第1讲 什么是并行计算

本章内容

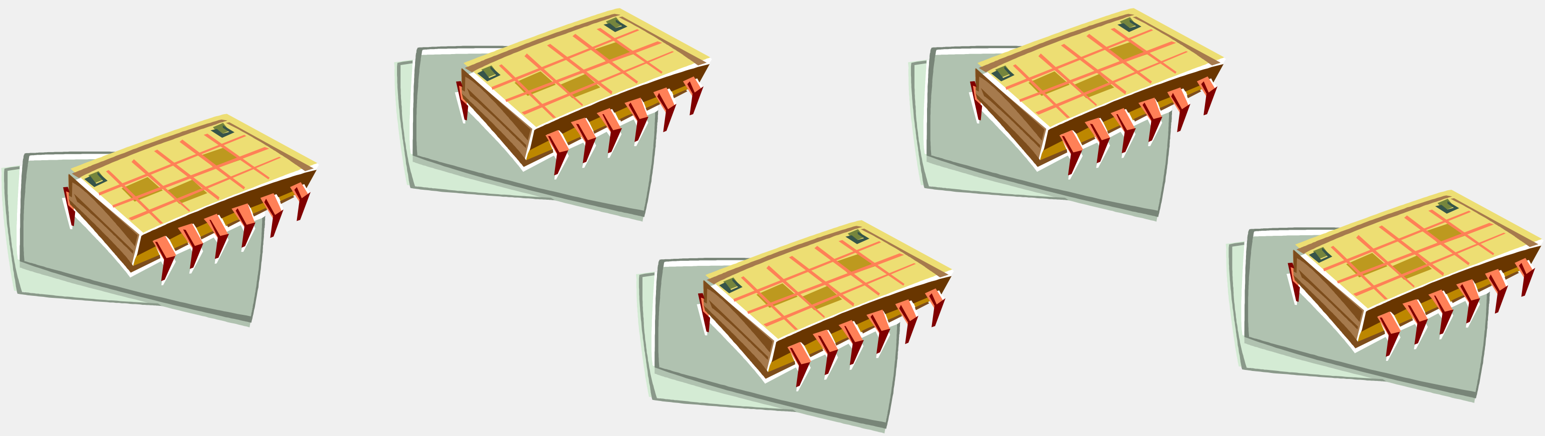
- 为什么需要不断提升的性能
- 为什么需要构建并行系统
- 为什么需要编写并行程序
- 怎样编写并行程序
- 我们将做什么
- 并发、并行、分布

变化的时代

- 1986-2002，微处理器像火箭一样增长，其性能平均每年增长50%
- 后来，平均每年性能增长降到约20%

一个好的解决方案

- CPU性能：单位时间执行指令数
- 要性能更好
  - 更快的单核处理器
  - 多核处理器——一个集成电路中有多个处理器



程序员的多处理器编程问题

- Adding more processors doesn't help much if programmers aren't aware of them…
- … or don't know how to use them.

