并行计算
Parallel Computing

主讲 孙经纬

2024年 春季学期

概要

- 第一篇 并行计算硬件平台：并行计算机
  - 第一章 并行计算与并行计算机结构模型
  - 第二章 并行计算机系统互连与基本通信操作
  - 第三章 典型并行计算机系统介绍
  - **第四章 并行计算性能评测**

第四章 并行计算性能评测

- **4.1 并行机的一些基本性能指标**
- 4.2 加速比性能定律
  - 4.2.1 Amdahl定律
  - 4.2.2 Gustafson定律
  - 4.2.3 Sun和Ni定律
- 4.3 可扩放性评测标准
  - 4.3.1 并行计算的可扩放性
  - 4.3.2 等效率度量标准
  - 4.3.3 等速度度量标准
  - 4.3.4 平均延迟度量标准
- 4.4 基准测试程序

一些基本性能指标

- 工作负载
  - 执行时间
  - 浮点运算数: FLOPs (FLoating-point OPerations)
- 执行速度
  - 指令执行速度: MIPS (Million Instructions Per Second)
  - 浮点运算速度: FLOPS

    FLoating-point Operations Per Second

一些基本性能指标

- 无重叠的假定下：并行执行时间包括：
  - T_{comput} 计算时间，T_{paro} 并行开销时间，T_{comm}相互通信时间

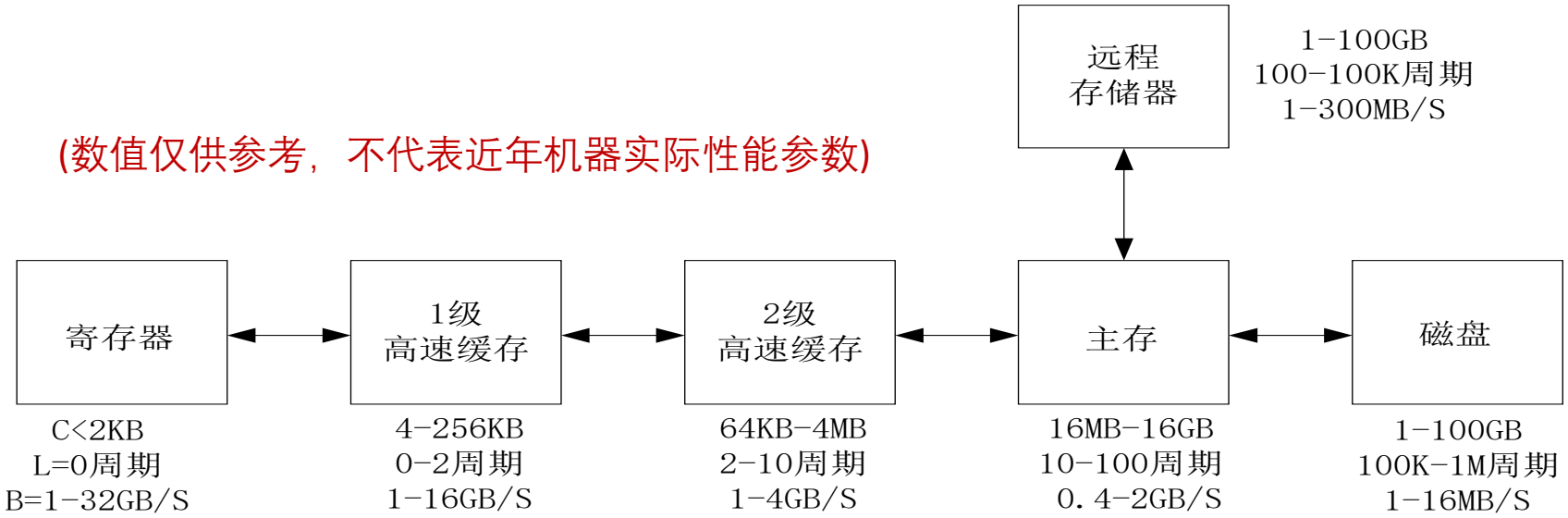
    并行执行时间 $T_n = T_{comput} + T_{paro} + T_{comm}$

例：PowerPC每个周期15ns，可执行4FLOPs；
创建一个进程1.4ms，可执行372000FLOPs

  - T_{paro}：进程管理（如进程生成、结束和切换等），组操作（如进程组的生成与消亡等），进程查询（如询问进程的标志、等级、组标志和组大小等）
  - T_{comm}：同步（如路障、锁、临界区、事件等），通信（如点到点通信、整体通信），聚合操作（如规约、前缀运算等）

存储器性能

- 存储器的层次结构(容量C,延迟L,带宽B)



- 估计存储器的带宽

RISC的加法可在单拍内完成（从寄存器取两个数相加，结果写回寄存器），假定字长8bytes，时钟频率100MHz，则

带宽B = 3*8*100*10^6 B/s= 2.4GB/s

如何估计其他级别的存储器带宽?

为什么是3?

通信开销：点到点通信

- 通信开销的测量：乒-乓方法（Ping-Pong Scheme）

a) 节点0发送m个字节给节点1；

b) 节点1从节点0接收m个字节后，立即将消息发回节点0

总的时间除以2，即可得到点到点通信时间，也就是执行单一发送或接收操作的时间。

- 对于n个节点，可一般化为热土豆法（Hot-Potato）

0——1 —— 2 —— … —— n-1 —— 0

通信开销：点到点通信

- Ping-Pong Scheme

```
if  (my _node _id =0)  then /*发送者*/
        start _time = second()
                send an m-byte message to node 1
                receive an m-byte message from node 1
        end_time = second()
        total_time = end_time – start_time
    communication_time[i] = total_time/2
else if  (my_node_id = 1)  then  /*接收者*/
                receive an m-byte message from node 0
                send an m-byte message to node 0
     endif
endif
```

通信开销：点到点通信

- 通信开销 $t(m) = t_0 + m/r_\infty$　m为消息长度（字节数）

  通信启动时间 t_0

  渐近带宽r_∞：传送无限长的消息时的通信速率

  半峰值长度$m_{1/2}$：达到一半渐近带宽所要的消息长度

  特定性能π_0：表示短消息带宽　$t_0 = 1/\pi_0$

- 以上是由Roger W. Hockney提出的

Roger W. Hockney: The Communication Challenge for MPP: Intel Paragon and Meiko CS-2. Parallel Computing 20(3): 389-398 (1994)

