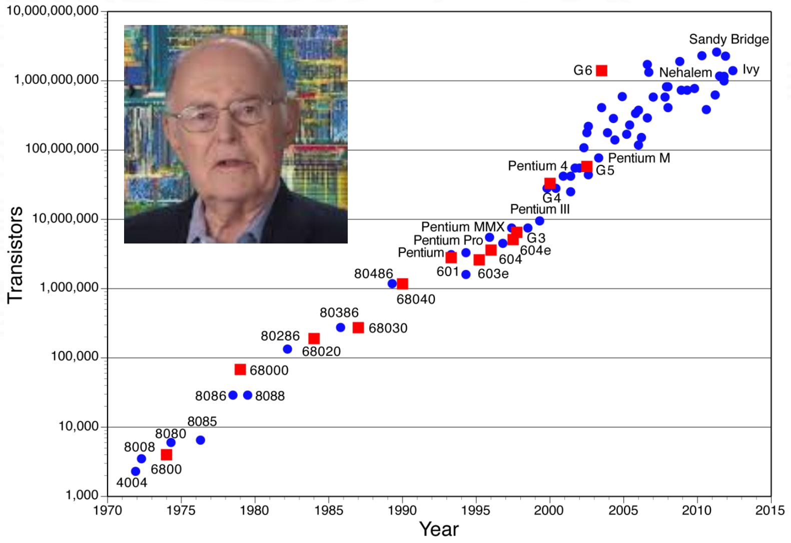
**Need for performance enhancement:**

- Need a faster, more capable system to support increasing transmission speed and communication bandwidth
- Increasing resolution of image/video data
- Increasing security need
- Increasing use of video conferencing and surveillance
- New applications, e.g., drug discovery with high volume DNA data processing, scientific computing, weather foresting

2.5 quintillion ($2.5\times10^{18}$) bytes of data created everyday —sensors and RFID, social network, digital pictures and videos, purchase records, GPS signals, email communications[IBM bigdata]

# Availability of computing resources: Moore's law

**Moore's law (1965):** the **number of transistors** in an IC chip **doubles approximately every eighteen months**.
Named after Gordon E. Moore, co-founder of Intel Corporation.



We get double more computing hardware resources after every 1.5 years!!

What are we going to do with that??

Computer architectures should be designed to use them judiciously.

# Traditional approaches for performance enhancement

**Performance is reciprocal of execution time**

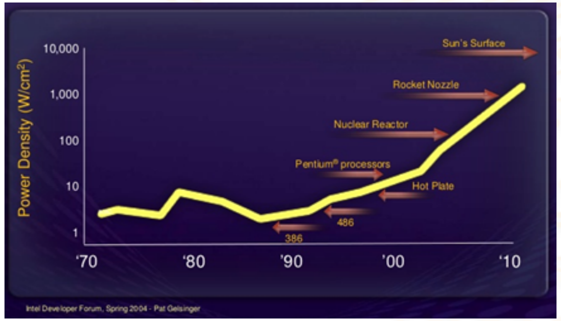
Execution time = IC x CPI x clock period

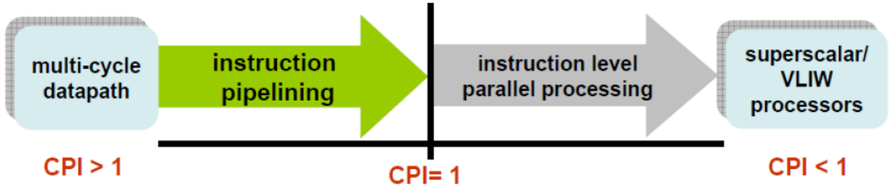
Two key traditional approaches to boost CPU performance:

- Increase the clock speed

  Not a good idea

- Decrease clocks per instruction (CPI) by instruction level parallelism



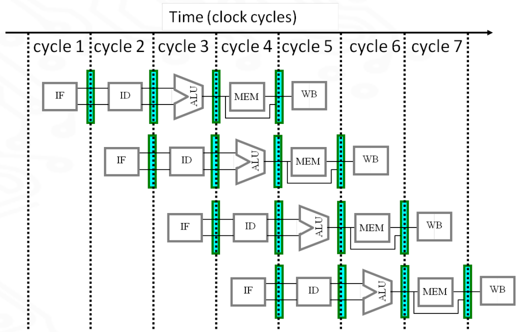

multi-cycle datapath
instruction pipelining
instruction level parallel processing
superscalar/ VLIW processors
CPI > 1
CPI= 1
CPI < 1

