

并行与分布式计算基础：第二讲

杨超

chao_yang@pku.edu.cn

2019 秋



内容提纲

- ① 上次课程回顾
- ② 硬件体系架构
- ③ 小议并行算法与编程

上次课程回顾

① 上次课程回顾

② 硬件体系架构

③ 小议并行算法与编程

课程基本情况

- 课程名称：并行与分布式计算基础
- 授课教师：杨超 (chao_yang@pku.edu.cn, 理科 1 号楼 1520)
- 课程助教：尹鹏飞 (pengfeiyin@pku.edu.cn)
- 微信群



并行与分布式计算基础2019



该二维码7天内(9月16日前)有效，重新进入将更新

主要内容（暂定）

- 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

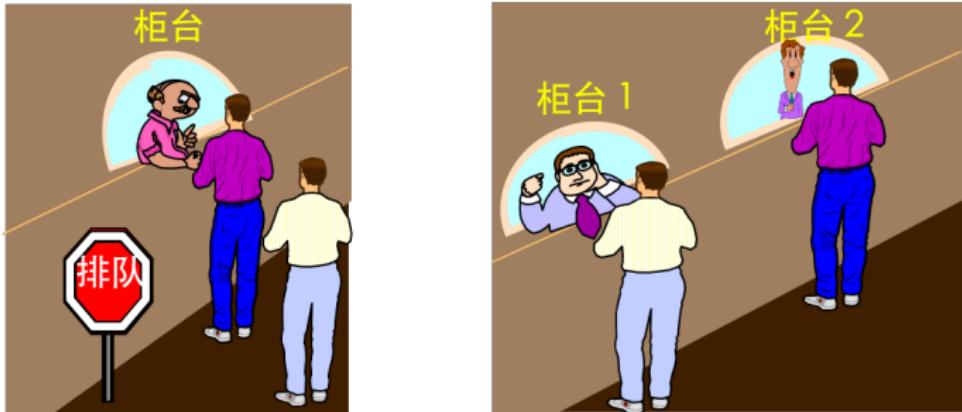
上课时间 (地点: 二教 211)

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)					