群集通信

典型的群集通信包括:

- 广播（Broadcast）：处理器0发送m个字节给所有的n个处理器
- 收集（Gather）：处理0接收所有n个处理器发来的m个字节消息，所以处理器0最终接收了mn个字节;
- 散播（Scatter）：处理器0发送了m个字节的不同消息给所有n个处理器，因此处理器0最终发送了mn个字节;
- 全交换（Total Exchange）：每个处理器均彼此相互发送m个字节的不同消息给对方，所以总通信量为mn^2个字节;
- 循环移位（Circular-shift）：处理器i发送m个字节给处理器i+1，处理器n-1发送m个字节给处理器0，所以通信量为mn个字节。

机器的成本、价格与性价比

- 机器的成本与价格
- 机器的性能/价格比 Performance/Cost Ratio：

  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MFLOPS表示）

- PVP、SMP、MPP、COW的对比
- 利用率（Utilization）： 可达到的速度与峰值速度之比

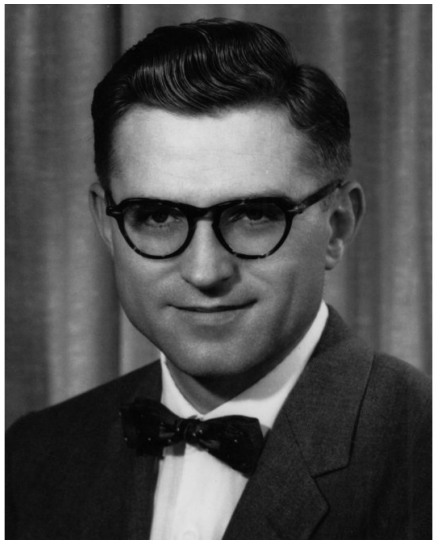
第四章 并行计算性能评测

- 4.1 并行机的一些基本性能指标
- **4.2 加速比性能定律**
  - 4.2.1 Amdahl定律
  - 4.2.2 Gustafson定律
  - 4.2.3 Sun和Ni定律
- 4.3 可扩放性评测标准
  - 4.3.1 并行计算的可扩放性
  - 4.3.2 等效率度量标准
  - 4.3.3 等速度度量标准
  - 4.3.4 平均延迟度量标准
- 4.4 基准测试程序

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）的执行速度加快多少倍。
- Amdahl 定律
- Gustafson定律
- Sun Ni定律

Gene Amdahl (1922 — 2015)



Famous for formulating a computer science concept known as Amdahl's Law (1967) and for establishing a major IT company called the Amdahl Corporation, Amdahl is also notable for his work with IBM.

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（$f = W_s/W$，$W_s=W_1$）；
  - 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：Base on Fixed Problem Size**
  - 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的，
  - 增加处理器加快执行速度，从而达到了加速的目的。

Amdahl 定律

- Amdahl's Law 表明:
  - 适用于实时应用问题。当问题的计算负载或规模固定时，必须通过增加处理器数目来降低计算时间;
  - 加速比受到算法中串行工作量的限制。
  - **Amdahl's law: argument against massively parallel systems**
- 加速比计算:

可串行部分的耗时

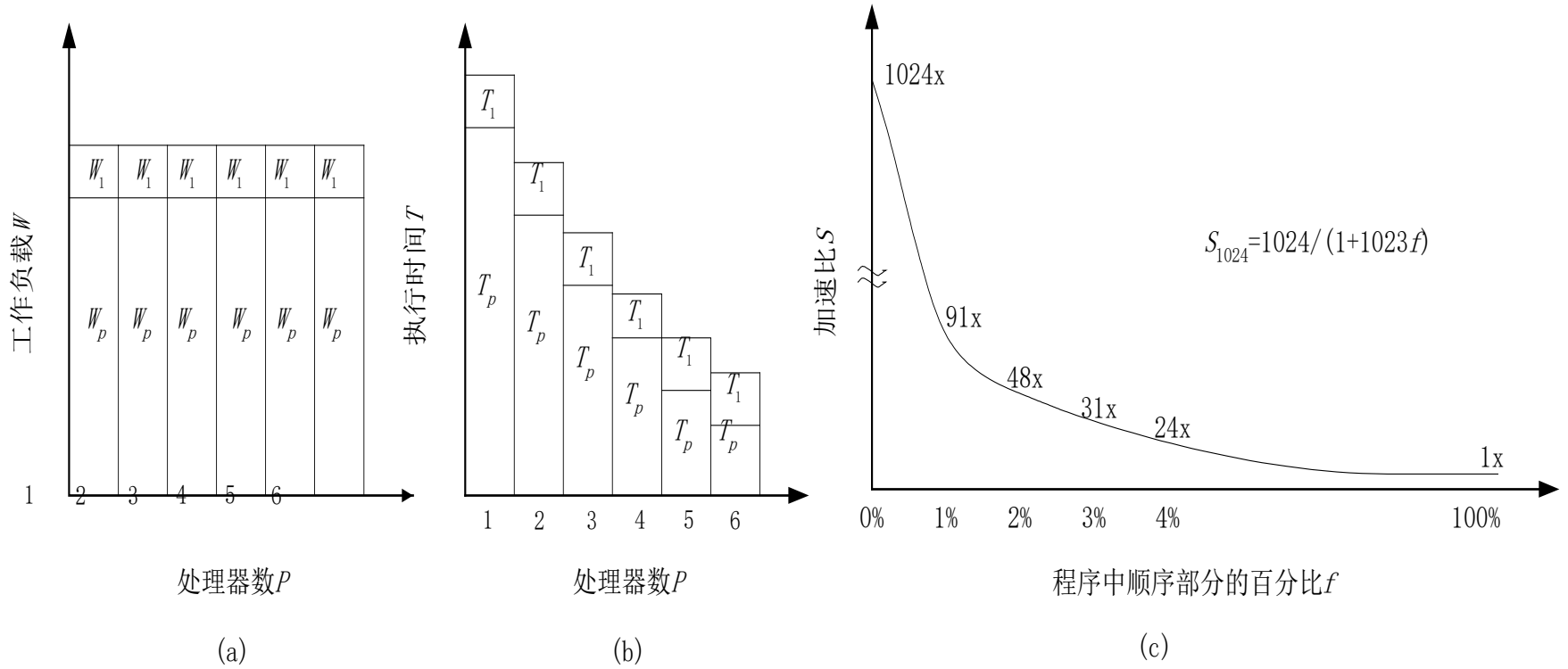
可并行部分的耗时

将可并行部分并行化

$$T_s = fW + (1-f)W \qquad T_p = fW + \frac{(1-f)W}{p}$$

$$S_p = \frac{W}{fW + \frac{(1-f)W}{p}} = \frac{p}{pf + 1 - f} = \frac{1}{\frac{(p-1)f + 1}{p}} \xrightarrow{p\to\infty} \frac{1}{f}$$

Amdahl 定律



并行加速不仅受限于串行分量，而且也受并行实现时的额外开销的限制。

John Gustafson (1955—)

Dr. Gustafson is an American computer scientist and businessman, chiefly known for his work in High Performance Computing (HPC) such as the invention of Gustafson's Law, introducing the first commercial computer cluster, etc.

Gustafson定律

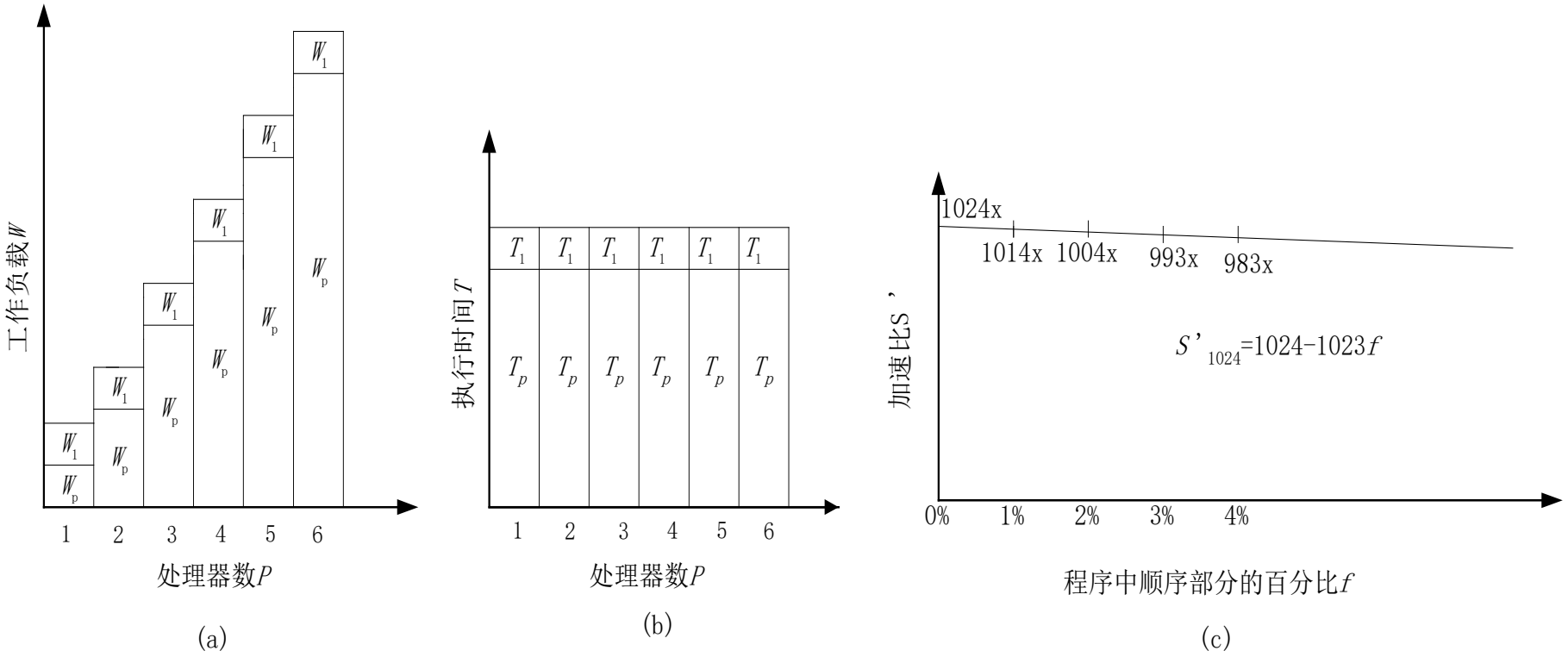
- **出发点：Base on Fixed Execution Time**
  - 对于很多大型计算，精度要求很高，即在此类应用中精度是个关键因素，而计算时间是固定不变的。此时为了提高精度，必须加大计算量，相应地亦必须增多处理器数才能维持时间不变；
  - 随着处理器数目的增加，串行执行部分*f*不再是并行算法的瓶颈。
- 加速比计算：$S' = \dfrac{W_S + pWp}{W_S + p \cdot Wp / p} = \dfrac{W_S + pWp}{W_S + W_P}$

可并行部分
随p线性增加

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

- 并行开销w_o：$S' = \dfrac{W_S + pW_P}{W_S + W_P + W_O} = \dfrac{f + p(1-f)}{1 + W_O / W}$

Gustafson定律



除非学术研究，在实际应用中没有必要固定工作负载

有了更多处理器，应当用来解决更大规模的问题

孙贤和（Xian-He Sun）
倪明选（Lionel M. Ni）

Xian-He Sun
Professor at Illinois Institute of Technology

倪明選
香港科技大學（廣州）創校校長

Sun-Ni定律

- **出发点：Base on Memory Bounding**
  - 充分利用存储空间等计算资源，尽量增大问题规模以产生更好/更精确的解。是Amdahl定律和Gustafson定律的推广。
  - 加速比计算:

    设单机上的存储器容量为M，其工作负载W=*f*W+(1-*f*)W

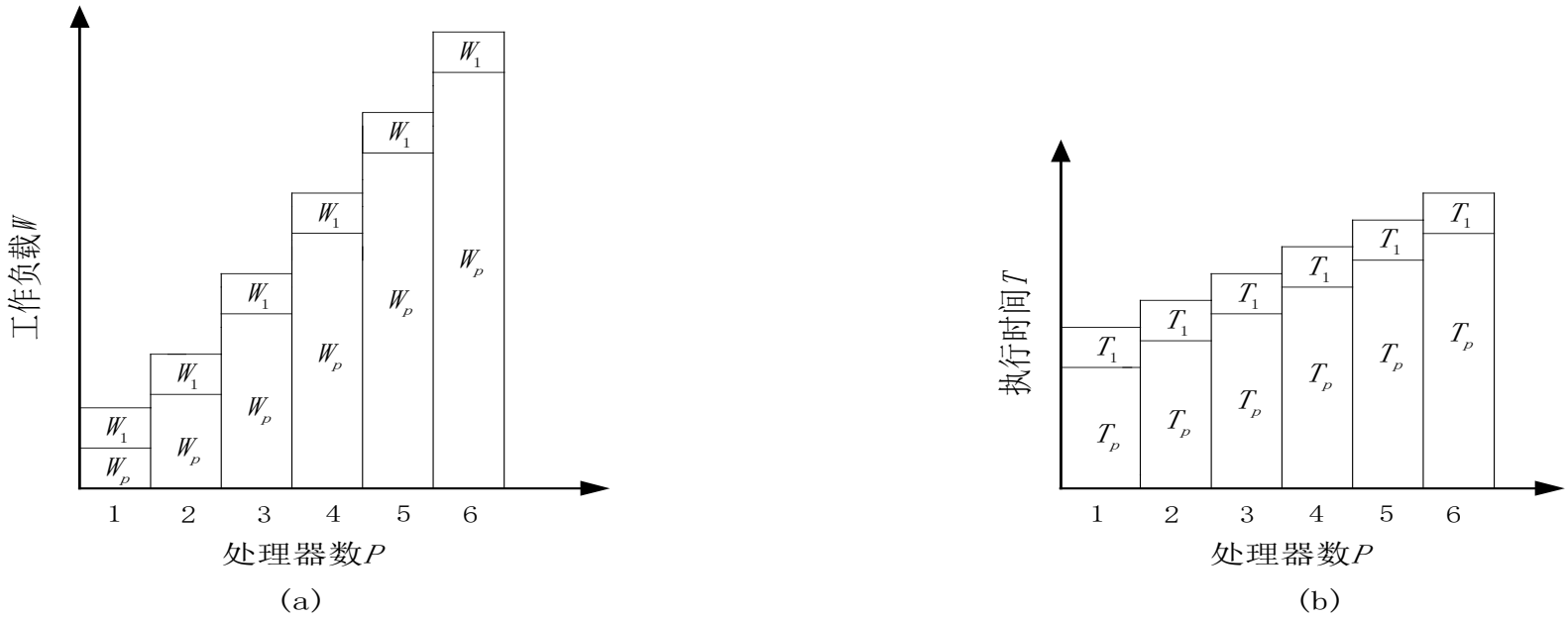
    当并行系统有p个结点时，存储容量扩大了pM，**用G(p)表示系统的存储容量增加p倍时可并行工作负载的增加量**。则存储容量扩大后的工作负载为W=fW+(1-f)G(p)W，所以存储受限的加速为

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

- 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



- 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，可并行工作负载比存储要求增加得更快，此时 Sun-Ni 加速比 Amdahl 加速和 Gustafson 加速高。

加速比讨论

- 参考的加速经验公式： $p/\log p \le S \le p$
  - 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

- 通信密集类的应用问题： $S = 1 / C(p)$

  C(p)是p个处理器的某一通信函数，常见为线性或者对数
- 超线性加速
  - 可能在某些并行搜索问题中出现

- 绝对加速：最佳串行算法与并行算法
- 相对加速：同一算法在单机和并行机上

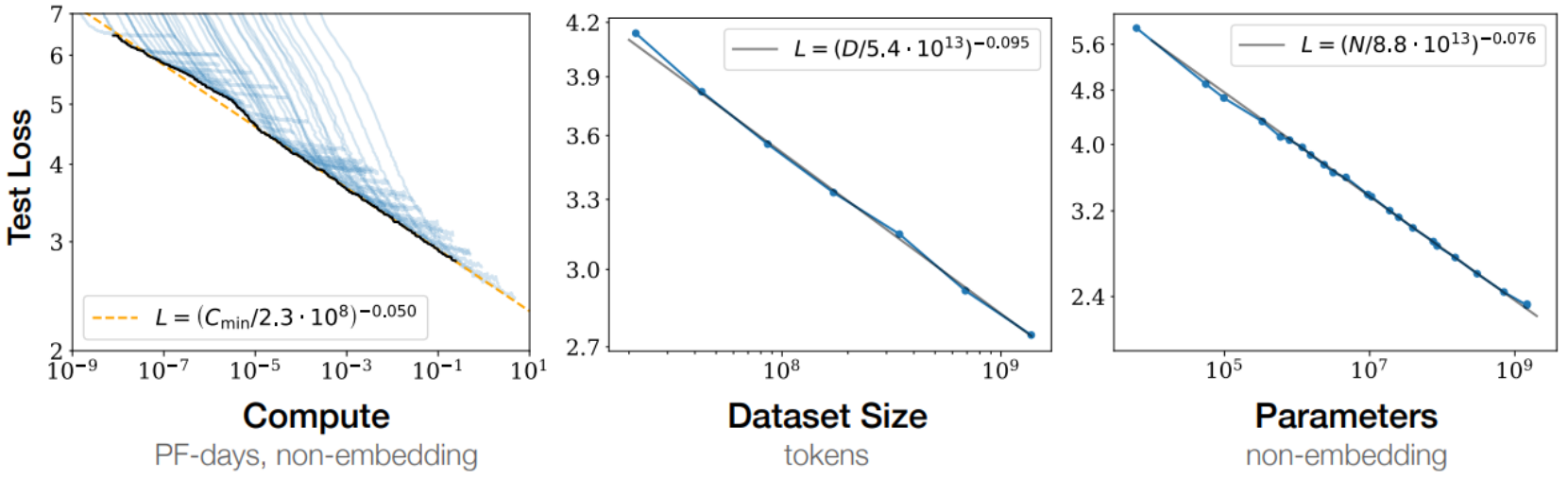
可以通过把串行算法实现得比较慢、加入大量冗余且易并行的计算，使得并行算法的加速比更好看

