# 高性能计算技术

## 第四讲 并行计算性能评测

v ὦ
kjhe@scut.edu.cn

华南理工大学计算机学院

# 内容概要

- 什么是并行计算性能评测？
- 加速比性能定律
- 可扩展性评测标准
- 基准测试程序

# 什么是性能评测（Performance Evaluation）？

- 性能评测：性能评价和性能分析
- 性能评价和性能分析可以揭示高性能计算机、并行算法和并行应用程序的性能特点和性能瓶颈，指导高性能计算机、并行算法和应用程序的设计与改进
- 性能评测的目的：提高性能
  - 性能预测
  - 性能诊断/优化

# 并行计算性能评测的意义

- 发挥高性能计算机长处，提高高性能计算机的使用效率
- 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、公正的评价高性能计算机的标准

# 如何进行并行计算性能评测？

- 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；并行机的可用性与好用性以及机器成本、价格与性/价比
- 算法级性能评测：加速比、效率、扩展性
- 程序级性能评测：基准程序（Benchmark）

# 计算机的性能

- 性能（Performance）：通常是指机器的速度，它是程序执行时间的倒数
- 程序执行时间（Elapsed time）：是指用户的响应时间（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）
  - 机器的时钟周期为TC，程序中指令总条数为IN，执行每条指令所需的平均时钟周期数为CPI（Cycle Per Instruction），则一个程序在CPU上运行的时间 为：IN×CPI×TC
- CPU时间：它表示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

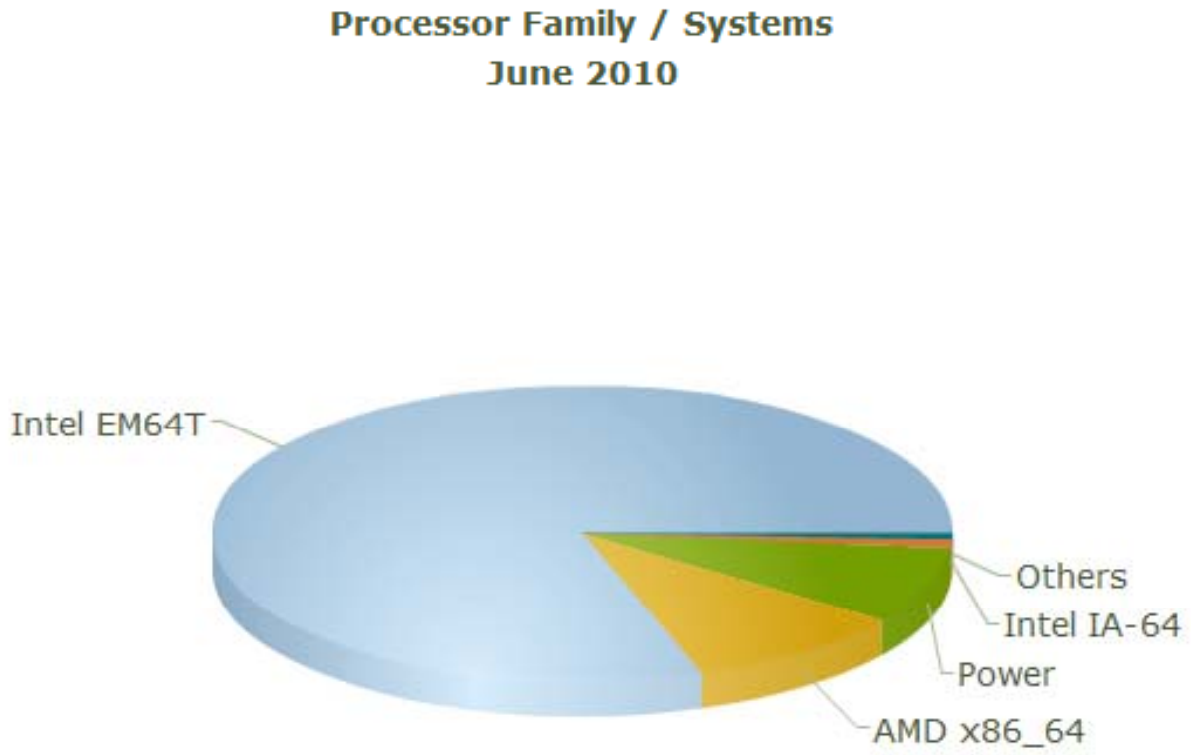
# 工作负载（Workload）和速度

- 工作负载：
  - 百万浮点计算数MFLOP (million floating point operations)
  - 百万指令数MI (million instruction)
- 速度：
  - 每秒百万浮点计算数MFLOPS (million floating point operations per second):
    - Flops/sec = number of pipes (flops/cycle) *clock speed (cycles/sec) * processors

| CPU | Flops/Cycle | CPU | Flops/Cycle | CPU | Flops/Cycle |
|---|---|---|---|---|---|
| IBM Power | 4 | Ultra SPARC | 2 | AMD Opteron | 2、4 |
| PA-RISC | 4 | SGI MIPS | 2 | Xeon | 2 |
| Alpha | 2 | Itanium | 4 | Xeon EM64T | 4 |

  - 每秒百万指令数目MIPS (million instructions per second):
    - instruction count/(execution time × $10^6$)

# Top 500中的处理器家族




Intel EM64T：80.2%， AMD x86_64：9.8%，Intel IA-64： 1.0% （2010.6）

# 并行机的性能指标

| 名 称 | 符 号 | 含 意 | 单 位 |
|---|---|---|---|
| 机器规模 | $n$ | 处理器的数目 | 无量纲 |
| 时钟速率 | $f$ | 时钟周期长度的倒数 | MHz |
| 工作负载 | $W$ | 计算操作的数目 | Mflop |
| 顺序执行时间 | $T_1$ | 程序在单处理机上的运行时间 | s（秒） |
| 并行执行时间 | $T_n$ | 程序在并行机上的运行时间 | s（秒） |
| 速度 | $R_n = W/T_n$ | 每秒百万次浮点运算 | Mflop/s |
| 加速 | $S_n=T_1/T_n$ | 衡量并行机有多快 | 无量纲 |
| 效率 | $E_n = S_n/n$ | 衡量处理器的利用率 | 无量纲 |
| 峰值速度 | $R_{peak} = n R'_{peak}$ | 所有处理器峰值速度之积，<br>$R'_{peak}$为一个处理器的峰值速度 | Mflop/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 |
| 渐近带宽 | $r_\infty$ | 传送长消息通信速率 | MB/s |

# 算法级性能评测

- 加速比（Speedup）：对于一个给定的应用，并行算法（或并行程序）相对于串行算法（或串行程序）的性能提高程度
  - Amdahl 定律
  - Gustafson定律
  - Sun Ni定律
- 可扩展性（ Scalability ）：当系统和问题规模增大时，可维持相同性能的能力，即指应用、算法和结构能否充分利用不断增长的处理器的能力
  - 等效率度量标准
  - 等速度度量标准
  - 平均延迟度量标准

# 并行性能测度

- 加速比（Speedup）：性能的提高通常表现为执行速度的加快

$$S_p = \frac{\text{Sequential Execution Time}}{\text{Parallel Execution Time}}$$

- 并行效率（Parallel Efficiency）：处理器的利用率
  - Efficiency = (Sequential execution time) / (Number of processors * Parallel execution time)
  - Efficiency = Speedup / Number of processors

# 内容概要

- 什么是并行计算性能评测？
- 加速比性能定律
- 可扩展性评测标准
- 基准测试程序

# 性能提高的测度－Speedup

$$performance = \frac{work}{time}$$

$$speedup(p) = \frac{performance(p)}{performance(1)} = \frac{work(p)/time(p)}{work(1)/time(1)}$$