计算、串行计算、并行计算

计算已成为科技创新的第三大手段，已在各行各业中大显身手：

- 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。



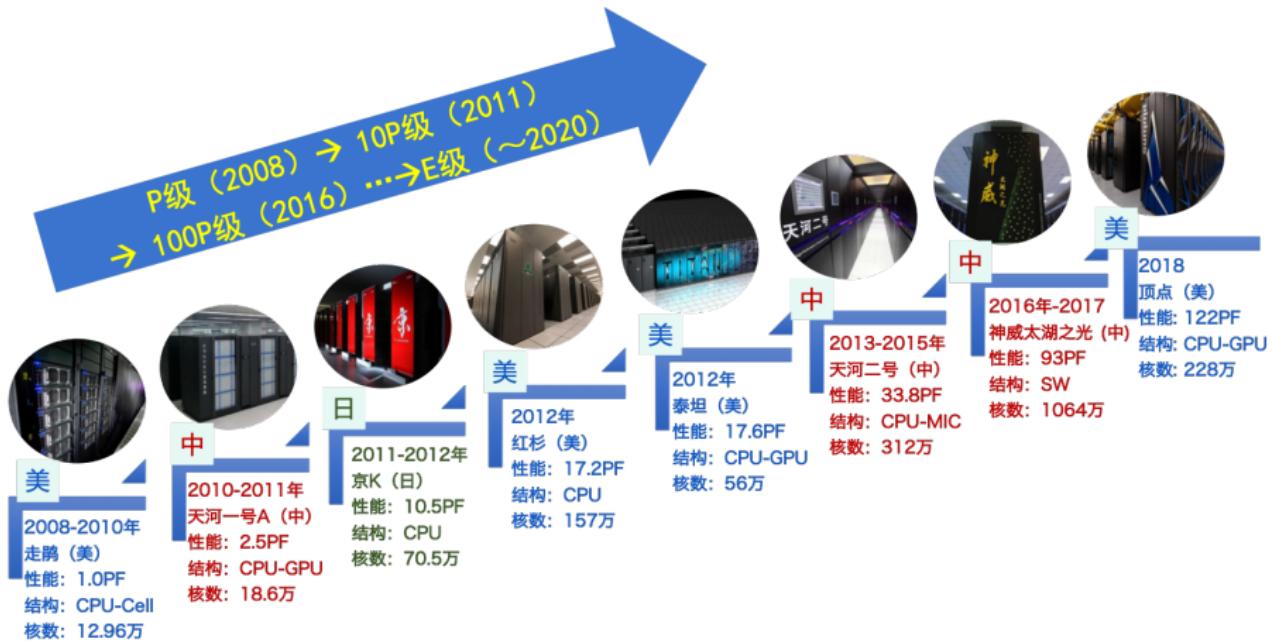
并行计算的主要目的：加速求解问题的速度、提高求解问题的规模等。

世界超级计算机 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/>



TOP500 排名第一的超级计算机

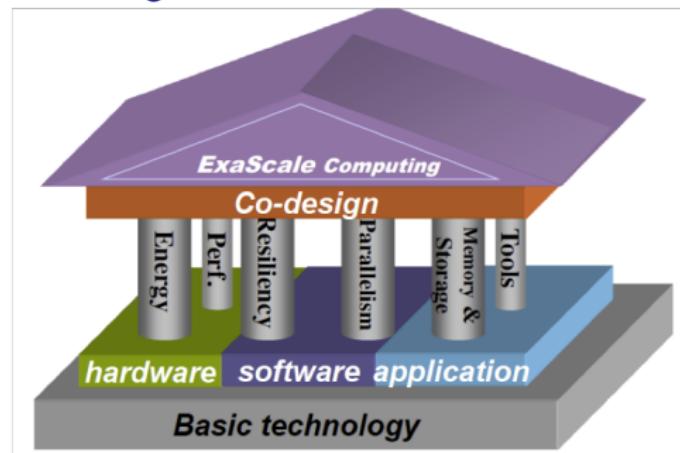


硬件发展趋势



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。

硬件体系架构

① 上次课程回顾

② 硬件体系架构

③ 小议并行算法与编程

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)

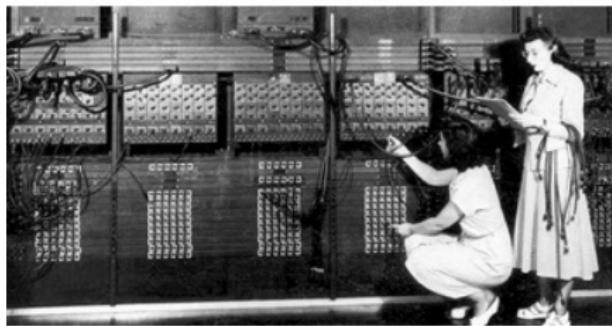
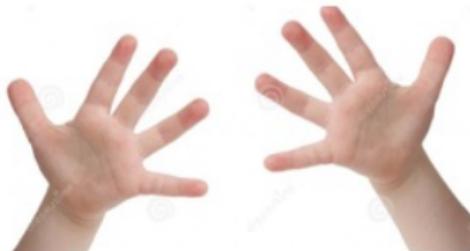


获奖人：John Hennessy (左) 与 David Patterson (右)

- 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。



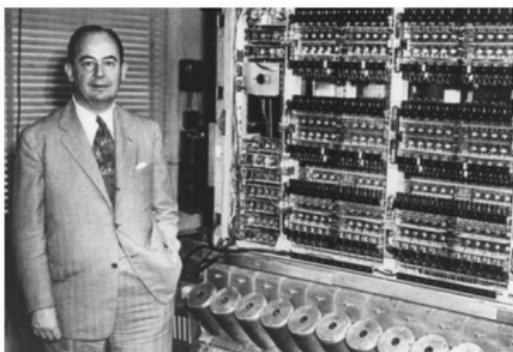
存储程序计算机 (Stored-Program Computer)

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)



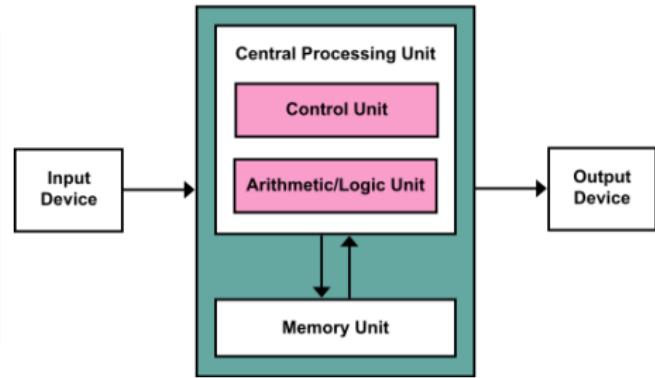
John von Neumann and EDVAC (1949)

