M7 Data and Thread Level Parallelism

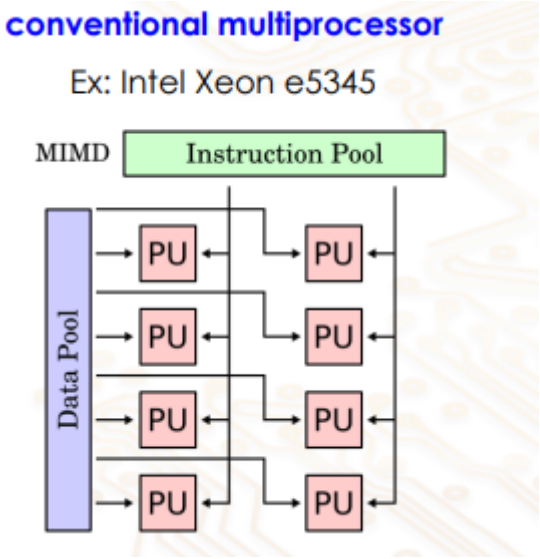
1. Data-Level Parallelism

- The same operation is performed on **multiple data values** concurrently in multiple processing units, this reduces instruction count (IC) to enhance performance
    - Example: converting all characters in a string to uppercase

1.1 Flynn's Classification

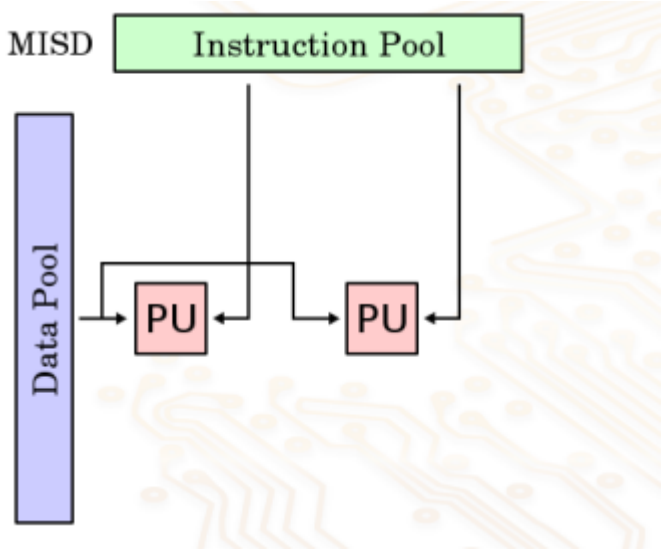
Multiple Instruction, Multiple Data Stream

- True multiprocessor system that issues multiple instructions and support data-level parallelism



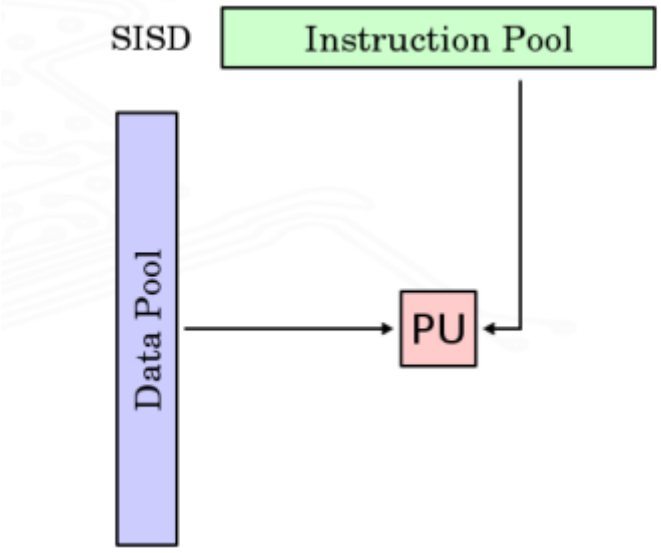
Multiple Instruction, Single Data Stream

