

# 并行计算

## (Parallel Computing)

# 课程介绍

## 学习目的：

- ✓ 学习和掌握并行计算的基本概念
- ✓ 学习并行算法的分析与设计方法
- ✓ 学习并行程序设计、环境与工具等

## 课时安排：

- ✓ 课堂讲授 32学时

## 考核方法：

- ✓ 作业 + 报告

# 课程介绍

## 参考教材：

- ✓ An Introduction to Parallel Programming, Peter S. Pacheco, Elsevier, 2011
- ✓ Introduction to Parallel Computing (2ed) , Ananth, Addison Wesley, 2003
- ✓ 并行算法的设计与分析, 陈国良, 高等教育出版社, 2009

# 课程介绍

授课教师：

✓ 丁建睿

联系方式：

✓ dingjianrui@163.com, 研究院一号楼北505

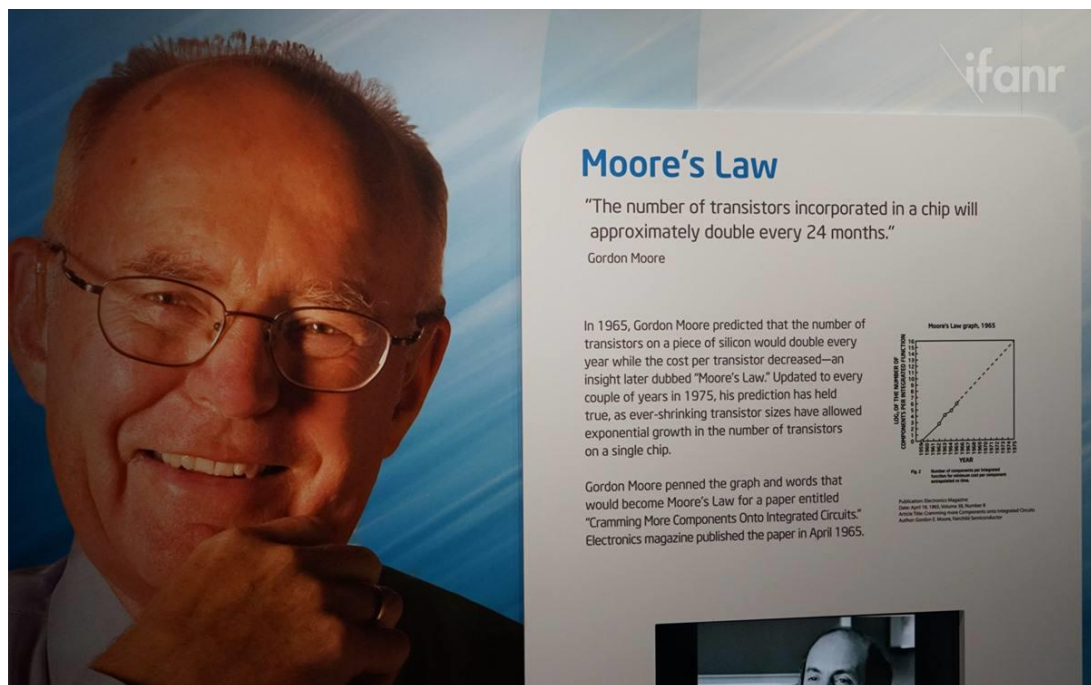
# 并行计算概述

## 学习内容：

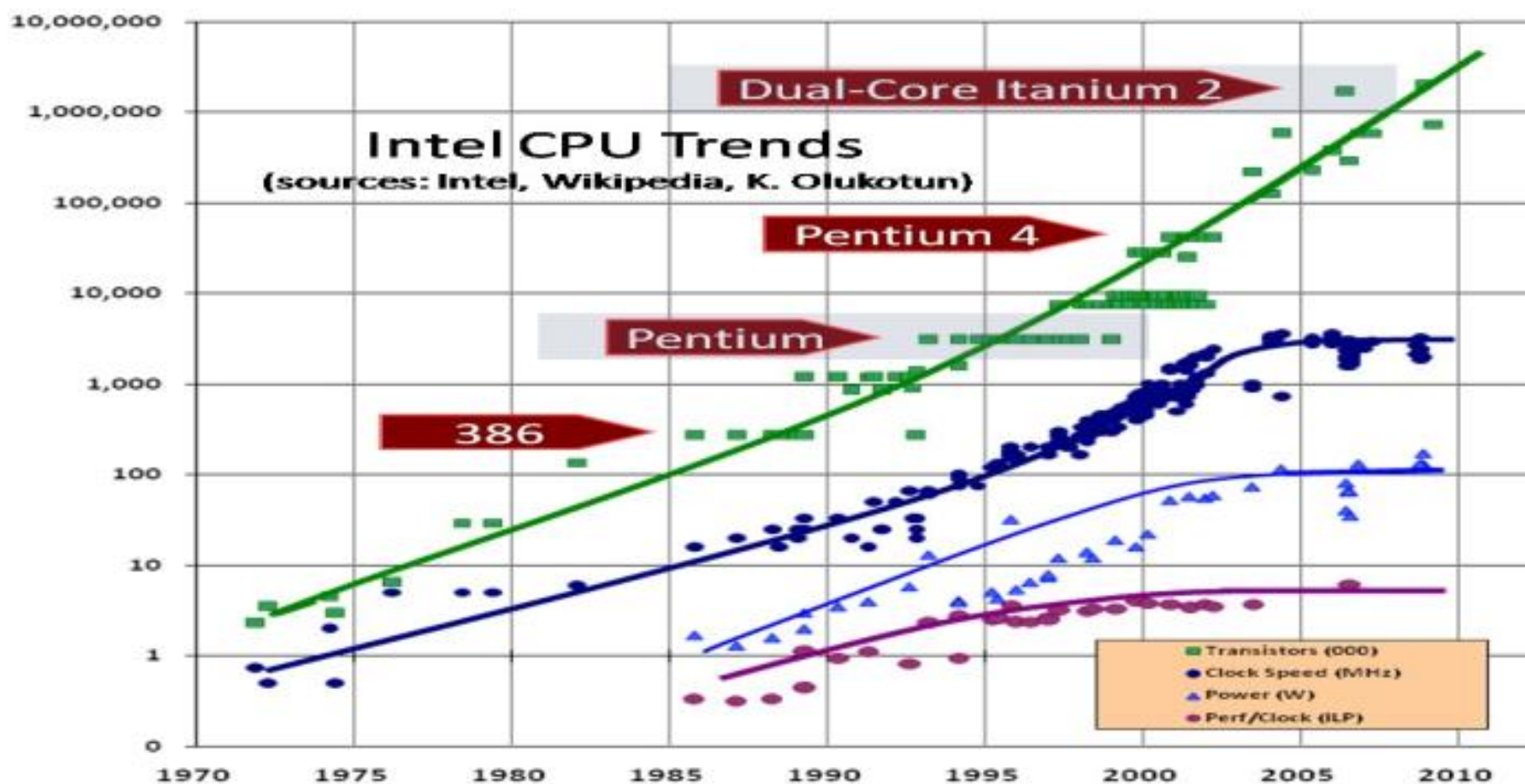
- 为何需要构建并行系统
- 什么是并行计算
- 为何要使用并行计算
- 为何需要编写并行程序
- 如何编写并行程序