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

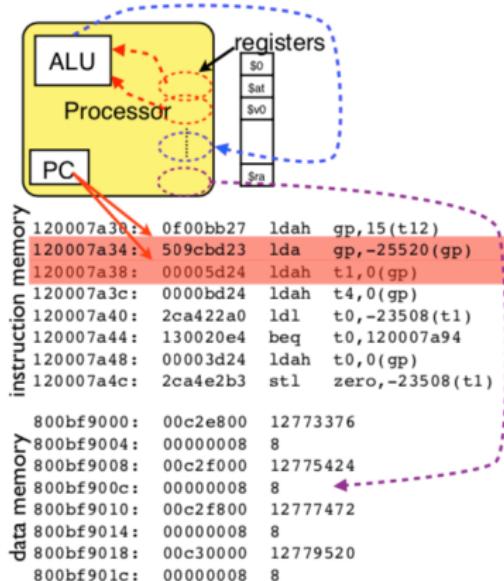
- 控制单元：解释指令
- 处理单元：执行指令
- 内存：存储数据和指令
- 输入/输出：与外界交互



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

指令是如何执行的？

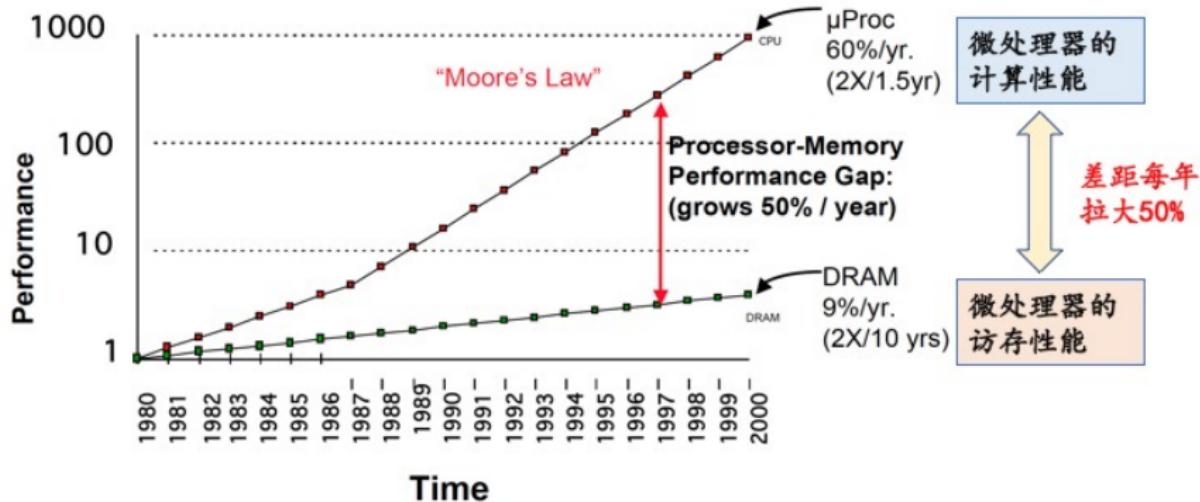
- Instruction fetch: where?
instruction memory
- Decode:
 - What's the instruction? **registers**
 - Where are the operands? **ALUs**
- Execute
- Memory access **data memory**
 - Where is my data?
- Write back **registers**
 - Where to put the result
- Determine the next PC



冯诺依曼架构的主要缺陷

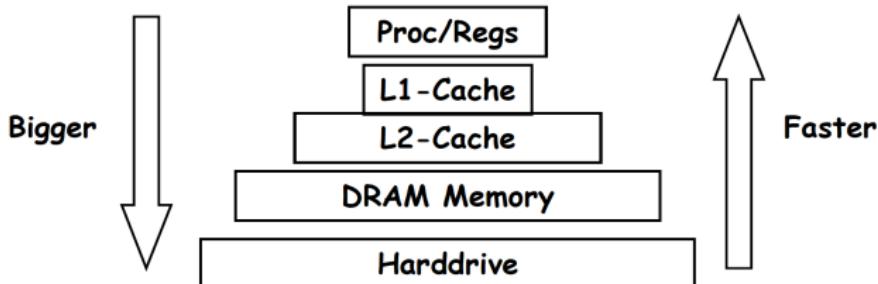
访存墙 (Memory Wall)

