太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

高性能计算

岳俊宏
E-mail:yuejunhong@tyut.edu.cn; Tel:18234095983


并行系统与
并行程序

并行发展关键时刻

- 从1986-2002，微处理器发展的速度堪比火箭,每年性能增加50%.
- 从2002年之后，每年只能增长20%.

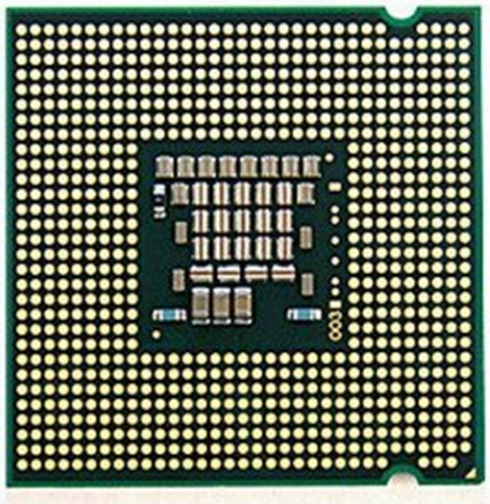
并行系统与并行程序

- 为什么需要建立并行系统
- 为什么需要编写并行程序

为什么需要构建并行系统

单处理器性能大幅度提升的主要原因之一，是日益增加的集成电路晶体管密度。

但是这种方法不再可行



为什么需要构建并行系统

- 较小的晶体管（更密集晶体管集成电路）=更快的处理器。
- 更快的处理器=增加功耗。
- 增加功耗=增加热量。
- 增加热量=不可靠的处理器.

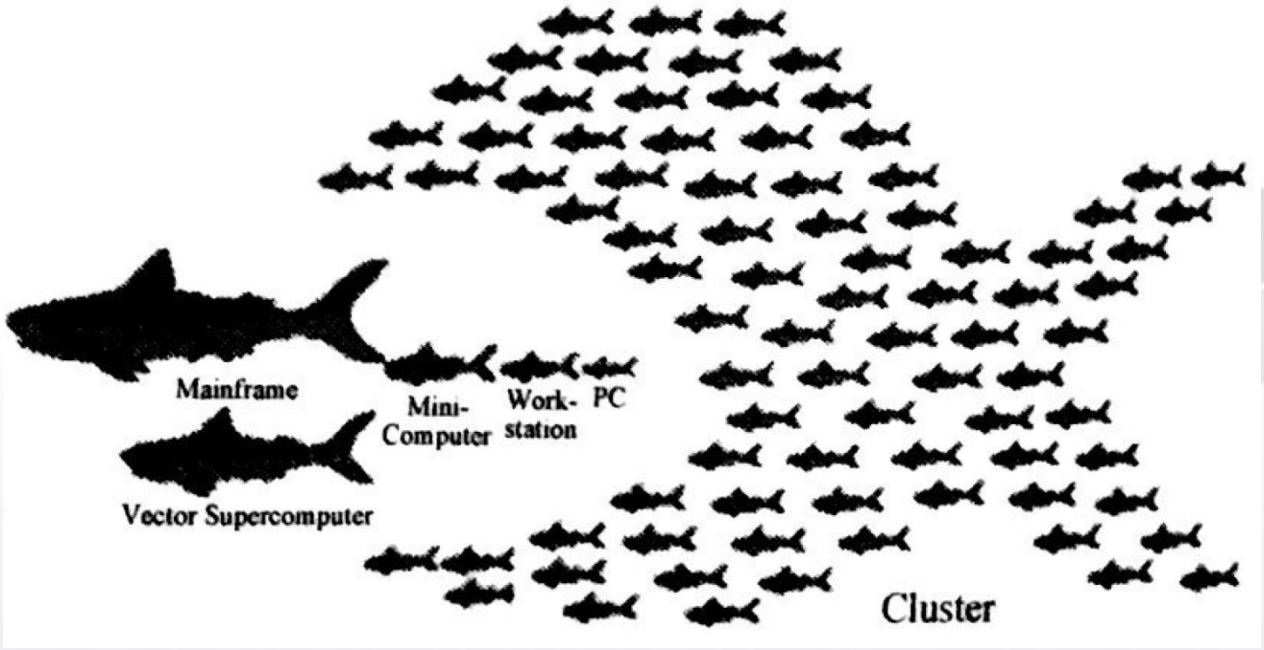
- 怎么办？？？

解决方案

- 将多个核放在一个集成电路上，来取代设计更快的单核微处理器。
- 从单核系统转移到多核处理器。
- 多核处理器=中央处理单元（中央处理器,CPU）

并行系统

- 单个计算机性能的不断提升：个人电脑、工作站、大型电脑、一些向量型超级计算机



- 集群：分布式集群的计算环境

为什么需要编写并行程序

- 如果程序员要是没有意识到或不会使用它们的话，使用具有再多核的处理器也没有任何意义。

- 大多数时候串行程序并不能在这种设计方案上得到更快的运行速度.

