

分布式并行计算

卢国明（研究员）

计算机科学与工程学院



电子科技大学
University of Electronic Science and Technology of China

自我介绍

联系方式

- 邮箱: lugm@uestc.edu.cn
- 办公室: 创新中心B栋201B

研究方向

- 并行计算
- 异构计算
- 人工智能

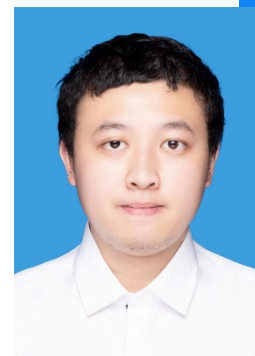


课程助教

- 姓名：董谦，博士研究生
- 邮箱：dongqian43@outlook.com
- 电话：15826197708

- 姓名：贺学万，硕士研究生
- 邮箱：773490498@qq.com
- 电话：15573441255

- 姓名：罗炼，硕士研究生
- 邮箱：1321718054@qq.com
- 电话：17844533259



课程目标

- 并行硬件和软件基础知识
- 并行算法设计和分析
- 并行编程技能
 - MPI
 - OpenMP(自学)
 - CUDA
 - OpenAcc



先修课程

- 数据结构与算法
- 操作系统
- 计算机体系结构



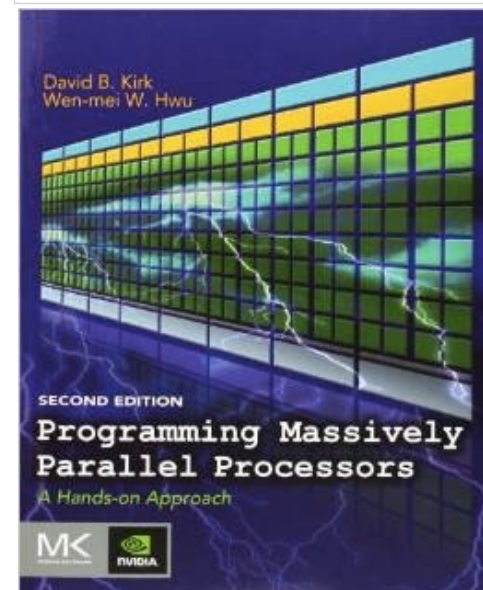
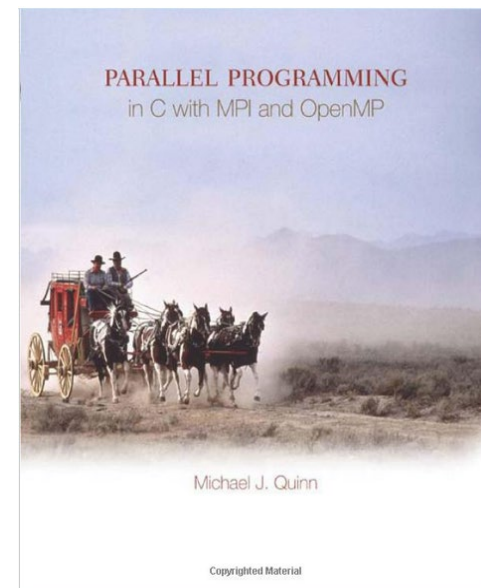
课程计划

- 课程介绍 (2课时)
- 并行架构 (4课时)
- 并行算法设计 (2课时)
- 性能评估 (2课时)
- MPI: 分布式内存并行编程 (10课时)
- CUDA: 共享内存并行编程 (10课时)
- 编程实践环节 (16课时)
 - MPI (8课时)
 - CUDA (8课时)



参考资料

- 《Parallel programming in C with MPI and OpenMP》, Michael J. Quinn, McGraw-Hill Science, 2003
- 《Programming Massively Parallel Processors(第二版)》David B Kirk, Wen-mei W. Hwu, Morgan Kaufmann, 2010.



其他资料

- <http://www.mpich.org/static/docs/latest/www3/>
- 高性能计算训练(LLNL)
<https://computing.llnl.gov/?set=training&page=index>
- 深度学习学院(Nvidia)
<https://courses.nvidia.com/dli-event>



课件来源

- 改编自Peter Pacheco的课件
- 改编自Ananth Grama的课件
- 改编自Michael Quinn的课件
- 改编自Kathy Yelick (Berkeley) 的课件
- 改编自John Mellor-Crummey (Rice) 的课件
- 改编自Jack Dongarra (Tennessee) 的课件
- 改编自Zhiling Lan (IIT)
- 改编自NVIDIA的课件



成绩构成

- 考勤与随堂测试： 20%
- 编程实验 30%
 - MPI: 15%
 - CUDA: 15%
- 期末考试： 50%
- 注： 考勤不合格的将挂科处理



课程状况和建议

- 认真听讲
- 认真练习
- 强化实践
- 选择需谨慎



第一节：课程介绍

- 什么是并行计算？
- 为什么需要并行计算？
- 如何编写并行程序？



分布式与并行计算

- 并行计算 (Parallel Computing)
 - 在单一系统中使用两个或多个处理器 (计算机) 同时工作以解决单个问题。
 - the use of two or more **processors** (computers), *usually within a single system*, working simultaneously to solve a **single problem**.
- 分布式计算
 - 为解决一个计算或者信息处理问题, 多台分布的计算机进行协同计算的方式
 - any computing that involves **multiple computers remote from each other** that each have a role in a computation problem or information processing.



并行计算机

- 并行计算机是支持并行编程的多处理器计算机系统。
- 两类并行计算机
 - 多计算机 (Multicomputer) : 利用多台计算机和互连网络构建的并行计算设备。 (消息传递)
 - 集中式多处理器 (Centralized multiprocessor) (对称多处理器或 SMP) : 一种更高度集成的系统, 其中所有的CPU共享访问一个单一的全局内存。 (通过共享内存进行通信和同步)



随堂测试：分布式系统举例

- 谷歌搜索
- 微信
- ...



