Parallel and Concurrent Programming with Java

## Chapter 1. Computing Hardware

# Sequential vs Parallel computing

sequential/serial computing

- simple

- simplest form: only use 1 processor

- program is series of discrete instructions that is executed one after another

- can only execute one instruction at any one time, no overlap

- time to complete limited by how fast processor can process series of instructions

parallel computing

- complex

- 2 or more processors running in parallel

- different discrete instructions of a program can be executed simultaneously by different processors

- program runs faster, but does not mean 2x faster

- needs communication to coordinate. One may need to wait for other/s to finish step.

- need logic in codes to coordinate and for waiting.

- benefit: improve throughput.

* More tasks finished per unit time.
* A large task, broken down into smaller steps, can be finished, and finished faster.

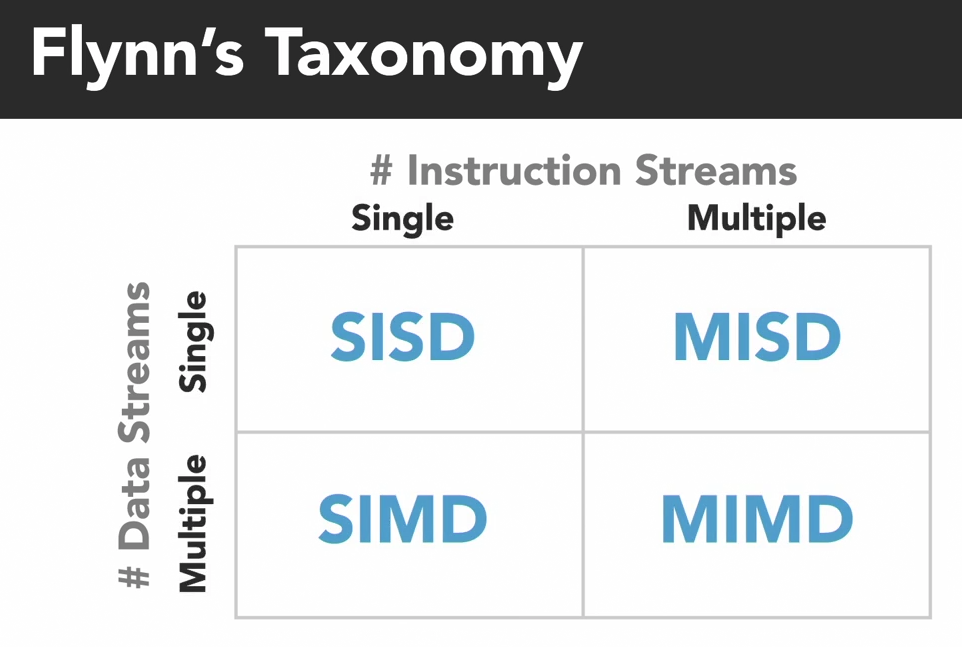
- some computing problems too large/complex to be completed with one processor, e.g., serving millions of search engine queries per second, nuclear modelling, weather forecasting.

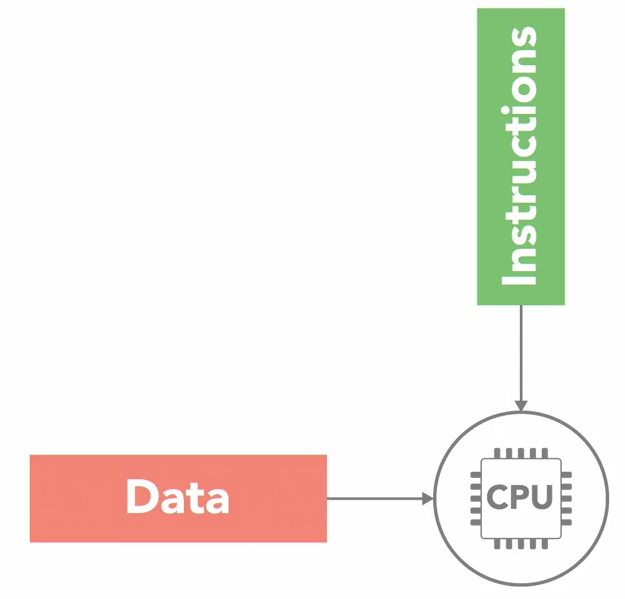
- time saved = money saved.