为什么需要编写并行程序

- 运行多个串行程序实例，通常不是很有用的
- 想想你最喜欢的游戏运行多个实例。
- 什么是你真正想要的？是它跑得更快，图像更加逼真

如何解决串行问题

- 将串行程序重写成并行程序
- 编写翻译程序，自动将串行程序转换为并行程序。

问题及方案

- **效率非常低效**：一些编码结构可以被自动程序生成器所识别，并转换为并行结构。
- **并行解决方案**：一步步回溯，设计完全新的算法。

实例

- 计算n个数的值并求累加和.
- 串行解法:

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

实例

- 假设我们有p个核
- 每个核计算n/p个数

考虑n能被p整除的情况

```
my_count = n / p;
my_first_i = my_rank * my_ count;
my_last_i = my_first_i + my_ count;
```

```
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;
}
```

Each core uses it's own private variables and executes this block of code independently of the other cores.

实例

- 每个核都执行完代码后，变量my_sum中就会存储调用compute_next_value获得的值的和。
- 例如，8个核，n = 24，
- 调用compute_next_value获得如下值：

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

实例

- 当各个核都计算完各自的my_sum值后，将自己的结果值发送给一个指定为‘master’的核，master核将收到的部分和累加而得到全局总和.

实例

```
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;
}
```

实例

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

Global sum

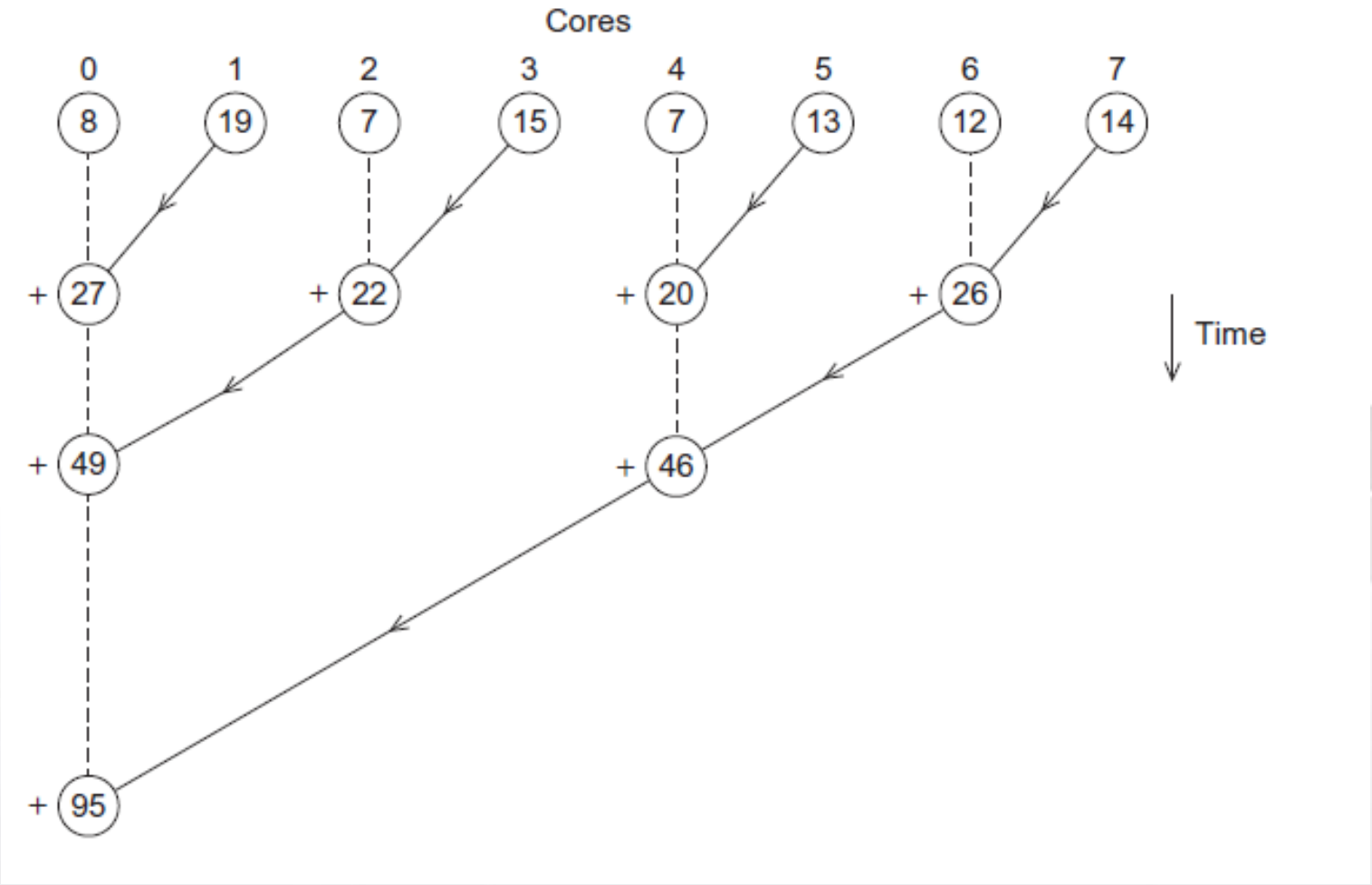
8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