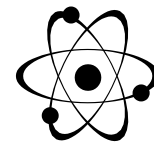
# 1. 为何需要构建并行系统

- 摩尔定律：集成电路上可容纳的晶体管数目，约每隔18个月便会增加一倍，性能也将提升一倍



# 1. 为何需要构建并行系统





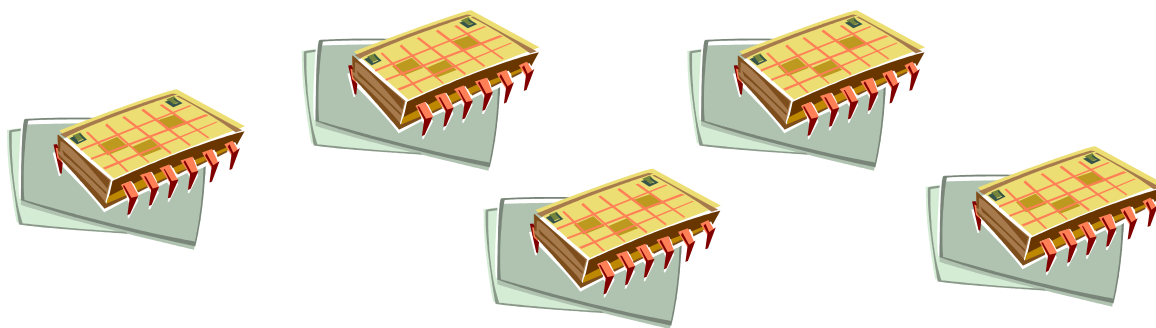
# 1. 为何需要构建并行系统

- 更小的晶体管 = 更快的处理器
- 更快的处理器 = 更大的功耗
- 更大的功耗 = 更大的散热
- 更大的散热 = 不稳定的处理器



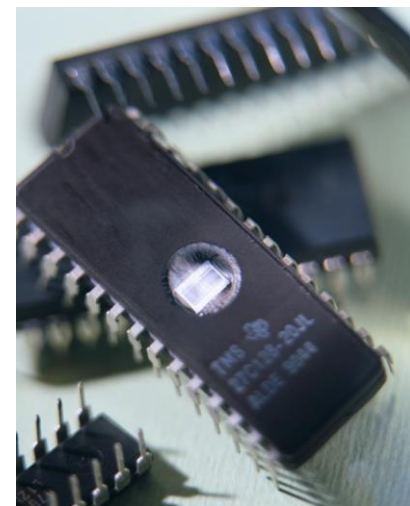
# 1. 为何需要构建并行系统

- 与其设计和制造速度更快的微处理器，不如将多个处理器放在一个集成电路上



# 1. 为何需要构建并行系统

- 从单核系统（single-core systems）转向多核处理器（multicore processors）
- “core” = central processing unit (CPU)
- 引入了并行性！！！！



# 1. 为何需要构建并行系统

## ●为何需要处理器的性能持续增长

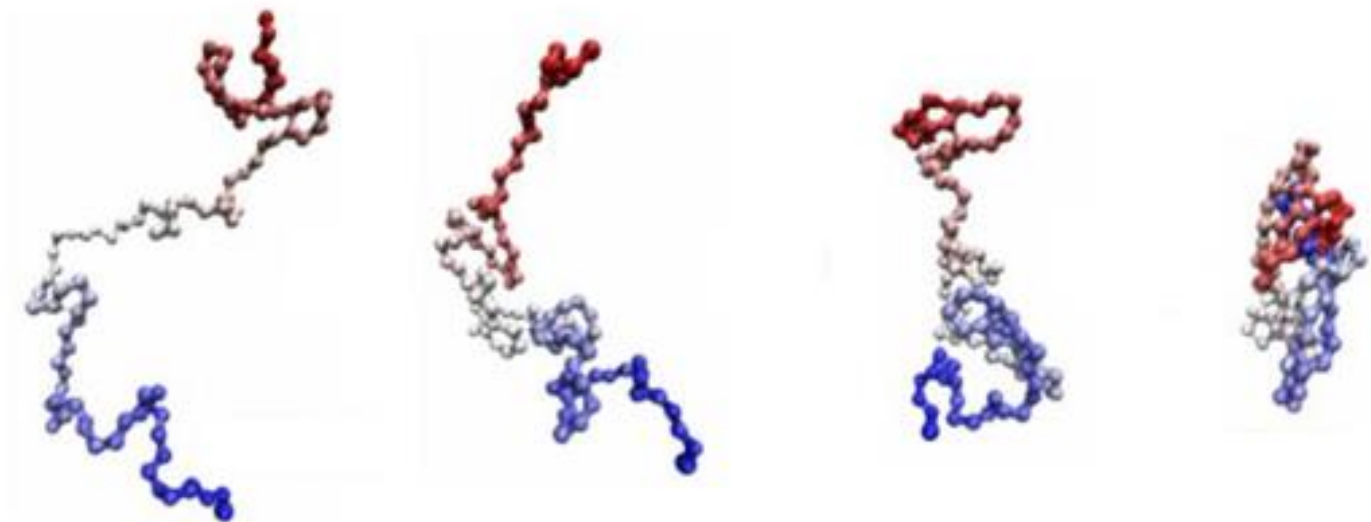
- 计算能力在增加，但我们的计算问题和需求也在增加
- 由于过去的增长，一些难题已被攻克
  - 如：解码人类基因组
- 但更复杂的问题仍有待解决