案例——Neural Scaling Law

- OpenAI在2020年提出：

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei: *Scaling Laws for Neural Language Models*. CoRR abs/2001.08361 (2020)

Decoder-only语言模型的最终性能主要与计算量、模型参数量和训练数据量三者相关，而与模型的具体结构(层数/深度/宽度)基本无关



PF-days: 如果每秒钟可以进行10^{15}次方运算，也就是1 Peta FLOPS，那么一天就可以进行约$8.64*10^{19}$次方运算，这个算力消耗被称为1个PF-day

案例——Neural Scaling Law

思考:

- Neural Scaling Law和Sun-Ni定律有什么关联?
- 如何估计语言模型并行训练中的G(p)?

第四章 并行计算性能评测

- 4.1 并行机的一些基本性能指标
- 4.2 加速比性能定律
  - 4.2.1 Amdahl定律
  - 4.2.2 Gustafson定律
  - 4.2.3 Sun和Ni定律
- **4.3 可扩放性评测标准**
  - 4.3.1 并行计算的可扩放性
  - 4.3.2 等效率度量标准
  - 4.3.3 等速度度量标准
  - 4.3.4 平均延迟度量标准
- 4.4 基准测试程序

可扩放性评测标准

- 并行计算的可扩放性（Scalability）也是主要性能指标
  - 可扩放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
- 影响因素：处理器数与问题规模，还有
  - 求解问题中的串行分量;
  - 并行处理引起的额外开销（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；

可扩放性评测标准

- 增加问题规模的好处：
  - 提供较高的并发机会；
  - 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）。

因此，在可扩放性的讨论中，不妨假设串行分量比例f=0

在加速比的讨论中，我们假设额外开销是不随p变化的固定值，是一种简化情况

- 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，而度量这种能力的就是可扩放性。

可扩放性评测标准

- 可扩放性:调整什么和按什么比例调整
  - 并行计算要调整的是处理数p和问题规模W，
  - p和W并非自由变量，两者可按不同比例进行调整，此比例关系（线性的，多项式的或指数的等）就反映了可扩放的程度。
  - 可以设定约束条件，形成特定的可扩放性评价标准
    - 等效率
    - 等速度
- 与并行算法和体系结构相关

可扩放性评测标准

- 可扩放性研究的主要目的:
  - 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
- 目前没有公认的、标准的和普遍接受的严格定义和评判标准

等效率度量标准

- 令t_{ie} 和t_{io} 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）

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

- 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

每个处理器的耗时都是并行算法总耗时

$T_p = t_e^i + t_o^i$，且 $T_e + T_o = pT_p$

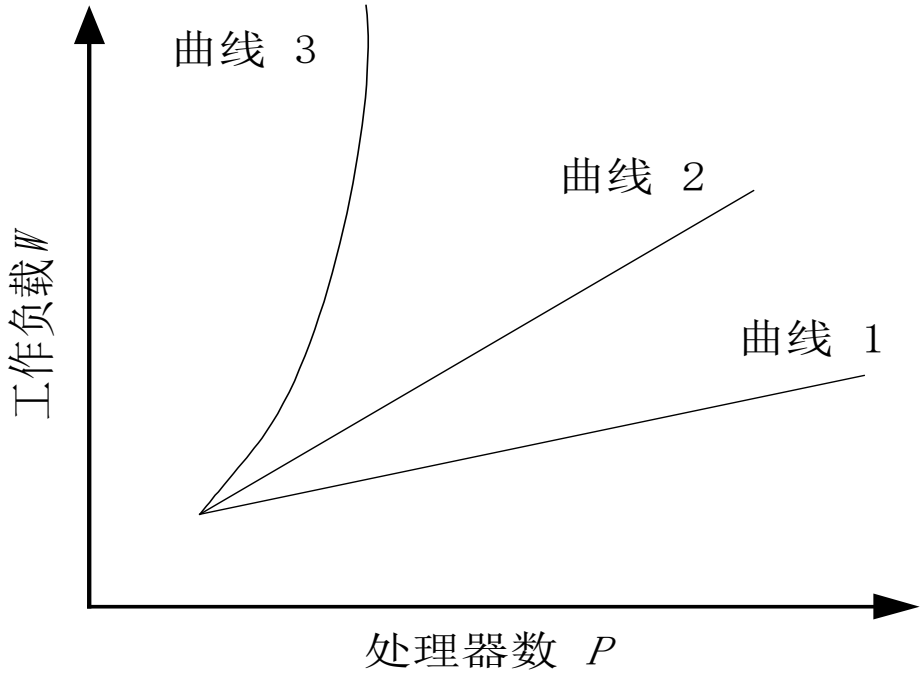
- 问题的规模W定义为最佳串行算法所完成的计算量，$W=T_e$

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

- 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（介于0与1之间），当处理数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放性；曲线2表示算法是可扩放的；曲线 3表示算法是不可扩放的。
- 优点：简单可定量计算的、少量的参数计算等效率函数
- 缺点：T_o不一定容易计算（例如在共享存储并行机中）



