

# 并行与分布式计算基础 I：基础理论

杨超

chao\_yang@pku.edu.cn

2023 秋



# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 内容提纲

① 课程介绍

② 引言

③ 硬件体系架构

④ 小议并行算法与编程

⑤ 并行计算三大定律

⑥ 共享内存并行计算模型

⑦ 分布式并行计算模型

⑧ 并行编程模型

# 课程基本信息

- 课程名称：并行与分布式计算基础
- 英文名称：Foundations of Parallel and Distributed Computing
- 授课教师：杨超 (chao\_yang@pku.edu.cn, 理科 1 号楼 1520)
- 课程助教 1：张晨晨 (zhangchenchen@stu.pku.edu.cn)
- 课程助教 2：刘思然 (liusr25@stu.pku.edu.cn)
- 学分：3, 周学时：3, 总学时：48
- 授课对象：计算数学、数据科学、应用统计、计算机等专业
- 先修课程：计算概论、数据结构、C/C++ 语言、机器学习等
- 考核方式：平时作业 (50%) + 期末考试 (50%)
- 教材：无

# 课程定位

## 基本定位

- 在过去的十年里，平行与分布式计算的需求正经历爆炸式增长，已经从一门选修课程变成了计算科学和数据科学课程体系的核心组成部分；
- 本课程包含关于平行与分布式计算的计算模型理论的基本概念和分布式内存体系架构上的 MPI 编程技术、共享内存体系架构上的 OpenMP 编程技术以及在 GPU 众核体系架构上的 CUDA 编程技术等；
- 通过本课程的学习，将对平行与分布式计算的基础理论、编程方法及其在计算科学和数据科学中的应用有较为系统性的了解，从而提高算法设计、编程与应用等相关能力。

# 授课内容

## 主要内容

- 引言
- 硬件体系架构
- 并行计算模型
- 编程与开发环境
- MPI 编程与实践
- OpenMP 编程与实践
- GPU 编程与实践
- 前沿问题选讲

# 上课时间 (地点: 地学 108)

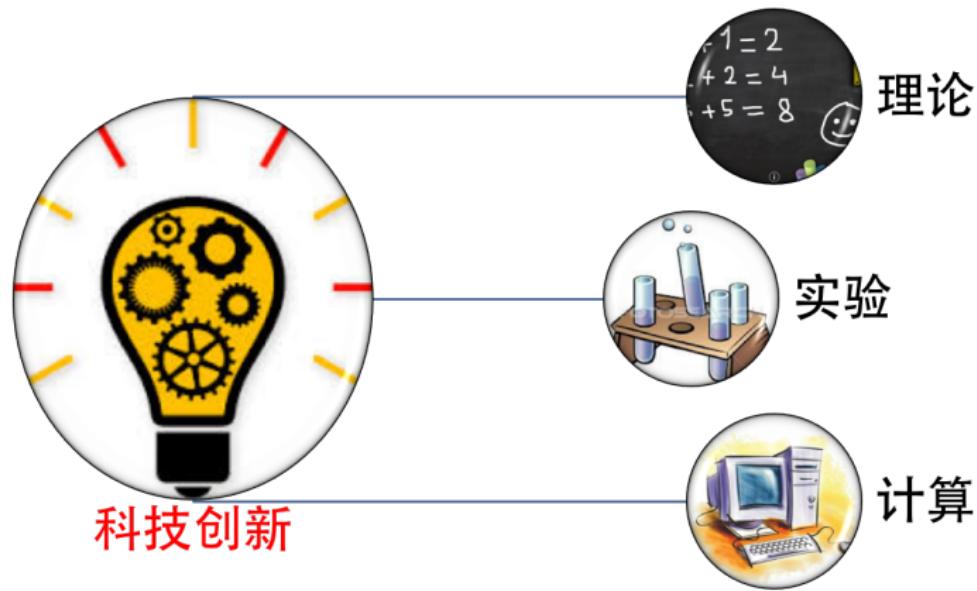
| 上课时间                 | 星期一 | 星期二 | 星期三 | 星期四 | 星期五 |
|----------------------|-----|-----|-----|-----|-----|
| 第 1 节 (8:00-8:50)    |     |     |     |     |     |
| 第 2 节 (9:00-9:50)    |     |     |     |     |     |
| 第 3 节 (10:10-11:00)  |     |     |     |     | 单周  |
| 第 4 节 (11:10-12:00)  |     |     |     |     | 单周  |
| 第 5 节 (13:00-13:50)  |     | 每周  |     |     |     |
| 第 6 节 (14:00-14:50)  |     | 每周  |     |     |     |
| 第 7 节 (15:10-16:00)  |     |     |     |     |     |
| 第 8 节 (16:10-17:00)  |     |     |     |     |     |
| 第 9 节 (17:10-18:00)  |     |     |     |     |     |
| 第 10 节 (18:40-19:30) |     |     |     |     |     |
| 第 11 节 (19:40-20:30) |     |     |     |     |     |
| 第 12 节 (20:40-21:30) |     |     |     |     |     |

# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

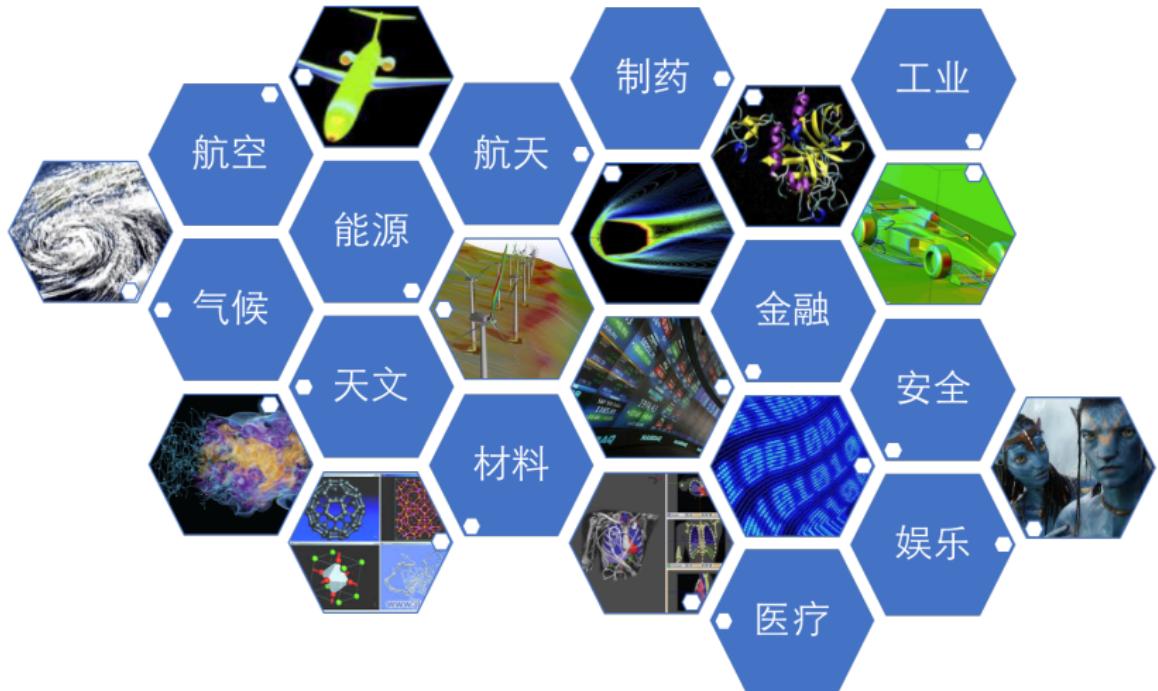
# 计算的重要性

- 计算已经成为科技创新的第三大手段



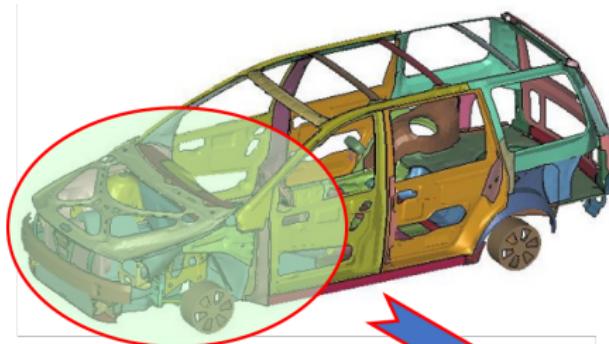
# 计算的普遍性

- 计算已经在各行各业中大显身手

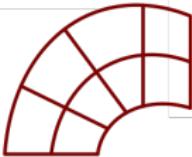
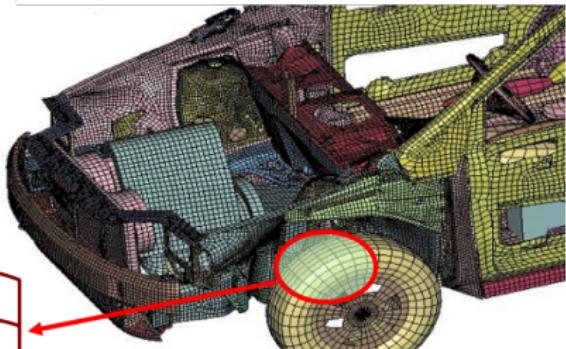


# 举例：汽车碰撞仿真

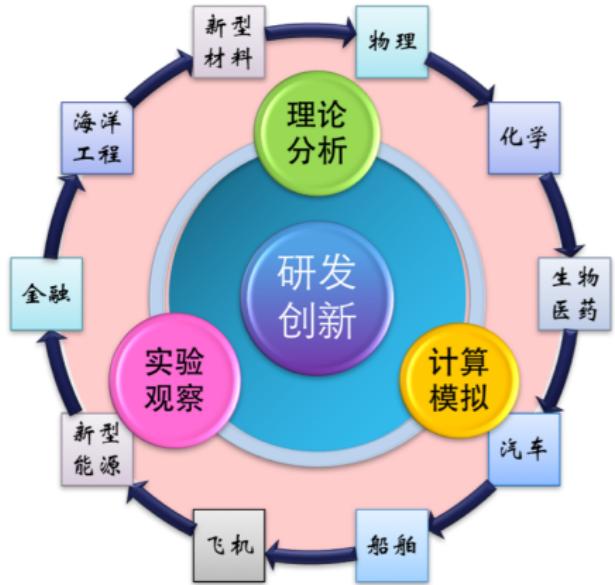
- 每一种车型的汽车出厂前都需要进行大量安全性测试，其中，通过计算机进行汽车碰撞的数值仿真实验，可以相当程度地代替真车实验，大幅度节省开支。



$$\frac{d}{dt} \left\{ \frac{\partial E_K}{\partial \dot{z}_i} \right\} - \frac{\partial E_K}{\partial z_i} + \frac{\partial E_P}{\partial z_i} + \frac{\partial E_D}{\partial \dot{z}_i} = F_{z_i}, \\ i = 1, 2, \dots, 8,$$



# 计算的复杂性



## 应用领域的强烈需求

- 人口愈来愈多
- 数据愈来愈大
- 节奏愈来愈快
- 灾害愈来愈频
- 雾霾愈来愈重
- ...

# 串行计算 vs 并行计算

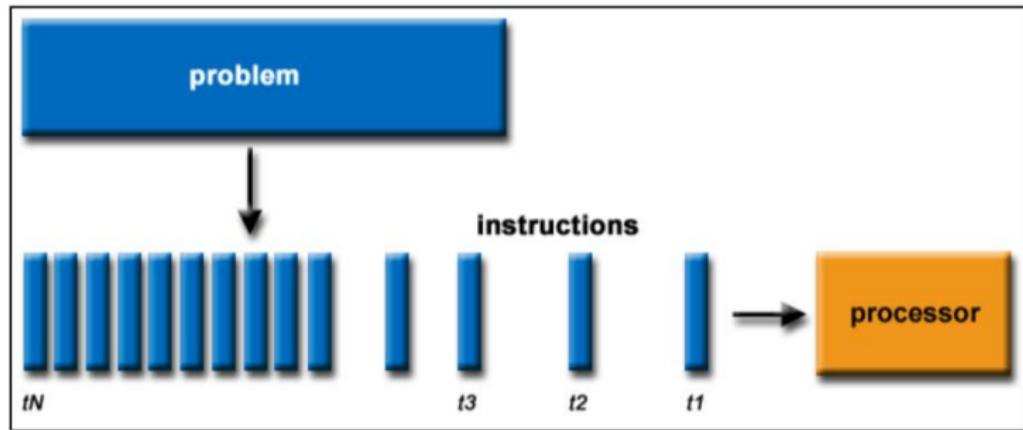
- 串行计算将问题被分为一系列独立指令，按照先后顺序逐一执行；
- 并行计算则将问题分为能并发执行的若干部分，分别串行执行。



# 什么是并行计算?

## 串行计算

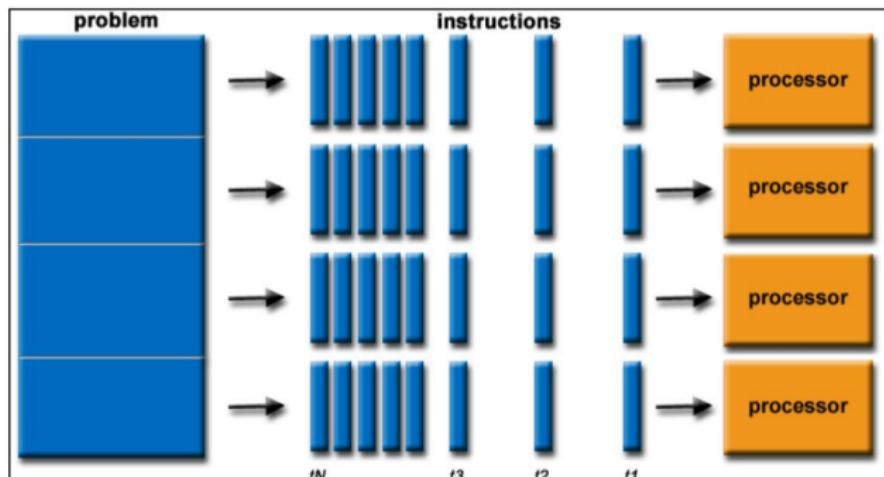
- 问题被分为一系列独立指令
- 指令按照先后顺序逐一执行
- 一般只在一个处理器上执行
- 在任何时间只有一条指令执行



# 什么是并行计算?

## 并行计算

- 问题被分为能够并发执行部分
- 每个部分进一步被分为一系列指令
- 来自不同部分的指令可以同时在不同处理器上执行
- 需要全局协同机制



# 为什么并行计算

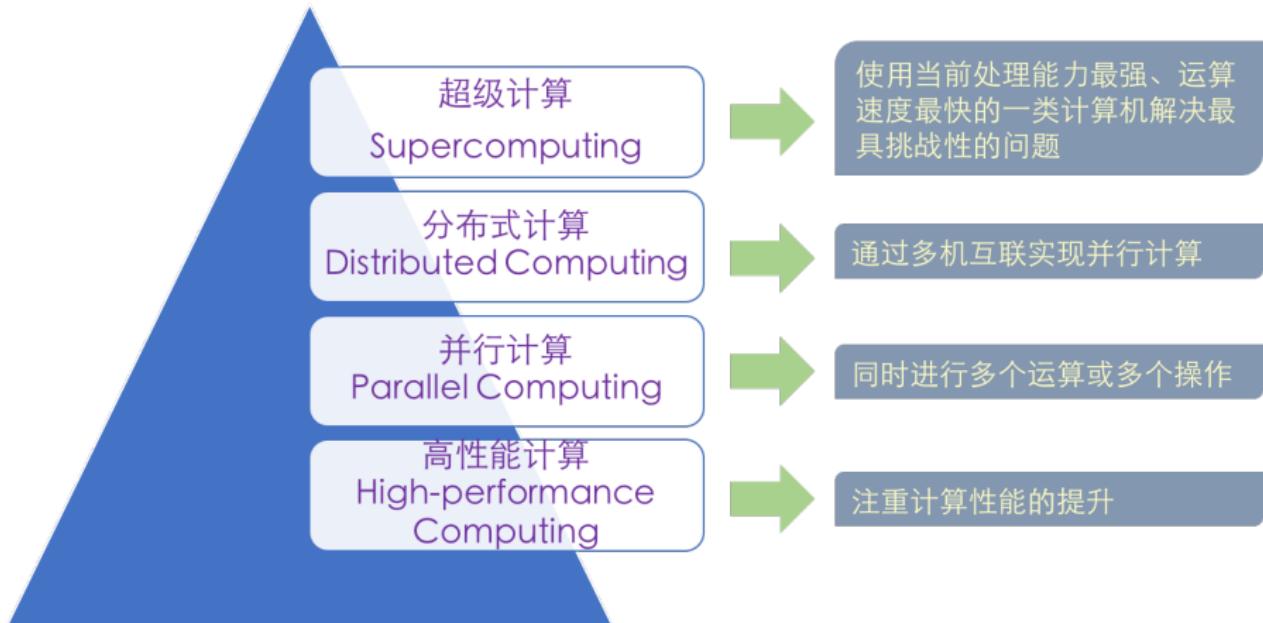
## 加速求解问题的速度

给定某应用，在单处理器上，串行执行需要 2 个星期（14 天），这个速度对一般的应用而言，是无法忍受的。而借助并行计算，使用 100 台处理器，如果加速 50 倍，将执行时间缩短为 6.72 个小时。