- 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 mm²

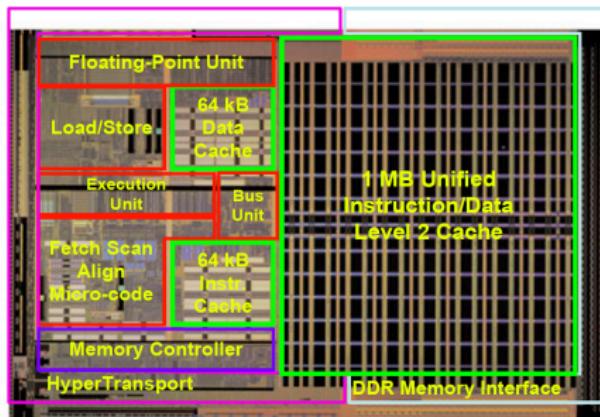
Look at the relative
sizes of each block:

50% cache

23% I/O

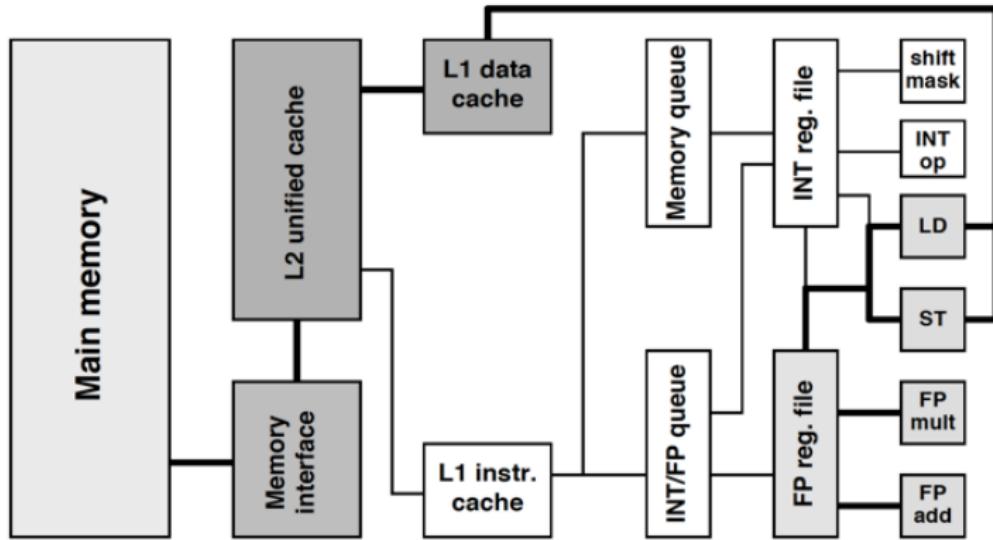
20% CPU logic

+ 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)

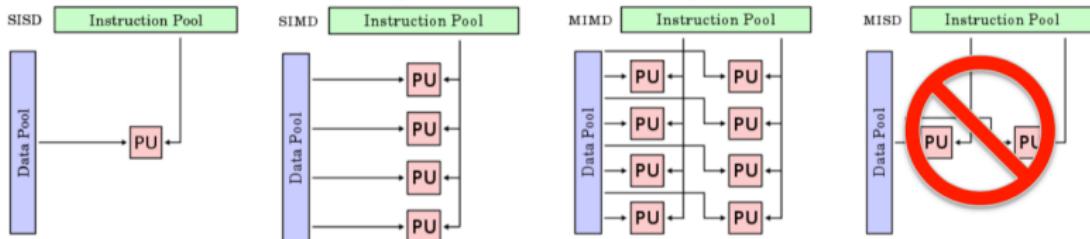
- 并行计算的一种分类方式，由 Michael J. Flynn 于 1966 年提出。
- 从两个正交的维度考虑：指令流（Instruction Stream）和数据流（Data Stream），其中每个维度有 Single 和 Multiple 两种可能选择。

S I S D	S I M D
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
M I S D	M I M D
Multiple Instruction stream Single Data stream	Multiple Instruction stream 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