并行系统 V.S. 分布式系统

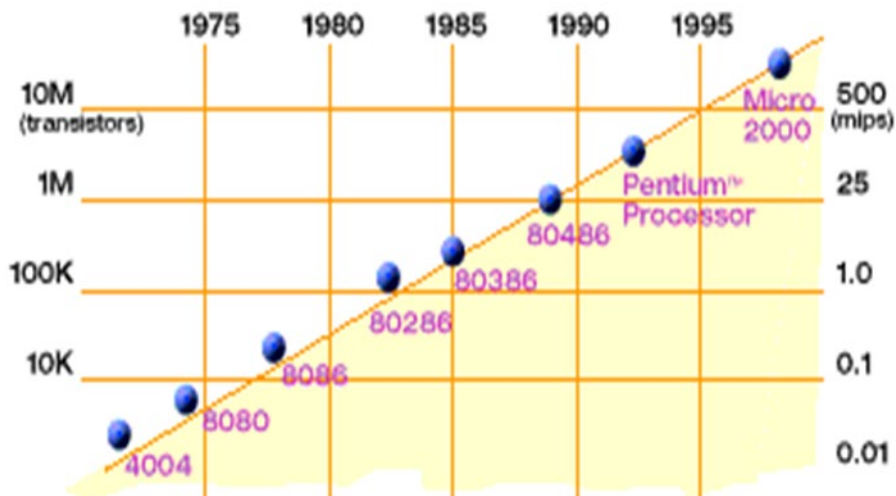
	并行系统	分布式系统
内存	紧耦合共享内存 UMA, NUMA 分布内存	分布内存 消息传递,远程过程调用, 分布式共享内存
控制	全局时钟 SIMD, MIMD	无全局时钟 Synchronization algorithms needed
处理器互联	Tbps Bus, mesh, tree, mesh of tree, and hypercube (-related) network	Gbps Ethernet(bus), token ring and SCI (ring), myrinet(switching network)
粒度	细粒度	粗粒度
可靠性	考虑	不考虑
聚集问题	性能(时间和扩展性) 科学计算	性能(成本和扩展性) 可获得性 信息/资源共享

