

Parallelism Computing

并行计算基础





并行计算基础

参考书目



Peter S.P. 《并程序程序设计导论》，
机械工业出版社



张林波 《并行计算导论》，清华大
学出版社



教学目标

- 熟悉高性能科学计算环境
 - --Linux操作系统(BASH,AWK,PERL,Python)
 - --编译器、常用函数库(BLAS,LAPACK)
- 掌握至少一种高级计算语言(FORTRAN 90,C/C++)
- 理解并行计算的基本原理
- 掌握编写并行程序的基本方法
 - --MPI(Message Passing Interface)
 - --OpenMP(Open Multi-Processing)

多练习！多实践！

An Introduction to Parallel Programming

Peter Pacheco

Chapter 1

为什么要并行计算？





Roadmap

- 为什么需要不断提升的性能
- 为什么需要构建并行系统
- 为什么需要编写并行程序
- 我们将做什么
- 并发、并行、分布式！



并行计算的概念

- 并行机上所做的计算， 又称高性能计算(High Performance Computing, HPC)或超级计算(Supercomputing)。



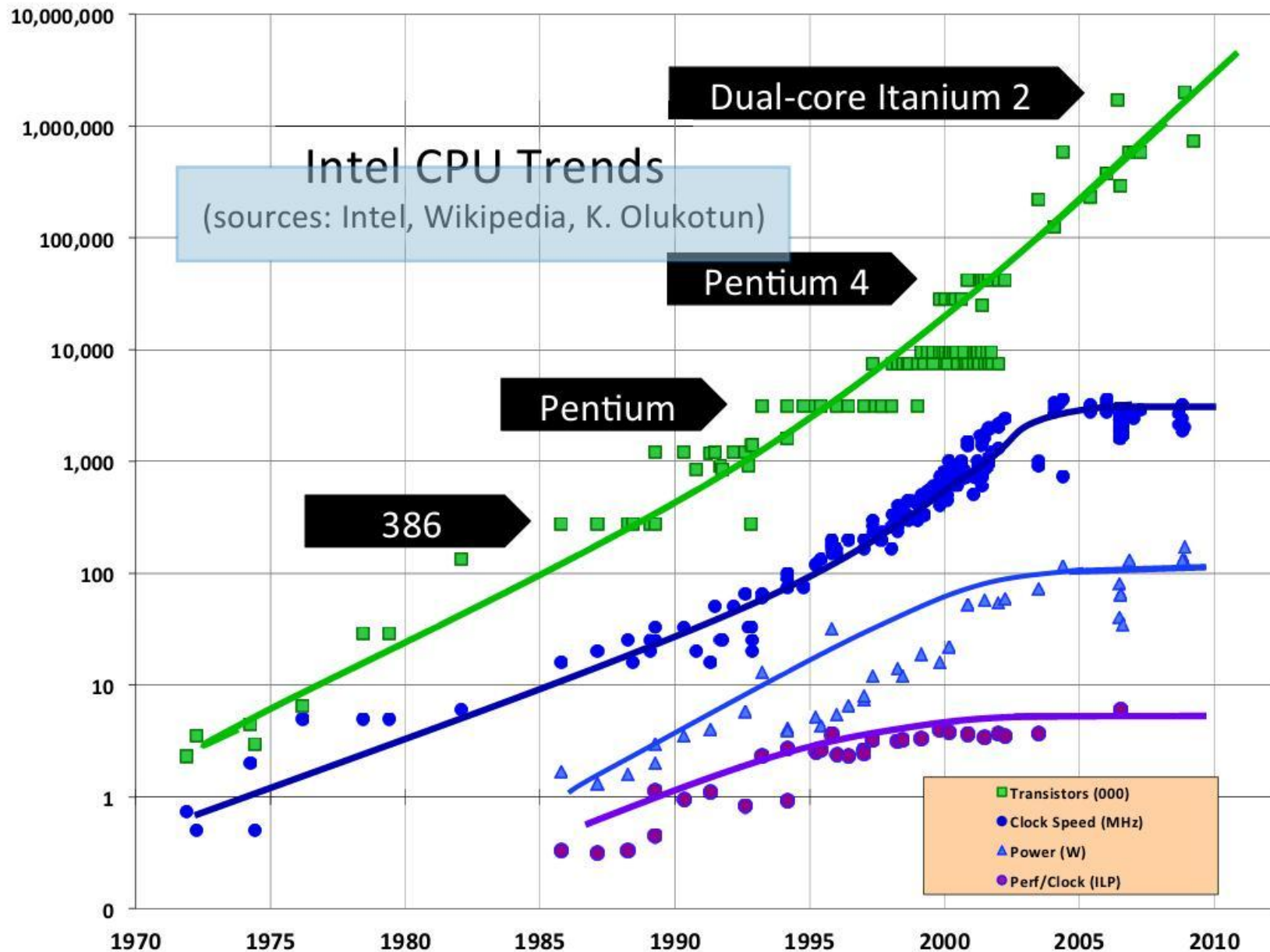
- 基本条件
 - 硬件（并行机）、并行算法设计、并行编程环境
- 主要目标
 - 提高求解速度、扩大问题规模



处理器的发展

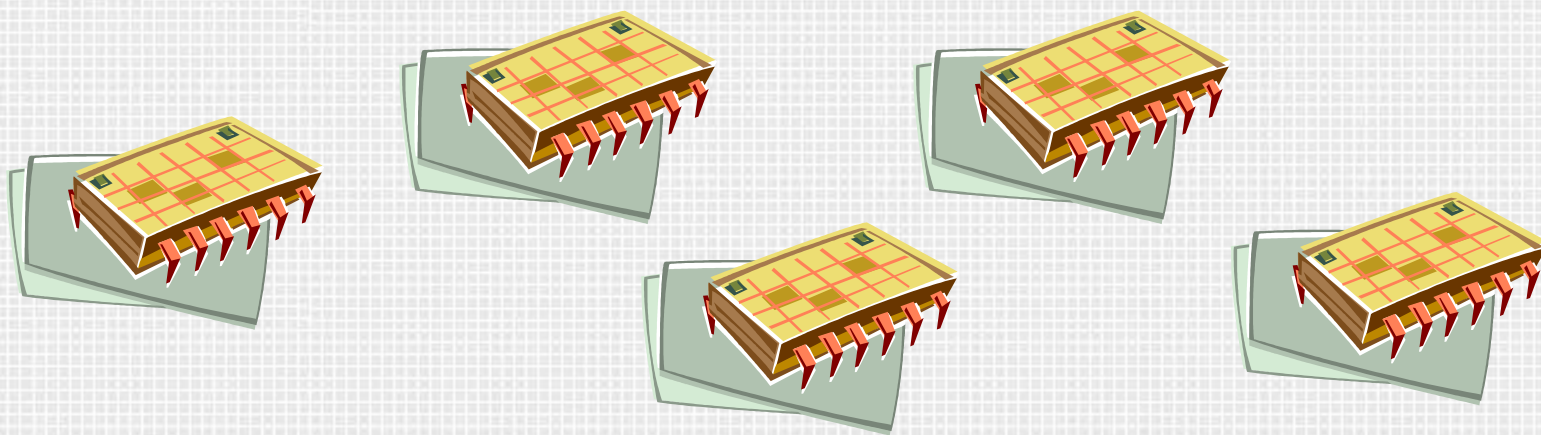
- 从 **1986 – 2002**, 微处理器的处理速度火箭式的飞跃发展, 平均每年的**50%**的速度不断提升。
- 从**2002**年开始, 单处理器的性能提升降低到每年大约**20%**。