## 提高求解问题的规模

在单处理器上，受内存资源的限制，只能求解 10 万个未知数，但是当前数值模拟要求计算千万个未知数。于是，也可以借助并行计算，使用 100 个处理器，将问题求解规模线性地扩大 100 倍。

# 高性能计算、并行计算、分布式计算以及超级计算

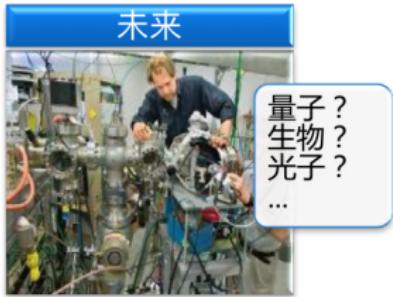


# 计算能力与存储能力的度量

| 前缀     | 简称 | 量级        | 计算能力                | 存储能力           |
|--------|----|-----------|---------------------|----------------|
| Kilo-  | K  | $10^3$    | KiloFLOPS (KFLOPS)  | KiloByte (KB)  |
| Mega-  | M  | $10^6$    | MegaFLOPS (MFLOPS)  | MegaByte (MB)  |
| Giga-  | G  | $10^9$    | GigaFLOPS (GFLOPS)  | GigaByte (GB)  |
| Tera-  | T  | $10^{12}$ | TeraFLOPS (TFLOPS)  | TeraByte (TB)  |
| Peta-  | P  | $10^{15}$ | PetaFLOPS (PFLOPS)  | PetaByte (PB)  |
| Exa-   | E  | $10^{18}$ | ExaFLOPS (EFLOPS)   | ExaByte (EB)   |
| Zetta- | Z  | $10^{21}$ | ZettaFLOPS (ZFLOPS) | ZettaByte (ZB) |
| Yotta- | Y  | $10^{24}$ | YottaFLOPS (YFLOPS) | YottaByte (YB) |

其中，FLOPS (flops or flop/s) 指每秒浮点运算次数：floating point operations per second.

# 计算机的发展历史



# 萌芽：真空管电子计算机

研发主要受第二次世界大战和冷战的军事需求推动。

## Colossus

英国研制它来破解截获纳粹德国的无线电报信息。

## ENIAC

美国研制它来快速计算炮兵射击表。

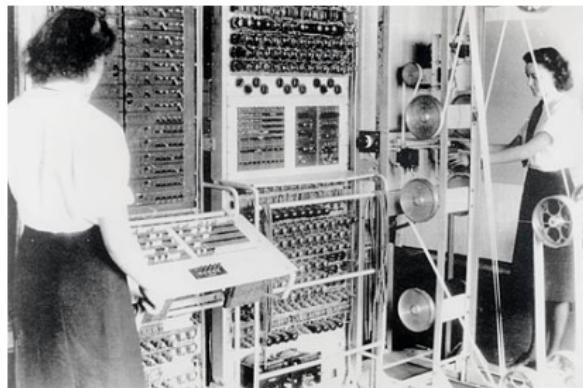


图: Colossus, 1943, 英国

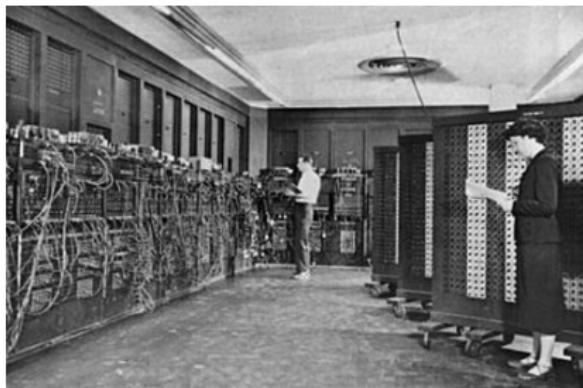


图: ENIAC, 1945, 美国

# 早期：向量机

向量机（Vector Machine）使用单个指令同时处理多份数据。

## Cray-1 向量机：

| 处理器个数 | 处理器频率 | 内存大小    | 存储大小  | 性能         |
|-------|-------|---------|-------|------------|
| 1     | 80MHz | 8.39 MB | 303MB | 160 MFLOPS |



图: Cray-1, 1976

# 中期：并行向量处理机

并行向量处理机（Parallel Vector Processors, PVP）包括多个向量机，并通过共享内存实现交互。

## Cray-2 多处理机：

| 处理器个数 | 处理器频率  | 内存大小   | 总性能        |
|-------|--------|--------|------------|
| 4     | 244MHz | 256 MB | 1.9 GFLOPS |

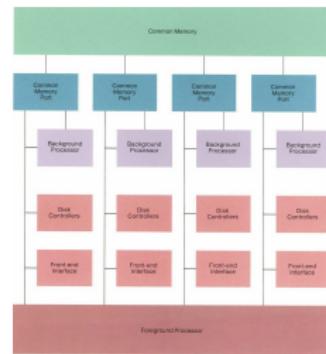
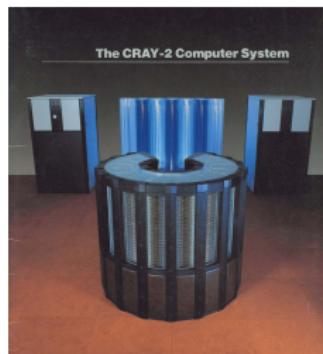


图: Cray-2, 1985

# 中后期：分布式并行机

分布式并行机（Parallel Processors, PP）通过高性能网络连接多个分布式存储节点，每个节点由商用微处理芯片组成。

## Intel Paragon XP/S 140 并行机：

| 处理器个数 | 处理器性能     | 处理器频率  | 内存大小   | 访存带宽     | 网络带宽     | 总性能        |
|-------|-----------|--------|--------|----------|----------|------------|
| 3680  | 75 MFLOPS | 50 MHz | 128 MB | 400 MB/s | 175 MB/s | 143 GFLOPS |



图: Intel Paragon XP/S 140, 1994

# 中后期：对称多处理机

对称多处理机（Symmetric Multiprocessors, SMP）通过高性能网络连接多个高性能微处理芯片，芯片之间通过共享内存交互。

## SUN Ultra E10000 多处理机：

| 处理器个数 | 处理器性能    | 处理器频率   | 内存大小 | 网络带宽      | 总性能       |
|-------|----------|---------|------|-----------|-----------|
| 64    | 1 GFLOPS | 250 MHz | 64GB | 12.8 GB/s | 25 GFLOPS |



图: SUN Ultra E10000, 1997

# 中后期：分布式共享并行机

分布式共享并行机（Distributed Share Memory, DSM）通过高性能网络连接多个高性能微处理芯片，每个芯片拥有局部内存，但所有局部内存都能实现全局共享。

## SGI Origin 2000 并行机：

| 处理器个数 | 处理器频率   | 内存大小  | 网络带宽     | 总性能       |
|-------|---------|-------|----------|-----------|
| 128   | 195 MHz | 512GB | 1.5 GB/s | 50 GFLOPS |

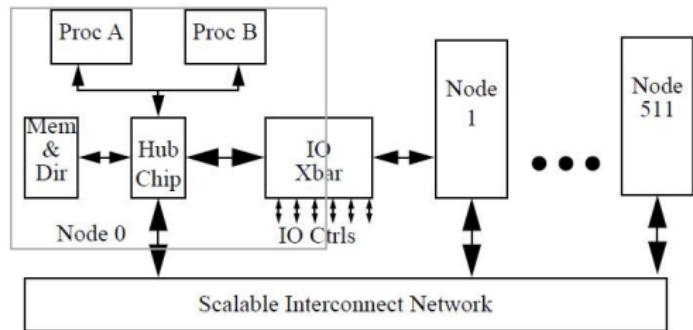


图: SGI Origin 2000, 2002

# 近期：大规模并行机

大规模并行机（Massively Parallel Processors, MPP）通过高性能网络连接上万个分布式存储节点，每个节点包含多个高性能芯片。

## Intrepid(Blue Gene/P) 并行机：

| 节点个数  | 节点核数 | 节点性能        | 处理器频率  | 网络带宽    | 总内存   | 总性能        |
|-------|------|-------------|--------|---------|-------|------------|
| 40960 | 4    | 13.6 GFLOPS | 850MHz | 88 GB/s | 80 TB | 557 TFLOPS |

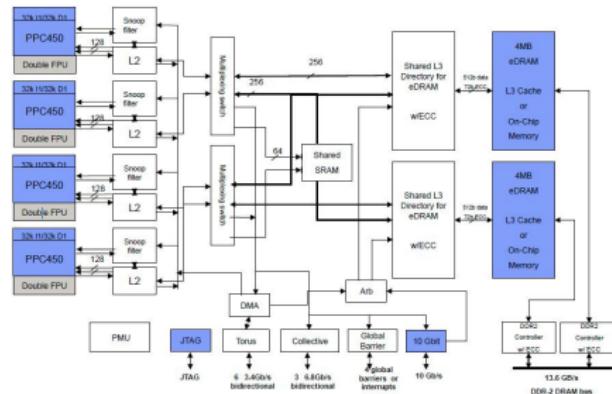


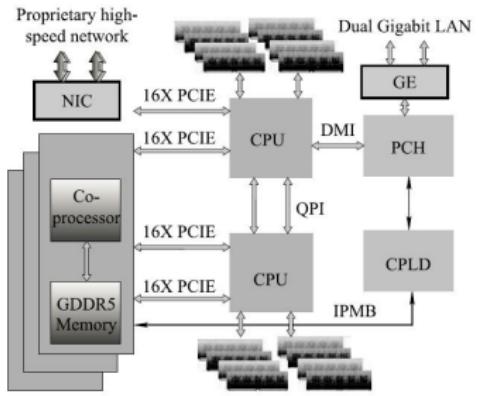
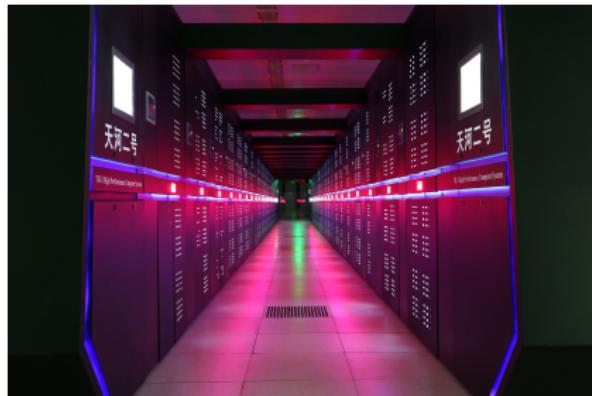
图: Intrepid(Blue Gene/P) , 2008

# 近期：大规模加速并行机

大规模加速并行机（Massively Parallel Processors with Accelerators, MPPA）通过高性能网络连接上万个分布式存储节点，每个节点包含多个拥有加速器的高性能芯片。

## 天河 2 号：

| 节点个数   | 节点核数 | 节点性能       | 处理器频率   | 加速器          | 网络带宽    | 总内存     | 总性能         |
|--------|------|------------|---------|--------------|---------|---------|-------------|
| 16,000 | 195  | 3.4 TFLOPS | 2.2 GHz | 3 × Xeon Phi | 12 GB/s | 1.34 PB | 54.9 PFLOPS |



图：天河 2 号，2013

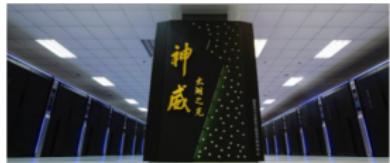
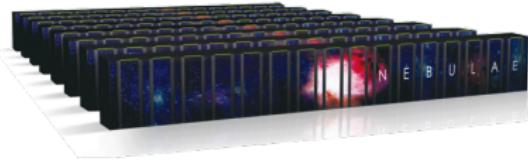
# TOP500 排名

- TOP500 榜单每年更新两次，列举公开发布的最强性能的超级计算机前 500 名。
- 排榜根据 LINPACK 基准测试结果，程序核心是利用高斯消去法求解稠密线性系统。
- 我国联想、曙光、天河、神威超级计算机多次入围 TOP500 前列。
- 我国在 TOP500 中的系统份额已经超过美国，达到世界第一。
- 网址：<https://www.top500.org/>



# 国产计算机 TOP500 入榜情况

- 联想深腾系列
  - 2003年11月，深腾 6800 Top500 第14位
  - 2009年6月，深腾7000 Top500 第19位
- 曙光系列
  - 2004年6月，曙光4000A Top500 第10位
  - 2009年6月，曙光5000A（魔方）Top500 第10位
  - 2010年11月，曙光6000（星云）Top500 第2位
- 天河（银河）系列
  - 2010年11月，天河1A Top500 第1位
  - 2013年6月、11月，天河2 Top500 第1位
  - 2014年6月、11月，天河2 Top500 第1位
  - 2015年6月、11月，天河2 Top500 第1位
- 神威（神州）系列
  - 2016年6月、11月，神威太湖之光 Top500 第1位
  - 2017年6月、11月，神威太湖之光 Top500 第1位



# 2020 年度 TOP500 排名

| #  | Site   | Manufacturer | Computer   | Country     | Cores      | Rmax [PFlops] | Power [MW] |
|----|--|--------------|--|-------------|------------|---------------|------------|
| 1  | RIKEN Center for Computational Science           | Fujitsu      | Fugaku<br>Supercomputer Fugaku,<br>A64FX 48C 2.2GHz, Tofu interconnect D               | Japan       | 7,299,072  | 415.5         | 28.3       |
| 2  | Oak Ridge National Laboratory                    | IBM          | Summit<br>IBM Power System,<br>P9 22C 3.07GHz, Mellanox EDR, NVIDIA GV100              | USA         | 2,414,592  | 148.6         | 10.1       |
| 3  | Lawrence Livermore National Laboratory           | IBM          | Sierra<br>IBM Power System,<br>P9 22C 3.1GHz, Mellanox EDR, NVIDIA GV100               | USA         | 1,572,480  | 94.6          | 7.4        |
| 4  | National Supercomputing Center in Wuxi           | NRCPC        | Sunway TaihuLight<br>NRCPC Sunway SW26010,<br>260C 1.45GHz                             | China       | 10,649,600 | 93.0          | 15.4       |
| 5  | National University of Defense Technology        | NUDT         | Tianhe-2A<br>ANUDT TH-IVB-FEP,<br>Xeon 12C 2.2GHz, Matrix-2000                         | China       | 4,981,760  | 61.4          | 18.5       |
| 6  | Eni S.p.A  | Dell EMC     | HPC5<br>PowerEdge C4140,<br>Xeon 24C 2.1GHz, NVIDIA T. V100, Mellanox HDR              | Italy       | 669,760    | 35.5          | 2.25       |
| 7  | NVIDIA Corporation                               | NVIDIA       | Selene<br>DGX A100 SuperPOD,<br>AMD 64C 2.25GHz, NVIDIA A100, Mellanox HDR             | USA         | 277,760    | 27.6          | 1.34       |
| 8  | Texas Advanced Computing Center / Univ. of Texas | Dell         | Frontera<br>Dell C6420,<br>Xeon Platinum 8280 28C 2.7GHz, Mellanox HDR                 | USA         | 448,448    | 23.5          |            |
| 9  | CINECA   | IBM          | Marconi-100<br>IBM Power System AC922,<br>P9 16C 3GHz, Nvidia Volta V100, Mellanox EDR | Italy       | 347,776    | 21.6          | 1.98       |
| 10 | Swiss National Supercomputing Centre (CSCS)      | Cray         | Piz Daint<br>Cray XC50,<br>Xeon E5 12C 2.6GHz, NVIDIA Tesla P100, Aries                | Switzerland | 387,872    | 21.2          | 2.38       |