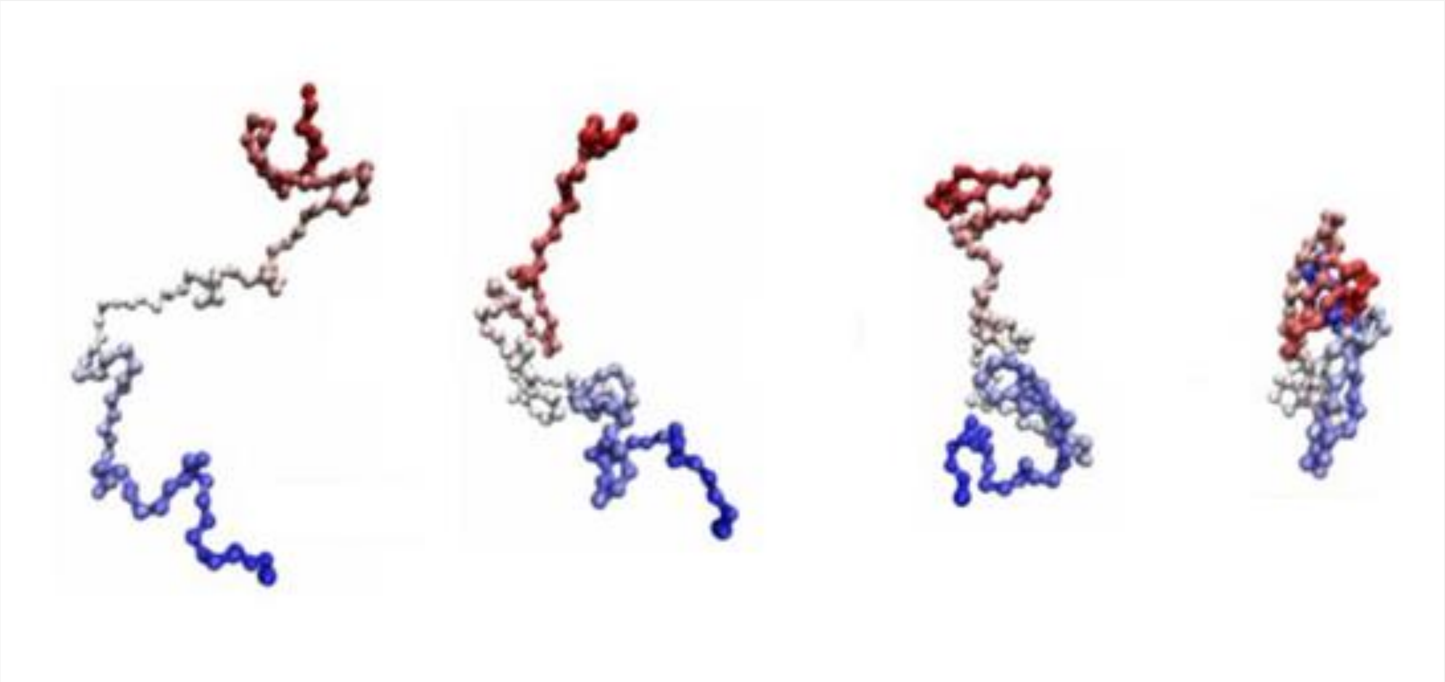
- Serial programs don't benefit from this approach (in most cases).

为什么我们需要不断增长的性能

- 计算能力在增加，而我们的计算问题和需求也是如此。
- 由于性能增加，一些未曾解决的问题得到解决，比如破译人类基因组。
- 更复杂的问题仍有待解决。

气候模型

蛋白质折叠



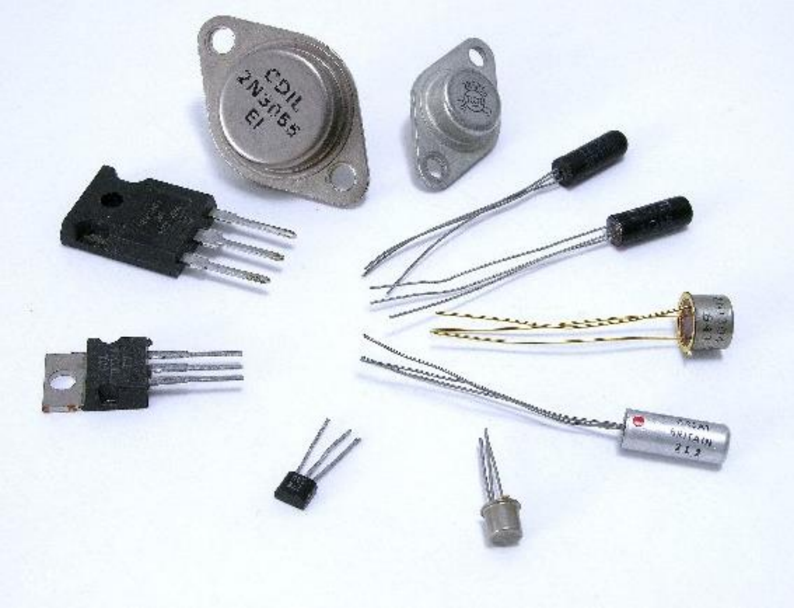
药物发现

能源研究

数据分析

为什么要构建并行系统

- Up to now, performance increases have been attributable to increasing density of transistors.到目前为止，性能的提高是由于晶体管密度的增加。
- But there are inherent problems. 但也存在内在的问题