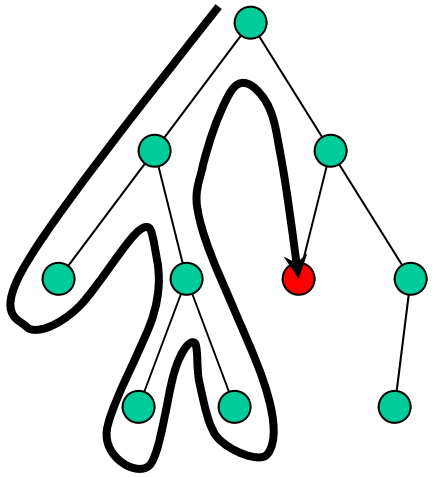
多处理器性能的基本测度

# 加速比的特性

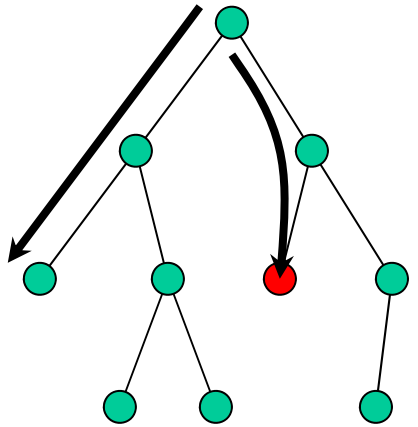
- 一般地，我们想象“2>1+1>1”
  - 基本原因：通信、同步、协调开销过大
- 不幸的是，也可能会有“1+1<1”！（学习并行计算技术的目的之一就是要理解这种情况发生的条件，避免其发生）
- 是否存在1+1>2？
  - 超线性加速比（Superlinear Speedup）

# 超线性加速比

- 例如在搜索问题时的不同搜索策略



**Sequential search – 12 moves**



**Parallel search – 2 moves**

# 超线性加速比

- 并行计算机有更多的内存。由于高速缓存（cache）的影响导致的“超线性”加速现象
  - 问题的数据规模大 – 一台处理机的cache放不下，多台处理机的cache分摊能够容纳；命中与否对应的访问时间有很大差别。
  - 访问模式随机 – 一台处理机也就没有“局部性”可利用
  - 数据并行问题 – 多处理机之间没有太多的通信，在这个数据集上的反复循环迭代操作

# 不同约束条件下的加速比

- 固定问题规模：Problem constrained (PC）
- 固定时间：Time constrained (TC)
- 固定存储：Memory constrained (MC)

# 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,
  - $f$：串行分量比例（$f = W_s/W$，$W_1 = W_s + W_o$）；
  - $W_P$：应用程序中可并行化部分，1-$f$为并行分量比例
  - $W_s + W_p = W$
- $T_1$：串行执行时间，$T_p$：并行执行时间；
- $S$：加速比
- 出发点：
  - 固定不变的计算负载，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
    - 例子：视频压缩，搜索引擎，OLTP等

# 固定规模（ Fixed-Size）的加速比

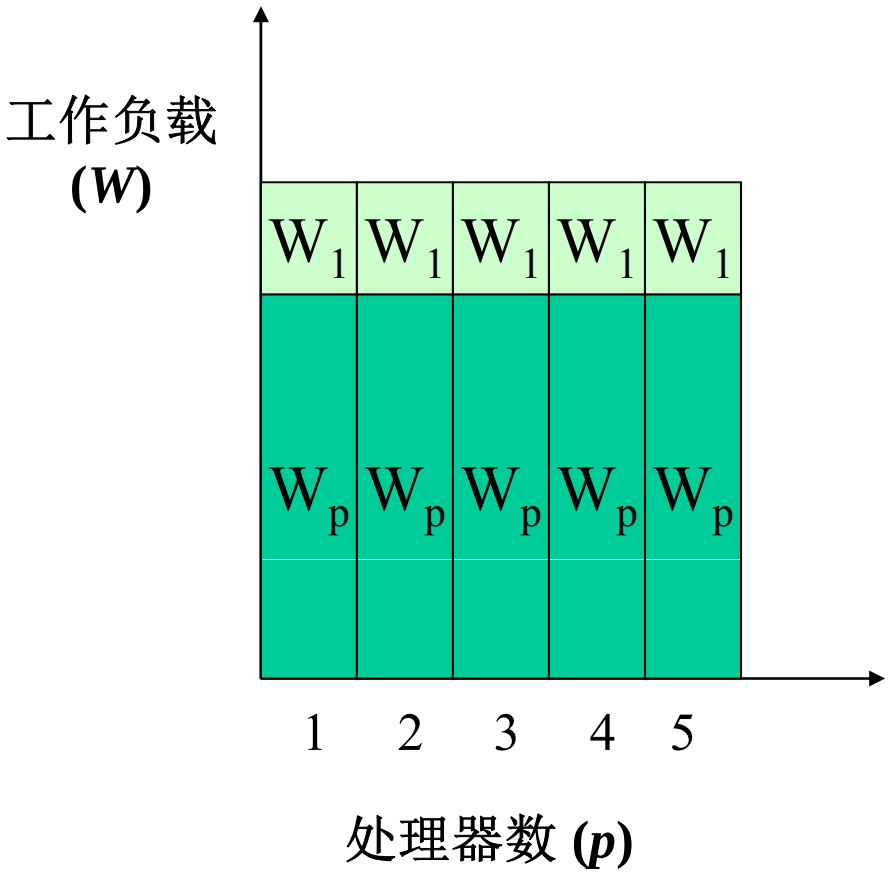
$$performance = \frac{work}{time}$$

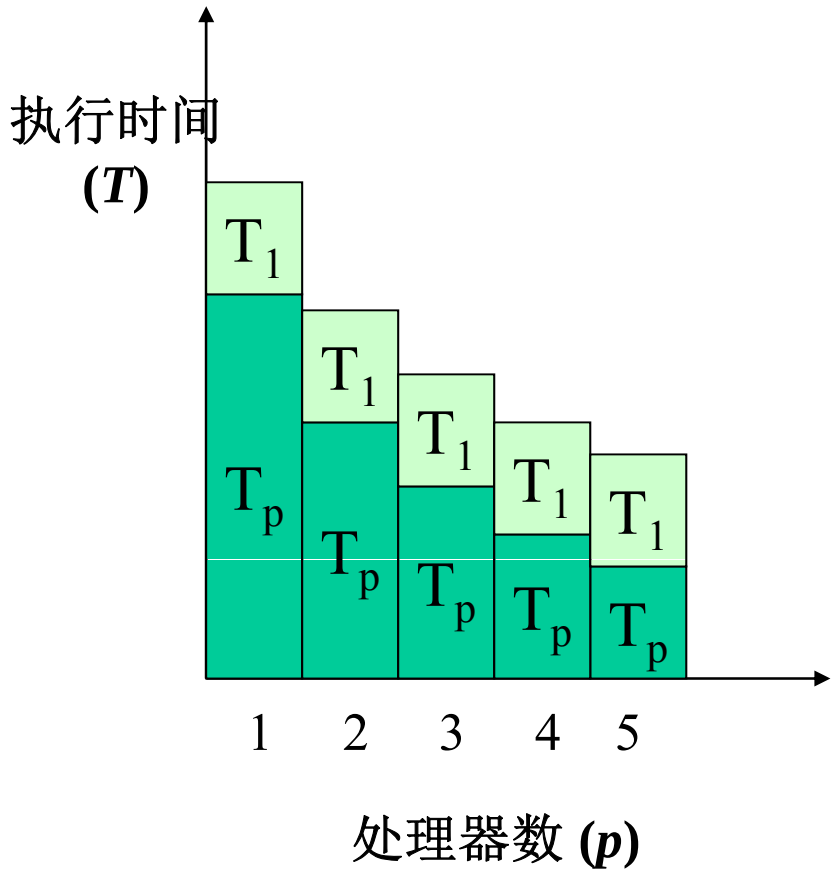
当工作固定

$$speedup(p) = \frac{performance(p)}{performance(1)} \Rightarrow speedup(p) = \frac{time(1)}{time(p)}$$

