


# 课程信息

 教材：以课堂讲义为主

 主要参考资料：

- 《并行计算导论》，张林波等，清华大学出版社，2006
- 课件及作业发布通过课程公共邮箱，Internet
- njumath\_bxjs@163.com，密码njumath2010
- 提交作业发送至：xym@nju.edu.cn

 上课时间：周三下午 18:30 开始，综合实验楼丙区504



# 第一部分：

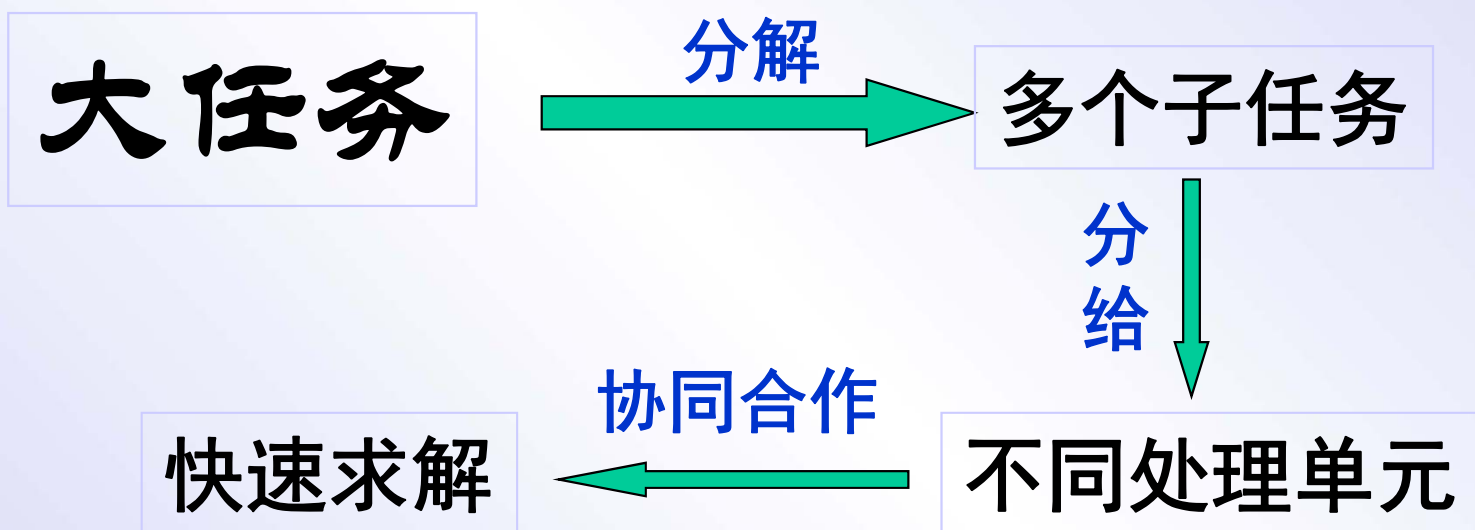
## 并行计算介绍与

## 程序开发环境



# 并行计算介绍

- 并行计算（高性能计算、超级计算）



- 基本条件：  
硬件（并行机）、并行算法设计、并行编程环境
- 主要目标： 提高求解速度，扩大问题规模



# 并行计算的应用需求

- 数值天气预报

全球气象中期天气预报要求在 24 小时内完成 48 小时天气预测数值模拟，至少需 635 万个网格点，内存需求大于 1 T，计算性能高达 25 万亿次/秒

- 核武器数值模拟

美国1996年实施的ASCI计划，分四个阶段实现万亿次、十亿亿次、30万亿次和100万亿次大规模并行数值模拟，实现全三维、全物理过程、高分辨率的核武器数值模拟

- 天体物理、航天器设计、油藏模拟、地震数据处理、密码破译、新药研制、生物信息处理、图像处理、.....