注意这些物理知识

- 更小的晶体管 = 更快的处理器.
- 更快的处理器 = 更多的能源消耗.
- 更多的能源消耗 = 产生更多热量.
- 产生更多热量 = 不可靠的处理器.

注意这些物理知识

处理器性能 = 主频x单位时钟周期内的指令执行

- 提高处理器性能的两大途径
  - 增加处理器主频
  - 增加每个时钟周期内的指令执行数
- 单核处理器提升性能的主要途径是提升主频
  - 事实：功耗正比于主频的三次方
  - 提升主频将导致功耗快速增长

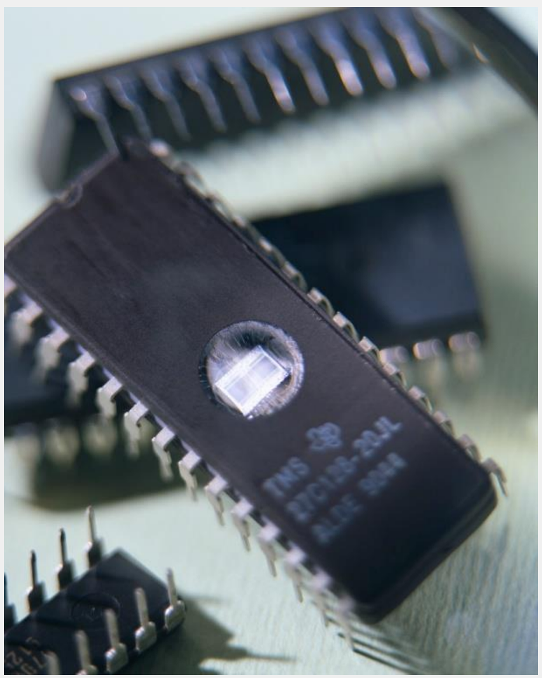
•处理器功耗正比于

电流 x 电压 x 电压 x 主频

•主频正比于 电压

解决方法

- 从单核处理器转到多核处理器.
- “核” = CPU



- 引入并行!!!
  - 多核处理器的功耗随核心数线性增长
    - 每个时钟周期内的指令执行数正比于核心数
    - “加核 = 加性能”

需要写并行程序？

- 运行串行程序的多个实例通常不是很有用。
- 考虑运行您喜欢的游戏的多个实例。

- 你真正想要的是
  – 它运行得更快。

对串行问题的处理方法

- Rewrite serial programs so that they’re parallel.

- Write translation programs that automatically convert serial programs into parallel programs.
  - This is very difficult to do.
  - Success has been limited.

More problems

- 一些编码结构可以被转换程序识别，并转换为并行结构。
- 然而，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后退一步，设计出一种全新的算法。

Example

- Compute n values and add them together.
- Serial solution:

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

Example (cont.)

- We have p cores, p much smaller than n.
- Each core performs a partial sum of approximately n/p values.

```
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.

Example (cont.)

- After each core completes execution of the code, is a private variable my_sum contains the sum of the values computed by its calls to Compute_next_value.

- Ex., 8 cores, n = 24, then the calls to Compute_next_value return:

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

Example (cont.)

- Once all the cores are done computing their private my_sum, they form a global sum by sending results to a designated "master" core which adds the final result.

Example (cont.)