- No commercial programmable system



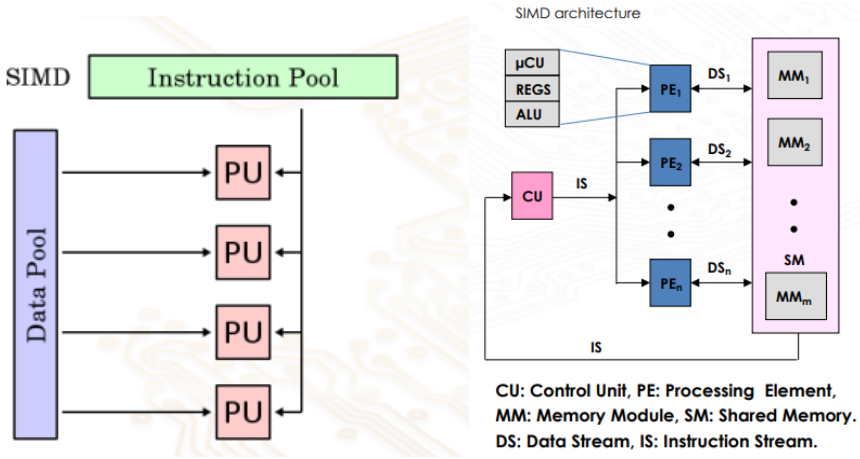
Single Instruction, Single Data Stream

- Conventional sequential processor
- Not suitable to realise data-level parallelism



Single Instruction, Multiple Data Stream

- Efficiently utilises data-level parallelism



- In each cycle only one machine instruction is issued for different data elements by different processor units
- Central CU issues instructions and controls simultaneous execution of instruction; all PEs work synchronously and each has its own MM and can also interact via shared memory.
  - 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
$t_{00} = a_{00}\, b_0$
$t_{10} = a_{10}\, b_0$
$t_{20} = a_{20}\, b_0$

INSTRUCTION-2
$t_{01} = a_{01}\, b_1$
$t_{11} = a_{11}\, b_1$
$t_{21} = a_{21}\, b_1$

INSTRUCTION-3
$t_{02} = a_{02}\, b_2$
$t_{12} = a_{12}\, b_2$
$t_{22} = a_{22}\, b_2$

INSTRUCTION-4
Initialize
$c_0 = 0$, $c_1 = 0$ and $c_2 = 0$

INSTRUCTION-5
$c_0 = c_0 + t_{00}$
$c_1 = c_1 + t_{10}$
$c_2 = c_2 + t_{20}$

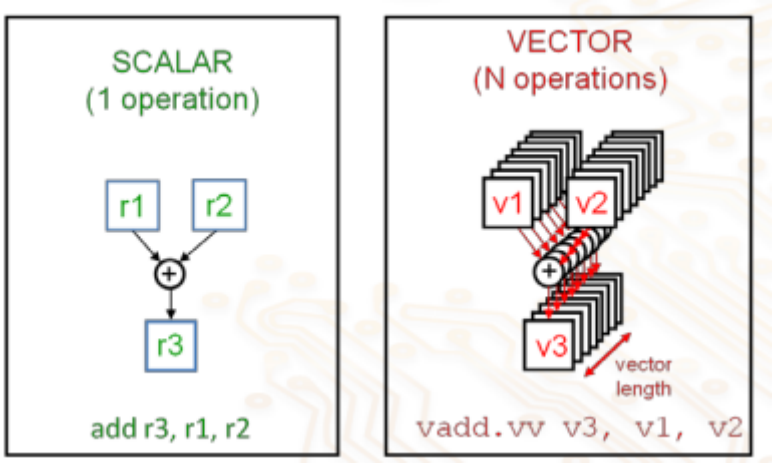
INSTRUCTION-6
$c_0 = c_0 + t_{01}$
$c_1 = c_1 + t_{11}$
$c_2 = c_2 + t_{21}$

