中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

并行程序设计与算法

引言

陶钧

taoj23@mail.sysu.edu.cn

中山大学 计算机学院

国家超级计算广州中心

课程概要

- 课程目标与评分标准
- 为什么需要并行?

课程信息

- 主讲教师
  - 陶钧：taoj23@mail.sysu.edu.cn
- 课程网站
  - https://easyhpc.net/course/193
- 助教信息
  - 叶龄徽：yelh5@mail2.sysu.edu.cn
  - 余涛：yutao58@mail2.sysu.edu.cn
- 课程群：稍后公布

课程主要内容

- 相关知识背景（1-2周）
  - 并行与并发、并行软硬件
- 并行编程工具（3-13周）
  - MPI, Pthreads, OpenMP, CUDA
- 并行算法设计（14-18周）
  - 一般流程
  - 并行分治算法、并行动态规划、并行贪心算法、并行回溯算法

课程需要

- 知识储备需要
  - 熟练使用一门编程语言（最好是C或C++）
  - 最好了解：指针与内存、计算机体系结构
- 硬件需要
  - 支持CUDA的显卡（NVIDIA）
    - 往年学院会提供GPU集群供大家使用
    - 但效率可能较低

课程评分标准

- 课程总分组成
  - 期末考试 60%
  - 平时（编程+理论+考勤） 40%
- 编程作业评分示例
  - 结果正确性 55%
  - 性能 20%
  - 编程规范 5%
  - 书面报告 20%

课程评分标准

- 编程规范举例
  - 提高可读性
    - 应组织到适当文件中
    - 应根据功能适当划分成函数
    - TA应该能轻松读懂你们的代码

文件开头应包含程序说明

```
########################################
##  姓名:  XXX
##  文件说明:  **********************
##  **********************************
##  **********************************
##  **********************************
##  **********************************
########################################

#include <abc.h>

int main(){
    ...
}
```

函数开头也应包含说明
程序应适当缩进

```
########################################
##  函数:  do_something
##  函数描述:  **********************
##  **********************************
##  参数描述:
##  par1: **********************
##  par2: **********************
########################################
void do_something(par1, par2){
    for( ... ){
        if( ... ){
            ...
        }
    }
}
```

课程评分标准

- 其他
  - 允许讨论代码，严禁抄袭
  - 3 slip days!
    - 应提前通过邮件正式通知TA
    - 包括周末节假日

参考书目（文档）

- 并行程序设计导论，Peter S Pacheco，机械工业出版社， 2016
- 并行多核体系结构基础，汤孟岩，机械工业出版社， 2019
- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, Introduction to Parallel Computing (2nd Edition), Pearson Education Limited, 2003.（有中译本）
- 陈国良，并行计算——结构.算法.编程　高等教育出版社，2003.

参考书目（文档）

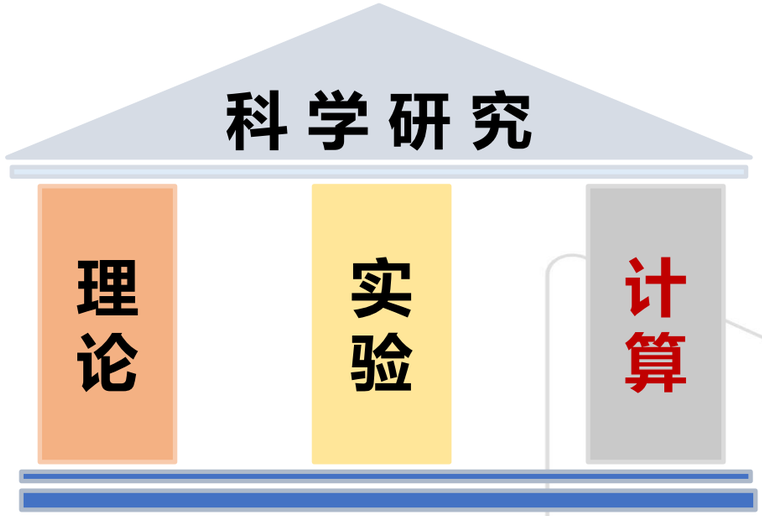
- E. Kandrot and J. Sanders, CUDA by Example: An Introduction to General-Purpose GPU Programming, Pearson, 2010.
  - 中译版：GPU高性能编程：CUDA实战
- N. Wilt, The CUDA Handbook: A Comprehensive Guide to GPU Programming, Prentice Hall, 2013.
  - 中译版：CUDA专家手册：GPU编程权威指南
- J. Cheng, M. Grossman, and T. McKercher, Professional CUDA C Programming, Wrox, 2014.
  - 中译版：CUDA C编程权威指南
- NVIDIA官方文档：
  - CUDA C Programming Guide
  - CUDA C Programming Best Practice Guide

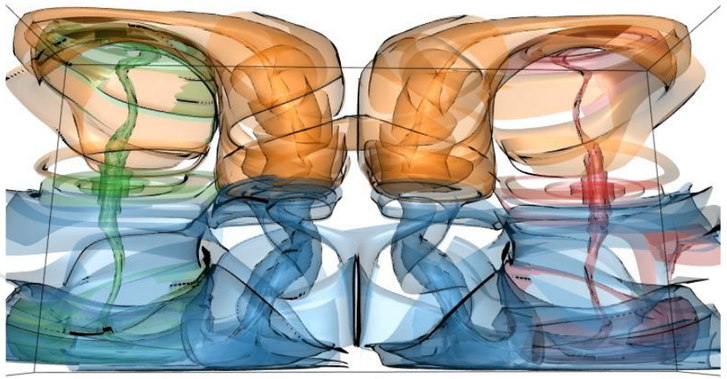
课程概要

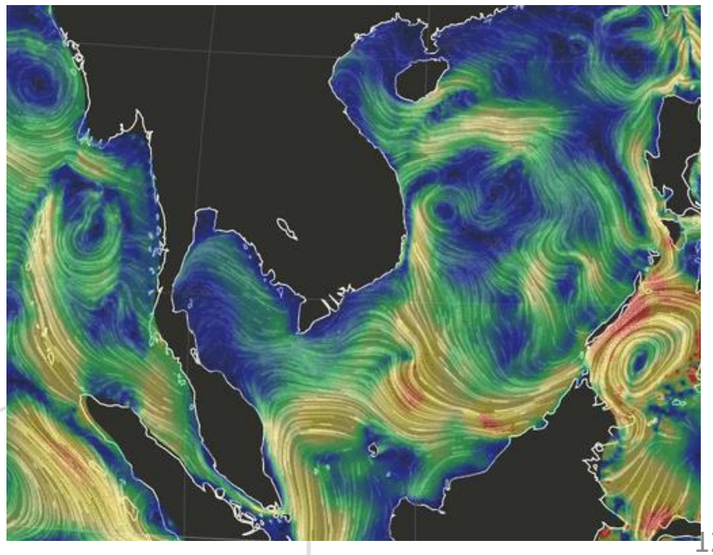
- 课程目标与评分标准
- 为什么需要并行?
- 如何利用并行?

为什么需要并行?

- 科学方法的革命
  - 2007年，图灵奖得主，关系型数据库鼻祖Jim Gray将科学研究分为四类范式：实验归纳，模型推演，仿真模拟，数据密集型科学发现
- 第三范式：计算机仿真
  - 在理论模型的指导下使用计算机模拟科学实验
  - 通过模拟仿真产生科学猜想、验证科学假设、修正模型

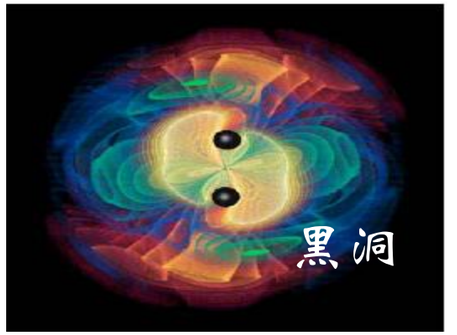


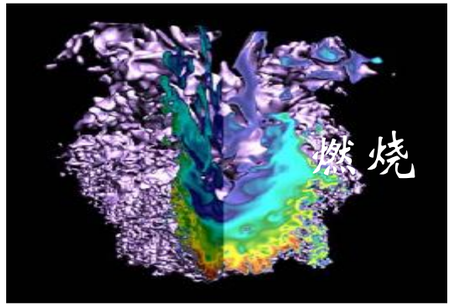


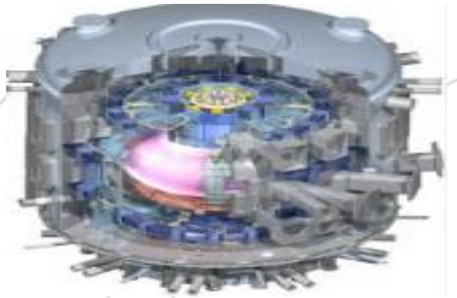


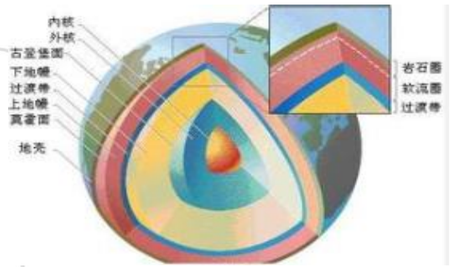
为什么需要并行?