```
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;
}
```

Example (cont.)

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

有更好的方法求和？

更好的并行算法

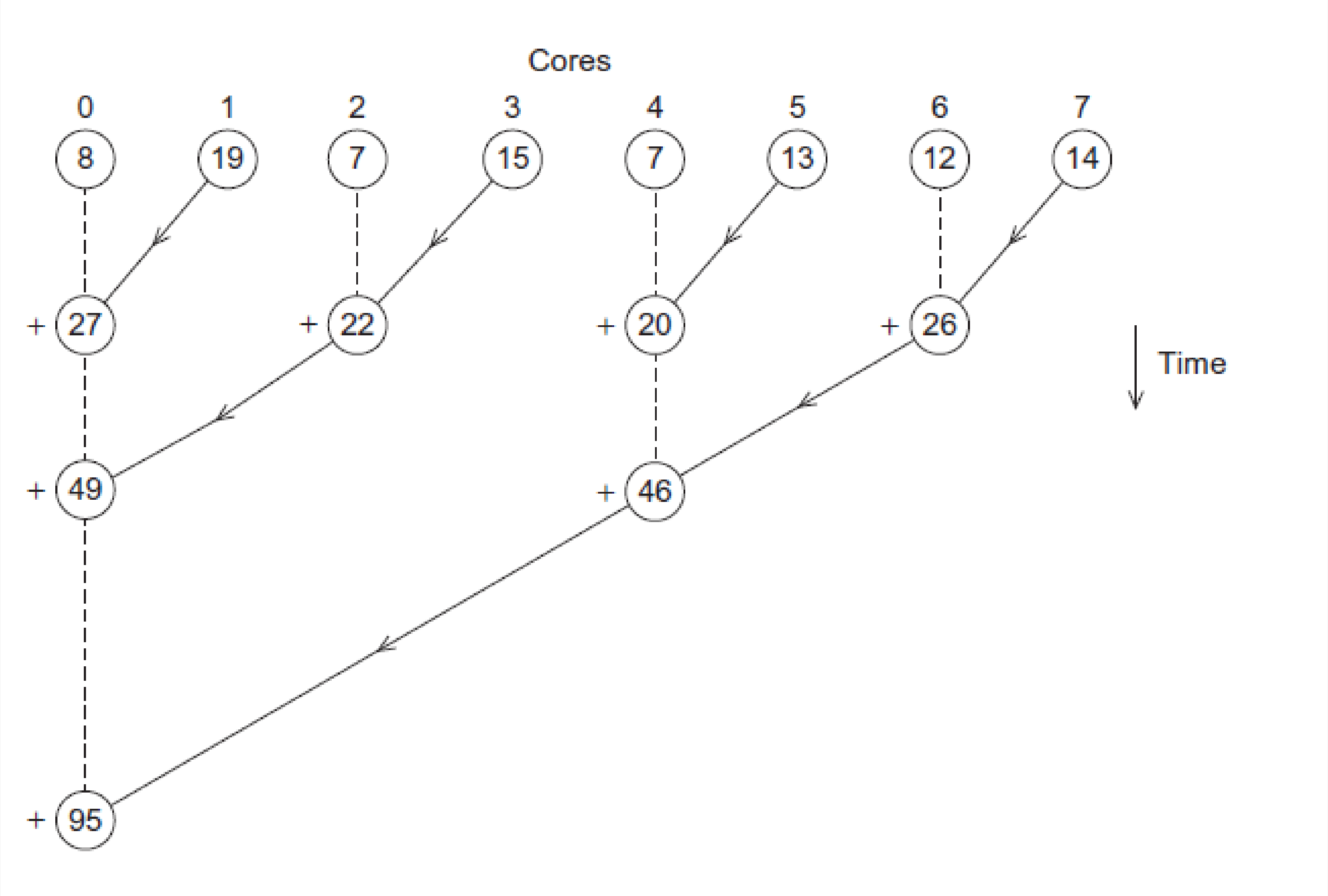
- Don’t make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1’s result.
- Core 2 adds its result with core 3’s result, etc.
- Work with odd and even numbered pairs of cores.

更好的并行算法(cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.

- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.

多核协同求和



分析

- In the first example, the master core performs 7 receives and 7 additions.
- In the second example, the master core performs 3 receives and 3 additions.
- 改进了2倍多！

分析 (cont.)