并行计算能力已成为衡量一个  
国家综合实力的重要标志之一！

# 并行计算研究内容

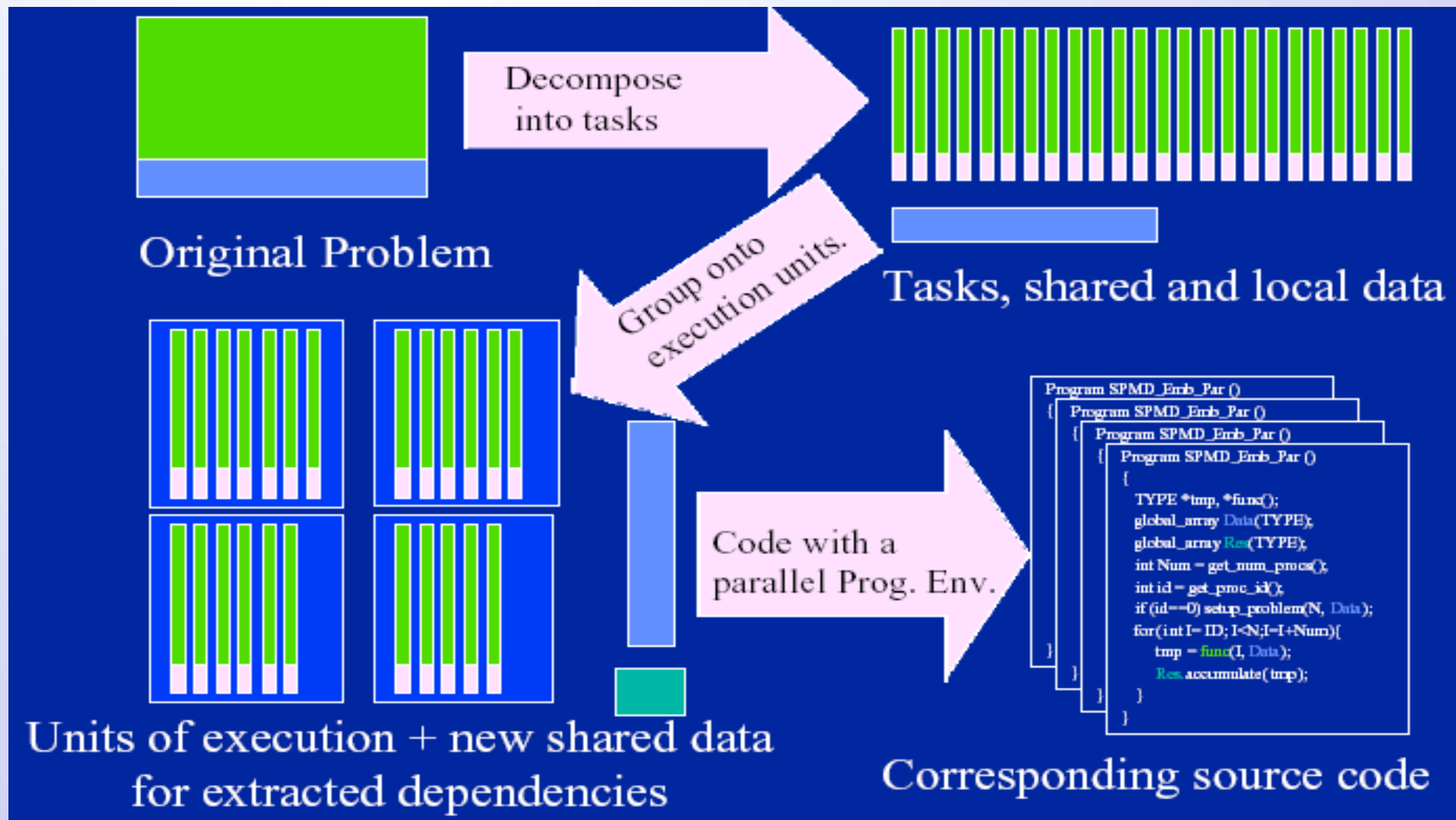
## □ 研究内容

- 并行计算机的体系结构
- 并行算法设计与分析
- 并行实现技术：编程实现，优化性能
- 并行应用：开发并行应用软件





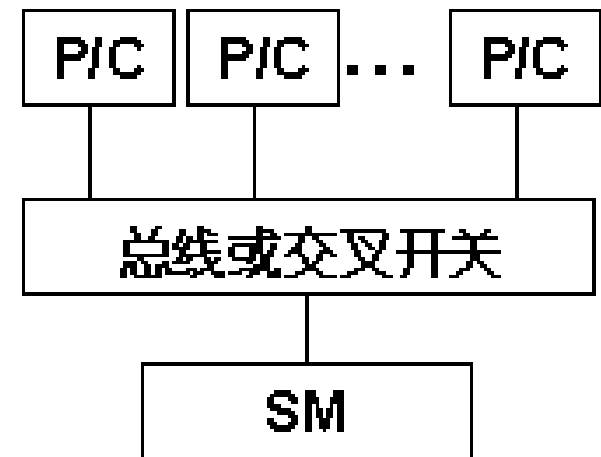
# 如何实现并行计算? 分而治之!



# 共享存储对称多处理机系统 (SMP)

- SMP:

- 对称式共享存储：任意处理器可直接访问任意内存地址,且访问延迟、带宽、几率都是等价的；系统是对称的；
- 微处理器：一般少于64个；
- 处理器不能太多，总线和交叉开关的一旦作成难于扩展；
- 例子：IBM R50, SGI Power Challenge, SUN Enterprise, 曙光一号



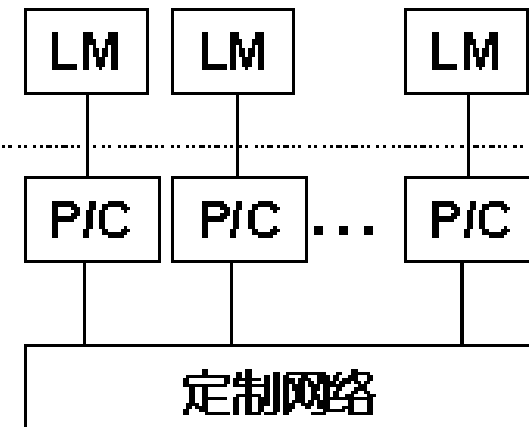
(b) SMP, 物理上单一地址空间

# 分布共享存储多处理机系统 (DSM)

- DSM:

- 分布共享存储: 内存模块物理上局部于各个处理器内部,但逻辑上(用户)是共享存储的; 这种结构也称为基于Cache目录的非一致内存访问(CC-NUMA)结构; 局部与远程内存访问的延迟和带宽不一致,3-10倍→高性能并行程序设计注意;
- 与SMP的主要区别: **DSM**在物理上由分布在各个节点的局部内存从而形成一个共享的存储器;
- 微处理器: 16-128个,几百到千亿次;
- 代表: SGI Origin 2000, Cray T3D;

虚拟分布共享存储(DSM)