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

实例

- 如果有很多很多的核的话，这样算好么？有更好的方法么？

由多核共同计算累加和



分析

- 在第一个方法里，主核需要接收7次，并做7次加法.
- 在第二个方法中，主核只要接收3次，并做3次加法.
- 性能提高了两倍，如果核数更多的话，效果更明显，1000个核，第一种需要999次接收和加法，第二种只需要10次，提高了100倍!

改进后的并行算法

- 不用主核计算所有部分和的累加工作.
- 将工作分散到其他核。
- 各个核两两结对

0号核和1号核的结果相加，2号核和3号核结果相加……

- 然后再在偶数核上重复累加和的工作

总结

微处理器	性能提升	10年	制造商	软件开发人员
1986-2002	50%	60倍	提高单处理器的性能	等待下一代微处理器的出现
2002以后	20%	6倍	到2005年，通过并行处理来快速提升微处理器的性能	串行程序在单个核上运行的，不会因增加更多的处理器就获得极大的性能提高
问题	20%很可观，为什么要不断提升性能		研制更快的单处理器，为什么要构建并行系统	为什么不编写程序将串行自动转化为并行
答案	各个领域的飞速发展，数据量的增大: ◆ 气候模拟; ◆ 蛋白质折叠; ◆ 药物发现; ◆ 能源研究; ◆ 数据分析		性能的提升是依靠不断增加晶体管的密度。集成电路太热就会变得不稳定；与其构建更快、更复杂的单处理器，不如在单个芯片上放置多个相对简单的处理器——多核处理器（CPU）	◆ 能识别的结构有限，不够智能; ◆ 效率低下