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
  - The first example would require the master to perform 999 receives and 999 additions.
  - The second example would only require 10 receives and 10 additions.

- 这加速100倍

如何写并行程序?

并行化算法：

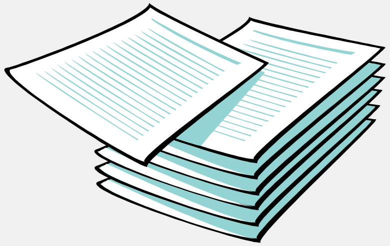
- 任务并行（Task parallelism）
  - Partition various tasks carried out solving the problem among the cores.

- 数据并行（Data parallelism）
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it’s part of the data.

P教授的测验

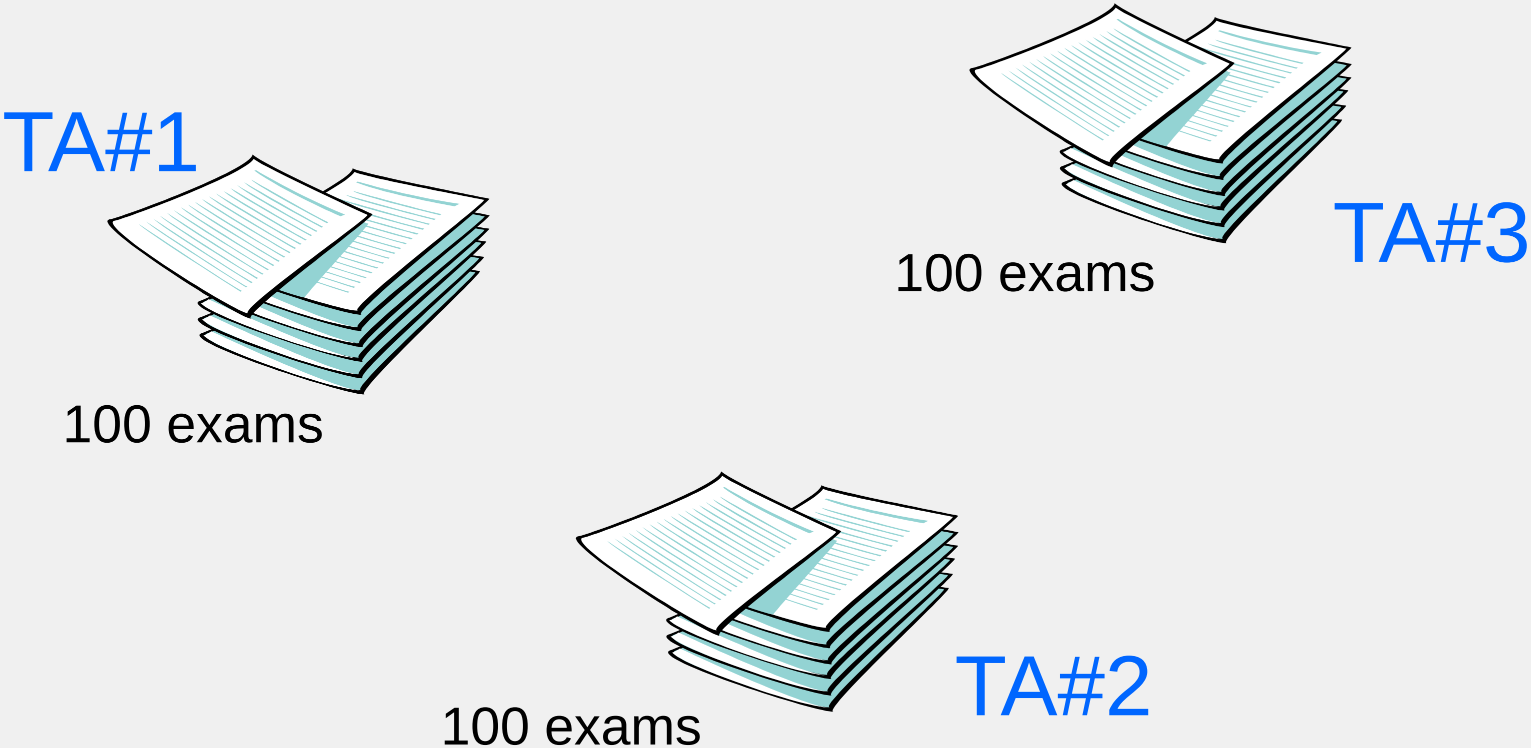
15 题目

300份试卷



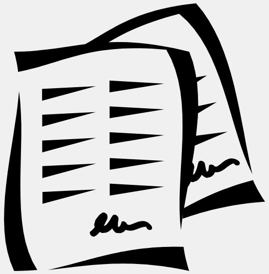
P教授的测验打分助教

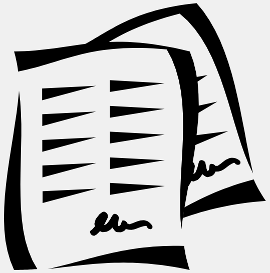
并行化方法– 数据并行



并行化方法–任务并行

TA#1

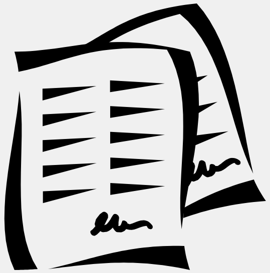