# 1. 为何需要构建并行系统



气候建模（ Climate modeling ）

# 1. 为何需要构建并行系统




蛋白质折叠（ Protein folding ）

# 1. 为何需要构建并行系统



药物发现（ Drug discovery ）

# 1. 为何需要构建并行系统



5	+2.688
0	+5.000
1	+1.500
0	+1.125
0	+1.062

数据分析（Data analysis）

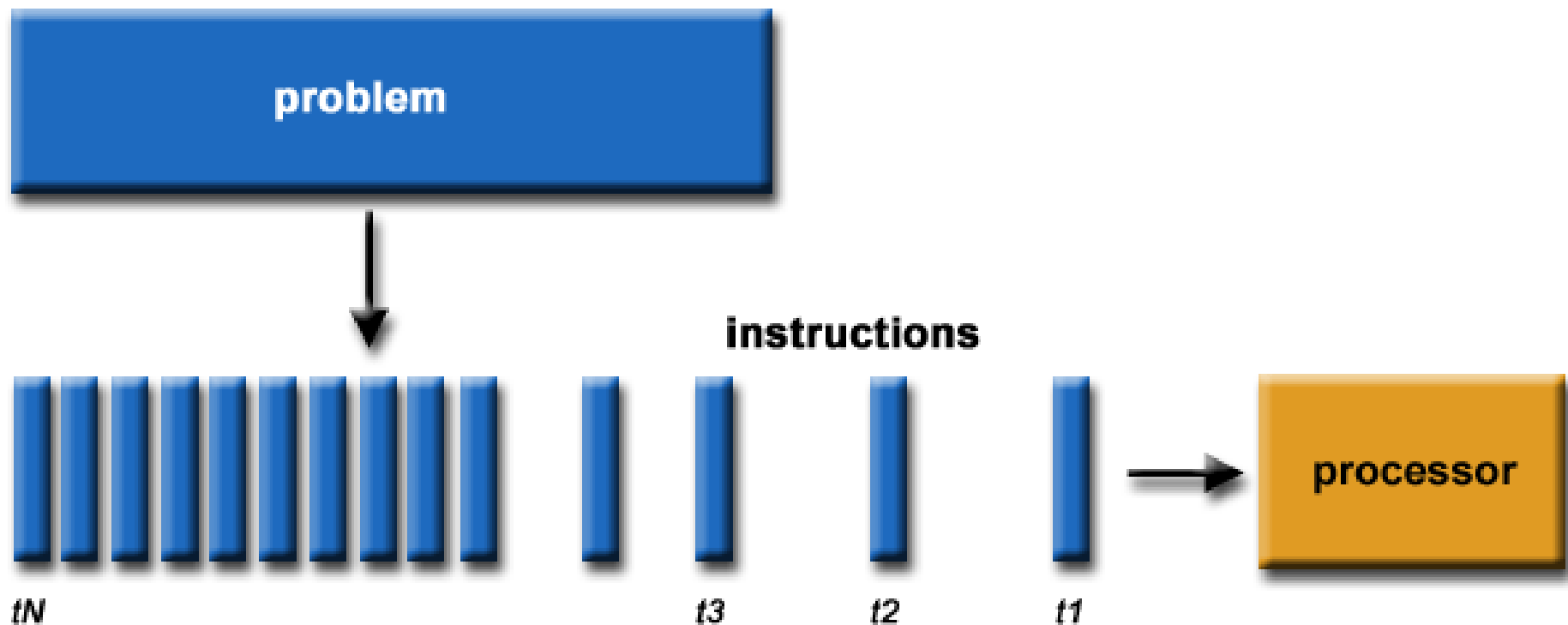
## 2. 什么是并行计算？

- 串行计算（Serial Computing）
  - 问题被分解为离散的指令序列
  - 指令被一个接一个的顺序执行
  - 指令在一个处理器上执行
  - 某一时刻，只有一个指令可以执行



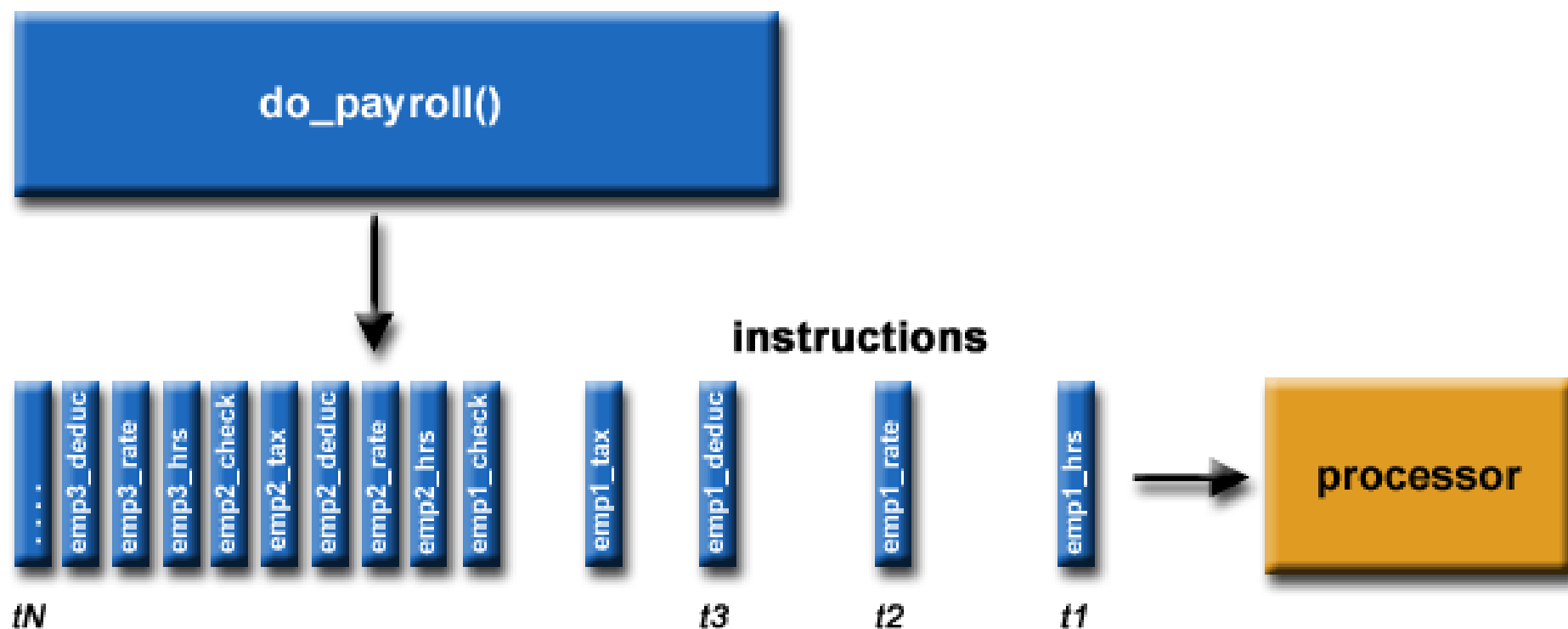
## 2. 什么是并行计算？

- 串行计算（Serial Computing）



## 2. 什么是并行计算？

- 串行计算（Serial Computing）



## 2. 什么是并行计算？

- 并行计算（Parallel Computing）

- 同时利用多个计算资源解决一个计算问题

- 问题被分解为可以被同时解决的离散部分
    - 每个部分被分解为一系列指令
    - 每个部分的指令同时在不同的处理器上执行
    - 采用整体控制/协调机制