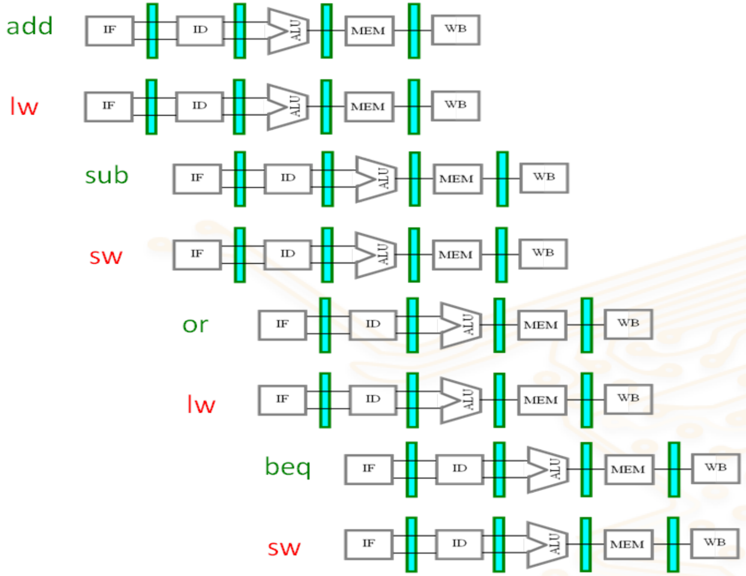
# Instruction level parallelism

1991-2000 was the decade of instruction level parallelism

- Pipelined instruction execution

- Multiple instructions issued per cycle: superscalar and VLIW processors



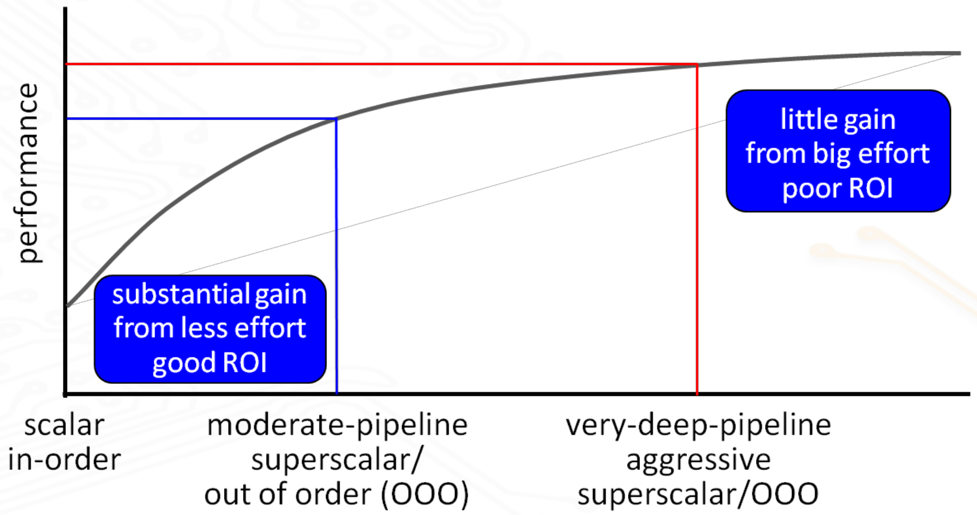


2-way superscalar

- Branch prediction, speculative execution, out-of-order execution

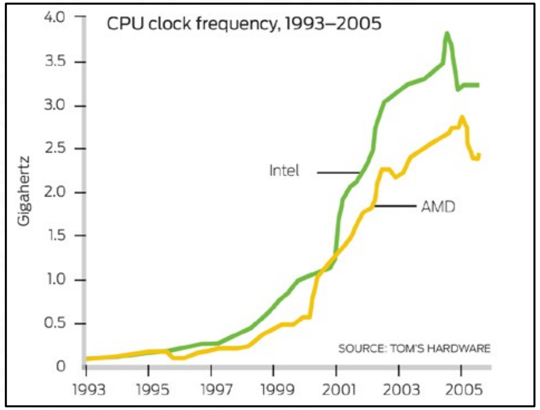
- Advanced cache design

## The ILP Wall



- High design and verification time and cost
- Diminishing returns in more ILP
- Instruction-level parallelism is near its limit

