**Parallel computing**

# 1 并行计算历史回顾（[Parallel Computing Overview](https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial" \l "#Overview)）

## 什么是并行计算

1. 通常情况下软件是采用serial computing的，串行计算有如下的几个问题

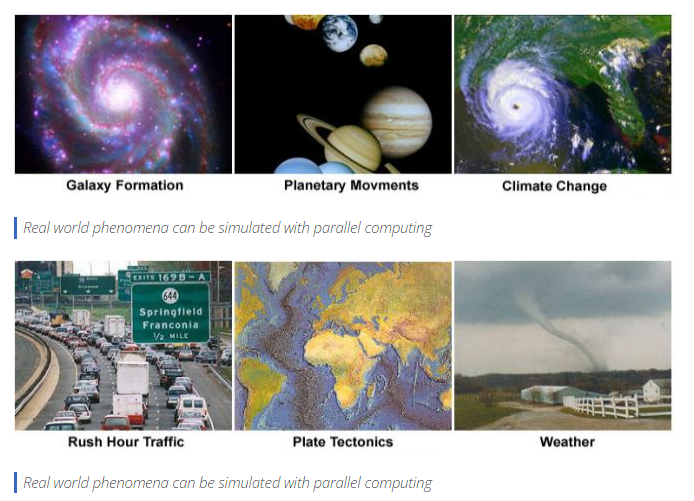
* A problem is broken into a discrete series of instructions
* Instructions are executed sequentially one after another
* Executed on a single processor
* Only one instruction may execute at any moment in time
  1. 被分解成一些列分解的指令（A problem is broken into a discrete series of instructions）
  2. 指令是一个接一个的执行，执行完一个再执行另一个
  3. 只能在单个进程上执行
  4. 同一时间只有单个指令在执行

1. 并行计算有下边的优点

* A problem is broken into discrete parts that can be solved concurrently
* Each part is further broken down to a series of instructions
* Instructions from each part execute simultaneously on different processors
* An overall control/coordination mechanism is employed

## 1.2 为什么要用并行计算

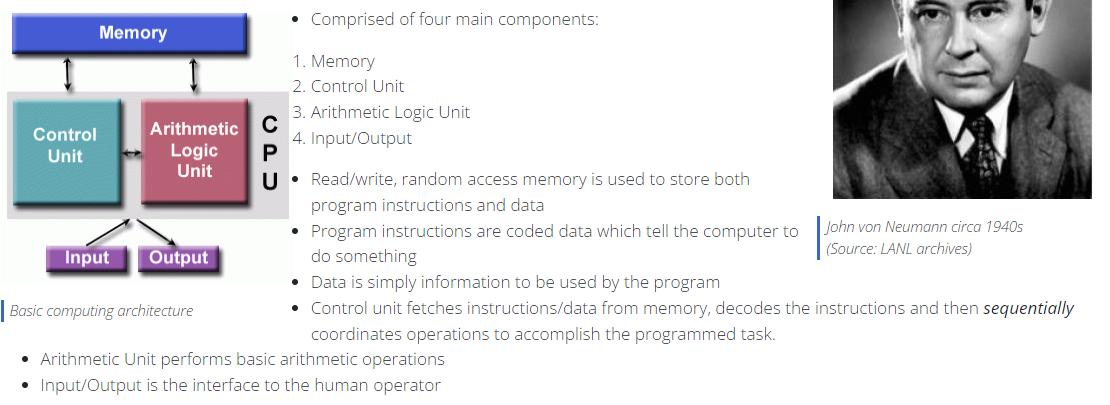
1. 真实世界如此复杂 The Real World Is Massively Complex
   1. 自然世界瞬息间同时发生很多实践。推及到计算机，采用parallel computing来建模这个真实世界



1. 使用并行编程的原因
   1. 省时间和金钱
   2. 解决更大更多更复杂的问题
   3. 解决这种petaflops （每秒千万亿次浮点运算）的问题
   4. 提供并发性：PROVIDE CONCURRENCY
   5. TAKE ADVANTAGE OF NON-LOCAL RESOURCES
   6. 更好地利用底层并行硬件MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE
2. 谁在使用并行计算？

