# CH1:

Parallel Computing: s a form of computation in which many calculations are carried out **simultaneously** (at the same time).

Parallel Programming:

- Decomposing a programming problem into tasks
- Deploy the tasks on multiple processors and run them simultaneously (at the same time)

**hardware supports parallelism:**

Multi-core computers have multiple processing elements (cores) within a single machine.

Symmetric Multiprocessing (SMP): multiple processors sharing a single address space, All processors are treated equally by the OS. **(one processor control all the processors)**

Cluster: A parallel system consisting of a combination of commodity units (processors or SMPs) **(grouped together by hardware)**

Grid: is a group of networked computers that work together, only when idle, as a virtual supercomputer to perform large tasks. (**grouped together by network**)

Opportunities to use parallelism:

| | |
|---|---|
| Instruction Level Parallelism | Hidden Parallelism in computer programs (المسؤول عنها الكومبايلر) |
| Single computer level | Multi-core computers – multi processor computers |
| Multiple computers level | Clusters, Servers, Grid computing |

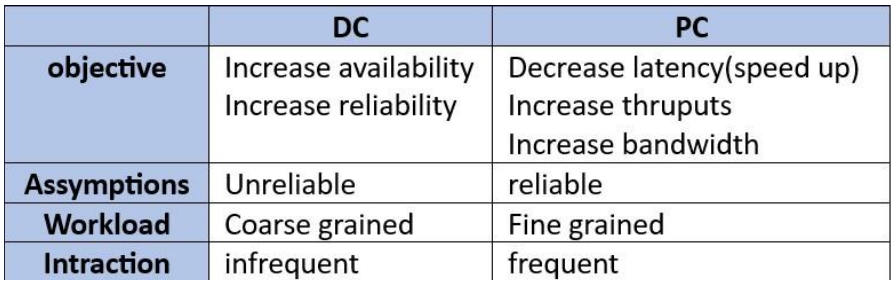
**programs are written assuming that:**

- Instructions are executed in sequential.
- Most programming languages embed sequential execution.

Task parallelism: Partition various tasks carried out solving the problem among the cores. (الخوارزمية تتغير تكن الداتا نفسها)

Data parallelism: Partition the data used in solving the problem among the cores. (الخوارزمية نفسها لكن الداتا تتغير)

| | |
|---|---|
| latency | How much time to finish task. |
| Throughput | How much of work we can finish in period of time. |
| Bandwidth | How much data we can process in unit of time. |



| | DC | PC |
|---|---|---|
| **objective** | Increase availability<br>Increase reliability | Decrease latency(speed up)<br>Increase thruputs<br>Increase bandwidth |
| **Assymptions** | Unreliable | reliable |
| **Workload** | Coarse grained | Fine grained |
| **Intraction** | infrequent | frequent |

What is a Distributed System? A distributed system is one that looks to its users like an ordinary, centralized, system but runs on multiple independent CPUs.

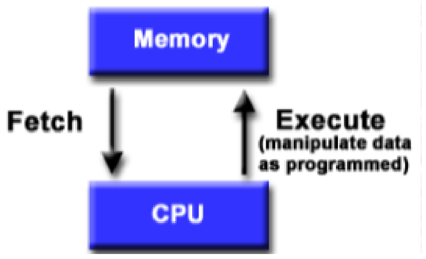
Processing unit failures are independent:

- No single point of failure
- No global clock
- No global state

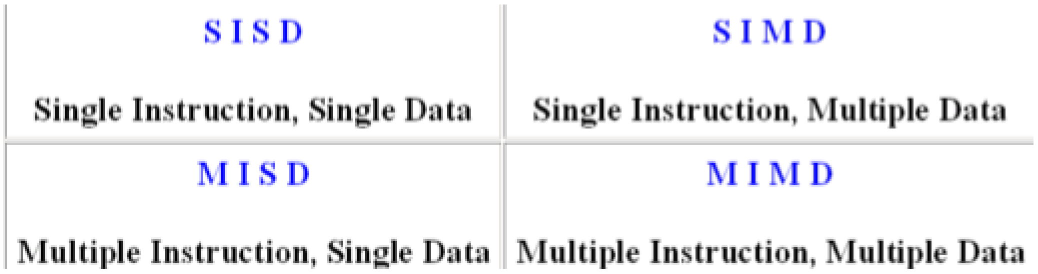
# CH2:

Von Neumann Architecture: For over 40 years, the von Neumann architecture, named after John von Neumann, has been the standard for computer design, utilizing the stored-program concept where the CPU executes a program that dictates memory read and write operations.

The basic design of a computer involves using memory to store both program instructions and data. Program instructions are coded commands for the computer, while data is the information processed by these commands. The central processing unit (CPU) retrieves and decodes instructions from memory, executing them sequentially.



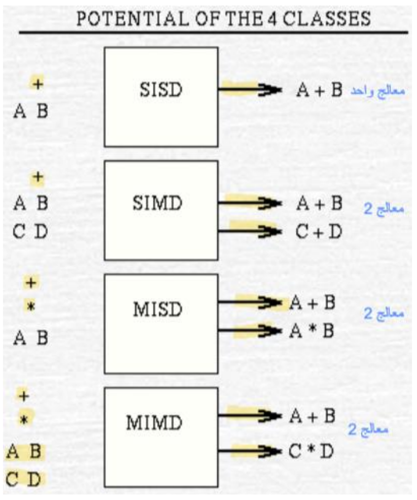
Flynn's Classical Taxonomy



| | |
|---|---|
| **SISD**<br>Single Instruction, Single Data | **SIMD**<br>Single Instruction, Multiple Data |
| **MISD**<br>Multiple Instruction, Single Data | **MIMD**<br>Multiple Instruction, Multiple Data |

A stream of instructions (the algorithm) tells the computer what to do.

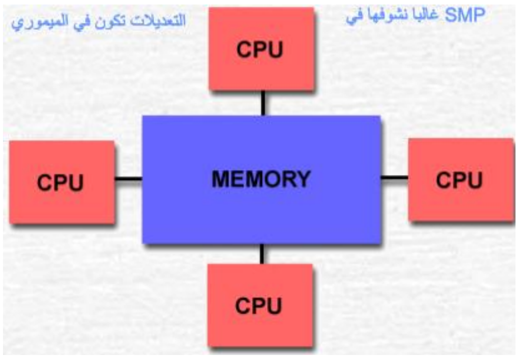
A stream of data (the input) is affected by these instructions.

| | | |
|---|---|---|
| **SISD** | A serial (non-parallel) computer , A serial computer operates on a single instruction and a single data stream per clock cycle, representing the oldest and, until recently, most common computer form, including most PCs, single CPU workstations, and mainframes. |   POTENTIAL OF THE 4 CLASSES<br>+<br>A B<br>SISD<br>A + B<br>معالج واحد<br>+<br>A B<br>C D<br>SIMD<br>A + B<br>C + D<br>معالج 2<br>+<br>*<br>A B<br>MISD<br>A + B<br>A * B<br>معالج 2<br>+<br>*<br>A B<br>C D<br>MIMD<br>A + B<br>C * D<br>معالج 2 |
| **SIMD** | In the Single Instruction, Multiple Data (SIMD) model, all processing units execute the same instruction simultaneously on different data elements, making it ideal for regular, specialized tasks like image processing. (Data parallelism) | |
| **MISD** | Multiple processing units operate independently on a single data stream using their unique instruction streams, enabling applications like applying various frequency filters to a signal or employing multiple cryptography algorithms to decrypt a single message. (Task parallelism) | |
| **MIMD** | The prevalent form of parallel computing today involves Multiple Instruction, Multiple Data (MIMD) systems where each processor executes different instructions on different data streams, exemplified by modern supercomputers, parallel computer grids, and multi-processor systems, including certain PCs. (you can do Data parallelism or Task parallelism) | |