图：2020 年 6 月 TOP500 排名

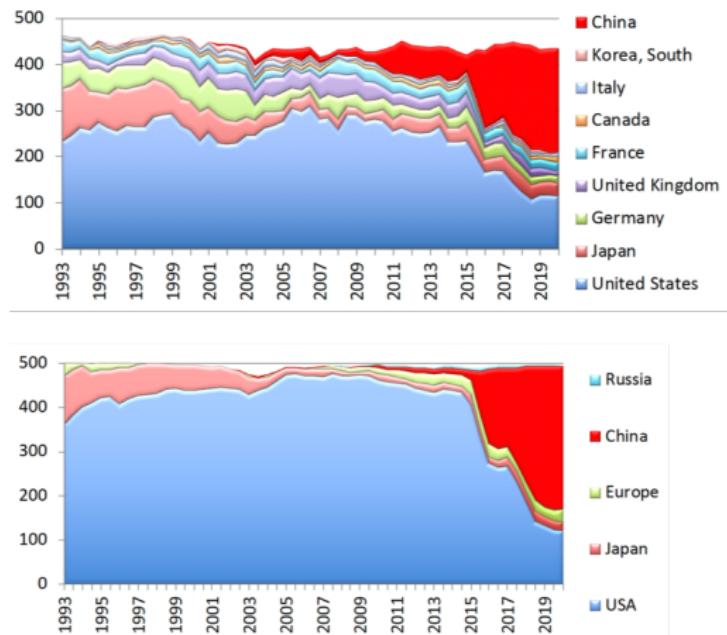
# TOP500 发展趋势：性能

## PERFORMANCE DEVELOPMENT



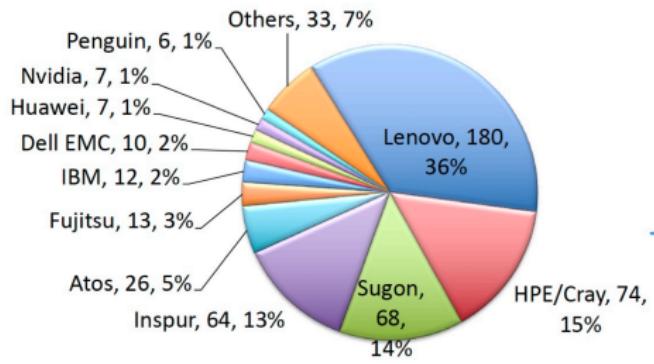
图: TOP500 性能发展趋势

# TOP500 发展趋势：份额——按国家



无论从安装地（上图）还是生产地（下图）来看，中国近年份额大增，已经超过美国跃居世界第一

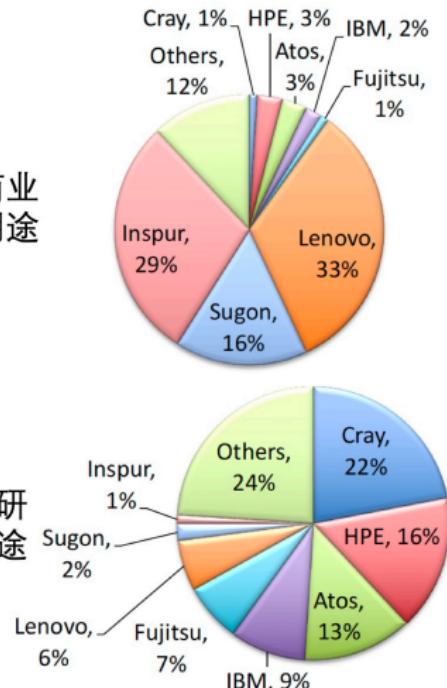
# TOP500 发展趋势：份额——按厂商



中国厂商份额占比较大，但大多集中在商业用途的服务器

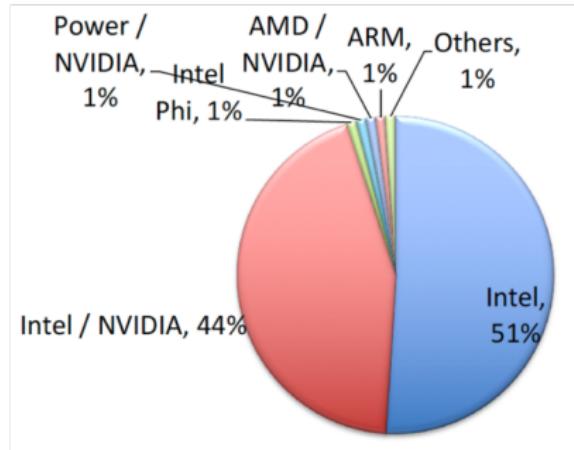
商业用途

科研用途

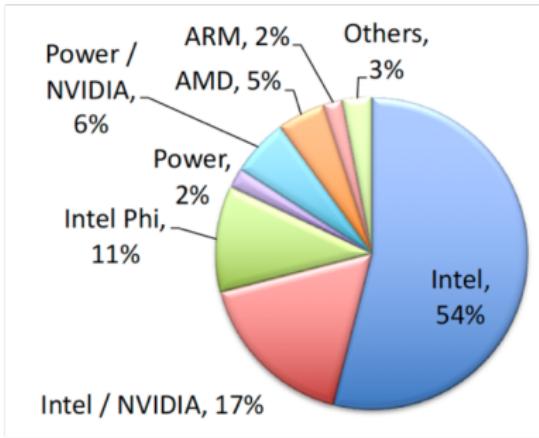


# TOP500 发展趋势：份额——按芯片

商业用途



科研用途



以Intel、NVIDIA为代表的处理器仍是主流，科研领域对其他类型处理器更为包容

# TOP500 排名第一的超级计算机



# 硬件发展趋势

## CPU-MIC异构计算

2个12核CPU  
+  
3块57核MIC



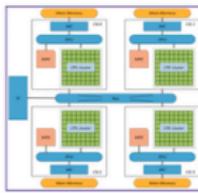
16000x

## 天河2 (312万核)



## 片上异构计算

4个主核  
+  
256个从核



40000x

## 神威太湖之光 (1064万核)



算法?  
软件?

## CPU-GPU异构计算

2个22核CPU  
+  
6块80核GPU

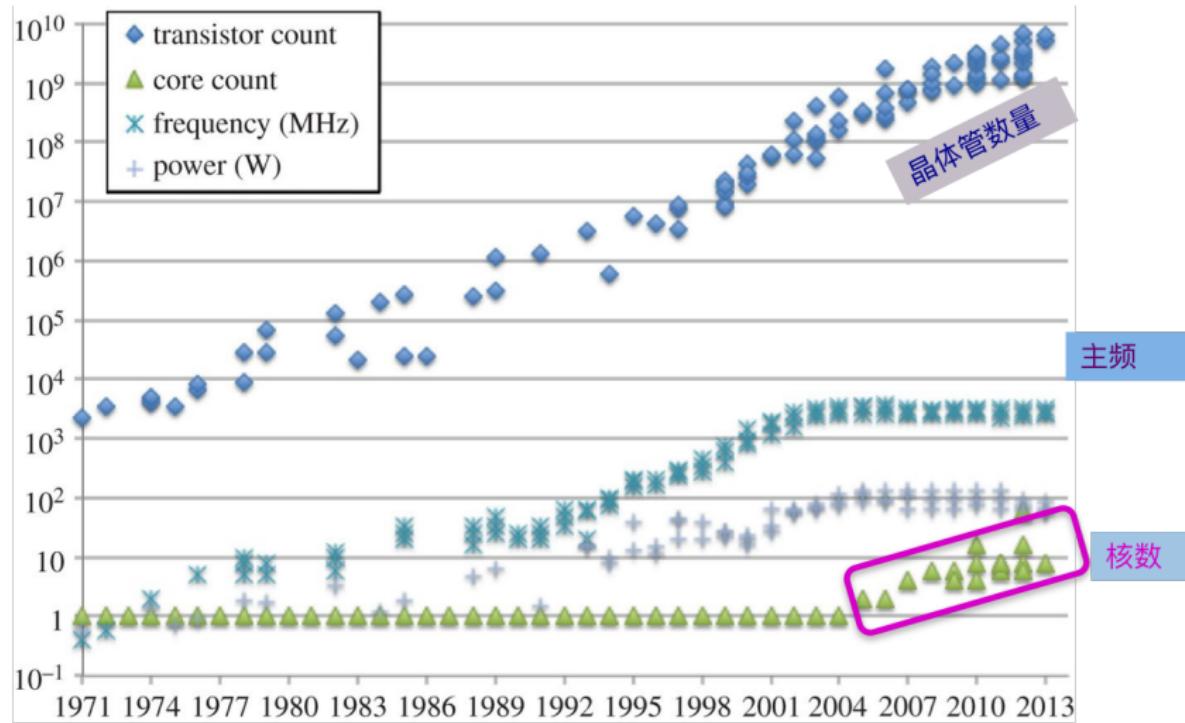


4608x

## 顶点 (241万核)



# 为什么核数越来越多？



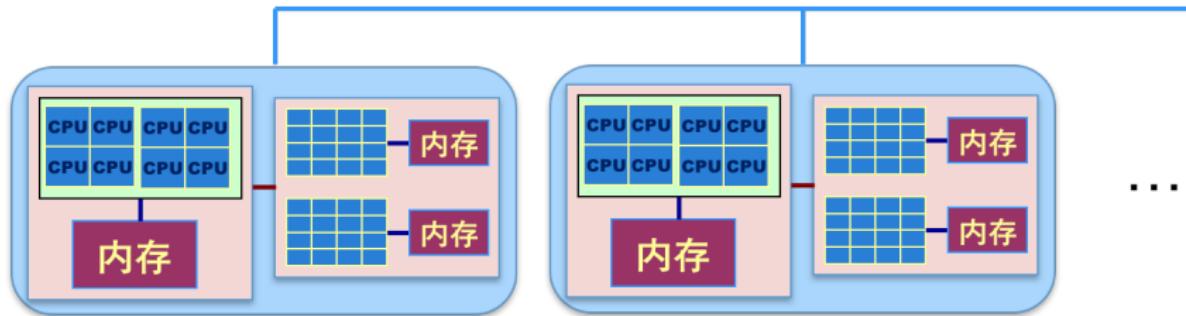
Courtesy: Giles & Reguly, 2014

# 为什么核数越来越多？

- 功耗问题成为了高性能计算机发展趋势变化的一大动因
  - ❖ 公式1：性能  $\propto$  主频  $\times$  核数
  - ❖ 公式2：功耗  $\propto$  性能  $\times$  电压<sup>2</sup>
  - ❖ 公式3：主频  $\propto$  电压
  - ❖ 提高性能方案1：提高主频
    - 为了提高性能1倍，主频增大1倍，但功耗提高7倍！
  - ❖ 提高性能方案2：提高核数
    - 为了提高性能1倍，核数增多1倍，功耗只需提高1倍！
    - 保持性能不变，核数增多1倍，功耗降低75%！
    - 保持功耗不变，核数增多1倍，性能提升58%！
- 结论：提高核数是维持性能提高并降低功耗增涨的有效途径！
  - ❖ 单核 → 双核 → 多核 → 众核 → ...

# 为什么采用异构设计？

- 异构分布式并行机

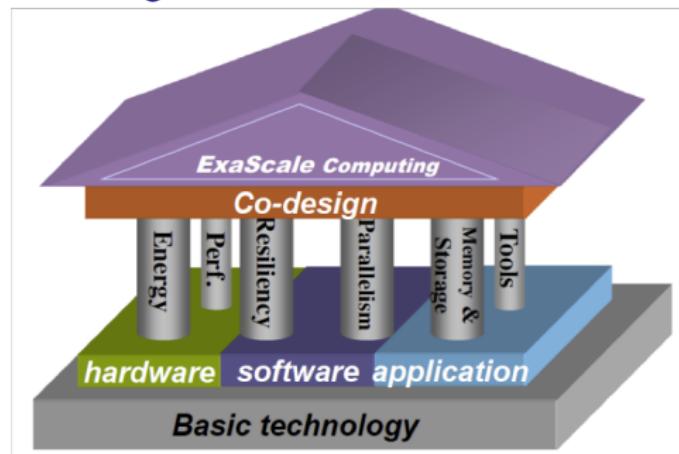


- 异构：意味着处理器之间地位不平等，为什么？

- ❖ 不同类型的处理器具备不同的能力
  - 处理复杂操作能力、计算能力、访存能力、通信能力等
- ❖ 例子：CPU、GPU、MIC ...
- ❖ 对比：军、师、旅、团、营、连、排...

# 并行算法与软件的研究价值

- 算法和软件是应用和计算机之间的桥梁
- 算法是软件的灵魂，不同类型的应用所需的算法可能不同
- 不同的算法各自适用于不同的计算机
  - ❖ 比如：传统的FFT算法在向量机上很好，但在分布式系统上不够理想
- 需要多方面专家协同设计（Co-design）



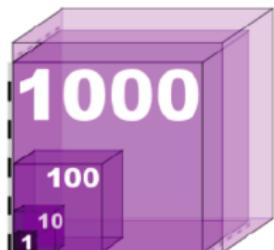
# 并行（与分布式）计算研究内容

- **硬件架构**: 认识当代高性能计算机体系架构特征, 理解并行计算模型和并行性能评价方法, 指导并行算法设计和并行程序实现。
- **并行算法**: 针对应用领域专家求解各类应用问题的计算方法, 设计高效率的并行算法 (将应用问题分解为可并行计算的多个子任务), 并分析算法的可行性和效果。
- **并行编程**: 学习不同类型的高性能编程模型和工具, 例如消息传递平台 MPI 或者共享存储平台 OpenMP, 编程实现相应的并行算法, 在此基础上结合高性能硬件特征和应用问题特性, 优化程序性能。

# 并行计算关心的一些要点



Time to Solution



Scalability

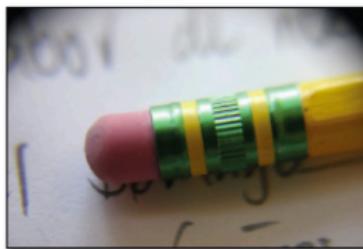


ENERGY STAR

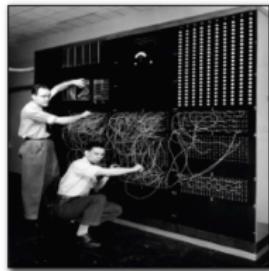
Efficiency



Concurrency &  
Data Locality



Resiliency



Programmability

# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 计算机硬件体系架构

## 指令集架构 (Instruction Set Architecture, ISA)

主要指处理器所支持的机器语言（指令）的种类、格式、长度等，以及内存与寄存器的抽象模型等，例子：x86, alpha, MIPS, RISC-V、ARM ...