第一节：课程介绍

- 什么是并行计算？
- 为什么需要并行计算？
 - 技术趋势
 - 应用程序驱动
- 如何编写并行程序？

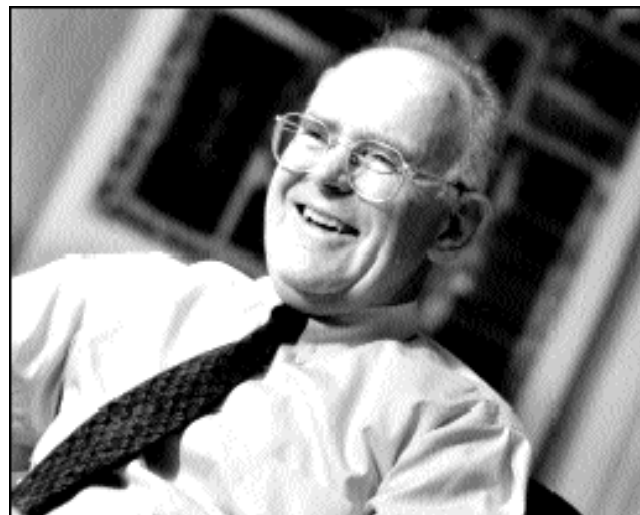


技术趋势：微处理器容量



摩尔定律：
每 1.5 年 晶体管/芯片数翻倍

微处理器变得更小、更密集、
更强大。

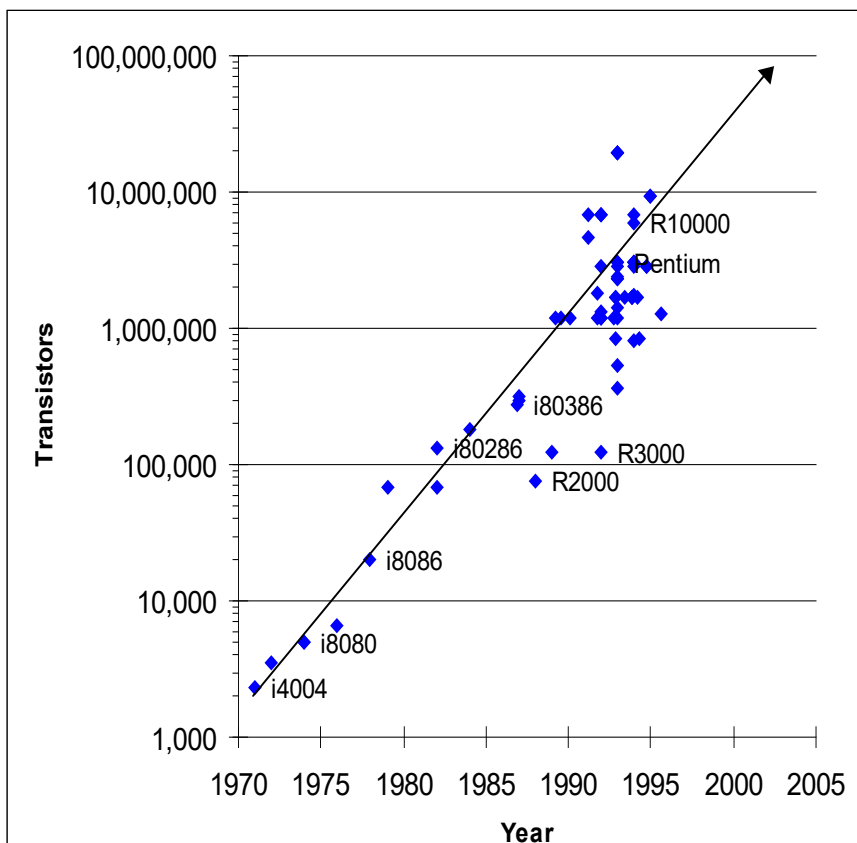


戈登·摩尔（英特尔联合创始人）在1965年预测，半导体芯片的晶体管密度大约每18个月翻一番。

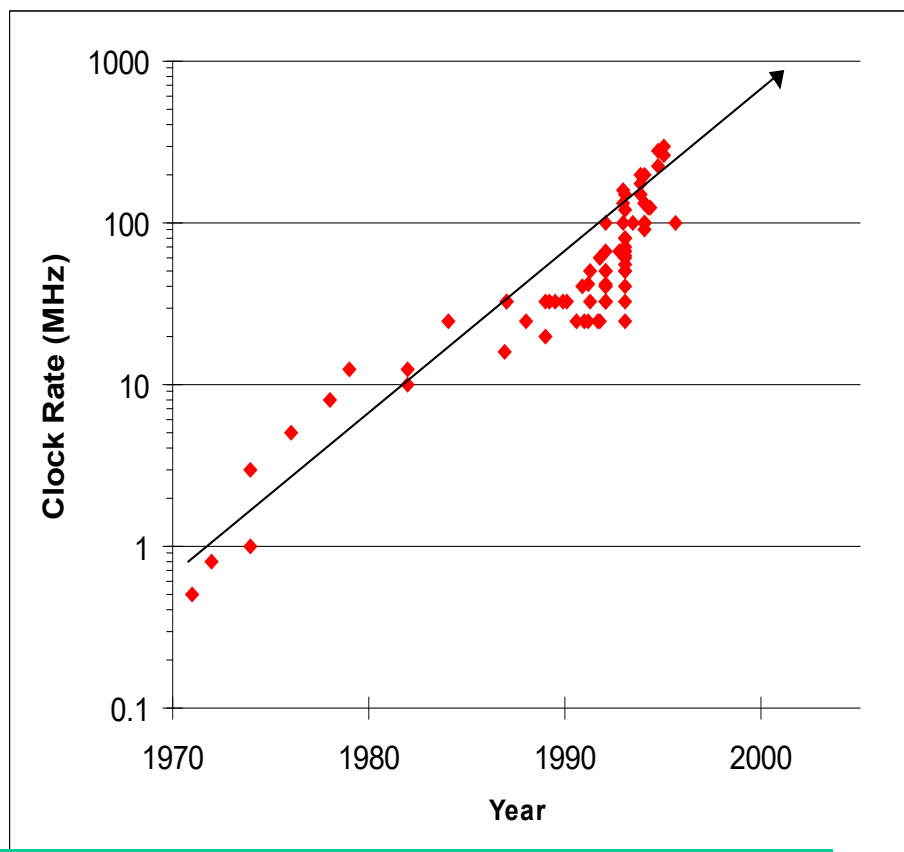


微处理器速度

单个芯片晶体管数的增长



时钟速率的提高

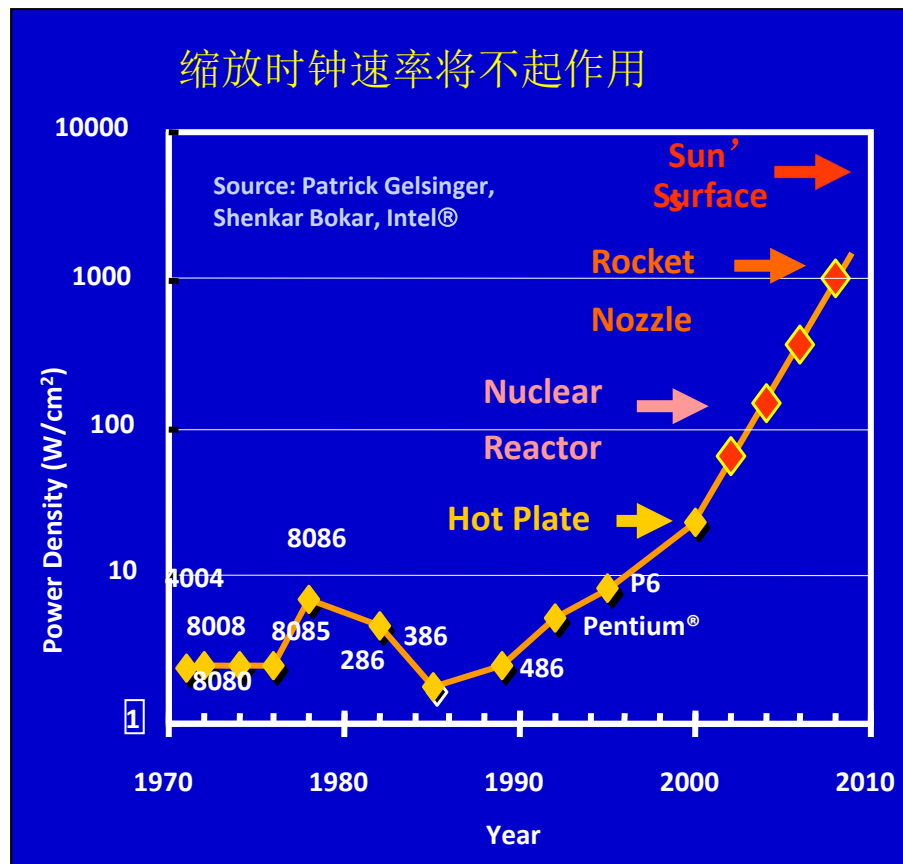


那我们为什么要进行并行编程？只需要等待一两年就好了...



局限一：功率密度

- 并行系统更加节能
 - 动态功耗与 V^2fC 成正比
 - 增加频率 (f) 也会增加电源电压 (V) -> **立方效应**
 - 增加内核只会线性增加电容 (C)
 - 通过降低时钟速度来降低功耗
- 高性能串行处理器能耗大
 - 推测、动态依赖检查等操作功耗大
 - 隐式并行发现



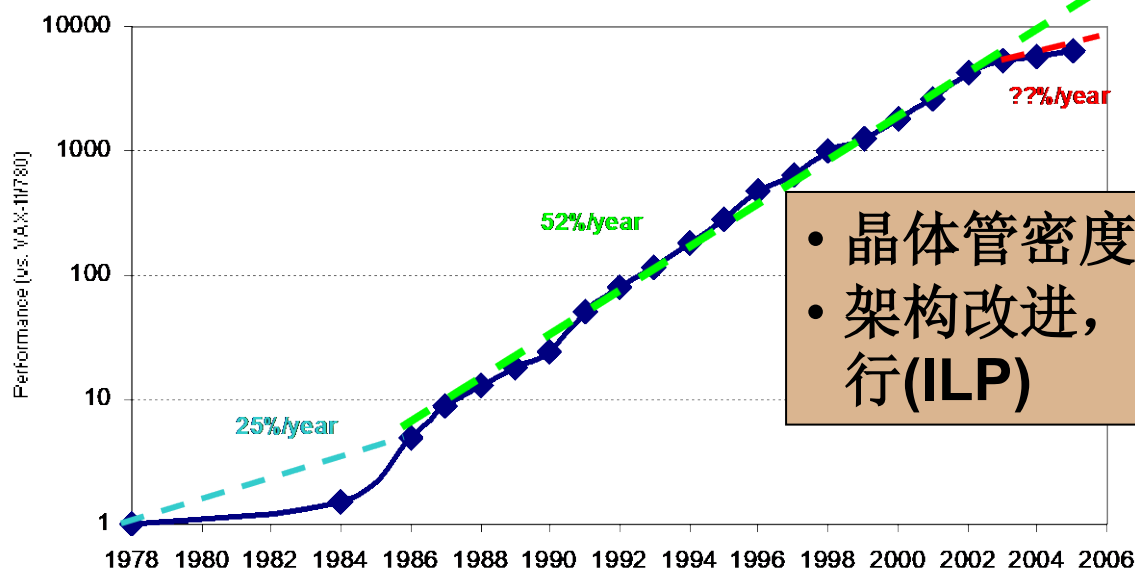
- 更多的晶体管，但不是更快的串行处理器



局限二：ILP发掘

应用程序性能每年增长 52%(SpecInt benchmarks)

摘自Hennessy和Patterson, 《计算机体系结构：
定量方法》，2006年第4版



- 晶体管密度
- 架构改进，如：指令层并行(ILP)

- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002



局限二：ILP能力发掘

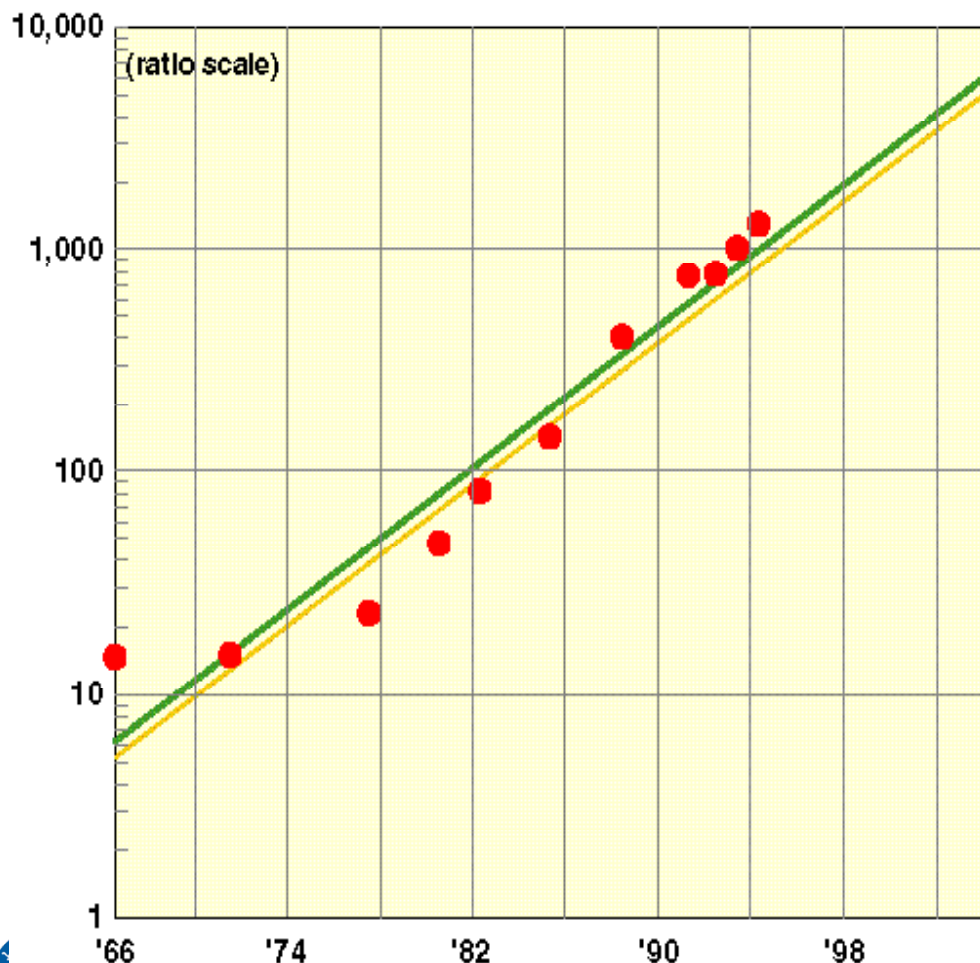
- 超标量 (SS)
 - 多发射指令问题
 - 动态调度：硬件发现指令之间的并行性
 - 推测执行：查看预测的分支
 - 非阻塞缓存：cache未命中时也能发射新的指令
- 对程序员透明的并行形式，程序员并不需要知道ILP
- 不幸的是，ILP进步空间已经很小了



局限三：芯片良率

制造成本和良率问题限制了密度的使用

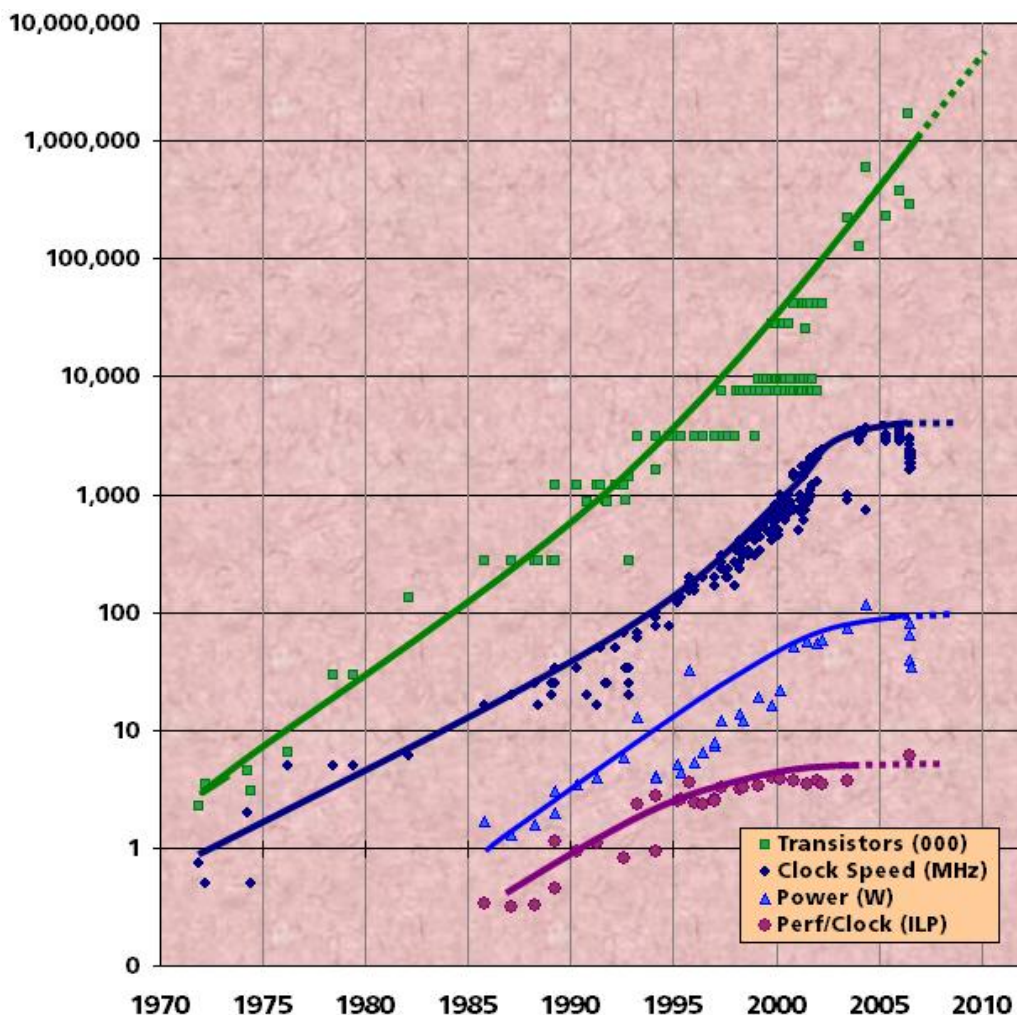
Cost of semiconductor factories in millions of 1995 dollars