解决方案

- 将多个完整的单处理器放到一个集成的电路芯片上。



带给软件开发人员的问题

- 程序串行
- 多处理系统与单处理系统性能不发生变化
- 怎么去使用多核？





1.1 为什么需要不断提升的性能

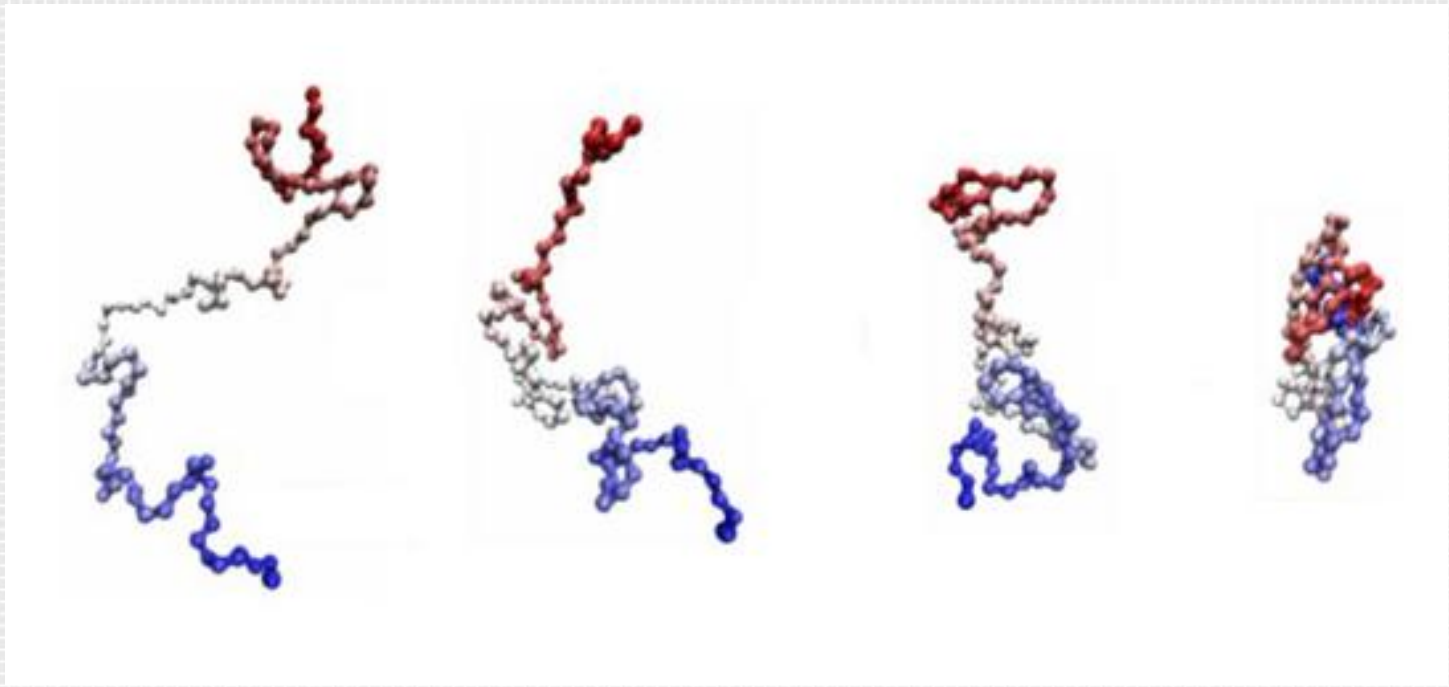
- 不断提升的计算能力已经成为我们生活中一部分。
- 现实的未知性，更复杂的问题需要高性能。
 -

气候模拟





蛋白质折叠



药物发现



能量研究



数据分析



A close-up view of a table with numerical data. The table has a grid structure with rows and columns. The numbers are positive values, likely representing financial or statistical data. The values are: +2.688, +5.000, +1.500, +1.125, and +1.062.

3	+2.688
0	+5.000
	+1.500
	+1.125
	+1.062



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 **1T**，算性能高达 **25** 万亿次 / 秒

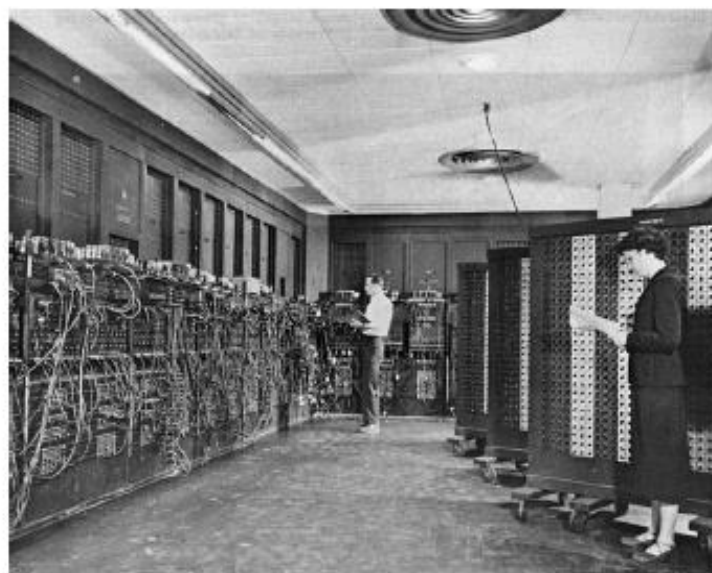
- 核武器数值模拟

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

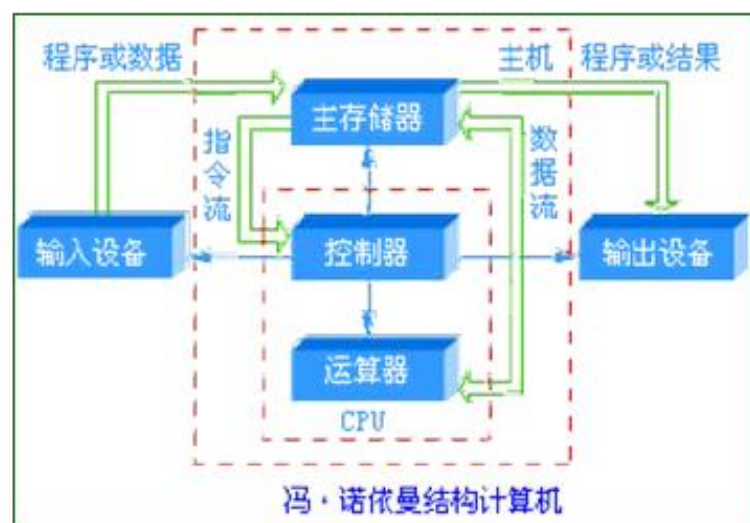
- 天体物理、航空航天、油藏模拟、地震数据处理、密码破译、 新药研制、生物信息处理、图像处理等.

第一台计算机

- 1946 年，世界上第一台计算机 ENIAC 诞生
(Electronic Numerical Integrator And Computer,
电子数值积分计算机，美国宾夕法尼亚大学)



计算机之父



- 占地约 170 平方米，重约 30 吨，耗电150千瓦
- 运算速度：5000 次加法/秒或 500 次乘法/秒
- 15分钟换一个零件
- 用途：弹道计算和氢弹研制

超级计算元年

● 超级计算元年的标志：Cray-1 向量机



Cray-1原型



西摩·克雷 Seymour Cray

(1925-1996)，电子工程学学士，应用数学硕士，超级计算之父，Cray研究公司的创始人，亲手设计了Cray机型的全部硬件与操作系统，作业系统由他用机器码编写完成。1984年时，公司占据了

超级计算机市场 70%的份额。1996年Cray研究公司被SGI收购，2000年被出售给Tera计算机公司，成立Cray公司。2012年12月全球排名第一的超级计算机由克雷公司制造的Cray XK7组建。

- 一般将 Cray-1 投入运行的 1976 年称为“**超级计算元年**”
- 编程方便，但可扩展性差
- 以 Cray 为代表的向量机称雄超级计算机界十几载