## 微架构 (Micro-architecture, μarch)

主要指 ISA 的一种具体的处理器实现，比如处理器核数、缓存大小、流水线长度等，例子：Intel Xeon E5 处理器，...

## 系统架构 (System Architecture)

主要指与处理器不直接相关的其他部分，比如访存、I/O、网络、软件等。

# 2018 年图灵奖 (ACM A. M. Turing Award)

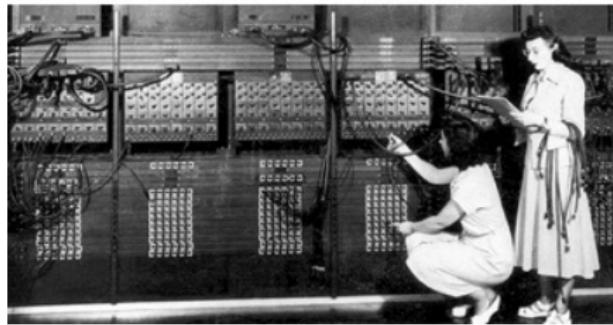
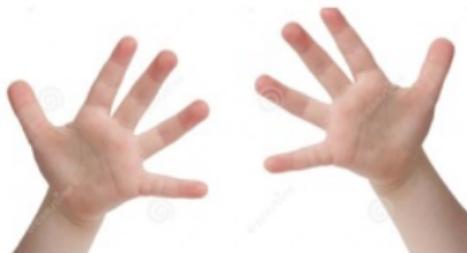


获奖人：John Hennessy (左) 与 David Patterson (右)

- 获奖理由：“开创用于计算机体系架构设计和评估的系统化、定量方法，并对微处理器工业具有持久影响”。
- 评价：“今天，全世界每年生产的 160 亿颗微处理器中 99% 是 RISC 架构，包括手机、平板、数以几十亿计的嵌入式设备”。

# 固定程序计算机 (Fixed-Program Computer)

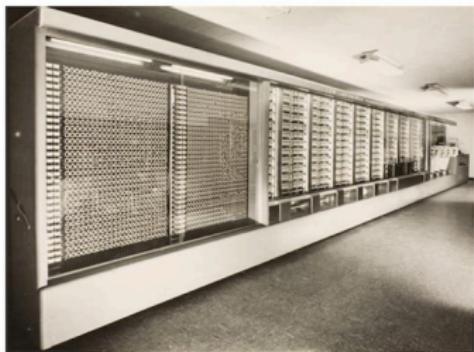
固定程序计算机：无法重新编程，只能解决固定问题，通用性差。



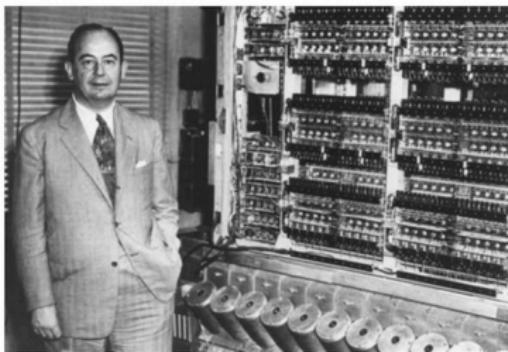
# 存储程序计算机 (Stored-Program Computer)

存储程序计算机：可重新编程，能解决不同问题，更加通用。

- 程序：指令的执行序列集合。
- Harvard 架构：将程序与数据存储在不同的内存中。
- Princeton 架构：将程序与数据共同存储在内存。  
又称冯诺依曼架构，是现代计算机体系架构的基础。



Harvard Mark I (1945)



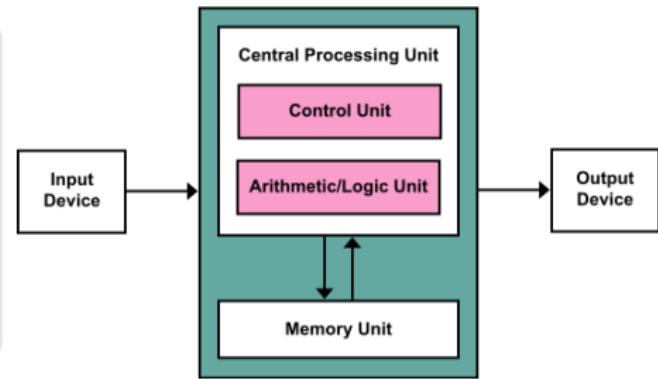
John von Neumann and EDVAC (1949)

# 冯诺依曼架构

主要思想：把指令也当作数据，与数据用同样的方式储存。

## 主要组成部分

- 控制单元：解释指令
- 处理单元：执行指令
- 内存：存储数据和指令
- 输入/输出：与外界交互

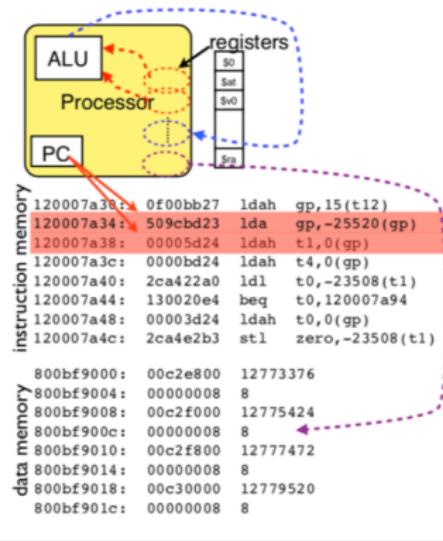


First Draft of a Report on the EDVAC, John von Neumann, 1945.

# 指令是如何执行的？

大致分四个步骤：取指、译码、执行、写回

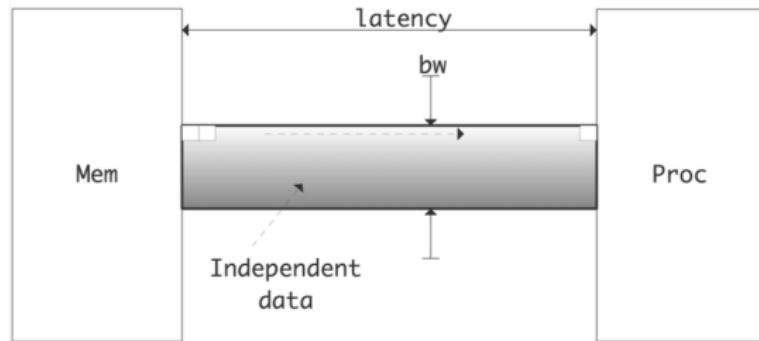
- Instruction fetch: where?  
**instruction memory**
- Decode:
  - What's the instruction? **registers**
  - Where are the operands? **ALUs**
- Execute
- Memory access **data memory**
  - Where is my data?
- Write back **registers**
  - Where to put the result
- Determine the next PC



寄存器 (register) : 是CPU进行运算和保存结果的临时存储空间

# 数据是如何读写的？

前端总线 (Front-Side Bus, FSB): 从内存中将数据移入移出的线路



延迟: 从发送内存请求到实际完成数据移动所需的时间——路长

带宽: 单位时间的数据移动量——路宽

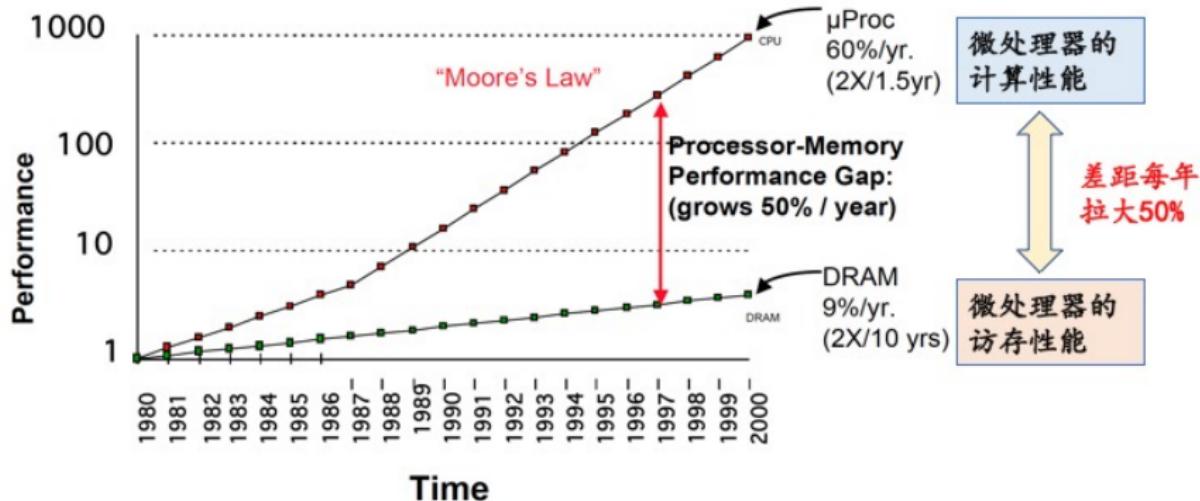
Little's Law:  
 $\text{Concurrency} = \text{Bandwidth} \times \text{Latency}$

说明: 程序需要有足够的并发度, 从而  
隐藏延迟并占满带宽——车要多!

# 冯诺依曼架构的主要缺陷

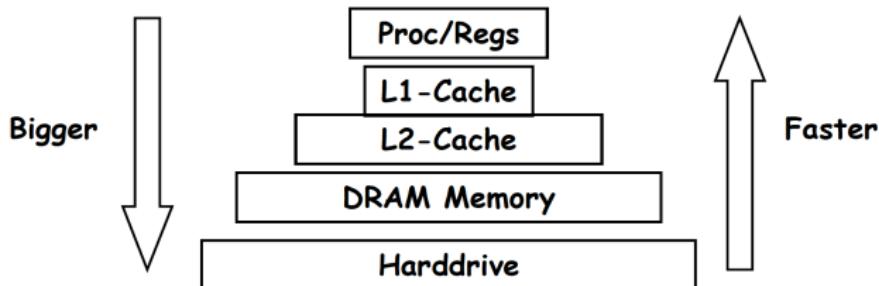
## 访存墙 (Memory Wall)

- 仅具有单一的线性内存，指令与数据仅在使用时才隐式区分
- 总性能往往受限于内存的读写总线所能提供的延迟和带宽



# 多级存储技术

小（快）→ 大（慢）：寄存器 → 各级缓存 → ... → 内存 → 外存



This is an  
AMD Opteron CPU

Total Area: 193 mm<sup>2</sup>

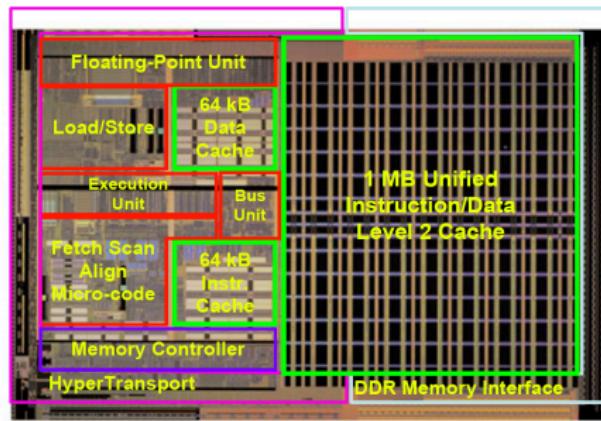
Look at the relative  
sizes of each block:

50% cache

23% I/O

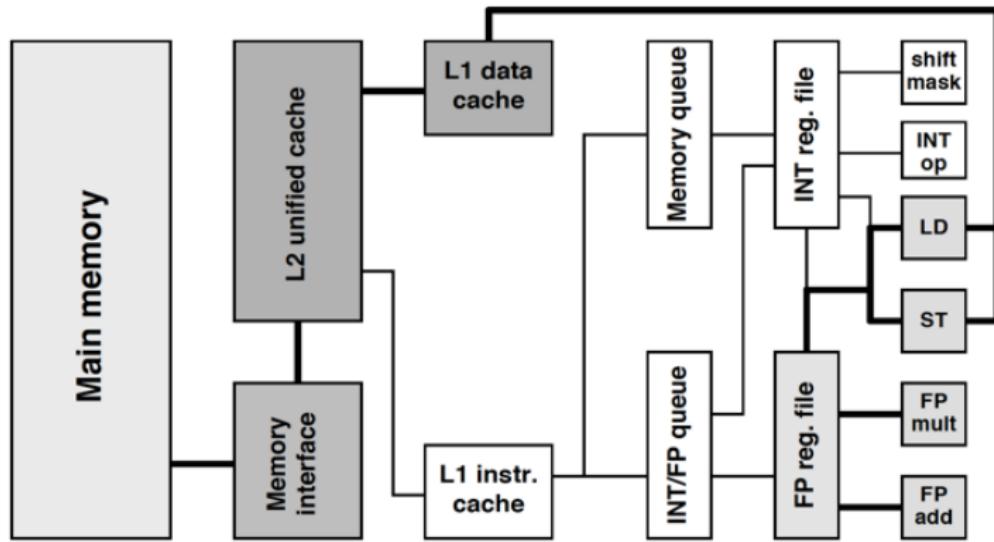
20% CPU logic

+ extra stuff



# 缓存 (Cache) 的设计

一些基本概念：一级/二级缓存、指令/数据缓存、缓存命中/不命中、数据的局部性、预取等。



# 提高处理器性能的其他重要手段

## 简化指令 (Simplified Instruction)

- 复杂指令集计算机 (Complex Instruction Set Computer, CISC);
- 精简指令集计算机 (Reduced Instruction Set Computer, RISC);
- 1980 年代开始, 主流计算机从 CISC 逐渐向 RISC 过渡。

## 指令级并行 (Instruction Level Parallelism, ILP)

- 超标量 (superscalar): 同时译码多个指令;
- 流水线 (pipeline): 多个指令流水执行 (流水线宽度、深度);
- 乱序执行 (out-of-order execution): 设法改变指令执行顺序。

## 数据级并行 (Data Level Parallelism, DLP)

- 向量化 (vectorization): 单指令多数据 (如: 乘加指令)。

# 福林分类 (1)

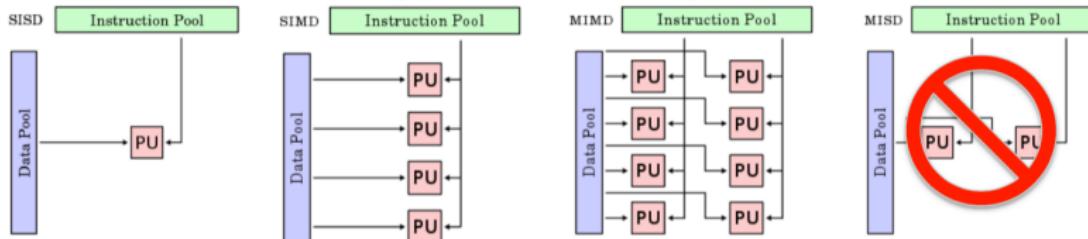
- 并行计算的一种分类方式，由 Michael J. Flynn 于 1966 年提出。
- 从两个正交的维度考虑：指令流（Instruction Stream）和数据流（Data Stream），其中每个维度有 Single 和 Multiple 两种可能选择。

|   |   |
|---|---|
| S I S D   | S I M D   |
| Single Instruction stream<br>Single Data stream   | Single Instruction stream<br>Multiple Data stream   |
| M I S D   | M I M D   |
| Multiple Instruction stream<br>Single Data stream | Multiple Instruction stream<br>Multiple Data stream |