- 摩尔（洛克）第二定律：随着时间推移，制造成本会上升
- 而良率（可用芯片百分比）会下降
- **并行化优势**
 - 更小、更简单的处理器更易于设计和验证
 - 部分工作的芯片：Cell处理器（PS3）

现状

- 芯片密度持续增加
 - 而时钟速度几乎不变
 - 处理器内核数可能会增加一倍
- ILP潜力消耗殆尽
- 并行性不能对程序员保持透明！



Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)



2023年的并行

- 所有主要处理器供应商都在生产多核芯片
 - 每台机器都将很快成为一台并行机器
 - 为了保持性能翻倍，并行度必须翻倍
- 哪些（商业）应用程序可以使用这种并行性？
 - 它们必须从头开始重写吗？
- 所有程序员都必须是并行程序员吗？
 - 需要新的软件模型
 - 尝试向大多数程序员隐藏复杂性
 - 同时，部分程序员需要了解它
- 计算机行业必须面对这一重大变化，但并没有所有的答案



Blue Gene/Q Packaging Hierarchy

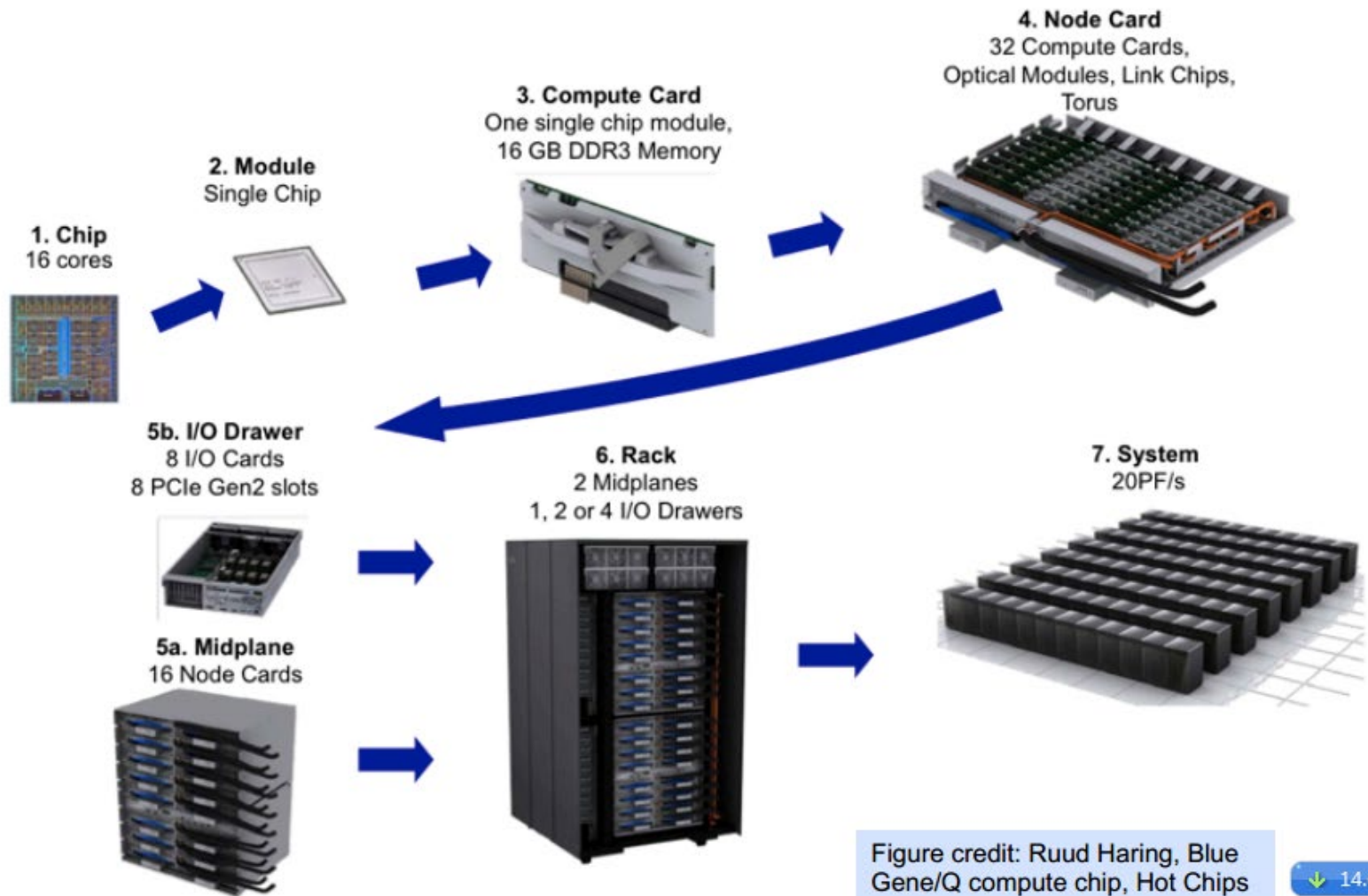


Figure credit: Ruud Haring, Blue Gene/Q compute chip, Hot Chips

14.6KB/S



世界五百强项目

- 世界最强大的500台计算机列表
- 评价标准：Linpack 的 Rmax
 - 求解 $Ax=b$ ，稠密问题，矩阵是随机的
 - 以稠密矩阵的矩阵乘法为主
- 每年更新两次
 - ISC'xy在德国于六月更新
 - SCxy在美国于十一月更新
- 所有信息可从TOP500网站获得： www.top500.org



计量单位

- 高性能计算 (HPC) 单元是：
 - flop: 浮点运算, 通常是双精度, 除非特别注明
 - flop/s: 每秒的浮点运算
 - Bytes: 数据大小 (双精度浮点数为8)
- 典型的大小是数百万、数十亿、数万亿.....