- 解决大科学、大工程、产业升级的挑战性问题
  - 尺度超大 Too big
  - 尺度超小 Too small
  - 时变超快 Too fast
  - 时变超慢 Too slow
  - 过程超昂贵 Too expensive
  - 过程超危险 Too dangerous









为什么需要并行?

- 传统计算机问题
  - 算法不够算力来补!
  - P=NP?
  - 无限猴子理论（infinite monkey theorem）
  - 神经网络

为什么需要并行?

- 处理器发展史
  - 更宽的数据
    - 4 bits → 8 bits → 16 bits → 32 bits → 64 bits
  - 更高效的流水线
    - 平均每个指令3.5时钟周期→平均每个指令1.1时钟周期
  - 更高效的指令级并行
    - Superscalar：每时钟周期发出最多4个指令
    - Out-of-order：指令流中提取并行指令
  - 更快的时钟周期
    - 10 MHz → 200 MHz → 3/4 GHz

为什么需要并行?

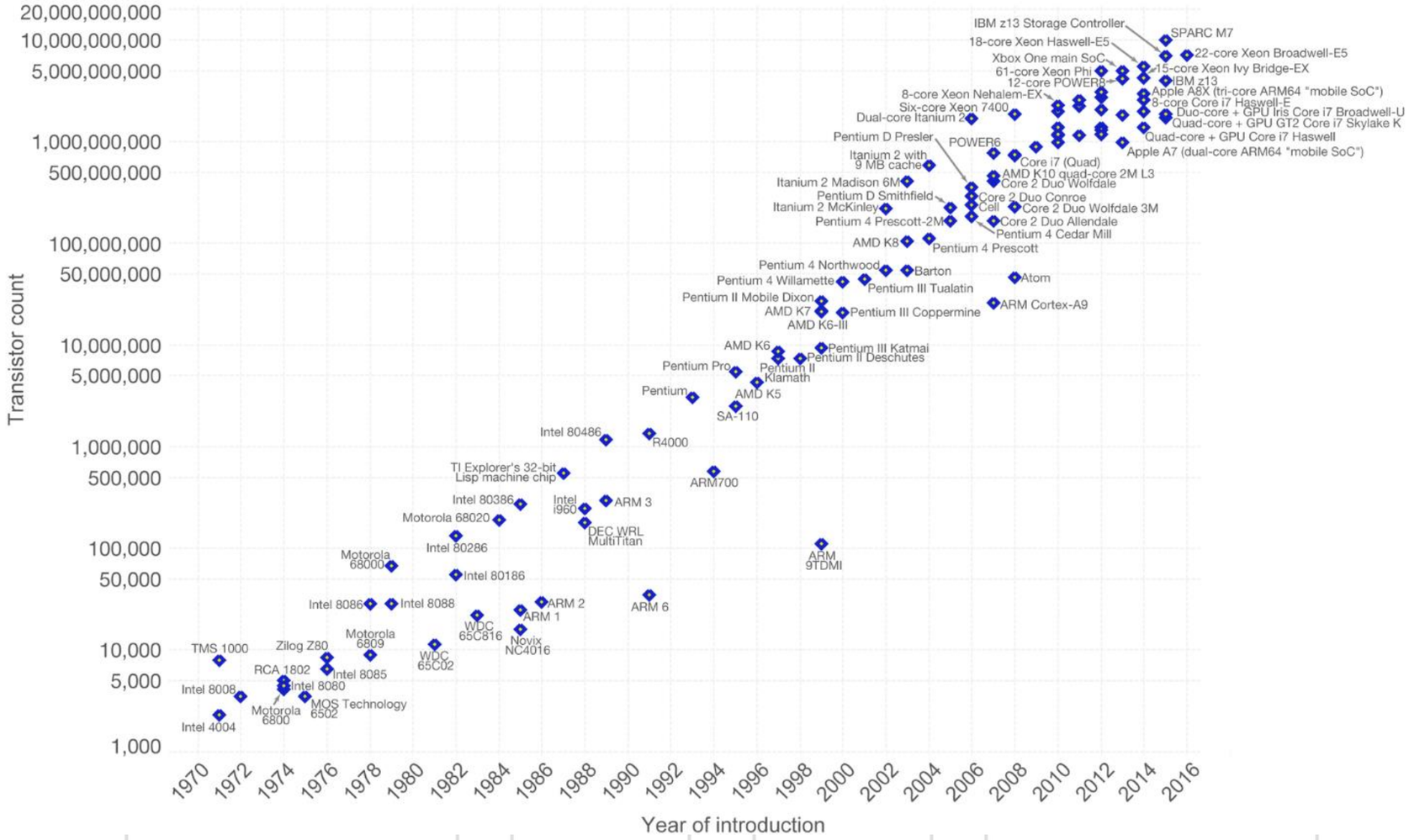
- 摩尔定律
  - 英特尔创始人之一戈登·摩尔提出:

"集成电路上可容纳的晶体管数目,
约每隔两年便会增长一倍"

  - 另一版本为18个月（由英特尔首席执行官大卫·豪斯提出）

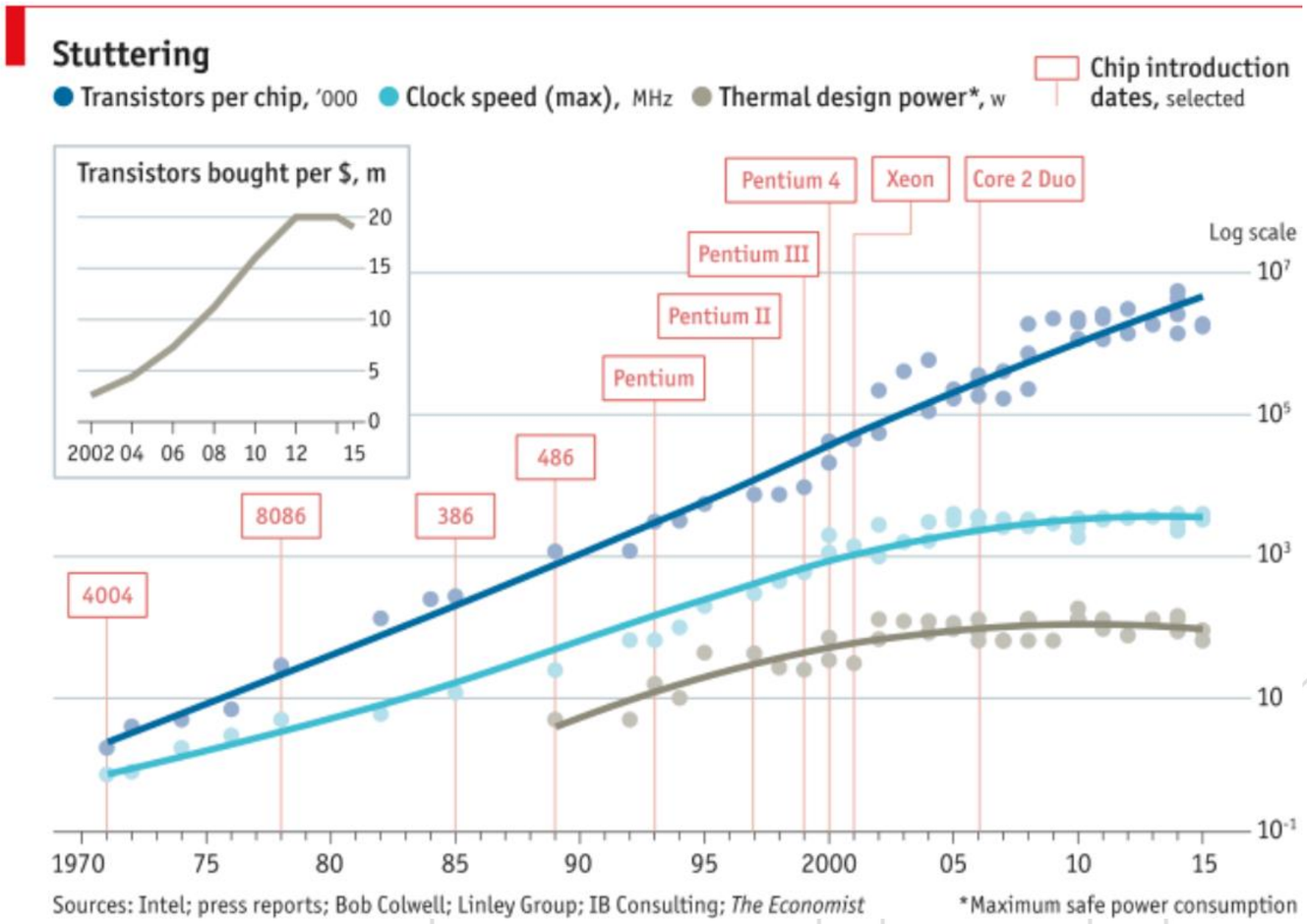
为什么需要并行?