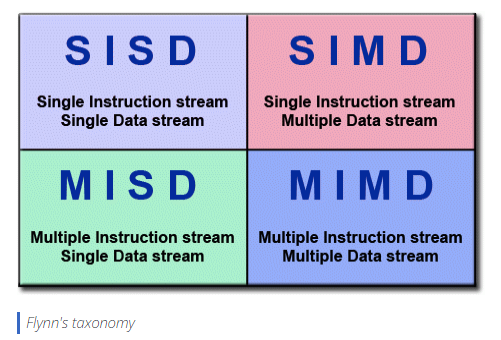
# 2 概念和术语 Concepts and Terminology

## 2.1 冯诺依曼计算机架构von Neumann Computer Architecture



## 2.2 弗林的古典分类学 Flynn's Classical Taxonomy

* Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of**Instruction Stream** and ****Data Stream****. Each of these dimensions can have only one of two possible states: ****Single**** or ****Multiple****.
* The matrix below defines the 4 possible classifications according to Flynn:、
* **单指令单数据、单指令多数据、多指令单数据、多指令多数据**



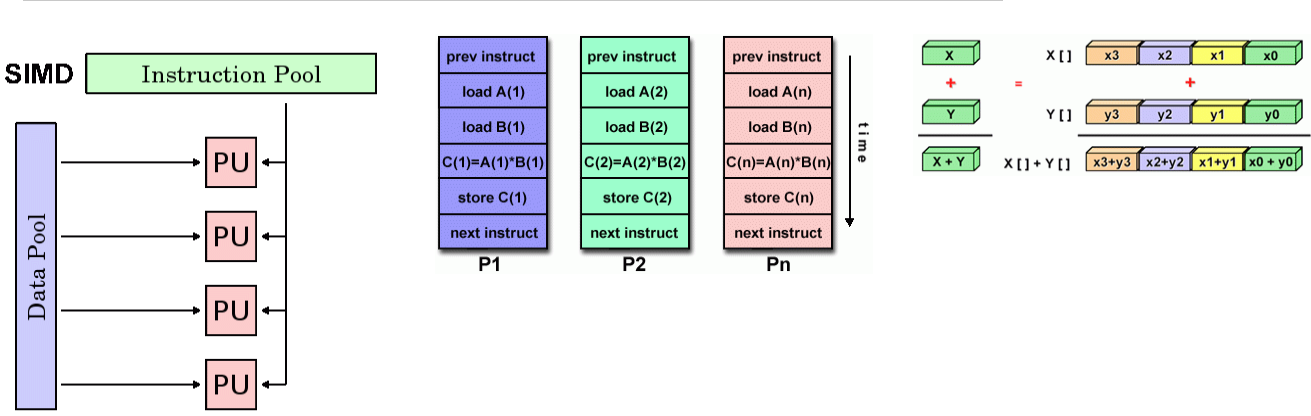
### Single Instruction, Single Data (SISD)

* A serial (non-parallel) computer
* ****Single Instruction:**** Only one instruction stream is being acted on by the CPU during any one clock cycle
* ****Single Data:**** Only one data stream is being used as input during any one clock cycle
* Deterministic execution
* This is the oldest type of computer
* Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.



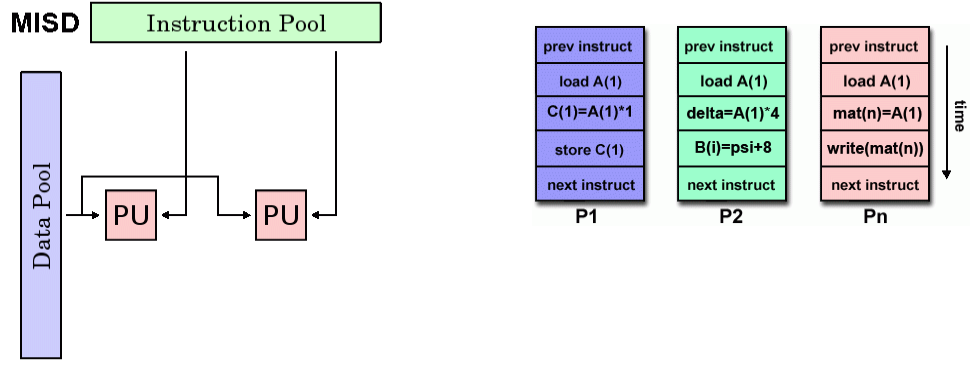
### Single Instruction, Multiple Data (SIMD)