(d) DSM, 逻辑上单一地址空间

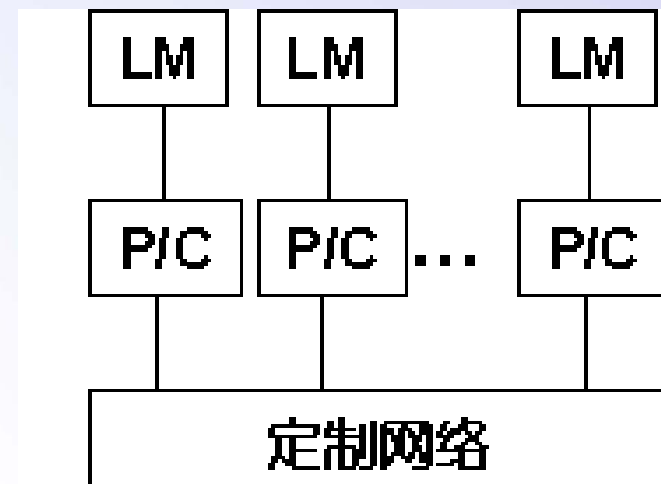




# 大规模并行计算机系统 (MPP)

- **MPP:**

- 物理和逻辑上均是分布内存
- 能扩展至成百上千个处理器(微处理器或向量处理器)
- 采用高通信带宽和低延迟的互连网络(专门设计和定制的)
- 一种异步的**MIMD**机器; 程序系由多个进程组成, 每个都有其私有地址空间, 进程间采用传递消息相互作用;
- 代 表 :**CRAY T3E(2048), ASCI Red(3072), IBM SP2, 曙光1000;**



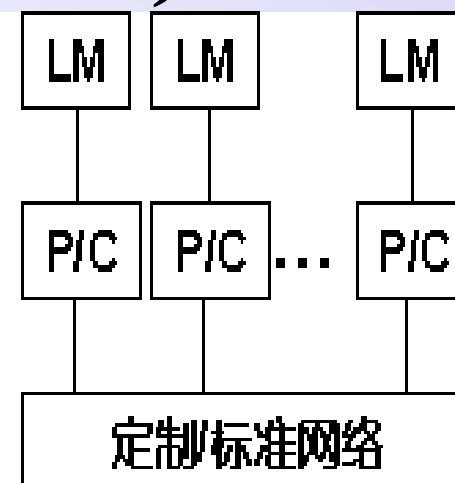
(c) MPP, 物理/逻辑上多地址空间



# 机群系统(Cluster)

- Cluster:

- 每个节点都是一个完整的计算机
- 各个节点通过高性能网络相互连接
- 网络接口和I/O总线松耦合连接
- 每个节点有完整的操作系统
- 曙光 2000, 3000, ASCI Blue Mountain (48 台 128-way DSM Origin 2000, 6144 个处理器)



(e) Cluster/COW, 物理/逻辑上多地址空间



# 并行计算机的发展

## □ 并行计算机发展趋势

- 由于**向量机**和 **MPP** 受研制费用高、售价高等因素的影响，其市场受到一定的限制
- **SMP** 由于共享结构的限制，系统的规模不可能很大
- 由于**机群**系统具有投资风险小、可扩展性好、可继承现有软硬件资源和开发周期短、可编程性好等特点，目前已成为并行处理的热点和主流