# Parallel computing architecture

about hardware

a popular classification method: Flynn’s Taxonomy



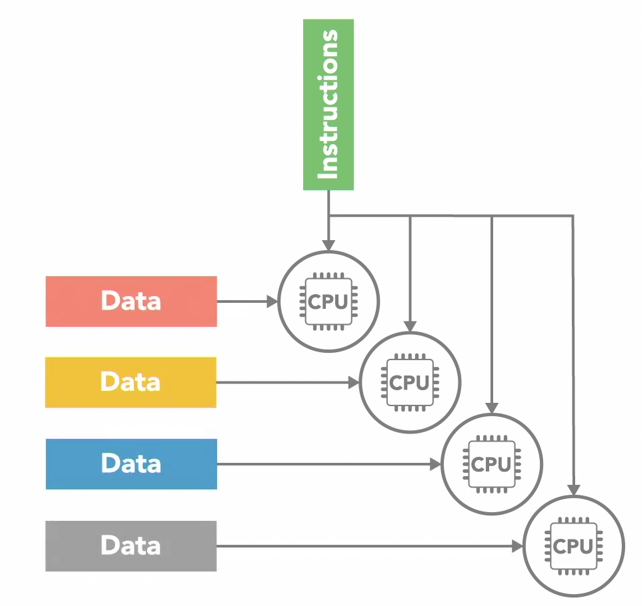
Single Instruction, Single Data (SISD)

- simplest

- 1 CPU

- example of

* single instruction: chop
* single data: a carrot



Single Instruction, Multiple Data (SIMD)

- >1 CPU

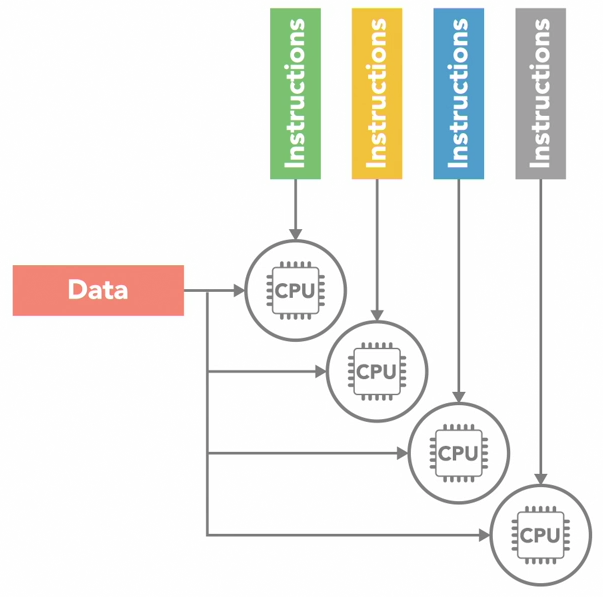
- example of

* single instruction: chop
* multiple data: a carrot and an onion

- single instruction, so CPUs work in sync

- multiple different data streams

- suitable for performing simple set of instructions on big data set, e.g., image processing by GPUs.

Multiple Instructions, Single Data (MISD)

- >1 CPU

- multiple processors execute multiple different instructions, independently

- but on same single stream of data

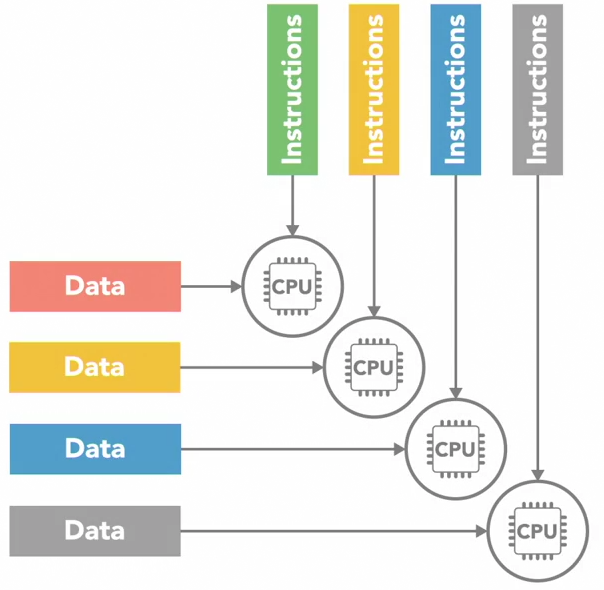
example of

* multiple instructions: chop, peel
* single data: a carrot

- not commonly used

- not practical

Multiple Instructions, Multiple Data (MIMD)



- >1 CPU

- each processing unit executing different series of instructions

- each processor working on a different stream of data

example of

* multiple instructions: chop, peel
* multiple data: carrot, onion
* chop carrot, peel onion