## 2. 什么是并行计算？

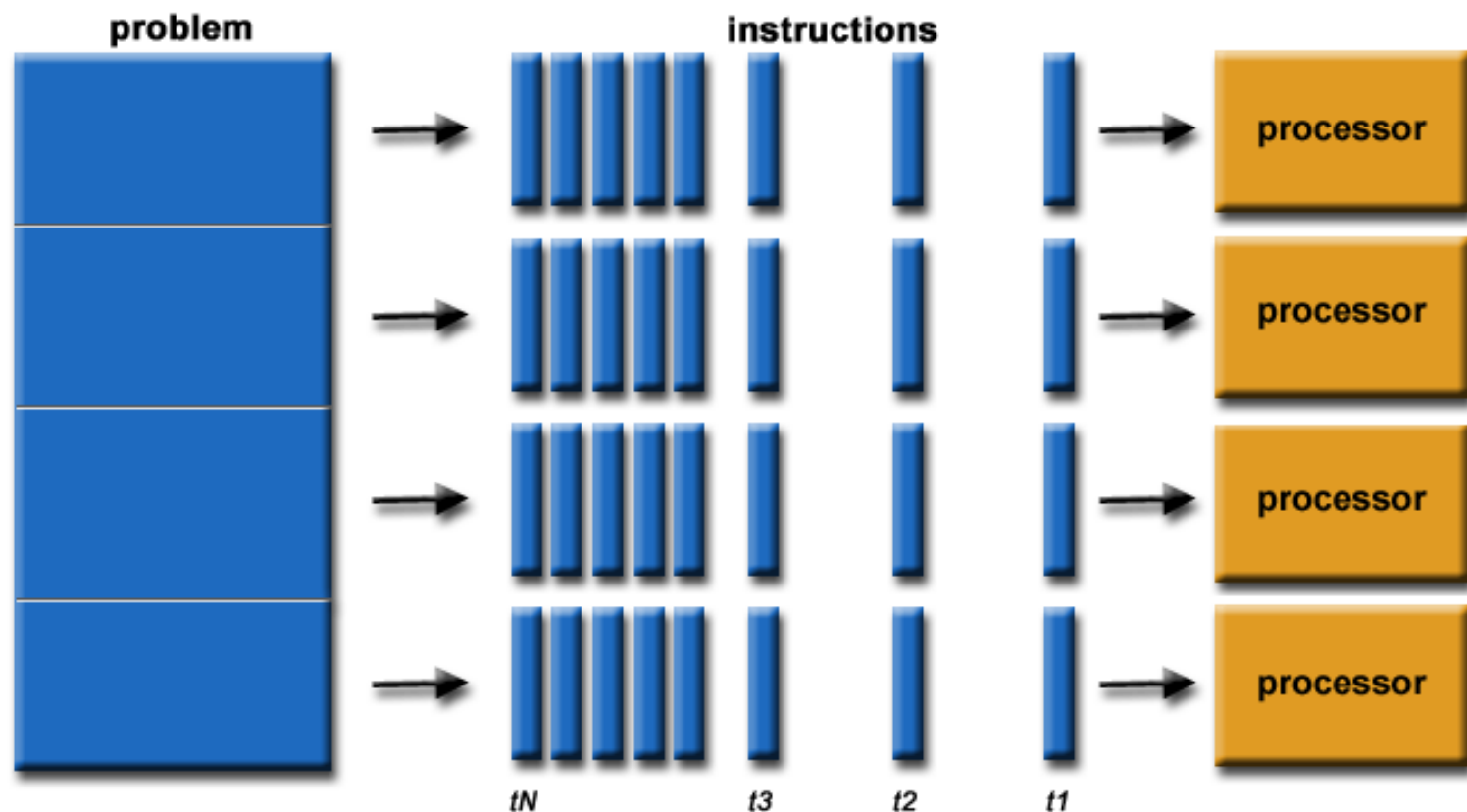
- 并行计算（Parallel Computing）

- 同时利用多个计算资源解决一个计算问题

- 每个部分可以同时求解
    - 任意时刻，多个指令在执行
    - 一台计算机，多个处理器/核
    - 多台计算机，联网

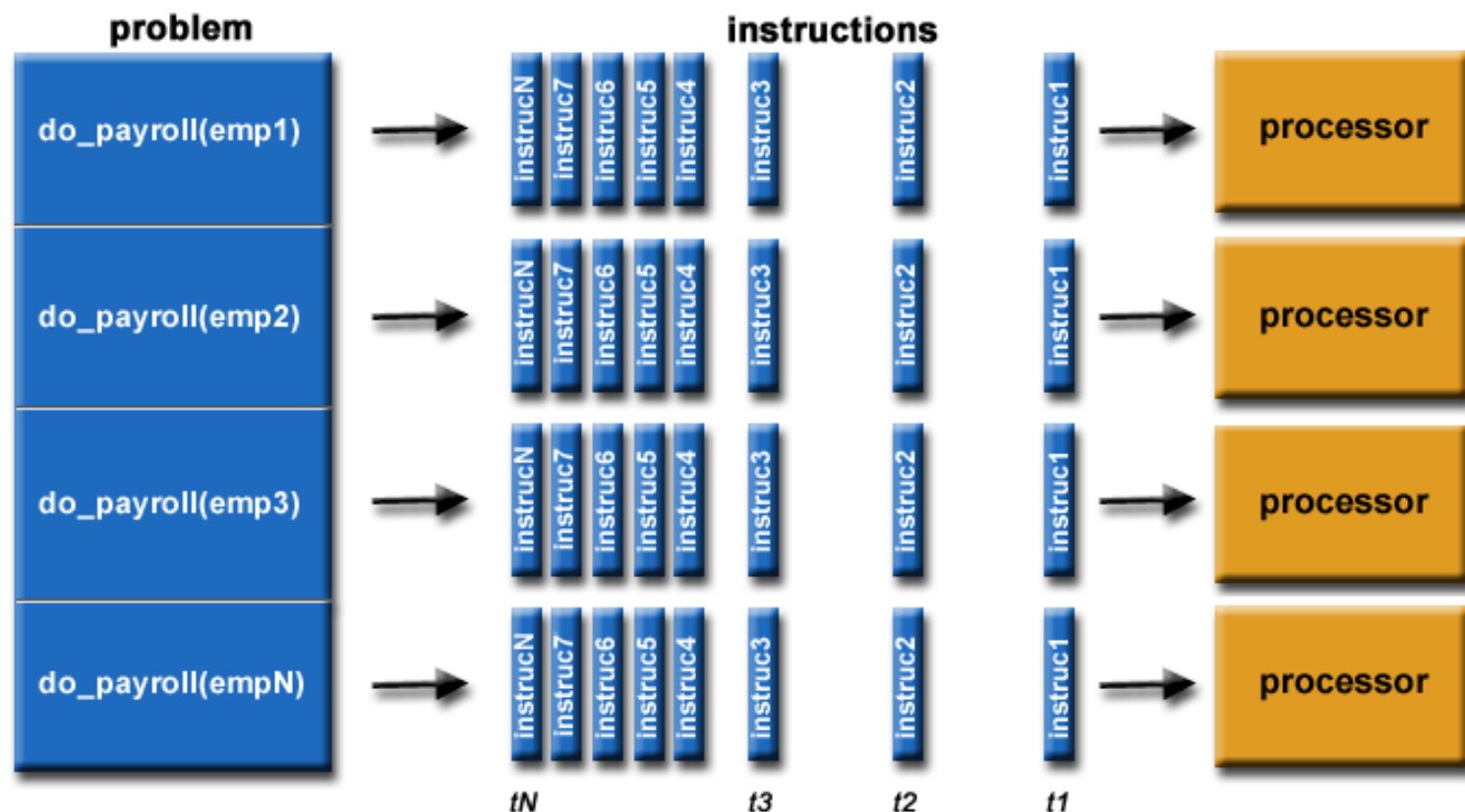
## 2. 什么是并行计算？

- 并行计算（Parallel Computing）

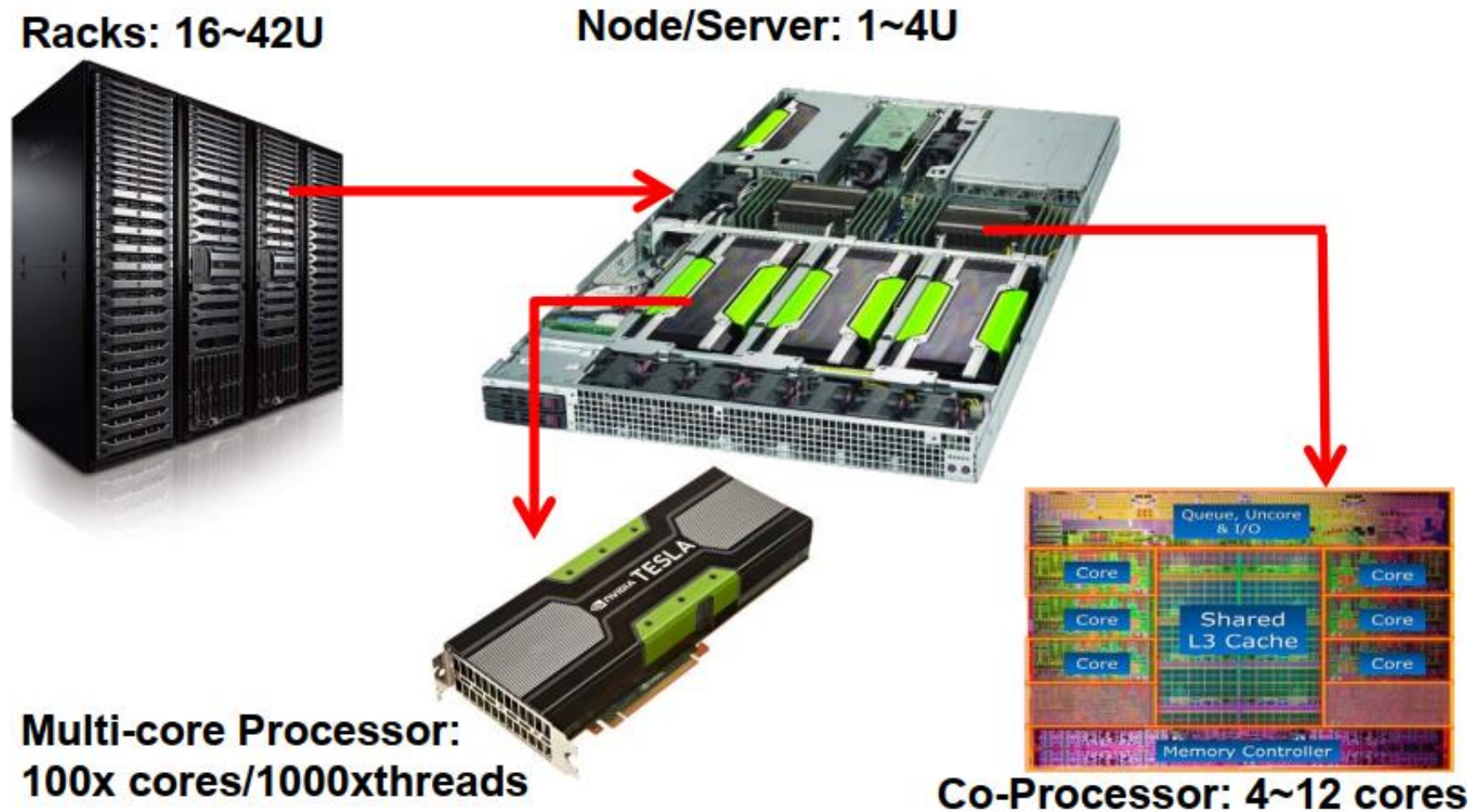


## 2. 什么是并行计算？

- 并行计算（Parallel Computing）



## 2. 什么是并行计算？



## 2. 什么是并行计算？

- 超级计算机（Supercomputers）

- 性能由FLOPS（floatingpoint operations per second）来衡量，而不是MIPS（million instructions per second）
- 1993年开始，TOP500排名（<https://www.top500.org/lists>）
- HPL（the High-Performance Linpack Benchmark）用于测试高性能计算机系统浮点性能的benchmark

$$Ax = b; \quad A \in \mathbf{R}^{n \times n}; \quad x, b \in \mathbf{R}^n$$

- 技术和财富的比拼



## 2. 什么是并行计算？

- 超级计算机（Supercomputers）

- MFLOPS（megaFLOPS）每秒一百万（ $=10^6$ ）次浮点运算
- GFLOPS（gigaFLOPS）每秒十亿（ $=10^9$ ）次浮点运算
- TFLOPS（teraFLOPS）每秒一万亿（ $=10^{12}$ ）次浮点运算
- PFLOPS（petaFLOPS）每秒一千万亿（ $=10^{15}$ ）次浮点运算
- EFLOPS（exaFLOPS）每秒一佰京（ $=10^{18}$ ）次的浮点运算

## 2. 什么是并行计算？

- 超级计算机（Supercomputers）

TOP500 List (2018 June)

	Country	System	Vendor	#cores	Accelerator	Rmax (PFLOPS)	Rpeak (PFLOPS)	Power (kW)
1	US	Summit	IBM	2.2M	V100 GPU	122	187.7	8,806
2	China	TaihuLight	NRCPC	10M		93.0	125.4	15,371