# 集群（机群）——Cluster

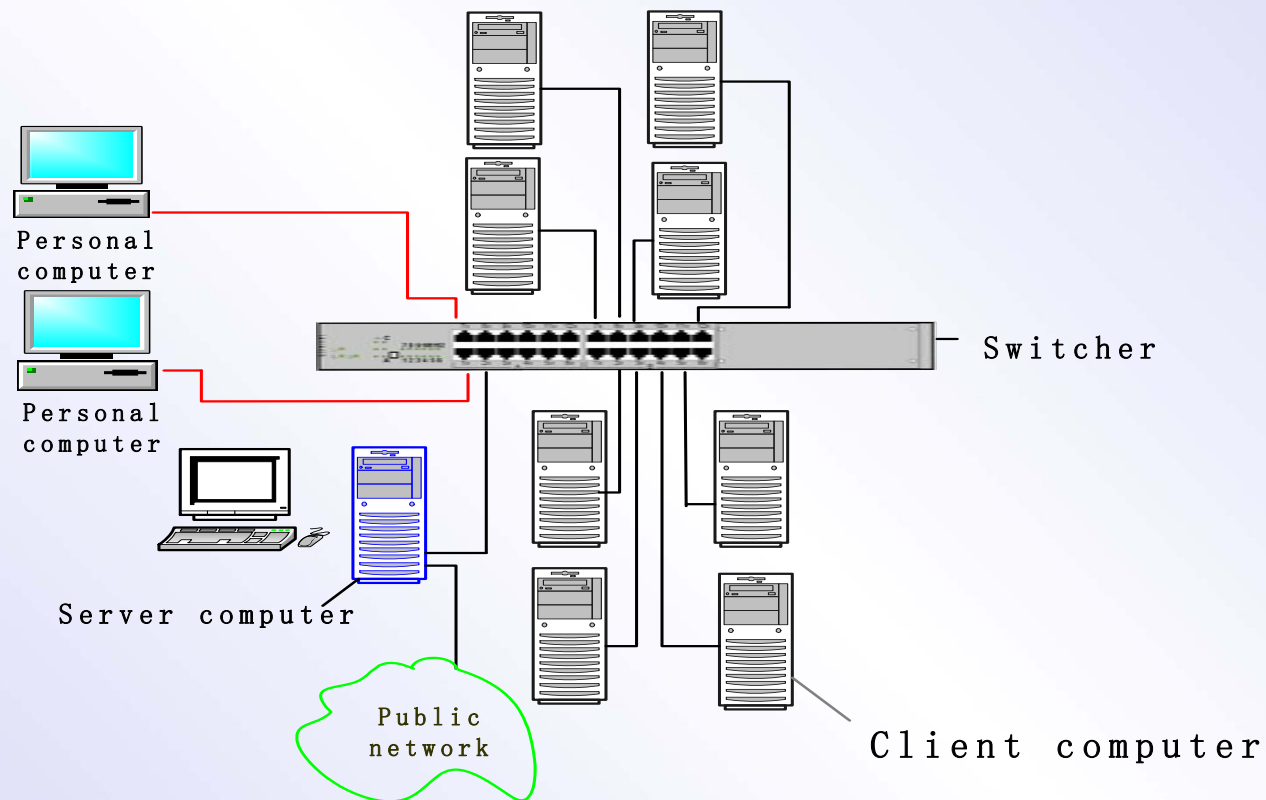
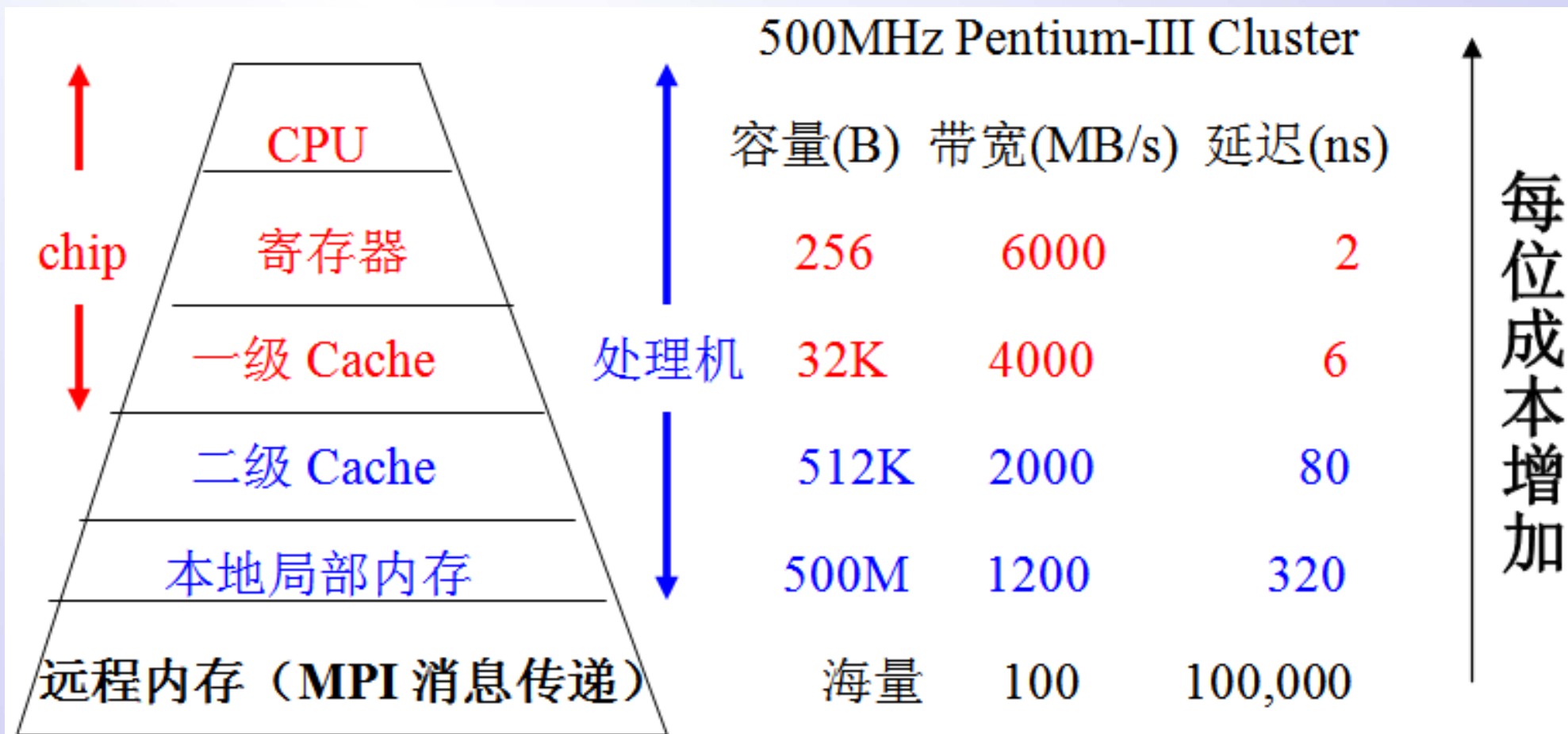


Fig.1 Diagram of cluster structure





# 微处理器的多级存储结构



微机机群的一次消息传递延迟相当于 **50000** 次峰值浮点运算



# 微处理器的多级存储结构

- 微处理器主频越来越高，内存容量越来越大，  
但内存访问速度的增长较慢
- 缓解内存墙性能瓶颈：Cache 高速缓存
- Cache 工作原理：略  
(参见《并行计算导论》)



# GPU 架构概览 《CPU编程与优化》

- GPU 特别适用于
  - 密集计算，高度可并行计算
  - 图形学
- 晶体管主要被用于：
  - 执行计算
  - 而不是：
    - 缓存数据
    - 控制指令流



