



中山大學
SUN YAT-SEN UNIVERSITY

计算机体系结构

主讲教师：胡 磊

中山大学 计算机学院
2024 年 秋季

教学内容

- 第 1 章 绪论
- **第 2 章 基准评测集**
- 第 3 章 高性能计算机的体系结构
- 第 4 章 高性能处理器的并行计算技术
- 第 5 章 高性能计算机的存储层次
- 第 6 章 高性能计算机的互连网络
- 第 7 章 异构计算体系结构
- 第 8 章 领域专用体系结构

第2章 基准评测集

教学目的

- **掌握**: CPU性能; 加速比; Amdahl定律;
- **熟悉**: Gustafson 定律; 并行算法 (程序) 的设计步骤;
- **了解**: Sun-Ni 定律; 可扩展性评测标准; 基准评测集。

课程内容

- Computer Architecture, A Quantitative Approach (6th Ed.), 2017
 - Chapter 1. Fundamentals of Quantitative Design and Analysis
- 并行计算机体系结构 (第2版), 2021
 - 第二章 性能评测

提 纲

- (一) 计算机性能
- (二) 基准评测集
- (三) 如何提高高性能

(一) 计算机性能



计算机性能

- 计算机性能
 - 通常指机器的速度，是程序执行时间的倒数
- 程序执行时间
 - Wall-clock time (response time, or elapsed time)
 - 指用户的响应时间 (访问磁盘和访问存储器的时间, CPU时间, I/O时间以及操作系统的开销)
- CPU时间
 - 表示CPU的工作时间, 不包括I/O等待时间和运行其它任务的时间

The Processor Performance Equation

CPU time for a program can then be expressed two ways:

$$\left\{ \begin{array}{l} \text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time} \\ \text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}} \end{array} \right.$$

clock cycles per instruction (CPI): $\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$



$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

The Processor Performance Equation

- Different instruction types having different CPIs

$$\text{CPU clock cycles} = \sum_{i=1}^n \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \left(\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

- Overall CPI

$$\text{CPI} = \frac{\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i}{\text{Instruction count}} = \sum_{i=1}^n \frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i$$

The Processor Performance Equation

- It is difficult to change one parameter in complete isolation from others
 - Clock cycle time: Hardware technology and organization
 - CPI: Organization and instruction set architecture
 - Instruction count: Instruction set architecture and compiler technology

硬件/软件	影响什么	如何影响
算法	程序的执行指令数	算法决定源程序执行指令的数目，好的算法可以大幅度减少运算的次数
编程语言	程序的执行指令数	编程语言可能对执行指令数产生巨大的影响，比如解释执行、即时编译或者原生编译的三类语言完成同样的功能所需要的指令数可能有数量级的差异
编译器和库	程序的执行指令数、CPI	编译器和库决定了源程序到计算机指令的翻译过程，编译程序的效率既影响到程序的执行指令数又影响到CPI，如Intel的ICC编译器编出来的程序，效率可比GCC高30%，其能充分利用向量化指令和针对处理器结构的优化
指令系统结构	程序的执行指令数、CPI和时钟频率	指令系统结构影响到CPU性能的3个方面，因为它影响到完成某个功能所需的指令数、每条指令的周期数，以及处理器的时钟频率
微体系结构	CPI和时钟频率	微体系结构的改进可以降低CPI，也可以细分流水线来提高频率
物理设计	时钟频率	物理设计和电路的进步可以降低时钟周期，从而提高时钟频率
工艺	时钟频率	工艺的进步使得晶体管变快从而提高时钟频率

CPU性能公式

- MIPS (Million Instructions Per Second)
 - 为了简化对于不同计算机的横向比较，一个常用的计算机性能评价指标是每秒百万次指令数MIPS

$$\text{MIPS} = I_N / (T_E \times 10^6) = R_C / (\text{CPI} \times 10^6)$$

其中， T_E 表示程序执行时间， R_C 表示时钟速率，是 T_C 的倒数

有时还使用相对MIPS_{Ref}这一标准

$$\text{MIPS}_{\text{Rel}} = (T_{\text{Ref}} / T_V) \times \text{MIPS}_{\text{Ref}}$$

MIPS指标忽略了不同指令系统之间的差异

计算机性能

- FLOPS (Floating-point Operations Per Second)
 - 由于 MIPS 指标的缺陷，来自美国劳伦斯·利弗莫尔国家实验室的弗兰克·H·麦克马洪 (Frank H. McMahon) 提出了更通用的计算机性能瓶颈指标，即每秒浮点运算次数 FLOPS

$$\text{MFLOPS} = I_{FN} / (T_E \times 10^6)$$

其中， I_{FN} 表示程序中的浮点运算次数

一般认为在标量计算机中执行一次浮点运算平均需要3条指令，故有 $1\text{MFLOPS} \approx 3\text{MIPS}$

FLOPS

- 为了方便表示，基于 FLOPS，延伸出来一系列扩展指标，包括：
 - 每秒百万次浮点运算次数 (Mega Floating-point Operations Per Second, MFLOPS, megaFLOPS, MFlop/s)
 - 每秒十亿次浮点运算次数 (Giga Floating-point Operations Per Second, GFLOPS, gigaFLOPS, GFlop/s)
 - 每秒万亿次浮点运算次数 (Tera Floating-point Operations Per Second, TFLOPS, teraFLOPS, TFlop/s)
 - 每秒千万亿次浮点运算次数 (Peta Floating-point Operations Per Second, PFLOPS, petaFLOPS, PFlop/s)
 - 每秒**百亿亿次**浮点运算次数 (Exa Floating-point Operations Per Second, EFLOPS, exaFLOPS, **EFlop/s**)

计算机性能

名称	符号	含 意	单 位
机器规模	P	处理器的数目	/
时钟速率	f	时钟周期长度的倒数	GHZ
工作负载	W	计算操作的数目	Gflop
顺序执行时间	T _s (or T ₁)	程序在单处理机上的运行时间	s(秒)
并行执行时间	T _p (or T _n)	程序在并行机(规模为n)上的运行时间	s(秒)
速度	R _n =W/T _n	每秒十亿次浮点运算	Gflop/s
加速比	S _n =T ₁ /T _n	衡量并行机有多快	/
效率	E _n =S _n /n	衡量并行机的效率	/
峰值速度	R _{peak} =n R' _{peak}	所有处理器峰值速度之积, R' _{peak} 为一个处理器的峰值速度	Gflop/s
利用率	U=R _n /R _{peak}	可达速度与峰值速度之比	无量纲
通信延迟	t _o	传送0字节或单字的时间	μs(微秒)
渐近带宽	r _∞	传送长消息通信速率	MB/s

计算机性能评测的意义

- 计算机性能评测的意义
 - 提高计算机的使用效率
 - 减少用户购机盲目性，降低投资风险
 - 改进系统结构设计，提高机器的性能
 - 促进软/硬件结合，合理功能划分
 - 优化 “结构-算法-应用” 的最佳组合
 - 提供客观、公正的评价计算机的标准

计算机性能

1. 机器级性能评测：

- CPU和存储器的某些基本性能指标；并行和通信开销分析；计算机的可用性与好用性以及机器成本、价格与性/价比

2. 算法级性能评测：

- 加速比、效率、可扩展性

3. 程序级性能评测：

- Benchmark, 包括基本测试、数学库测试和并行测试程序等

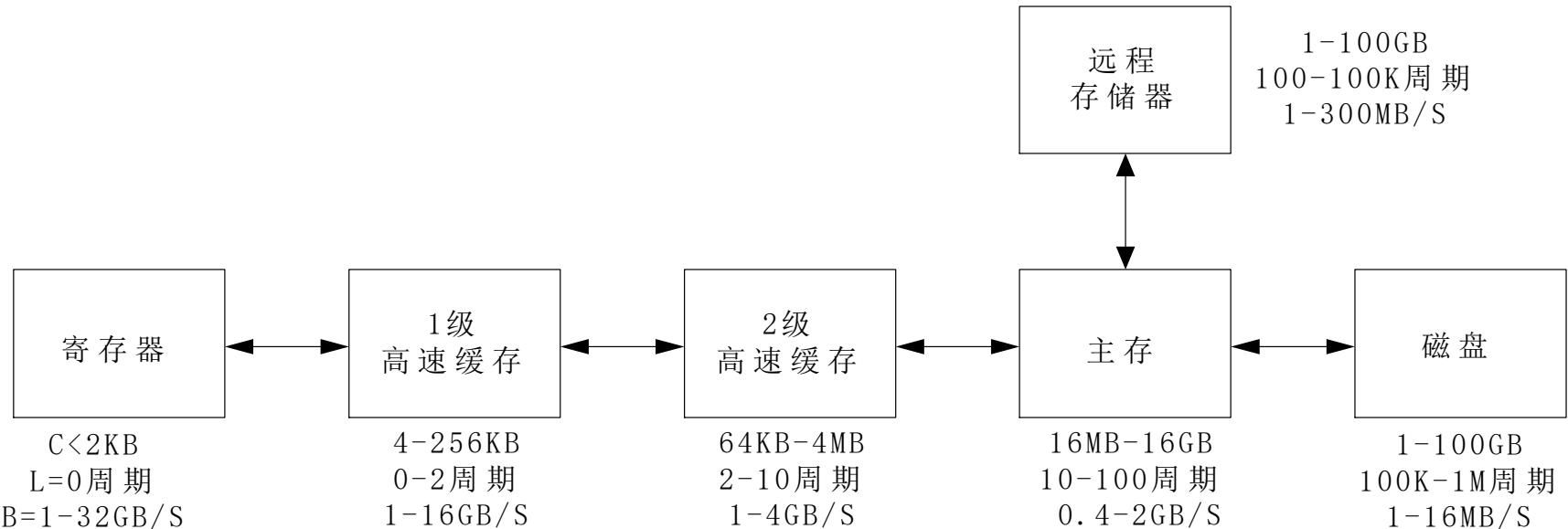
机器级性能评测

- CPU的某些基本性能指标
 - 工作负载
 - 执行时间
 - 浮点运算数
 - 指令数目
 - 并行执行时间 T_{comput} 为计算时间, T_{paro} 为并行开销时间, T_{comm} 为相互通信时间

$$T_n = T_{comput} + T_{paro} + T_{comm}$$

机器级性能评测

- 存储器的某些基本性能指标
 - 存储器层次结构



- 存储器带宽

机器级性能评测

- 机器的成本、价格与性/价比
 - 机器的性能/价格比 Performance/Cost Ratio：
系指用单位代价（通常以百万美元表示）所
获取的性能（通常以MIPS或GFLOPS表示）
 - 利用率（Utilization）：可达到的速度与峰值
速度之比

算法级性能评测

- 加速比性能定律
 - 并行系统的加速比是指对于一个给定的应用，并行算法（或并行程序）的执行速度相对于串行算法（或串行程序）的执行速度加快了多少倍
 - Amdahl 定律
 - Gustafson 定律
 - Sun-Ni 定律
- 可扩展性评测标准
 - 等效率度量标准
 - 等速度度量标准
 - 平均延迟度量标准

算法级性能评测

- 加速比性能定律
 - 并行系统的加速比是指对于一个给定的应用，平行算法（或并行程序）的执行速度相对于串行算法（或串行程序）的执行速度加快了多少倍
 - Amdahl 定律
 - Gustafson 定律
 - Sun-Ni 定律
- 可扩展性评测标准
 - 等效率度量标准
 - 等速度度量标准
 - 平均延迟度量标准

Speedup

- Speedup of computer X relative to computer Y
 - X is n times as fast as Y

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{1}{\frac{\text{Performance}_Y}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

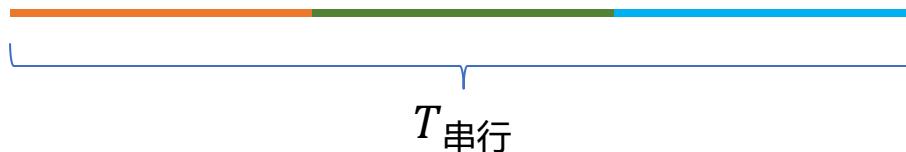
并行加速比

- 一个并行程序的加速比 (Speedup)：
 - T_p : 并行执行时间
 - T_s : 串行执行时间

$$S = \frac{T_s}{T_p}$$

线性加速比

- 加速前：



- 在 3 核的计算机：



$$T_{\text{并行}} = T_{\text{串行}} / 3$$

- 对于 p 核的高性能计算机：

$$T_{\text{并行}} = \frac{T_{\text{串行}}}{p}$$

此时，并行程序拥有 **线性加速比 (p)**

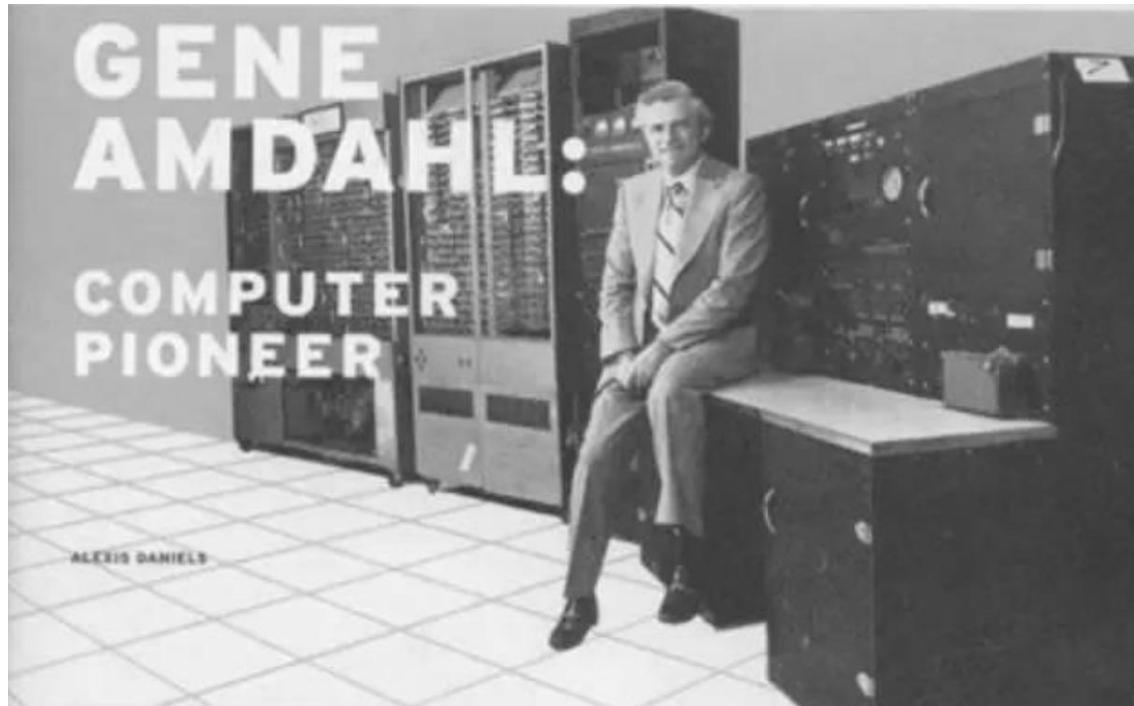
效率

- 效率 (Efficiency)：
 - 加速比 S 与线性加速比 p 的比值

$$E = \frac{S}{p} = \frac{\frac{T_s}{T_p}}{p} = \frac{T_s}{p \cdot T_p}$$

Amdahl 定律

- Amdahl 定律
 - 1967 年，作为 IBM360 系列机的主要设计者，美国计算机科学家吉恩·阿姆达尔（Gene Amdahl）提出了 Amdahl 定律