* A type of parallel computer
* ****Single Instruction:**** All processing units execute the same instruction at any given clock cycle
* ****Multiple Data:**** Each processing unit can operate on a different data element
* Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
* Synchronous (lockstep) and deterministic execution
* Two varieties: Processor Arrays and Vector Pipelines
* Examples:
  + Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
  + Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
* Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.



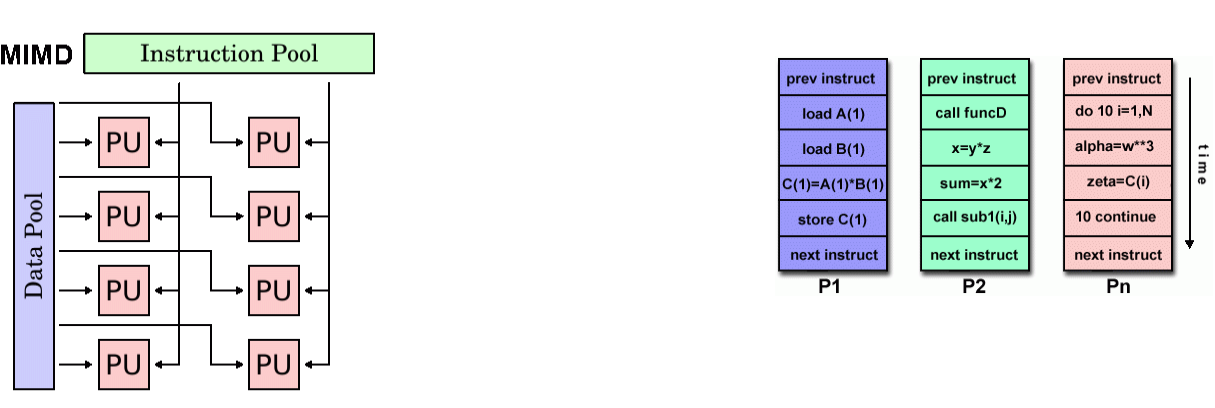
### Multiple Instruction, Single Data (MISD)

* A type of parallel computer
* ****Multiple Instruction:**** Each processing unit operates on the data independently via separate instruction streams.
* ****Single Data:**** A single data stream is fed into multiple processing units.
* Few (if any) actual examples of this class of parallel computer have ever existed.
* Some conceivable uses might be:
  + multiple frequency filters operating on a single signal stream
  + multiple cryptography algorithms attempting to crack a single coded message.



### Multiple Instruction, Multiple Data (MIMD)

* A type of parallel computer
* ****Multiple Instruction:****Every processor may be executing a different instruction stream
* ****Multiple Data:**** Every processor may be working with a different data stream
* Execution can be synchronous or asynchronous, deterministic or non-deterministic
* Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
* Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
* ****Note**** Many MIMD architectures also include SIMD execution sub-components



## 2.3 通用并行计算术语General Parallel Computing Terminology

#### **Cpu**：硬件

* 1. Cpu由一个或者多个核心（一个拥有自己指令流的独立执行单元）组成
  2. 路属于同一个cpu的cores（核心）可能被组织成一个或多个socket（每个socket拥有自己distinct memory）
  3. 一个CPU有两个以上的socket时，硬件就会supports memory sharing across sockets.

#### Node：硬件

#### Task：理解为一个程序指令流

* 1. . A task is typically a program or program-like set of instructions that is executed by a processor. A parallel program consists of multiple tasks running on multiple processors.