Mega	Mflop/s = 10^6 flop/sec	Mbyte = $2^{20} = 1048576 \sim 10^6$ bytes
Giga	Gflop/s = 10^9 flop/sec	Gbyte = $2^{30} \sim 10^9$ bytes
Tera	Tflop/s = 10^{12} flop/sec	Tbyte = $2^{40} \sim 10^{12}$ bytes
Peta	Pflop/s = 10^{15} flop/sec	Pbyte = $2^{50} \sim 10^{15}$ bytes
Exa	Eflop/s = 10^{18} flop/sec	Ebyte = $2^{60} \sim 10^{18}$ bytes
Zetta	Zflop/s = 10^{21} flop/sec	Zbyte = $2^{70} \sim 10^{21}$ bytes
Yotta	Yflop/s = 10^{24} flop/sec	Ybyte = $2^{80} \sim 10^{24}$ bytes
- 当前最快的 (公共) 机器 ~ 188Pflop/s
 - 最新排名访问: www.top500.org

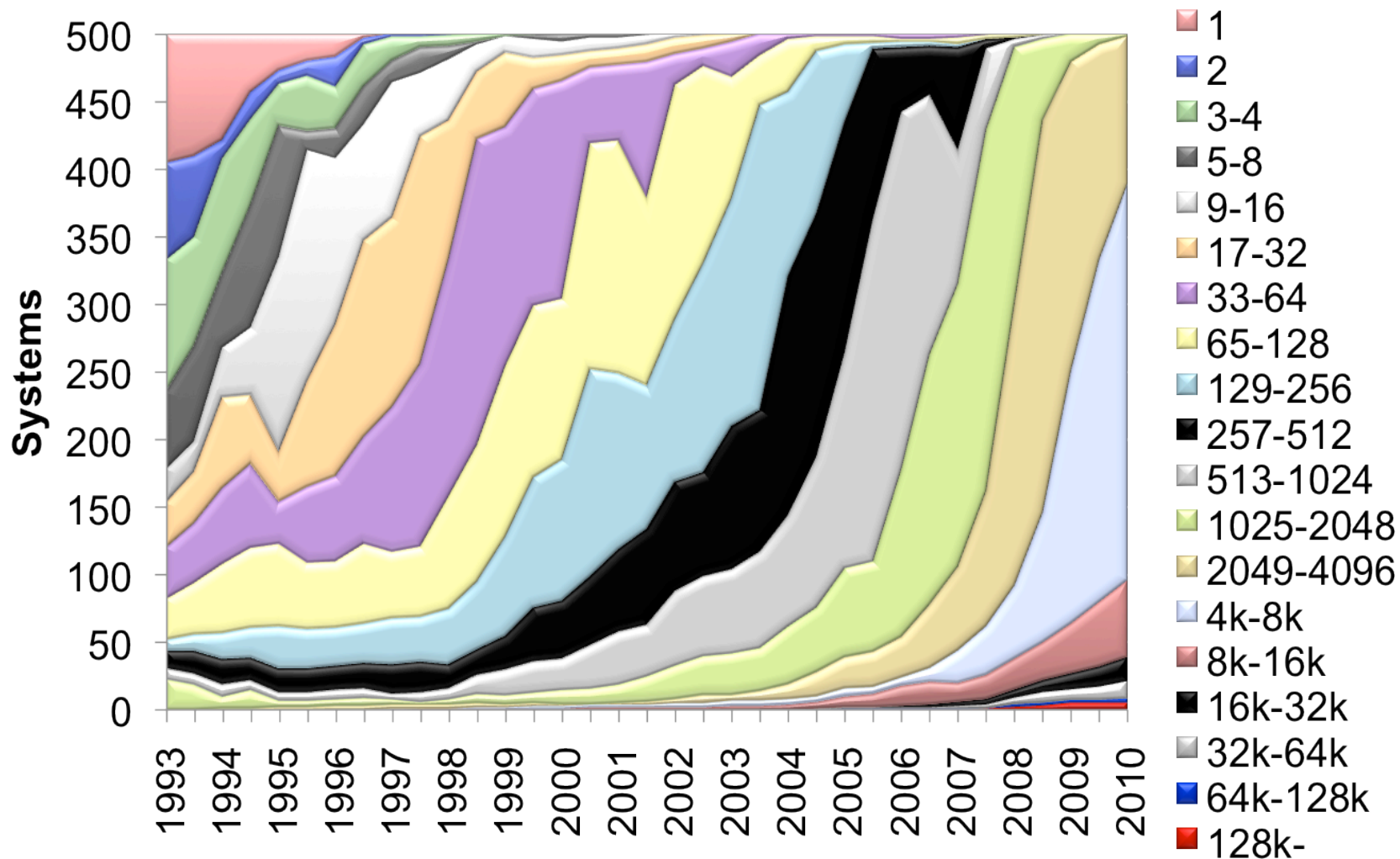


2021十二月TOP500(TOP500.ORG)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.00	537,212.00	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.00	200,794.90	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.00	125,712.00	7,438
4	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.60	125,435.90	15,371
5	Perlmutter - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10 DOE/SC/LBNL/NERSC United States	761,856	70,870.00	93,750.00	2,589
6	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, NVIDIA Corporation NVIDIA Corporation United States	555,520	63,460.00	79,215.00	2,646
7	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, National Super Computer Center in Guangzhou National Super Computer Center in Guangzhou China	4,981,760	61,444.50	100,678.70	18,482
8	JUWELS Booster Module - Bull Sequana XH2000, AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.00	70,980.00	1,764
9	HPC5 - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband Eni S.p.A. Italy	669,760	35,450.00	51,720.80	2,252
10	Voyager-EUS2 - ND96amsr_A100_v4, AMD EPYC 7V12 48C 2.45GHz, NVIDIA A100 80GB, Mellanox Azure East US 2 United States	253,440	30,050.00	39,531.20	



核心数



重新解读摩尔定律

- 每个芯片的内核数量每两年翻一番
- 时钟速度不会增加（甚至可能降低）
- 需要处理具有数百万个并发线程的系统
- 需要处理芯片间并行性和片内并行性



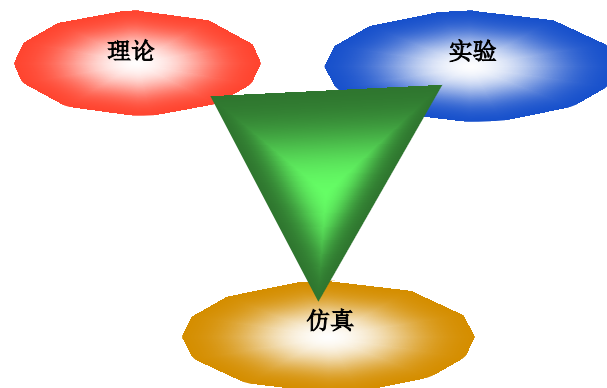
应用程序驱动变革

- 计算能力持续呈指数级增长
 - 可以模拟理论和实验做不到的事情
 - 求解深度神经网络 (DNN)更为简单
- 数据持续呈指数级增长
 - 科学大装置每天产生巨量数据
 - 互联网时代，随着社交媒体的出现，数据量激增，内容以图像和文本为主
 - 物联网时代，数据几何倍数增长