Amdahl 定律

- Amdahl 定律
 - Amdahl 定律给出了 **固定负载** 条件下程序并行化效率提升的理论上限
 - 在并行计算领域，Amdahl 定律常用于预测使用多个处理器后的理论加速比上界

Amdahl定律

- P: 处理器数
- W: 问题规模 (计算负载、工作负载, 给定问题的总计算量)
 - W_s : 应用程序中的串行分量, *f是串行分量比例* ($f = W_s/W$)
 - W_p : 应用程序中可并行化部分, $1-f$ 为并行分量比例
 - $W_s + W_p = W$
- $T_s = T_1$: 串行执行时间, T_p : 并行执行时间
- S: 加速比, E: 效率
- 出发点:
 - 固定不变的计算负载
 - 固定的计算负载分布在多个处理器上
 - 增加处理器加快执行速度, 从而达到了加速的目的

Amdahl定律

- 固定负载的加速公式： $S = \frac{W_s + W_p}{W_s + W_p / p}$
- $W_s + W_p$ 可相应地表示为 $[f + (1 - f)]W$

$$S = \frac{f + (1 - f)}{f + \frac{1 - f}{p}} = \frac{p}{1 + f(p - 1)}$$

Amdahl定律

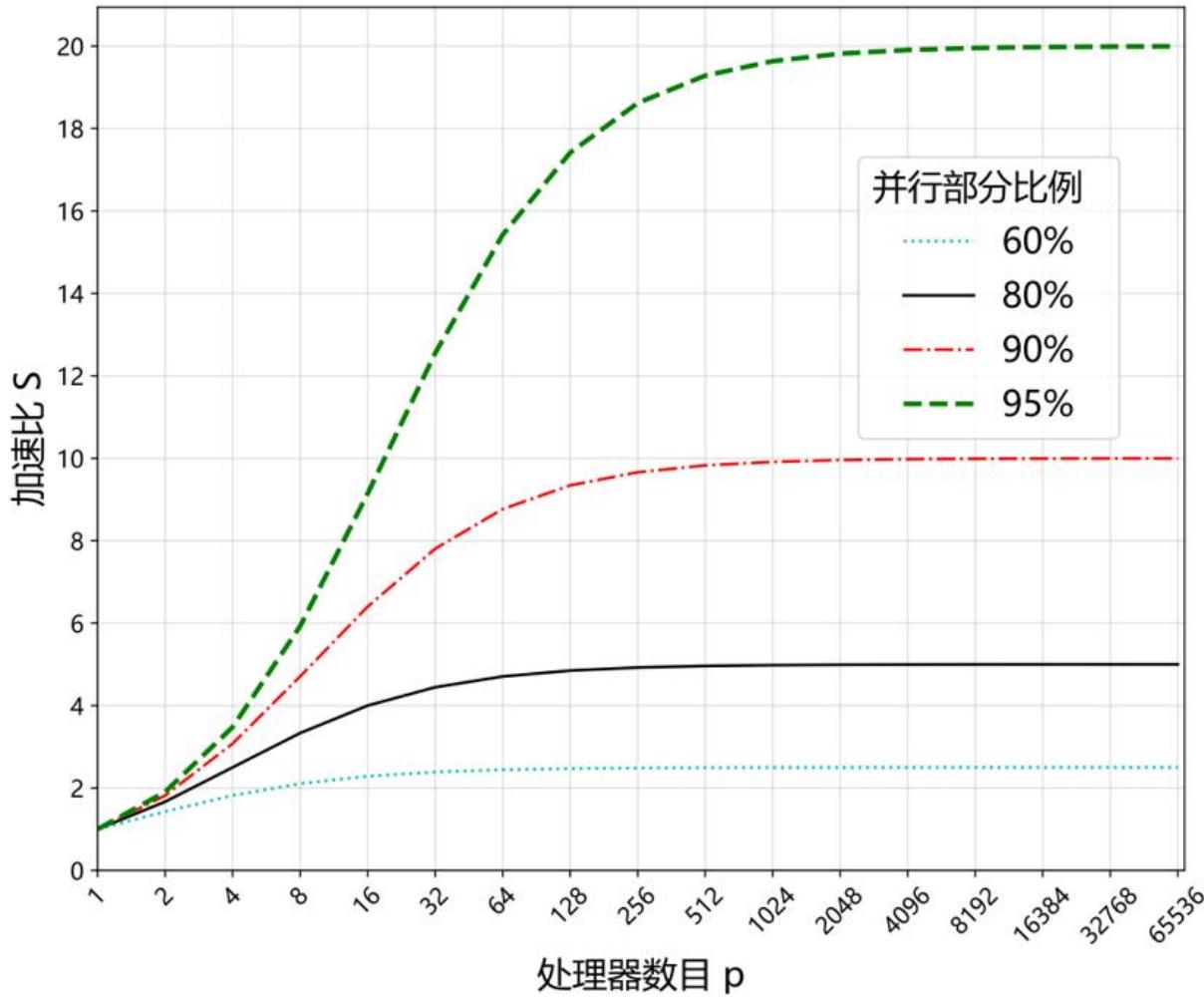
- 固定负载的加速公式：

$$S = \frac{f + (1 - f)}{f + \frac{1 - f}{p}} = \frac{p}{1 + f(p - 1)}$$

- $p \rightarrow \infty$ 时， 上式极限为： $S = 1 / f$

加速比是有上限的

Amdahl定律



Amdahl定律

- 除非一段串行程序的 **所有部分** 都可以 **并行化**,
否则可以达到的加速比是 **非常有限** 的
 - 不论使用多少核
- 并行程序的加速比与串行执行所需要的时间无关,**
只与 可并行部分的长度 w_p **和 并行系数 P 有关**

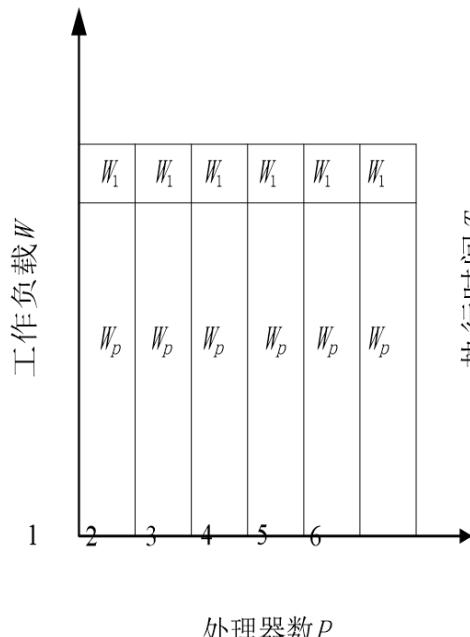
Amdahl定律

- 实际并行加速不仅受限于程序的串行分量，也受到并行程序运行时额外开销的影响，令 W_o 为额外开销

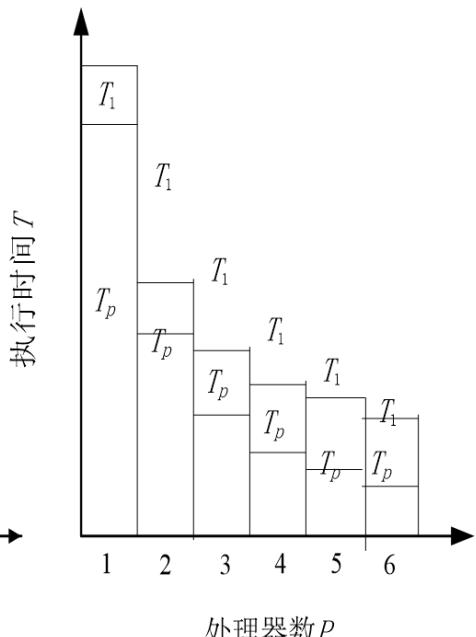
$$S = \frac{W_S + W_P}{\frac{W_S}{p} + \frac{W_P}{p} + W_o} = \frac{W}{fW + \frac{W(1-f)}{p} + W_o} = \frac{p}{1 + f(p-1) + W_o p / W}$$

- $p \rightarrow \infty$?

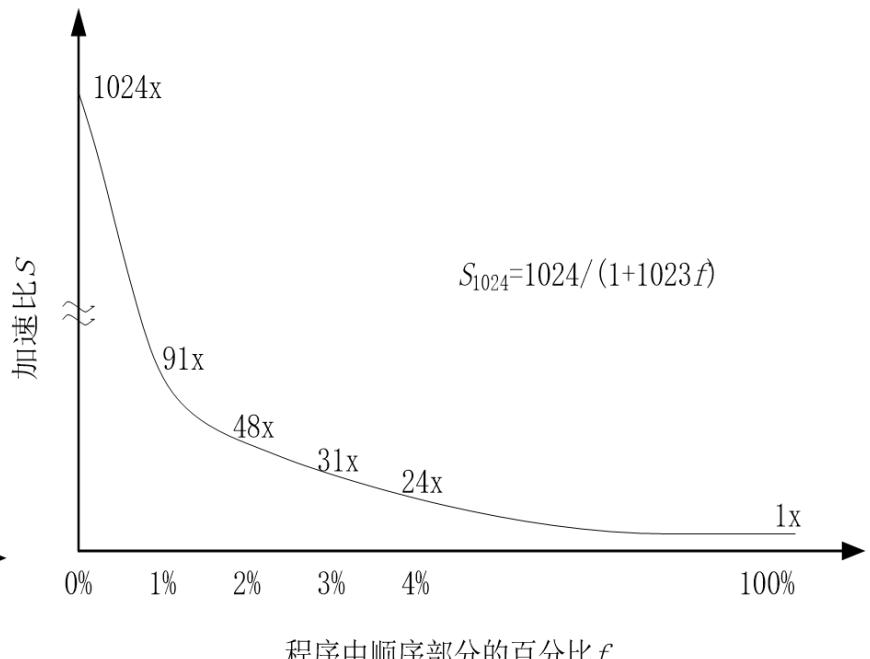
Amdahl定律



(a)



(b)



(c)

Amdahl定律

- Amdahl 定律是适用于**固定负载**情况下的加速比模型，它的基本出发点是：
 - (1) 在科学计算领域，有很多问题由于规模较大导致无法满足实时性要求，即问题求解的时间开销是个关键因素，而问题的规模也就是负载是固定不变的
 - (2) 由于计算负载是固定不变的，因此增加参与计算的处理器数量能够提高程序的执行速度，达到加速的目的

Gustafson 定律

- Amdahl 定律：当处理器核数增加到无穷大时，加速比会趋近于一个上限
- 但 Gustafson (古斯塔夫森) 在 1988 年进行的实际实验，在一个拥有 1024 个处理器的计算机，分别获得了 $1021x/1020x/1016x$ 的加速比



Gustafson 得到了与 Amdahl 不一样的结论！

Gustafson 定律

- 分析：
 - 对于很多大型计算，精度要求很高，即在此类应用中**精度**是个关键因素，而**计算时间是固定不变的**。此时为了提高精度，必须加大计算量，相应地亦必须增多处理器数才能维持时间不变
 - 除非学术研究，在实际应用中没有必要固定工作负载而计算程序运行在不同数目的处理器上，**增多处理器必须相应地增大问题规模才有实际意义**

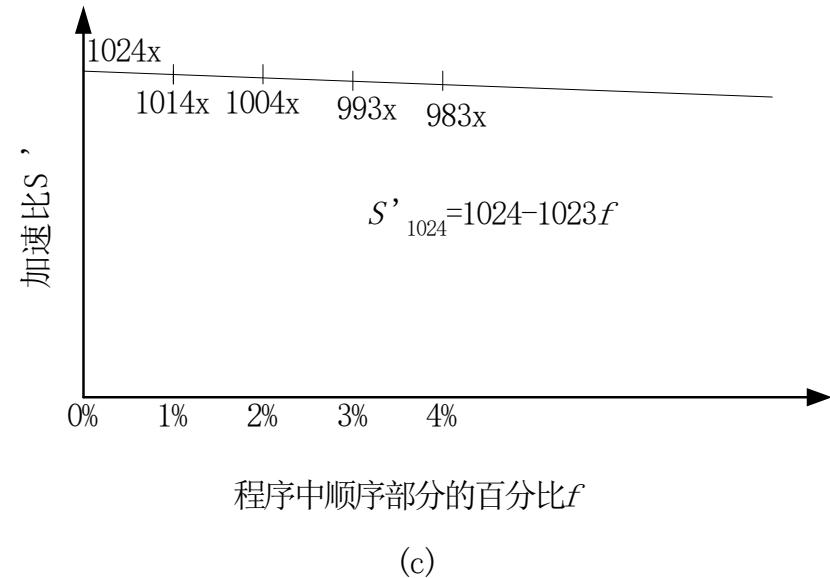
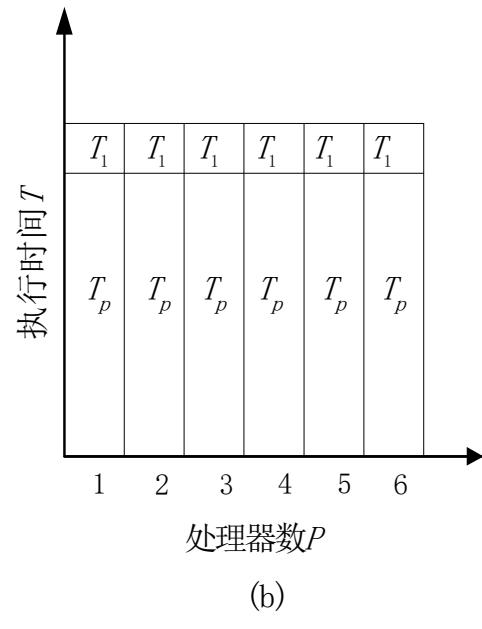
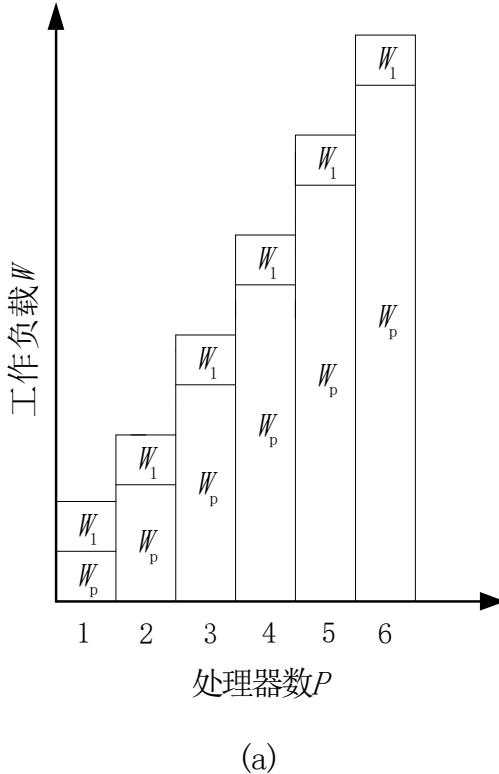
- Gustafson 加速定律： $S' = \frac{W_s + pW_p}{W_s + p \cdot W_p / p} = \frac{W_s + pW_p}{W_s + W_p}$
(放大问题规模的加速公式)