多处理器性能的基本测度

# 固定规模的加速比


工作负载
(W)
W1 W1 W1 W1 W1
Wp Wp Wp Wp Wp
1 2 3 4 5
处理器数 (p)


执行时间
(T)
T1
Tp
T1
Tp
T1
Tp
T1
Tp
T1
Tp
1 2 3 4 5
处理器数 (p)

# Amdahl定律

$$S_{PC} = \frac{W_S + W_p}{W_S + W_P / p} = \frac{f + (1-f)}{f + \frac{1-f}{p}} = \frac{p}{1 + f(p-1)} \rightarrow \frac{1}{f} \quad as\ p \rightarrow \infty$$

Portion of sequential computation

# of processors

式3.5

例如：如果 $f$=10%, 则最大的加速比是10，无论用多少个个处理器

# 增强的Amdahl定律

考虑开销$W_o$，包括并行和通信的开销

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

$$\rightarrow \frac{1}{f + \frac{W_o}{W}} \quad as\ p \rightarrow \infty$$

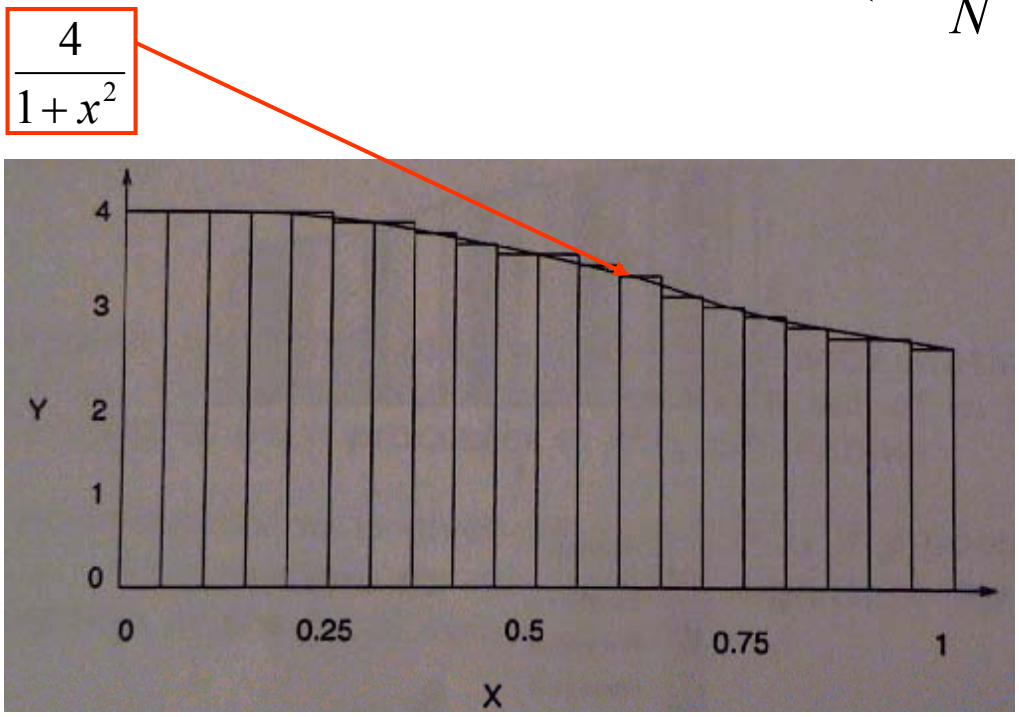
式3.8

# 例子：计算π

- 如何用数值方法计算π=3.1415…？ (P311)

$$\pi = \int_0^1 \frac{4}{1+x^2}dx \approx \sum_{0 \le i < N} \frac{4}{1+(\frac{i+0.5}{N})^2} \cdot \frac{1}{N}$$

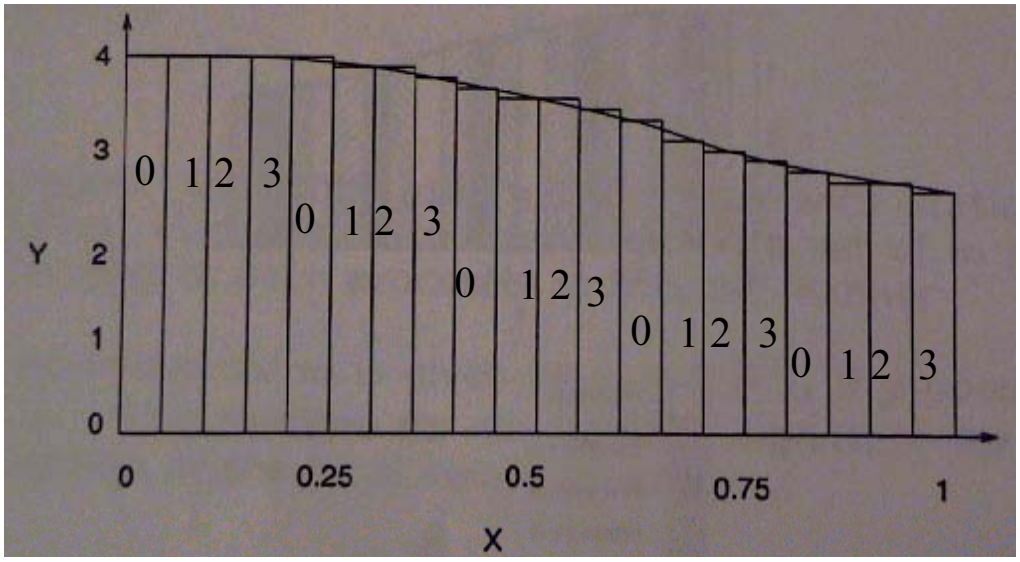
串行算法




```
computepi()
{
    h=1.0/n;
    sum =0.0;
    for (i=0;i<n;i++) {
        x=h*(i+0.5);
        sum=sum+4.0/(1+x*x);
    }
    pi=h*sum;
}
```

# 计算π：并行算法

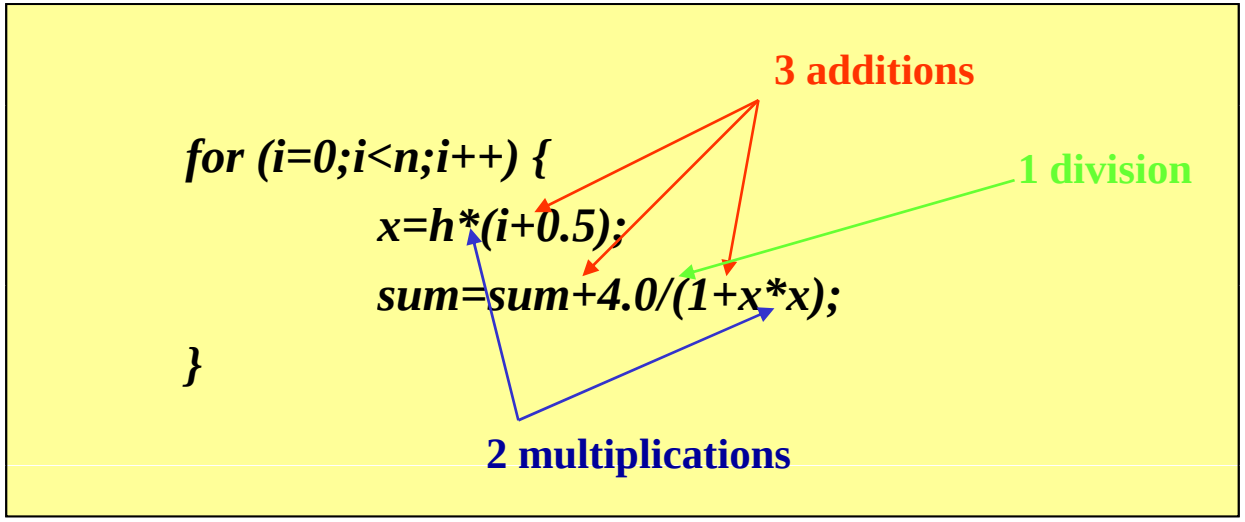
- 循环分配计算任务，每个处理器计算$n/p$ 个点
- 将$p$个处理器上的本地和相加就得到π的值