## The Power Wall

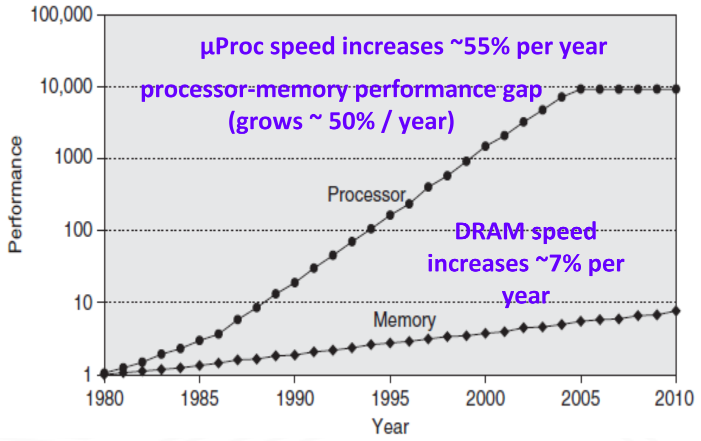


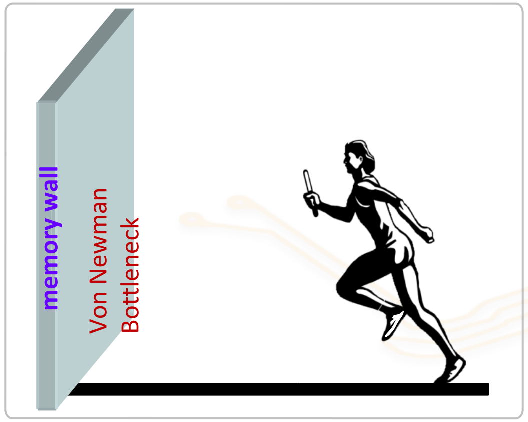
$$P = \alpha \cdot C \cdot V_{dd}{}^{2} \cdot f$$

$V_{dd}$ increases with $f$

- Power consumption increases more rapidly with the increasing operating frequency.
- Around the beginning of 2003, the ever-increasing processor speed was checked due to very high power consumption.
- Intel Tejas dissipated 150W heat at 2.80GHz: The Tejas had been projected to run 7 GHz. The project was cancelled in May 2004.

## The Memory Wall

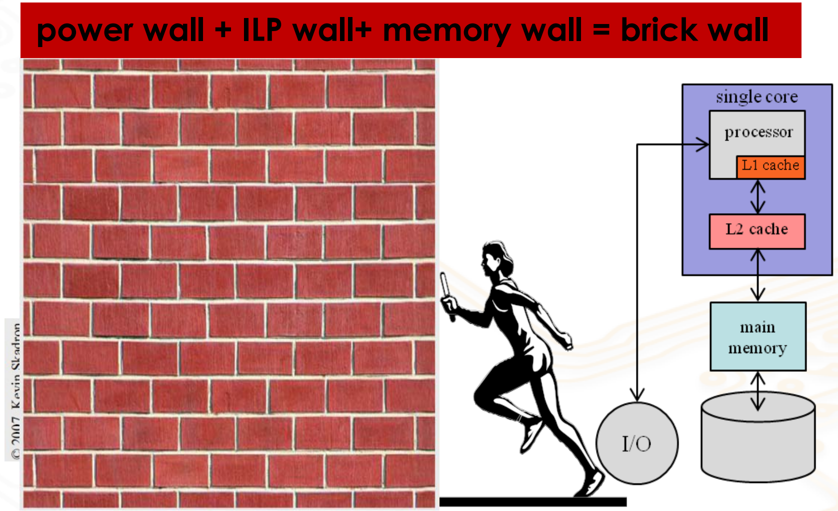


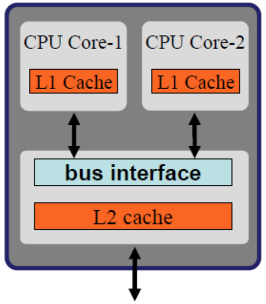


- Widening gap between compute bandwidth and memory bandwidth could not be bridged - resulting in memory latency.
- Memory hierarchy was proposed as a solution.
- But the increase in the size of the on-chip cache and cache optimization no longer yield improvement.