$$S' = f + p(1-f) = p + f(1-p) = p - f(p-1)$$

- 并行开销 W_o ：

$$S' = \frac{W_s + pW_p}{W_s + W_p + W_o} = \frac{f + p(1-f)}{1 + W_o / W}$$

Gustafson 定律



Sun-Ni 定律

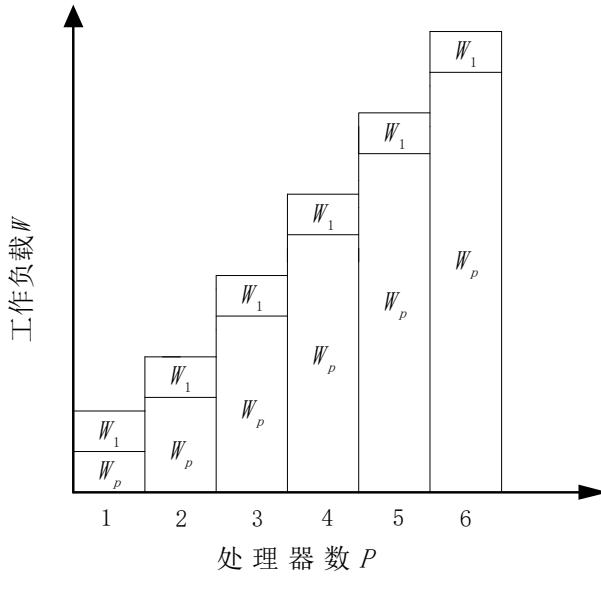
- 基本思想 (**存储受限的加速定律**)：
 - 只要存储空间许可，应尽量增大问题规模以产生更好和更精确的解 (此时可能使执行时间略有增加)
 - 假定在单节点上使用了全部存储容量 M 并在相应于 W 的时间内求解之，此时工作负载 $W = fW + (1-f)W$
 - 在 p 个节点的并行系统上，能够求解较大规模的问题是因为存储容量可增加到 pM 。令因子 $G(p)$ 反映**存储容量增加到 p 倍时并行工作负载的增加量**，所以扩大后的工作负载 $W = fW + (1-f)G(p)W$
- 存储受限的加速公式：

$$S'' = \frac{fW + (1-f)G(p)W}{fW + (1-f)G(p)W / p} = \frac{f + (1-f)G(p)}{f + (1-f)G(p) / p}$$

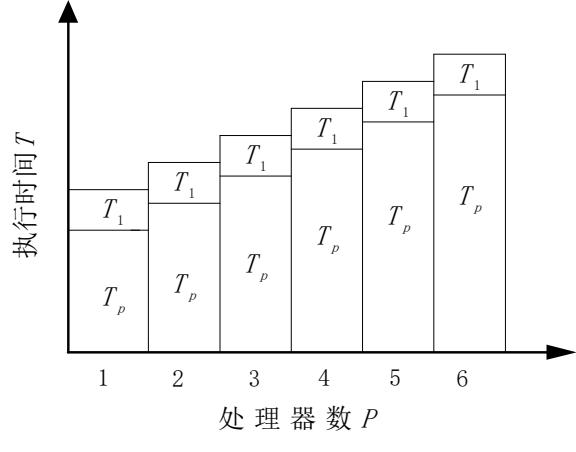
- 并行开销 W_o :

$$S'' = \frac{fW + (1-f)WG(p)}{fW + (1-f)G(p)W / p + W_o} = \frac{f + (1-f)G(p)}{f + (1-f)G(p) / p + W_o / W}$$

Sun-Ni 定律



(a)



(b)

- $G(p) = 1$ 时就是Amdahl加速定律；
- $G(p) = p$ 变为 $f + p(1-f)$, 就是Gustafson加速定律
- $G(p) > p$ 时, 相应于计算机负载比存储要求增加得快, 此时 Sun-Ni 加速均比 Amdahl 加速和 Gustafson 加速高

加速比讨论

- 参考的加速经验公式： $P/\log P \leq S \leq P$
- 线性加速比：
 - 很少通信开销的矩阵相加、内积运算等
- $P/\log P$ 的加速比：
 - 分治类的应用问题
 - 类似二叉树，树的同级可并行执行，但向根逐级推进时，并行度将逐渐减少
- 通信密集类的应用问题：
 - $S = 1 / C(P)$ ，其中， $C(P)$ 是 P 个处理器的某一通信函数（线性/对数）
- 超线性加速
 - 并行搜索算法（提前取消无谓的搜索分支）
- 绝对加速：最佳并行算法与串行算法相比
- 相对加速：同一算法在单机和并行机的运行时间之比

算法级性能评测

- 加速比性能定律
 - 并行系统的加速比是指对于一个给定的应用，并行算法（或并行程序）的执行速度相对于串行算法（或串行程序）的执行速度加快了多少倍
 - Amdahl 定律
 - Gustafson 定律
 - Sun Ni 定律
- 可扩展性评测标准
 - 等效率度量标准
 - 等速度度量标准
 - 平均延迟度量标准

可扩展性评测标准

- 并行计算的可扩展性 (Scalability)
 - 可扩展性最简朴的含意是在确定的应用背景下，计算机系统（或算法或程序等）性能随处理器数的增加而按比例提高的能力
- **影响加速比的因素：**处理器数 p 与 问题规模 W
 - 求解问题中的串行分量
 - 并行处理所引起的额外开销（通信、等待、竞争、冗余操作和同步等）
 - 加大的处理器数超过了算法中的并发程度
- 增加问题的规模有利于提高加速的因素：
 - 较大的问题规模可提供较高的并发度
 - 额外开销的增加可能慢于有效计算的增加
 - 算法中的串行分量比例不是固定不变的（串行部分所占的比例随着问题规模的增大而缩小）
- **增加处理器数会增大额外开销和降低处理器利用率**，所以对于一个特定的并行系统（算法或程序），它们能否有效利用不断增加的处理器的能力应是受限的，而度量这种能力就是可扩展性这一指标

可扩展性评测标准

- 可扩展性：调整什么 和 按什么比例调整
 - 并行计算要调整的是处理器数 p 和问题规模 W
 - 两者可按不同比例进行调整，此比例关系（可能是线性的，多项式的或指数的等）就反映了可扩展的程度
- 并行算法和体系结构
- 可扩展性研究的主要目的：
 - 确定解决某类问题用何种并行算法与何种并行体系结构的组合，可以有效地利用大量的处理器
 - 对于运行于某种体系结构的并行机上的某种算法当移植到大规模处理机上后运行的性能
 - 对固定的问题规模，确定在某类并行机上最优的处理器数与可获得的最大的加速比
 - 用于指导改进并行算法和并行机体系结构，以使并行算法尽可能地充分利用可扩充的大量处理器
- 目前无一个公认的、标准的和被普遍接受的严格定义和评判它的标准

等效率度量标准

- 令 t_e^i 和 t_o^i 分别是并行系统上第 i 个处理器的 **有用计算时间** 和 **额外开销时间** (包括通信、同步和空闲等待时间等)

$$T_e = \sum_{i=0}^{p-1} t_e^i \quad T_o = \sum_{i=0}^{p-1} t_o^i$$

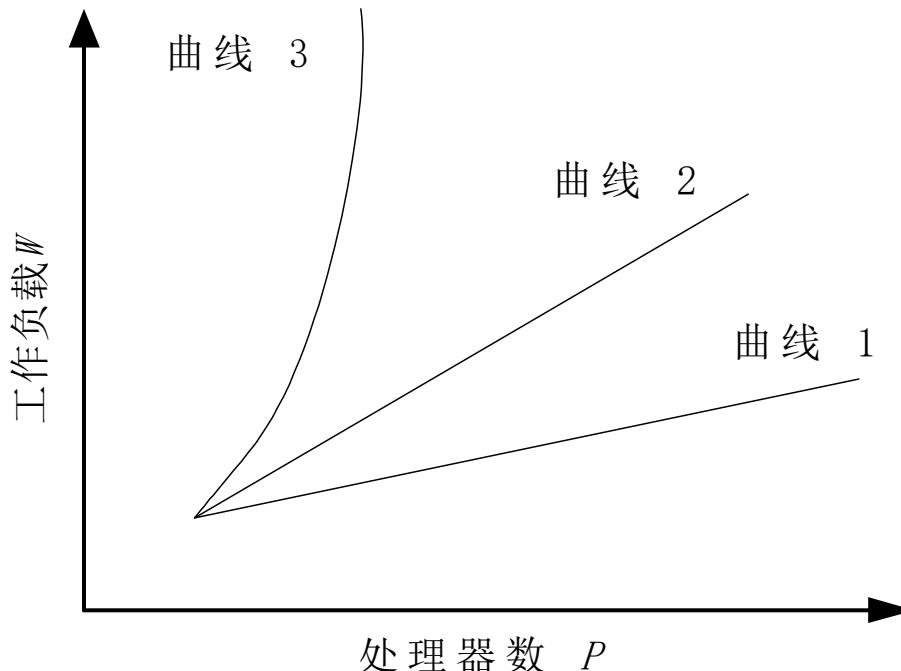
- T_p 是 p 个处理器系统上并行算法的运行时间，对于任意 i 显然有 $T_p = t_e^i + t_o^i$ ，且 $T_e + T_o = pT_p$
- 问题的规模 W 为最佳串行算法所完成的计算量 $W = T_e$

$$S = \frac{T_e}{T_p} = \frac{T_e}{\frac{T_e + T_o}{p}} = \frac{p}{1 + \frac{T_o}{T_e}} = \frac{p}{1 + \frac{T_o}{W}} \quad E = \frac{S}{P} = \frac{1}{1 + \frac{T_o}{T_e}} = \frac{1}{1 + \frac{T_o}{W}}$$

- 如果问题规模 W 保持不变，处理器数 p 增加，开销 T_o 增大，效率 E 下降。为了**维持一定的效率 E** (0与1之间)，当处理器数 p 增大时，需要相应地增大问题规模 W 的值。由此定义函数 $f_E(p)$ 为问题规模 W 随处理器数 p 变化的函数，为**等效率函数 (iso-efficiency function)** (Kumar1987)

等效率度量标准

- 曲线 1 表示算法具有很好的扩展性；曲线 2 表示算法是可扩展的；曲线 3 表示算法是不可扩展的
- **优点：**简单可定量计算的、少量的参数计算等效率函数
- **缺点：**可能 T_o 无法计算出



等效率度量标准

- T_o 可能难以计算得到
 - 例如：非局部访问的读写时间、进程调度时间、存储竞争时间以及高速缓存一致性操作时间等都是难以准确计算的
- 因此，使用解析计算的方法来度量可扩展性不应被视为唯一的方法
- Sun Xianhe 和 Zhang Xiaodong 两位学者于 1994 年分别提出了以试验测试为主要手段的两种评测可扩展性的标准
 - 等速度度量标准 (iso-speed)
 - 平均延迟度量标准 (average latency)

有关可扩展性评测标准的讨论

- 等效率度量标准
 - 在**保持效率 E 不变**的前提下，研究问题规模 W 如何随处理器个数 p 而变化
- 等速度度量标准
 - 在**保持平均速度不变**的前提下，研究处理器个数 p 增多时应该相应的增加多少工作量 W
- 平均延迟度量标准
 - 在**保持效率 E 不变**的前提下，用**平均延迟的比值**来标志随处理器个数 p 增加需要增加的工作量 W

三种度量可扩展性的标准是彼此等效的

有关可扩展性评测标准的讨论

- 三种评价可扩展性的标准的基本出发点都是抓住了影响算法可扩展性的基本参数 T_o , 区别在于:
 - 等效率指标是采用解析计算的方法得到 T_o
 - 等速度指标是将 T_o 隐含在所测量的执行时间中
 - 而平均延迟指标则是保持效率为恒值时, 通过调节 W 与 p 来测量并行和串行执行时间, 最终通过平均延迟反映出 T_o

可扩展性研究的主要目的

- (1) 探索算法和并行架构的组合
确定解决某类问题使用何种并行算法和并行架构的组合，更有利于利用大量的硬件资源
- (2) 预测算法性能
根据某个算法在特定架构下的小规模处理器上的性能，预测该算法移植到较大规模的处理器上后算法的运行性能
- (3) 计算最大加速比
探索在固定的问题规模下，确定利用的处理器数量及其能获得的最大加速比
- (4) 指导算法改进和并行架构设计
根据拓展性指标，指导开发、研究人员改进并行算法或者系统架构，以提高处理器的利用效率

HPC架构的关键特性

- **速度**
 - 系统组件的速度 (如处理器的时钟频率)
- **并行性**
 - 系统的并发度，同时能执行的操作数
- **效率**
 - 系统的利用率
 - 常用度量：持续浮点性能与理论峰值性能的比值

峰值计算性能

- **峰值性能** 指的是超级计算机理论上在其硬件资源的支持下能够获得的最大性能值
- 峰值性能分为 理论峰值性能 和 实测峰值性能
 - 其中，理论浮点峰值性能是指计算机理论上能达到的每秒钟完成浮点计算最大次数，它由 CPU 的主频、CPU 每个时钟周期执行浮点运算的次数和系统总 CPU 核数共同决定
 - 理论峰值性能 = CPU 时钟频率 × CPU 每个时钟周期执行的浮点运算次数 × CPU 数量

峰值计算性能

- 理论峰值性能适用于通用的硬件体系结构，但忽略了内存架构及带宽、I/O 性能、缓存一致性等因素的影响，从而无法真正反映系统的真实峰值性能
- 为此许多组织试图用一些标准的程序来测试计算机的运算速度，用于评价计算机的实际峰值运算能力