#### Pipelining：将一个任务切开分配到几个不同的处理器上去，就像一个分配线

#### Shared memory： 共享内存（一种计算机架构---具体还不理解）

#### Symmetric Multi-Processor (SMP)硬件

#### Distributed Memory ：分布式内存

#### Communications：

* 1. 并行任务需要交换数据（方式两种）：
     1. Shared memory bus
     2. network

#### Synchronization

## 2.4 Potential Beneíts, Limits and Costs of Parallel Programming

下边的术语应该是描述并行计算优略势的 定义

1. 是的

## 2.5 进程和线程

进程和线程是操作系统中进行任务调度和管理的基本单位。它们之间的区别如下：

定义：进程是一个正在执行中的程序实例，它具有独立的内存空间和系统资源。线程是进程中的一个执行单元，一个进程可以包含多个线程，它们共享进程的内存空间和系统资源。

内存关系：每个进程都有自己独立的内存空间，包括代码、数据、堆栈等。进程之间的内存是相互隔离的，一个进程无法直接访问另一个进程的内存。而线程是共享进程的内存空间的，它们可以直接访问和修改相同的内存地址。

资源占用：每个进程都有自己的资源副本，包括文件描述符、网络连接、打开的文件等。而线程共享进程的资源，它们可以直接访问进程的资源，不需要额外的资源副本。

创建和销毁：创建和销毁进程需要较大的系统开销，包括分配独立的内存空间和资源。而线程的创建和销毁比较轻量级，只需要为线程分配堆栈空间和保存线程上下文的数据结构。

切换开销：在多任务环境下，操作系统需要对进程和线程进行切换。进程切换需要保存和恢复整个进程的上下文信息，涉及到内存切换和TLB（转换后备缓冲器）刷新等操作，开销较大。而线程切换只需要保存和恢复线程的上下文信息，开销相对较小。

并发性：由于线程共享进程的内存空间，线程之间的通信和同步比较方便，可以通过共享内存进行数据交换和共享变量进行同步。而进程之间的通信和同步需要通过进程间通信（IPC）机制，如管道、消息队列、共享内存等。

总的来说，进程和线程的区别主要体现在内存关系上。进程具有独立的内存空间和资源，线程共享进程的内存空间和资源。这使得线程的创建和销毁更轻量级，线程之间的通信和同步更方便，但也增加了线程安全性的挑战。同时，进程切换的开销较大，线程切换的开销较小。

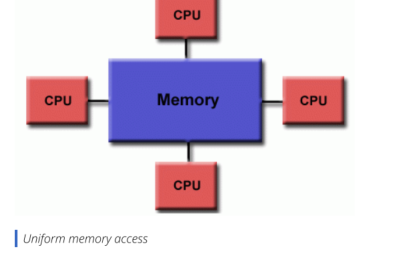
# 3 并行计算机内存体系

## 3.1 共享内存share memory

<https://blog.51cto.com/liangchaoxi/5244860> 这个网址讲了共享内存的计中设计架构

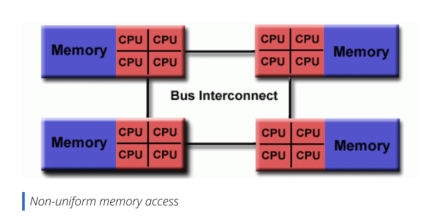
#### Uniform Memory Access (UMA)

* 1. 典型的及其类型就是 Symmetric Multiprocessor (SMP)
  2. 均匀内存访问架构则是与NUMA相反，所以处理器对共享内存的访问距离和时间是相同的。由此可知，不论是NUMA还是UMA都是SMP架构的一种设计和实现上的选择



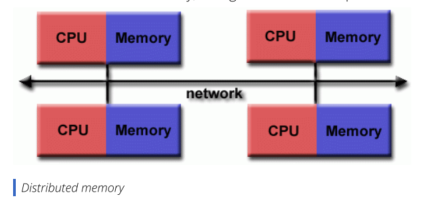
#### Non-Uniform Memory Access (NUMA)

1. 非均匀内存访问架构是指多处理器系统中，内存的访问时间是依赖于处理器和内存之间的相对位置的。 这种设计里存在和处理器相对近的内存，通常被称作本地内存；还有和处理器相对远的内存， 通常被称为非本地内存。