# 福林分类 (2)

|                       | Single instruction stream | Multiple instruction streams | Single program | Multiple programs |
|-----------------------|---------------------------|------------------------------|----------------|-------------------|
| Single data stream    | SISD                      | MISD                         |                |                   |
| Multiple data streams | SIMD                      | MIMD                         | SPMD           | MPMD              |

commonly used  
scientific model

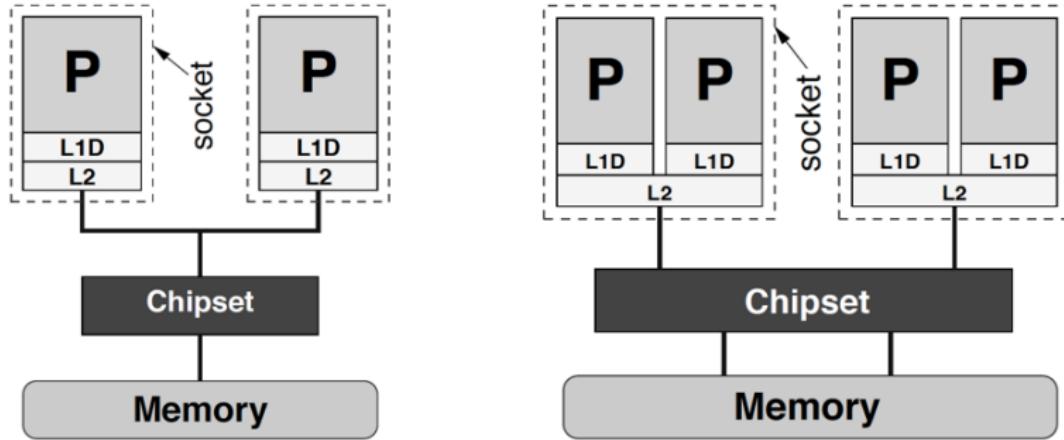


Single Program, Multiple Data (SPMD) is a natural generalization of Single Instruction, Multiple Data (SIMD) when each processing unit executes its own local copy of the instruction stream. These copies can branch differently depending upon the data.

c/o The Wikipedia

# 共享内存并行机 (1)

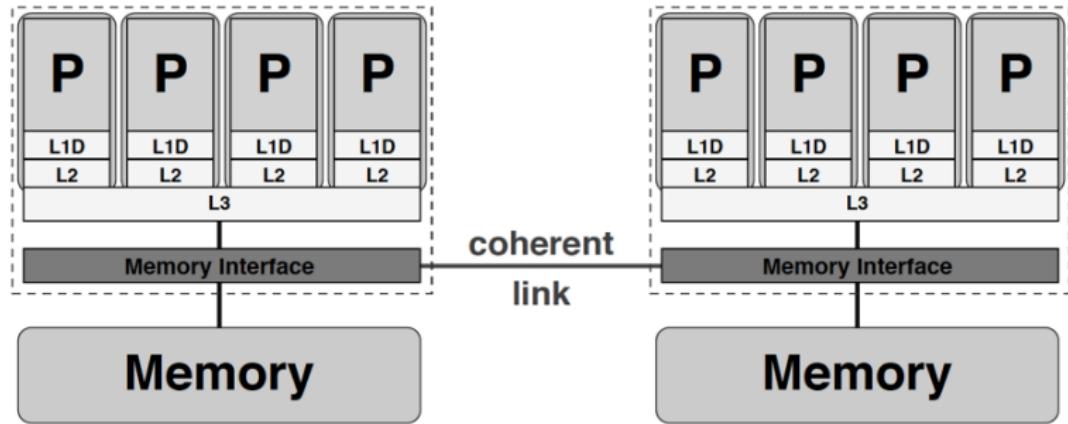
UMA ( Uniform Memory Access) : 一致内存访问架构。



“几路几核”：表示有多少个插槽 (socket)，每个插槽有多少核。  
如：上图分别可以描述为双路单核和双路双核的 UMA 架构。

## 共享内存并行机 (2)

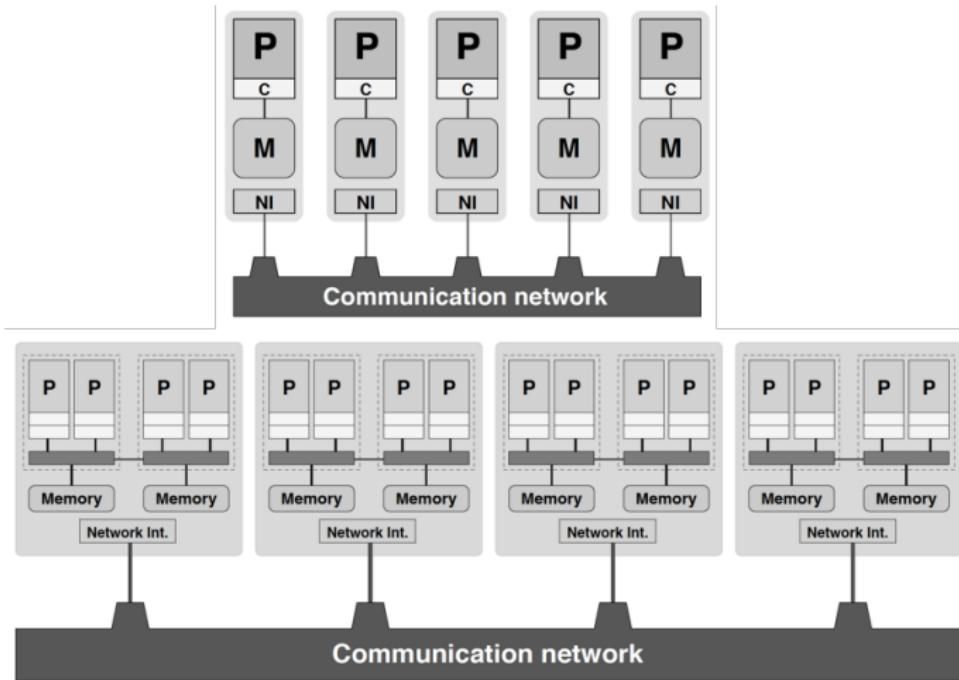
ccNUMA (cache-coherent Nonuniform Memory Access) : 缓存一致性的非一致内存访问架构。



思考：上图表示几路几核的 ccNUMA 架构？

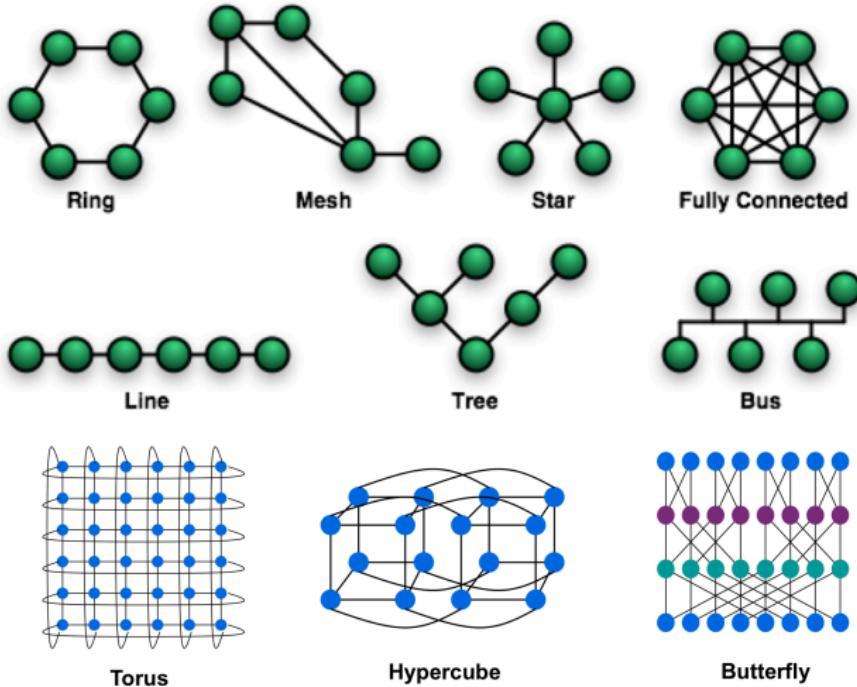
# 分布式并行机

分布式并行机/集群 (cluster): 计算节点 (compute node) 之间通过高速网络互联，计算节点内部可以是任意类型的单核或共享内存架构。



# 互联网络的拓扑架构

一些重要指标如直径 (diameter)、二分宽度 (bisection width) 等会影响网络通信性能。

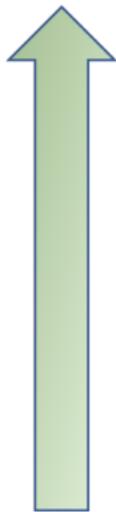


# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 并行计算研究的几个主要视角

应用



- 应用视角：数学建模，算法设计与分析等。
- 算法视角：算法的并行化以及相关的主要原则等。
- 编程视角：采用何种方式编程实现并行算法。
- 性能视角：通过建立并行计算模型，指导算法设计、编程实现、性能优化等。
- 硬件视角：并行计算机体系架构。

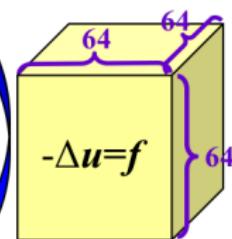
硬件

课程从应用与硬件切入，通过讨论算法与性能，最终聚焦并行编程。

# 算法不同，效果天地之别

一个简单的例子：求解立方体区域 Poisson 方程，未知数个数  $N = n^3$

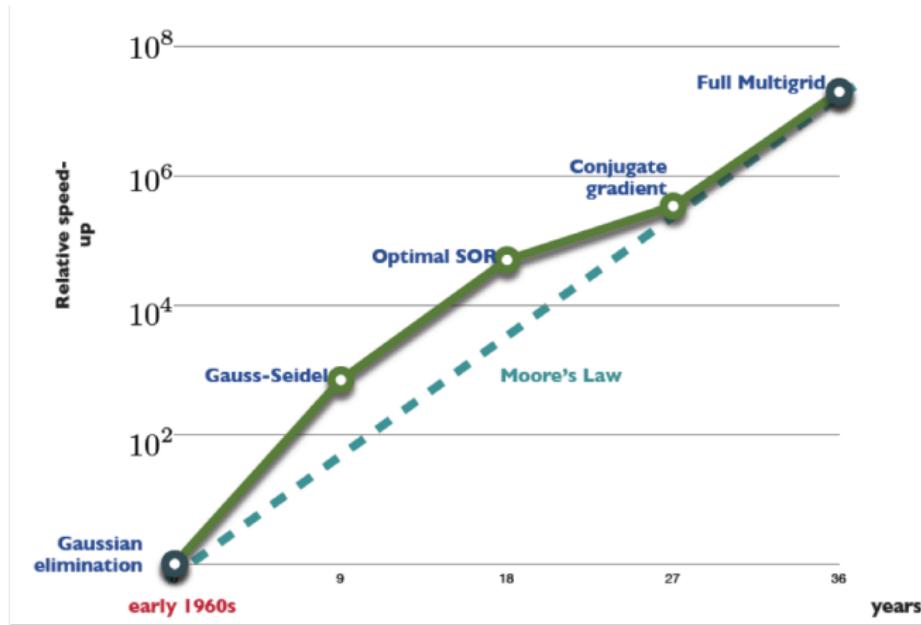
| Year | Method      | Reference               | Storage | Flops            |
|------|-------------|-------------------------|---------|------------------|
| 1947 | GE (banded) | Von Neumann & Goldstine | $n^5$   | $n^7$            |
| 1950 | Optimal SOR | Young                   | $n^3$   | $n^4 \log n$     |
| 1971 | CG          | Reid                    | $n^3$   | $n^{3.5} \log n$ |
| 1984 | Full MG     | Brandt                  | $n^3$   | $n^3$            |



对  $n=64$ ，节省时间 1600 万倍，求解时间从 6 个月缩减为 1 秒！

# 算法的发展规律

算法的贡献也遵循类似摩尔定律的发展规律！

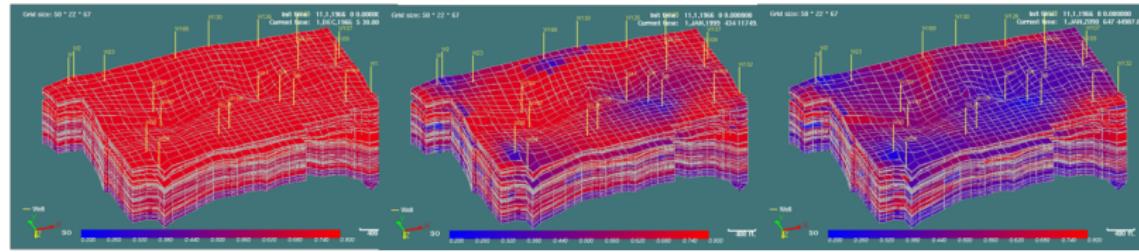


Algorithms can often speed up science as much as the Moore's law.  
——摘自2005年美国总统报告, PITAC.

# 算法举例

## 一个应用的例子：大规模油藏模拟

- 1998→2002, 性能提升1600倍, 其中
  - ❖ 来自机器的: 40倍
    - 单处理器性能提升: 5倍
    - 并行实现优化: 8倍
  - ❖ 来自算法的: 40倍
    - 新的数值计算方法: 8倍
    - 并行算法优化: 5倍

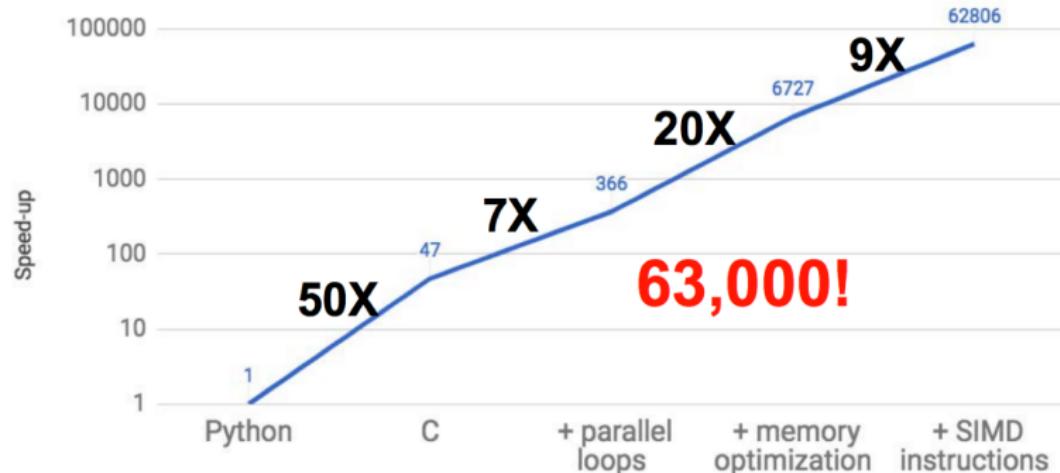


# 编程不同，效果天地之别

一个简单的例子：稠密矩阵乘

$$C = C + A * B, \quad A \in \mathbb{R}^{m,k}, B \in \mathbb{R}^{k,n}, C \in \mathbb{R}^{m,n}.$$

Matrix Multiply Speedup Over Native Python



摘自 John Hennessy 和 David Patterson 图灵奖报告: A New Golden Age for Computer Architecture, 2018.