- 摩尔定律



为什么需要并行?

- 摩尔定律的瓶颈
  - 晶体管数目≠频率



摩尔定律还能持续多久?

- 2021/2025年假说
  - 没有失效
  - 制程仍在降低
  - 但有减慢的趋势
- 量子穿隧效应

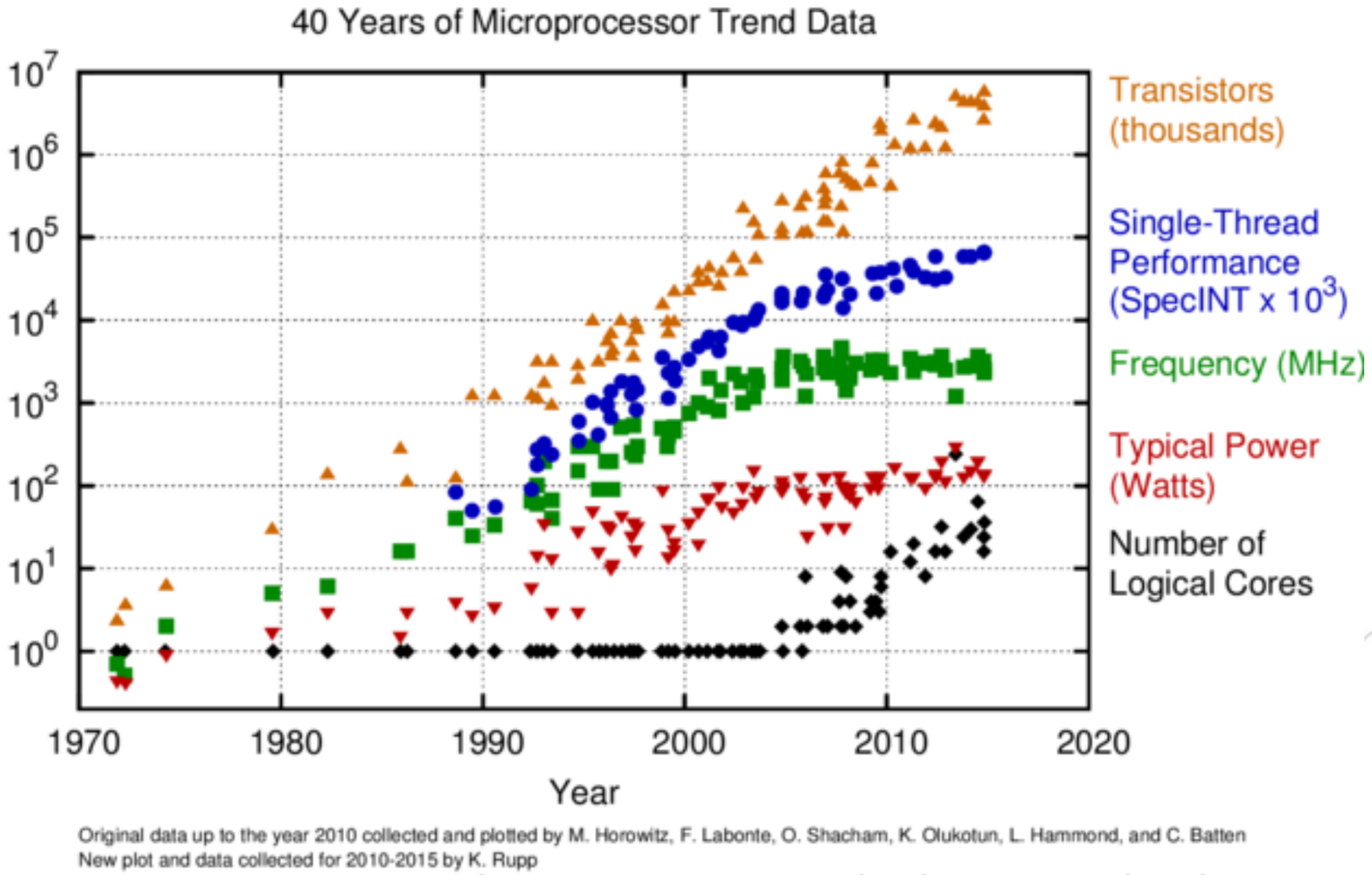
"The number of people predicting the death of Moore's Law doubles **every two years.**"

-Peter Lee, Vice President, Microsoft Research

图片来源：The Economist's Technology Quartlerly

为什么需要并行?

- 摩尔定律的瓶颈
  - 晶体管数目≠频率



摩尔定律还能持续多久?

- 2021/2025年假说
  - 没有失效
  - 制程仍在降低
  - 但有减慢的趋势
- 量子穿隧效应

"The number of people predicting the death of Moore's Law doubles **every two years.**"

-Peter Lee, Vice President, Microsoft Research

图片来源：Ars Technica

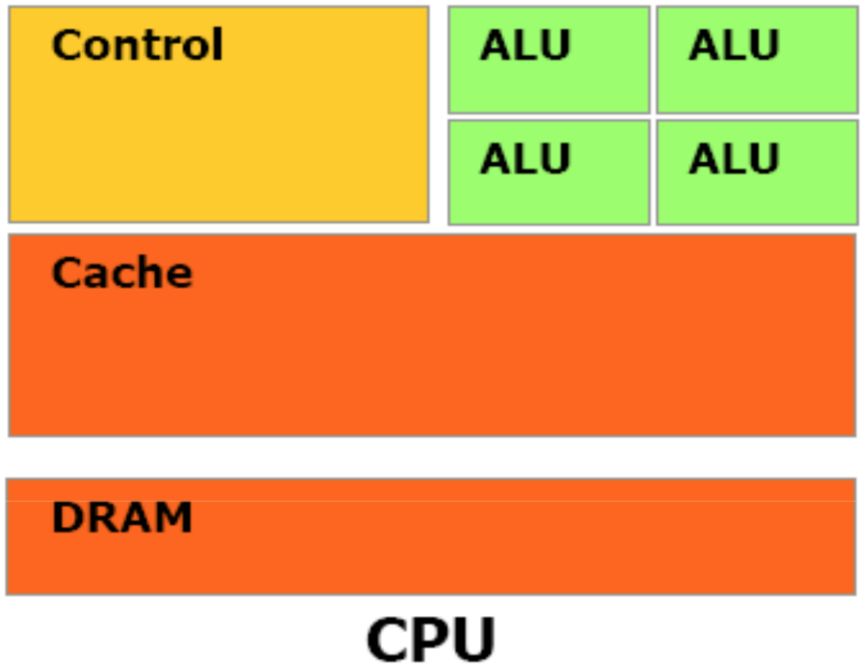
为什么需要并行?