HPC架构的关键特性

- **功率**
 - 处理器主频与功率相关
 - 冷却系统
- **可靠性**
 - 硬件和软件故障
 - Checkpoint机制
- **可编程性**
 - 涉及系统的易用性和生产率

(二) 基准评测集

基准评测介绍

- 性能基准评测就是从基准测试程序和测试规范的角度评价和预测系统的性能
- 基准评测：
 - 可以帮助机构确定所需采购的超级计算机
 - 可以指导制造商高性能计算机系统的设计方向
 - 是探索 HPC 趋势的重要历史记录

Benchmarks

- The best choice of benchmarks to measure performance is **real applications**
 - Examples:
 - **Kernels**: small, key pieces of real applications. (e.g., matrix multiply)
 - **Toy programs**: 100-line programs from beginning programming assignments (e.g., Quicksort)
 - **Synthetic benchmarks**: fake programs invented to try to match the profile and behavior of real applications (e.g., Dhrystone)
 - **benchmark suites**: collections of benchmark applications (e.g., SPEC06fp, TPC-C)
- Discredited today!*

商用基准测试程序

商用基准测试程序

- **Transaction-processing (TP) benchmarks**
 - measure the ability of a system to **handle transactions** that consist of **database accesses and updates**
- **Transaction Processing Council (TPC)**: create realistic and fair benchmarks for TP
 - The first TPC benchmark, **TPC-A**, was published in 1985
 - **TPC-C simulates a complex query environment**
 - **TPC-H** models ad hoc decision support
 - **TPC-DI** benchmark, a new data integration (DI) task
 - **TPC-E** is an online transaction processing (OLTP) workload that simulates a brokerage firm's customer accounts

商用基准测试程序

- **TPC** (Transaction Processing Performance Council)

- TPC-A和TPC-B于1995年6月废弃
 - TPC-C是目前最流行的在线事务处理 (On Line Transcation Processing, OLTP) 商用基准测试程序，能模拟一个大公司的整个销售环境 (仓库、区域、用户、定单、各种事务的比例、响应时间)，TPC-C用于测试事务处理系统的性能与价格/性能比；
 - 性能结果 tpmC：描述了系统在执行Payment、Order-status、Delivery、Stock-Level这四种交易的同时，每分钟可以处理多少个New-Order交易
 - 所有交易的响应时间必须满足TPC-C测试规范的要求
 - 流量指标值越大越好
 - 和价格/性能结果 \$ /tpmC：结果越小越好

商用基准测试程序

- **TPC** (*Transaction Processing Performance Council*)
 - 各厂商的TPC-C测试结果都按TPC组织规定的两种形式发布：
 - 测试结果概要 (*Executive Summary*)
 - 测试结果概要中描述了主要的测试指标、测试环境示意图以及完整的系统配置与报价
 - 详细测试报告 (*Full Disclosure Report*)
 - 详细测试报告中除了包含上述内容外，还详细说明了整个测试环境的设置与测试过程
 - TPC-C允许被测系统放大和缩小，但终端数和数据库规模也必须按比例调整

SPEC 测试程序

SPEC和SPEC CPU测试程序

- **SPEC是由计算机厂商、系统集成商、大学、研究机构、咨询等多家公司组成的非营利性组织**
 - System Performance Evaluation Cooperative (网址: www.spec.org)
 - 随CPU性能提高已发展了6轮: 1989、1992、1995、2000、2006、2017
 - SPEC CPU2000: 12个定点程序, 14个浮点程序
 - SPEC CPU2006: 12个定点程序, 17个浮点程序
 - SPEC CPU2017: 10个定点, 14个浮点程序, 分为Rate (多道程序) 和 Speed (多线程)
- **SPEC CPU系列测试程序集合, 主要关注CPU的性能**
 - SPEC CPU中的SPECratio (单进程) 分值主要反映PC系统性能
 - 衡量服务器的吞吐率通常采用SPECrate, 在多CPU执行SPEC CPU的多份拷贝, 并且把CPU的时间转换为比率 (rate), 即SPECrate

SPEC CPU2000测试程序套件

Benchmark	Type	Source	Description
gzip	Integer	C	Compression using the Lempel-Ziv algorithm
vpr	Integer	C	FPGA circuit placement and routing
gcc	Integer	C	Consists of the GNU C compiler generating optimized machine code.
mcf	Integer	C	Combinatorial optimization of public transit scheduling.
crafty	Integer	C	Chess playing program.
parser	Integer	C	Syntactic English language parser
eon	Integer	C++	Graphics visualization using probabilistic ray tracing
perlmbk	Integer	C	Perl (an interpreted string processing language) with four input scripts
gap	Integer	C	A group theory application package
vortex	Integer	C	An object-oriented database system
bzip2	Integer	C	A block sorting compression algorithm.
twolf	Integer	C	Timberwolf: a simulated annealing algorithm for VLSI place and route
wupwise	FP	F77	Lattice gauge theory model of quantum chromodynamics.
swim	FP	F77	Solves shallow water equations using finite difference equations.
mgrid	FP	F77	Multigrid solver over 3-dimensional field.
apply	FP	F77	Parabolic and elliptic partial differential equation solver
mesa	FP	C	Three dimensional graphics library.
galgel	FP	F90	Computational fluid dynamics.
art	FP	C	Image recognition of a thermal image using neural networks
equake	FP	C	Simulation of seismic wave propagation.
facerec	FP	C	Face recognition using wavelets and graph matching.
ammp	FP	C	molecular dynamics simulation of a protein in water
lucas	FP	F90	Performs primality testing for Mersenne primes
fma3d	FP	F90	Finite element modeling of crash simulation
sixtrack	FP	F77	High energy physics accelerator design simulation.
apsi	FP	F77	A meteorological simulation of pollution distribution.

SPEC CPU2006测试程序套件

SPEC CPU2000 Integer Benchmarks		
400.perlbench	C	PERL Programming Language
401.bzip2	C	Compression
403.gcc	C	C Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: go
456.hmmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: chess
462.libquantum	C	Physics: Quantum Computing
464.h264ref	C	Video Compression
471.omnetpp	C++	Discrete Event Simulation
473.astar	C++	Path-finding Algorithms
483.xalancbmk	C++	XML Processing

SPEC CPU2000 Floating Point Benchmarks		
410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics: Quantum Chromodynamics
434.zeusmp	Fortran	Physics / CFD
435.gromacs	C/Fortran	Biochemistry/Molecular Dynamics
436.cactusADM	C/Fortran	Physics / General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.namd	C++	Biology / Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-tracing
454.calculix	C/Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.1bm	C	Fluid Dynamics
481.wrf	C/Fortran	Weather Prediction
482.sphinx3	C	Speech recognition

SPEC CPUint 2017 测试程序套件

SPECrate 2017 定点程序	SPECspeed 2017 定点程序	语言	描述
500.perlbench_r	600.perlbench_s	C	文本处理, Perl解释器
502.gcc_r	602.gcc_s	C	编译, GNU C编译器
505.mcf_r	605.mcf_s	C	组合和优化, 求解车辆调度问题
520.omnetpp_r	620.omnetpp_s	C++	计算机网络, 离散事件模拟
523.xalancbmk_r	623.xalancbmk_s	C++	通过XSLT将XML转换为HTML
525.x264_r	625.x264_s	C	视频压缩, x264视频编解码
531.deepsjeng_r	631.deepsjeng_s	C++	人工智能, 下棋程序, alpha-beta树搜索
541.leela_r	641.leela_s	C++	人工智能, 下棋程序(go), 蒙特卡洛树搜索
548.exchange2_r	648.exchange2_s	Fortran	人工智能, 9x9数独, 递归方式求解
557.xz_r	657.xz_s	C	压缩和解压缩, xz压缩程序

两个子测试集:

rate: 多个拷贝多核运行

speed: 单个拷贝, 但代码针对openMP自动并行化做了修改

SPEC CPUfp 2017测试程序套件

SPECCrate 2017 浮点程序	SPECspeed 2017 浮点程序	语言	描述
503.bwaves r	603.bwaves s	Fortran	计算流体动力学, 爆炸建模
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	广义相对论和数值相对论, 求解真空中的爱因斯坦方程
508.namd r		C++	结构生物学, 模拟大规模的生物分子系统
510.parest r		C++	分子医学成像, 光学层析成像问题的有限元求解器
511.povray r		C++, C	计算机可视化, 光线追踪应用POV-Ray
519.lbm r	619.lbm s	C	流体动力学
521.wrf r	621.wrf s	Fortran, C	天气预报建模, 基于新一代中尺度数值天气预报系统WRF
526.blender r		C++, C	三维渲染和动画, 基于开源的三维制作套件Blender
527.cam4 r	627.cam4 s	Fortran, C	大气环流建模, 地球系统模型CESM中的大气建模部分
	628.pop2_s	Fortran, C	大规模海洋建模 (气候层面), 地球系统模型CESM中的海洋建模部分
538.imagick r	638.imagick s	C	图像处理软件包ImageMagick中convert部分
544.nab_r	644.nab_s	C	分子动力学, 基于生命科学计算领域中分子建模应用NAB (核酸构建器)
549.fotonik3d_r	649.fotonik3d_s	Fortran	计算电磁学, 利用时域有限差分方法计算光子波导的透射系数
554.roms r	654.roms s	Fortran	区域海洋建模, 基于区域海洋建模系统ROMS

Summarizing Performance Results

- **Summarize performance results of the suite in a unique number**
 - (1) arithmetic means of execution times of programs in the suite
 - (2) the weighted arithmetic mean
 - (3) normalize execution times to a reference computer
- **SPECRatio**
 - A ratio by dividing the time on the reference computer by the time on the computer being rated
 - For example, suppose that the SPECRatio of computer A on a benchmark is 1.25 times as fast as computer B

$$1.25 = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_A}}{\frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_B}} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

SPEC CPU分值计算

- SPECratio是被测机器执行时间除以参考机器执行时间
 - $\text{SPECratio}_A = \text{Execution time}_{\text{reference}} / \text{Execution time}_A$
- 综合测试结果是取SPECratio的几何平均值

Benchmarks	Ultra 5 time (sec)	Opteron time (sec)	SPECRatio	Itanium 2 time (sec)	SPECRatio	Opteron/Itanium times (sec)	Itanium/Opteron SPECRatios
wupwise	1600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16	2.16
art	2600	92.4	28.13	21.0	123.67	4.40	4.40
quake	1300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric mean			20.86		27.12	1.30	1.30

SPEC CPUint2006的IPC值

- Intel Xeon (2.66GHz) 的 SPEC CPU2006 定点分值
 - IPC 在 0.38 到 2 之间 (5个小于1.0, 四个大于 1.5)
 - SPEC CPU 分值是一个相对比值, IPC 高分值不一定高

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	-	-	-	-	-	-	25.7

龙芯 3A5000 处理器性能分析实例 (SPEC CPU数据)

性能分析和测试实例

- 分析基于 GS464V 的龙芯 3A5000 处理器的性能特性，并和 AMD 的 Zen1 以及 Intel 的 Skylake 处理器进行对比
 - 3A5000: GS464V, 主频 2.5 GHz, 内存 DDR4 3200, 16MB LLC
 - 比较时其它处理器主频均设置为 2.5 GHz, 内存为 DDR4 3200 或者 2400

基本参数和微结构参数

厂商	Loongson	AMD	Intel
处理器型号	3A5000	r3 1200	i3 9100f
上市时间	2021	2017	2019
工艺	12nm	14nm	14nm
指令集	LoongArch	X86-64	X86-64
微结构型号	GS464V	Zen1	SkyLake
频率	2.5GHz	2.5GHz	2.5GHz
内存类型和频率	DDR4 3200MHz	DDR4 3200MHz	DDR4 2400MHz
Cache层次	64KB L1 ICache 64KB L1 DCache 256KB L2 Cache 16MB L3 Cache	64KB L1 ICache 32KB L1 DCache 512KB L2 Cache 8MB L3 Cache	32KB L1 ICache 32KB L1 DCache 256KB L2 Cache 6MB L3 Cache
核心队列和重命名寄存器数量	128项ROB, 64项load队列, 48项store队列, 32项分支队列, 32项定点、32项浮点和32项访存保留站, 128项定点和128项浮点重命名寄存器	192项ROB, 72项load队列, 44项store队列, 84项定点和96项浮点保留站, 168项定点和160项浮点重命名寄存器	224项ROB, 72项load队列, 56项store队列, 97项统一保留站, 180项定点和168项浮点重命名寄存器
功能部件数	4个定点, 2个访存部件, 2个256位的浮点乘加点	4个定点部件, 2个访存部件, 4个128位浮点部件 (其中2个FMA/FMUL, 2个FADD)	4个定点、3个访存、3个256位浮点乘加

SPEC CPU2006的分值对比

- 3A5000和Zen1、Skylake的SPEC CPU2006分值对比

SPEC CPU2006	loongson 3A5000	zen1 r3- 1200	skylake i3 9100f
400.perlbench	29.1	31.4	35.3
401.bzip2	17.3	18.5	19.2
403.gcc	23.9	30.8	38.1
429.mcf	27.1	27.6	37
445.gobmk	25.6	20.1	21.7
456.hmmr	39.2	46.2	54.7
458.sjeng	22.4	17.8	22.5
462.libquantum	78.8	141	123
464.h264ref	38	45.1	51
471.omnetpp	18.1	17.2	21.2
473.astar	19.3	15.6	15.8
483.xalancbmk	28.7	25.9	36.9
SPECint2006	27.87	29.05	33.47

SPEC CPU2006	loongson 3A5000	zen1 r3- 1200	skylake i3 9100f
410.bwaves	54.4	99.1	85.2
416.gamess	22.9	28.4	30.2
433.mile	18	38.8	32.3
434.zeusmp	25	51	56.2
435.gromacs	15.3	25.8	23.5
436.cactusADM	84.8	135	228
437.leslie3d	36.2	48.2	62.8
444.namd	20.7	21.5	23.8
447.dealII	39.8	48.7	46.7
450.soplex	28.5	34.6	39.8
453.povray	39.1	34.2	41.5
454.calculix	17.6	29.2	30.1
459.GemsFDTD	35.2	65.1	56.7
465.tonto	28.4	25.2	37.2
470.lbm	28.8	67.5	74.6
481.wrf	36.5	53.9	65
482.sphinx3	34.1	40.6	48.4
SPECfp2006	30.29	43.85	48.34

SPEC CPU测试数据的初步分析

- 通过SPEC CPU分值了解整体的性能概况
 - 3A5000和Zen1同频定点性能基本相当（差4%），浮点差距比较明显(约44%)
 - skylake整体表现最好
 - 结合微结构参数，可推测浮点的性能差距与功能部件数和队列大小有关系
- 通过分析具体程序的行为寻找优化方向
 - 不同程序性能对比有较大差异
 - SPEC 2006中462.libquantum, Zen1比3A5000快78.9%：经分析和自动向量化有关，关闭自动向量化差距缩小到16.5%，提示3A5000的编译器自动向量化有待改进
 - 深入分析应结合微结构参数和进一步的细节数据进行，可通过仿真平台验证推测