3	US	Sierra	IBM	5.6M	V100 GPU	71.6	119.2	
4	China	Tianhe-2	NUDT	5M	Matrix 2000	61.4	100.7	18,482
5	Japan	ABCI	Fujitsu	391K	V100 GPU	19.8	32.6	1,649

## 2. 什么是并行计算？

- 超级计算机（Supercomputers）

TOP500 List (2020 June)

	Country	System	Vendor	#cores	Rmax (PFLOPS)	Rpeak (PFLOPS)	Power (kW)
1	Japan	FUGAKU	Fujitsu	7.3M	415.5	513.9	28,335
2	US	Summit	IBM	2.4M	148.6	200.8	10,096
3	US	Sierra	IBM	1.6M	94.6	125.7	7,438
4	China	TAIHULIGHT	NRCPC	10.6M	93.0	125.4	15,371
5	China	Tianhe-2A	NUDT	5.0M	61.4	100.7	18,482

### 3. 为何要使用并行计算？

- 在自然界中，许多复杂的、相互关联的事件同时发生
- 与串行计算相比，并行计算更适合于建模、模拟和理解复杂的现实世界

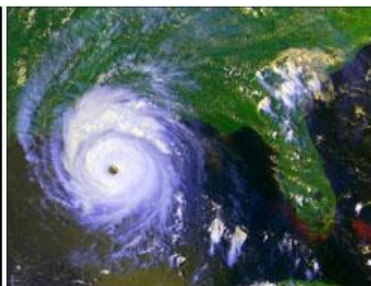
### 3. 为何要使用并行计算？



Galaxy Formation



Planetary Movments



Climate Change



Rush Hour Traffic



Plate Tectonics



Weather



Auto Assembly



Jet Construction



Drive-thru Lunch

### 3. 为何要使用并行计算？

#### ●节省时间和金钱

- 从理论上讲，在一个任务上投入更多的资源将缩短其完成时间，并可能节省成本





### 3. 为何要使用并行计算？

#### ● 解决更大、更复杂的问题

- 许多问题太大或太复杂，在一台计算机上解决它们是不切实际或不可能的
  - Web搜索引擎、数据库每秒处理数百万事务



### 3. 为何要使用并行计算？

#### ●提供并行性

- 一个计算资源一次只能做一件事。多个计算资源可以同时做很多事情
  - 网络协作





### 3. 为何要使用并行计算？

#### ● 充分利用非本地资源

- 在本地计算资源稀缺或不足时使用广域网络，甚至是Internet上的计算资源
  - SETI@home (setiathome.berkeley.edu)
  - Folding@home (folding.stanford.edu)