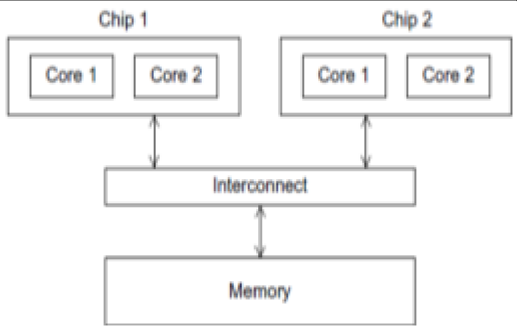
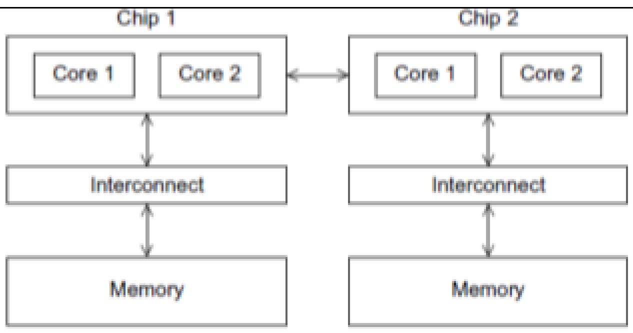
Parallel Paradigms:

| | | |
|---|---|---|
| 1. Shared Memory | 2. Message Passing | 3. Multi-threading |

1- Shared Memory: **is memory that may be simultaneously accessed by multiple programs**. Shared memory is an efficient means of passing data between programs.



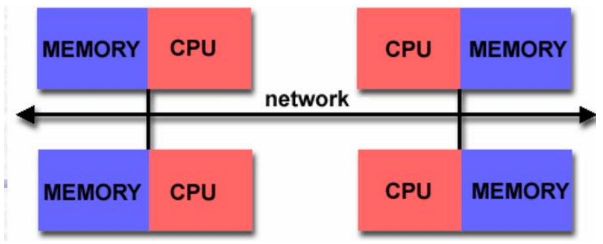
**Shared memory machines** can be divided into two main classes **UMA and NUMA.**

| | | |
|---|---|---|
| **Uniform Memory Access (UMA)** | • Represented by Symmetric Multiprocessor (SMP) machines.<br>• Identical processors ( نكون نفس نوع المعالج بالضبط)<br>• **Equal** access and access times to memory.<br>• Cache Coherent UMA (CC-UMA) ensures that when one processor updates a shared memory location, all other processors are immediately aware of the change, maintaining consistency through hardware-level mechanisms. |  Chip 1<br>Chip 2<br>Core 1<br>Core 2<br>Core 1<br>Core 2<br>Interconnect<br>Memory |
| **Non-Uniform Memory Access (NUMA)** | • Often made by physically linking two or more SMPs<br>• One SMP can directly access memory of another SMP.<br>• Not all processors have equal access time to all memories.<br>• Memory access across link is slower.<br>• If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA. |  Chip 1<br>Chip 2<br>Core 1<br>Core 2<br>Core 1<br>Core 2<br>Interconnect<br>Interconnect<br>Memory<br>Memory |

## Centralized Shared Memory

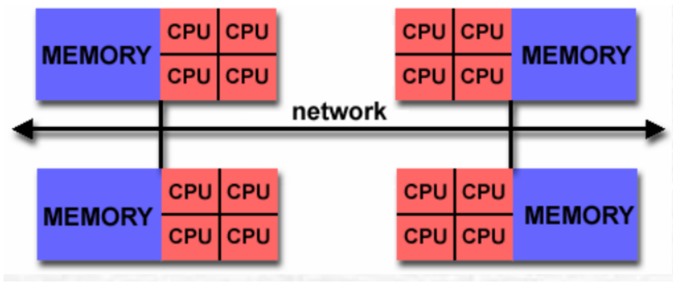
| Advantages | Disadvantages |
|---|---|
| Global address space provides a user-friendly programming perspective to memory | Adding more CPUs to shared memory systems challenges scalability by increasing traffic on memory paths and complicating cache/memory management. |
| Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs | Programmer responsibility for synchronization constructs that insure "correct" access of global memory. |