TA#3

Questions 11 - 15

Questions 1 - 5



TA#2

Questions 6 - 10

Division of work – 数据并行

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

Division of work – 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) Receiving
2) Addition

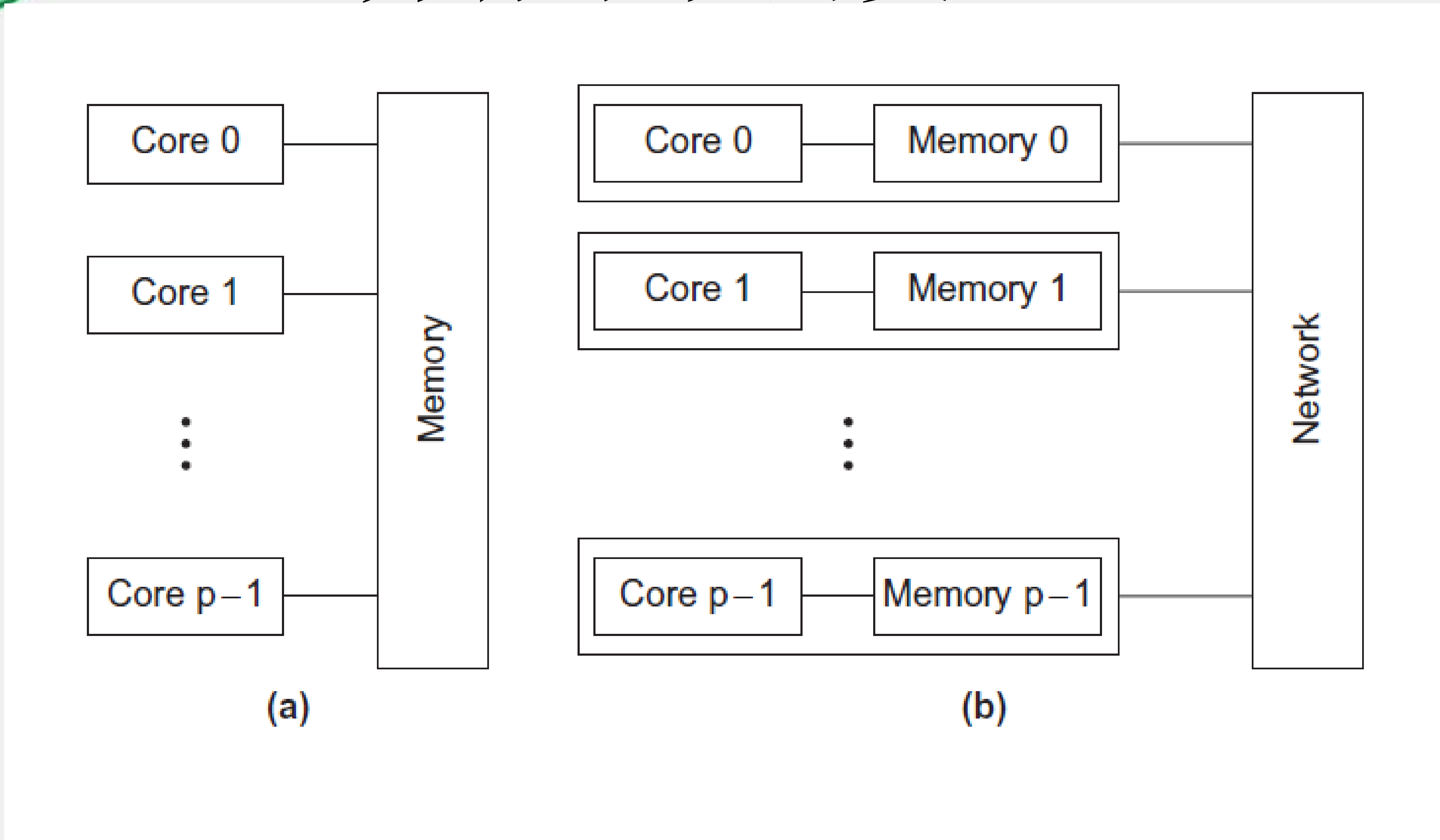
Coordination协调

- Cores usually need to coordinate their work.
- Communication – one or more cores send their current partial sums to another core.
- Load balancing – share the work evenly among the cores so that one is not heavily loaded.
- Synchronization – because each core works at its own pace, make sure cores do not get too far ahead of the rest.

What we'll be doing

- Learning to write programs that are explicitly parallel.