# 计算π：分析

- 假设π的计算是在 $n$ 个点上进行的。串行算法对x轴上每个点都要进行6次操作（两次乘法、一次除法、三次加法）
- 因此，对于$n$个点，串行算法要执行的操作数是（$t_0$是单位计算时间）：

$$T_s = 6n * t_0$$



# 计算π：分析（2）

- 使用$p$个处理器的并行算法，每个处理器计算 $m$ 个点

$$m \leq \frac{n}{p} + 1$$

- 因此计算π的并行算法的运行时间是：

$$T_p = 6m * t_0 = (6\frac{n}{p} + 6)t_0$$

# 计算π：分析（3）

- 最后得到π，还要进行本地和的累加，这可以通过基于树的合并算法，可在$\log_2(p)$ 个步骤内完成
- 因此并行算法的总的计算时间是（$t_c$是单位通信时间）：

$$T_p = 6m * t_0 = (6\frac{n}{p} + 6)t_0 + \log(p)(t_0 + t_c)$$

- 并行算法的加速比是：

$$S_p = \frac{T_s}{T_p} = \frac{6n}{6\frac{n}{p} + 6 + \log(p)(1 + t_c / t_0)}$$

# 计算π：分析（4）

- 改写上式：

$$S_p = \frac{p}{1+\frac{p}{n}+\frac{pc\log(p)}{6n}} \Rightarrow S_p = \frac{p}{1+(p-1)f(n,p)}$$

- 可以得到Amdahl定律的串行分量$f$(n,p)为：

$$f(n,p) = \frac{p}{(p-1)n} + \frac{pc\log(p)}{6n(p-1)}$$

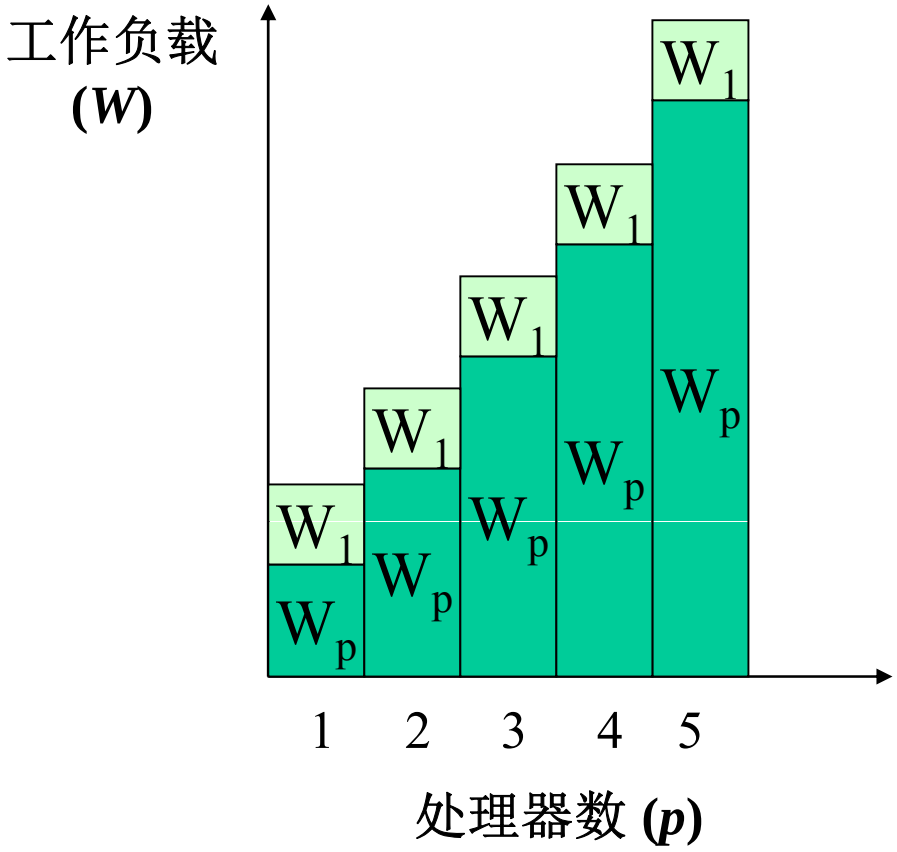
- 这样的并行算法是有效（effective）的，因为：

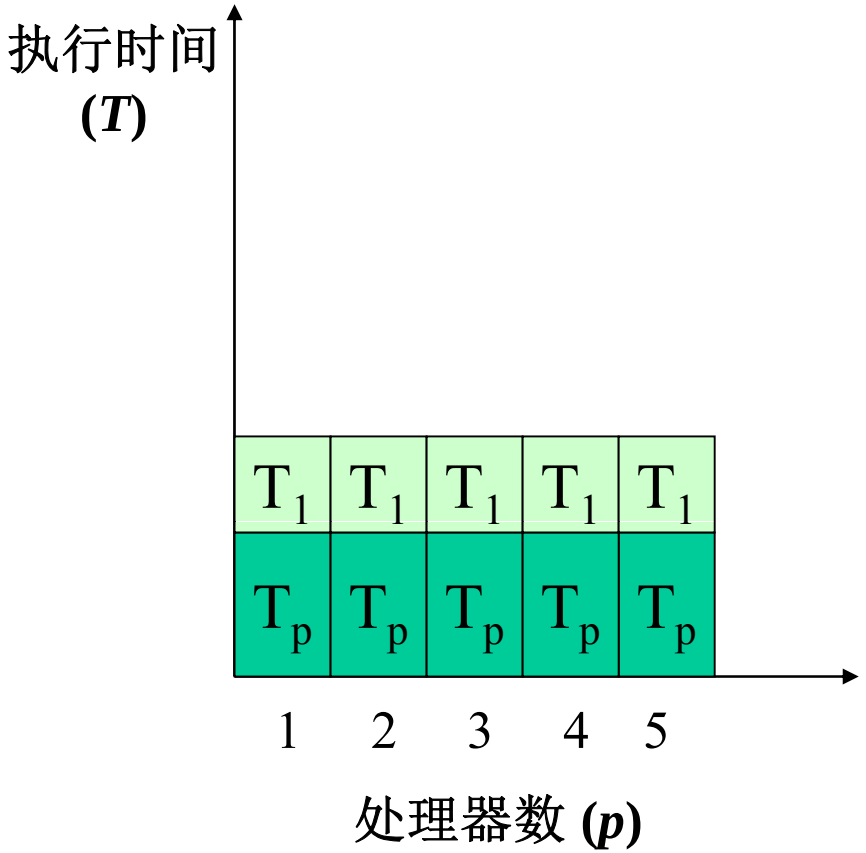
$$f(n,p) \to 0 \text{ as } n \to \infty \text{ for fixed } p$$

# Gustafson定律

- 许多重要应用要求在固定时间内能处理的数据越多越好（例如数值天气预报）；而在固定时间里，依靠增加处理器数总可以指望能处理更多的数据（尽管它们之间的关系随问题而变）
- 其他例子：
  - 结构分子中的有限元方法FEM（Finite Element Method）
  - 流体力学的有限差分方法FDM（Finite Difference Method）