### 3.2 分布式内存 Distributed Memory

<https://www.cnblogs.com/focusonoutput/p/12439752.html> 分布式内存和共享内存的区别



### 3.3 混合分布式共享内存 Hybrid Distributed-Shared Memory

Hybrid Distributed-Shared Memory（混合分布式共享内存）是一种计算机系统架构概念，旨在通过将分布式系统的内存和共享内存系统结合起来，实现更高效的内存访问和数据共享。

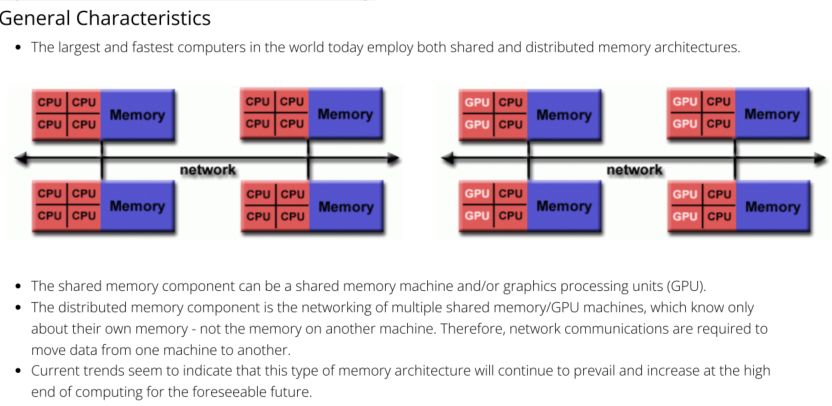
传统上，分布式系统使用消息传递的方式进行通信和数据共享。这种方式需要显式地将数据从一个节点发送到另一个节点，导致通信开销和复杂性增加。而共享内存系统使用一块物理内存（或虚拟内存）作为多个处理单元之间共享的共享内存空间，进程可以直接访问和操作这块内存空间，使得数据共享更简单、高效。

Hybrid Distributed-Shared Memory 的概念就是将分布式系统和共享内存系统结合起来，通过将分布式系统节点上的一部分内存作为共享内存空间，允许节点之间的进程直接访问共享内存，以降低数据传递的网络通信开销，提高数据共享和访问的效率。

这种混合的架构可以提供更高的灵活性和可扩展性。不同的节点之间可以通过消息传递进行通信，适用于需要处理大量分布式数据的任务。而对于一些需要频繁访问和共享数据的任务，可以使用共享内存空间来减少通信开销，提高性能。

需要注意的是，Hybrid Distributed-Shared Memory 并不是一个具体的系统或技术，而是一个概念性的框架。具体实现可以根据系统的需求和约束进行设计和开发。在实际中，可以使用分布式共享内存技术（如 Apache Ignite、Hazelcast 等）或具备共享内存特性的分布式计算框架（如 Apache Spark）来实现 Hybrid Distributed-Shared Memory 架构。

综上所述，Hybrid Distributed-Shared Memory 是一种将分布式系统和共享内存系统结合起来的架构概念，通过提供直接访问共享内存的方式，实现更高效的内存访问和数据共享。它可以在需要高性能数据共享的场景下提供更好的效果，并通过结合分布式系统的优势，提供更高的灵活性和可扩展性。



# 4 并行编程/编码的模式

There are several parallel programming models in common use:

* Shared Memory (without threads)
* Threads
* Distributed Memory / Message Passing
* Data Parallel
* Hybrid
* Single Program Multiple Data (SPMD)
* Multiple Program Multiple Data (MPMD)

**Parallel programming models exist as an abstraction above hardware and memory architectures.** 并行编程模式是一个在硬件和内存架构之上的抽象

## SHARED memory model on a DISTRIBUTED memory machine

在分布式内存机器上的共享内存编程模式

在分布式内存机器上的共享内存编程模式是一种编程模式，用于在分布式系统中实现共享内存的功能。它允许多个节点上的进程直接访问和操作共享的内存空间，从而实现高效的数据共享和通信。