# GPU 架构概览

## 晶体管用途



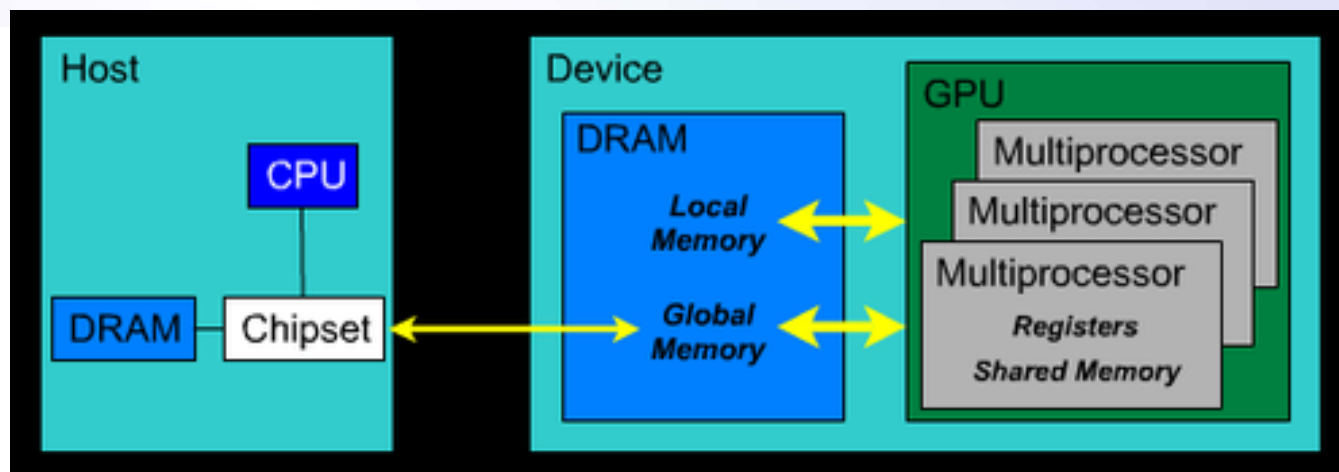
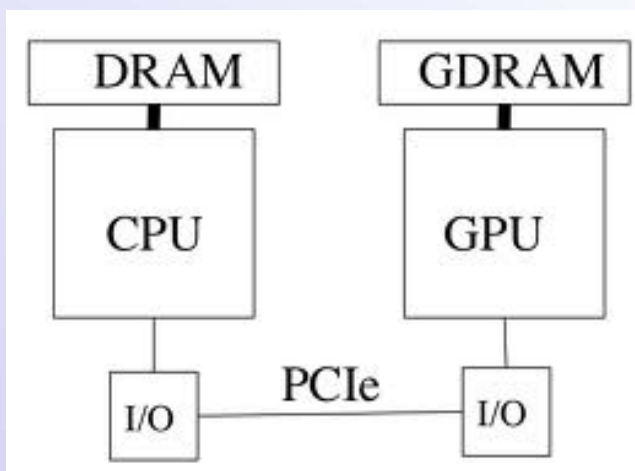
Figure 1-2. The GPU Devotes More Transistors to Data Processing