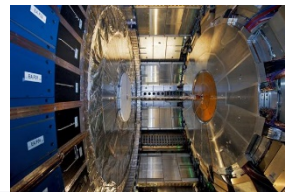
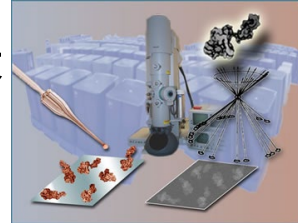
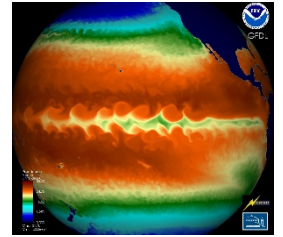
仿真：科学的第三大支柱

- 传统的科学和工程方法：
 - 做理论或论文设计
 - 进行实验或构建系统
- 限制：
 - 太难——需要建造大型风洞
 - 太贵——需要建造一架一次性客机
 - 太慢——需要等待气候或星系演化
 - 太危险——武器、药物设计、气候试验
- 计算科学与工程范式：
 - 利用计算机对现象进行模拟分析
 - 基于已知的物理定律和高效的数值方法
 - 使用超出手动分析范围的计算工具和方法分析模拟结果。



数据驱动科学

- 科学数据集呈指数级增长
 - 生成数据的能力超出了我们存储和分析的能力
 - 模拟系统和一些观测设备的能力随着摩尔定律而增长
- PB 数据集很快就会普及：
 - 气候建模：下一个 IPCC 数据的估计数为 10 PB
 - 基因组：仅 JGI 今年就有 0.5 PB 的数据，并且每年翻一番
 - 粒子物理学：大型强子对撞机预计每年产生 16 PB 的数据
 - 天体物理学：LSST 和其他将产生 5 PB/年（通过 3.2 Gigapixel 相机）
- 通过数据的“科学门户”创建科学社区



一些特别具有挑战性的计算

- 科学
 - 全球气候建模
 - 生物学：基因组学； 蛋白质折叠； 药物设计
 - 天体物理建模
 - 计算化学
 - 计算材料科学和纳米科学
- 工程
 - 半导体设计
 - 地震和结构建模
 - 计算流体动力学（飞机设计）
 - 燃烧（发动机设计）
 - 碰撞模拟
- 商业
 - 金融和经济建模
 - 交易处理、网络服务和搜索引擎
- 防御
 - 核武器 - 模拟试验
 - 密码学



数据驱动的人工智能

• 算法

- 深度神经网络 (DNN) 求解器和卷积神经网络 (CNN) 等技术的改进使得它们更简单易用
- 2012 年推出的 AlexNet 架构

• 数据

- 每天产生 250 亿亿字节数据
- 随着社交媒体的出现，数据量激增，内容以图像和文本为主

• 算力

- 算力的进步提供了在合理时间内解算 DNN 所需的计算能力
- GPU 让数据科学家能更快地构建更好的模型。与使用 CPU 相比，数据科学家使用 GPU 可以运行更多的试错



• GPT-3: 175 billion parameters

• Cost (2020): \$4.6 million

• GPT-4 (Human Brain): 100 trillion parameters

• Cost (2020): \$2.6 billion

• Cost (2024): \$325 million

• Cost (2028): \$40 million

• Cost (2032): \$5 million



第一节：课程介绍

- 什么是并行计算？
- 为什么需要并行计算？
- 并行计算中的问题
- 如何编写并行程序？



架构问题

- 流水线、ILP.....
- 缓存一致性
- 连接：单一共享总线与网络
- UMA、NUMA、CC-NUMA、集群.....



编程模型问题

- 数据共享——单个地址空间与多个地址空间。
- 进程协调——使用锁、消息和其他方式进行同步。
- 分布式与集中式内存。
- 容错/可靠性。



性能问题

- 指标：规模、加速、可扩展性.....
- 模型：PRAM, BSP, PPRAM...
- 评估并行算法的方法



其他问题

- 并行计算机语言
- 并行编程工具
- 可移植的并程序序
- 并行计算机自动编程
- 并行计算教育方法
-



第一节：课程介绍

- 什么是并行计算？
- 为什么需要并行计算？
- 并行计算中的问题
- 如何编写并行程序？



什么是并行编程

- 并行编程是一种语言编程，允许你明确指示不同处理器如何同时执行计算的不同部分。
- Parallel Programming is programming in a language that allows you to explicitly indicated how different portions of the computation may be executed concurrently by different processors.



串行程序并行化

- 重写并行程序
- 编译指示技术
- 翻译程序（自动并行）



例子

- 计算 n 个值并将它们相加。
- 串行方案：

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



例子

- 我们有 p 个核心, p 比 n 小得多。
- 每个内核执行大约的部分求和

```
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;
}
```

每个核心都使用自己的私有变量并独立于其他核心执行此代码块。



例子

- 每个内核完成代码执行后，其私有变量 `my_sum` 将存储调用 `Compute_next_value` 计算的值的总和。
- Ex., 8 核, $n=24$, 调用 `Compute_next_value` 返回: 1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

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



例子

- 一旦所有核心都完成了它们的私有 my_sum 计算，它们就会通过将结果发送到指定的“主”核心来形成全局总和，该核心将最终结果相加。

```
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;  
}
```