# Brick Wall

It could be possible to overcome the brick wall by a multicore processor. Containing two or more processors in the same chip, that provides enhanced performance by efficient simultaneous processing of multiple tasks **and consumes less power due to lower clock frequency**.



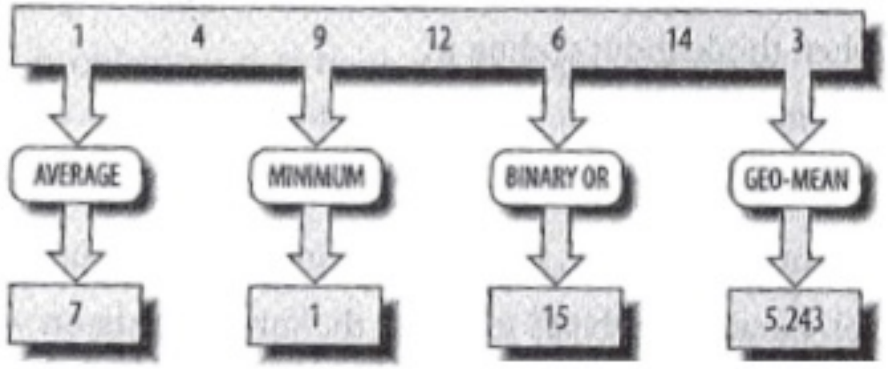


A generic dual core processor

# Need for Multicore (Part 1/2)

- Task Parallelism: Several functions on the same data: average, minimum, binary or geometric mean

- No dependencies between the tasks, so all can run in parallel



## Need for Multicore (Part 2/2)

Overcome power wall using multiple slow cores

- Cores running at lower clock frequency and lower voltage can still deliver the desired performance using less power
- Scale up the number of cores rather than frequency

Overcome memory wall with memory parallelism

Derive parallelism by thread-level parallelism

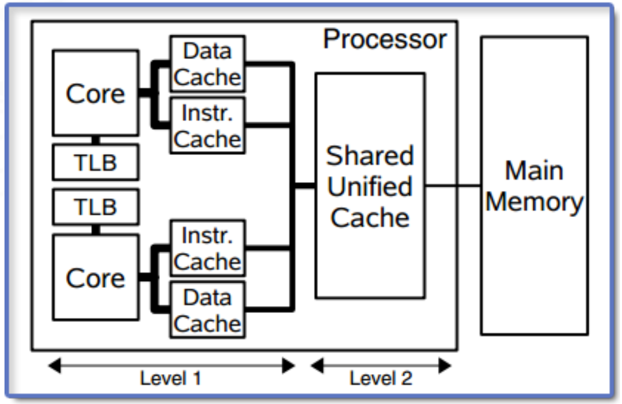
Homogenous multicore systems: consists of identical cores: less design and verification cost

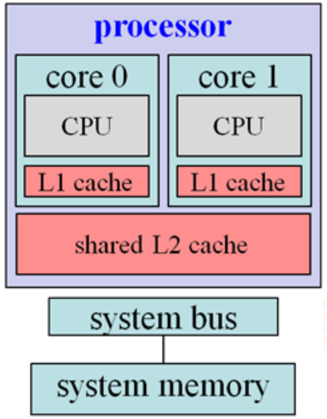
Heterogeneous multicore systems: contains different types of cores, each optimized for a different role

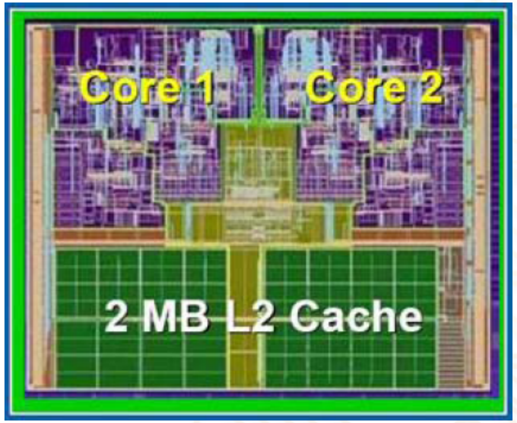
# Examples of multicore processors (Part 1/2)

A multicore processor is a single computing component with two or more “independent” processors (called "cores").