# CPU-GPU交互

- 各自的物理内存空间
- 通过PCIE总线互连(8GB/s~16GB/s)
- 交互开销较大



© NVIDIA Corporation

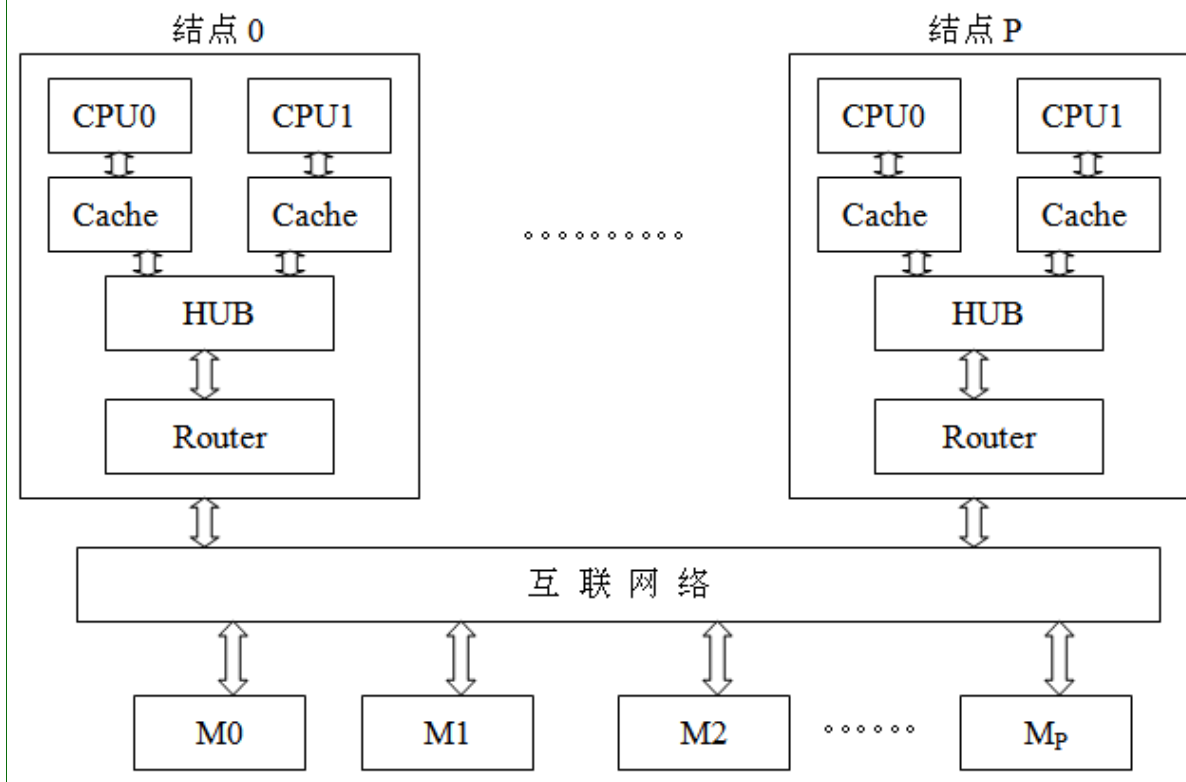


# 并行计算机体系结构

## □ 组成要素

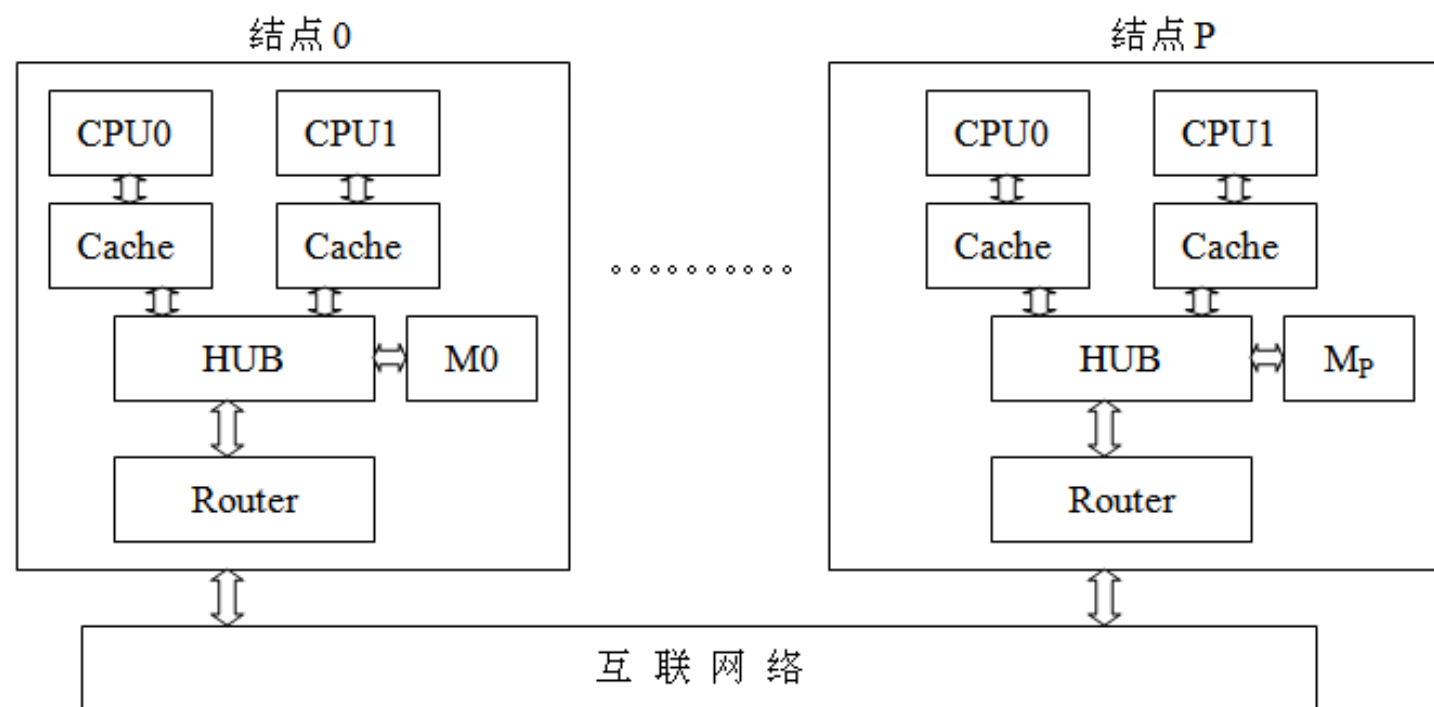
- 结点 (node) : 一个或多个处理器组成
- 互联网络 (interconnect network) : 连接结点
- 内存 (memory) : 多个存储模块组成





- 并行计算机体系  
结构示意图  
内存模块与结点分离

并行计算机体系  
结构示意图  
内存模块位于  
结点内部



# 并行计算机体系结构

- 并行计算机互联网络拓扑结构：略  
（参见《并行计算导论》）
- 并行计算机内存访问模式：略  
（参见《并行计算导论》）





# 操作系统与并行编程环境

UNIX

GNU C/C++, GNU FORTRAN

Linux——Ubuntu 16.04 LTS

<http://www.ubuntu.com/>



## (一) 进程、进程间通信与线程

消息传递或者共享存储并行编程所必须了解的基本概念: 进程和线程.

### 1. 进程 (Process)

进程可表示成四元组(P, C, D, S)

- P 是程序代码
- C 是进程的控制状态
- D 是进程的数据
- S 是进程的执行状态



任何进程总和程序联系在一起，程序一旦在具体操作系统环境中投入运行，就变成了进程。

各个进程拥有独立的执行环境，其中包括内存数据和指令地址空间、程序计数器、寄存器、栈空间、文件系统、I/O设备等，并在操作系统的控制、管理、保护和调度下，在不同的时刻，动态地申请和占有计算资源。

特别地，称进程的内存地址空间为该进程的局部内存空间。





进程具有两个明显的特征:

- ◆ 资源特征, 包括程序执行所必需的計算资源
- ◆ 执行特征, 包括在进程执行过程中动态改变的特征

任何进程在执行过程中均涉及如下几种状态:

- 非存在状态
- 就绪状态
- 运行状态
- 挂起状态
- 退出状态





## 2. 进程间通信

无论位于同一台处理机，还是位于不同处理机，进程始终是操作系统资源调度的基本单位，且各个进程不能直接访问其他进程的局部内存空间。

操作系统提供基本的系统调用函数，允许位于同一台处理机或不同处理机的多个进程之间相互交流信息。具体表现为3种形式：**通信、同步和聚集**。

(1) 通信：进程间的数据传递称为进程间通信。在同一台处理机中，通信可以通过读/写操作系统提供的**共享数据缓存区**来实现。在不同处理机中，通信可以通过**网络传输数据**来实现。



特别地，称两个进程之间传递的数据为**消息**，  
称这种操作为**消息传递**。

(2) 同步： 同步是使位于相同或不同处理机中的多个进程之间相互等待的操作，它要求进程的所有操作均必须等待到达某一控制状态之后才进行。

(3) 聚集(或归约)： 聚集将位于相同或不同处理机中的多个进程的局部结果综合起来，通过某种操作，例如求最大值、最小值、累加和，产生一个新的结果，存储在某个指定的或者所有的进程的变量中。



将进程间相互操作的3种形式: 通信、同步和聚集, 统称为**进程间通信**, 而操作的具体数据对象为**消息**, 具体操作作为**消息传递**。

进程间通信的具体实现大体可以分为两类:

- ◆ 在共享存储环境中, 通过读/写操作系统提供的共享数据缓存区来实现;
- ◆ 在分布式存储网络环境中, 通过网络通信来实现。





### 3. 线程

线程(threads), 它是在进程的基础上, 基于对称多处理的现代操作系统的一个重要发明.

由于进程具有独立的局部内存空间, 使得操作系统对它们的管理非常费时。

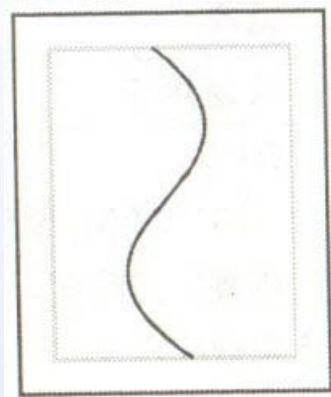
#### 重量级进程

该类进程不适合细粒度的共享存储并行程序设计

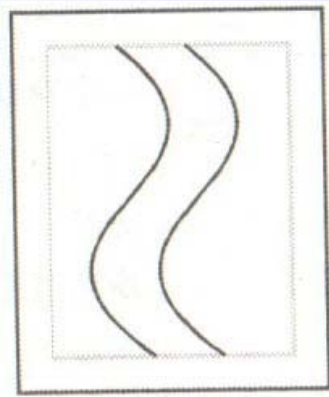
为了在共享存储环境下有效地开发应用程序的细粒度并行度, 将一个进程分解两个部分, 其中一部分由其资源特征构成, 仍称之为进程; 另一部分由其执行特征构成, 称之为线程, 或者轻量级进程。



具体地，如图所示，



(a)单进程单线程执行



(b)单进程双线程执行



(c)单进程多线程执行

进程产生时，其执行特征构成一个线程，称之为**主线程**。主线程调用线程库函数，可以动态地创建新的线程，称之为**从线程**。主线程和从线程**共享进程的资源特征**。





## (二) 并行编程环境

### 3 种并行编程环境主要特征一览表

特征	消息传递	共享存储	数据并行
典型代表	MPI, PVM	OpenMP	HPF
可移植性	所有流行并行计算机	SMP, DSM	SMP, DSM, MPP
并行粒度	进程级大粒度	线程级细粒度	进程级细粒度
并行操作方式	异步	异步	松散同步
数据存储模式	分布式存储	共享存储	共享存储
数据分配方式	显式	隐式	半隐式
学习入门难度	较难	容易	偏易
可扩展性	好	较差	一般



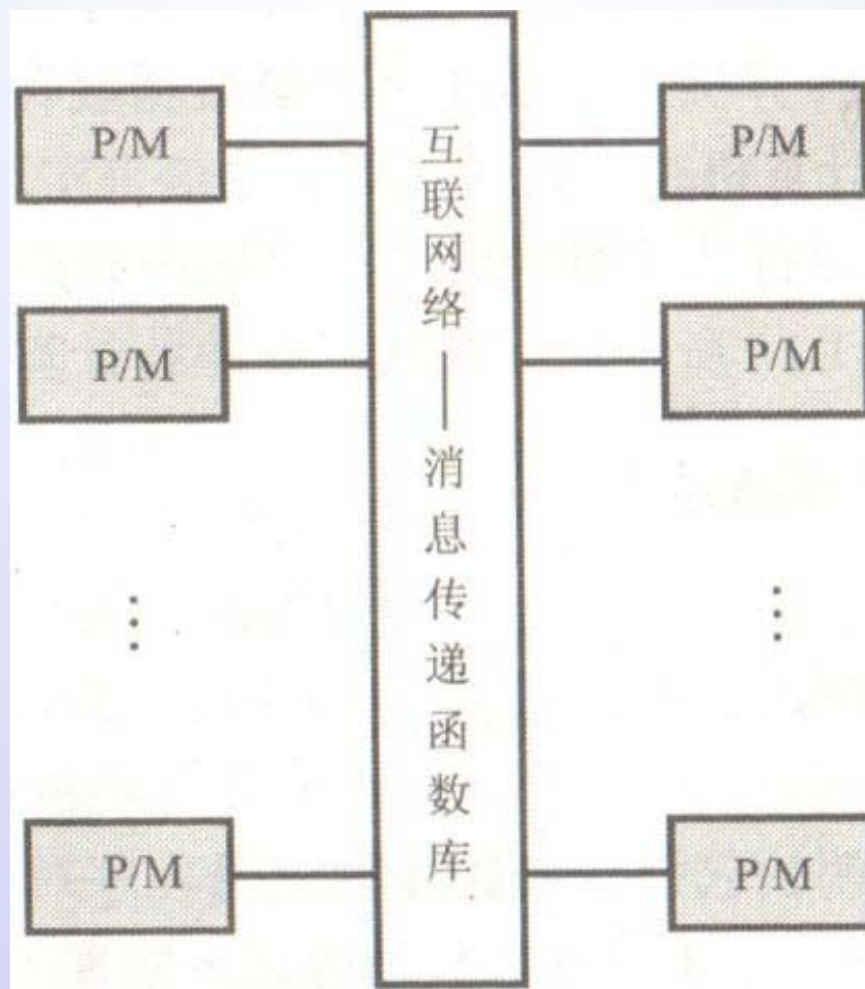
消息传递并行编程基于大粒度的进程级并行，具有最好的可移植性，且具有很好的可扩展性。