Distributed Memory: processors possess their own local memory without a global address space, each operates independently, with changes to local memory not affecting others, **no need for cache coherency**. Data exchange and synchronization across processors require explicit programming, placing the burden of managing inter-processor communication and task **synchronization programmer responsibility.**



Hybrid Distributed-Shared Memory:

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
- The shared memory component is usually a cache coherent SMP machine.
- The distributed memory component is the networking of multiple SMPs.



2- Message Passing:

- Allows for communication between a set of processors.
- Each processor is required to have a local memory**, no global memory is required**.
- Communication occurs between processors by **sending and receiving** messages.

Point-to-Point Communication: One process sends a message to another.

Different types of point-to-point communication:

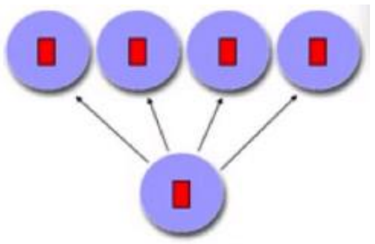
- **synchronous send:** the sender send a message and wait to the receiver to confirm that he receive it before send the next massage.
- **Asynchronous (buffered) send**: the sender sends a massages without wait the confirmation.

---

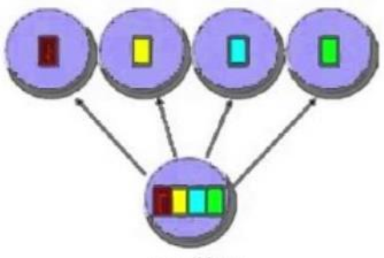
- **Blocking Operations:** This model waits for an operation to be completed before proceeding. For example, in a synchronous send operation, the process is blocked until the matching receive is issued, akin to a conversation where one person waits for the other to respond before continuing.
- **Non-Blocking Operations:** Operations return immediately, allowing other tasks to be performed concurrently. This can be likened to sending an email and moving on to other work without waiting for a reply.
- **Blocking Non-Buffered Send/Receive:** The sender waits for the receiver to be ready before sending data, similar to a phone call where the caller waits for the receiver to pick up.
- **Blocking Buffered Send/Receive:** The sender can proceed after placing data in a buffer, regardless of the receiver's readiness, akin to leaving a voicemail.
- **Non-Blocking Non-Buffered Send/Receive (Approach 1):** The sender can continue with other tasks, issuing a send request and proceeding without waiting for the receiver. This resembles sending a text message and continuing with your day, with the message being sent when the recipient is ready.
- **Non-Blocking Non-Buffered Send/Receive (Approach 2):** Like Approach 1, but the sender creates a child process for handling the send operation, allowing it to focus on other computations. This is like delegating a task to an assistant.
- **Non-Blocking Buffered Send/Receive:** The sender uses direct memory access to place data into a buffer, continuing with other tasks, while the receiver accesses the data from the buffer when ready. This can be compared to using a Dropbox where documents are left for someone to pick up at their convenience.