在传统的共享内存系统中，多个进程可以通过共享的内存空间来进行数据的读写和通信。而在分布式系统中，每个节点通常拥有自己的独立内存空间，无法直接访问其他节点的内存。因此，共享内存编程模式在分布式系统中需要特殊的机制和技术来实现。以下是在分布式内存机器上的共享内存编程模式的一些常见技术和方法：

1. 数据分区：将共享数据分割成多个部分，并将其分配到不同的节点上。每个节点可以直接访问其分配的数据部分。这样可以将数据分散存储在多个节点上，实现并行处理和分布式共享。

2. 一致性协议：通过一致性协议来保证分布式系统中的共享数据的一致性。常见的一致性协议包括基于锁的协议（如分布式锁）和基于版本的协议（如MVCC）等。

3. 远程访问和远程过程调用（RPC）：使用远程访问和RPC机制，允许进程在一个节点上访问和操作其他节点上的数据。这样，节点之间可以通过远程调用来实现数据的读写和通信。

4. 分布式共享内存库或框架：使用特定的分布式共享内存库或框架，如OpenMPI、Apache Ignite、Hazelcast等，提供共享内存功能的封装和抽象，简化共享内存编程的实现。

需要注意的是，在分布式系统中使用共享内存编程模式必须谨慎，因为它需要解决一致性、可扩展性和并发控制等问题。同时，共享内存编程模式不适用于所有分布式应用场景，对于不需要频繁数据访问和共享的任务，使用消息传递的方式可能更为适合。

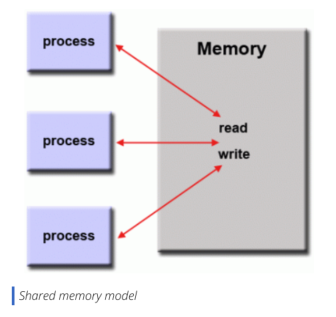
综上所述，在分布式内存机器上的共享内存编程模式是通过特定的技术和方法，使多个节点上的进程能够直接访问和操作共享的内存空间，从而实现高效的数据共享和通信。

## DISTRIBUTED memory model on a SHARED memory machine

在共享内存机器上的分布式内存编程模型

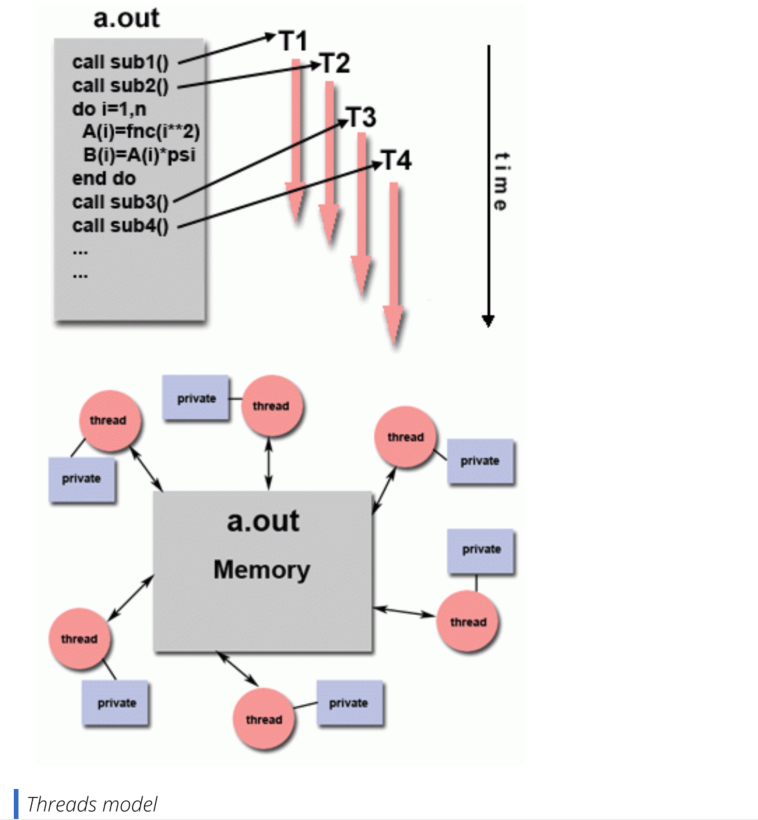
## Shared Memory Model (without threads)