Dual core processors

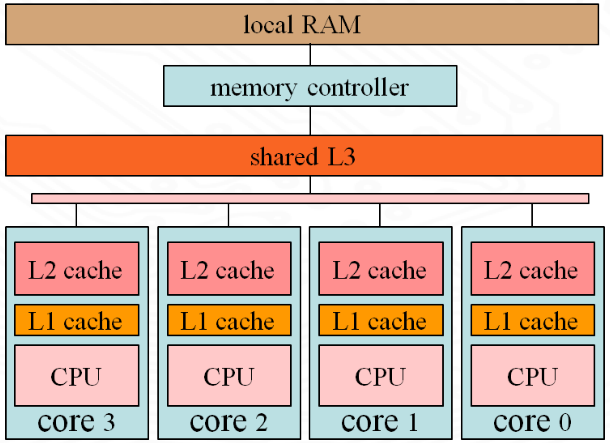






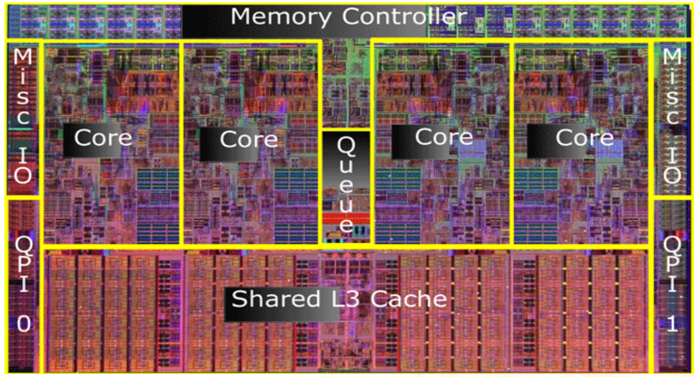
Ex: AMD Phenom II X2, Intel Core 2 Duo E8500, iPhone 6

# Examples of multicore processors (Part 2/2)



Quad core processor

HTC One X,
Samsung galaxy note 2



EX: AMD Phenom II X4,
Intel Core i5 2500T,
Intel Nehalem

Intel Quad Core Nehalem

## Applications of Multicore (Part 1/2)

What applications benefit from multi-core?

- Database servers
- Web servers
- Telecommunication markets
- Multimedia applications
- Scientific applications
- In general, applications with Thread-level parallelism (as opposed to instruction-level parallelism)

## Applications of Multicore (Part 2/2)

Examples

- Editing a photo while recording a TV show through a digital video recorder.
- Downloading software while running an anti-virus program.
- "Anything that can be threaded today will map efficiently to multi-core".

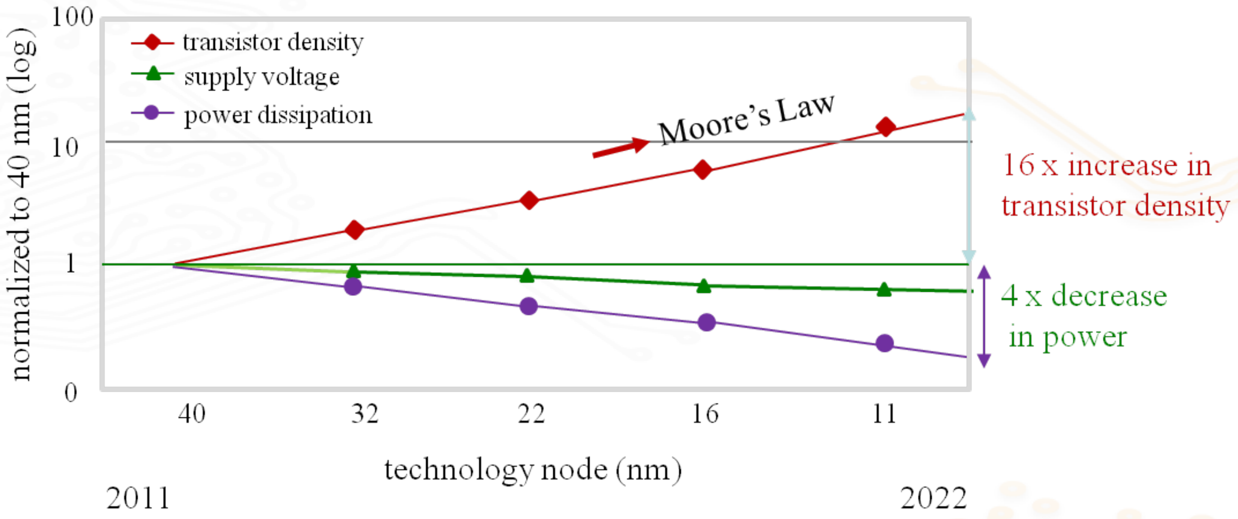
**BUT: some applications difficult to parallelize.**