### 3. 为何要使用并行计算？

- 充分利用并行硬件

	DUAL XEON CPU server	DGX-1 GPU server ( 8 GPUs)
FLOPS	3TF	<b>170TF</b>
Node Mem BW	76GB/s	768GB/s
Alexnet Train Time	150 Hr	<b>2Hr</b>
Train in 2Hr	<b>&gt;250Nodes</b>	1Node

## 4. 为何需要编写并程序序

### ●对于程序员

- 如果没有意识到增加的处理器，则并没有什么帮助
- 传统的串程序序（Serial programs）大多数情况下并不能从多处理器中获益



## 4. 为何需要编写并行程序

- 在多核处理器上运行一个串行程序的多个实例通常用处不大
  - 如：运行你所喜欢的游戏的多个实例
- 你真正想要的是让它跑得更快



## 4. 为何需要编写并行程序

- 重写串行程序（Serial Programs），使其并行化
- 编写转换程序（Translation Programs），自动将串行程序转换为并行程序
  - 非常困难！
  - 一些编码结构可以被自动程序生成器识别，并转换为并行结构
  - 结果很可能是一个非常低效的程序
- 有时最好的并行解决方案是根据问题设计一个全新的算法


## 4. 为何需要编写并行程序： 举例

- 计算n个值并将它们相加
- 串行解决方案：


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

## 4. 为何需要编写并行程序：举例

- 假设我们有  $p$  个核 (cores),  $p < n$
- 每个核可以执行部分累加操作 ( $n/p$  个值)



```
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;
}
```



每个核使用自己的私有变量，执行独立于其他核的代码块

## 4. 为何需要编写并行程序：举例

- 每个核执行完代码后，其私有变量 *my\_sum* 包含调用 *Compute\_next\_value* 并累加得到的值
- 如：  $p = 8, n = 24$ , 调用 *Compute\_next\_value* 返回值为：

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



## 4. 为何需要编写并行程序：举例

- 当所有的核计算完其私有的 *my\_sum* 后，它们可以将结果发送给指定的“主”核（*master core*）来得到最终全局累加和（*global sum*）

```
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;  
}
```

# 4. 为何需要编写并行程序： 举例

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

## 4. 为何需要编写并程序：例子

*有没有更好的方法来计算全局累加和呢？*



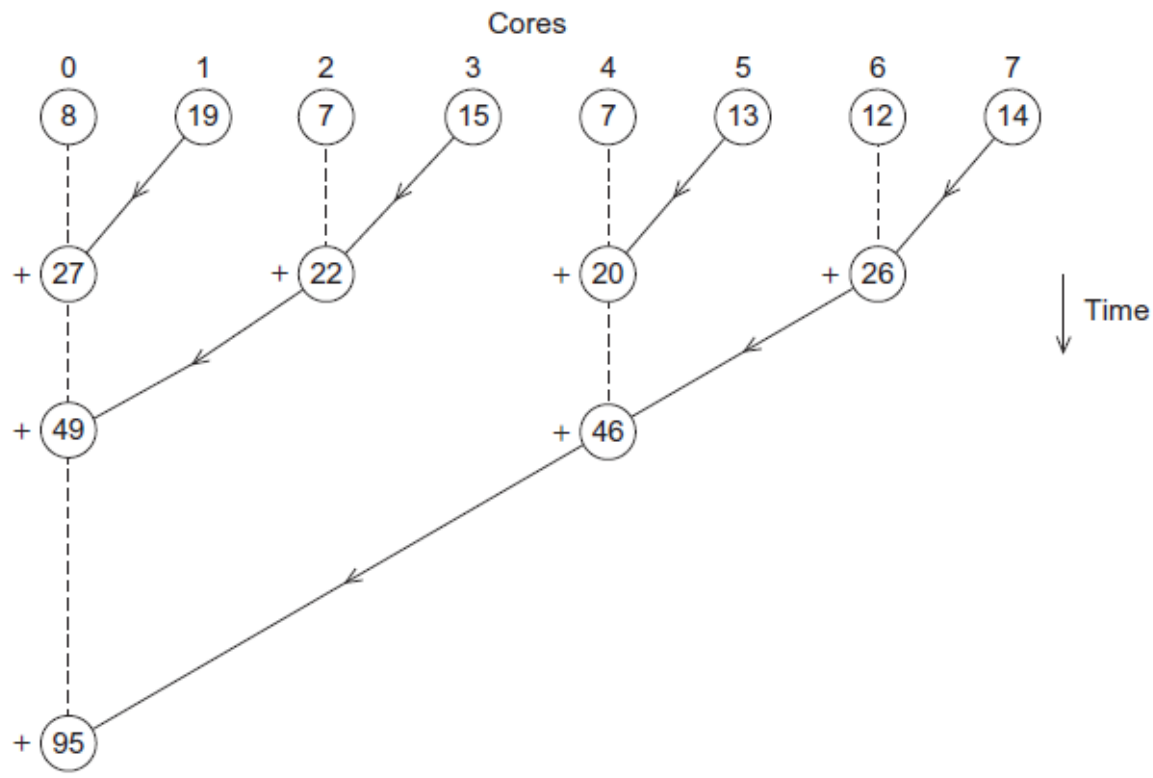
## 4. 为何需要编写并程序：例子

### ●更好的并行算法

- 不要让 master core 做所有的工作，将它的工作分给其他 core
- 将 core 进行配对，core 0 将其结果与 core 1 累加
- core 2 将其结果与 core 3 累加，以此类推
- 对偶数核得到的结果重复上述过程
- core 0 将其结果与 core 2 累加，core 4 将其结果与 core 6 累加，以此类推
- 将被4整除的核重复上述过程（core 0 将其结果与 core 4 累加），得到最终结果

# 4. 为何需要编写并行程序： 例子

## ●更好的并行算法



## 4. 为何需要编写并行程序：例子

### ●分析

- 在第一个算法中，master core 执行了 7 次接收和 7 次累加操作
- 在第二个算法中，master core 执行了 3 次接收和 3 次累加操作
- 操作减少了 1 倍

## 4. 为何需要编写并行程序：例子

### ●分析

- 随着核的数量增加，这种性能差异会更加明显（假如我们有 1000 个核）
- 在第一个算法中，master core 执行了 999 次接收和 999 次累加操作
- 在第二个算法中，master core 执行了 10 次接收和 10 次累加操作
- 性能提升了接近 100 倍