80 年代早期

- 80 年代百家争鸣

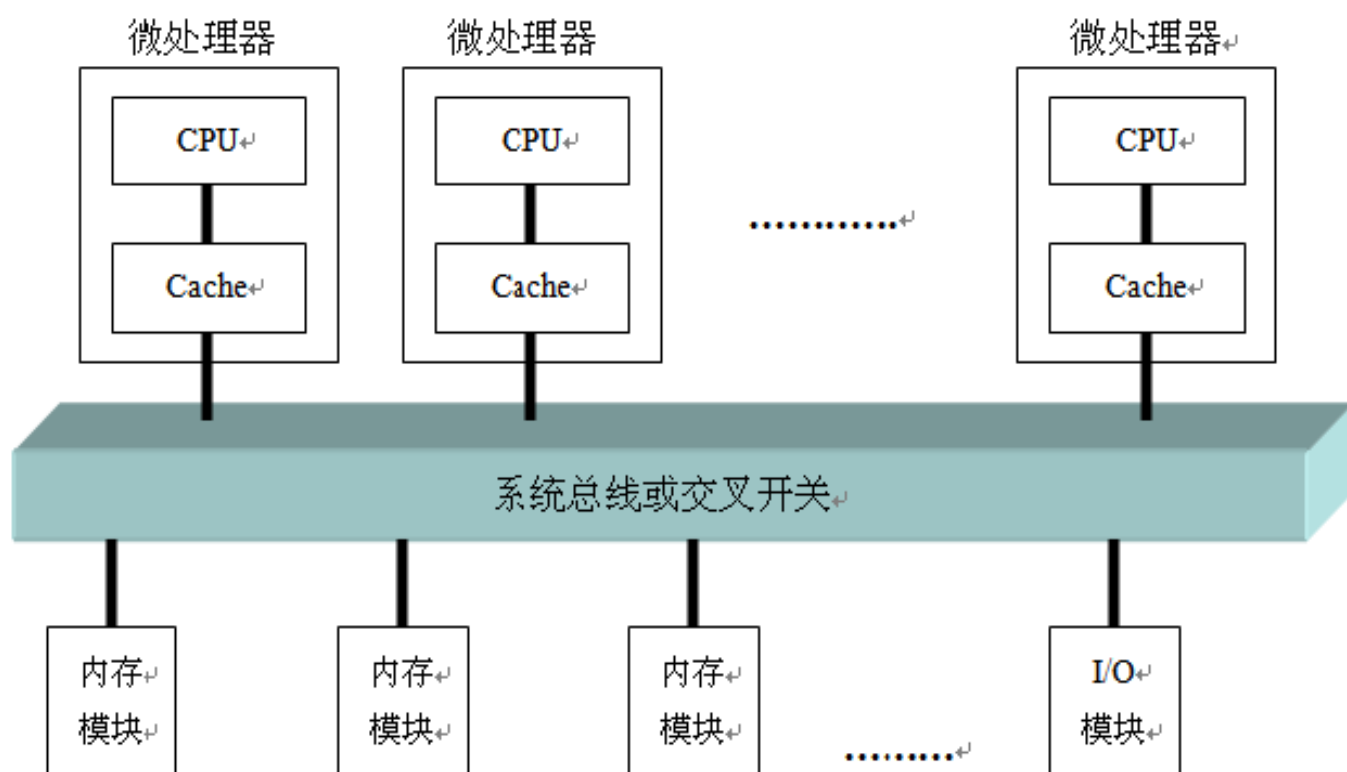
早期：以 MIMD 并行计算机的研制为主

- Denelcor HEP (1982年) 第一台商用 MIMD 并行计算机
- IBM 3090 80 年代普遍为银行所采用
- Cray X-MP Cray 研究公司第一台 MIMD 并行计算机

80 年代中期

● 中期：共享存储多处理机 Shared-Memory MultiProcessor

SMP (Symmetrical Multi-Processing): 在一个计算机上汇集一组处理器，各处理器对称共享内存及计算机的其他资源，由单一操作系统管理，极大提高整个系统的数据处理能力。

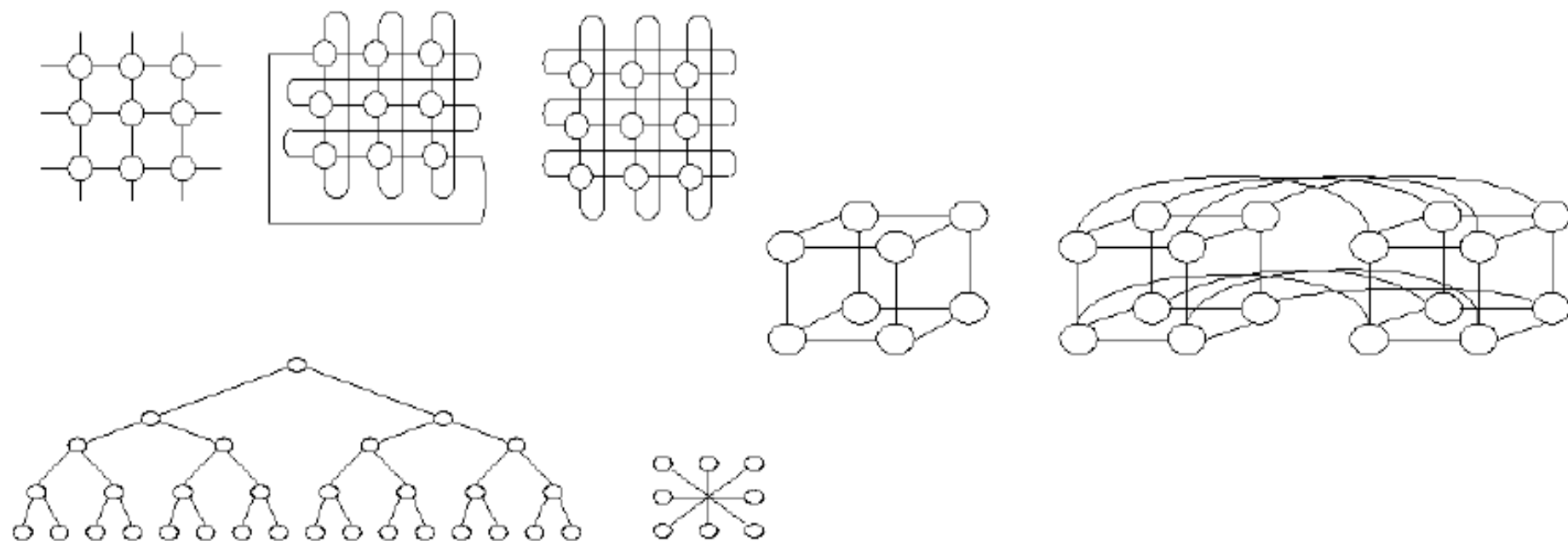


- 扩展性较差
- 可靠性较差
- 内存访问瓶颈

80 年代后期

- 后期：具有强大计算能力的并行机

- 通过二维Mesh连接的Meiko (Sun) 系统
- 超立方体连接的 MIMD 并行机：nCUBE-2、iPSC/80
- 共享存储向量多处理机 Cray Y-MP
-

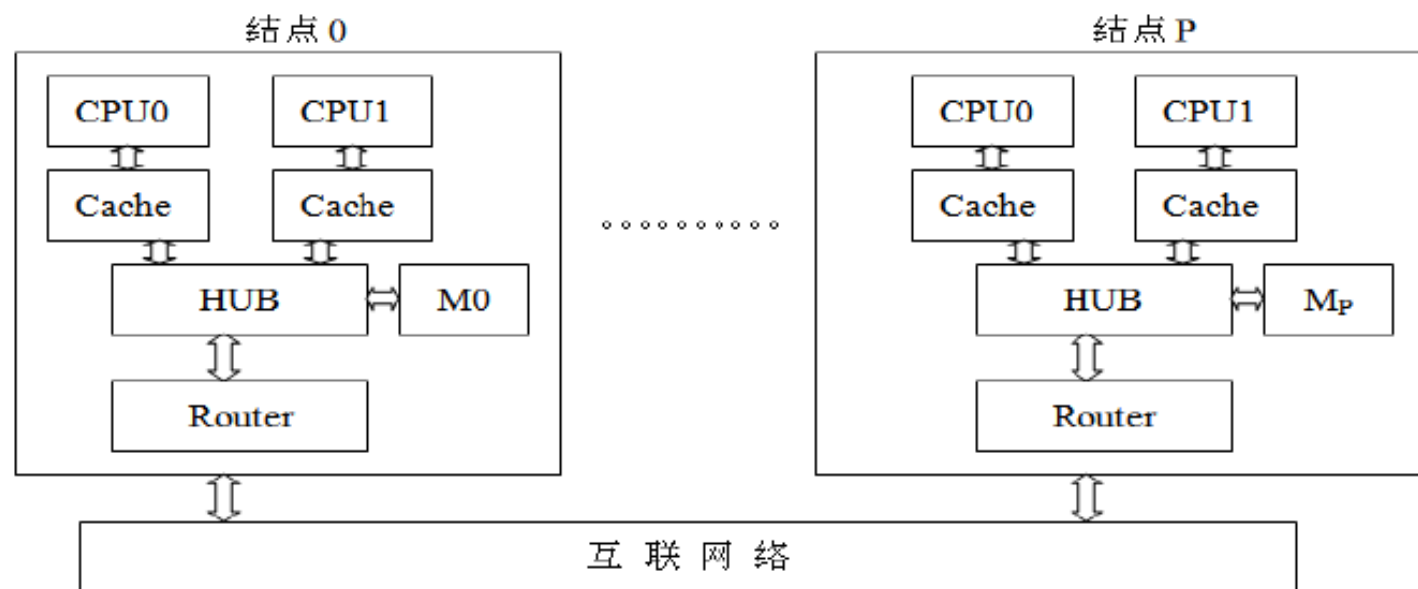


90 年代: DSM

● 90 年代: 体系结构框架趋于统一 (DSM、MPP、NOW)