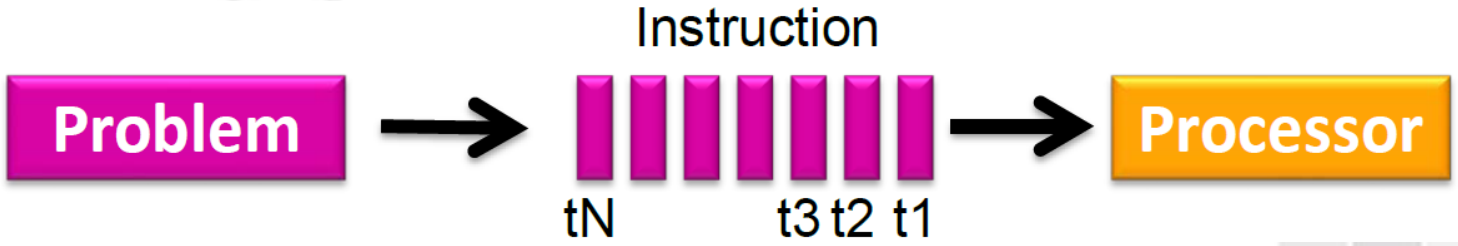

并行程序

并行程序

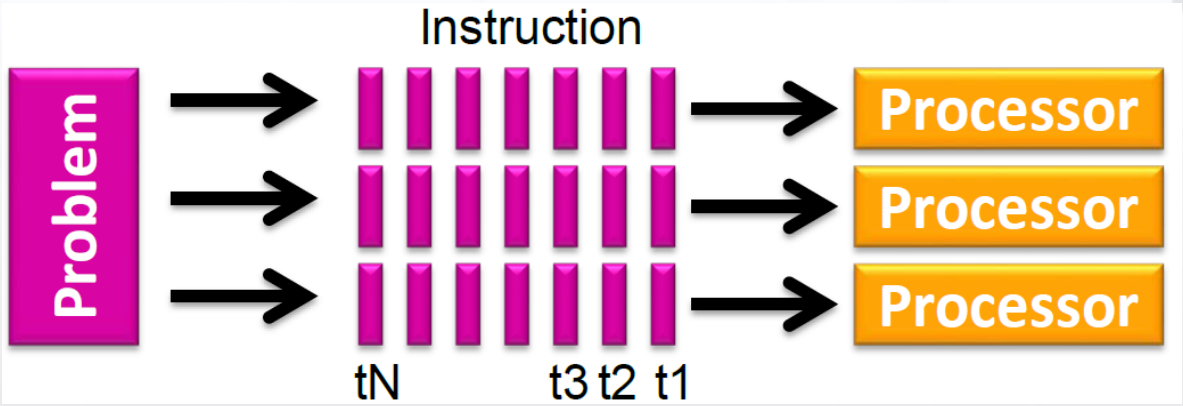
- 并行程序的定义及作用
- 并行程序的编写（任务并行和数据并行）
- 并行编程模型及发展
- 常见概念

并行程序

- **并行程序能够**使用多个核（处理器）一起工作来解决单一问题
- 传统上，程序是为串行程序编写的



- 在并行程序中，使用多台或多核计算机资源来解决一个计算问题

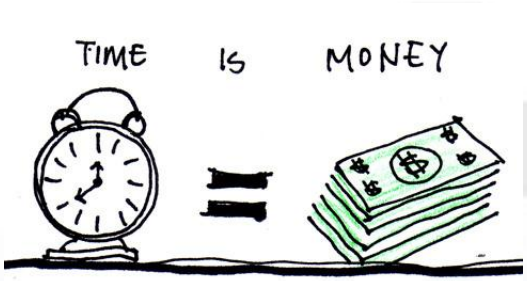


并行程序的作用

- 节省时间
  - ✓ 使用更多的资源缩短执行时间，进而可能节约成本
  - ✓ 缩短执行时间，进而允许运行更多的程序或有更多的调试机会

4 hours of work

Finish in 1 hour!!!



- 求解更大的问题
  - ✓ 可解决在单台计算机上无法解决的问题
  - ✓ 科学计算（Scientific computing）

并行程序的编写

- 任务并行

  将待解决问题的各个任务分配到各个核上执行.

- 数据并行

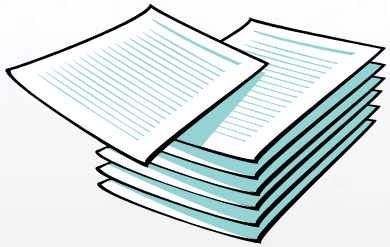
  将待解决问题需要处理的数据分配到各个核上.

  每个核在分配的数据集上执行大致相似的操作.