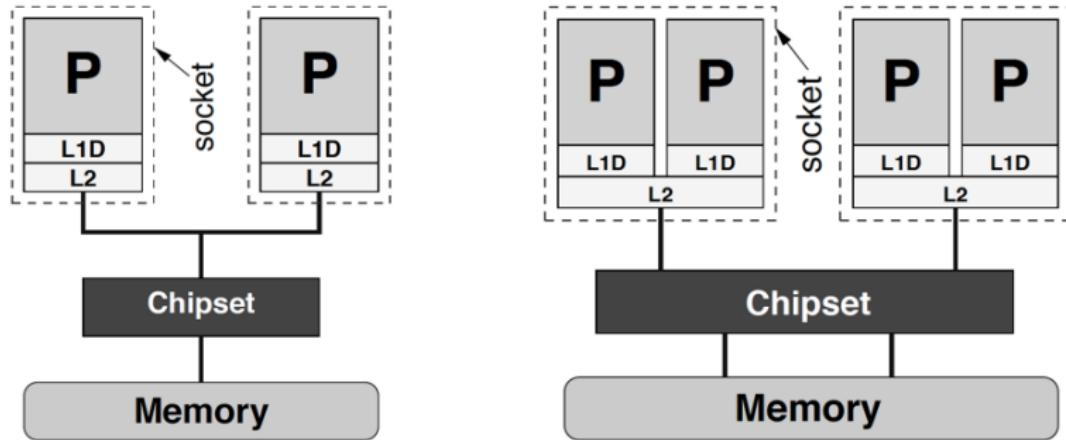


Single Program, Multiple Data (SPMD) is a natural generalization of Single Instruction, Multiple Data (SIMD) when each processing unit executes its own local copy of the instruction stream. These copies can branch differently depending upon the data.

c/o The Wikipedia

共享内存并行机 (1)

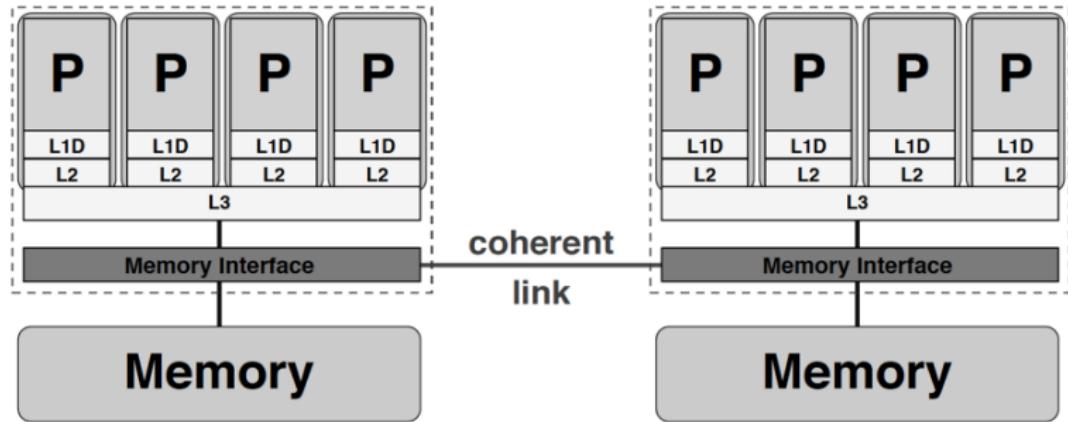
UMA (Uniform Memory Access) : 一致内存访问架构。



“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)

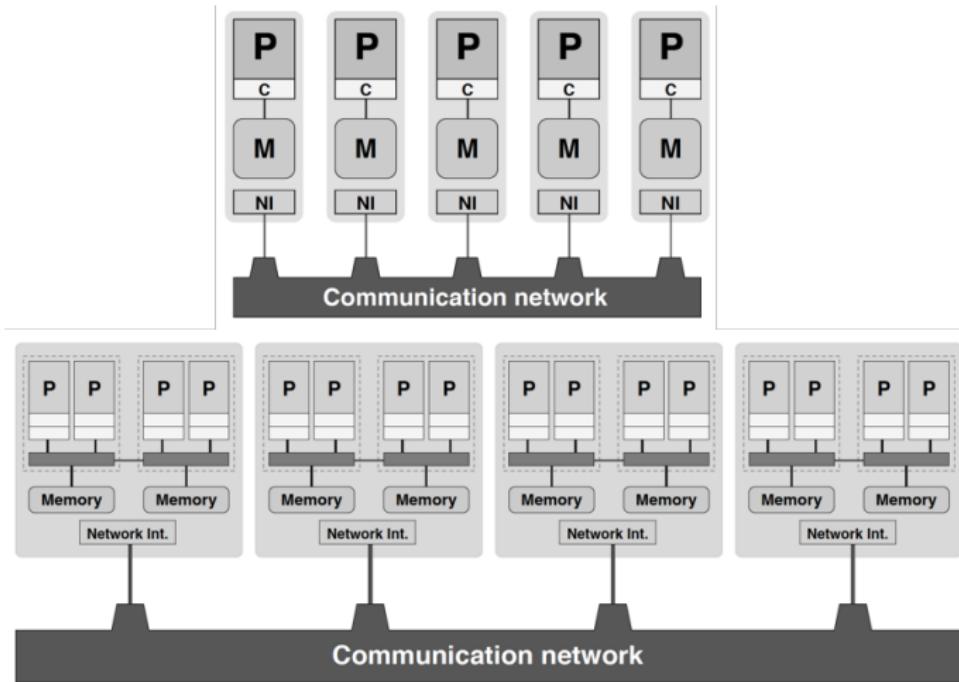
ccNUMA (cache-coherent Nonuniform Memory Access) : 缓存一致性的非一致内存访问架构。