等速度度量标准

- 出发点：对于共享存储的并行机，T_o难以计算。如果速度能以处理器数的增加而线性增加，则说明系统具有很好的扩放性。
- p 表示处理器个数，W表示要求解问题的工作量或称问题规模（在此可指浮点操作个数），T为并行执行时间，定义并行计算的速度V为工作量W除以并行时间T
- p个处理器的并行系统的单处理器平均速度为并行速度V除以处理器个数 p:

$$\overline{V} = \frac{V}{p} = \frac{W}{pT}$$

- W是使用p个处理器时算法的工作量，令W’表示当处理数从p增大到p’时，为了保持整个系统的平均速度不变所需执行的工作量，则可得到处理器数从p到p’时平均速度可扩放度量标准公式 (介于0与1之间，越靠近1越好):

$$\Psi(p,p') = \frac{W/p}{W'/p'} = \frac{p'W}{pW'}$$

等速度度量标准

- 优点：直观地使用易测量的机器性能速度指标来度量
- 缺点：某些非浮点运算可能造成性能的变化没有考虑
- 等速度度量标准的扩放性与传统加速比之间的关系：

  当p=1时，等速度度量标准为

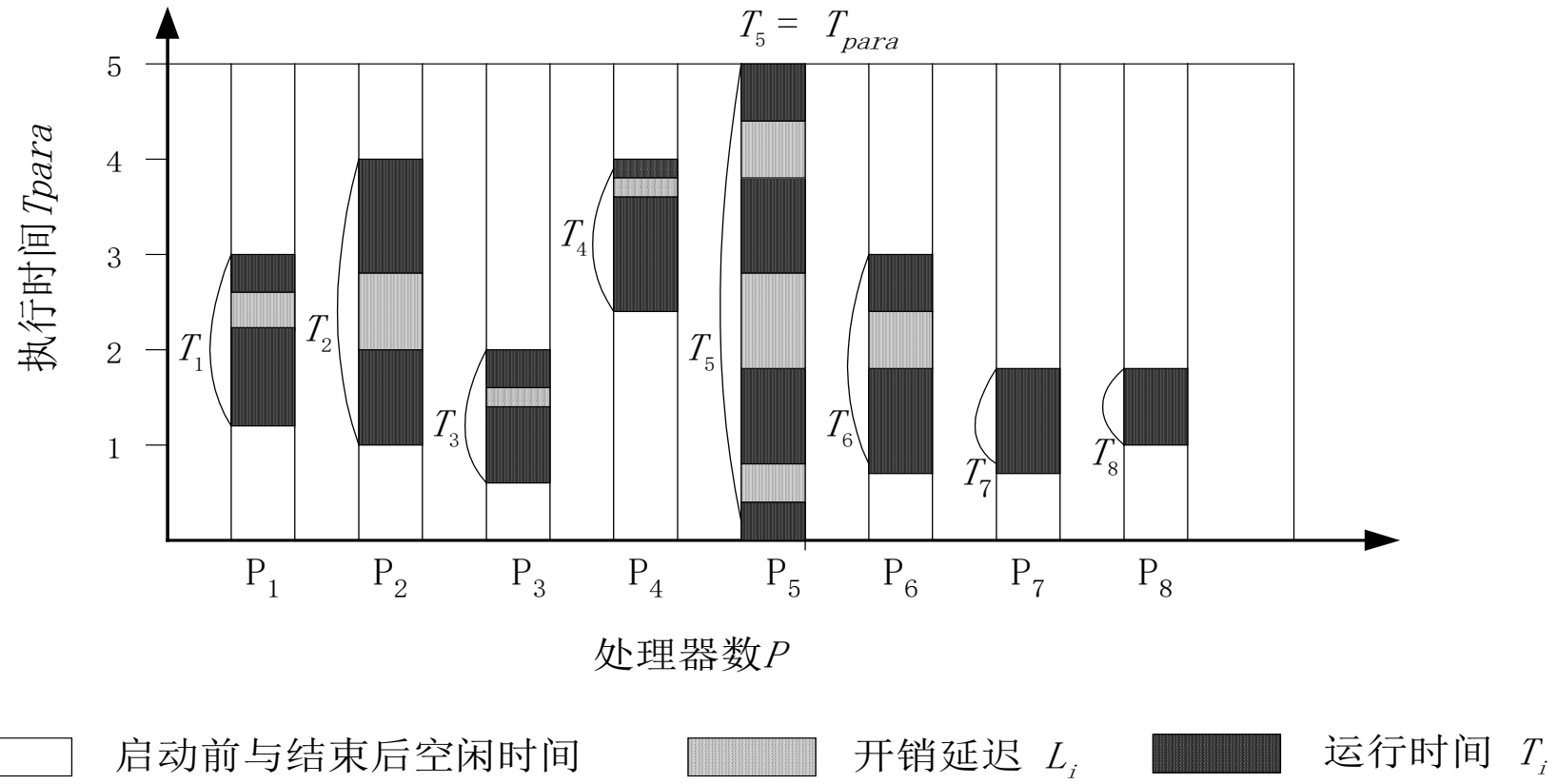
$$\Psi(p') = \Psi(1, p') = \frac{W/1}{W'/p'} = \frac{p'W}{W'} = \frac{T_1}{T_{p'}}$$

$$= \frac{\text{解决工作量为W的问题所需串行时间}}{\text{解决工作量为W′的问题所需并行时间}}$$

  其主要差别：加速比定义是保持问题规模不变，表示相对于串行系统的性能增加；扩放性定义是保持平均速度不变，表示对于小系统到大规模系统所引起的性能变化

平均延迟度量标准

- 一个并行系统执行的时间图谱



平均延迟度量标准

- T_i为P_i的执行时间，包括延迟L_i。P_i的总延迟时间为“L_i+启动时间+停止时间”。定义系统平均延迟时间为

$$\overline{L}(W,p)=\sum_{i=1}^{p}\left(T_{para}-T_i+L_i\right)\Big/p$$

又有 $pT_{para}=T_o+T_{seq}$，$T_o=p\overline{L}(W,p)$

所以 $$\overline{L}(W,p)=T_{para}-T_{seq}\Big/p$$

- 在p’个处理器上求解工作量为W’问题的平均延迟 $\overline{L}(W',p')$

平均延迟度量标准

- 当处理器数由p变到p’，而维持并行执行效率不变，则定义平均延迟可扩放性度量标准为

$$\Phi(E,p,p')=\frac{\overline{L}(W,p)}{\overline{L}(W',p')}$$

该值在0、1之间，值越接近1越好。

- 优点：平均延迟能在更低层次上衡量机器的性能
- 缺点：需要特定的软硬件才能获得平均延迟

可扩放性评测小结

- 可扩放性：随着处理器个数p增加，为了按照某个标准保持并行性能，问题规模W需要相应地增加。
- 需要增加的W越少，说明可扩放性越好
- 三个典型的标准：
  - 等效率：保持效率E=加速比S/处理器个数p不变
  - 等速度：保持单个处理器平均速度不变
  - 平均延迟：保持效率E不变，但通过W和p调整前后的平均延迟比值衡量。