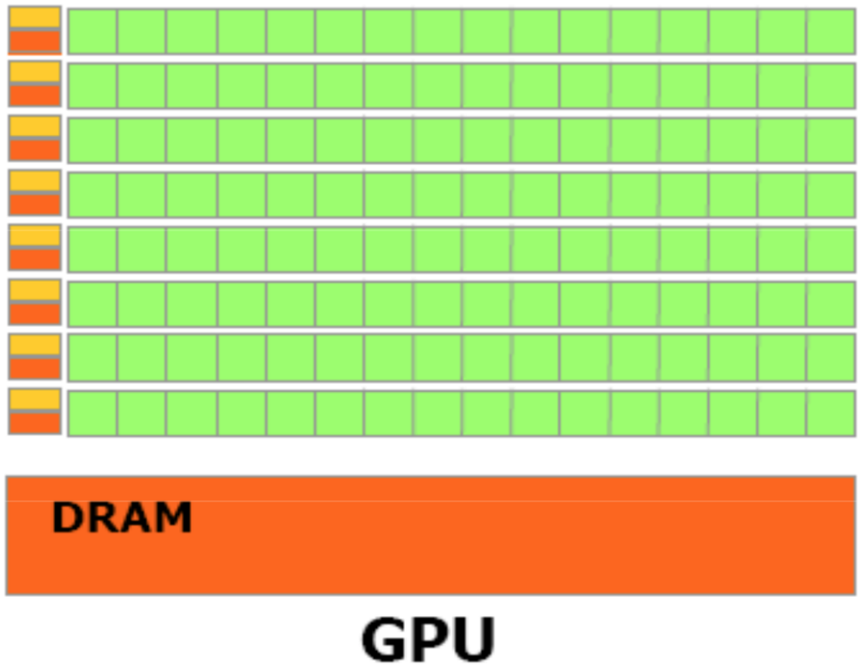
- 串行速度提升已经接近结束
  - 无法继续提升频率
  - 难以继续降低功耗
- 当前计算机性能提升趋势
  - 计算机没有变得更快，而是变得更宽
    - 多核CPU、GPU、超级计算机
  - 数据级别的并行
    - 同样的指令作用于多个数据
  - 线程级别的并行

图片来自网络

为什么需要并行?

- 更宽的处理器：图形处理器GPU
  - 高度并行化：同样指令应用于大量数据上
  - 大量运算核心

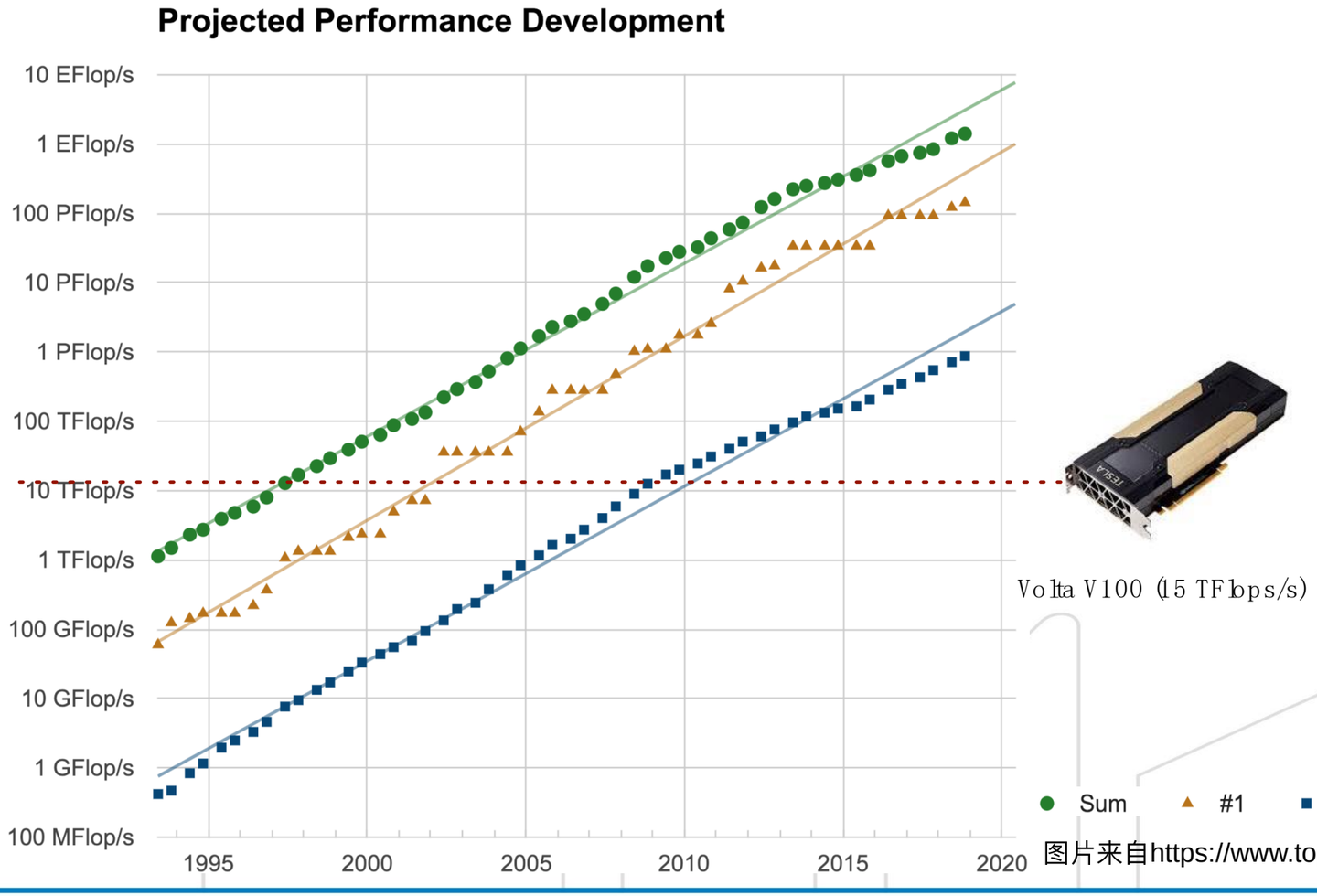




图片来自NVIDIA

为什么需要并行?

- 更宽的处理器：图形处理器GPU
  - GPU与超算



Volta V100 (15 TFlops/s)

图片来自https://www.top500.org/statistics/perfdevel/

为什么需要并行?

- 更宽的计算机：超级计算机
  - 异构集群
  - Oak Ridge National Lab: Summit：4,608 计算节点
    - 每个计算节点拥有2个22-core CPU及6个GPU

课程概要

- 课程目标与评分标准
- 为什么需要并行?
- 如何利用并行?

如何利用并行?

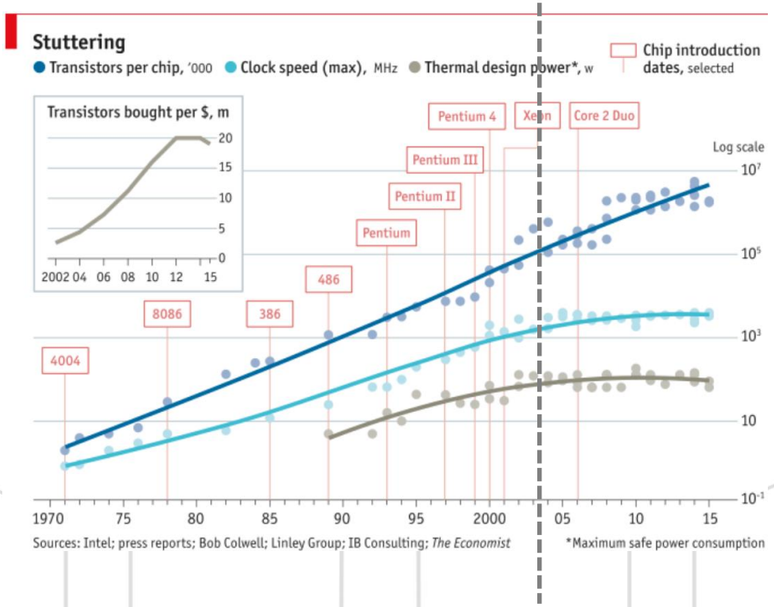
- 程序员视角：如何让你的程序变得更快?