**Collective Communications**:

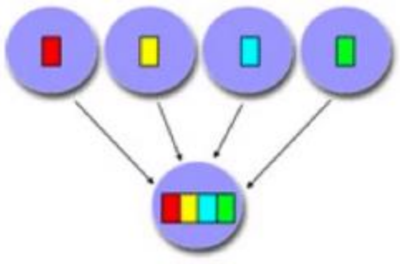
**Broadcast:** This function allows one process (called the root) to send the same data to all communicator members.


broadcast

**Scatter**: Allows one process to give out the content of its send buffer to all processes in a communicator. **(data parallelism)**

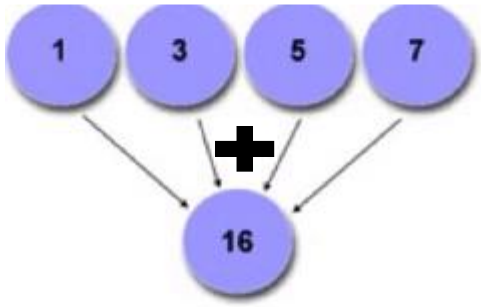

scatter

**Gather**: Each process gives out the data in its send buffer to the root process which stores them according to ranks.


gather

Reduction: Gather operation combined with a specified arithmetic or logical operation.

**example with + operation**



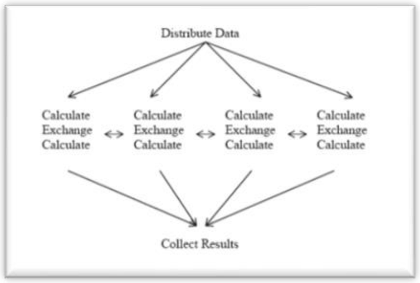
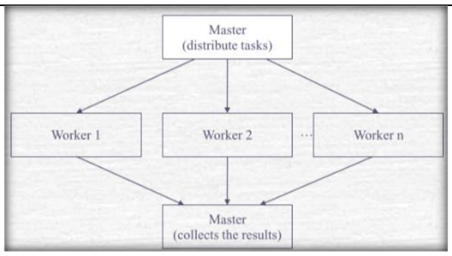
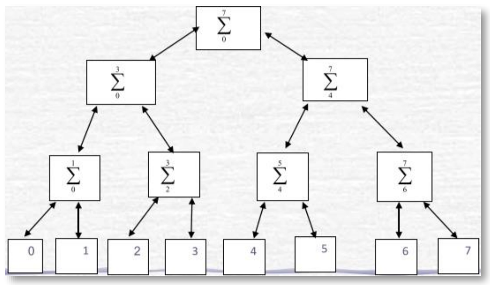
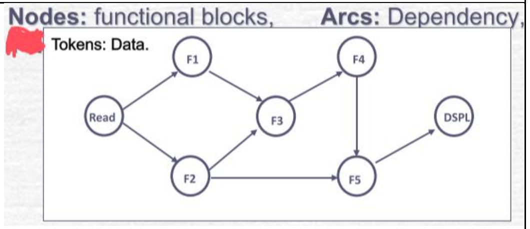
reduction

---

**3-** Multi-threading Paradigm

- In a single-core (superscalar) system, we can define multithreading as the ability of the processor's hardware to run two or more threads in an overlapping fashion by allowing them to share the functional units of that processor.
- in a multi-core system, we can define multithreading as the ability of two or more processors to run two or more threads simultaneously (in parallel) where each thread run on a separate processor.

## CH3:

| Parallel Programming Models | | Example |
|---|---|---|
| 1. SPMD | The SPMD model involves parallel nodes running the same program on different data sets, with coordination through self-scheduling and dynamic task assignment.<br>SIMD |  Distribute Data<br>Calculate Exchange Calculate<br>Collect Results |
| 2. Task Farming | Task farming involves a **master** and multiple **workers**: the master decomposes problems into tasks, distributes them among workers, and aggregates their results for the final outcome. Workers receive tasks, process them, and return the results to the master. |  Master (distribute tasks)<br>Worker 1<br>Worker 2<br>Worker n<br>Master (collects the results) |
| 3. Divide and conquer | The Divide and Conquer model address complex problems by breaking them into smaller, manageable sub-problems of the same nature, solving these recursively with the same algorithm until reaching a directly solvable base case. Sub-problems are often solved independently and concurrently, with their results merged to form the final solution. The model emphasizes recursive task execution on varied data sets. |  $\sum_0^7$<br>$\sum_0^3$<br>$\sum_4^7$<br>$\sum_0^1$<br>$\sum_2^3$<br>$\sum_4^5$<br>$\sum_6^7$<br>0 1 2 3 4 5 6 7 |
| 4. Dataflow | Dataflow programming partitions computation into distinct functional blocks that address different problem aspects, connected to denote dependencies and execution flow, facilitating parallelism expression. Data-pipelining, a subset of this model, represents computation as a directed graph, enhancing structured data processing. |  Nodes: functional blocks,<br>Arcs: Dependency,<br>Tokens: Data.<br>Read<br>F1<br>F2<br>F3<br>F4<br>F5<br>DSPL |