可扩放性评测小结

- 三个标准本质上都是在讨论并行的额外开销T_o的影响
  - 等效率将T_o直接写在表达式中，但在实际中不一定容易测量
  - 等速度将T_o隐含在并行执行时间T的测量中
  - 平均延迟将T_o隐含在延迟的测量中

- 三个标准可以互相转换，本质上是等价的

第四章 并行计算性能评测

- 4.1 并行机的一些基本性能指标
- 4.2 加速比性能定律
  - 4.2.1 Amdahl定律
  - 4.2.2 Gustafson定律
  - 4.2.3 Sun和Ni定律
- 4.3 可扩放性评测标准
  - 4.3.1 并行计算的可扩放性
  - 4.3.2 等效率度量标准
  - 4.3.3 等速度度量标准
  - 4.3.4 平均延迟度量标准
- **4.4 基准测试程序**

基准测试程序（Benchmark）

- 用于测试和预测计算机系统的性能，为用户决定购买或使用哪种机器最满足应用需求提供决策依据
- 试图提供一种客观、公正的评价机器性能的标准
- 客观、公正并非易事，涉及很多因素
  - 体系结构、编译优化、编程环境、测试条件、求解算法等

基准测试程序

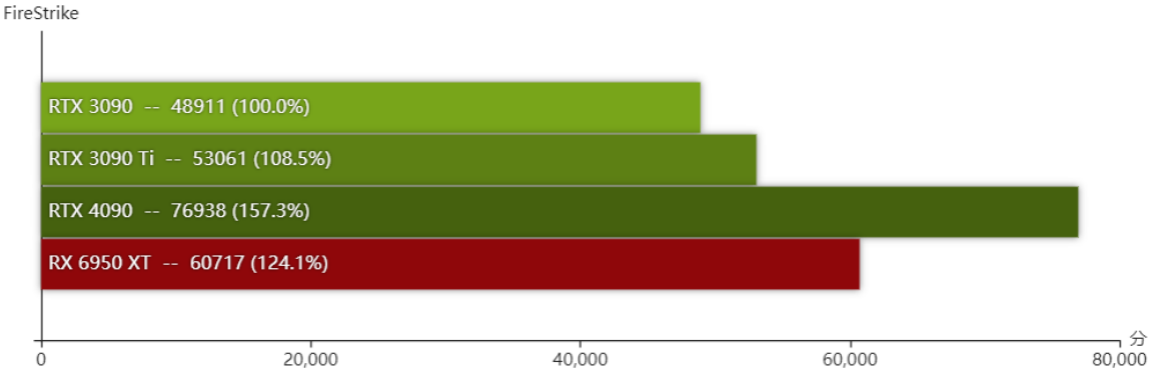
- Benchmark相比于直接运行测量应用程序的优点：
  - 从应用程序的工作负载中选取有代表性的的简化子集，时间和硬件资源开销通常更低
  - 避免复杂的移植（准备数据集、安装依赖库、配置运行环境等）
  - 避免商业软件许可费用昂贵、代码闭源、数据保密等问题

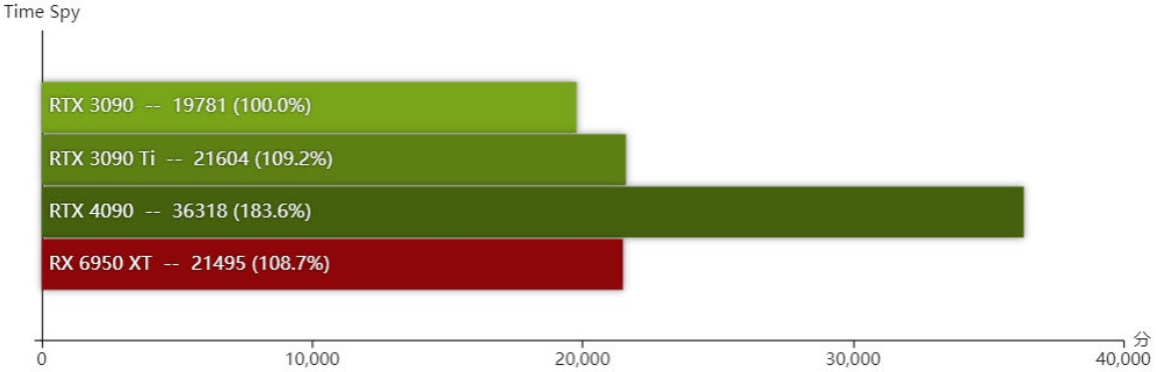
基准测试程序

https://topic.expreview.com/GPU/

基准测试程序

家用显卡经典基准测试程序是3DMark，通过运行一组3D游戏，根据过程中的延迟、帧率等指标，给出评价分数





典型的并行计算基准测试程序

- LinPACK
- NPB
- HPCG
- Graph500
- SPEC
- Rodinia
- MLPerf

LinPACK

- LinPACK是一个数值线性代数的软件库，由Jack Dongarra、Jim Bunch、clive Moler和Gilbert Stewart在20世纪70年代使用Fortran语言编写
- LinPACK使用BLAS(基本线性代数子程序)库来执行基本的向量和矩阵操作
- 包括: LinPACK100、LinPACK1000和HPL等多个版本

- 作用和意义
  - 提供一个衡量和比较各种计算系统在执行计算密集型浮点任务时性能的标准
  - 其中HPL用于TOP500超级计算机基准测试和排名

HPCG (High Performance Conjugate Gradients)