# 编程举例

如此简单的一个例子，面临一个两难的棘手问题：

~ 10 lines of Fortran/C

```
DO I = 1, N
    DO J = 1, N
        DO K = 1, N
            C(I,J) = C(I,J) + A(I,K) * B(K,J)
        ENDDO
    ENDDO
ENDDO
```

性能： 峰值\*1%

工作量： 5分钟

~ 2000 lines of assembly code

```
VMAD     a0,b0,t00,t00
ADDL     A,16*SIZE(A
LDDE     nb0,4*SIZE(B)

VMAD     a0,b1,t04,t04
LDDE     nb1,5*SIZE(B)

VMAD     a4,b0,t01,t01
VLD      na12,12*SIZE(A)

VMAD     a4,b1,t05,t05
VLD      na8,8*SIZE(A)

VMAD     a0,b2,t08,t08
LDDE     nb2,6*SIZE(B)

VMAD     a0,b3,t12,t12
LDDE     nb3,7*SIZE(B)
```

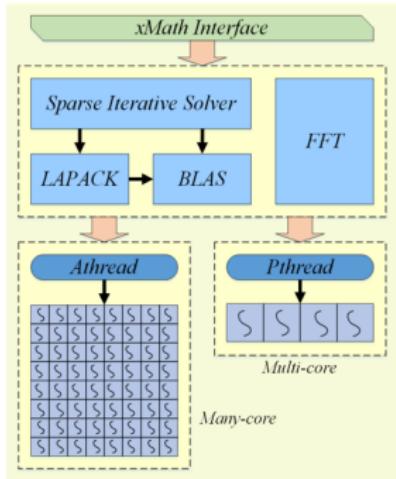
性能： 峰值\*95%

工作量：2-3个月



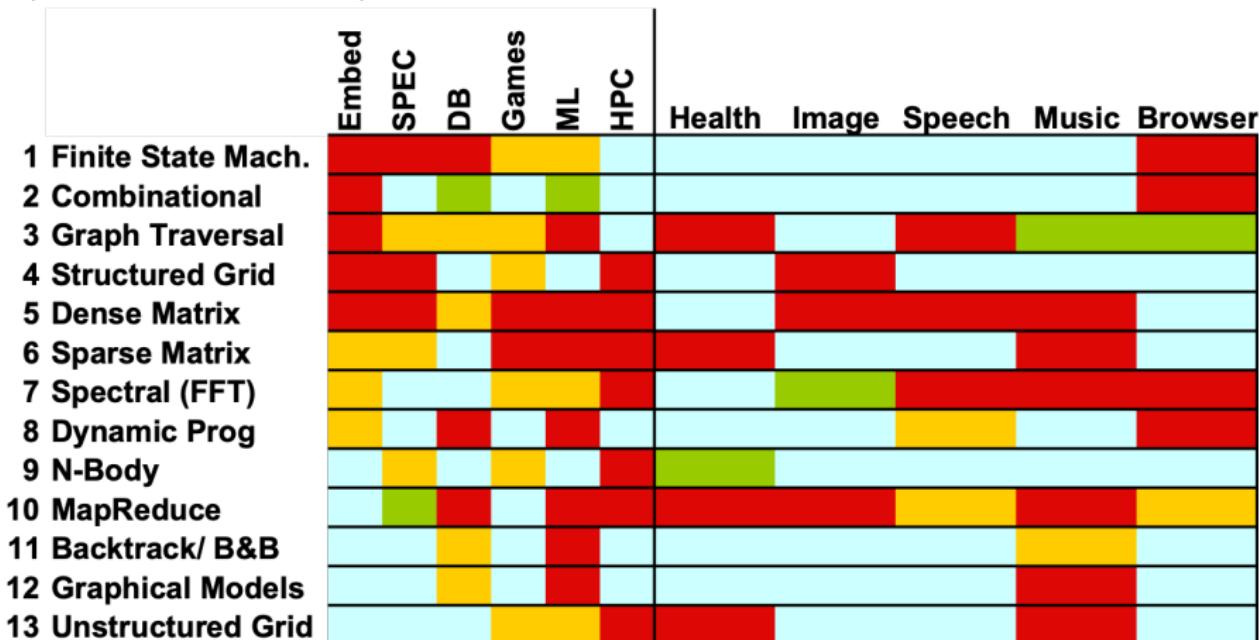
# 解决手段：模型驱动的高性能代码生成器（硬件建模 + 算法建模）

- 可在极短时间生成接近人类深度优化的高性能代码！
- 已应用于神威太湖之光的 xMath 高性能库（100 万行代码）。



# 评价应用特征的十三个“小矮人”

Motif/Dwarf: Common Computational Patterns  
(Red Hot → Blue Cool)



The Landscape of Parallel Computing Research: A View from Berkeley 2.0

# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 一些约定/假定

- 任务 (task): 并行计算所处理的对象。
- 工作量 (workload): 处理某任务的所需的各种开销的总和。
- 处理器 (processor): 并行计算中所使用的最基本的处理器单元。
- 执行率 (execution rate): 每个处理器单位时间能完成的工作量<sup>1</sup>。
- 执行时间 (execution time): 处理某任务所需的时间。
- 加速比 (scalability): 当处理器个数增多时，完成某固定工作量任务所需执行时间的减少倍数。
- 理想加速比 (ideal scalability): 处理器个数增多的比例。
- 并行效率 (parallel efficiency): 加速比  $\div$  理想加速比  $\times 100\%$ .

注 1: 这里假定每个处理器的执行率相同。

# 阿姆达尔定律

## 阿姆达尔定律 (Amdahl's Law, 1967)

记  $\alpha \in [0, 1]$  是某任务无法并行处理部分所占的比例. 假设该任务的工作量固定, 则对任意  $n$  个处理器, 相比于 1 个处理器, 能够取得的加速比满足:  $S(n) < 1/\alpha$ .

**证明:** 设每个处理器的执行率是  $R$ , 处理该任务的总工作量是  $W$ . 记  $T(1)$  和  $T(n)$  分别为使用 1 个和  $n$  个处理器处理该任务所需要的时间, 则有

$$T(1) = \frac{W}{R}, \quad T(n) = \frac{\alpha W}{R} + \frac{(1 - \alpha)W}{nR}.$$

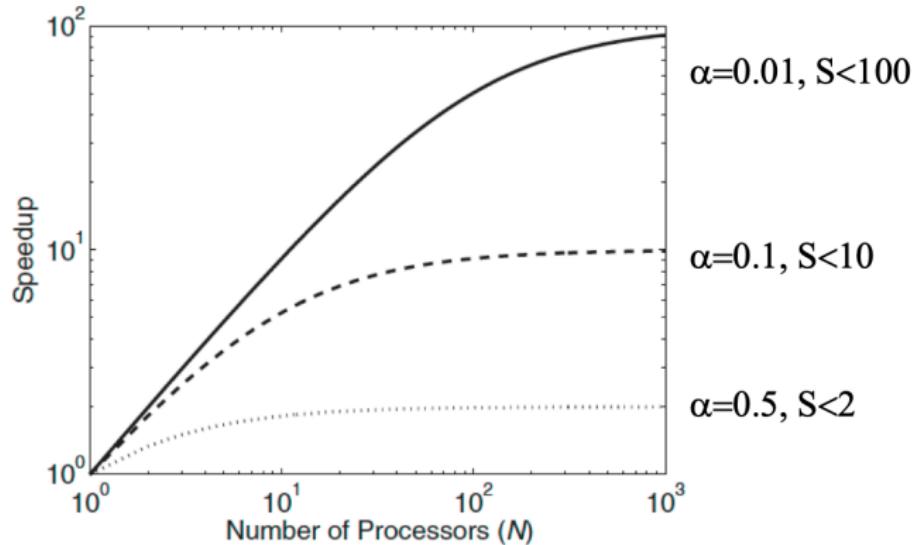
据此, 可计算出使用  $n$  个处理器相比于 1 个处理器的加速比

$$S(n) = \frac{T(1)}{T(n)} = \frac{1}{\alpha + \frac{1-\alpha}{n}} < \frac{1}{\alpha}.$$

# 阿姆达尔定律的推论



Gene M. Amdahl  
(1922-2015)



484 Spring Joint Computer Conf., 1967

by DR. GENE M. AMDAHL  
International Business Machines Corporation  
Sunnyvale, California

Validity of the single processor approach to achieving large scale computing capabilities

# 卡普挑战



NCUBE 10 (1,024 processors)

## Background:

In 1983, there was a new tech trend of **massively parallel computing** with over 1,000 processors.

People started to wonder how to use them efficiently.



Alan H. Karp

## 卡普挑战 (Karp Challenge, 1985)

打赌 100 美元没人能在十年内为三个实际应用实现 200 倍的并行加速。

- 结果：两年之内，没有胜者（大多是模型问题，加速比最多数十）。

# 戈登贝尔奖



Gordon Bell  
(1934-)

In 1987, Alan Karp was approached by Gordon Bell, who was a founding assistant director of the **CISE Directorate at NSF**.

Bell persuaded Karp to replace the **Karp Challenge** with the **Gordon Bell Prize**.

The Gordon Bell Prize **does not require a speedup of 200X** or any particular number but emphasizes on **technical innovations in HPC applications**.

## 戈登贝尔奖 (Gordon Bell Prize, 1987)

每年奖励 1000 美元给在高性能计算应用取得杰出成就的团队。

- 2006 年起由 ACM 负责, 2011 年起奖金提升至 1 万美元。

# 首届戈登贝尔奖



John L. Gustafson  
(1955-)

In 1987 the first Gordon Bell Prize was awarded to **Robert Benner, John Gustafson and Gary Montry** from Sandia National Laboratories.

Achieved **400~600** speedup on NCUBE 10 in 3 applications:

- \* Beam Stress Analysis
- \* Surface Wave Simulation
- \* Unstable Fluid Flow Modeling

In fact, they also won the Karp Challenge!

## REEVALUATING AMDAHL'S LAW

JOHN L. GUSTAFSON

# 古斯塔法森定律

## 古斯塔法森定律 (Gustafson's Law, 1988)

记  $\alpha \in [0, 1]$  是某任务无法并行处理部分所占的比例。假设该任务的工作量可以随着处理器个数缩放，从而保持处理时间固定。则对任意  $n$  个处理器，相比于 1 个处理器，能够取得的加速比  $S'(n)$  不存在上界。

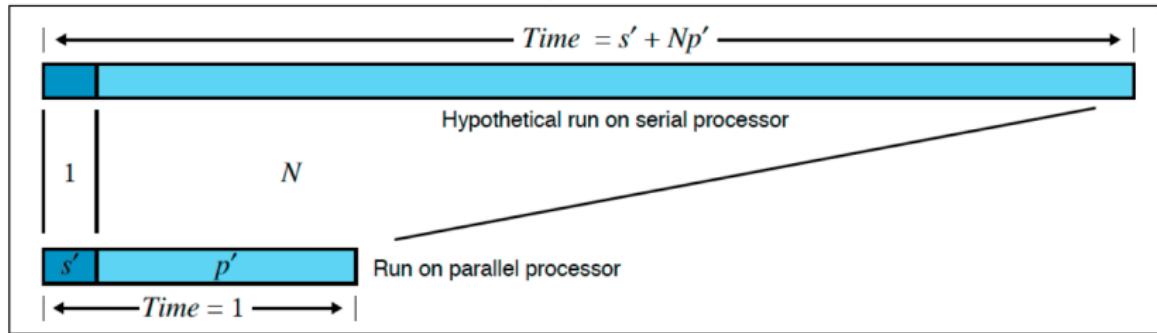
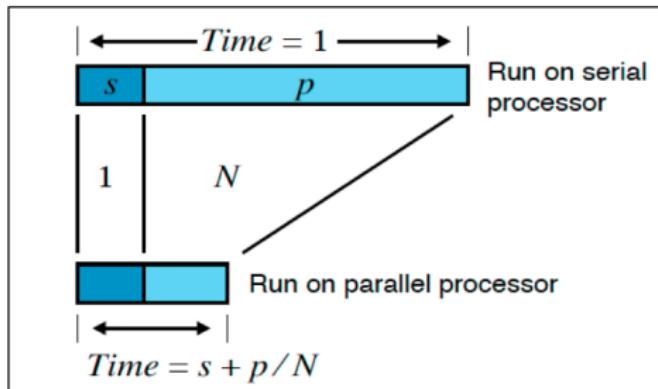
**证明：** 设每个处理器的执行率是  $R$ ，单处理器情况下该任务的基准工作量是  $W$ 。在处理时间固定的情况下，可以得知采用  $n$  个处理器时该任务的缩放工作量  $W'$  应为

$$W' = \alpha W + (1 - \alpha)nW.$$

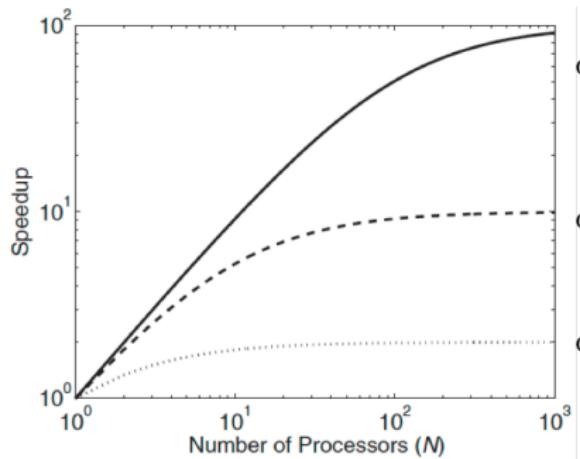
据此，可计算出使用  $n$  个处理器相比于 1 个处理器的加速比

$$S'(n) = \frac{W'}{\frac{W}{R}} = \alpha + (1 - \alpha)n.$$

# 从阿姆达尔加速比到古斯塔法森加速比



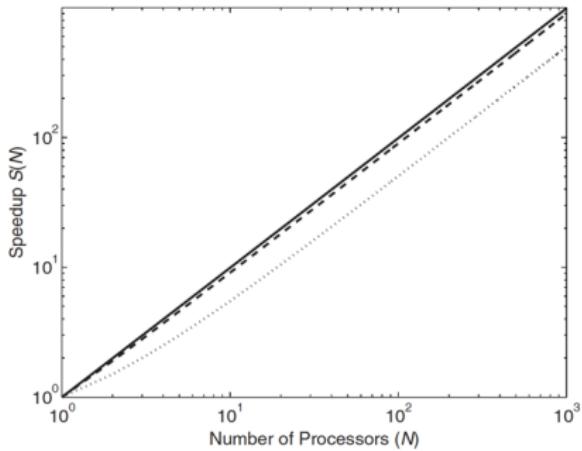
# 从阿姆达尔加速比到古斯塔法森加速比



$\alpha=0.01$

$\alpha=0.1$

$\alpha=0.5$



# 孙-倪定律

## 孙-倪定律 (Sun-Ni's Law, 1990)

记  $\alpha \in [0, 1]$  是某任务无法并行处理部分所占的比例。假设该任务的可并行部分随着处理器个数  $n$  按照因子  $G(n)$  缩放，则对任意  $n$ ，相比于 1 个处理器，能够取得的加速比  $S^*(n)$  满足

$$S^*(n) = \frac{\alpha + (1 - \alpha)G(n)}{\alpha + (1 - \alpha)\frac{G(n)}{n}}.$$

**证明：**设每个处理器的执行率是  $R$ ，单处理器情况下该任务的基准工作量是  $W$ 。使用  $n$  个处理器时该任务的缩放工作量为

$$W^* = \alpha W + (1 - \alpha)G(n)W.$$

据此，可计算出使用  $n$  个处理器相比于 1 个处理器的加速比

$$S^*(n) = \frac{\alpha W \frac{1}{R} + (1 - \alpha)G(n)W \frac{1}{R}}{\alpha W \frac{1}{R} + (1 - \alpha)G(n)W \frac{1}{nR}} = \frac{\alpha + (1 - \alpha)G(n)}{\alpha + (1 - \alpha)\frac{G(n)}{n}}.$$

# 三种模型的关系

## 加速比分析

$$S^*(n) = \frac{\alpha + (1 - \alpha)G(n)}{\alpha + (1 - \alpha)\frac{G(n)}{n}} \begin{cases} = \frac{1}{\alpha + \frac{1-\alpha}{n}} & \text{if } G(n) = 1, \\ = \alpha + (1 - \alpha)n & \text{if } G(n) = n, \\ > \alpha + (1 - \alpha)n & \text{if } G(n) > n. \end{cases}$$

|                       | 加速比 ( $n \rightarrow \infty$ )      | 并行效率 ( $n \rightarrow \infty$ ) |
|-----------------------|-------------------------------------|---------------------------------|
| 阿姆达尔                  | $S(n) \rightarrow \frac{1}{\alpha}$ | $E(n) \rightarrow 0$            |
| 古斯塔法森                 | $S'(n) \rightarrow \infty$          | $E'(n) \rightarrow 1 - \alpha$  |
| 孙-倪 ( $G(n) > O(n)$ ) | $S^*(n) \rightarrow \infty$         | $E^*(n) \rightarrow 1$          |

# 应用举例

例：矩阵乘  $C = AB$ ，这里  $A, B, C$  都是  $N \times N$  阶矩阵。

- 计算复杂度： $\mathcal{C}(N) = 2N^3$ .
- 存储复杂度： $\mathcal{S}(N) = 3N^2$ .