● DSM (Distributed Shared Memory) 分布式共享存储

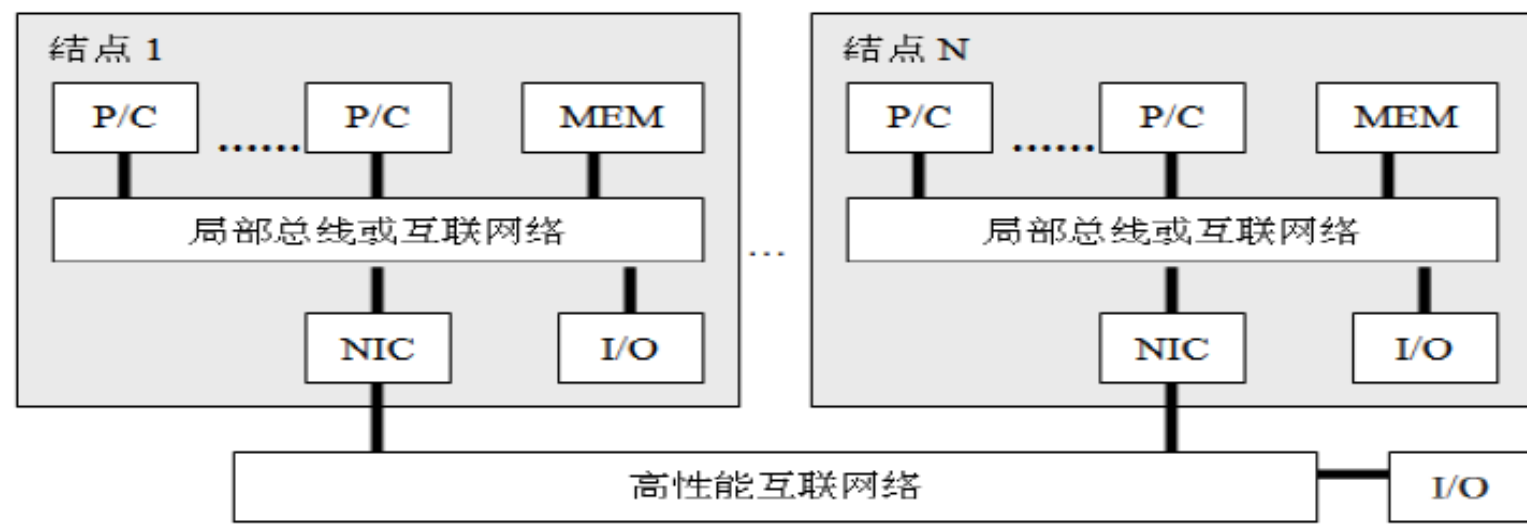
- 以结点为单位, 每个结点有一个或多个 CPU
- 专用的高性能互连网络连接 (Myrinet, Infiniband, ...)
- 分布式存储: 内存模块局部在每个结点中
- 单一的操作系统, 单一的内存地址空间
- 可扩展到上百个结点, 支持消息传递、共享存储并程序序设计



90 年代: MPP

● MPP (Massively Parallel Processing) 大规模并行处理结构

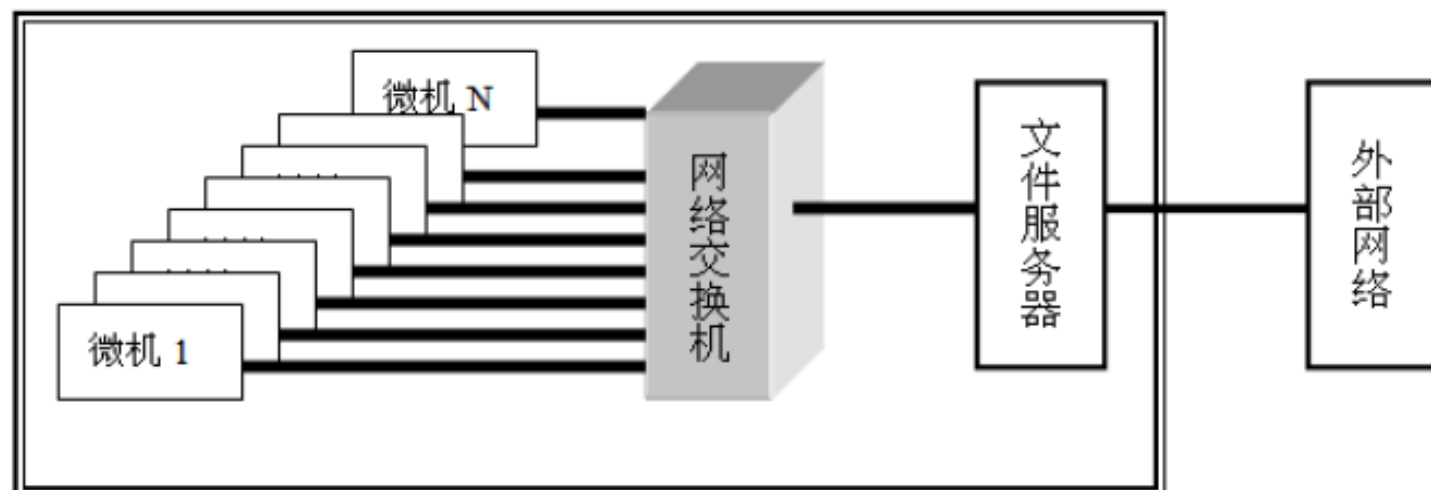
- 每个结点相对独立，有一个或多个微处理器
- 每个结点有自己的操作系统和独立内存，避免内存访问瓶颈
- 各个结点只能访问自己的内存模块
- 扩展性较好
- DM-MPP: 每个结点仅包含一个微处理器;
- SMP-MPP: 每个结点是一台 SMP 并行机
- DSM-MPP: 每个结点是一台 DSM 并行机



90 年代: NOW

● NOW (Network of Workstations) 工作站机群

- 每个结点都是一个完整的工作站，有独立的硬盘与UNIX系统
- 结点间通过低成本的网络（如千兆以太网）连接
- 每个结点安装消息传递并行程序设计软件，实现通信、负载平衡等
- 投资风险小、结构灵活、可扩展性强、通用性好、异构能力强，被大量中小型计算用户和科研院校所采用
- 也称为 COW (Cluster of Workstations)
- NOW (COW) 与 MPP 之间的界线越来越模糊



2000 年至今

● 2000 年至今：前所未有的大踏步发展

● Cluster 机群 / 集群

- 每个结点含多个商用处理器，结点内部共享存储
- 采用商用机群交换机通过前端总线连接结点，结点分布存储
- 各个结点采用 Linux 操作系统、GNU 编译系统和作业管理系统