但是，消息传递并行编程只能支持进程间的分布存储模式，即各个进程只能直接访问其局部内存空间，而对其他进程的局部内存空间的访问只能通过消息传递来实现。



# 1. 消息传递并行计算机模型

理想的消息传递进程拓扑结构



P表示 MPI 进程

M表示每个进程的  
局部内存空间

分布式存储 的进程拓扑结构.



将图中每个P/M模块替换成处理器，且规定每个处理器只能分配用户程序的一个进程，则所得的理想并行计算机模型就是消息传递并行计算机模型。

程序研制成功后，并可以在任何支持该并行计算机模型隐含的进程拓扑结构的所有具体并行计算机上运行。

消息传递分布式存储并行计算机模型和具体并行计算机体系结构没有必然的联系。





## 2. 标准消息传递界面 MPI

- **Message Passing Interface**:是消息传递函数库的标准规范，由MPI论坛开发，支持**Fortran**和**C**
  - 一种新的库描述，不是一种语言。共有上百个函数调用接口，在**Fortran**和**C**语言中可以直接对这些函数进行调用
  - **MPI**是一种标准或规范的代表，而不是特指某一个对它的具体实现
  - **MPI**是一种消息传递编程模型，并成为这种编程模型的代表和事实上的标准

标准串行程序设计语言

+

MPI消息传递库函数

MPI并行程序设计所依赖的并行编程环境



# MPI的发展过程

- 发展的两个阶段
  - MPI 1.1: 1995
    - **MPICH**: 是MPI最流行的非专利实现, 由Argonne国家实验室和密西西比州立大学联合开发, 具有更好的可移植性.
  - MPI 1.2~2.0: 动态进程, 并行 I/O, 支持F90和C++(1997).



# 为什么要用MPI?

- 高可移植性
  - **MPI**已在**IBM PC**机上、**MS Windows**上、所有主要的**Unix**工作站上和所有主流的并行机上得到实现。使用**MPI**作消息传递的**C**或**Fortran**并行程序可不加改变地运行在**IBM PC**、**MS Windows**、**Unix**工作站、以及各种并行机上。



# 从简单入手!

- 下面我们首先以C语言的形式给出一个最简单的MPI并行程序Hello (下页).
- 该程序在终端打印出Hello World!字样.
- “Hello World”:一声来自新生儿的问候.



# Hello world(C)

```
#include <stdio.h>
#include "mpi.h"

main(
    int argc,
    char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello,
world!\n" );
    MPI_Finalize();
}
```



### 三、并行算法

并行算法是适合在并行计算机上实现的算法，一个好的并行算法应该具备充分发挥并行计算机潜在性能的能力。

#### (一) 并行算法的分类

根据运算基本对象的不同可分为

- ① 数值并行算法: 主要为数值计算方法而设计的并行算法。
- ② 非数值并行算法: 主要为符号运算而设计的并行算法，如图论算法、遗传算法等。





根据并行进程间相互执行顺序关系的不同可分为

- ① 同步并行算法：进程间由于运算执行顺序而必须相互等待的并行算法，如通常的向量算法、SIMD算法、MIMD并行计算机上进程间需要相互等待通信结果的算法等。
- ② 异步并行算法：进程间执行相对独立，不需要相互等待的一种算法，通常针对消息传递MIMD并行计算机设计，其主要特征是在计算的整个过程中均不需要等待，而是根据最新消息决定进程的继续或终止。
- ③ 独立并行算法：进程间执行是完全独立的，计算的整个过程不需要任何通信。



根据各进程承担的计算任务粒度的不同，可分为：

- ① 细粒度并行算法：通常指基于向量和循环级并行的算法；
- ② 中粒度并行算法：通常指基于较大的循环级并行；
- ③ 大粒度并行算法：通常指基于子任务级并行的算法，例如通常的基于区域分解的并行算法，它们是当前并行算法设计的主流。

并行算法的粒度是一个相对的概念



## (二) 并行算法的发展阶段

1. 基于向量运算的并行算法设计阶段
2. 基于多向量处理机的并行算法设计阶段
3. SIMD类并行计算机上的并行算法设计阶段
4. MIMD类并行计算机上的并行算法设计阶段
5. 现代并行算法设计

随着微处理器和互联网络速度的发展，可扩展高性能的获取必须要求并行算法设计兼顾两个发展方向：

1. 可扩展、可移植的大粒度任务级并行；
2. 在每个进程，组织便于发挥单机性能的合理数据结构、程序设计和通信方式。