- HPCG由桑迪亚国家实验室的Michael Heroux和田纳西大学的Jack Dongarra和Piotr Luszczek提出
- 包括稀疏矩阵-向量乘法、全局点积、局部对称Gauss-Seidel平滑器、稀疏三角求解，以及多重网格预条件的共轭梯度算法
- 作用和意义
  - 模拟实际应用(例如稀疏矩阵计算)的数据访问模式，从而测试超级计算机的内存子系统和内部互连的限制对其计算性能的影响
  - HPCG旨在作为高性能LINPACK (HPL)基准的补充。自2014年以来，HPCG结果已经成为TOP500榜单的一部分，为超级计算机性能的评估提供了另一个重要维度

NPB (NAS Parallel Benchmark)

- NPB是一组针对高度并行超级计算机性能评估的基准测试，由位于NASA高级超级计算(NAS)部门(前身为NASA数值空气动力学模拟项目)开发和维护。NPB于1991年开发，并于1992年发布
- NPB源自计算流体动力学(CFD)应用程序，最早由5个内核和3个伪应用程序组成，现已经扩展到包括非结构化自适应网格、并行I/O、多区域应用程序和计算网格的新基准测试。

- 作用和意义

  并行计算研究中最常用的基准测试程序之一，简便易用

Graph500

- 由美国劳伦斯伯克利国家实验室等于2010年创建
- Graph500中有三个计算内核:
  - 第一个内核是生成图并将其压缩成稀疏结构CSR或CSC;
  - 第二个内核对一些随机顶点进行并行BFS搜索;
  - 第三个内核运行单源最短路径(SSSP)
- 作用和意义

  Graph500的主要指标是GTEPS，即每秒处理的十亿图边数。重点是系统如何有效地处理数据密集的图计算问题，而不是TOP500关注的计算密集的浮点运算

SPEC CPU

- SPEC CPU由Standard Performance Evaluation Corporation (SPEC) 创建
- SPEC CPU 2017 包含43个基准测试，分为四个套件:
  - SPECspeed®2017 Integer和SPECspeed®2017 Floating Point套件用于比较计算机完成单个任务的时间。
  - SPECrate®2017 Integer和SPECrate®2017 Floating Point套件测量每单位时间的吞吐量或工作
- 作用和意义
  - 用于评估CPU性能，涵盖浮点和整数计算密集型应用程序
  - 使用从真实用户应用程序开发的工作负载，在最广泛的实际硬件范围内提供计算密集型性能的比较度量
  - 编译器、体系结构等领域最常用的基准测试程序之一

Rodinia

- 由S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, 和 K. Skadron在2009年的IISWC首次发布
- 包括针对多核CPU和GPU平台的应用程序和内核

- 作用和意义
  - 定量测量并行通信模式、同步技术、功耗以及数据布局和带宽限制的影响
  - 填补之前基准测试中关于GPU等加速器的空白，推动异构计算平台的研究和发展
  - GPU并行计算研究中最常用的基准测试程序之一

MLPerf

- MLPerf是一个由MLCommons推动的开放性行业标准基准测试套件
- 包括MLPerf Training、MLPerf Inference和MLPerf HPC
  - MLPerf Training 测量跨9个不同用例训练模型的时间，包括大型语言模型(llm)、图像生成、计算机视觉、医学图像分割、语音识别和推荐
  - MLPerf Inference 使用七种不同的神经网络度量推理性能，包括llm、自然语言处理、计算机视觉和医学图像分割
  - MLPerf HPC 在四种不同的科学计算用例中测量训练性能，包括气候、大气、河流识别、宇宙学参数
- 作用和意义
  - 评估和比较机器学习硬件、软件和服务的性能

主要是深度学习

基准测试程序开发的困难

1. 人工开发

- 需要同时懂并行计算和领域应用（物理、化学、人工智能……）的专家

2. 滞后于应用发展

- 应用本身的重要性更高，分配到基准测试程序开发上的人力物力有限

AI模型：	AI Bench:
AlexNet: 2012	MLPerf: 2018
ResNet: 2015	MLBench: 2018
Transformer: 2017	AIBench: 2018
	SuperBench: 2021

3. 性能特征准确性

- 应用和系统都十分复杂，两者之间的交互更加复杂
- 专家开发的基准测试程序也不一定能准确、全面地反映性能特征

基准测试程序

- 合成（synthetic）基准测试程序
  - 不需要理解应用程序的算法、逻辑
  - 由无意义的、具有特定性能特征的程序片段组成
  - 典型代表：Whetstone、Dhrystone
  - 缺点：和真实应用脱节，导致评测结果对用户没有参考意义

节省了领域专家

基准测试程序

- 用于合成基准测试程序的6个程序片段示例

```
1  //block type 1: add
2  //int i1,i2,i3...in;
3  //double d1,d2,d3...dn;
4  i1 = i2+i3+...+in;
5  d1 = d2+d3+...+dn;
6
7  //block type 2: mul
8  //int i1,i2,i3...in;
9  //double d1,d2,d3...dn;
10 i1 = i2*i3*...*in;
11 d1 = d2*d3*...*dn;
12
13 //block type 3: div
14 //int i1,i2,i3...in;
15 //double d1,d2,d3...dn;
16 i1 = i2/i3/.../in;
17 d1 = d2/d3/.../dn;
```

```
18 //block type 4: branch
19 ibranch = rand()%(1<<20);
20 for (register long j = 0;j < 20;j++)
21     if ((ibranch>>j)&1)
22         (add or mul or div);
23 //block type 5: memory access
24 //register:i0,i1,j
25 //a is array of int or double
26 //memory_access_step in
27 // [2,4,8,16,32,...4096]
28 for (j = 0;j < iteration;j++){
29     a[i0]=(add or mul or div);
30     i0 += memory_access_step;
31 }
32 //block type 6 : empty cycle for
       branch
33 for (long i = 0;i<iteration;i++);
```

基准测试程序

- 自动合成基准测试程序
  - 不需要理解应用程序的算法、逻辑
  - 由无意义的、具有特定性能特征的程序片段组成
  - 程序自动分析应用行为和性能、自动写出基准测试程序的代码
  - 基本合成步骤：
    1. 跟踪记录应用程序的行为序列
    2. 将行为序列压缩编码，提取和表达关键行为模式
    3. 将行为模式重新转换为可执行代码

节省了领域专家

节省了并行专家

保持和应用程序的关联