# CH4:

GPU: GPUs are highly parallel, many-core processors optimized for computer graphics and designed for executing hundreds of concurrent threads across many cores, achieving teraflops of peak performance, and specializing in **fine-grained data-parallel computation.**

## What is GPGPU?

**G**eneral **p**urpose computing on **GPU**s

**GPGPU** is the use of a GPU, which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (**CPU**).

## Why GPGPU?

– Massively parallel computing power

– Inexpensive

| GPGPU frameworks | | |
|---|---|---|
| CUDA | OpenCL | DirectCompute |

**CUDA:** **C**ompute **U**nified **D**evice **A**rchitecture.

- A parallel computing architecture developed by NVIDIA.
- Heterogeneous serial-parallel computing.
- The computing engine in GPU.
- CUDA gives developers access to the instruction set and memory of the parallel computation elements in GPUs.

Heterogeneous Computing(serial-parallel)

**Host** The CPU and its memory (host memory)

**Device** The GPU and its memory (device memory)

**Kernal:** code than run in Parallel.

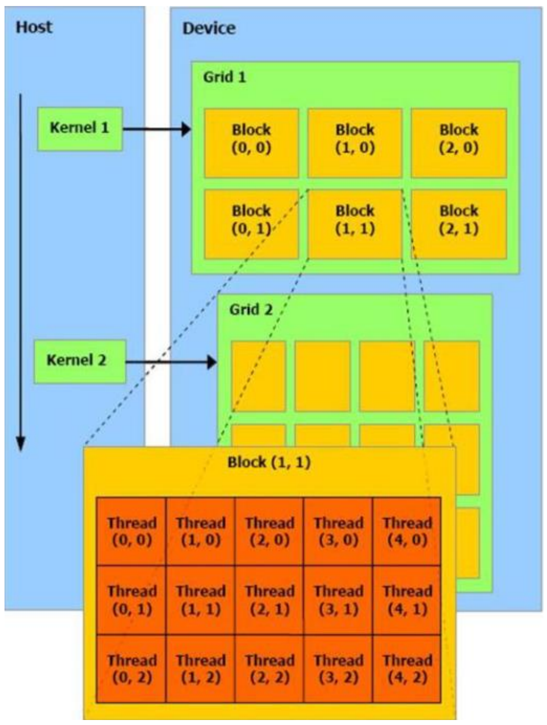
- One kernel is executed at a time.
- Many threads execute each kernel.

Differences between CUDA and CPU threads: (انشاء الثريد اسرع بكثير من المعالج)

A kernel is executed by a grid of thread blocks.

A thread block is a batch of threads that can cooperate with each other by:

- Sharing data through shared memory
- Synchronizing their execution



**Threads from different blocks cannot cooperate.**

**All threads within a block can**

– Share data through 'Shared Memory'

– Synchronize using '_syncthreads()' (نقطة تجمع للثريدز في نفس البلوك)