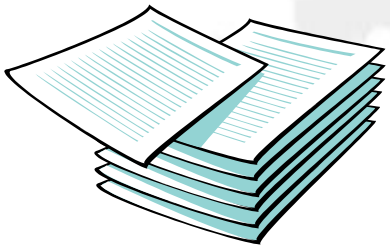
判卷子

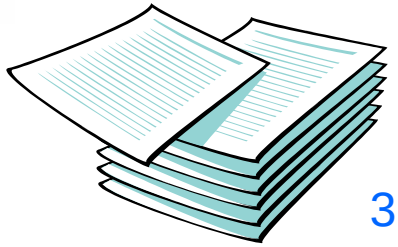
3道大题

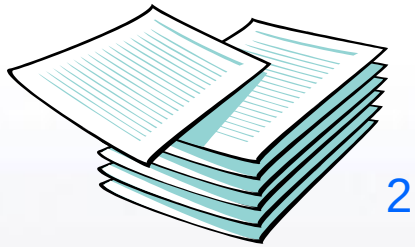
300份试卷



数据并行







每个助教100份试卷

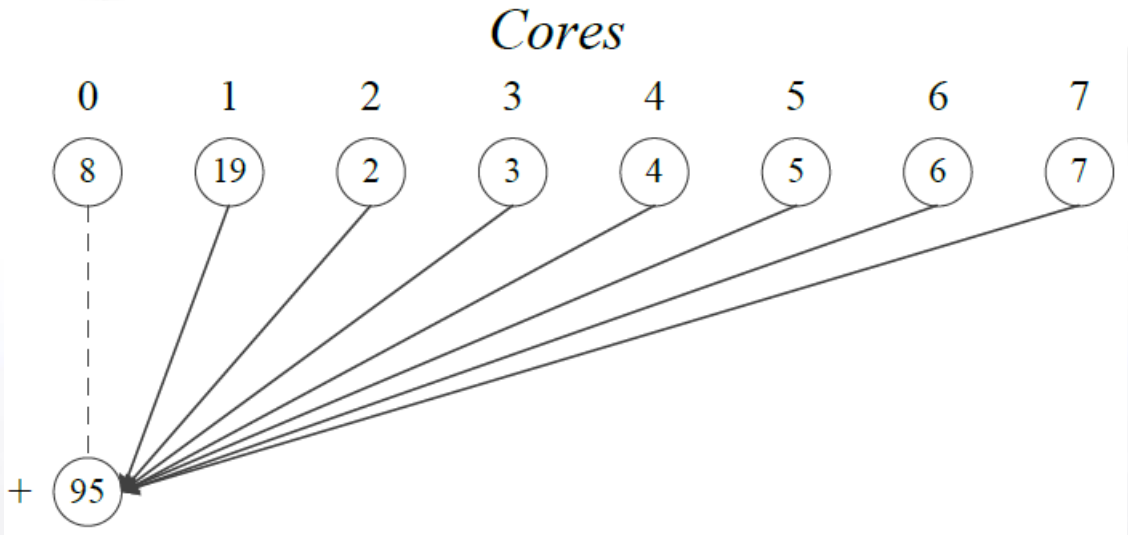
有三个助教——任务并行

第一题

第二题

第三题

如下的全局求和实例是数据并行还是任务并行？



数据并行

```
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;
}
```

任务并行

任务:

- 接收并累加部分和
- 计算各自的部分和并传递给master

```
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;
}
```

协调

- 各个核之间经常需要协调工作.
- **通信**：一个或多个核需要将自己的部分和发送给其他核.
- **负载平衡**：将任务量平均分配给各个核.
- **同步**：每个核都以自己的速度工作，核之间不会自动同步。