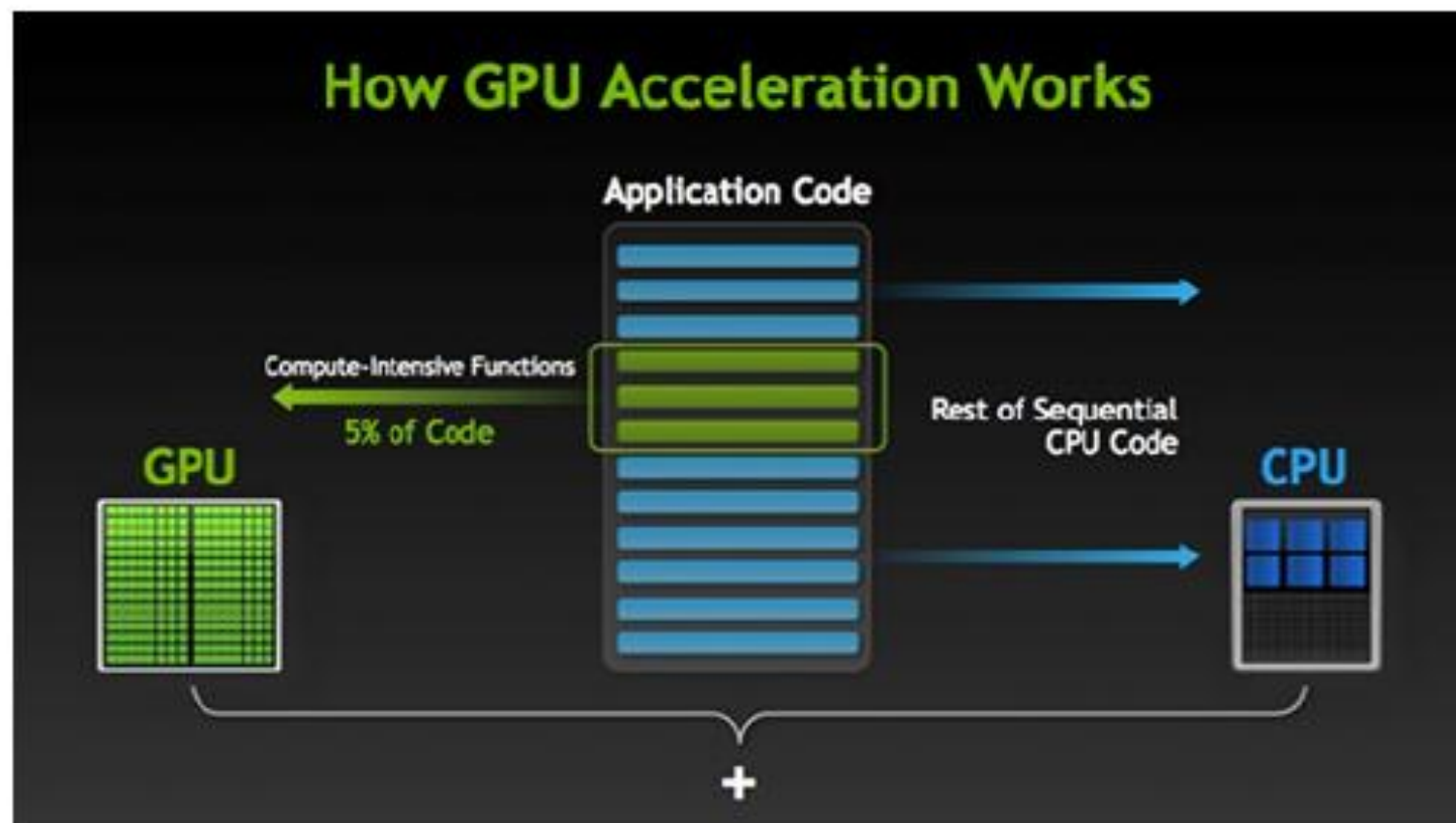
● Constellation 星群

- 每个结点是一台子并行机
- 采用商用机群交换机通过前端总线连接结点，结点分布存储
- 各个结点运行专用的结点操作系统、编译系统和作业管理系统

● MPP

- 专用高性能网络，大多为政府直接支持

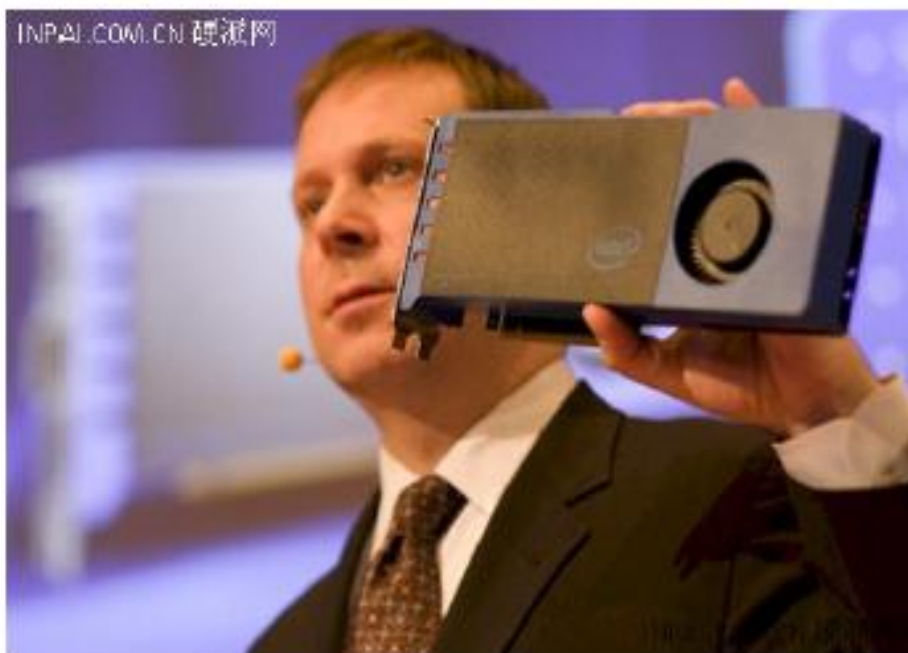
2007: GPU用于科学计算



运算能力可达TFLOPS量级

一个芯片上的超级计算机

2012: Intel MIC协处理器



Knights Ferry



- Software development platform
- Growing availability through 2010
- 32 cores, 1.2 GHz
- 128 threads at 4 threads / core
- 8MB shared coherent cache
- 1-2GB GDDR5
- Bundled with Intel HPC tools

Software development platform for Intel® MIC architecture

ChinaByte 比特网

32核心! Intel MIC Knights Ferry

Knights Ferry内置有32个X86处理器核心，频率为1.2GHz，支持quad-HyperThreading技术。相关产品将会基于PCI Express 2.0插槽，配备有最多2G GDDR5内存。该芯片本身内置有8M L2缓存，这点让人觉得非常有趣，因为高度并行应用并不需要如此之大的缓存。Intel并没有透露Knights Ferry的计算性能到底达到了多少GigaFLOPS或TeraFLOPS。

2012年推出，计划达到2.5Tflop/s

From Research to Realization. Announcing...



Intel® Many Integrated Core Architecture

The Newest Addition to the Intel Server Family.
Industry's First General Purpose Many Core Architecture

ChinaByte 比特网



世界最快超级计算机

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	Piz Daint - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890



世界最快超级计算机

6	Cori - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939
7	Oakforest-PACS - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu Joint Center for Advanced High Performance Computing Japan	556,104	13,554.6	24,913.5	2,719
8	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu RIKEN Advanced Institute for Computational Science (AICS) Japan	705,024	10,510.0	11,280.4	12,660
9	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom , IBM DOE/SC/Argonne National Laboratory United States	786,432	8,586.6	10,066.3	3,945
10	Trinity - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	301,056	8,100.9	11,078.9	4,233

神威·太湖之光超级计算机



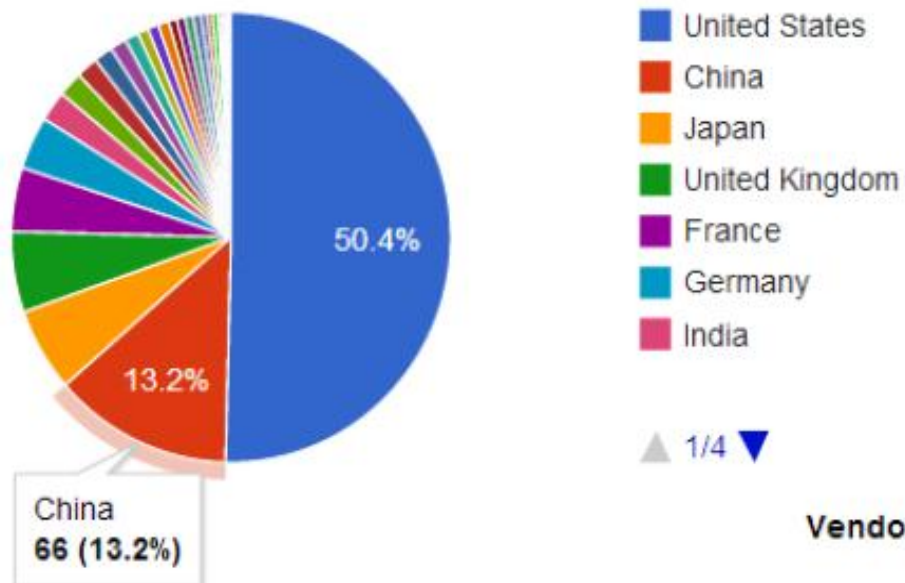
神威·太湖之光超级计算机系统的峰值性能
125.436PFlops，世界第一；持续性能
93.015PFlops，世界第一；性能功耗比
6051MFlops/W，还是世界第一。