# 时间固定（Fixed-Time）的加速比



# Gustafson定律

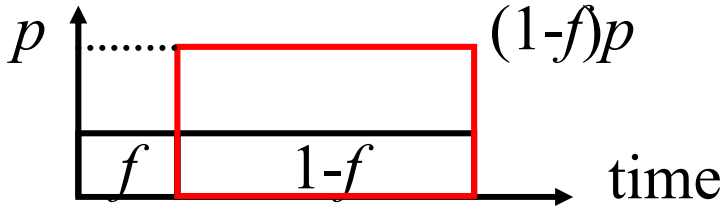
- Gustafson加速定律：

$$S_{TC} = \frac{W_S + pW_P}{W_S + p \cdot W_P / p} = \frac{W_S + pW_P}{W_S + W_P}$$

式3.9

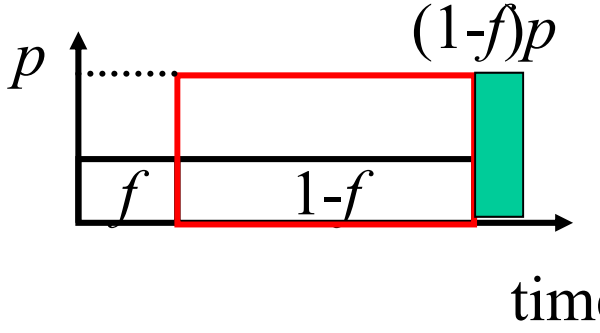
$$S_{TC} = f + p\,(1\text{-}f) = p + f\,(1\text{-}p) = p\text{-}f\,(p\text{-}1)$$

式3.10



# Gustafson定律

- 考虑并行开销$W_o$：




$$S_{TC} = \frac{W_S + pW_P}{W_S + W_P + W_o} = \frac{f + p(1-f)}{1 + W_o / W}$$

式3.11

# Sun & Ni定律

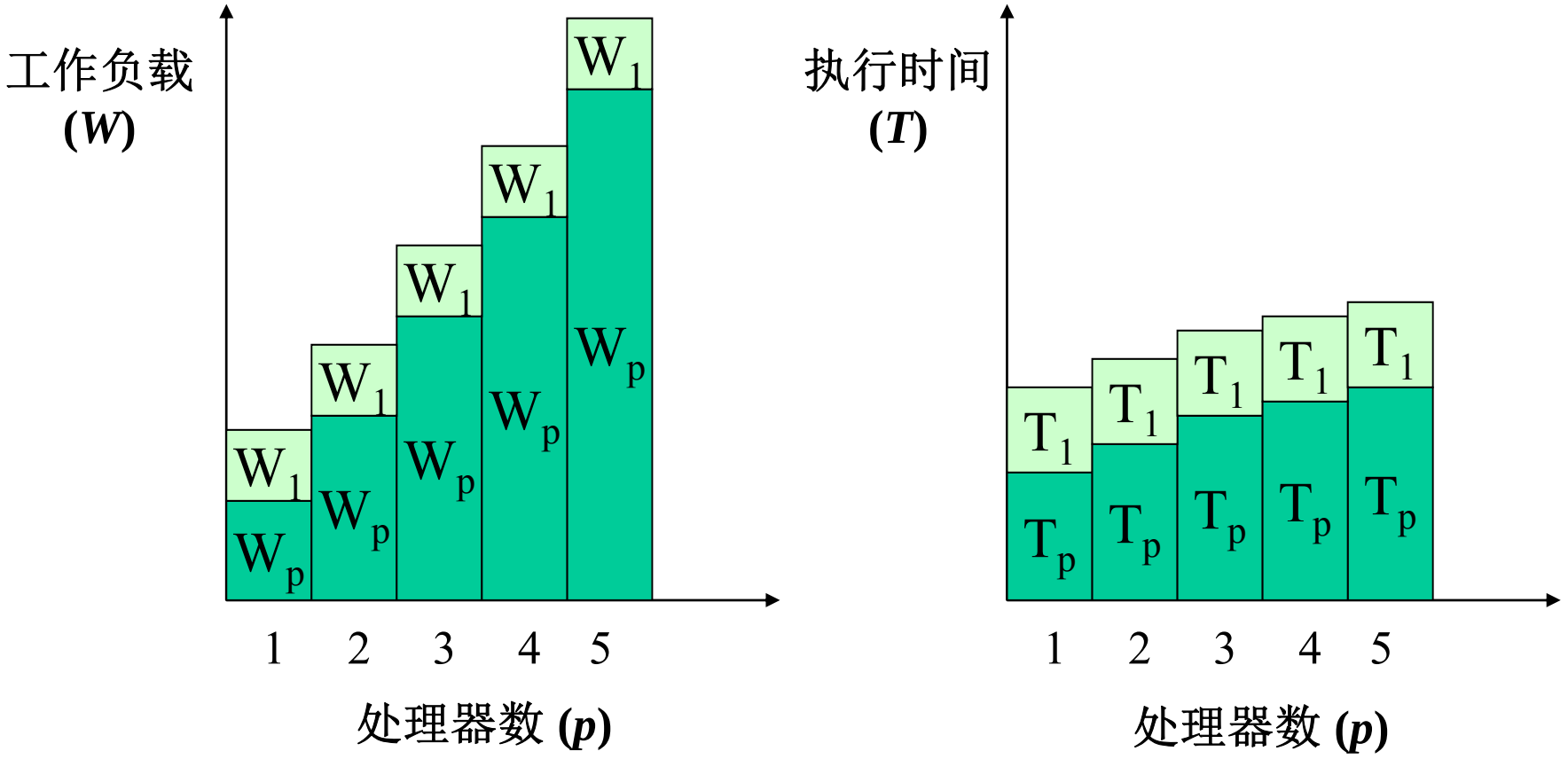
- 只要存储空间许可，应尽量增大问题规模以产生更好和更精确的解（此时可能使执行时间略有增加）
  - 例如N体（N-body）问题
- 假定在单节点上使用了全部存储容量$M$并在相应于$W$的时间内求解之，此时工作负载：

$$W = fW + (1-f)W$$

- 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$G(p)$表示存储容量增加到$p$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（Memory-Boundary）的加速比

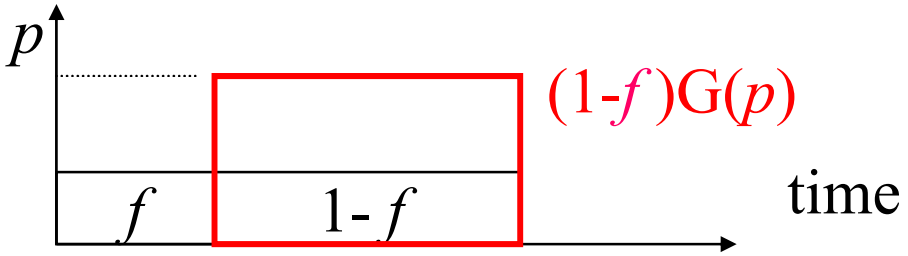


# Sun & Ni定律

- 存储受限的加速公式：

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

式3.13




- 并行开销$W_o$:

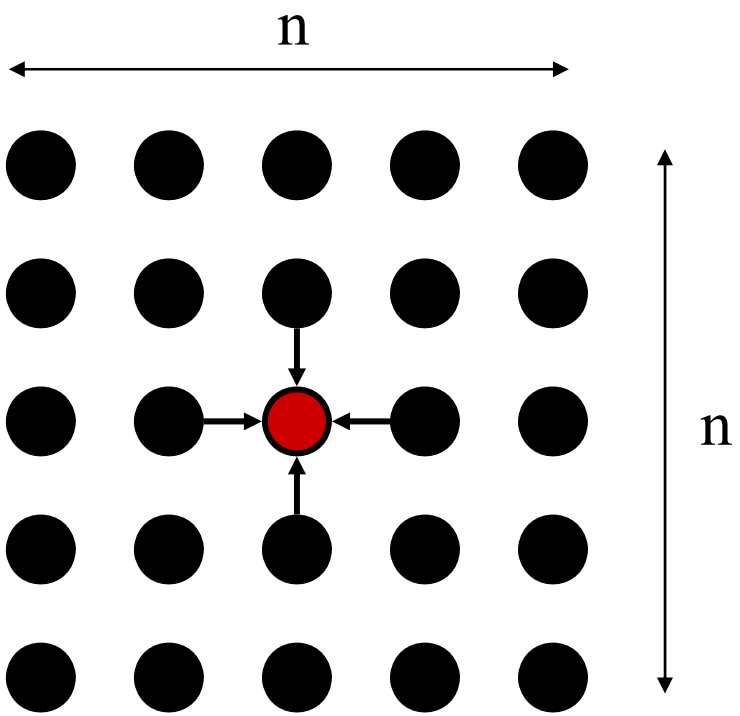
$$S_{MC}=\frac{fW+(1-f)WG(p)}{fW+(1-f)G(p)W/p+W_o}=\frac{f+(1-f)G(p)}{f+(1-f)G(p)/p+W_o/W}$$

式3.14

# Sun Ni定律特性

- 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 加速为高。

# 例子：方程求解器（Equation Solver）




```
procedure solve (A)
   …
   while(!done) do
      diff = 0;
      for i = 1 to n do
         for j = 1 to n do
            temp = A[i, j];
            A[i, j] = …
            diff += abs(A[i,j] – temp);
         end for
      end for
      if (diff/(n*n) < TOL) then done =1 ;
   end while
end procedure
```

**A[i,j] = 0.2 * (A[i, j] + A[i, j-1] + A[i-1, j] + A[i, j+1] + A[i+1, j])**

# 工作负载

- 基本性质（问题规模：$n$）：
  - 存储要求：$O(n^2)$
  - 计算复杂度： $O(n^3)$，假设收敛所需要的迭代次数是 $O(n)$
- 如果加速比等于 $p$，讨论以下三种情况下使用$p$个处理器时可处理的问题规模（Size）、存储需求和计算复杂度：
  - 问题规模受限（PC）
  - 时间受限（TC）
  - 存储受限（MC）

# 方程求解器的问题规模

**PC：** **fixed**

**TC：** $n^3 \times p = k^3 \ \therefore k = \sqrt[3]{p} \times n$ 

**MC：** $n^2 \times p = k^2 \ \therefore k = \sqrt{p} \times n$ 

# 方程求解器的存储需求

**PC :** $\frac{n^2}{p}$ ↓

**TC :** $\frac{k^2}{p} = \frac{(n \times \sqrt[3]{p})^2}{p} = \frac{n^2}{\sqrt[3]{p}}$ , $n^3 \times p = k^3$ ↓

**MC :** $n^2 \times p$ ↑

# 方程求解器的时间复杂度

**PC :** $\frac{n^3}{p}$ ↓

**TC :** $n^3$

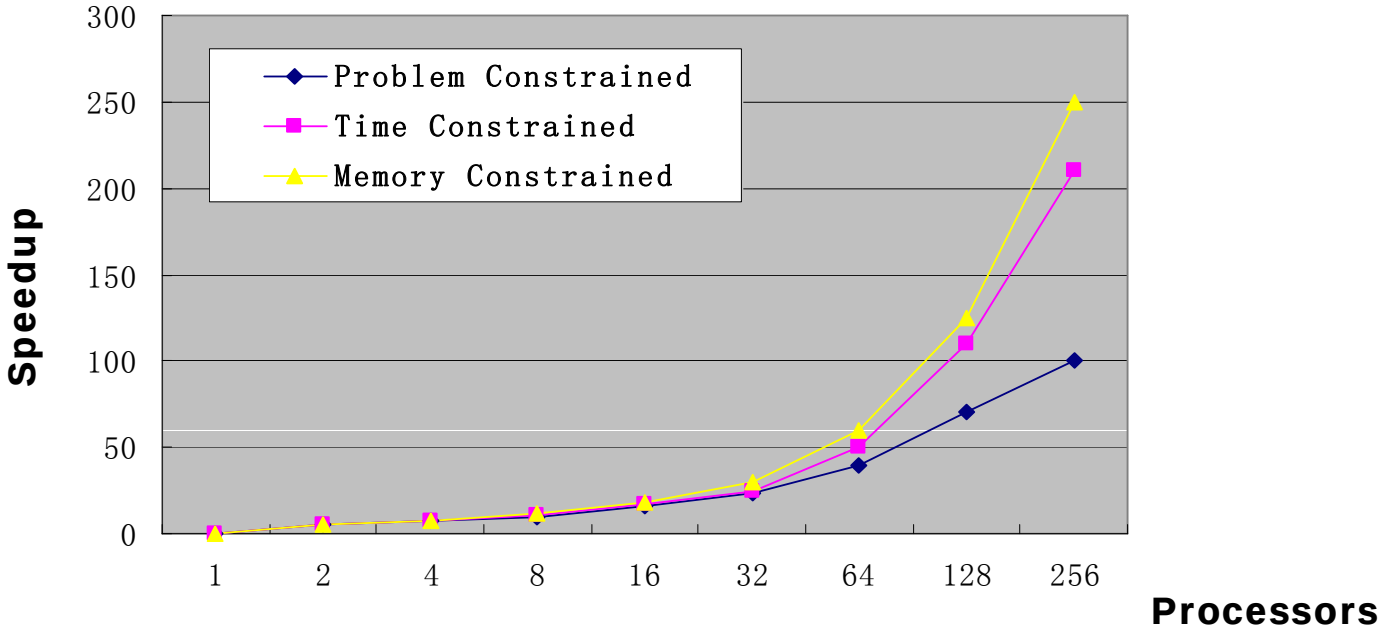
**MC :** $\frac{(n\sqrt{p})^3}{p} = n^3\sqrt{p}$ ↑

**Sequential time complexity**

$k^3 = (n \times \sqrt{p})^3$ , $n^2 \times p = k^2$

# 例子：STAP APT

## 加速比的模型的比较




《可扩展并行计算》 例 **3.22**， 图**3-9**

# 加速比讨论

- 参考的加速比经验公式： $p/\log p \le S \le P$
  - 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 ： $S = 1 / C(p)$ (式3.16)
- 超线性加速
- 绝对加速（科学研究）： 最佳并行算法与串行算法
- 相对加速（工程应用）： 同一算法在单机和并行机的运行时间

# 内容概要

- 什么是并行计算性能评测？
- 加速比性能定律
- 可扩展性评测标准
- 基准测试程序

# 可扩展性

- 影响加速比的因素：系统规模与问题规模
  - 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）
- 增加系统规模（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放性这一指标。

# 可扩展性（2）

- 经常，在一定规模下求解某个问题的“执行时间”对理解系统的“性能”来说还不够
- 针对并行计算机系统的性能，关心它在系统规模和应用数据规模变化时的执行时间往往更有意义
- 可扩展性（Scalability）：系统的“规模性能”（通常译为“可扩展性”，有时也称为“可扩放性”，“可伸缩性”）

**{性能,系统规模,数据规模}的综合测度**

# 对scalability的追求

- 当系统规模扩大时，希望系统的处理能力能相应增强
- 能力增强
  - 同样的时间能处理更多的数据
  - 处理同样的数据花更少的时间
- 相应
  - 成线性比例，对数比例，…