由  $\mathcal{S}(N) = x$  可得  $\mathcal{S}^{-1}(x) = \left(\frac{x}{3}\right)^{\frac{1}{2}}$ ，因此

$$G(n) = \frac{\mathcal{C}(\mathcal{S}^{-1}(nx))}{\mathcal{C}(\mathcal{S}^{-1}(x))} = \frac{2\left(\frac{nx}{3}\right)^{\frac{1}{2} \cdot 3}}{2\left(\frac{x}{3}\right)^{\frac{1}{2} \cdot 3}} = n^{\frac{3}{2}},$$

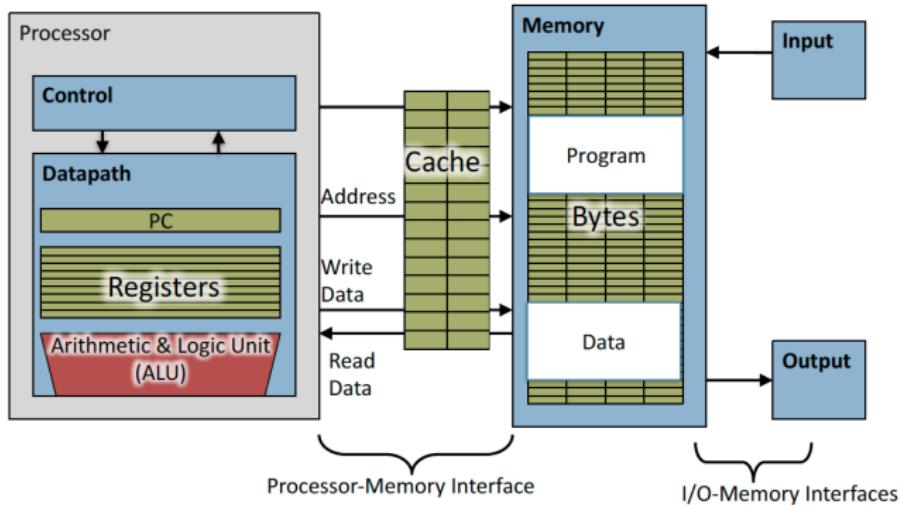
从而

$$S^*(n) = \frac{\alpha + (1 - \alpha)G(n)}{\alpha + (1 - \alpha)\frac{G(n)}{n}} = \frac{\alpha + (1 - \alpha)n^{\frac{3}{2}}}{\alpha + (1 - \alpha)n^{\frac{1}{2}}}.$$

# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 影响性能的主要因素是什么？



## 主要因素分析

- 冯诺伊曼体系架构 → 计算、访存之间的关系。
- 多级存储 → 多级数据访问。
- 多核/众核 → 并行计算/访存。

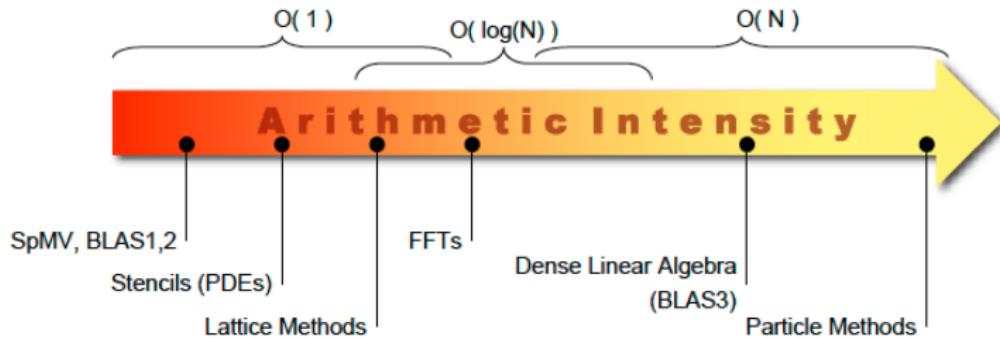
# 从计算、访存之间的关系考虑

## 基本思想

$$\frac{flop}{second} = \min\{\text{peak\_flops}, \frac{byte}{second} * \frac{flop}{byte}\}$$

性能              峰值              带宽      计算密度

因此，引入计算密度 (Arithmetic Intensity) :



# Roofline 模型 (2009)

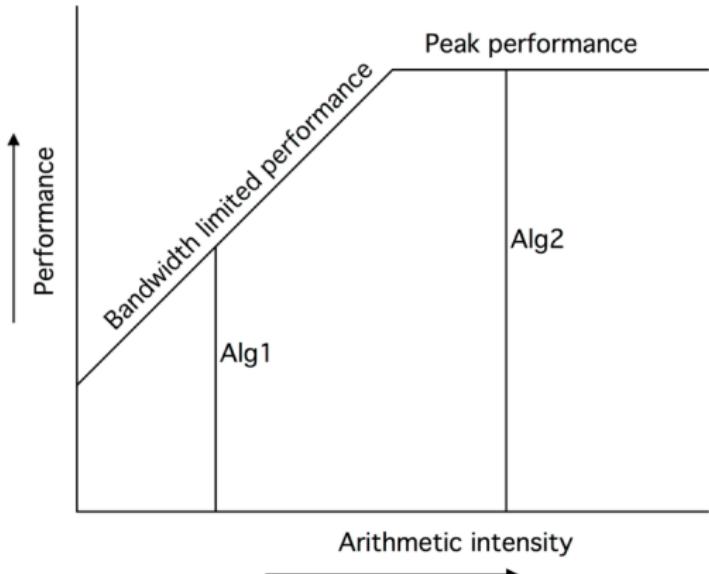
DOI:10.1145/1498765.1498785

The Roofline model offers insight on how to improve the performance of software and hardware.

BY SAMUEL WILLIAMS, ANDREW WATERMAN, AND DAVID PATTERSON

# Roofline: An Insightful Visual Performance Model for Multicore Architectures

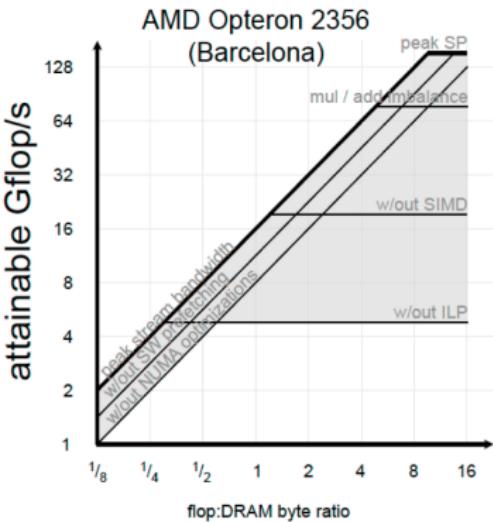
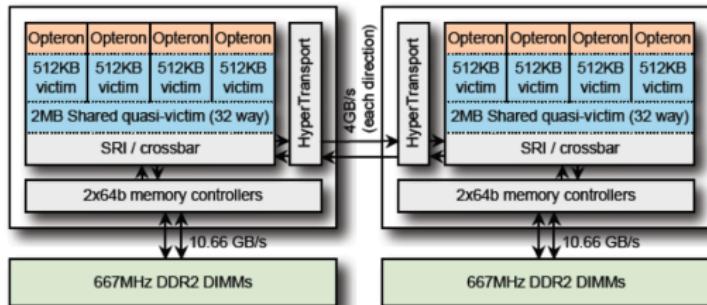
APRIL 2009 | VOL. 52 | NO. 4 | COMMUNICATIONS OF THE ACM 65



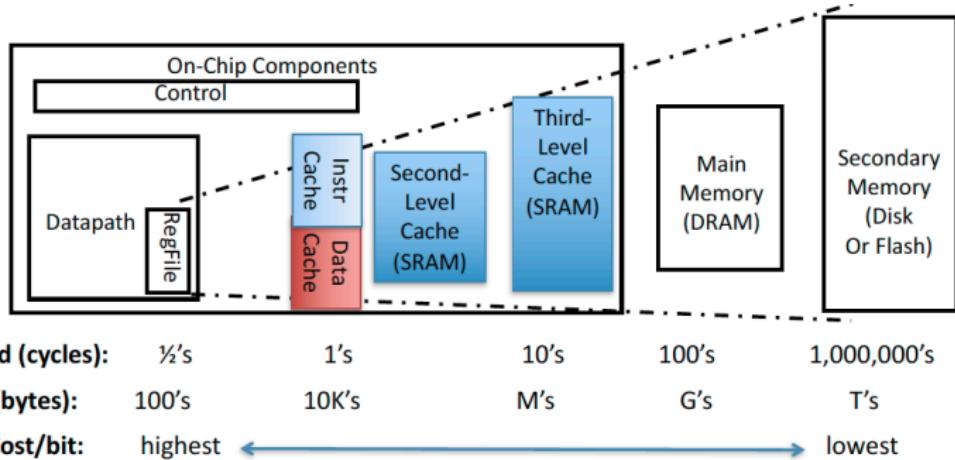
# 受约束的 Roofline 模型

## 举例

### AMD Opteron 2356 (Barcelona)



# 从多级存储的角度考虑



重点考虑缓存命中/失效 (hit/miss) 对时间的影响：

- 命中：数据在该级存储中找到；
- 不命中：需要在下一级存储中寻找。
- 需要引入命中率/失效率 (hit/miss rate) 的概念。

# 单层 AMAT 模型

## 单层 AMAT (Average Memory Access Time) 模型

假设只有一级缓存，AMAT 模型预测的平均访存时间为：

$$\begin{aligned} \text{AMAT} &= (1 - r)T_{\$} + r(T_{\$} + T_M) \\ &= T_{\$} + rT_M, \end{aligned}$$

其中  $T_{\$}$  为缓存访问时间（也叫命中时间, hit time）， $T_M$  为内存访问时间（也叫缓存失效损失, miss penalty）， $r$  为缓存失效率 (miss rate)。

假如某平台 CPU 主频为 1GHz，即 CPU 时钟周期是 1ns，且

- 缓存访问开销为 2 拍 (cycle)，即  $T_{\$} = 2\text{ns}$ ；
- 缓存失效损失为 300 拍 (cycle)，即  $T_m = 300\text{ns}$ ；
- 缓存命中率为 90%，即  $r = 0.1$ ；则

$$\text{AMAT} = 2\text{ns} + 0.1 * 300\text{ns} = 32\text{ns}.$$

# 多层 AMAT 模型

## 多层 AMAT (Average Memory Access Time) 模型

假设有两级/三级缓存， $T_1, T_2, T_3$  分别为 L1, L2, L3 缓存访问时间， $T_M$  为内存访问时间， $r_1, r_2, r_3$  分别为 L1, L2, L3 缓存的局部失效率 (local miss rate)，则两层/三层 AMAT 模型预测的平均访存时间为：

$$\begin{aligned}\text{AMAT}_2 &= T_1 + r_1(T_2 + r_2 T_M) \\ &= T_1 + R_1 T_2 + R_2 T_M,\end{aligned}$$

$$\begin{aligned}\text{AMAT}_3 &= T_1 + r_1 [T_2 + r_2(T_3 + r_3 T_M)] \\ &= T_1 + R_1 T_2 + R_2 T_3 + R_3 T_M.\end{aligned}$$

其中  $R_1 = r_1$ ,  $R_2 = r_1 r_2$ ,  $R_3 = r_1 r_2 r_3$  分别为 L1, L2, L3 缓存的整体失效率 (global miss rate)。

注：局部失效率指该层次缓存失效的概率，整体失效率表示该层次缓存以及其上层所有缓存同时失效的概率。

# AMAT 模型应用举例

某平台的多级缓存信息：

|           | L1 缓存       | L2 缓存        | L3 缓存        | 内存          |
|-----------|-------------|--------------|--------------|-------------|
| 访问时间 (ns) | $T_1 = 2$   | $T_2 = 10$   | $T_3 = 50$   | $T_M = 400$ |
| 局部失效率     | $r_1 = 0.1$ | $r_2 = 0.5$  | $r_3 = 0.4$  | 0           |
| 整体失效率     | $R_1 = 0.1$ | $R_2 = 0.05$ | $R_3 = 0.02$ | 0           |

如果只有 L1 缓存：

$$AMAT_1 = 2\text{ns} + 0.1 * 400\text{ns} = 42\text{ns}.$$

如果增加 L2 缓存：

$$AMAT_2 = 2\text{ns} + 0.1 * 10\text{ns} + 0.05 * 400\text{ns} = 23\text{ns}.$$

如果再增加 L3 缓存：

$$AMAT_3 = 2\text{ns} + 0.1 * 10\text{ns} + 0.05 * 50\text{ns} + 0.02 * 400\text{ns} = 13.5\text{ns}.$$

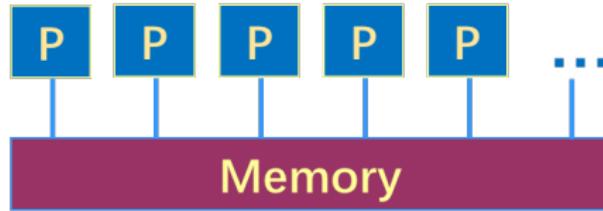
# 从多 (众) 核并行角度考虑

PRAM 模型 (Parallel Random Access Machine Model, 1978)

是 RAM (Random Access Machine) 模型在共享内存系统上的扩展。

- 所有处理器共享一个连续的内存空间；
- 每个处理器执行相互独立的指令；
- 处理器执行任意一种计算或访存操作的时间开销都相等。

模型参数：处理器个数  $p$ , 单位执行时间  $\tau$ .