## 5. 如何编写并行程序

- 任务并行（Task parallelism）

- 对任务进行划分，并分配给各个 core 来执行

- 数据并行（Data parallelism）

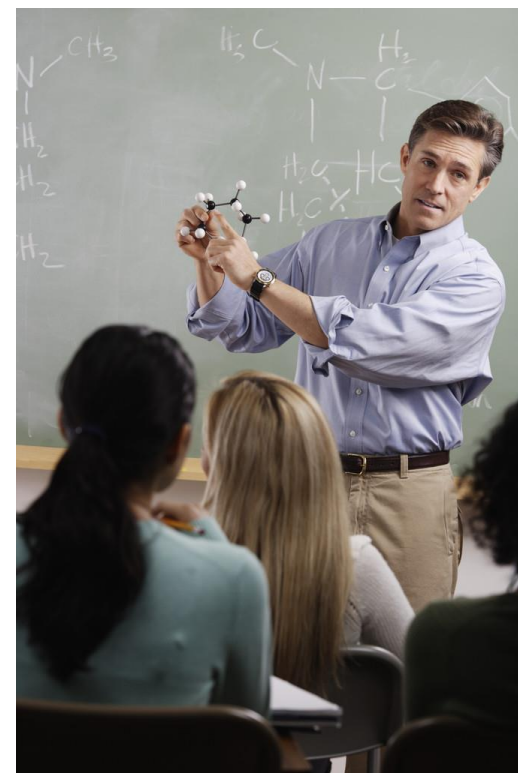
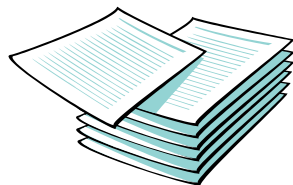
- 对数据进行划分，并分配给各个 core
- 每个 core 在其数据上执行相似的操作