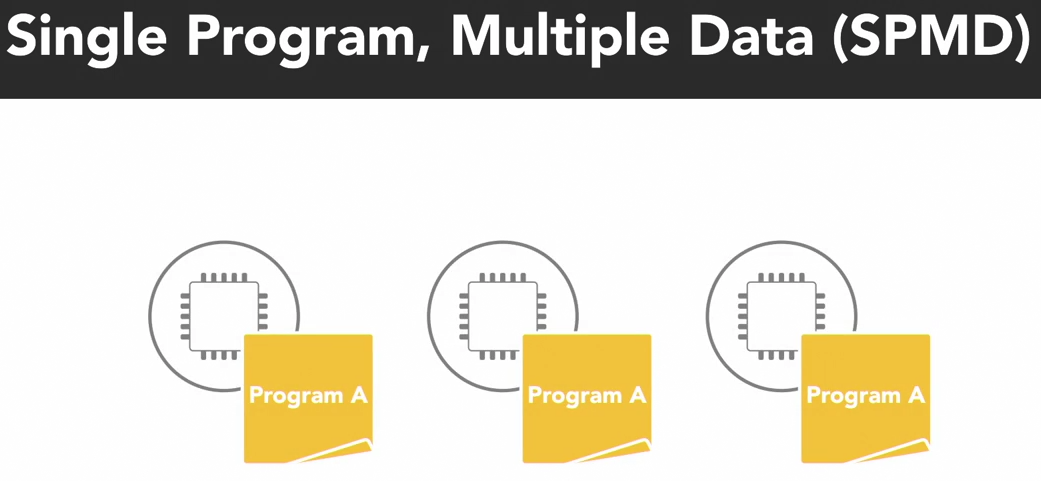
- most used design

- from multi-core PCs to networked super-computers

MIMD can be subdivided

-> Single Program, Multiple Data (SPMD)

-> Multiple Program, Multiple Data (MPMD)



SPMD

- analogy: one program (recipe) -> 2 processing units (cooks) -> executing different instructions (chop, peel) on different data (carrot, onion).

– multiple processors

- executing copies of same single program

- a program is a series of instructions

- each processor is executing a different instruction. Not the same single instruction.

- processors are asynchronous

- program usually has conditional logic that allows processor to execute specific sub-set of instructions only.

- e.g., single program running on multi-core processor (MIMD).

MPMD

- analogy: 2 cooks following different recipes in cookbook

- multiple processors

- concurrently, each processor executing a different program. Independent.

- typically, a processing node is selected as host/manager, running a program. Program farms out data to other processing nodes running a different program.

- after worker nodes complete work, report back to manager node.

- this architecture used for functional decomposition.

# Shared vs Distributed Memory

* instructions and data are stored in memory
* Diagram

  Description automatically generatedprocessors must be able to access memory fast
* adding processors does not help if memory access is slow
* usually, memory speed < processor speed
* when 1 memory element is read/write by 1 processor, other processors have no access
* 2 main memory architectures for // computing: shared and distributed memory

Shared memory system

* All processors access same memory in one global address space
* Processors operate independently
* But if 1 processor makes change to a spared memory address, all other processors can see that change
* Shared memory need not be on same physical disk, can be spread across cluster of systems
* Divided into 2 categories

Text

Description automatically generated

* Diagram

  Description automatically generatedBased on how processors are connected to memory and how quickly processors can access memory
* UMA
  + All processors have equally fast access to memory
  + There are several types of UMA architectures. Most common is Symmetric Multiprocessing System (SMP)
    - SMP – 2 or more identical processors connected to a single shared memory, often through a system bus
    - In multi-core CPU, each core treated as separate processor
    - Diagram

      Description automatically generatedModern multi-core CPU has cache memory for each core.
    - Cache is very fast, small local memory
    - Only that processor can access it
    - Used to store data it is frequently working with
    - Cache coherency: the challenge when processor copies data from main shared memory and makes change in local cache; main memory not updated before another processor copies outdated value to its local cache.

Diagram

Description automatically generated

* + - Handled by hardware in multi-core processors
    - Be aware of issue for large, complex // computing systems.
* NUMA

Diagram

Description automatically generated

* Usually made by physically connecting multiple SMP systems together
* A core has faster access to memory within its SMP unit, but slower access to memory from another SMP unit across system bus
* Overall, every processor can still see everything in memory
* Disadvantages of Shared Memory Architectures (SMAs)
  + Adding processors will increase traffic on shared memory bus
  + Cache coherency
  + Programmer needs to synchronise memory access to ensure correct behavior

Distributed Memory System
Description automatically generatedDistributed memory system

* each processor has its own local memory, own address space
* No global memory spaces
* All processors connected by a network, e.g., ethernet

Diagram, timeline

Description automatically generated

* Each processor makes changes to data in its local memory. Not reflected in local memory of other processors
* A processor is oblivious to what instructions/data other processors are executing
* Disadvantage: programmer must code when and how data is communicated between the nodes
* Advantage: easy to scale. Add off-the-shelf CPU/memory units and networking equipment build large, distributed memory systems.

Timeline

Description automatically generated