# PRAM 模型的并行访存策略

基于不同的处理访存冲突的策略，有四类 PRAM 模型

- Exclusive-read, exclusive-write (EREW) 模型；
- Concurrent-read, exclusive-write (CREW) 模型；
- Exclusive-read, concurrent-write (ERCW) 模型；
- Concurrent-read, concurrent-write (CRCW) 模型。

其中，concurrent-write 的处理又分为

- Common：所有处理器写的数值完全相同，没有冲突；
- Arbitrary：任意一个处理器完成写操作，其他处理器不操作；
- Priority：按照某种实现约定的原则确定处理器的优先级，优先级高的处理器写；
- Reduction：规约操作，如 SUM, MAX 等。

# PRAM 模型的特点分析

- 正如串行算法设计者使用 RAM 来研究串行算法性能（比如，串行时间复杂度）一样，并行算法设计者使用 PRAM 来建模和量化分析并行算法性能（比如，在给定个数处理器上的时间复杂度）。
- 采用 PRAM 模型对并行算法进行理论分析的课程：  
<http://pages.cs.wisc.edu/~tvrdik/cs838.html>
- PRAM 模型可以用于共享内存并行计算机，也可以用于分布式并行计算机。
- 然而，PRAM 忽略计算机体系架构的诸多重要特性，比如访存、通信与计算开销的巨大差别，比如缓存、同步等机制，仅使用两个参数（单位时间  $\tau$ 、处理器个数  $p$ ）来估计算法成本，往往难以预测真实性能。
- 更精细的模型：PHM (Parallel Hierarchical Memory) 模型等。

# 内容提纲

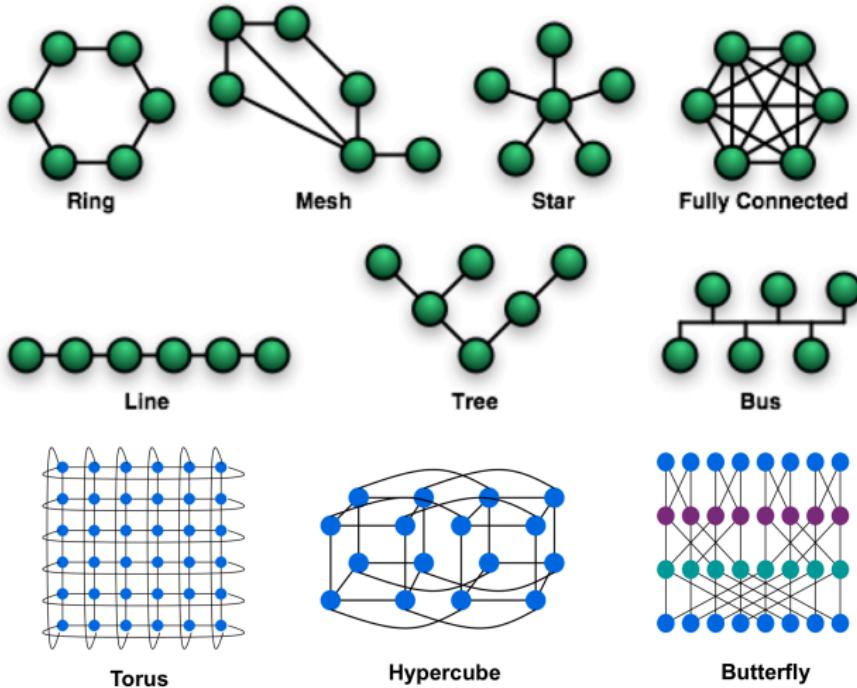
- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 网络系统相关的一些基本概念

- 计算节点 (compute node): 使用高速互联网络连接的计算机，每个计算节点大多是共享内存架构。
- 网络连线 (link): 计算节点之间的高速互联网络，有连线表示存在直接互联。
- 路由 (router): 决定网络通信策略的算法或设备。
- 延迟 (latency): 从一个计算节点到另一个计算节点的数据传输时间。
- 带宽 (bandwidth): 单位时间的网络传输量，单位 GB/s。

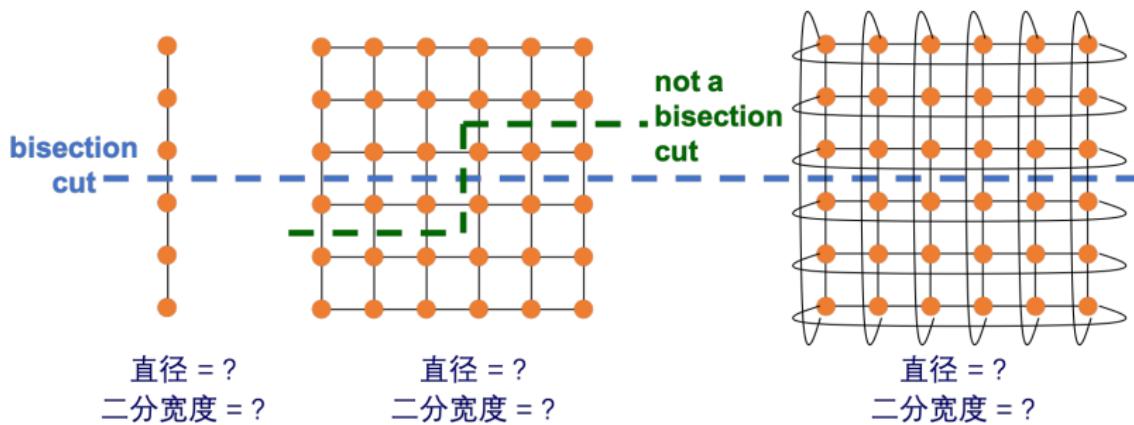
# 互联网络的拓扑架构

线/总线、环/环面、网、星、树、全连接、超立方、蝴蝶、蜻蜓等。



# 如何衡量不同网络拓扑的性能？

- 跳 (hop): 拓扑网络上一点到另一点的最短距离。
- 网络直径 (diameter): 拓扑网络上任意两个节点间的最大跳数。
- 二分宽度 (bisection width): 将拓扑网络平分为二的最小切割数。



思考：一个最优化问题

给定节点数  $n$ , 选取合适的总连线数, 最小化直径、最大化二分宽度!

## $\alpha$ - $\beta$ 模型

网络通信时间由延迟  $\alpha$ , 带宽  $1/\beta$ , 和消息长度  $L$  决定 (忽略拓扑架构):

$$T_{\text{comm}} = \alpha + \beta L.$$

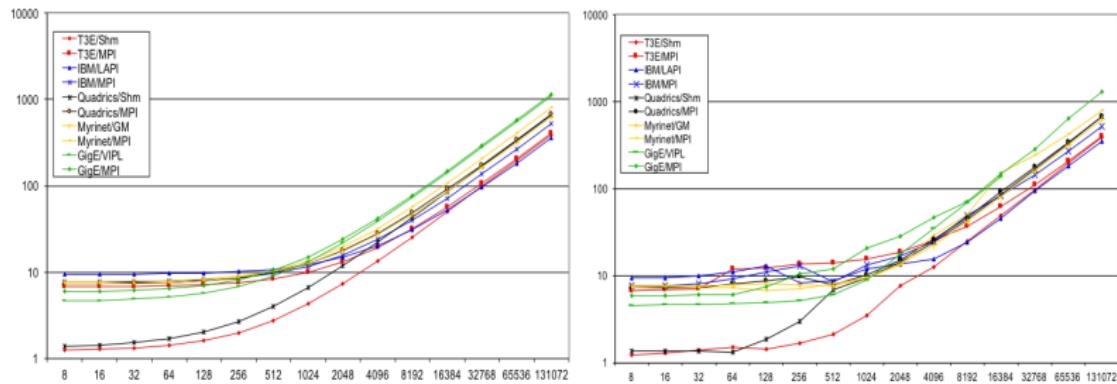
| machine      | $\alpha$ | $\beta$ |
|--------------|----------|---------|
| T3E/Shm      | 1.2      | 0.003   |
| T3E/MPI      | 6.7      | 0.003   |
| IBM/LAPI     | 9.4      | 0.003   |
| IBM/MPI      | 7.6      | 0.004   |
| Quadrics/Get | 3.267    | 0.00498 |
| Quadrics/Shm | 1.3      | 0.005   |
| Quadrics/MPI | 7.3      | 0.005   |

$\alpha$ : latency (us)  
 $\beta$ : 1/BW (us/Byte)

- 推论: 延迟、带宽分别是影响短消息和长消息通信性能的主要因素, 多个短消息不如一个长消息, 因为  $n(\alpha + \beta L) >> \alpha + \beta nL$ 。

# $\alpha$ - $\beta$ 模型的准确度

- 左：预测时间，右：实测时间。

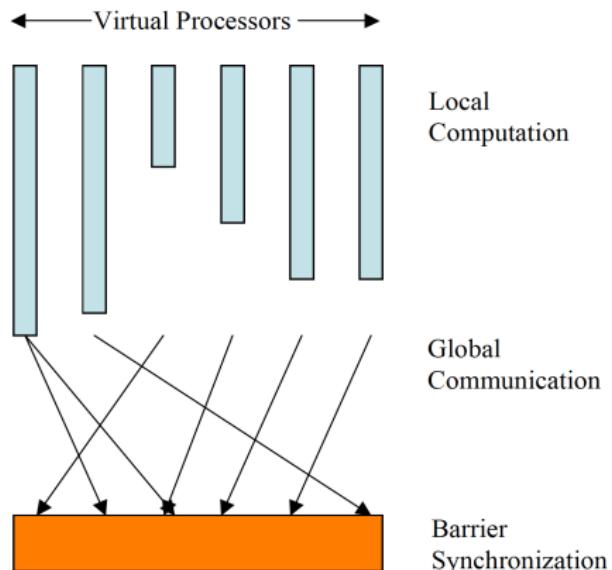


# BSP (Bulk Synchronous Parallel) 模型

## 基本假设/约定

- 每个处理器拥有一个独立的内存空间；
- 所有处理器可以通过一个公共网络采用点对点方式通信；
- 所有处理器可以通过该网络实现同步；
- 程序以超步 (superstep) 为单位并行执行；
- 每个超步末进行栅栏同步，从而保证所有处理器同时进行下一个超步。

每个超步执行本地计算、全局通信两种操作：



# BSP 模型的量化评估

## BSP 模型参数

- $p$ : 处理器个数;
- $S$ : 总超步数;
- $g$ : 每单位消息的单边通信时间 ( $1/g$ = 通信带宽, 由硬件决定);
- $\ell$ : 每次栅栏同步的时间 (由硬件决定);
- $w_s$ : 第  $s$  超步本地计算的最大时间;
- $h_s$ : 第  $s$  超步单边通信的最大消息量。

采用 BSP 预测程序执行总时间:

$$\text{Time}_{BSP} = \sum_{s=1}^S w_s + g \sum_{s=1}^S h_s + \ell S.$$

# BSP 模型的评价

- 在一些情况下，本地计算可以与全局通信重叠，甚至进一步与栅栏同步重叠，此时

$$\text{Time}_{BSP} = \sum_{s=1}^S \max\{w_s, gh_s\} + \ell S, \quad \text{Time}_{BSP} = \sum_{s=1}^S \max\{w_s, gh_s, \ell\}.$$

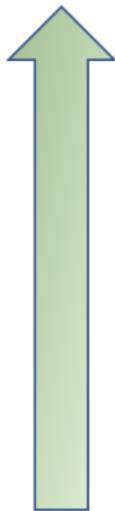
- 基于 BSP 模型的算法受计算机硬件体系架构的制约较小，容易编程实现，性能也相对容易预测。
- BSP 模型忽略了通信的延迟，因此传输  $m$  个长度为 1 的消息的开销等于传输 1 个长度为  $m$  的消息（事实上应该是大于）。
- 更精细的模型：LogP 模型 (Latency/overhead/gap/Proc) 等。

# 内容提纲

- ① 课程介绍
- ② 引言
- ③ 硬件体系架构
- ④ 小议并行算法与编程
- ⑤ 并行计算三大定律
- ⑥ 共享内存并行计算模型
- ⑦ 分布式并行计算模型
- ⑧ 并行编程模型

# 并行计算研究的几个主要视角

应用



- 应用视角：数学建模，算法设计与分析等。
- 算法视角：算法的并行化以及相关的主要原则等。
- 编程视角：采用何种方式编程实现并行算法。
- 性能视角：通过建立并行计算模型，指导算法设计、编程实现、性能优化等。
- 硬件视角：并行计算机体系架构。

硬件

课程从应用与硬件切入，通过讨论算法与性能，最终聚焦并行编程。

# 回顾三个重要的定律

$$S(n) = \frac{1}{\alpha + \frac{1-\alpha}{n}}, \quad (\text{阿姆达尔})$$

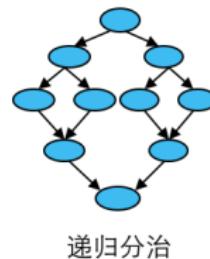
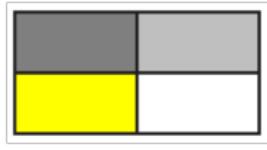
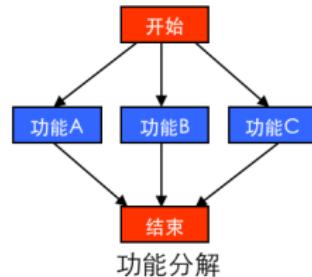
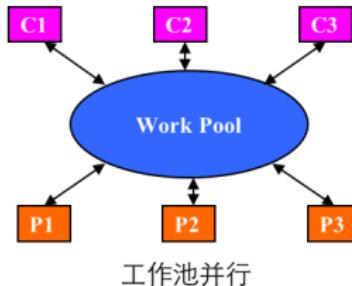
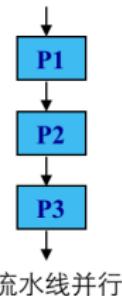
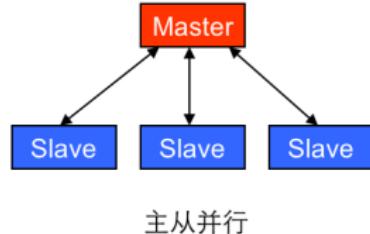
$$S'(n) = \alpha + (1 - \alpha)n, \quad (\text{古斯塔法森})$$

$$S^*(n) = \frac{\alpha + (1 - \alpha)G(n)}{\alpha + (1 - \alpha)\frac{G(n)}{n}}. \quad (\text{孙-倪})$$

|                       | 加速比 ( $n \rightarrow \infty$ )      | 并行效率 ( $n \rightarrow \infty$ ) |
|-----------------------|-------------------------------------|---------------------------------|
| 阿姆达尔                  | $S(n) \rightarrow \frac{1}{\alpha}$ | $E(n) \rightarrow 0$            |
| 古斯塔法森                 | $S'(n) \rightarrow \infty$          | $E'(n) \rightarrow 1 - \alpha$  |
| 孙-倪 ( $G(n) > O(n)$ ) | $S^*(n) \rightarrow \infty$         | $E^*(n) \rightarrow 1$          |

# 并行计算的原则与形式

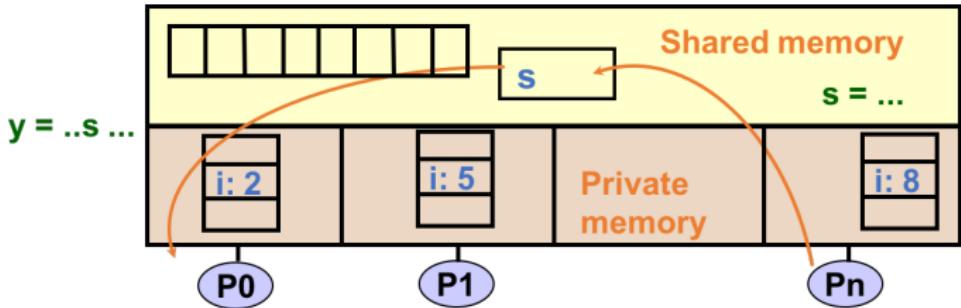
- 并行计算的八字原则：“负载均衡、通信极小”
- 并行计算的基本形式



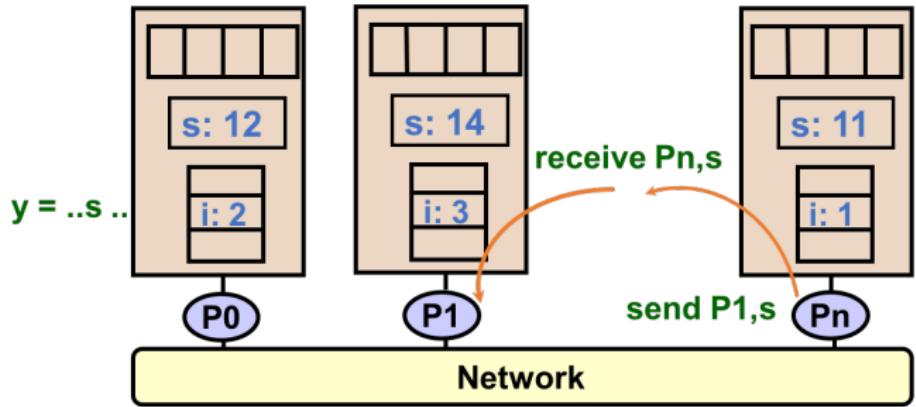
# 并行编程模型

- 自动并行
  - ❖ 提交串行程序，由编译器自动实现并行；
  - ❖ 希望十分渺茫。
- 共享内存并行编程
  - ❖ 适用于共享内存并行机，以线程为并行单位；
  - ❖ pthreads、OpenMP、CILK、TBB等。
- 消息传递
  - ❖ 适用于共享内存和分布式并行机；
  - ❖ PVM、MPI等。
- 数据并行
  - ❖ CUDA/OpenCL、Fortran 90、PGAS、MapReduce等。
- 混合并行
  - ❖ MPI + OpenMP、MPI + CUDA 等。

# 共享内存与消息传递



共享内存  
并行编程



消息传递  
并行编程