神威·太湖之光超级计算机由40个运算机柜和8个网络机柜组成。每个运算机柜比家用的双门冰箱略大，打开柜门，4块由32块运算插件组成的超节点分布其中。每个插件由4个运算节点板组成，一个运算节点板又含2块“申威26010”高性能处理器。一台机柜就有1024块处理器，整台“神威·太湖之光”共有40960块处理器。

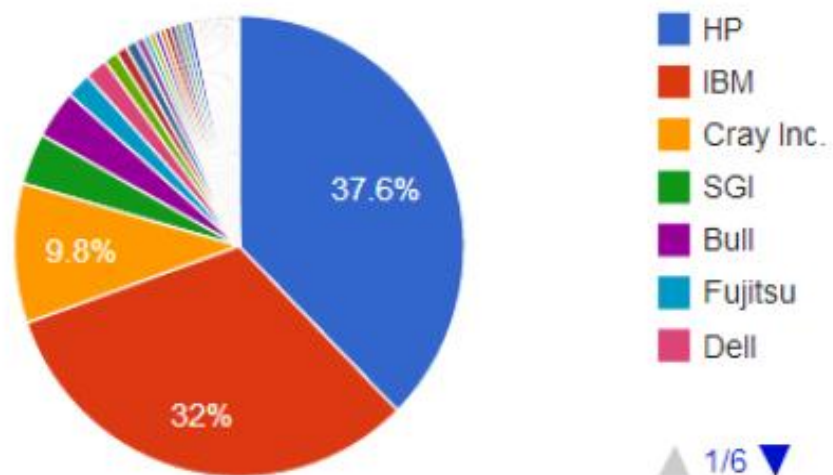
TOP500

● 国家和制造商

Countries System Share



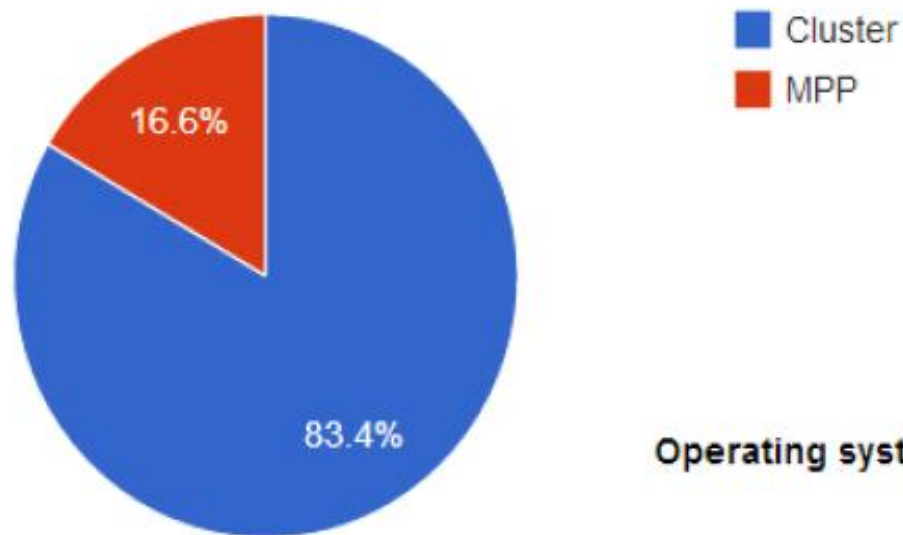
Vendors System Share



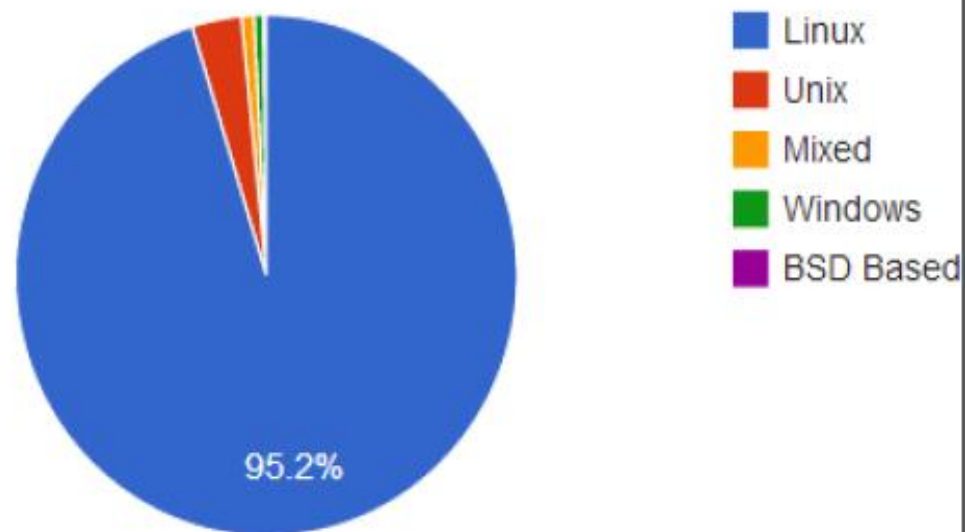
TOP500

● 计算机类型与操作系统

Architecture System Share



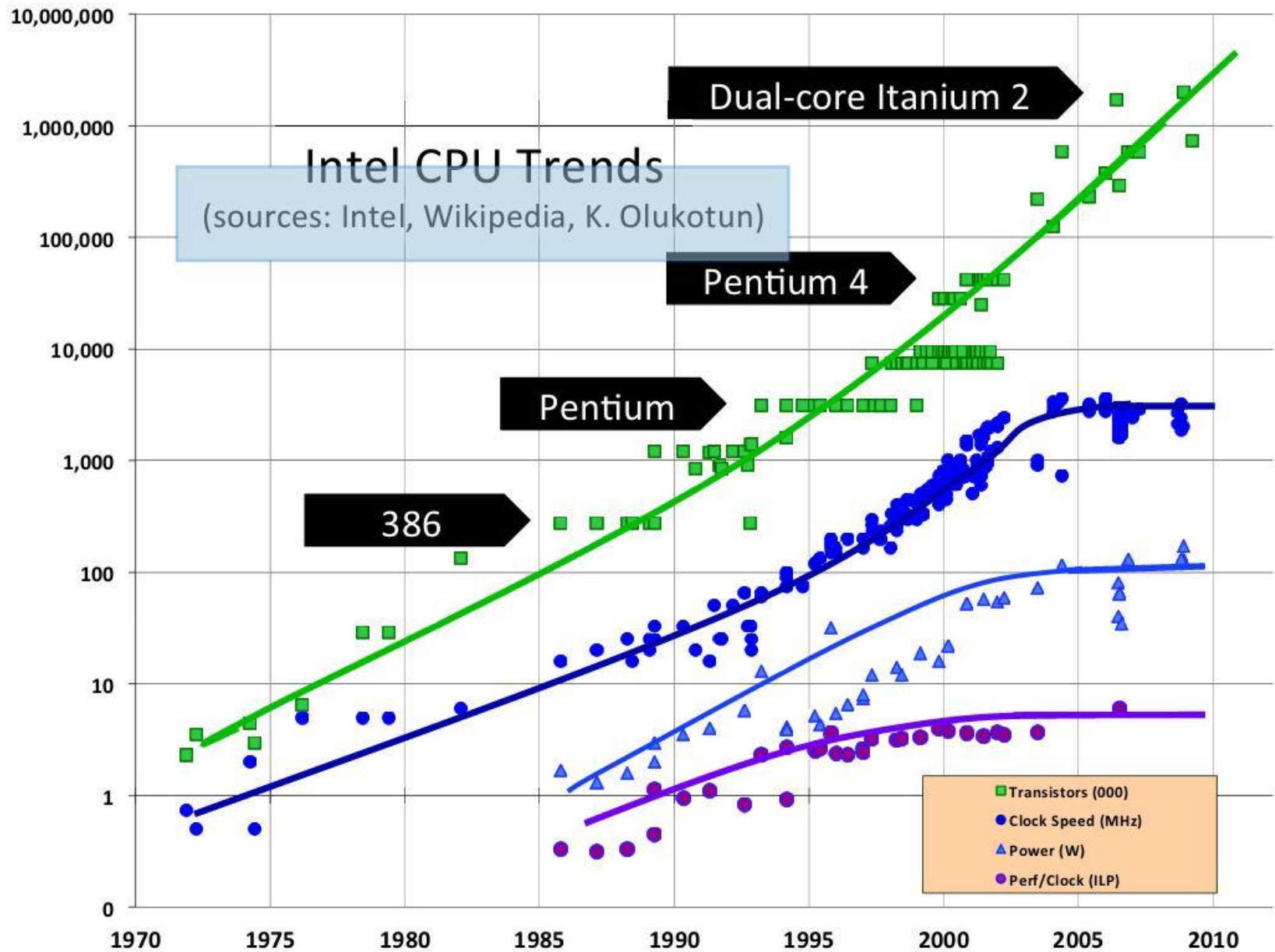
Operating system Family System Share

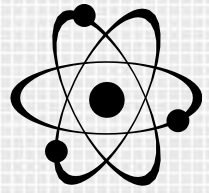


1.2 为什么需要构建并行系统

- 单处理器性能大幅度提升的原因之一，是日益增加的集成电路晶体管密度。
- 很重要的问题！！





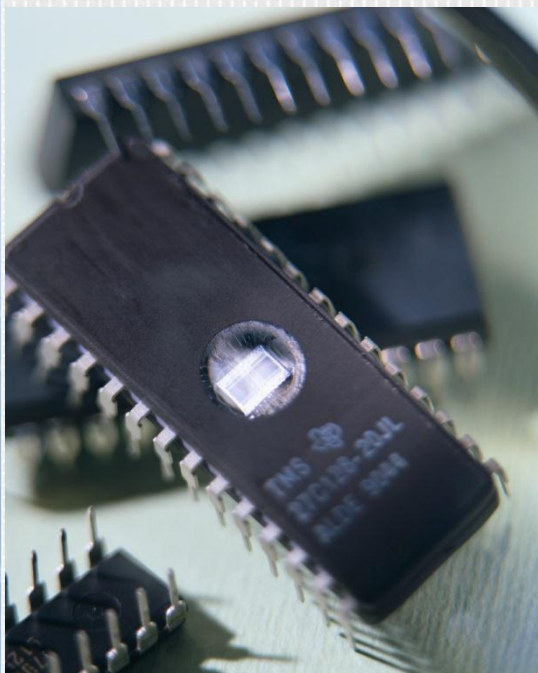


一个简单的推导

- 更小的晶体管= 更快的处理器.
- 更快的处理器= 更高的能耗.
- 更高的能耗= 更高的热量.
- 更高的热量= 不稳定的处理器.

解决办法

- 单核处理系统转向**多核处理系统**.
- 串行转向**并行**
- 核“core” = central processing unit (CPU)



1.3 为什么需要编写并行程序

- 单核系统编写的程序无法利用多核处理器.
- 在多核系统中运行多个程序的实例
- 我们想要的是这个程序更快的运行





解决办法

- 将串行程序改成并行程序。
- 编写一个翻译程序来自动将串行程序翻译成并行程序。

—难。

—鲜有突破。



问题

- 通过编写一些程序，让程序辨识串行程序的常见结构，并自动转换成并行程序的结构。
 - **for**循环
- 实际运行时间低效。



示例


- 计算n个数的值再累加求和。
- 串行代码