- 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）就反映了可扩放的程度。

# 可扩展性评测的三种量化方式

- 等效率测度（Efficiency Metrics）
  - 效率：加速比/处理器数
  - 简单情况下能得分析结果
- 等速度测度（Speed Metrics）
  - 速度：每秒处理的数据量
  - 便于通过实验数据得到结果
- 平均时延测度（Latency Metrics）
  - 时延：理想并行时间与实际并行时间的差距
  - 便于通过实验数据得到结果

# 等效率函数

**Speedup:**

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

式3.18

**Efficiency:**

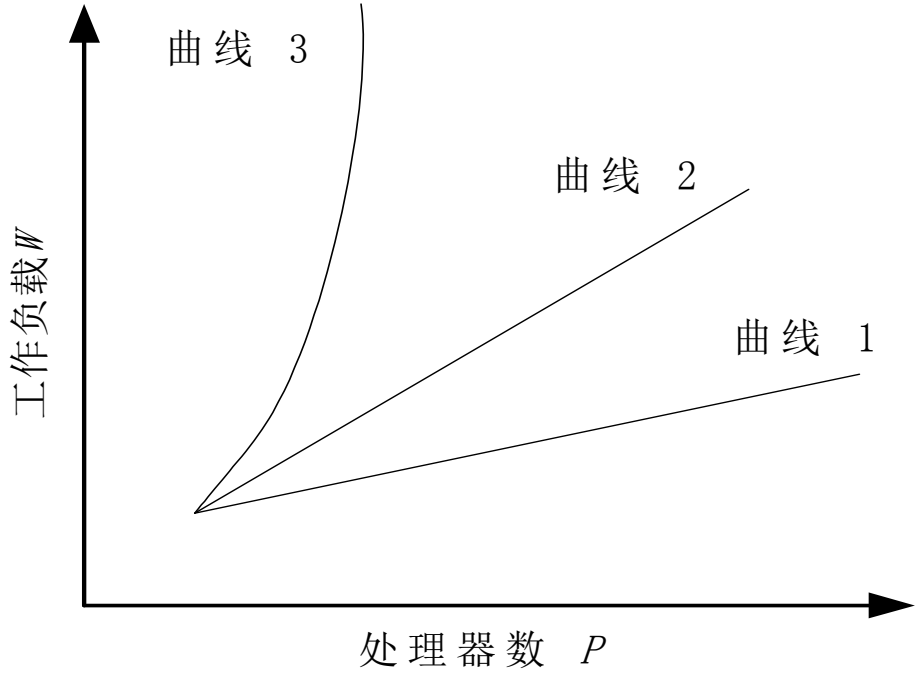
$$E = \frac{S}{P} = \frac{1}{1 + \frac{T_o}{T_e}} = \frac{1}{1 + \frac{W_o}{W}}$$

式3.19

- 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）（Kumar 1987）

# 等效率函数曲线

- 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。



# 可扩展性例子

- 例子：用p个处理器算N个数之和
- $T_p = N/p + 2\log p$, $T_s = N$
  - $p = 4, N = 64, E = T_s/pT_p = 64/4(16+4) = 64/80 = 0.8$
  - $p = 8, N = 192, E = 192/8(24+6) = 192/240 = 0.8$
- 求等效率函数 (E=0.8) ?
- E=0.8的等效率函数是$8p \log p$
- 如果 $N<8p \log p$
  - $p = 8, N = 144, E = 144/8(18+6) = 144/192 = 0.75$
- 如果 $N>8p \log p$
  - $p = 8, N = 384, E = 352/8(44+6) = 352/400 = 0.88$

# 等速度函数

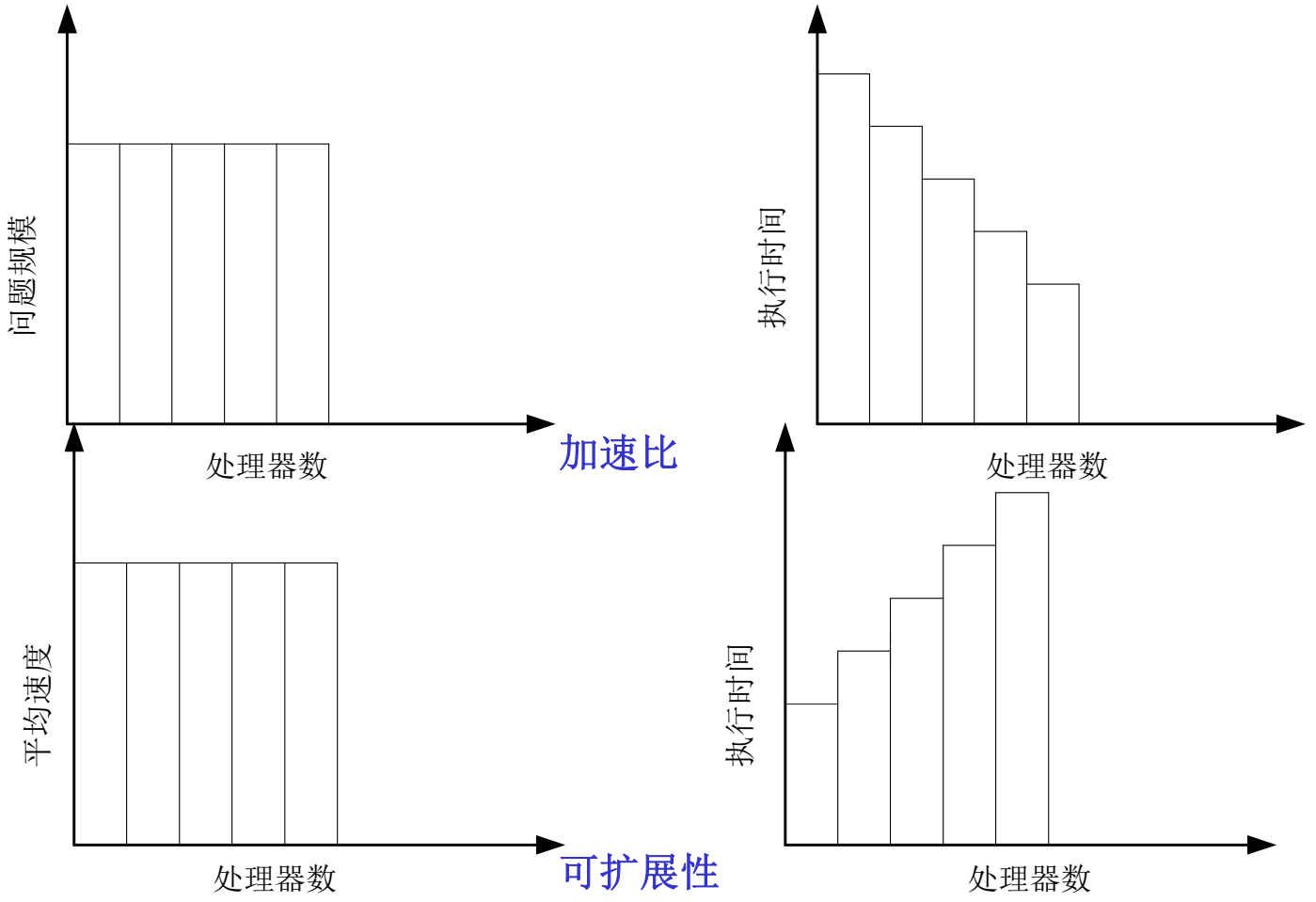
- $p$ 表示处理器个数，$W$表示要求解问题的工作量或称问题规模（在此可指浮点操作个数），$T$为并行执行时间，定义并行计算的速度$V$为工作量$W$除以并行时间$T$
- $p$个处理器的并行系统的平均速度定义为并行速度$V$除以处理器个数 $p$：