INSTRUCTION-7
$c_0 = c_0 + t_{02}$
$c_1 = c_1 + t_{12}$
$c_2 = c_2 + t_{22}$

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

1.2 SIMD

- **Advantages (over MIMD)**
  - Able to make use of data-level parallelism efficiently (if there are N processing units, N operations can be performed in one cycle), giving high performance
  - Several operations on different data can be realised by one instruction, reducing instruction bandwidth by a factor of N
  - Loop-overhead instructions (address increment and branch condition check) can also be reduced by a factor of N
  - Energy used in instruction fetch will be reduced; more energy efficient compared to MIMD
  - Programmer is able to think sequentially yet achieve parallel speedup by having parallel data operation.
- **Vector Processors**
  - Instructions operate on 1-D arrays of data (vectors)

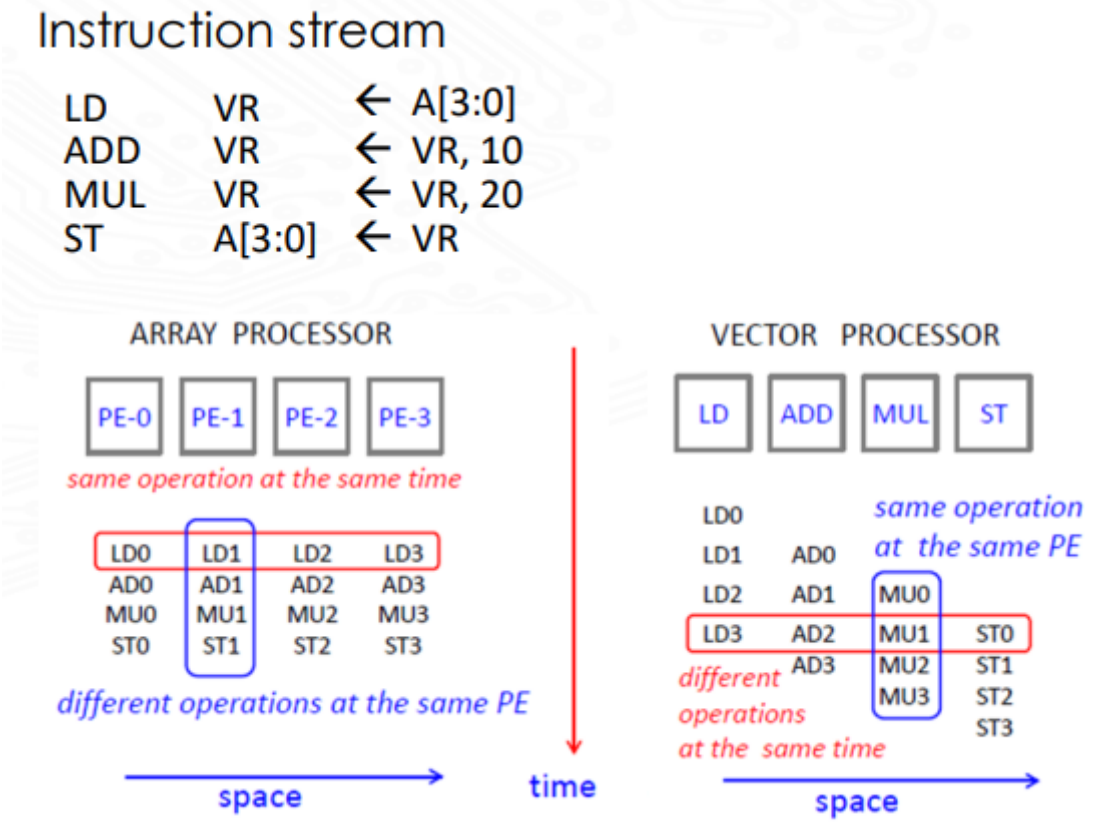


  - One vector instruction can perform N (length of array) computations
    - Involves less number of instructions, reducing execution time and improving performance
  - Computation in a given clock cycle is independent of previous result, meaning there is no need to check for dependencies, allowing simpler design and deep pipeline of functional unit, resulting in higher clock rate
  - Fewer branches (less branch overhead instructions)
  - Pattern of memory access per vector instruction known
    - Parallel memory access helps reduce memory latency
    - Can use a high-bandwidth memory system
    - Data caches need not be used