等待新电脑

如何利用并行?

- 程序员视角：如何让你的程序变得更快?

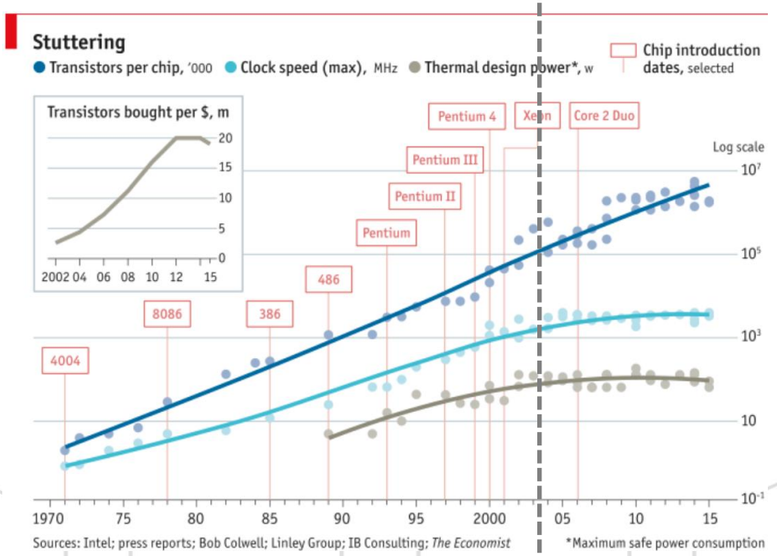
2004年前：等待新电脑
Problem solved!



如何利用并行?

- 程序员视角：如何让你的程序变得更快?

2004年前：等待新电脑
Problem solved!

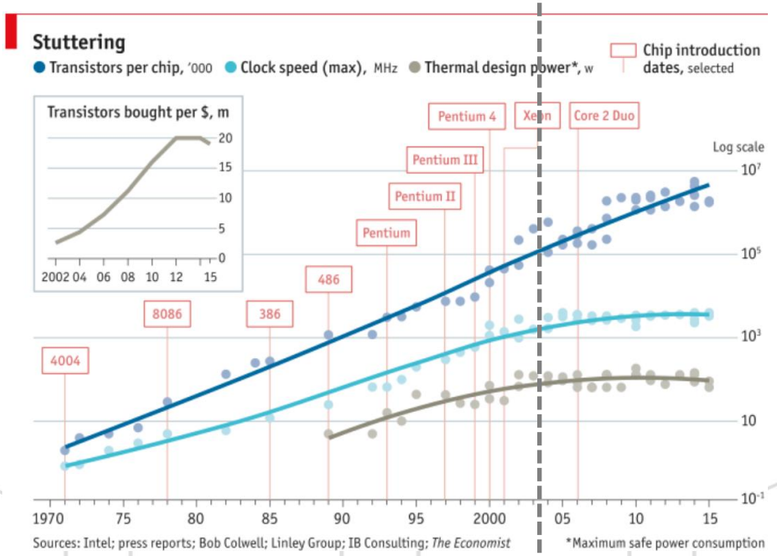


2004年后：等待新电脑
速度差不多☹

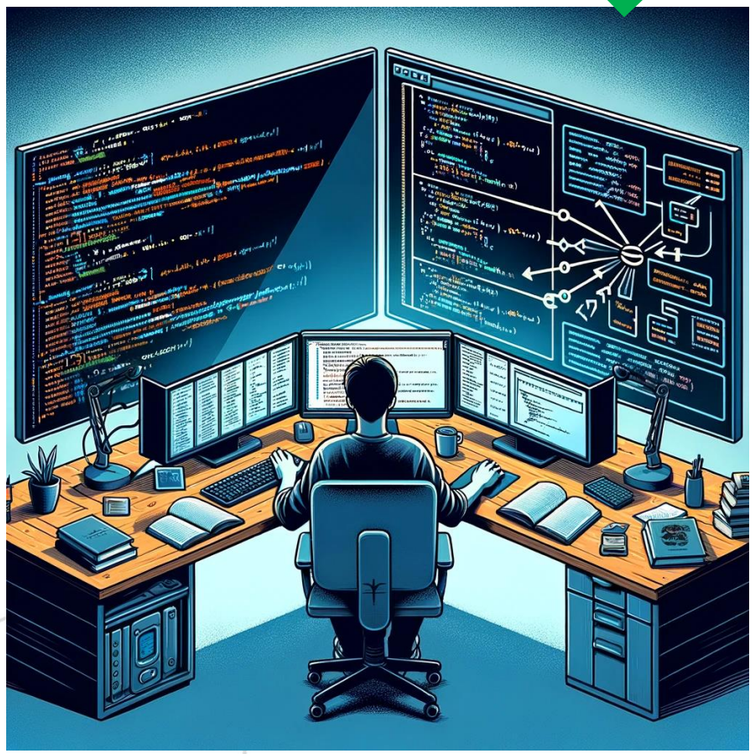
如何利用并行?

- 程序员视角：如何让你的程序变得更快?

2004年前：等待新电脑
Problem solved!

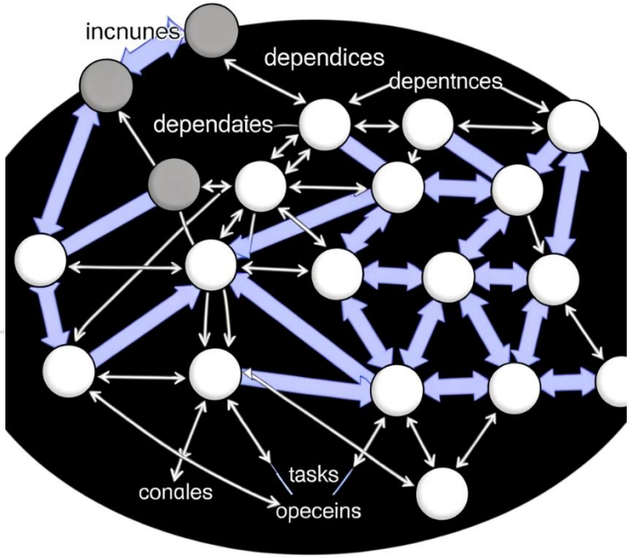


2004年后：串行改并行



如何利用并行?

- 如何并行化现有程序?
  - 方案1：将串行程序重写为并行程序
  - 方案2：通过程序自动将串行程序翻译为并行程序
    - 非常困难，目前不太成功
      - 通过代码识别任务依赖关系：无法改变固有串行计算模式



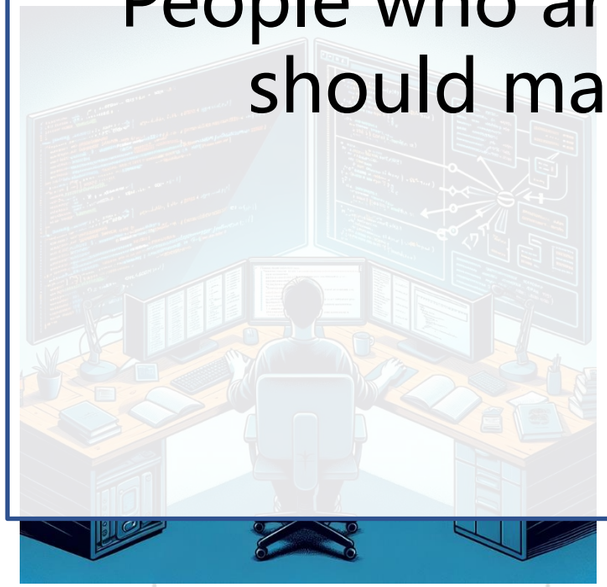
如何利用并行?