动态指令数目对比(SPECint2006)

- 456.hmmer 3A5000指令数显著多于另外两款处理器，其性能差距也相对较大

SPEC CPU2006	loongson 3A5000	zen1 r3-1200	Skylake i3 9100f	zen1/3A5000	skylake/3A5000
400.perlbench	2,120,668	1,893,315	1,875,825	89.3%	88.5%
401.bzip2	2,598,456	2,230,210	2,105,917	85.8%	81.0%
403.gcc	918,503	720,416	637,262	78.4%	69.4%
429.mcf	270,116	273,830	279,889	101.4%	103.6%
445.gobmk	1,447,758	1,364,925	1,349,346	94.3%	93.2%
456.hmmer	1,725,941	1,293,236	1,140,944	74.9%	66.1%
458.sjeng	2,448,077	2,256,035	2,259,544	92.2%	92.3%
462.libquantum	1,287,345	1,354,349	1,355,304	105.2%	105.3%
464.h264ref	3,906,022	3,057,443	2,574,503	78.3%	65.9%
471.omnetpp	455,925	515,436	535,127	113.1%	117.4%
473.astar	804,288	862,191	855,804	107.2%	106.4%
483.xalancbmk	845,446	857,273	858,951	101.4%	101.6%
SPECint_2006	18,828,544	16,678,659	15,828,415	88.6%	84.1%

动态指令数目对比 (SPECfp2006)

- 436.cactusADM skylake指令数只有另两款不到一半
 - 它自动采用256位向量，Zen采用128位

SPECint_2006	18,828,544	16,678,659	15,828,415	88.6%	84.1%
410.bwaves	867,049	643,946	563,805	74.3%	65.0%
416.gamess	6,326,795	4,818,900	4,737,215	76.2%	74.9%
433.milc	495,564	573,026	562,614	115.6%	113.5%
434.zeusmp	1,255,758	901,519	682,460	71.8%	54.3%
435.gromacs	1,523,919	1,606,588	1,621,933	105.4%	106.4%
436.cactusADM	1,059,689	913,079	460,976	86.2%	43.5%
437.leslie3d	586,968	663,348	350,522	113.0%	59.7%
444.namd	1,602,845	1,630,103	1,627,678	101.7%	101.5%
447.dealII	1,078,354	1,048,560	1,029,969	97.2%	95.5%
450.soplex	588,538	562,087	535,320	95.5%	91.0%
453.povray	715,602	816,008	838,778	114.0%	117.2%
454.calculix	1,941,778	1,615,746	1,473,997	83.2%	75.9%
459.GemsFDTD	773,507	832,552	471,723	107.6%	61.0%
465.tonto	2,028,792	2,212,799	1,674,174	109.1%	82.5%
470.lbm	877,312	817,999	821,754	93.2%	93.7%
481.wrf	1,437,760	953,405	794,095	66.3%	55.2%
482.sphinx3	2,229,029	1,850,550	1,867,714	83.0%	83.8%
SPECfp_2006	25,389,259	22,460,213	20,114,726	88.5%	79.2%

IPC的对比 (SPEC CPU2006)

- 3A5000定点IPC略高于Zen1，略低于Skylake；浮点低于另两者

SPEC CPU2006	loongson 3A5000	zen1 r3-1200	skylake i3 9100f	zen1/3A50 00	skylake/3A5 000
400.perlbench	2.54	2.45	2.72	96.5%	107.1%
401.bzip2	1.86	1.73	1.68	93.0%	90.3%
403.gcc	1.1	1.1	1.22	100.0%	110.9%
429.mcf	0.32	0.33	0.45	103.1%	140.6%
445.gobmk	1.42	1.05	1.12	73.9%	78.9%
456.hmmer	2.97	2.57	2.68	86.5%	90.2%
458.sjeng	1.81	1.33	1.68	73.5%	92.8%
462.libquantum	1.42	1.67	1.88	117.6%	132.4%
464.h264ref	2.69	2.5	2.38	92.9%	88.5%
471.omnetpp	0.54	0.57	0.73	105.6%	135.2%
473.astar	0.89	0.79	0.77	88.8%	86.5%
483.xalancbmk	1.4	1.3	1.84	92.9%	131.4%
SPECint_rate 2006	1.34	1.24	1.40	92.9%	105.0%

410.bwaves	1.42	1.9	1.41	133.8%	99.3%
416.gamess	2.97	2.82	2.92	94.9%	98.3%
433.milc	0.4	0.97	0.81	242.5%	202.5%
434.zeusmp	1.39	2.01	1.68	144.6%	120.9%
435.gromacs	1.31	2.32	2.13	177.1%	162.6%
436.cactusADM	1.79	1.72	1.44	96.1%	80.4%
437.leslie3d	0.91	1.38	0.93	151.6%	102.2%
444.namd	1.66	1.75	1.94	105.4%	116.9%
447.dealII	1.52	1.79	1.7	117.8%	111.8%
450.soplex	0.84	0.94	1.02	111.9%	121.4%
453.povray	2.06	2.12	2.6	102.9%	126.2%
454.calculix	1.66	2.3	2.16	138.6%	130.1%
459.GemsFDTD	0.88	1.27	0.82	144.3%	93.2%
465.tonto	2.33	2.27	2.53	97.4%	108.6%
470.lbm	0.74	1.62	1.79	218.9%	241.9%
481.wrf	1.67	1.95	1.73	116.8%	103.6%
482.sphinx3	1.52	1.55	1.85	102.0%	121.7%
SPECfp_rate2006	1.34	1.73	1.62	129.7%	121.0%

分支误预测率对比(SPEC CPU2006)

- 整体分支误预测率均已较低，3A5000还存在一定提升空间

SPEC CPU2006	loongson 3A5000	zen1 r3- 1200	skylake i3 9100f	zen1/3A5000	skylake/3A5000	410.bwaves	0.09%	0.25%	0.15%	2.78	1.67
400.perlbench	1.18%	1.10%	0.71%	0.93	0.60	416.gamess	0.94%	1.07%	0.69%	1.14	0.73
401.bzip2	5.38%	5.08%	4.94%	0.94	0.92	433.milc	6.58%	0.40%	0.23%	0.06	0.03
403.gcc	1.52%	1.31%	1.03%	0.86	0.68	434.zeusmp	1.38%	0.95%	0.12%	0.69	0.09
429.mcf	5.62%	3.57%	3.86%	0.64	0.69	435.gromacs	7.04%	6.19%	6.11%	0.88	0.87
445.gobmk	8.29%	9.80%	8.56%	1.18	1.03	436.cactusAD M	0.33%	1.49%	0.17%	4.52	0.52
456.hmmmer	1.12%	0.66%	0.65%	0.59	0.58	437.leslie3d	0.33%	1.66%	0.22%	5.03	0.67
458.sjeng	4.47%	5.89%	4.17%	1.32	0.93	444.namd	4.66%	4.38%	4.52%	0.94	0.97
462.libquantum	0.79%	0.21%	0.09%	0.27	0.11	447.dealII	2.48%	2.31%	2.06%	0.93	0.83
464.h264ref	2.09%	1.64%	1.69%	0.78	0.81	450.soplex	5.51%	4.22%	4.40%	0.77	0.80
471.omnetpp	2.53%	1.65%	1.83%	0.65	0.72	453.povray	1.86%	1.30%	0.56%	0.70	0.30
473.astar	13.60%	12.03%	12.75%	0.88	0.94	454.calculix	3.23%	2.80%	3.02%	0.87	0.93
483.xalancbmk	0.43%	0.51%	0.34%	1.19	0.79	459.GemsFDT D	0.29%	0.40%	0.10%	1.38	0.34
SPECint_2006	2.48%	1.97%	1.64%	0.79	0.66	465.tonto	1.20%	0.93%	0.91%	0.78	0.76
						470.lbm	0.46%	0.45%	0.38%	0.98	0.83
						481.wrf	1.08%	0.51%	0.24%	0.47	0.22
						482.sphinx3	2.35%	1.80%	1.88%	0.77	0.80
						SPECfp_2006	1.30%	1.24%	0.65%	0.95	0.50

STREAM 带宽的对比

3A5000和Zen1、Skylake的STREAM带宽对比 (MB/s)

STREAM四核 (openMP)	3A5000	Zen1	Skylake
Copy	23860.9	39896.4	26983.3
Scale	22347.3	25073.3	19110.9
Add	19323.0	29768.5	21516.0
Triad	21043.8	29146.8	21490.7

功能部件的操作延迟

- 相关数据需要进一步分析

lat_ops 延迟 (ns)	loongson 3A5000	zen1 r3-1200	skylake i3 9100f
integer bit	0.27	0.29	0.27
integer add	0	0	0
integer mul	0.02	0.02	1.24
integer div	6.21	8.03	10.86
integer mod	3.60	5.61	11.32
int64 bit	0.27	0.29	0.27
int64 add	0	0	0
int64 mul	0.02	0.02	1.22
int64 div	7.60	10.63	17.10
int64 mod	3.34	5.34	16.55
float add	2.00	1.20	1.60
float mul	2.00	1.20	1.61
float div	10.82	4.04	4.58
double add	2.00	1.20	1.60
double mul	2.00	1.60	1.60
double div	9.22	5.25	5.78
float bogomflops	5.04	1.53	1.20
double bogomflops	9.62	1.81	1.61

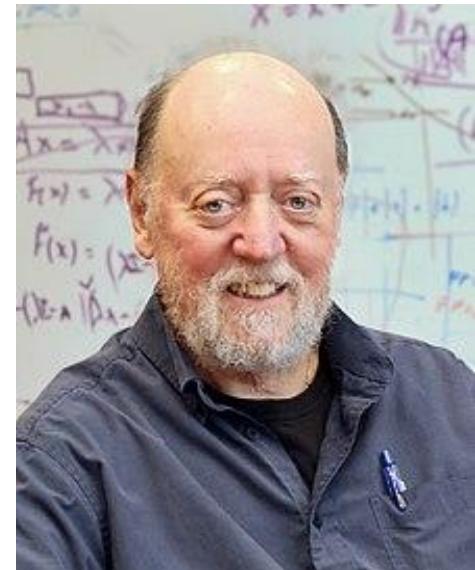
HPC 基准评测集

常见HPC基准评测分类

- **计算性能评测集：** Linpack, HPCG, Graph500
- **IO性能评测集：** MDTest, IOR, IO500
- **网络性能评测集：** IMB, OSU Benchmark
- **能耗评测集：** Green 500
- **应用评测集：** Miniapplication、戈登·贝尔奖

Linpack

- 杰克·唐加拉 (Jack Dongarra, 1950年7月18日 -) 美国计算机科学家，田纳西大学电气工程与计算机科学系特聘教授，美国国家工程院院士，英国皇家学会外籍院士
- 2021年，杰克·唐加拉因开创性的概念和方法获得了**图灵奖**，这些概念和方法导致了改变世界的计算。他的算法和软件被认为推动了高性能计算的发展，并在人工智能的计算科学到计算机图形学的许多领域产生了重大影响



杰克·唐加拉
(Jack Dongarra)

Linpack

- Linpack 是度图灵奖得主美国计算机科学家杰克·唐加拉 (Jack Dongarra) 等几位学者于 1979 年开发的线性系统软件库 (Linear system package, Linpack)
- Linpack 设计之初的目标是集成常用的线性方程组求解程序 (例如最小平方问题、稠密矩阵运算) 至一个通用的线性代数软件库，支持求解大部分常见的线性代数问题
- Linpack 评测基准在 HPC 计算性能评测领域发挥了强大的作用

Linpack

- 用来度量系统的浮点计算能力
- Linpack性能是指求解双精度线性代数方程组时所达到的实际性能
 - 包括Linpack100, Linpack1000, HPL
 - Linpack100 和 Linpack1000 分别求解规模为 100 阶和 1000 阶的线性代数方程组
 - HPL是针对现代的高性能计算机的评测基准
- 评测指标：每秒浮点运算次数（flops）

Linpack100

- Linpack100 使用 Fortran 中的 Linpack 软件求解规模为 100 阶的稠密线性代数方程组
- 为了运行该程序，需要提供一个称为 SECOND 的计时函数，报告已经过去的 CPU 时间
- 运行此基准评测的基本规则是不能对 Fortran 代码进行任何更改，甚至不允许更改注释，只允许采用编译优化选项进行优化

Linpack1000

- Linpack1000 要求求解规模为 1000 阶的线性代数方程组，在达到指定的精度要求下，可以在不改变计算量的前提下做算法和代码上做优化
- 运行此基准评测的基本规则要宽松一些，可以指定希望求解的任何线性方程，用任何语言实现
- 必须满足要求是必须计算出一个解决方案，并且该解决方案必须返回一个达到规定精度的结果

HPL

- HPL 具体为求一个 n 维的线性方程组的解：

$$\mathbf{A}\mathbf{x} = \mathbf{b}; \mathbf{A} \in \mathbb{R}^{n \times n}; \mathbf{x}, \mathbf{b} \in \mathbb{R}^n$$

- 对 $n \times (n+1)$ 的系数矩阵 $[\mathbf{A} \ \mathbf{b}]$ 进行LU分解：

$$[\mathbf{A}, \mathbf{b}] = [[\mathbf{L}\mathbf{U}], \mathbf{y}]$$

- 方程组的解 \mathbf{x} 转化为求解三角矩阵 \mathbf{U} 作为系数矩阵的线性方程组 $\mathbf{U}\mathbf{x} = \mathbf{y}$ 而得到

HPL 是国际超级计算系统 TOP500 的重要依据

HPL

- HPL 测试的目标是尽可能地使得高性能计算机对矩阵 A 的不同部分并行地做出更新，这就需要对矩阵进行分块
 - N 为最高 GFLOPS 值的矩阵规模、 N_B 为求解矩阵的分块大小
 - P 为处理器网格中水平方向处理器个数、 Q 为处理器网格中垂直方向处理器个数

A ₀₀	A ₀₁	A ₀₂	A ₀₃
A ₁₀	A ₁₁	A ₁₂	A ₁₃
A ₂₀	A ₂₁	A ₂₂	A ₂₃
A ₃₀	A ₃₁	A ₃₂	A ₃₃

(a) 数据矩阵图

A ₀₀	A ₀₂	A ₀₁	A ₀₃
A ₂₀	A ₂₂	A ₂₁	A ₂₃
A ₁₀	A ₁₂	A ₁₁	A ₁₃
A ₃₀	A ₃₂	A ₃₁	A ₃₃

(b) 矩阵分块在进程上的分布图分布图

如图(a)所示，矩阵 A 首先划分为 $N_B = 4 \times 4$ 个数据块，之后所有 N_B 个数据块映射到数目为 $P \times Q = 2 \times 2$ 的进程阵，如图(b)所示