思考：上图表示几路几核的 ccNUMA 架构？

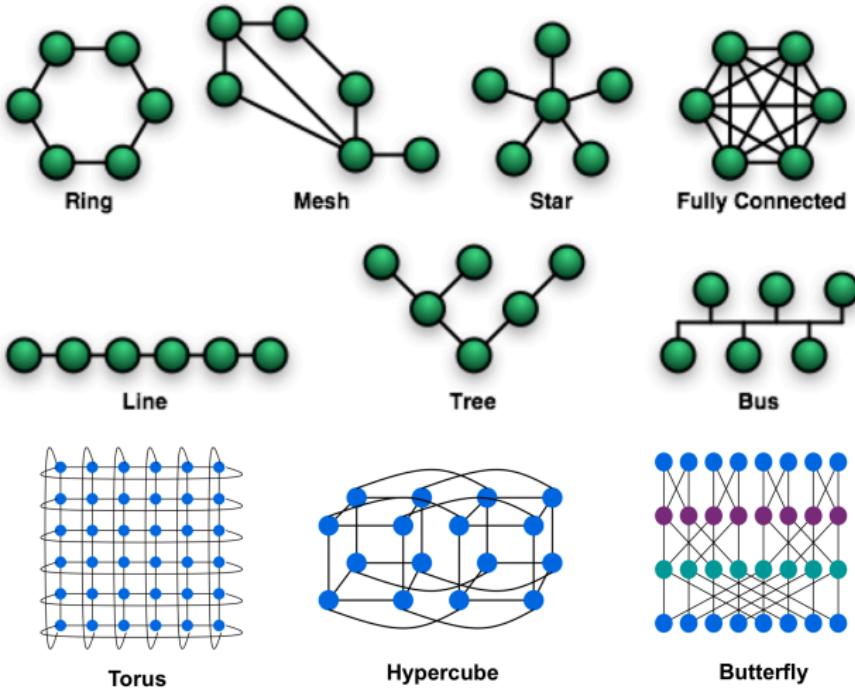
分布式并行机

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。



小议并行算法与编程

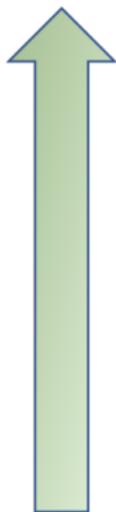
① 上次课程回顾

② 硬件体系架构

③ 小议并行算法与编程

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。

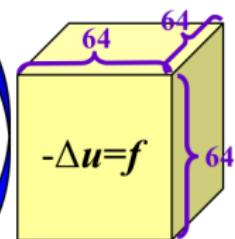
硬件

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 $N = n^3$

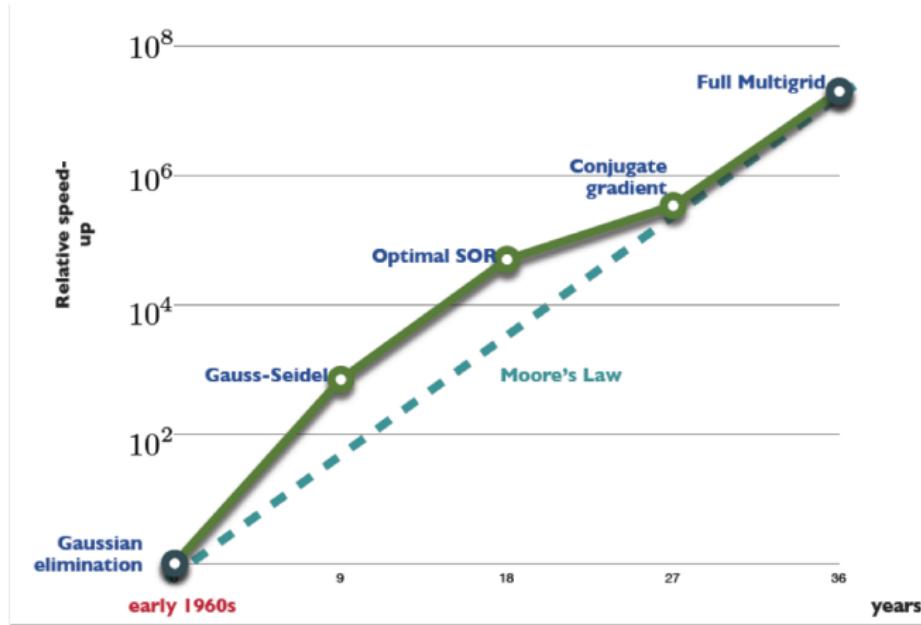
Year	Method	Reference	Storage	Flops
1947	GE (banded)	Von Neumann & Goldstine	n^5	n^7
1950	Optimal SOR	Young	n^3	$n^4 \log n$