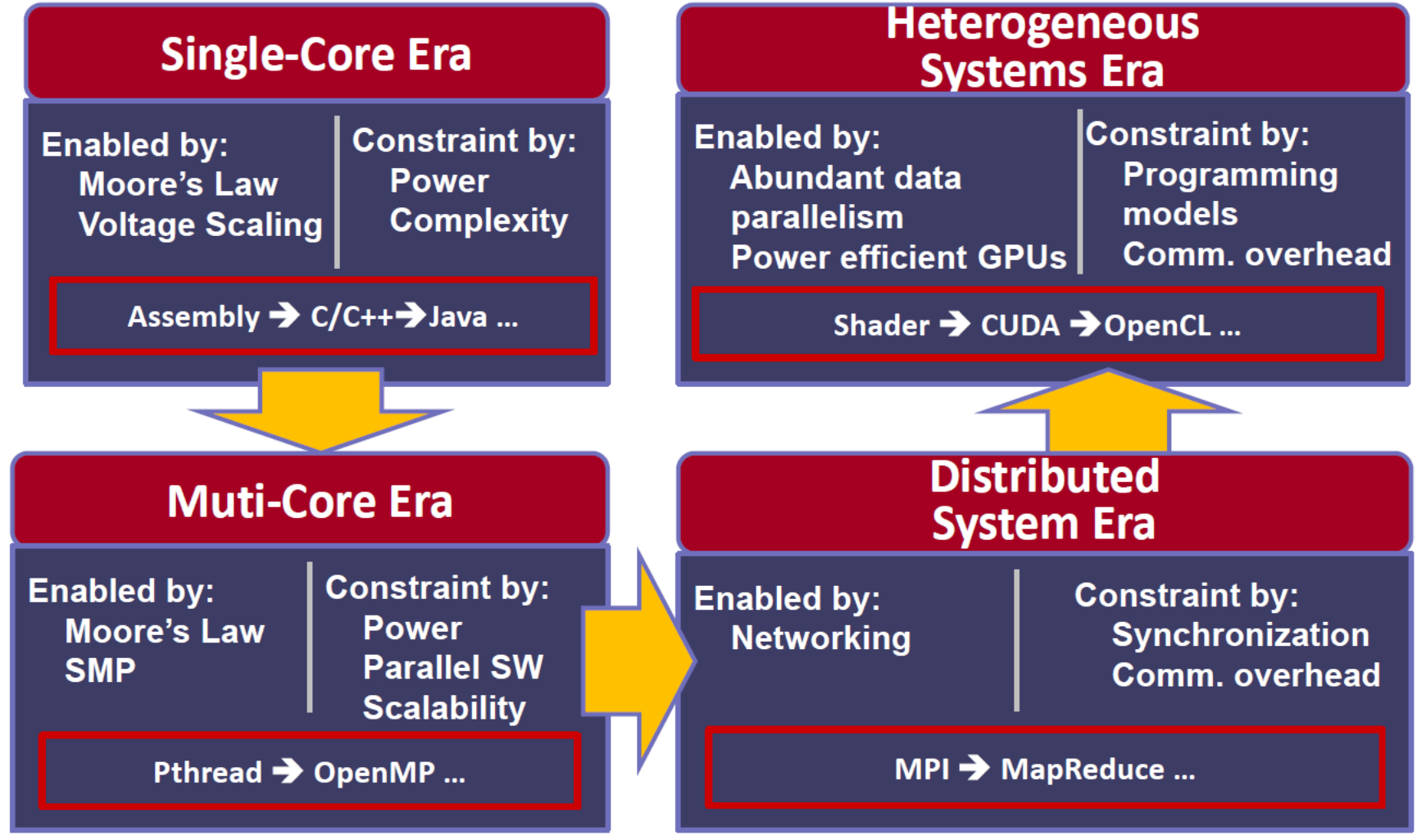
同步

- 假设需要累加的数据不再是通过计算给出，而是统一由master核读入，并存放在数据x中，

```
if(I'm the master core)
    for(my_i=0; my_i<n; my_i++)
        scanf("%lf",&x[my_i]);
```

- 在master核初始化完x数据并使数组能够被其他核访问之前，其他核不能开始工作，否则容易造成错误。
- 因此，需要在初始化x数组和计算部分和之间加入一个同步点：Synchronize_cores();

并行编程模式



From Jerry Chou, National TsingHua University

对比

	MPI	MapReduce
集群架构/容错性	分布式，容错性差	分布式，容错性好
硬件/价格/扩展性	刀片服务器、高速网、SAN，价格贵扩展性差	普通PC机，便宜，扩展性好
编程/学习难度	What-how, 难（C语言）	What,简单（Java）
应用	超级计算机性能上提升	大数据处理分析

常见概念

- 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)：

一个程序通过**多个任务紧密协作**来解决**某一个问题**；

- 分布式计算(distributed computing)：

一个程序需要与其它程序**协作**来解决某个问题；

并行计算与分布式计算都属于并发计算

并行计算与分布式计算的区别

- 相似点：都是为了实现比较复杂的任务，将大的任务分解成小的任务，在多台计算机上同时计算。
- **并行计算：**一个程序通过**多个任务紧密协作**来解决**某一个问题**；
  - ✓ **目的：**是用多个处理器去并行解决同一个问题，使得**性能和规模**增大。
    （1）faster加速求解问题的速度；（2）bigger提高求解问题的规模
  - ✓ **应用场合：**常用于科学计算，预测天气
  - ✓ **实现方式：**结构比较紧密，各节点之间通过高速网络连接；如超级计算机
  - ✓ **实时性：**并行计算每个节点的每一个任务块都是必要的，计算的结果相互影响，要求每个节点的计算结果要绝对正确，并且**在时间上做到同步**。

并行计算与分布式计算的区别

- **分布式计算：**一个程序需要与其它程序**协作**来解决某个问题；通常是一个需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。