## 5. 如何编写并程序

### ●教师批改试卷

- 每份试卷 15 个问题
- 共有 300 份试卷



## 5. 如何编写并行程序

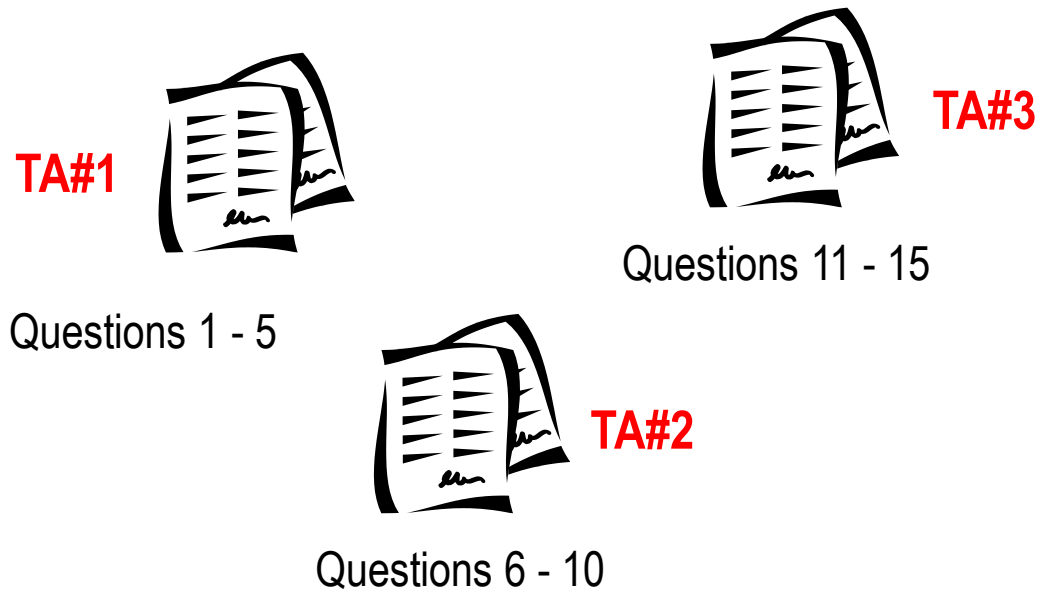
### ●教师批改试卷



## 5. 如何编写并行程序

### ●教师批改试卷

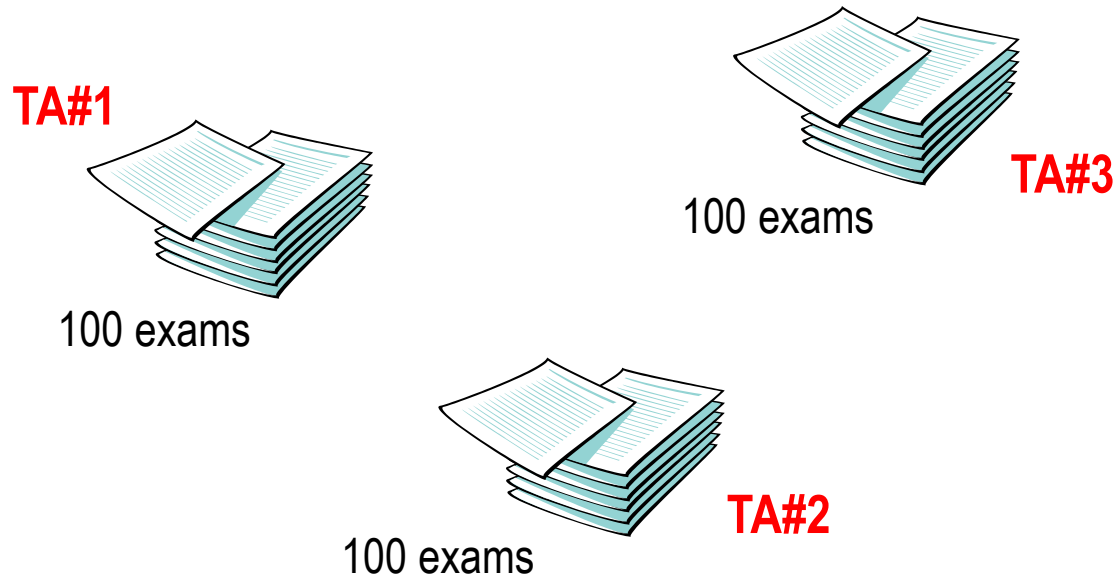
#### ➤任务并行（Task parallelism）



## 5. 如何编写并行程序

### ●教师批改试卷

#### ➤数据并行 (Data parallelism)



## 5. 如何编写并行程序

### ●数据并行（Data parallelism）

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

## 5. 如何编写并行程序

### ●任务并行（Task parallelism）

```
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. 接收并累加结果
2. 向 master core 发送结果

## 5. 如何编写并行程序

### ●协作（Coordination）

- 核之间通常需要进行协作、协调它们的任务
- 通信（Communication）：一个或多个核发送它们的结果给另一个核
- 负载均衡（Load balancing）：将工作均匀的分配给各个核，避免一些核很忙，一些核很闲
- 同步（Synchronization）：在进行某些操作时，需要统一各个核的步伐，不能有些太快，有些太慢

## 6. 本课程学习什么

- 学习如何编写并行程序
- 使用三个对 C 语言的扩展
  - Message-Passing Interface (MPI)
  - Posix Threads (Pthreads)
  - OpenMP



# 并行系统的类型

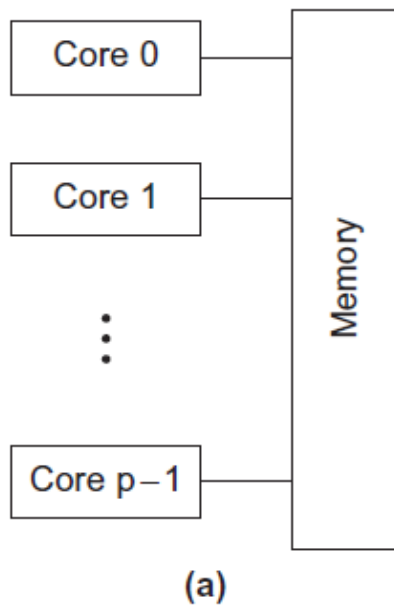
- 共享内存（Shared-memory）

- 各个核共享对内存的访问
- 需要协调各个核对共享内存的访问

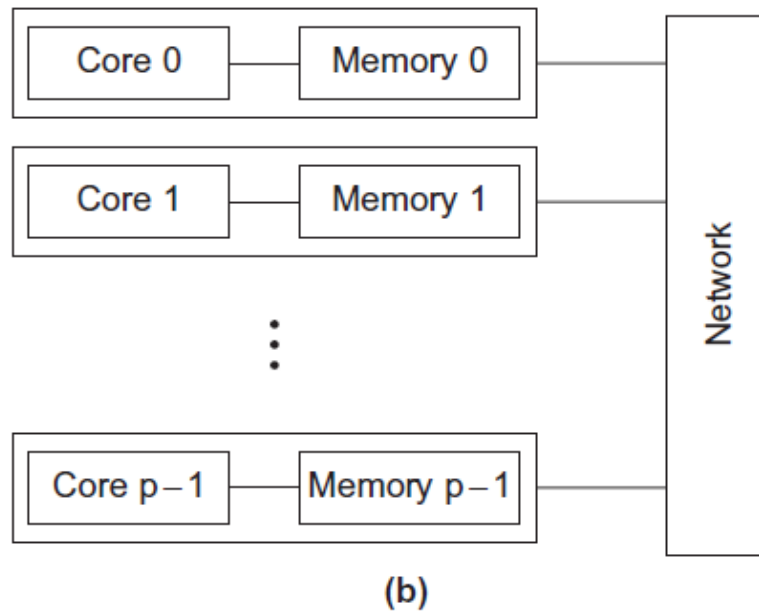
- 分布式内存（Distributed-memory）

- 每个核拥有自己的私有内存
- 核之间需要通过发送消息进行通信

# 并行系统的类型



Shared-memory



Distributed-memory

## 7. 几个术语

- Concurrent computing – a program is one in which multiple tasks can be in progress at any instant.
- Parallel computing – a program is one in which multiple tasks cooperate closely to solve a problem
- Distributed computing – a program may need to cooperate with other programs to solve a problem.

## 7. 几个术语

### ● 并发 vs. 并行

- 并发：是指多个线程任务在同一个CPU上快速地轮换执行，一种逻辑上的同时进行；
- 并行：是指多个线程任务在不同CPU上同时进行

# 7. 几个术语

- 并发 vs. 并行

## Concurrent and Parallel Programming

05 Apr 2013

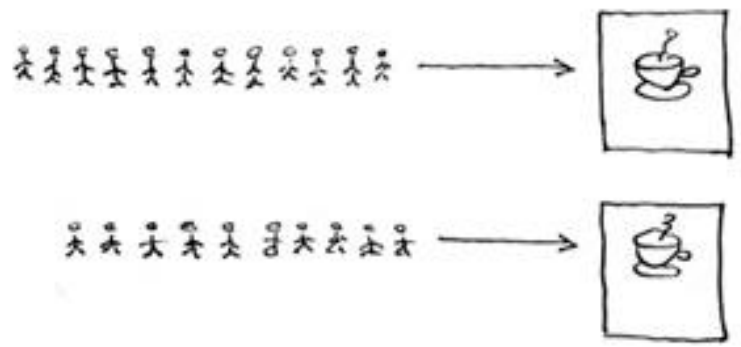
What's the difference between concurrency and parallelism?

Explain it to a five year old.

Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines

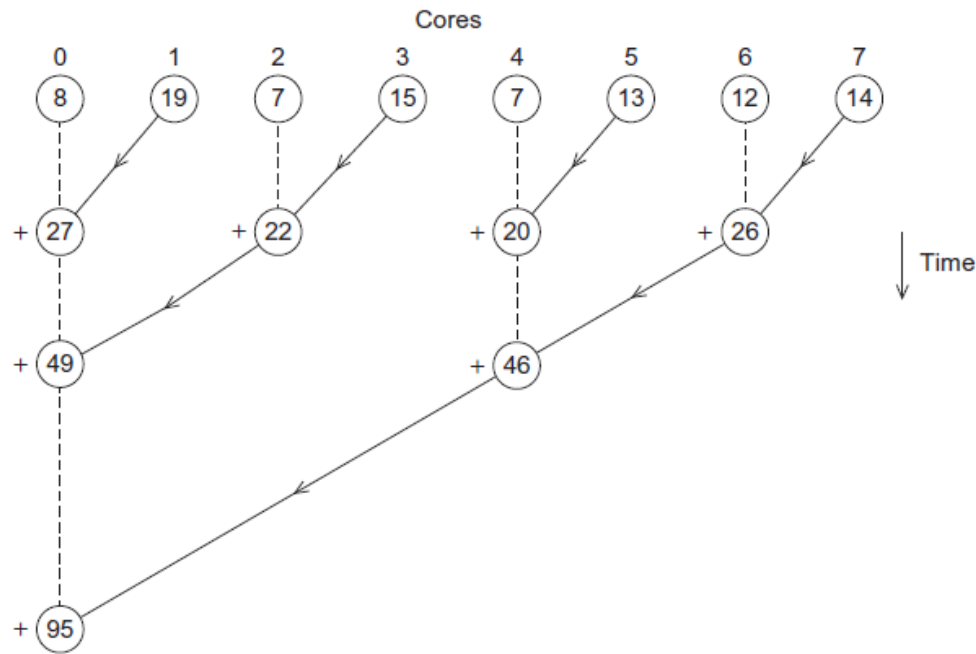


## 小结

- 物理定律将我们引向多核技术
- 串行程序通常不能从多核处理器中受益
- 从串行程序代码自动生成并行程序并不是获得高性能的最有效方法
- 编写并行程序包括学习如何协调各个核之间的数据、速度等
- 并行程序通常非常复杂

# 思考题

- 写出树形结构求和的伪代码
  - 假设core的数目为2的幂次（如：1,2,4,8,...）



## 思考题

### ● 写出树形结构求和的伪代码

- 提示：使用变量 `divisor` 来决定一个core是否应该发送其sum，还是接收并求和
- `divisor` 应该从 2 开始，并且每次迭代翻倍
- 变量 `core_difference` 决定当前 core 的 partner，从1开始，每次迭代翻倍
- 如：第一次迭代， $0 \% divisor = 0$  和  $1 \% divisor = 1$ ，所以 0 接收并累加，1 发送， $0 + core\_difference = 1$  和  $1 - core\_difference = 0$ ，所以 0 和 1 是一对





群名称: 18级并行计算  
群 号: 617564372