例子

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

- 全局和
- $8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95$

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

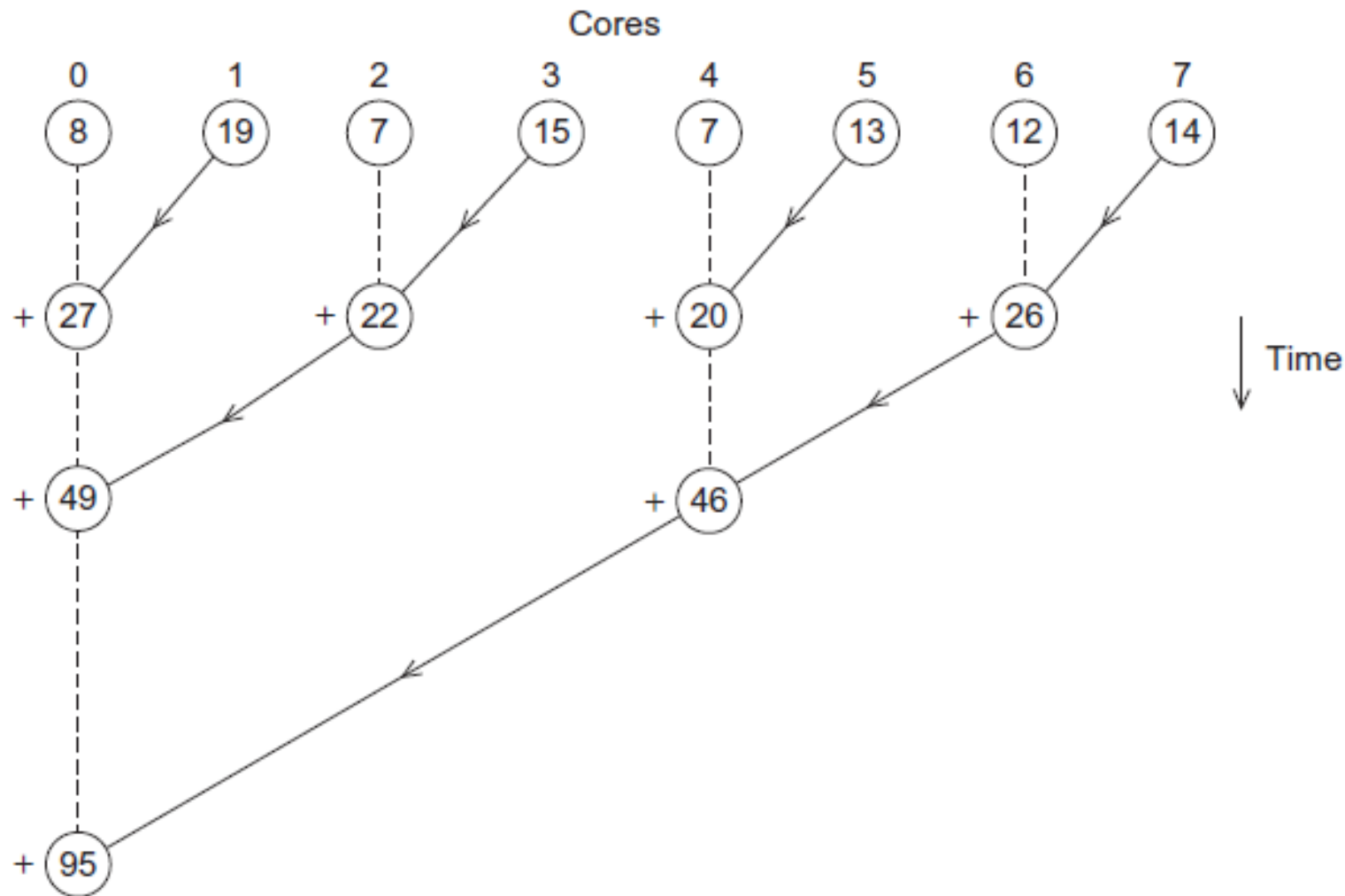


更好的并行算法

- 不要让主核心承担所有工作，将其与其他内核共享。使用奇偶数对内核进行操作。
- 将内核配对，使内核0将其结果与内核1的结果相加，内核2将其结果与内核3的结果相加，以此类推。
- 只使用排名为偶数的内核重复该过程。内核0添加来自内核2的结果，内核4添加来自内核6的结果，以此类推。
- 被4整除的内核重复该过程，直到内核0拥有最终结果。



多个内核生成全局和



分析

- 在第一个例子中，主核心执行7次接收和7次加法
- 在第二个例子中，主核心执行3次接收和3次加法
- 改进超过2倍！



分析

- 如果有更多的内核，差异将更加明显。
- 如果我们有1000个内核：
 - 第一个例子将需要主核心执行999次接收和999次加法。
第二个例子只需要执行10次接收和10次加法。
- 这是近100倍的改进！



分析

- 第一个全局求和是串行全局求和的明显推广。第二个全局求和与原始串行加法关系不大。
- 翻译程序不太可能“发现”第二个全局求和。翻译程序可以“识别”原始串行循环，并用预编码的高效并行全局求和替换它。
- 对于越来越复杂的串程序，识别结构变得越来越困难，预编码高效并行化的可能性也越来越小
- 因此我们必须编写并程序，利用多处理器的能力



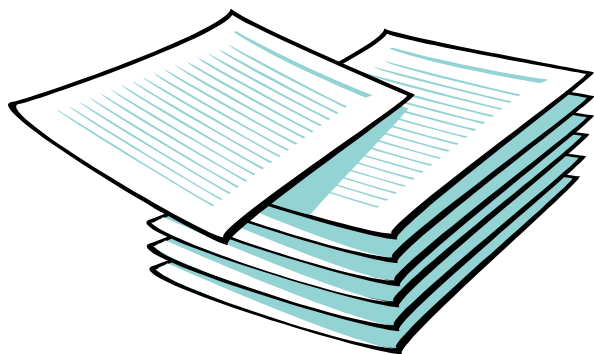
我们该如何编写并行程序

- 任务并行：将解决问题所需的不同任务分配给不同的核心。
- 数据并行：
 - 将用于解决问题的数据分配给不同的核心
 - 每个核心对其数据的部分执行类似的操作。



A 教授

- 15个问题
- 300名学生
- 300份试卷



A 教授的助教



TA#1

TA#2

TA#3



电子科技大学
University of Electronic Science and Technology of China

并行模式 — 任务并行



TA#1

Questions 1 - 5



TA#3

Questions 11 - 15



Questions 6 - 10

TA#2

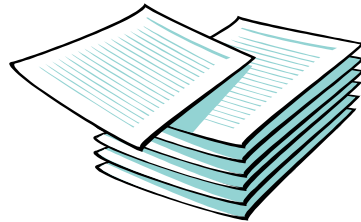


电子科技大学
University of Electronic Science and Technology of China

并行模式 — 数据并行

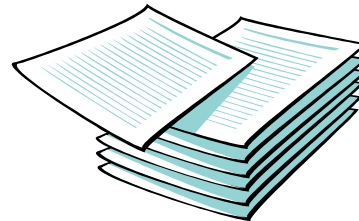
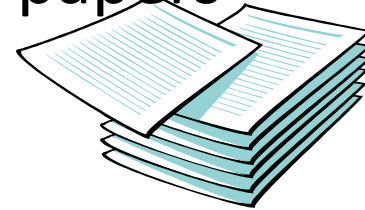
TA#1

100 exam Papers



TA#3

100 exam papers



TA#2

100 exam papers



并行模式—数据求和

- 数据并行性
 - 数据是由 `Compute_next_value` 计算得到的值，每个核心对其分配的元素执行大致相同的操作：通过调用 `Compute_next_value` 计算所需的值并将它们相加。
- 任务并行性
 - 两个任务：主核心执行接收与相加部分和；其他核心将部分和提供给主核心



同步

- 处理器通常需要协调它们的工作
- **通信** (Communication) : 一个或多个核心将它们当前的部分总和发送给另一个核心
- **负载均衡** (Load Balancing) : 均匀地分配工作给各个核心, 以避免其中一个核心过于繁重
- **同步** (Synchroniaztion): 由于每个核心以自己的节奏工作, 需要确保核心不会超前于其他核心



总结

- 阅读材料：
 - “The Landscape of Parallel Research: The View from Berkeley”