- ✓ **目的：**在做资源的分享和共用，然后直接在系统上互相沟通传递资料。
- ✓ **应用场合：**商业应用，如云计算、穷举暴力之类的计算等
- ✓ **实现方式：**结构比较松散，各节点可能会跨越局域网，或者直接部署在互联网上，节点之间几乎不互相通信。
- ✓ **实时性：**分布式的计算被分解后的小任务互相之间有独立性，节点之间的结果几乎不互相影响，**实时性要求不高**。（并发）

作业

- 习题1

为求全局总和例子中的my_first_i和my_last_i推导一个公式。需要注意的是：在循环中，应该给各个核分配大致相同的计算元素。提示：先考虑n能被p整除的情况。

作业

习题2

在下列情况中，推导公式求出0号核执行接收与加法操作的次数。

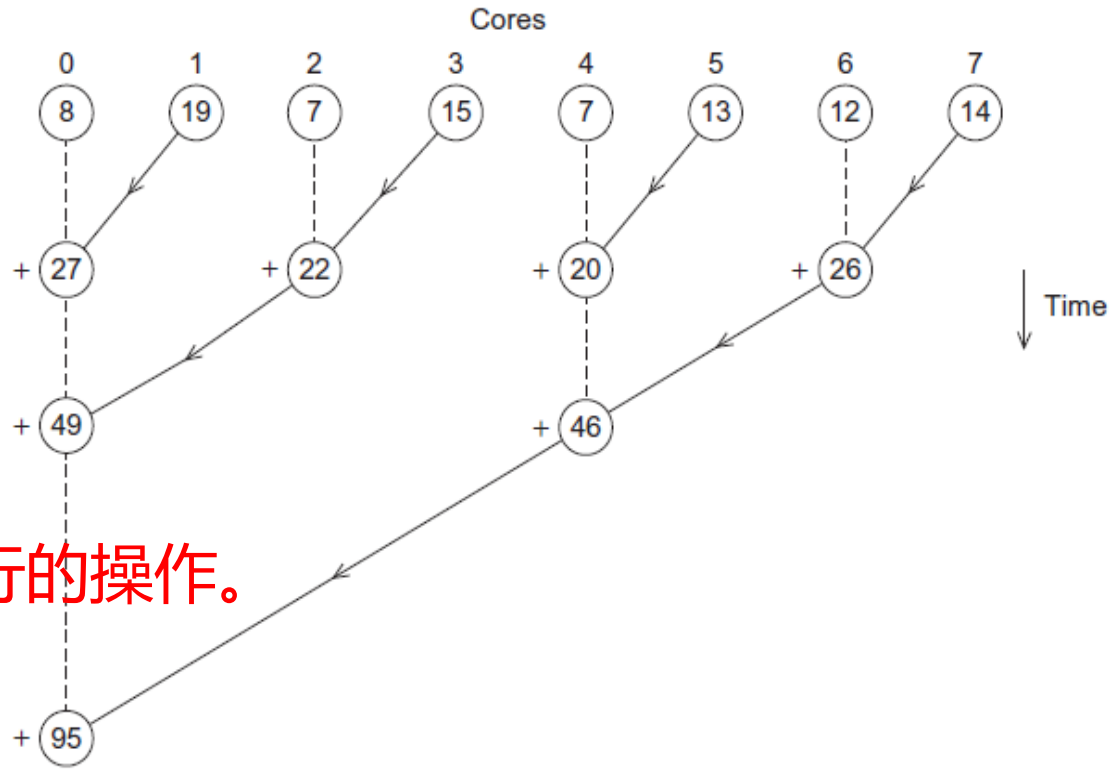
a. 最初的求全局总和的伪代码。

b. 树形结构求全局总和。

制作一张表来比较这两种算法在总核数是2,4,8,…,1024时，0号核执行的接收与加法操作的次数。

习题3

写出图2中树形结构求全局总和的伪代码。假设核的数目是2的幂（1,2,4,8,…）。



注意分析每个核所执行的操作。

作业

规律1：

第一次迭代中，对divisor=2取余为0的核负责接收，否则核负责发送；

第二次迭代中，对divisor=4取余为0的核负责接收，否则核负责发送；

第三次迭代中，对divisor=8取余为0的核负责接收，否则核负责发送；

…

规律2： partner = my_rank + core_difference

第一次迭代中，core_difference=1，核0,2,4,6负责接收，其partner为1,3,5,7；

第二次迭代中，core_difference=2，核0,4负责接收，其partner为2,6；

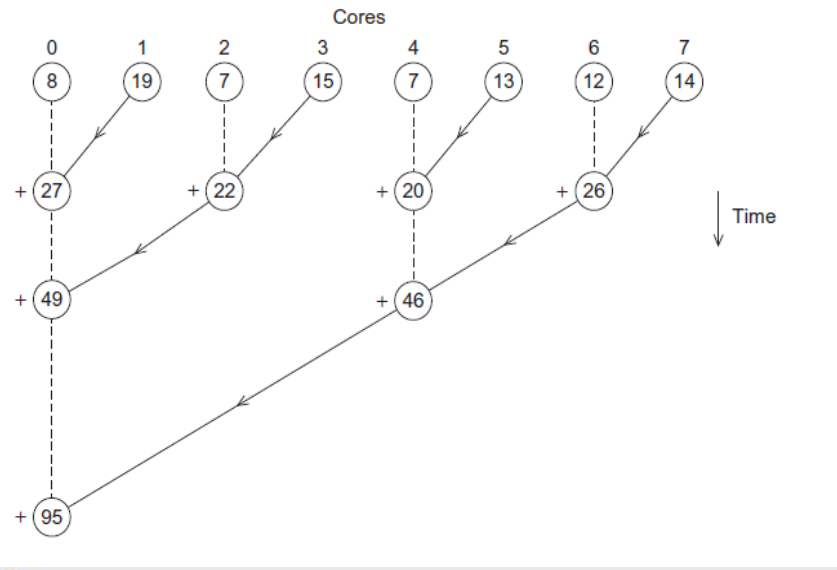
第三次迭代中，core_difference=4，核0负责接收，其partner为4；

习题4

使用C语言的位操作来实现图中树形结构求全局总和的的伪代码。

提示：1）注意每个阶段相互合作的成对的核的二进制编码。

2）可使用位异或操作或者左移操作编写伪代码来实现该算法。



作业

核	阶段		
	1	2	3
$0_{10}=000_2$	$1_{10}=001_2$	$2_{10}=010_2$	$4_{10}=100_2$