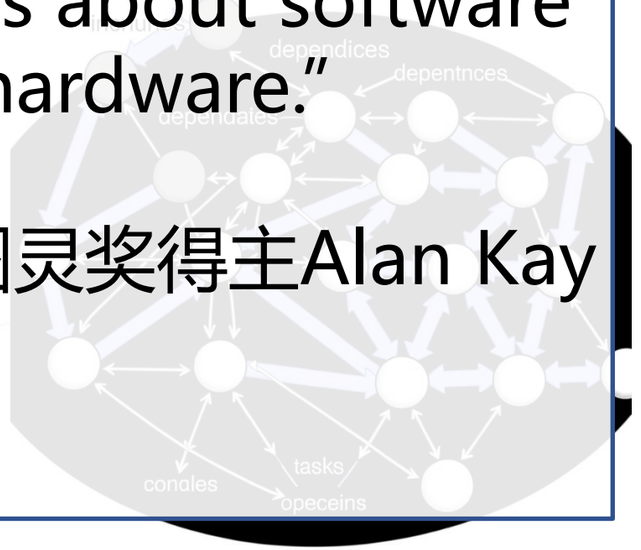
- 如何并行化现有程序?
  - 方案1：将串行程序重写为并行程序
  - 方案2：通过程序自动将串行程序翻译为并行程序
    - 非常困难，目前为止
      - 通过代码识别任务依赖关系：无法改变固有串行计算模式

需要为并行而设计的算法!

“People who are really serious about software should make their own hardware.”

-图灵奖得主Alan Kay





并行思维 vs 串行思维

- 例：计算n个数并对齐求和

串行解决方案：

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

并行思维 vs 串行思维

- 例：计算n个数并对齐求和
  - 如果我们有p个计算核心
    - 每个核独立执行n/p次循环，计算部分和
    - 每个核需要维护其独立的变量

串行直接转为并行：

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

并行思维 vs 串行思维

- 例：计算n个数并对齐求和
  - 每个核独立执行n/p次循环：效率如何?
    - 假设有p=8个核，n=24个数
    - 每个核处理3个数字，求得局部和之后将其发送给主核

`compute_next_value(...)` 函数返回以下值：

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

并行思维 vs 串行思维

- 例：计算n个数并对齐求和
  - 每个核独立执行n/p次循环：效率如何?
    - 假设有p=8个核，n=24个数
    - 每个核处理3个数字，求得局部和之后将其发送给主核

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

并行思维 vs 串行思维

- 例：计算n个数并对齐求和
  - 每个核独立执行n/p次循环：效率如何?
    - 如果有1024个核呢？需要1023次发送 & 加法
      - 需要层次结构：例如，32个核一组，则只需两轮，每轮31次发送 & 加法

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

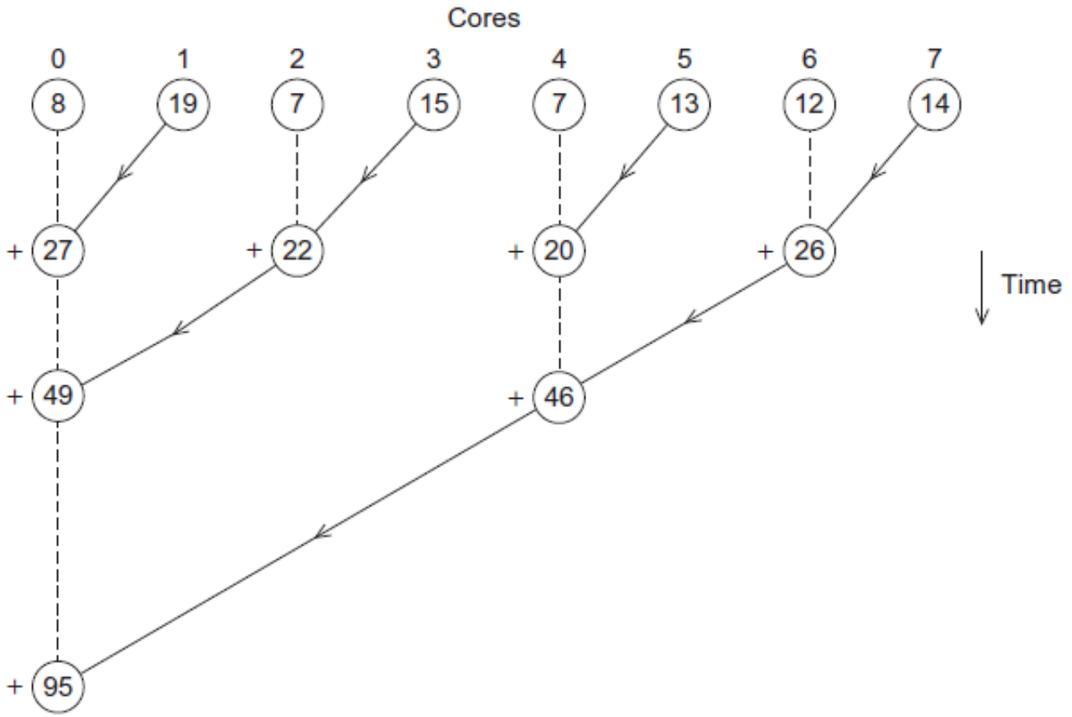
Global sum

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

并行思维 vs 串行思维

- 例：计算n个数并对齐求和
  - 每个核独立执行n/p次循环：效率如何？
    - 把层次推到极致，每2个核一组，可以将时间压缩到$\log_2 n$



如何设计并行程序?

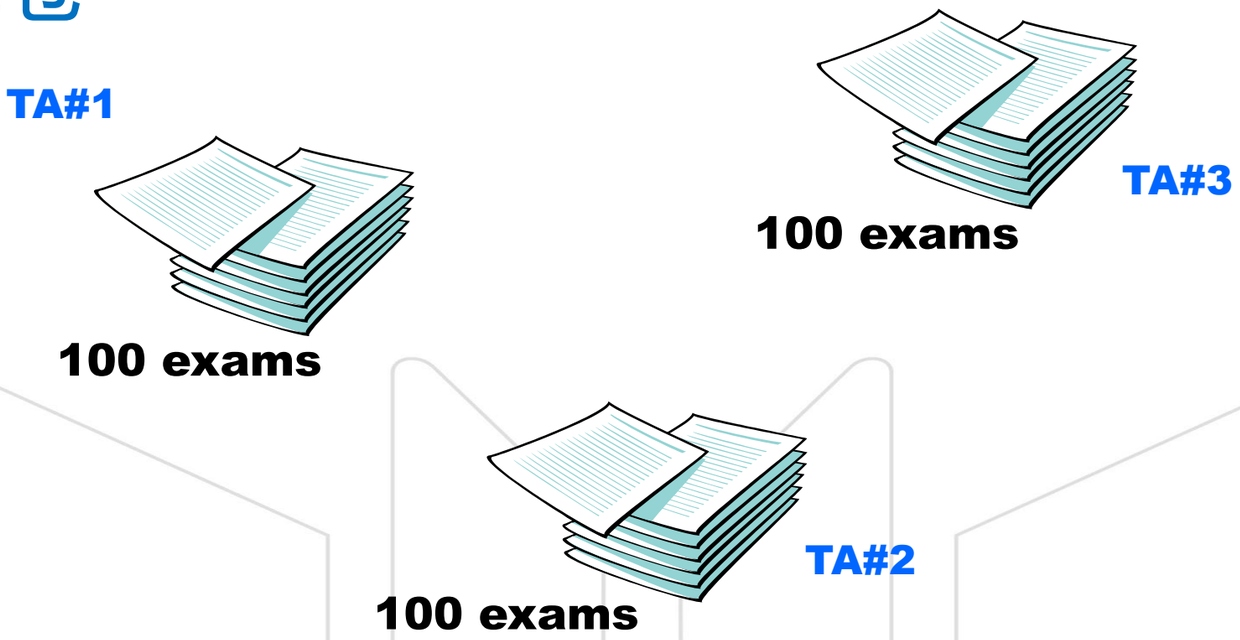
- 任务并行与数据并行
  - 任务并行：将不同任务分配到不同核上执行
  - 数据并行：将不同数据分配到不同核上执行
    - 但每个核执行的内容相似
- 例：批改试卷