1971	CG	Reid	n^3	$n^{3.5} \log n$
1984	Full MG	Brandt	n^3	n^3



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！

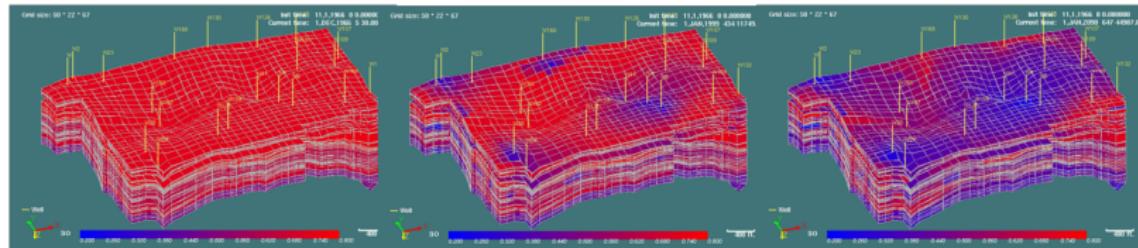


Algorithms can often speed up science as much as the Moore's law.
——摘自2005年美国总统报告, PITAC.

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

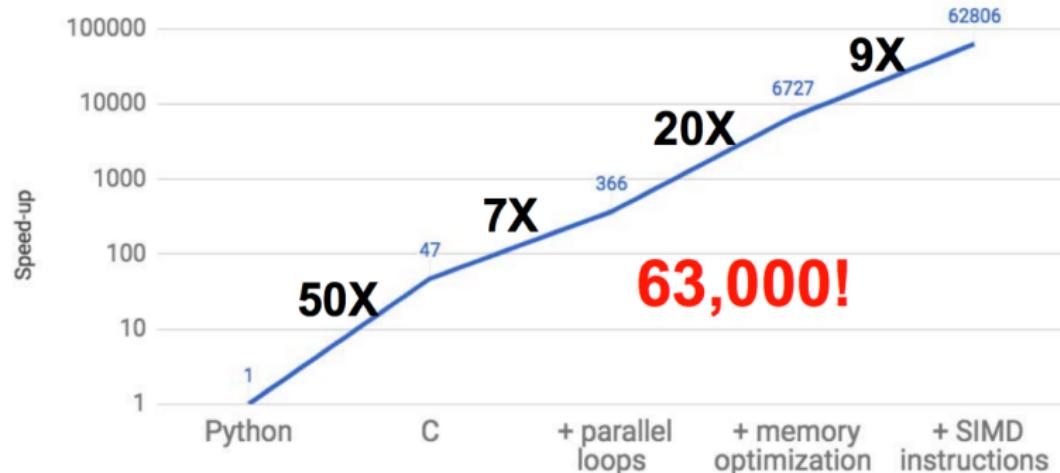


编程不同，效果天地之别

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：There's Plenty of Room at the Top, 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