## Does the performance of a multicore processor improve as the number of cores increases?

*The reality is that the quad-core could be better, it could be equal, or it could be appreciably worse. -- Nick DiCarlo, Samsung*

# Challenges : A new power wall

Technology scaling continues according to Moore's law → increasing transistor density → more heat dissipation per unit area

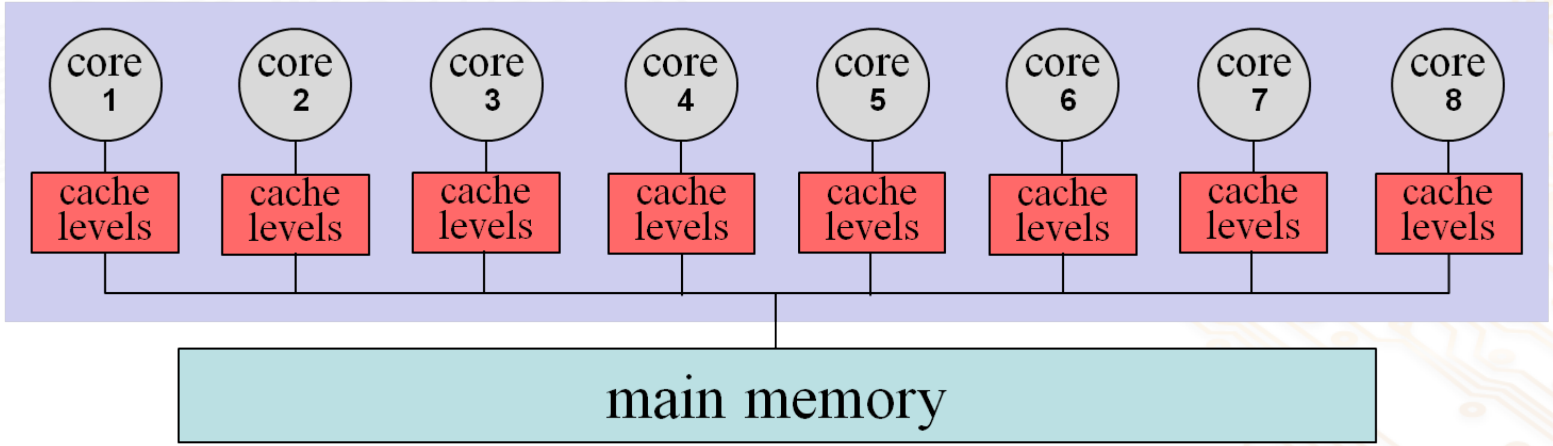


**The new power wall is going to constrain multicore too.**

## Challenges : Return of memory wall

**Increase in memory latency is coming as a new memory wall**

- As the number of cores increases, the performance enhancement is degraded:
  - Due to lack of memory bandwidth
  - Contention between cores over the memory bus



## Challenges : Interconnect problem

**On-chip interconnects are becoming a critical bottleneck**

- Increasing number of cores, interconnect length increases since data moves across the cores on the chip
- Interconnect delay increases with technology: would be reaching above 12 ns for 1 mm of copper wire by 2020
- The interconnect power density has been rapidly increasing with feature size
- Also requires mechanisms for efficient inter-processor coordination
    - Synchronization
    - Mutual exclusion
    - Context switching

**Anything more to note?**

## Summary

- Motivation for multicore systems
- Examples for multicore
- Application areas
- Challenges in efficient multicore system design