**• Threads and Blocks have unique IDs**

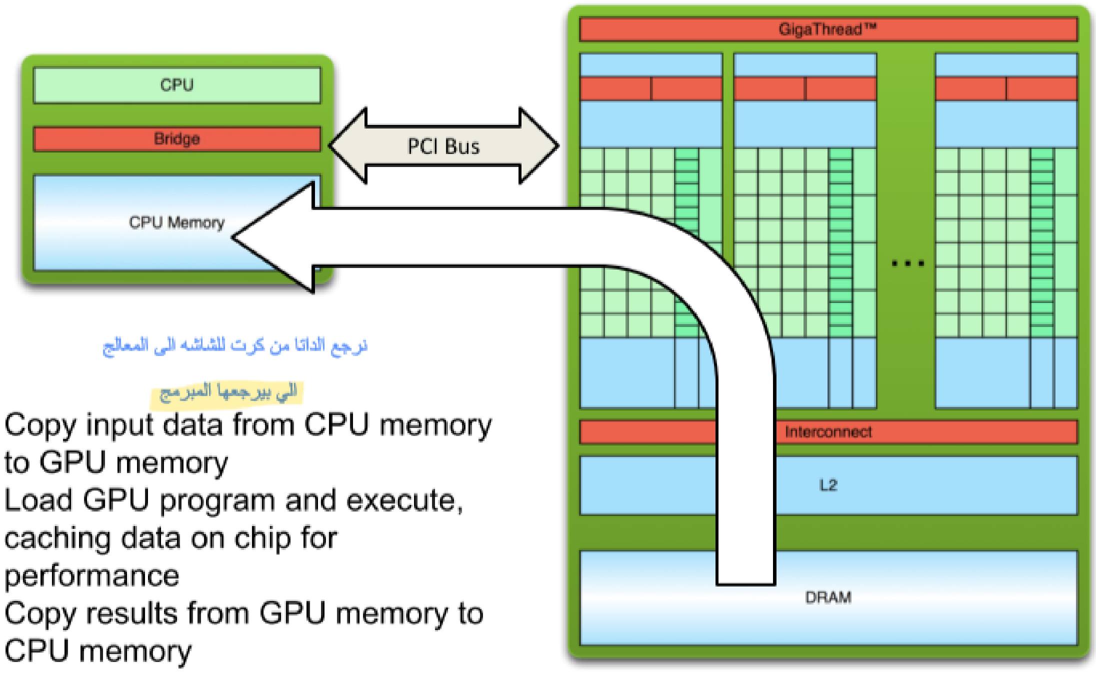
**SIMT (Single Instruction Multiple Threads) Execution**

Threads run in groups of 32 called **warps**.

Every thread in a warp executes **the same instruction** at a time.

**Number of warps = threads / 32(size of warp)**

## Simple Processing Flow



نرجع الداتا من كرت للشاشه الى المعالج

الي بيرجعها المبرمج

1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

| **Types of GPU (device) memory** | |
|---|---|
| **1- Registers – read/write per-thread**<br>• on chip<br>• fast access<br>• per thread<br>• limited amount<br>• 32 bit | **4- Global Memory – read/write across grids**<br>• in DRAM<br>• slow<br>• non-cached (لتجنب الاوفرهيد بسبب مشاكل الكونستانسي)<br>• per grid<br>• communicate between grids |
| **2- Local Memory – read/write per-thread**<br>• **Local Memory**<br>• **in DRAM**<br>• **slow**<br>• **non-cached**<br>• **per thread**<br>• **relative large** | **5- Constant Memory – read across grids**<br>• in DRAM<br>• cached ( لانه مافيه اوفرهيد الكونستانسي)<br>• per grid<br>• read-only |
| **3- Shared Memory – read/write per-block**<br>• **on chip**<br>• **fast access**<br>• **per block**<br>• **16 KByte**<br>• **synchronize between threads** | **6- Texture Memory – read across grids**<br>• in DRAM<br>• cached<br>• per grid<br>• read-only<br>الفرق بينها وبين الكونستانت ميموري ان ها الي راح يحدد لك البربشت |

Global Memory • Constant Memory • Texture Memory **are managed by host code, persistent across kernels.**