15 questions

300 exams

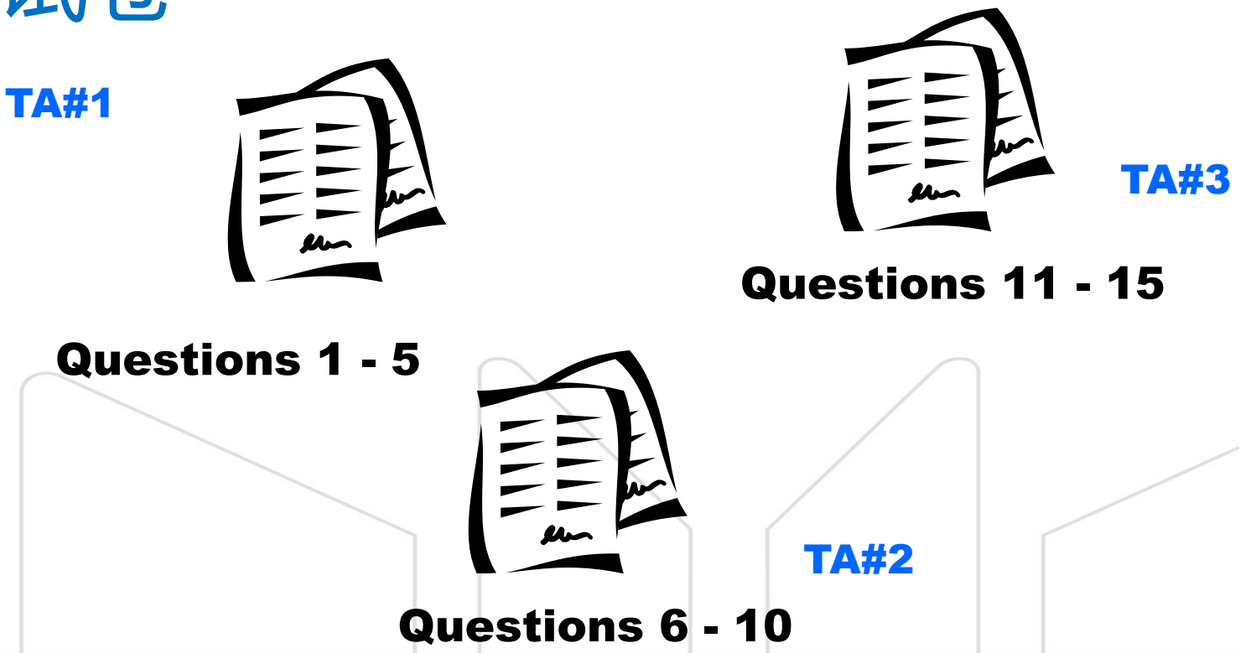
如何设计并行程序?

- 任务并行与数据并行
  - 任务并行：将不同任务分配到不同核上执行
  - 数据并行：将不同数据分配到不同核上执行
    - 但每个核执行的内容相似
- 例：批改试卷



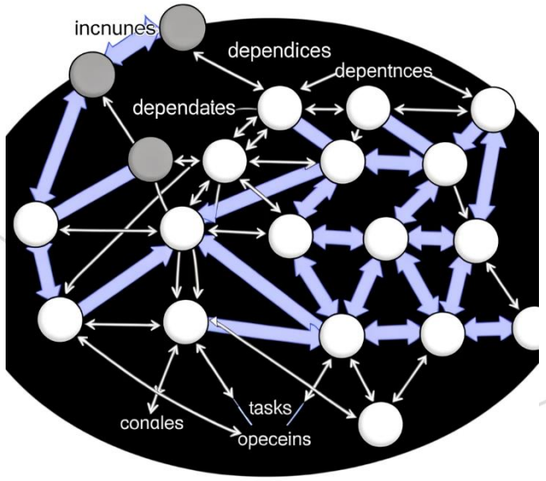
如何设计并行程序?

- 任务并行与数据并行
  - 任务并行：将不同任务分配到不同核上执行
  - 数据并行：将不同数据分配到不同核上执行
    - 但每个核执行的内容相似
- 例：批改试卷



如何设计并行程序?

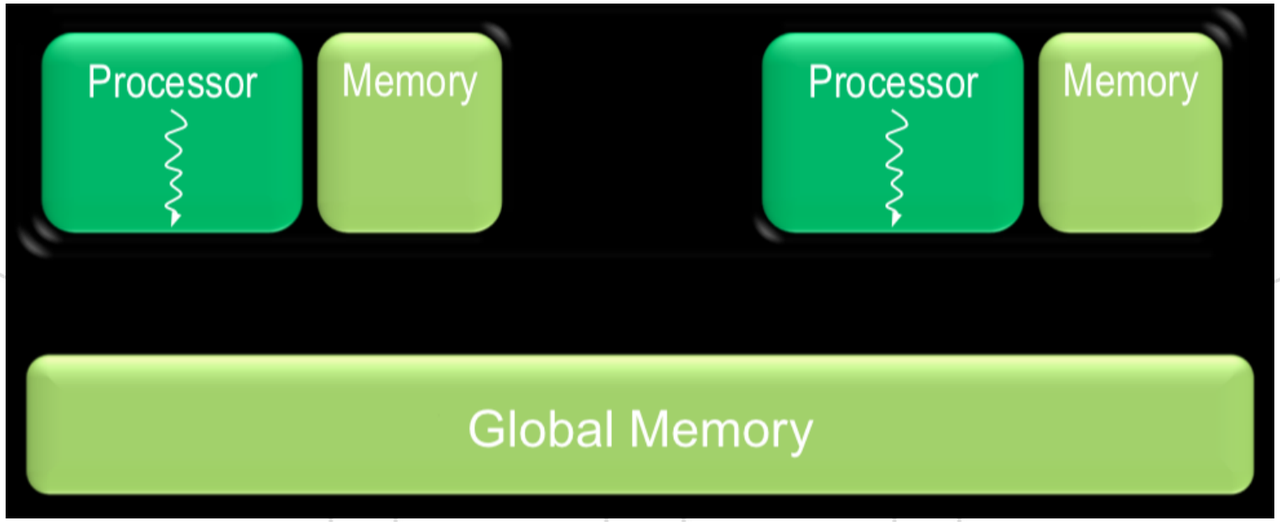
- 现实中，计算任务间往往存在复杂的依赖关系
  - 计算核心也需要协调其工作
    - 通信：计算核心之间需要发送计算结果
    - 负载均衡：计算核心之间工作量需保持相似
      - 否则就会出现部分核心长期等待，而效率下降
    - 同步：需要保持不同核心上任务执行的相对顺序



并行工具

- Pthreads/OpenMP vs CUDA vs MPI
  - Pthreads (Posix Threads); MP (multi-processing)
  - 多线程
  - 基于共享内存的多核系统
    - 多核个人主机
    - 超级计算机（截止2017年效率并不好）

- Pthreads/OpenMP vs CUDA vs MPI
  - 硬件结构
    - 每个处理器运行一个线程
    - 每个处理器有易于访问的片上存储（on-chip memory）
    - 所有处理器共享全局存储器