Classes

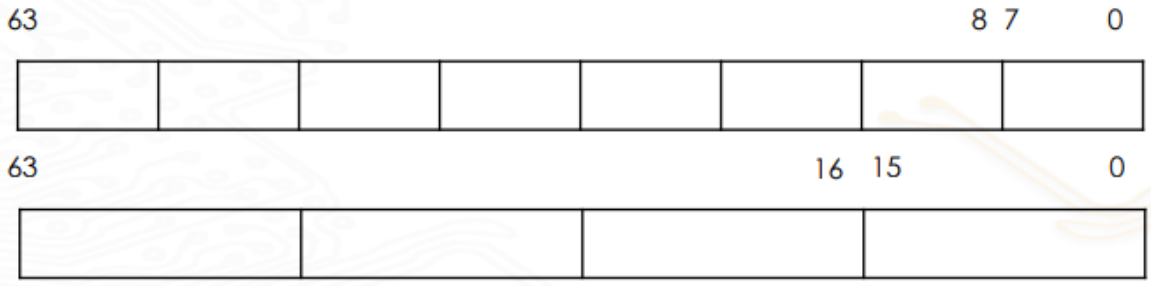
- **Memory-memory Vector Processors**
  - Vector operands are fetched from memory and results are also stored in the memory

- **Vector-register Processors**
  - All operations are between registers (except load and store)
  - Memory operations are made available in registers and then performed on register operands



Instruction Set Extension (MMX)

- Need an ISA to match multimedia whcih can operate on a group of data elements to reduce instruction bandwidth
- Storing one data element in a word inefficiently utilises the processor's resources. To solve this, data elements are packed in a word like short vectors



  - Up to 8 operations can be performed in parallel in a 64bit processor
- SIMD extension instructions specify fewer operands and use smaller register files compared to vector processors with large register files.
- Does not require much modification over existing architecture

2. Thread-Level Parallelism

- More than one independent threads / task are executed simultaneously, reduces the total execution time of multiple tasks
- **Traditional approaches for performance enhancements**
  - Execution time = instruction count x clocks per instructions x clock speed
  - To boost cpu performance (reduce execution time)
    - Increase clock speed (not recommended)
    - Decrease CPI using instruction level parallelism (previous chapter)

2.1 Brick Wall

- **ILP wall**
  - High design and verification time and cost
  - Diminishing returns in more ILP
  - ILP is near limit
- **Power Wall**
  - Power consumption increases exponentially with increasing operating frequency
  - $P = \alpha \times C \times {V_{dd}}^2 \times f$
- **Memory Wall**
  - Cannot bridge widening gap between compute bandwidth (55% growth in processing speed every year) and memory bandwidth (7% growth in DRAM speed every year)
  - Increasing size of on-chip cache plateaued
- Possibly overcome using a multicore processor which contains two or more processors in the same chip.
  - Provides enhanced performance by efficient simultaneous processing of multiple tasks and consumes less power due to lower clock frequency

Need for multicore

- Task Parallelism - Performing several functions (average / minimum / binary etc) on same set of data with no dependencies between each task (functions)
- Cores running at lower clock frequency and lower voltage can still deliver the desired performance using less power; performance scales up with number of cores rather than clock frequency

2.2 New challenges

- With technology scaling with moore's law, transistor density increases, causing heat dissipation per unit area to increase (**power wall**)
- Since number of cores are increased, performance enhancement is degraded due to lack of memory bandwidth and contention between cores over memory bus (**memory wall**)
- **Interconnect problem**

- Increasing number of cores causes interconnect length to increase since data moves across the cores on the chip $\implies$ interconnect delay will increase with technology
- Interconnect power density has been increasing with feature size
- Requires mechanisms like synchronisation, mutual exclusion, context switching for efficient inter-processor coordination