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```



示例

- 假设有 p 个核， p 远小于 n .
- 每个核的计算大约 n/p 个数的值并累加求和.



```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
    my_x = Compute_next_value( . . . );
    my_sum += my_x;
}
```

每个核都有自己的私有变量，并且独立执行。



示例

- **Ex., 8 核, $n = 24$, 调用24次**
Compute_next_value 获得数值:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9



示例

- 当每个核都计算完各自的 **my_sum** 之后，将各自的结果值发送給一个指定的“**master**”核，主核。
- **master**核进行全局总和。



示例

```
if (I'm the master core) {  
    sum = my_x;  
    for each core other than myself {  
        receive value from core;  
        sum += value;  
    }  
} else {  
    send my_x to the master;  
}
```




示例

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

$$8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95$$

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14



But wait!

有没有更好的办法求解全局总和.





并行优化

- 1. 不再由**master**核计算所有部分的累加工作。

各个核两两组队，得到部分和。

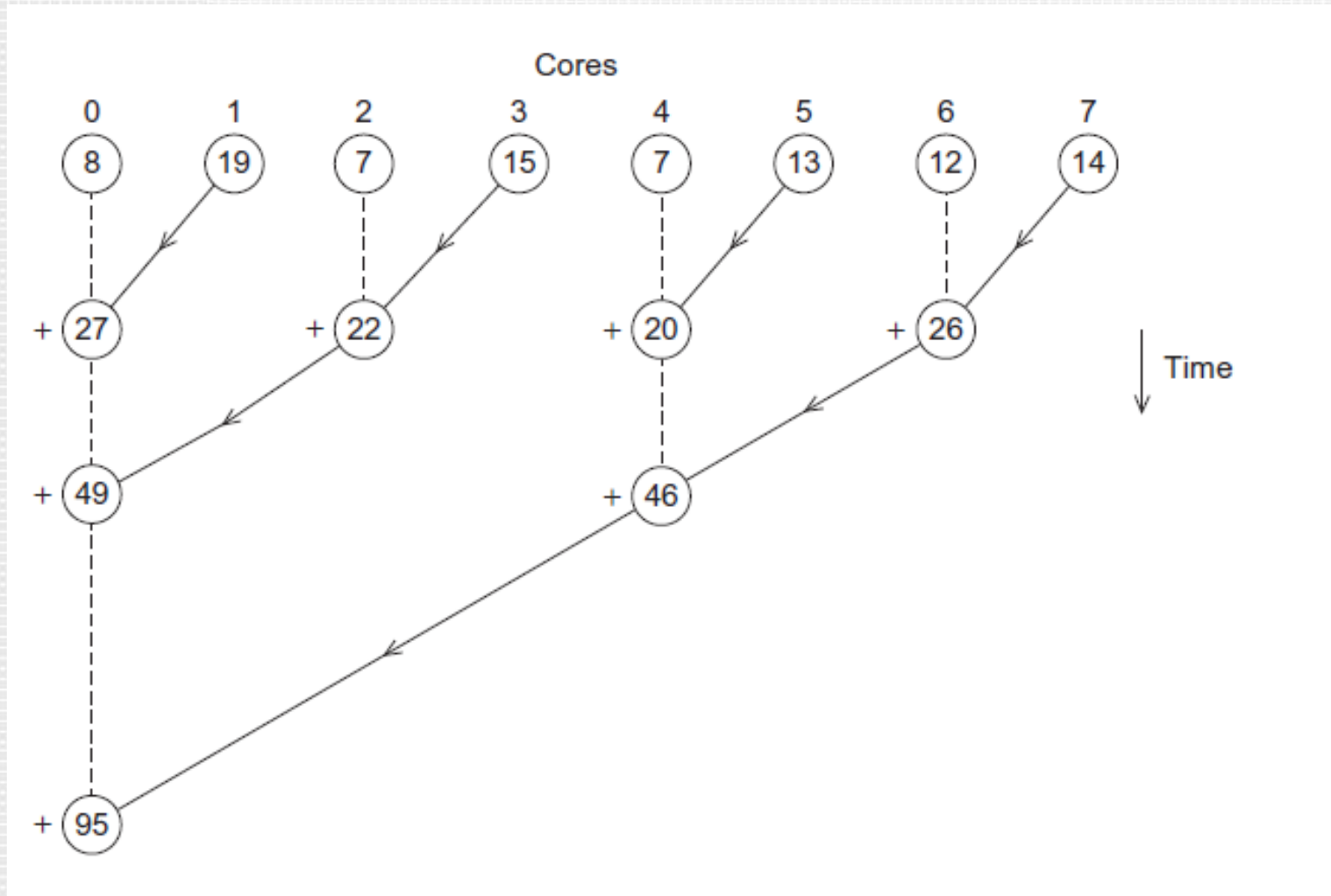
- **ex.** 0号核将自己的结果和1号核的结果相加；
2号核的结果和3号核的结果相加.....