Top500排行榜 (2024年6月)

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	Eagle - Microsoft NDV5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

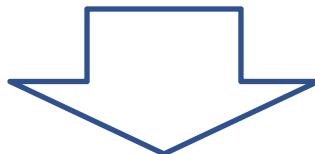
HPL 是国际超级计算系统 TOP500 的重要依据

LINPACK

- Linpack 用户手册可以从 <http://www.netlib.org/lapack/lug/> 下载得到
- Linpack100 测试程序可以从 <http://www.netlib.org/benchmark/linpackd> 下载得到
- Linpack1000 测试程序可以从 <http://www.netlib.org/benchmark/1000d> 下载得到
- HPL 采用 C 语言进行编写的测试程序可以从 <http://www.netlib.org/benchmark/hpl> 下载得到

HPCG

- 实际应用中存在大量用**偏微分方程建模**，稀疏计算和不规则的访存模式，无法用Linpack衡量



- **HPCG(高性能共轭梯度法)**
 - 作为Linpack的补充，是**求解稀疏矩阵方程组**的一种迭代算法

HPCG

- HPCG (高性能共轭梯度, High-Performance Conjugate Gradient) 基准测试程序于 2000 年首次发布, 代码采用 C++ 语言编写
- HPCG 首次发布于 2013 年 11 月举办的世界超算会议 (SC 2013)
- HPCG 基准评测方法目的是适应目前大部分应用程序所采用的计算和数据访问模式, 最终促进计算机性能的整体提升

HPCG

- HPCG模拟三维热力学运动问题，从而转化为求解离散的三维偏微分方程模型问题
 - 使用局部对称高斯·塞德尔 (Local Symmetric Gauss-Seidel, LSGS) 预条件子的预处理共轭梯度法
 - 主要数据为对称正定稀疏矩阵

三维热力学运动问题转化

- **三维热力学运动问题转化为求解离散的三维偏微分方程模型问题步骤：**
 - 网格划分：将热力学问题转化为离散的三维空间网格。根据实际情况，可以选择均匀网格或非均匀网格
 - 离散化：根据具体的热力学方程，将偏微分方程转化为差分方程。例如，可以使用有限差分法或有限元法来离散化方程
 - 边界条件：根据实际问题，在离散的网格上设置正确的边界条件。这些条件用于描述问题的物理特性
 - 代数方程组：将离散化的方程整理为矩阵形式，并构建一个稀疏线性方程组。方程组的未知数通常是网格点上的温度或其他物理量
 - 解方程组：使用HPCG或其他合适的求解器，通过迭代方法（如共轭梯度法）求解离散方程组。这涉及到矩阵向量乘法和共轭梯度法的实现
 - 并行计算：为了提高性能，利用并行计算技术（如MPI或OpenMP）将计算任务分配给多个处理器或多个计算节点

LSGS 预条件子

- 局部对称高斯-塞德尔 (LSGS) 预条件子是一种用于预处理共轭梯度法 (Conjugate Gradient, CG) 的方法。该方法通过将原始线性系统的系数矩阵分解为局部块矩阵，并使用高斯-塞德尔迭代来逐块求解。LSGS预条件子在求解稀疏线性系统时能够有效地加速CG方法的收敛速度。
- 以下是LSGS预处理共轭梯度法的基本步骤：
 - 定义原始线性系统：将原始线性系统表示为 $Ax=b$ 的形式，其中A是对称正定的稀疏矩阵， x 是解向量， b 是右侧向量。
 - 分解系数矩阵：将系数矩阵A分解为块对角矩阵形式。例如，将A分解为D - L - U的形式，其中D是主对角线矩阵，L是严格下三角矩阵，U是严格上三角矩阵。
 - 进行预处理：采用高斯-塞德尔迭代来对分解后的系数矩阵进行预处理。高斯-塞德尔迭代通过使用前一步求解得到的分量来更新当前分量，直到达到所需的精度。对于块矩阵来说，每个块的更新是独立进行的。
 - 执行共轭梯度法迭代：使用预处理后的分解系数矩阵（块对角矩阵）和预处理后的右侧向量进行共轭梯度法迭代求解。
- LSGS预条件子的主要优点是它能够利用块矩阵的局部性质和高斯-塞德尔迭代的迭代更新来加速共轭梯度法的收敛。然而，LSGS预条件子也有一些限制，如对块矩阵的分解和求解过程的复杂性，以及在某些情况下可能会受到块矩阵形状和特征值分布的影响。

HPCG

- 在HPCG基准测试程序中，稀疏线性方程组是通过配置文件中的参数来构建的。常见的参数设置及其含义：
 - 网格维度：通过nx, ny, nz参数设置三维网格的大小
 - 这决定了方程组中未知数的个数
 - 网格划分：通过px, py, pz参数设置三维网格的划分
 - 这定义了在并行计算中网格划分的方式
 - 通信模式：通过cg_iter, symmetry, overlap, global_mpi等参数设置通信模式
 - 这些参数决定了通信方式、是否引入对称性等
 - 矩阵类型：通过exec_set参数设置矩阵类型，如laplacian或elasticity
 - 这决定了稀疏矩阵的结构
 - 非零元素密度：通过toc参数设置非零元素密度
 - 这决定矩阵的稀疏程度
- 通过这些参数，HPCG基准测试程序会根据矩阵类型和非零元素密度生成一个稀疏线性方程组，其中未知数是网格点上的温度或其他物理量

HPCG

- HPCG基准测试可以提供以下指标：
 - **HPCG得分**: 作为一个综合评估指标, HPCG基准测试提供一个HPCG得分来度量计算机的整体性能
 - 这个得分反映了计算机在求解稀疏线性系统方面的速度和效率
 - **迭代次数**: HPCG基准测试可以提供求解稀疏线性系统所需的迭代次数
 - 通过迭代次数, 可以了解到计算机的计算能力和收敛速度
 - **内存带宽**: HPCG基准测试是一个内存密集型的应用程序, 通过测试可以了解计算机在高速缓存和主存之间的数据传输效率

HPCG

- HPCG 在利用差分方程迭代的过程中，采用了全局通信和邻近进程数据的通信方式以及向量更新、点乘、稀疏矩阵向量乘法和局部三角求解器等计算模式，覆盖了高性能计算领域中常见的计算操作
- 对于未来计算机系统，可以通过**异步通信**以及能够**降低通信延迟**的相关技术来改进 HPCG 的性能

HPCG

- 需要注意的是，HPCG要求：
 - 首先，问题规模所需占用的内存大小至少占系统总内存的四分之一以上
 - 这主要是由于问题规模太小会导致数据访存开销以及访存不规则不能体现
 - 其次，官方还禁止用户改变预处理器的设置，保证系统测试的合理性
 - 禁止包括修改数据在内存中的存储长度等修改数据的存储格式的操作，以及禁止通过修改算法来减少通信开销的操作

TOP500				Rmax (PFlop/s)	HPCG (TFlop/s)
Rank	Rank	System	Cores		
1	4	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	16004.50
2	1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	14054.00
3	2	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel	9,264,128	1,012.00	5612.60
HPCG 弥补了 Linpack 存在的不足，促使计算机系统的设计不仅仅只专注于计算能力的发展，还要向访存、通信这些方面发展					
4	5	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	4586.95
5	6	Alps - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	1,305,600	270.00	3671.32
					104

Graph500

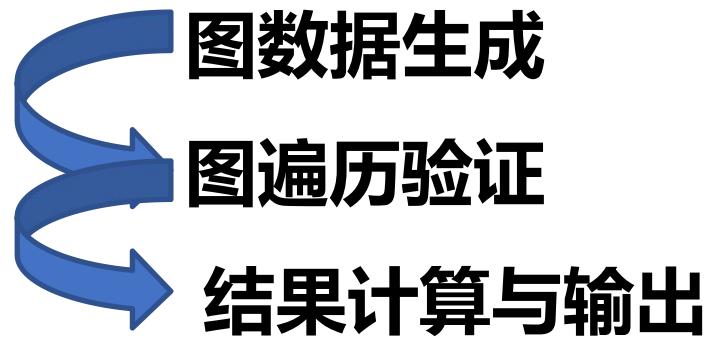
- 美国桑迪亚国家实验室（英语：Sandia National Laboratories, SNL）的理查德·墨菲（Richard Murphy）等多名研究人员在 2010 年举办的国际超算会议上提出了 Graph500 评测基准
- 不同于 Linpack 与 HPCG 基准评测集，Graph500 旨在评测高性能计算机对复杂数据的处理性能，它侧重于系统的通信子系统，而不再专注于计算性能
- 在大数据信息时代下，图遍历是一种典型的数据密集型应用，也是大数据科学的研究重点，Graph500 基准评测是在图遍历的基础上设计出来的



理查德·墨菲
Richard Murphy
美国桑迪亚国家实验室
Sandia National Laboratories

Graph500

- Graph500用来衡量计算机在处理**数据密集型应用**的能力
 - 利用**图遍历**中广度优先搜索（BFS）算法或单源点最短路径（SSSP）算法
- Graph500评测流程：



Graph500

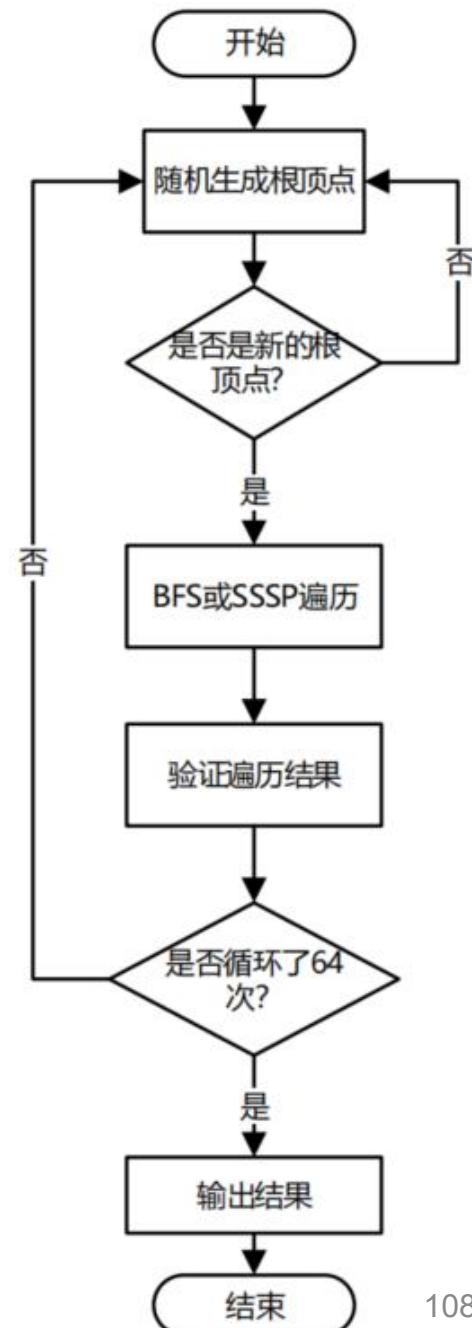
- Graph500 基准评测首先会构建一个大的图并且遍历这个图，记录构建表与遍历表所需要的时间
- Graph500 基准评测程序按照问题的规模划分为 toy、mini、small、medium、large 和 huge 六个等级，这些也被称为等级 10-15，其中等级 10 是 toy 等级，等级 15 是 huge 等级

表 2-4 Graph500 中不同问题大小、规模、节点数以及内存要求对应表

大小	规模 (2 的次方)	节点数 ($\times 10^8$)	内存要求/TB
Toy	26	1	0.02
Mini	29	5	0.14
Small	32	43	1.1
Medium	36	687	17.6
Large	39	5498	141
Huge	42	43980	1126

Graph500

- 图遍历过程采用 BFS 或者是 SSSP 作为核心搜索程序，随机选择一个根节点作为源点进行遍历并检查保存的结果与原来的生成图是否匹配
 - 程序随机生成 64 个不同的源点，因此这个遍历过程循环执行 64 次（如果节点数少于 64 个，则执行小于 64 次的循环遍历），并进行计时



Graph500

- 为了比较各种体系结构、编程模型、语言以及框架的性能，
Graph500 的性能指标通常定义为：每秒遍历的边数
(Traversed Edges Per Second, TEPS)
 - TEPS 由生成图的边数和核心搜索程序算法的耗时比值得到
 - 在访问搜索根源点之前立即开始对搜索时间计时，将输出结果写入内存后，停止计时

Graph500

- 按照重要性的顺序, Graph500 基准评测的目标是:
 - 公平遵守基准规范的意图
 - 给定机器的最大问题大小
 - 给定问题大小的最短执行时间
- 其他次要目标还包括:
 - 最小代码量 (不包括验证码)
 - 最少的开发时间
 - 最大的可维护性
 - 最大的可扩展性



HOME COMPLETE RESULTS GREEN GRAPH500 SUBMISSIONS BENCHMARK SPECIFICATION CONTACT US

Show 10 entries

Search:

RANK	PREVIOUS RANK	MACHINE	VENDOR	TYPE	NETWORK	INSTALLATION SITE	LOCATION	COUNTRY	YEAR	APPLICATION	USAGE	NUMBER OF NODES	NUMBER OF CORES	MEMORY
1	new	Supercomputer Fugaku	Fujitsu	Fujitsu A64FX	Tofu Interconnect D	RIKEN Center for Computational Science (R-CCS)	Kobe Hyogo	Japan	2020	Various scientific and industrial fields	Academic and industry	152064	7299072	4866048
2	2	Wuhan Supercomputer	HUST	Kunpeng 920+Tesla A100	Custom	Wuhan Supercomputing Center	Wuhan	China	2023	Research	Academic	252	6999552	584640
3	3	Frontier	HPE	HPE Cray EX235a	Slingshot-11	DOE/SC/Oak Ridge National Laboratory	Oak Ridge TN	United States	2021	government	open research	9248	8730112	9469952
4	4	Pengcheng Cloudbrain-II	HUST-Pengcheng Lab-HUAWEI	Kunpeng 920+Ascend 910	Custom	Pengcheng Lab	ShenZhen	China	2022	Research	Academic	488	93696	999424
5	new	Aurora	Intel/HPE	HPE Cray EX - Intel Exascale Compute	HPE Slingshot-11	DOE/SC/Argonne National Laboratory	Argonne IL	United States	2023	Open research	government	4096	25591808	4718592



HOME COMPLETE RESULTS GREEN GRAPH500 SUBMISSIONS BENCHMARK SPECIFICATION CONTACT US

JUNE 2024 SSSP

Show entries

Search:

RANK	PREVIOUS RANK	MACHINE	VENDOR	TYPE	NETWORK	INSTALLATION SITE	LOCATION	COUNTRY	YEAR	APPLICATION	USAGE	NUMBER OF NODES	NUMBER OF CORES
1	1	Wuhan Supercomputer	HUST	Kunpeng 920+Tesla A100	Custom	Wuhan Supercomputing Center	Wuhan	China	2023	Research	Academic	252	6999552
2	2	Pengcheng Cloudbrain-II	HUST-Pengcheng Lab-HUAWEI	Kunpeng 920+Ascend 910	Custom	Pengcheng Lab	ShenZhen	China	2022	Research	Academic	488	93696
3	3	Supercomputer Fugaku	Fujitsu	Fujitsu A64FX	Tofu Interconnect D	RIKEN Center for Computational Science (R-CCS)	Kobe Hyogo	Japan	2020	Various scientific and industrial fields	Academic and industry	82944	3981312
4	4	Tianhe Exascale Prototype Upgrade	National University of Defense Technology	FT-2000V	TH Express-3	National Supercomputer Center in Tianjin	Tianjin	China	2021	Industry	Research	2048	131072



June 2024 Green: SMALL DATA

Show entriesSearch:

RANK	MTEPS/W	SITE	MACHINE	GRAPH500 RANK	SCALE	GTEPS	NODES	CORES	POWER
1	22094.87	Haifa	RAZIEL	17	28	1679.210000	1	16	76
2	12512.81	HUST && ZHEJIANG LAB	JiFeng	50	20	461.222000	1	512	36.86
3	8635.11	National Supercomputer Center in Tianjin	Tianhe Exa-node Prototype@Newbenchmarking@GreenBFS	29	27	1165.740000	1	64	135
4	6396.14	BDTS-SCTS-CGCL	YITU	33	27	1010.590000	1	28	158
5	6099.61	National Supercomputer Center in Tianjin	Tianhe Exa-node Prototype@GraphV	36	26	884.443000	1	64	145
6	6028.85	UPenn	ENIAD	39	26	783.750300	1	16	130
7	4724.30	National Supercomputer Center in Tianjin	Tianhe Exa-node Prototype	-1	25	779.510000	1	64	165

June 2024 Green: BIG DATA

Show entries

Search:

RANK	MTEPS/W	SITE	MACHINE	GRAPH500 RANK	SCALE	GTEPS	NODES	CORES	POWER
1	16238.05	Haifa	RAZIEL	-1	30	1412.710000	1	16	87
2	6320.24	National Supercomputer Center in Tianjin	Tianhe Exa-node Prototype@Newbenchmarking@SuperCSR	32	30	1055.480000	1	64	167
3	4385.24	National Supercomputer Center in Tianjin	Tianhe Exa-node Prototype	38	30	811.270000	1	64	185
4	3938.92	BDTS-SCTS-CGCL	YITU	-1	30	909.891000	1	28	231
5	2057.08	University of Pennsylvania	ENIAD	51	30	436.100000	1	8	212
6	1135.38	NTT Musashino R-and-D Center	GMC6KA	-1	30	186.100000	1	16	163.91
7	701.36	IPADS@SJTU & Huawei	Polymer2-Arm	-1	30	238.463000	1	96	340

评估文件系统和存储性能

- **MDTEST**: MDTEST是一个广泛使用的文件和目录创建、访问和删除的基准测试工具。
- **IOR**: IOR (Interoperability of I/O Ribbons)是一个设计用于测量并行文件系统和存储系统I/O性能的基准测试工具。
- **IO500**: IO500是一个综合性的基准测试套件，旨在评估高性能计算环境下文件系统和存储系统的性能。

MDTest

- MDTest 用于评估文件系统的元数据性能的基准评测
 - 该程序通过在一组机器（通常是 HPC 集群中的计算节点）上并行创建、统计和删除目录树和文件树来评测 IO 性能
- 评测指标：每秒操作数 (OP/秒)

MDTest 测试程序可以通过<https://sourceforge.net/projects/mdtest/>下载得到

IOR

- IOR 可用于使用多种 IO 接口 (例如 POSIX, MPI-IO, HDF5 等) 和访问模式来测试并行文件系统的性能
 - 通过接收参数，在客户端上产生特定的工作负载从而测试系统的 IO 性能并输出评测结果
 - 评测结果中带宽是通过传输的数据量除以停止时间戳与开始时间的差值得到

IO500

- 由于测试方法、工具、参数甚至测试步骤的先后顺序不同，不同厂商发布的 IO 性能测试结果具有很大的差异性
- IO500可以对高性能存储系统进行标准的测试和比较

IO500

- 从评测方法上，IO500 主要分为两组评测：
 - 第一组评测的是高性能计算机 I/O 的**理想性能**，通过大文件的读写等方式来测评计算机存储系统的性能上限
 - 理想性能评测包括两类，即 IOR easy 和 MDTest easy
 - 理想性能评测能够有效地激励世界各大存储厂商不断地提升系统的极限性能
 - 第二组评测的是高性能计算机 I/O 的**极限性能**，通过海量小文件的读写来测评计算机存储系统的性能极限
 - 极限性能评测包括两类，即 IOR hard 和MDTest hard
- IO500 最终分数是**IOR分数**和**MDTest分数**的几何平均值

IO500 ISC22 List

IO500

10 Node

Full

Historical

Customize

Download

This is the ISC22 IO500 list

# ↑	INFORMATION							IO500		
	BOF	INSTITUTION	SYSTEM	STORAGE VENDOR	FILE SYSTEM TYPE	CLIENT NODES	TOTAL CLIENT PROC.	SCORE ↑	BW (GIB/S)	MD (KIOP/S)
1	ISC21	Pengcheng Laboratory	Pengcheng Cloudbrain-II on Atlas 900	Pengcheng	MadFS	512	36,864	36,850.40	3,421.62	396,872.82
2	ISC22	National Supercomputing Center in Jinan	Shanhe	PDSL	flashfs	10	2,560	3,534.42	207.79	60,119.50
3	SC21	Huawei HPDA Lab	Athena	Huawei	OceanFS	10	1,720	2,395.03	314.56	18,235.71
4	SC21	Olympus Lab	OceanStor Pacific	Huawei	OceanFS	10	1,720	2,298.69	317.07	16,664.88
5	SC21	Huawei Cloud		PDSL	Flashfs	15	1,560	2,016.70	109.82	37,034.00
6	ISC21	Intel	Endeavour	Intel	DAOS	10	1,440	1,859.56	398.77	8,671.65
7	ISC20	Intel	Wolf	Intel	DAOS	52	1,664	1,792.98	371.67	8,649.57
8	ISC22	LRZ	SuperMUC-NG Phase2	Lenovo	DAOS	20	1,280	1,371.30	321.75	5,844.40
9	ISC22	University of Cambridge	Cumulus	Dell/Intel	DAOS	200	2,000	1,107.17	283.19	4,328.68
10	ISC21	Lenovo	Lenovo-Lenox	Lenovo	DAOS	36	3,456	988.99	176.37	5,545.61

- 用于评估HPC集群在不同消息粒度下节点间点对点、全局通信的效率
 - 点对点通信
 - 评测测试的是两个进程间的消息传递
 - 包括了 Ping-Pong 和 Ping-Ping 测试
 - 并行通信
 - 评测测试的是全局负载下消息的收发效率
 - 包括 Sendrecv 和 Exchange 测试
 - 群体通信
 - 评测测试一对多或者是多对一的消息传递

OSU Benchmark

- 由Ohio State University提供
 - 程序生成不同规模的数据并执行各种不同模式的MPI通信，测试各种通信模式的延迟和带宽
 - 延迟测试采用前面提到的 Ping-Pong 的方式
 - 带宽测试分为点对点通信和组通信两种形式

Green500

- Green500 提供高性能计算机的能耗排名
 - 评测指标：使用 PPW (performance per watt)
每瓦特性能作为其指标来对能源效率进行排名

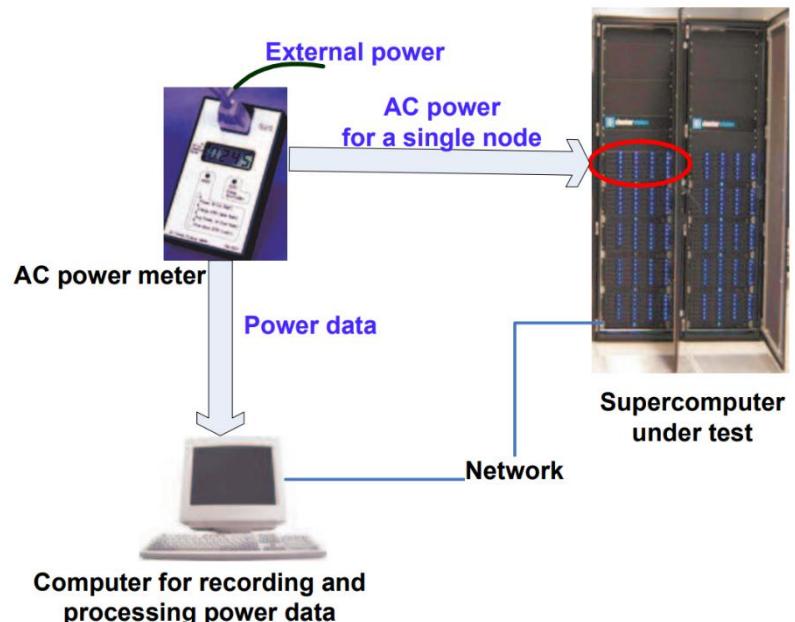
$$\text{PPW} = \frac{\text{Performance}}{\text{Power}} = \frac{R_{max}}{\bar{P}(R_{max})}$$

- 单位为 GFLOPS Per Watt

功率测量方法

- 直接测量方法：
 - 直接测量方法通过使用功率计或电流传感器直接测量超级计算机的电力输入。功率计可以接入电源线路来测量电流和电压，并计算功率。这种方法可以提供较准确的功率测量结果，但需要考虑电流和电源的稳定性。
- 间接测量方法：
 - 间接测量方法通过测量超级计算机的其他物理参数来间接估计功率。例如，可以使用温度传感器测量系统的温度，并根据热传导定律和热功率模型推导出功率。
- 硬件监控方法：
 - 硬件监控方法使用超级计算机中的硬件监控功能来获取功耗数据。
- 软件监控方法：
 - 软件监控方法使用特定的性能监测工具和软件来估计功耗。这些工具可以通过监测处理器利用率、内存访问模式和其他负载信息来推断功耗。

功率测量方法



超级计算机上单个单元的功率测量图

- GFLOPS Per Watt = $\frac{R_{max}(\text{in GFLOPS})}{\bar{P}(R_{max})(\text{in Watt})}$
- $\bar{P}(R_{max}) = N \cdot \bar{P}_{unit}(R_{max})$

功率测量方法

- 任何功率测量方法都有一定的误差范围，并考虑到测量不确定性。
- 考虑到超级计算机的整体功耗，包括计算节点、网络和存储等各部分的能耗。
- 定期进行功率测量，以监控和评估超级计算机的能源效率，并在必要时进行调整和优化。
- 总之，超级计算机的功率测量需要综合考虑硬件和软件监控方法，以获得更准确的数据，并用于评估和优化能源效率。

Green500 Data

TOP500				Rmax (PFlop/s)	Power (kW)	Energy Efficiency (GFlops/watts)
Rank	Rank	System	Cores			
1	189	JEDI - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, ParTec/EVIDEN EuroHPC/FZJ Germany	19,584	4.50	67	72.733
2	128	Isambard-AI phase 1 - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE University of Bristol United Kingdom	34,272	7.42	117	68.835
3	55	Helios GPU - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cyfronet Poland	89,760	19.14	317	66.948
4	328	Henri - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo Flatiron Institute United States	8,288	2.88	44	65.396
5	71	preAlps - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS)	81,600	15.47	240	64.381

Minapplication

- 评估超级计算机对于 动态应用程序的性能
 - 是真实应用程序的更小版本
 - 既足够复杂，又足够小，可以探索内存、网络、加速器和处理器元素的参数和交互空间
- Mantevo 集包含大量应用领域的 开源 Minapplication
 - MiniAMR：用于探索自适应网格细化和动态执行的小型应用程序
 - MiniFE：有限元代码的小型应用程序，提供了对多核节点上进行计算的支持
 - MiniGhost：有限差分小型应用程序，可在均匀三维域上实现有限差分
 - MiniMD：基于分子动力学工作负载的小型应用程序
 - Cloverleaf：使用显式的二阶精确方法来求解可压缩欧拉方程
 - TeaLeaf：求解线性热传导方程工作负载的小型应用程序

戈登贝尔奖

- **戈登贝尔奖 (ACM Gordon Bell Prize) : 超算界诺贝尔奖**
 - 美国计算机协会设立于1987年，每年颁发
 - 是一种超级电脑应用软件设计奖，奖金象征性1万美元，由 Gordon Bell 提供
 - 通常会由**当年前500排行名列前茅的超级电脑系统之上所跑的应用软件**获得
- **奖项通常分为：**
 - 最高性能奖 (Peak Performance)
 - 最高性价比奖 (Price/Performance)
 - 特别奖 (Special Achievement)

Gordon Bell

- **戈登·贝尔 (Gordon Bell)**
 - 1934年8月19日出生于美国密苏里州的柯克斯维尔
 - 1956年，获MIT电子工程学士学位
 - 1957年，获MIT电子工程硕士学位
 - 1960年-1983年，在DEC任副总裁，负责研发
 - 1983年7月，合伙创办核心(Encore)计算机公司
 - 1986年，全美科学基金会(NSF)计算机及信息科学和工程助理主任
 - 1991年-1995年，担任微软公司顾问
 - 1995年8月-至今，微软湾区研究中心高级研究员
- **作为DEC的技术灵魂，构思、设计和主持开发的超级计算机PDP-4, PDP-5, PDP-6, PDP-8, PDP-10及PDP-11**



戈登·贝尔
(Gordon Bell)