$1_{10}=001_2$	$0_{10}=000_2$	×	×
$2_{10}=010_2$	$3_{10}=011_2$	$0_{10}=000_2$	×
$3_{10}=011_2$	$2_{10}=010_2$	×	×
$4_{10}=100_2$	$5_{10}=101_2$	$6_{10}=110_2$	$0_{10}=000_2$
$5_{10}=101_2$	$4_{10}=100_2$	×	×
$6_{10}=110_2$	$7_{10}=111_2$	$4_{10}=100_2$	×
$7_{10}=111_2$	$6_{10}=110_2$	×	×

第一阶段，每个核与其二进制编号的最右位不同编号的核配对；
第二阶段，每个核与其二进制编号的最右第二位不同编号的核配对；
第三阶段，每个核与其二进制编号的最右第三位不同编号的核配对。

bitmask
001
010
100

作业

习题5

如果核的数目不是2的幂（例如3,5,6,7），那么在习题3或者4中编写的伪代码还能运行吗？修改伪代码，使得在核数未知的情况下仍然能运行。

作业

习题6

全局总和例子的第一部分（每个核对分配给它的计算值求和），通常认为是数据并行的例子；而第一个求全局总和例子的第二个部分（各个核将它们计算出的部分和发送给master核，master核将这些部分和再累加求和），认为是任务并行。

第二个树形结构的全局和例子（各个核使用树形结构累加它们的部分和），是数据并行的例子还是任务并行的例子？为什么？

习题7

全局求和实例中，已经隐含地假设每次调用Compute_next_value函数所执行的任务量与其他次调用Compute_next_value函数执行的任务量大致相同。但是，如果当i=k时调用这个函数的时间是当i=0时调用这个函数所花时间的k+1倍，应该如何使各个核分配大致相同的任务量？

i	0	1	2	3	4	5	6	7	8	9	10	11
t	2	4	6	8	10	12	14	16	18	20	22	24
Core	0			1			2			3		
T	12			30			48			66		

如果第一次（i=0）调用需要2毫秒，第二次（i=1）调用需要4毫秒，第三次（i=2）调用需要6毫秒，以此类推

高性能计算不仅仅是人工智能
高性能计算决定未来

结束!

思考

- 请分别解释并行计算和分布式计算的共同点和区别。
- 请解释传统并行计算框架MPI与大数据并行计算模型MapReduce的共同点和区别。