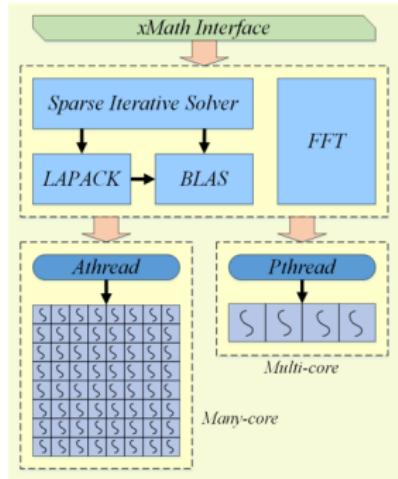
```
VMAD   a0,b0,t00,t00  
ADDL   A,16*SIZE(A  
LDDE   nb0,4*SIZE(B)  
  
VMAD   a0,b1,t04,t04  
LDDE   nb1,5*SIZE(B)  
  
VMAD   a4,b0,t01,t01  
VLD    na12,12*SIZE(A)  
  
VMAD   a4,b1,t05,t05  
VLD    na8,8*SIZE(A)  
  
VMAD   a0,b2,t08,t08  
LDDE   nb2,6*SIZE(B)  
  
VMAD   a0,b3,t12,t12  
LDDE   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)

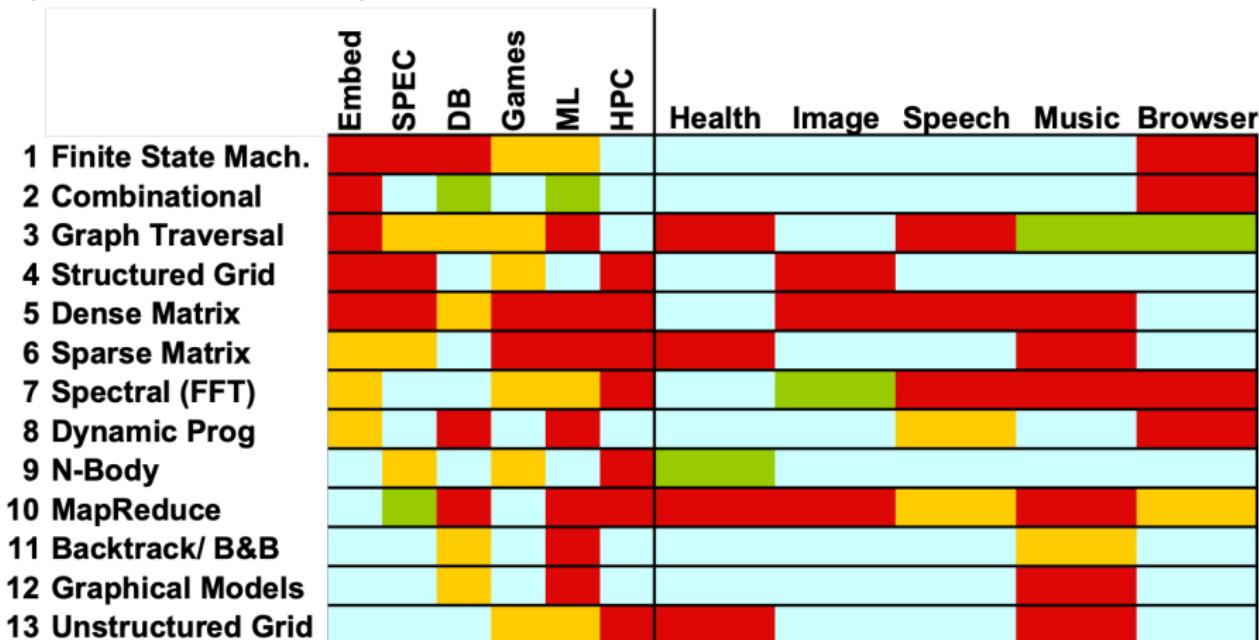
- 可在极短时间生成接近人类深度优化的高性能代码！
- 已应用于神威太湖之光的 xMath 高性能库 (100 万行代码)。



目录	
1	前言
2	使用环境
3	安装与配置
3.1	安装部署
3.2	运行部署
3.3	迁移部署及配置
4	API 参考
4.1	基本操作
4.2	矩阵操作与处理
5	附录
5.1	常见问题
5.2	鸣谢
5.3	高性能数学库最佳实践
6	附录
6.1	关于 xMath (含 API) 的更新与修正
6.2	关于 xMath 的帮助和支持
6.3	关于 xMath 的使用技巧
6.4	如何判断函数是否有效
6.5	如何判断函数返回值是否正确
7	参考文献
7.1	BLAS
7.2	LINPACK
7.3	QR
7.4	LU 分解
8	参考文献

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

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