多个进程使用相同的内存 肯定要用锁那些机制



## Threads Model

多线程并行编程模式，就是常规意义上的多线程跑一个程序



多线程编程模式有5中编程模式，文档介绍了两种

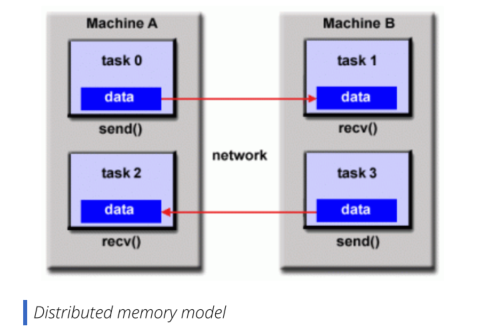
5钟参考链接 <https://zhuanlan.zhihu.com/p/626728437>

### POSIX Threads

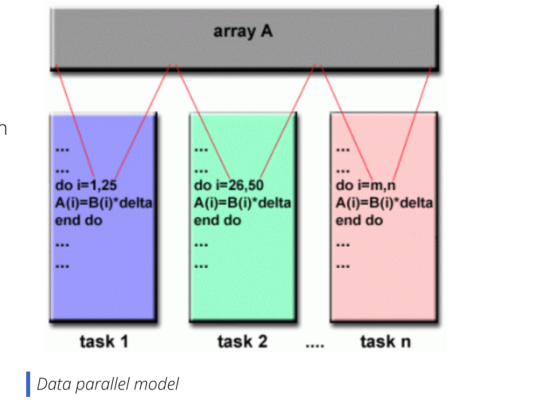
### OpenMP

## Distributed Memory / Message Passing Model

分布式内存/信息传递并行编程模式

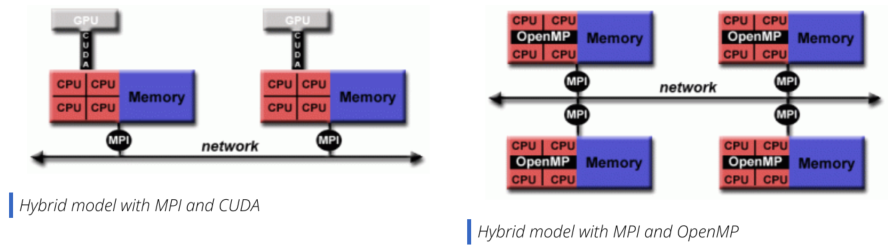


## Parallel Model

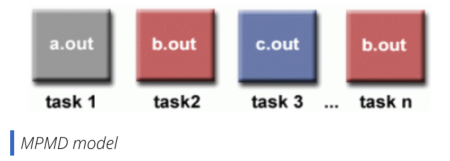
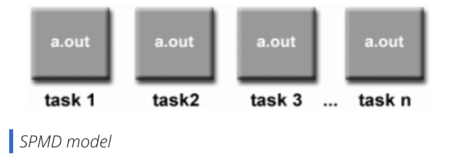


## Hybrid Model

Message Passing Interface (MPI)