- Using the C language.
- Using three different extensions to C.
  - Message-Passing Interface (MPI)
  - Posix Threads (Pthreads)
  - OpenMP
- GPU编程——CUDA

并行系统的类型



Shared-memory　　Distributed-memory

Type of parallel systems

- Shared-memory
  - The cores can share access to the computer's memory.
  - Coordinate(协调) the cores by having them examine and update shared memory locations.
- Distributed-memory
  - Each core has its own, private memory.
  - The cores must communicate explicitly by sending messages across a network.

术语

- 并发计算（Concurrent computing）

  —— 一个程序的多个任务在同一时段内可以同时执行。

- 并行计算（Parallel computing）

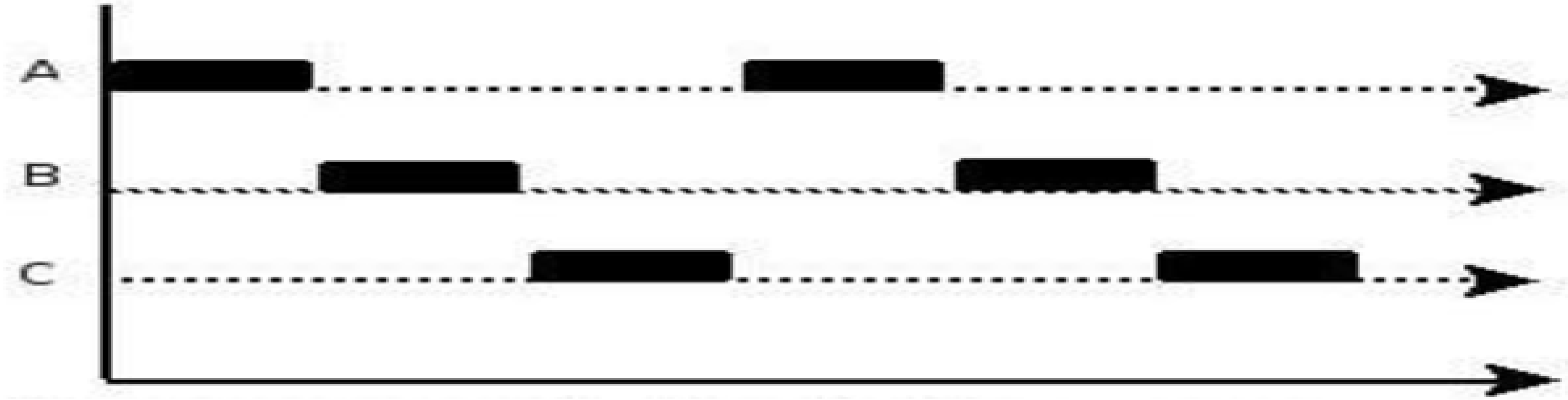
  —— 一个程序通过多个任务紧密合作以解决某个问题。