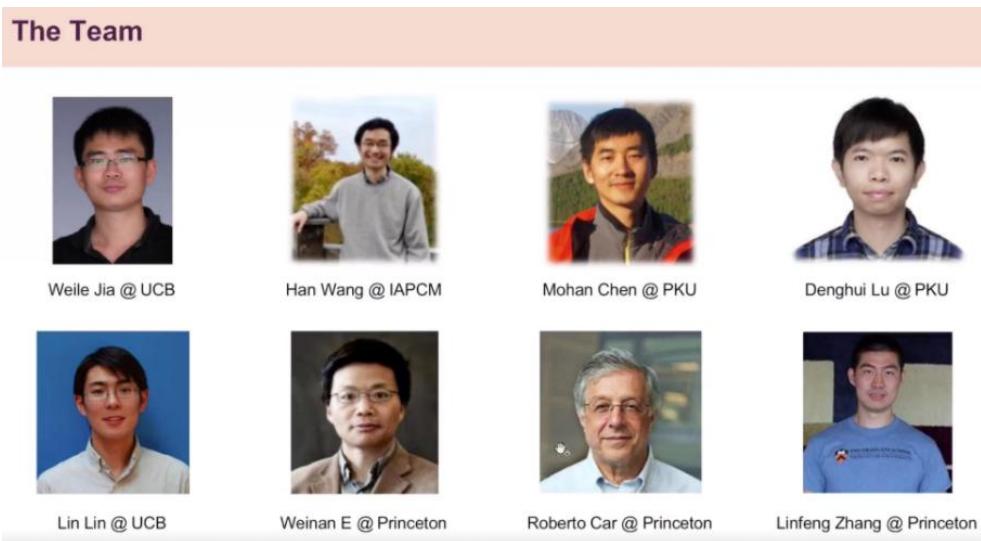
中国第一次获得戈登贝尔奖

- **2016年中国第一次获得戈登贝尔奖：**
 - 神威·太湖之光超级电脑上的“**全球大气非静力云分辨模拟**”应用软件得奖，使用了超过一千万个核来完成一次气候的数值仿真
 - 从2007年开始相关研究，到2011年冲奖团队正式成立，到2012年第一次尝试冲奖，再到2016年正式获奖



2020年戈登贝尔奖

- 2020年11月该奖项颁给了一支由中美科学家组成的研究团队
 - 因“结合分子建模、机器学习和高性能计算相关方法，将具有从头算精度的分子动力学模拟的极限提升至**1亿个原子规模**”获奖
 - 获奖的8人团队中，有7张中国面孔
 - 该团队将这一工作在美国超算Summit机器上全机运行，模拟分别实现了双精度**91PFlops**、混合单精度**162PFlops**和混合半精度275PFlops的峰值性能



2021年戈登贝尔奖

- 2021年11月18日下午于美国密苏里州圣路易斯举行的全球超级计算大会（SC21）上，国际计算机协会（ACM）将2021年度“戈登贝尔奖”授予中国超算应用团队
- 这是我国联合科研团队基于新一代神威超级计算机的应用“超大规模量子随机电路实时模拟”（SWQSIM）而获此殊荣

(三) 如何提高高性能

Take Advantage of Parallelism

- Using parallelism is one of the most important methods for improving performance
- Level of the system:
 - multiple processors and multiple storage devices can be used.
 - The workload of handling requests can then be spread among the processors and storage devices
- Level of individual processor.
 - pipelining: the best-known example of ILP
- Level of detailed digital design.
 - E.g., set-associative caches use multiple banks of memory that are typically searched in parallel to find a desired item

Principle of Locality

- The principle of locality:
 - programs tend to reuse data and instructions they have used recently.
- An implication of locality
 - we can predict with reasonable accuracy what instructions and data a program will use in the near future
 - based on its accesses in the recent past.
- Two different types of locality have been observed.
 - Temporal locality: recently accessed items are likely to be accessed soon.
 - Spatial locality: items whose addresses are near one another tend to be referenced close together in time.

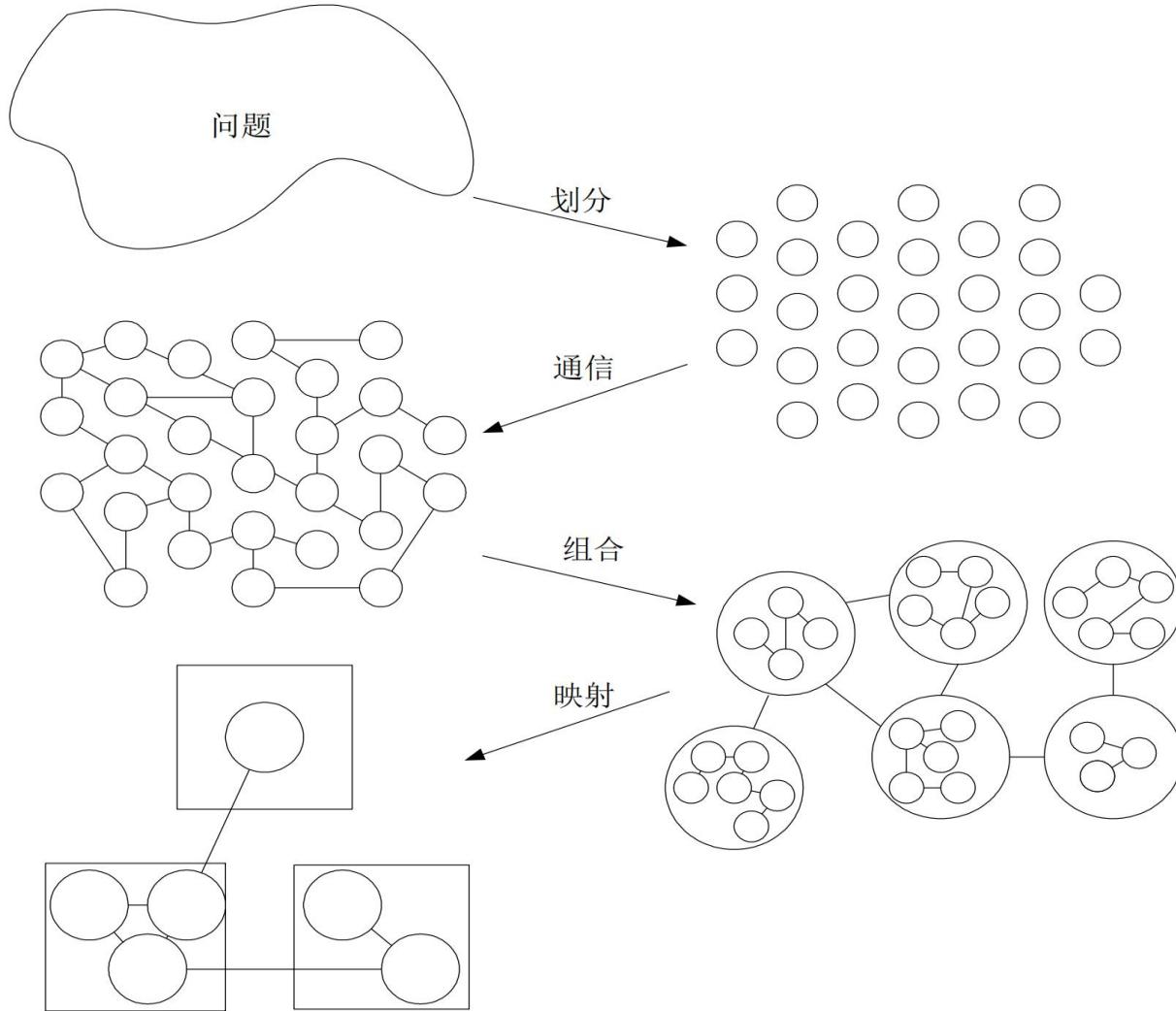
Focus on the Common Case

- The most important and pervasive principle of computer design is to **focus on the common case**:
 - in making a design trade-off, favor the **frequent case** over the infrequent case.
 - This principle applies when determining **how to spend resources**
- The common case is often **simpler** and can be done **faster** than the infrequent case.
 - E.g., overflow is rare, so first optimize the more common case of no overflow

并行算法（程序）的设计步骤

- **任务划分 (Partitioning)**
 - 将整个计算分解为一些小的任务，其目的是尽量开拓并行执行的机会
- **通信 (Communication) 分析**
 - 分析确定诸任务执行中所需交换的数据和协调诸任务的执行，由此可检测上述任务划分的合理性
- **任务组合 (Agglomeration)**
 - 按性能要求和实现的代价来考察前两阶段的结果，必要时可将一些小的任务组合成更大的任务以提高性能或减少通信开销
- **处理器映射 (Mapping)**
 - 将每个任务分配到一个处理器上，其目的是最小化全局执行时间和通信成本以及最大化处理器的利用率

并行算法（程序）的设计步骤



任务划分

- **域分解（Domain Decomposition）也叫数据划分**
 - 划分对象是数据
 - 算法（或程序）的输入数据、计算的输出数据、或者算法所产生的中间结果
 - 步骤：
 - 首先，分解与问题相关的数据，使这些小的数据片尽可能大致相等
 - 其次，将每个计算关联到它所操作的数据上
 - 由此将产生一系列的任务，每个任务包括一些数据及其上的操作
 - 当一个操作可能需要别的任务中的数据时，就会产生通信要求

任务划分

- **功能分解 (Functional Decomposition) 也称计算划分**
 - 关注于被执行的计算上，而不是计算所需的数据上
 - 如果所做的计算划分是成功的，再继续研究计算所需的数据，如果这些数据基本上不相交，就意味着划分的成功
 - 大多数并行算法采用域分解，但功能分解有时能揭示问题的内在结构

任务划分

- **一个并行程序通常同时存在数据和功能并行的机会**
 - 功能并行的并行度通常比较有限，并且不会随着问题规模的扩大而增加；不同的函数所涉及的数据集的大小可能差异很大，因此也难于实现负载均衡
 - 数据并行则一般具有较好的可扩展性，也易于实现负载均衡
 - 现有的绝大多数大规模的并行程序属于数据并行应用，但功能分解有时能提示问题的内在结构展示出优化的机遇

任务划分

- **合理性检查**

- (1) 所划分的任务数是否高于目标机上处理器数目1-2个量级?
 - 若不是, 在后面的设计步骤中将缺少灵活性
 - (2) 划分是否避免了冗余的计算和存储要求?
 - 若不是, 则产生的算法对大型问题可能不是可扩展的
 - (3) 各任务的尺寸是否大致相当?
 - 若不是, 则分配处理器时很难做到负载均衡
 - (4) 划分的任务数是否与问题尺寸成比例?
 - 理想情况下, 问题尺寸的增加应引起任务数的增加而不是任务尺寸的增加
 - 若不是这样, 算法可能不能求解更大的问题, 尽管有更多的处理器
 - (5) 是否采用了几种不同的划分法?
 - 多考虑几种选择可以提高灵活性。同时既要考虑域分解又要考虑功能分解

通信分析

- **局部/全局通信**: 较少的几个近邻或与很多别的任务通信
 - 局部通信: 每个任务只和较少的几个近邻通信
 - 全局通信: 每个任务与很多其他任务通信
- **结构化/非结构化通信**: 通信图 规整结构 (如树、网格等) 或任意图
 - 结构化: 规整结构, 如树、网格等
 - 非结构化: 通信网络可能是任意图, 如稀疏矩阵-向量乘 (消息传递编程困难)
- **静态/动态通信**: 不随时间改变或可变的且由运行时所计算的数据决定
 - 静态: 通信伙伴身份不随时间改变, 如矩阵相乘
 - 动态: 通信伙伴的身份可能由运行时数据决定且是可变的, 如15-puzzle问题, 消息传递编程较困难

通信分析

- **同步/异步通信**: 接收方和发送方协同操作或无需协同
- **One-way/two-way**:
 - Two-way: 通信存在生产者-消费者关系, 如读写
 - One-way: 通信只需要一方发起并完成, 如只读

任务组合

- 粒度控制
 - 大量细粒度任务有可能增加通信代价和任务创建代价
 - 表-容效应 (Surface-Volume Effect)
 - 一个任务通信需求比例于它所操作的子域的表面积，而计算需求却比例于子域的容积
 - 体积不变的前提下，增加计算操作深度（又称冗余计算），能够减少子域表面积，进而减少通信量

任务组合

- **组合判据**
 - 通过组合增加粒度是否减少了通讯成本？
 - 用重复计算换取通信代价是否已权衡了其得益？
 - 组合后是否保持了灵活性和可扩放性？
 - 组合的任务数是否与问题尺寸成比例？
 - 组合后的各个任务是否保持了类似的计算和通讯代价？
 - 有没有减少并行执行的机会？

处理器映射

- **处理器映射策略：**指定任务到哪个处理器上去执行，其主要目标是减少算法的总执行时间，策略有二：
 - 把那些可并发执行的任务放在不同的处理器上以**增强并行度**
 - 把那些需频繁通信的任务置于同一个处理器上以**提高局部性**
- **负载均衡：**使得所有处理器完成等量的任务
 - 减少同步等待的时间，这包括等待其它进程结束运行的时间和串行执行的代码部分（包括临界区代码和因数据相关造成的串行执行）
 - 通常采用的一种策略是在任务分配中，先集中目标使负载尽量均衡，然后再对任务分配进行调整，使得交互尽量少

任务分配与调度

- **静态调度：任务到进程的算术映射**
 - 静态地为每个处理器分配连续的循环迭代（要求处理器计算能力同构）
 - 轮转（将第 i 个循环迭代分配给第 $i \bmod P$ 个处理器）
- **动态调度：动态调度技术可以取得较好的负载均衡效果（但开销较大）**
 - 基本自调度SS (Self Scheduling) : 每个处理器空闲时从全局队列取一个任务
 - 块自调度BSS (Block Self Scheduling) : 每次取k个任务（块）
 - 指导自调度GSS (Guided Self Scheduling) : 每次取剩余任务的 $1/P$
 - 因子分解调度FS (Factoring Scheduling) : $C_i = R_i / 2P$, 每阶段所有处理器任务大小相等
 - 梯形自调度TSS (Trapezoid Self Scheduling) : 连续的块之间的差距固定不变
 - 安全自调度SSS (Safe Self Scheduling) : 任务分配使得累计执行时间刚刚超过平均负载
 - 亲和性调度AS (Affinity Scheduling) : 分布式任务队列（本地调度+远程调度）
 - 自适应耦合调度AAS (Adapt Affinity Scheduling) : 初始分配不平衡、计算能力异构
 - 重载、轻载、常载

本章小结

- **计算机性能**
 - 性能指标
 - Amdahl 定律
 - Gustafson 定律
 - 可扩展性评测标准
- **基准评测集**
- **计算机性能提升**
 - 并行算法（程序）的设计步骤

参考文献

- Dongarra J J, Luszczek P, Petitet A. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 2003, 15(9):803–820.
- Dongarra J J, Moler C B, Bunch J R, et al. LINPACK users' guide. SIAM, 1979.
- Dongarra J, Heroux M A, Luszczek P. High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems. *The International Journal of High Performance Computing Applications*, 2016, 30(1):3–10.
- Murphy R C, Wheeler K B, Barrett B W, et al. Introducing the graph 500. *Cray Users Group (CUG)*, 2010, 19:45–74.
- Kunkel J, Lofstead G F, Bent J. The virtual institute for i/o and the io-500. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2017.
- Bell G, Bailey D H, Dongarra J, et al. A look back on 30 years of the gordon bell prize. *The International Journal of High Performance Computing Applications*, 2017, 31(6):469–484.