## SPMD and MPMD



# 5 并行程序设计 Designing Parallel Programs

## 5.1 Automatic vs. Manual Parallelization

## 5.2 Understand the Problem and the Program

啥叫程序 **Programs = algorithms + data + (hardware)**

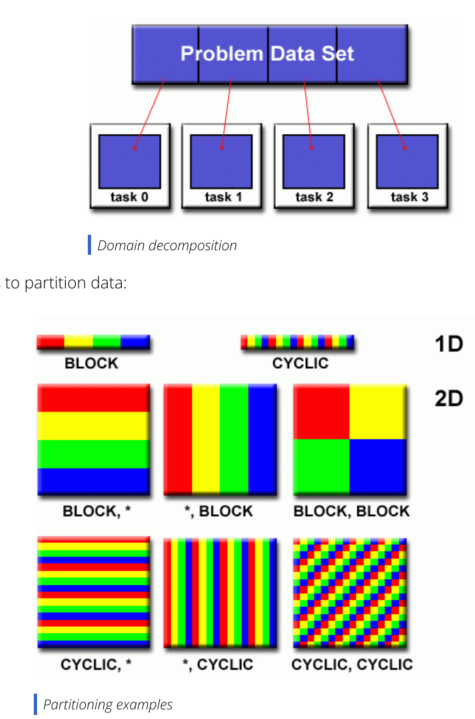
本节讲了，啥样的问题可以被并行处理：问题可以被并行处理才会去做并行程序设计否则不瞎扯淡。

## 5.3 Partitioning 任务分解2种形式

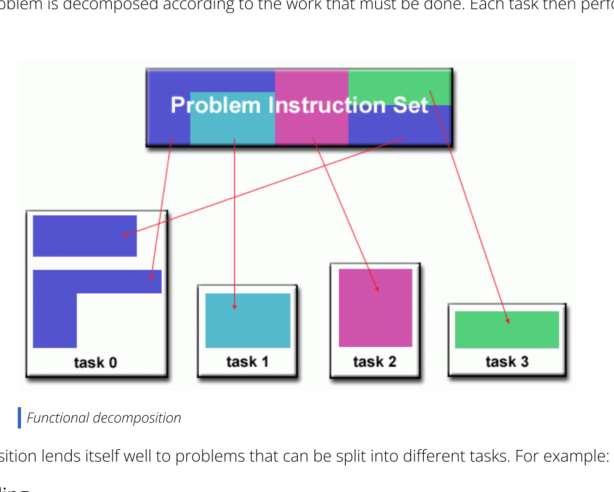
讲述：如何分解问题/功能/模块的到最小并行单元

一共两种：数据分解 和 功能分解

### 5.1.1 domain decomposition



### 5.1.2 functional decomposition



1. 举了俩例子：生态系统模型和信号处理Ecosystem Modeling、Signal Processing

## 5.4 Communications (data change)

理解为数据传输

Who Needs Communications?

The need for communications between tasks depends upon your problem:

### 哪些/哪种任务需要做communication

### 哪些/哪种任务不需要做communication

### 做communication时的一些需要考虑到的factor

* 1. **通讯开销 Communication overhead（任务等待，网络带宽占用等**）
  2. 延迟和带宽 Latency vs. Bandwidth
  3. 通讯可见性 Visibility of communications
     1. 程序员需要对这个通讯了如指掌？
  4. 同步vs 异步通讯 Synchronous vs. asynchronous communications

## 5.5 Synchronization 同步（数据/状态同步）

1. 是指协调并行计算中不同线程或进程之间的执行顺序和互动的机制。它用于确保多个并行执行的任务在某个特定点上达到一致状态，以避免数据竞争和不确定性的结果。
2. 常见实现同步的几个方式
   1. Barrier： 所有子任务都堵在一堵墙前边，最后一个完成后会进行同步
   2. Lock： 单个子任务访问global内容时上个锁 确保使用过程中不被更改
   3. Synchronous communication operations：相互通讯的子任务之间要保持同步

## 5.6 data dependencies （数据依赖问题）

1. Data Dependencies（数据依赖）指的是任务之间的数据关系或依赖关系。当一个任务需要依赖另一个任务的输出数据时，就存在数据依赖。
2. How to Handle Data Dependencies
   1. Distributed memory architectures - communicate required data at synchronization points. 分布式内存结构：在需要communicate的点进行数据同步
   2. Shared memory architectures -synchronize read/write operations between tasks
      1. 通过读写操作同步使用内存（例如锁住）

## 5.7 Granularity（任务划分细粒度问题）

1. Granularity（粒度）是指将任务划分为多个子任务的程度或大小。它决定了任务之间的相互依赖程度以及并行计算的效率。 Computation / Communication Ratio
2. 划分：
   1. 粗细力度
   2. 精细粒度

## 5.7 Debugging

1. Debug并行代码：

# 6 并行示例 parallel example

## 6.1 示例描述和对这个示例的并行思考