- 分布式计算（Distributed computing）

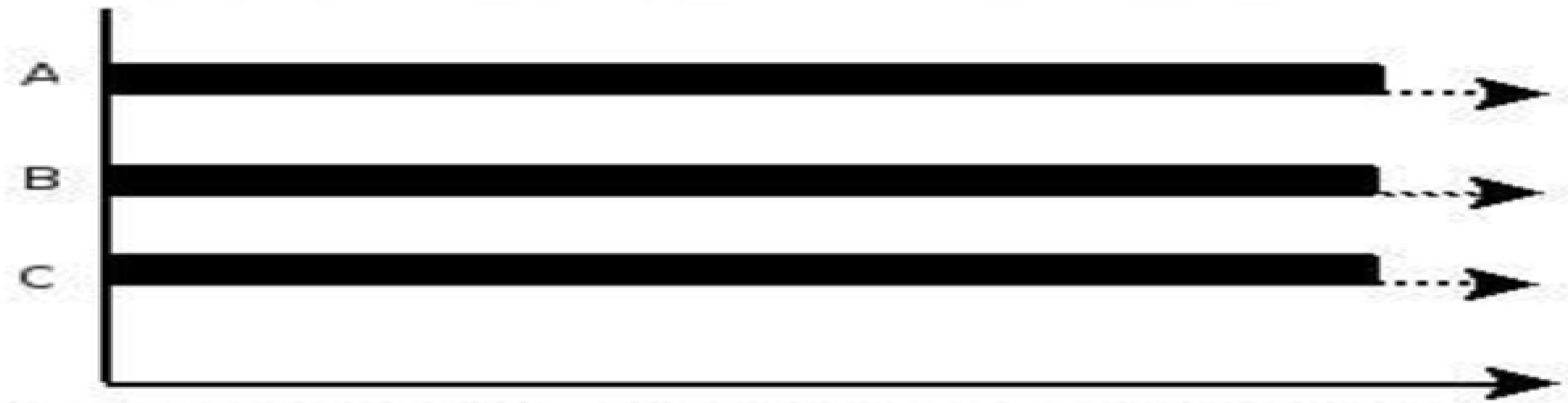
  —— 一个程序需要与其他程序协作来解决某个问题。

并发和并行

- 三个同时运行的程序A、B、C



并发：单核，逻辑上同时处理



并行：多核或多处理器，物理上同时处理

Concluding Remarks (1)

- The laws of physics have brought us to the doorstep of multicore technology.
- Serial programs typically don’t benefit from multiple cores.
- Automatic parallel program generation from serial program code isn’t the most efficient approach to get high performance from multicore computers.

Concluding Remarks (2)

- Learning to write parallel programs involves learning how to coordinate the cores.
- Parallel programs are usually very complex and therefore, require sound program techniques and development.