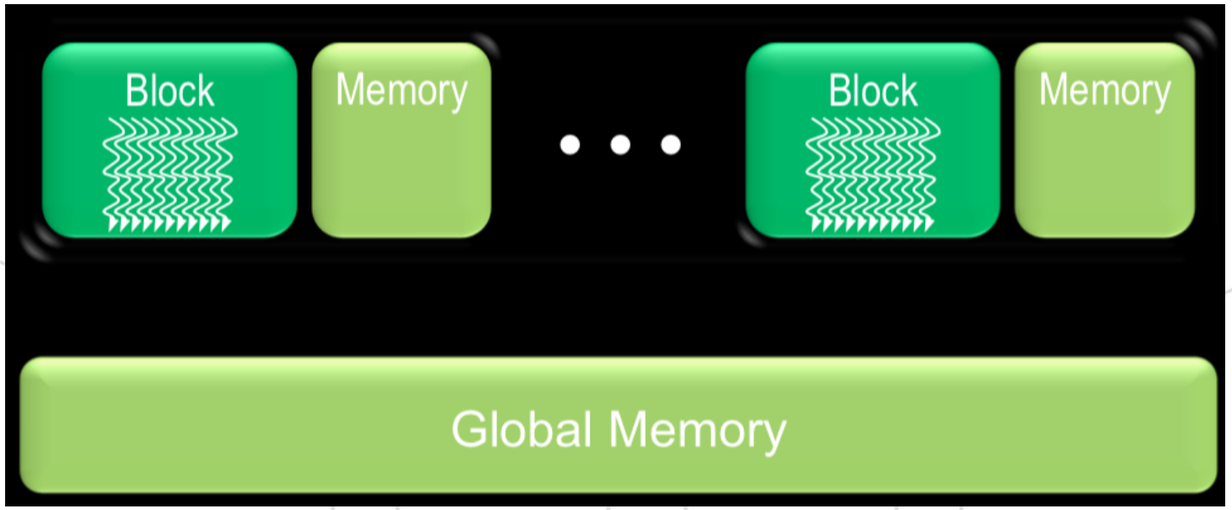


图片来自NVIDIA

并行工具

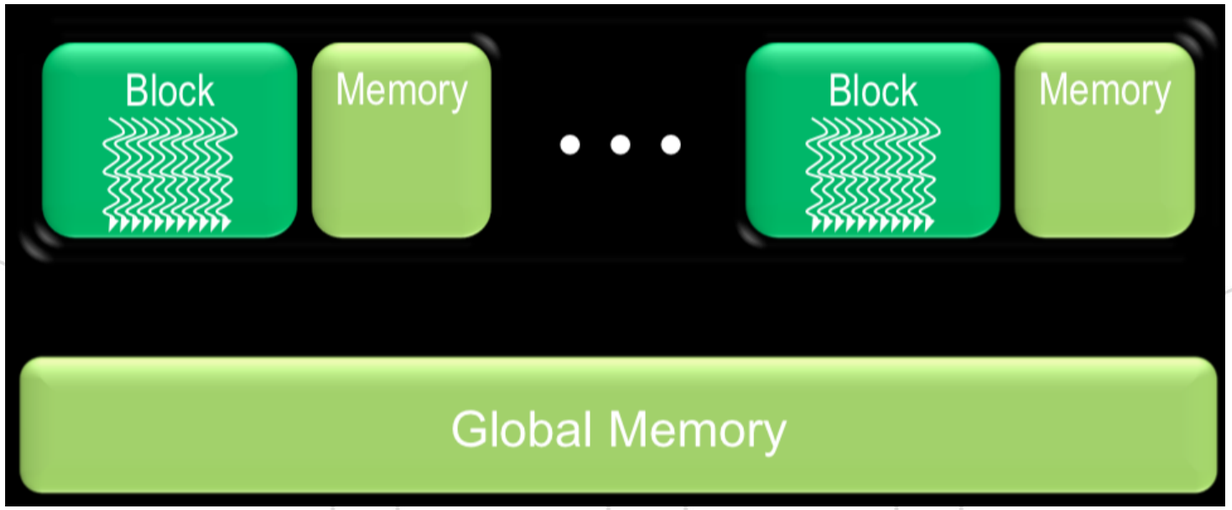
- Pthreads/OpenMP vs CUDA vs MPI
  - NVIDIA显卡
  - 基于GPU线程并行

- Pthreads/OpenMP vs CUDA vs MPI
  - 硬件结构
    - 图中每条波浪线代表一个线程
    - 每个block包含多个线程
      - 只有block内部线程能访问其共享内存
    - 所有线程均可访问全局存储器



图片来自NVIDIA

- Pthreads/OpenMP vs CUDA vs MPI
  - 硬件结构
    - 每个线程由一个CUDA core执行
    - Block中的线程以warp的形式在streaming multiprocessor上执行
      - 每个warp包含32个thread
      - warp中所有thread执行同样的指令 （Single Instruction Multiple Data）

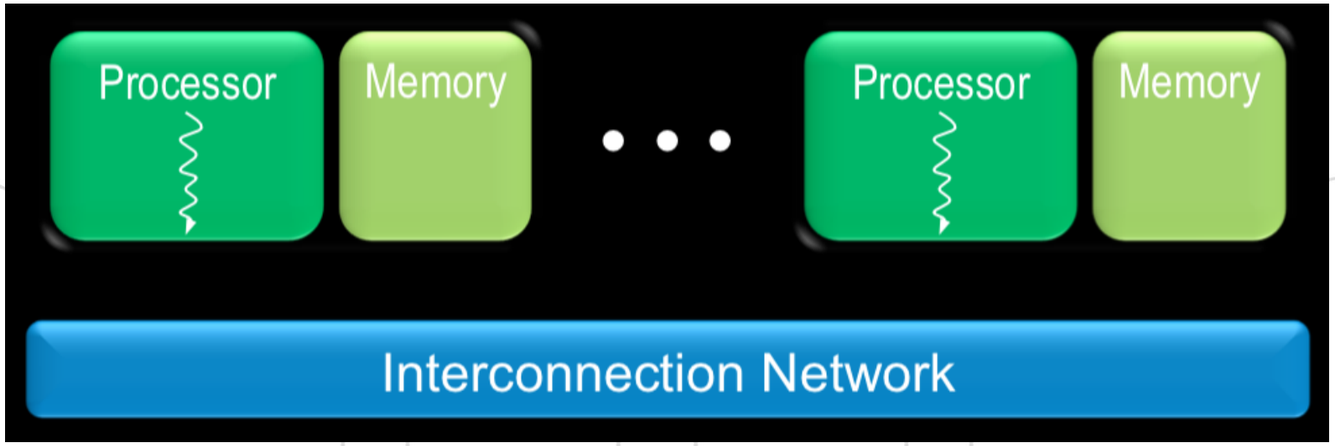


图片来自NVIDIA

并行工具

- Pthreads/OpenMP vs CUDA vs MPI
  - MPI（Message Passing Interface）
    - 跨平台
    - 多进程
    - 基于信息传递
    - 常用于计算机集群

- Pthreads/OpenMP vs CUDA vs MPI
  - 硬件结构
    - 多个处理器之间通过网络发送消息交互



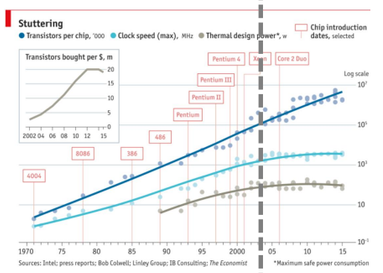
图片来自NVIDIA

并行工具

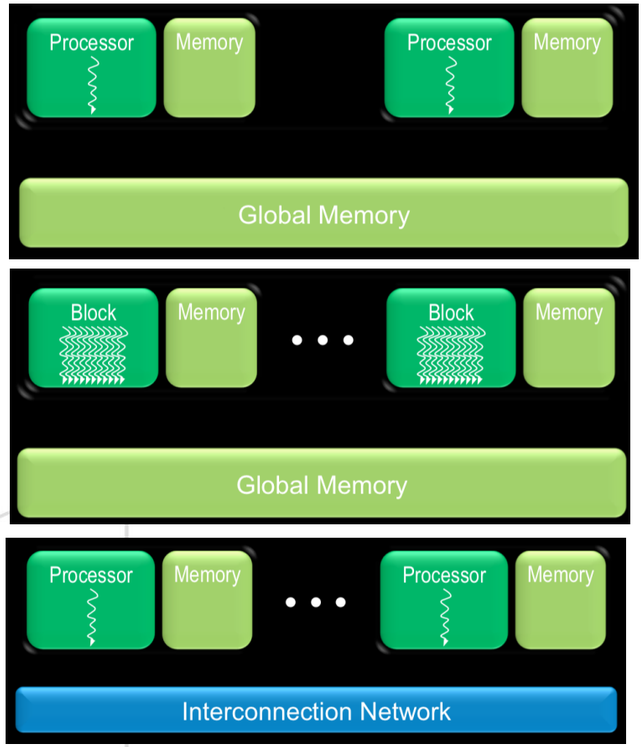
- Pthreads/OpenMP vs CUDA vs MPI
  - OpenMP/Pthreads
    - 创建线程开销高
    - 线程数目有限
    - 线程相互间可以完全独立
    - 控制流复杂
    - 总算力（相对）有限
  - CUDA
    - 创建线程开销低
    - 可创建大量线程
    - 同一warp内共享指令
    - 控制流简单
  - MPI
    - 进程相互间可以完全独立
    - 通信成本高
    - 价格昂贵

小结

程序无法天然利用更宽而非更快的计算机



- 为什么需要并行?
  - 频率提升已然停止
  - 计算机正在变得“更宽”而非更快
  - 应用广泛
    - 大规模科学计算乃至传统问题
- 如何更好地利用并行设备?
  - 需要为并行设备设计相应的并行程序、并行算法
  - 并行思维：尽可能将多个计算重叠
  - 并行工具：Pthreads/OpenMP, CUDA, MPI
    - 多线程与多进程



Questions?