并行优化

- **2.然后，再在偶数核上重复累加部分和。**
 - **Core 0 adds result from core 2.**
 - **Core 4 adds the result from core 6, etc.**
- **3.重复第2步，知道0号核（master核得到最终结果）。**

多个核共同计算一个全局总和





算法分析

- 在第一个算法中，**master**核执行了**7**次接收和相加操作。
- 在第二个算法中，**master**核执行了**3**次接收和相加操作。
- 提高了**2**倍！！！！



- 当有更多的核时，两者的差异会更大。
- 假设有**1000**个核的情况下：
 - 第一种算法需要执行**999**次接收操作和**999**次相加操作。
 - 第二种算法只需要执行**10**次接收操作和**10**次相加操作。
- 提高了**100倍!**



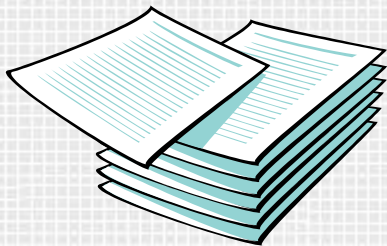
1.4 怎样编写并行程序？

- 任务并行 (**Task parallelism**)
 - 将待解决的问题所需要执行的各个任务分配到各个核上执行。
- 数据并行 (**Data parallelism**)
 - 将待解决的问题所需要处理的数据分配给各个核。
 - 每个核在分配到的数据集上执行大致相似的操作。



Professor P

15 questions
300 exams



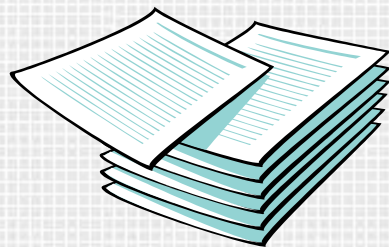


Professor P's 助教

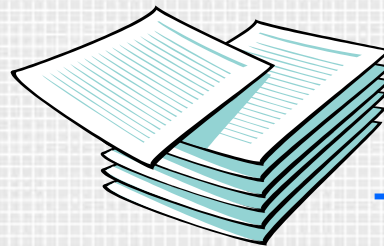


数据并行

TA#1

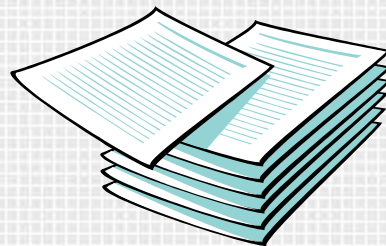


100 exams



TA#3

100 exams



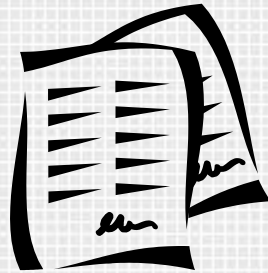
TA#2

100 exams



任务并行

TA#1



Questions 1 - 5



TA#3

Questions 11 - 15



TA#2

Questions 6 - 10



数据并行

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```



任务并行

```
if (I'm the master core) {  
    sum = my_x;  
    for each core other than myself {  
        receive value from core;  
        sum += value;  
    }  
} else {  
    send my_x to the master;  
}
```

Tasks

- 1) 接收和相加操作
- 2) 传递部分和操作



协调过程

- 通信（**Communication**） – 一个或者多个核将自己的部分和结果发送到其他核。
- 负载均衡（**Load balancing**） – 每个核分配大致相同数目的数据来计算。
- 同步（**Synchronization**） – 在大多数系统中，每个核有自己的执行空间，不能自动同步。



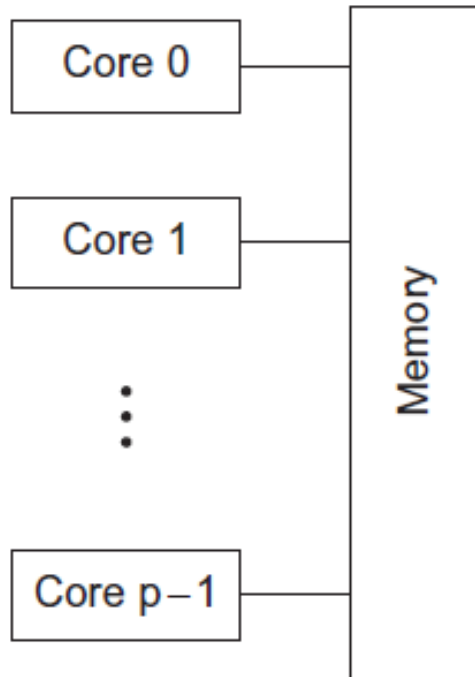
1.5 我们将做什么

- 学习如何编写显示并行的程序。
- **C语言为基础。Using the C language.**
- 利用**C语言**的三个不同的扩展库：
 - 消息传递接口 (**Message-Passing Interface , MPI**)
 - **Posix 线程 (Posix Threads , Pthreads)**
 - **OpenMP**



1.6 并行系统的类型

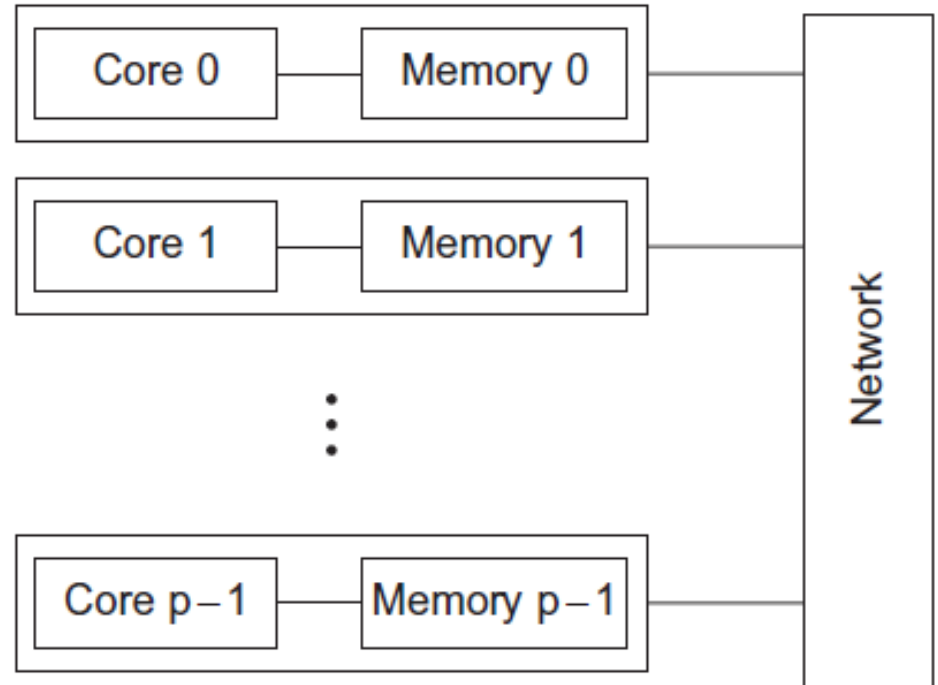
- 共享内存系统 (**Shared-memory**)
 - 各个核能够共享访问计算机的内存。
 - 通过检测和更新共享内存中的数据来协调各个核。
- 分布式内存系统 (**Distributed-memory**)
 - 每个核都有拥有自己的私有内存。
 - 核之间的通信是显式的，必须使用类似于在网络中发送消息的机制。



(a)

Shared-memory

OpenMP
Pthreads



(b)

Distributed-memory

MPI



- 并发计算（**Concurrent computing**）
 - 一个程序的多个任务在**同一个时间段内**可以同时执行。
- 并行计算（**Parallel computing**）
 - 一个程序**同一时刻**通过**多个任务紧密协作**来解决某个问题。
- 分布式计算（**Distributed computing**）
 - 一个程序需要**同一时刻**与**其他程序协作**来解决某个问题。