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

- $W$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。则平均速度可扩放度量标准：

$$\Psi(p,p') = \frac{W/p}{W'/p'} = \frac{p'W}{pW'} = \frac{T}{T'}$$

# 等速度度量标准



# 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

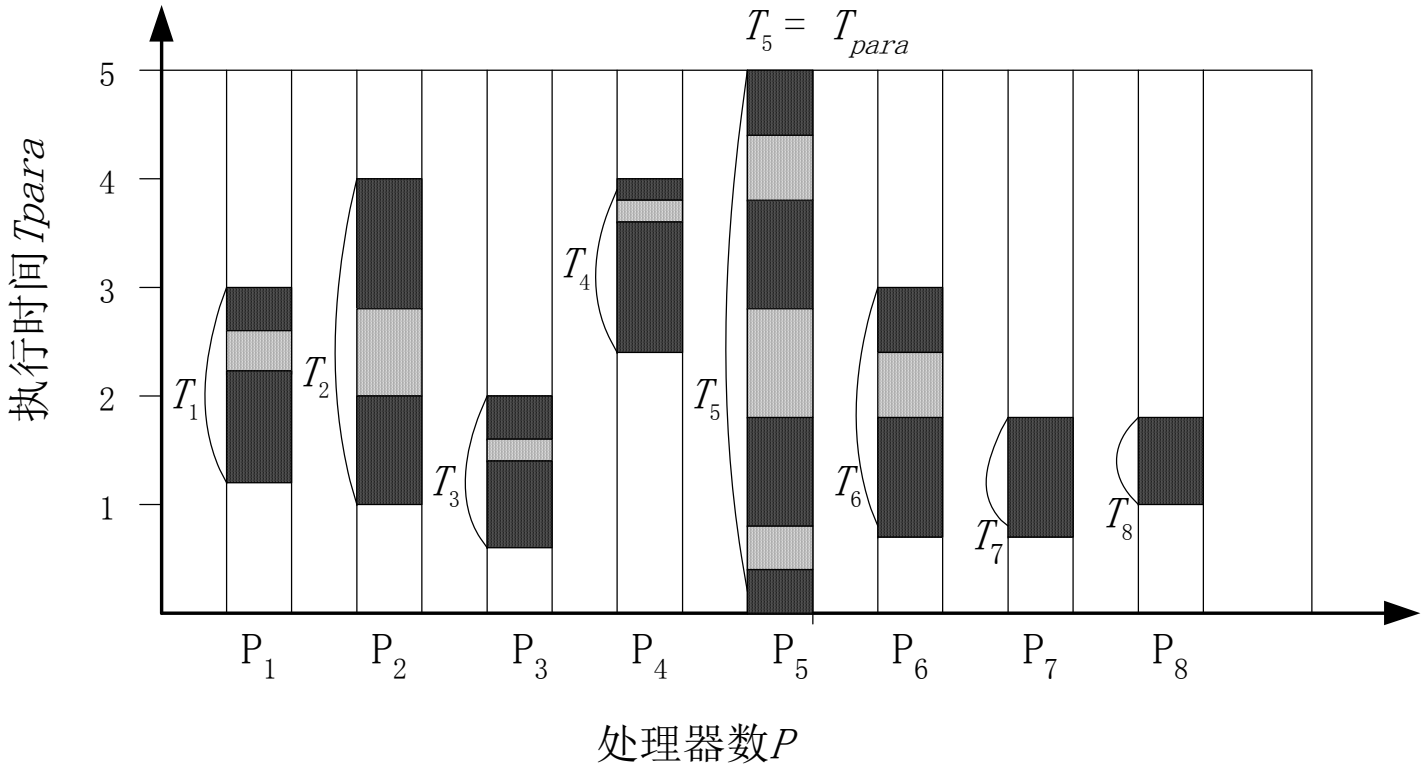
$$\overline{L}(W,p)=\sum_{i=1}^{p}\left(T_{para}-T_i+L_i\right)\Big/p=T_{para}-T_{seq}/p$$

$$pT_{para}=T_o+T_s,\quad T_o=p\overline{L}(W,p)$$

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

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

# 平均延迟度量标准




启动前与结束后空闲时间　　开销延迟 $L_i$　　运行时间 $T_i$

# 各种度量标准的优缺点

- 等效率测度
  - 优点：简单、可定量计算、参数少
  - 缺点：如果$T_o$无法计算出（在共享存储并行机中）
- 等速度度量
  - 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
- 平均延迟度量
  - 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

# 可扩展性评测标准

- 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

# 内容概要

- 什么是并行计算性能评测？
- 加速比性能定律
- 可扩展性评测标准
- 基准测试程序

# 程序级性能评测

- 基准测试程序（Benchmark）
  - 一组标准的测试程序
  - 提供一组控制测试条件
  - 步骤的规则说明（测试平台环境、输入数据、输出结果和性能指标等）
- 基准测试程序的分类
  - 宏观测试程序（Macro-benchmark）：计算机系统作为一个整体来测试其性能
  - 微观测试程序（Micro-benchmark）：测试机器的某一特定方面的性质

# 测试程序分类

- 按生成方式：真实、核心、小、综合程序
- 按应用类型：科学计算、商业应用、信息处理等
- 按程序功能：宏观测试程序、微观测试程序

# 基准测试程序
# （Benchmark Suites）

| 类　型 | 名　称 | 意 义 用 途 |
|---|---|---|
| 宏观测试程序 | PARKBENCH | 并行计算 |
| | NAS | 并行计算CFD |
| | SPEC | 混合基准测试程序 |
| | Splash | 并行计算 |
| | STAP | 信号处理 |
| | TPC | 商业应用 |
| 微观测试程序 | LINPACK | 数值计算（线性代数） |
| | LMBECH | 系统调用和数据移动（UNIX） |
| | STREAM | 存储器带宽 |

# LINPACK测试程序

- Linpack是国际上流行的用于测试高性能计算机系统浮点性能的benchmark
- 通过对高性能计算机采用LU分解求解线性代数方程组能力的测试，评价高性能计算机的浮点处理性能
  - Linpack 100：早期版本，程序不可修改
  - Linkpack 1000：用全精度64位字长的子程序求解1000阶线性方程组的速度，测试的结果以Mflops（每秒百万次浮点运算）为单位
  - HPC（High Performance Linpack）：可以修改问题规模、CPU数量及通信、问题分块等，用在Top500的测试中

# HPL简介

- 高性能Linpack：**HPL**(High Performance Linpack)
  - 针对大规模的并行计算机系统的测试， 2000年9月发布1.0版
  - 是TOP500超级计算机排名的主要依据
  - 采用MPI实现，基于BLAS库或VSIPL库
  - 用户可以选择矩阵的大小（问题规模）、使用各种优化方法来执行测试程序，寻求最佳的测试结果
  - 浮点峰值＝总计算量 / 计算时间
    - 总计算量＝$2/3 * N^3 - 2*N^2$
    - 计算时间：可通过系统时钟获得

# HPL测试

- 所必须的软件包
  - HPL
  - BLAS库：GOTO、ATLAS、ACML、MKL、Linux自带的BLAS库等等
  - 编译器：C语言和Fortran 77编译器，如gcc/g77、icc/ifc、pgcc/pgf77等等
  - 并行环境：MPI
- 安装编译器、MPI、BLAS库，编译HPL
- 运行xhpl
  - eg: mpirun –np 4 xhpl
- 查看结果文件，得到Linpack性能

# HPL性能调优

- 性能调优是一个很复杂的工作，涉及的因数众多，且和具体平台密切相关（CPU、内存、网络、OS、节点数及分布、体系结构、BLAS、编译器、内存和访存、IO等等）。主要包括：
  - HPL参数的调整
  - 编译优化
  - 网络拓扑、通讯模式
  - 负载平衡
  - 数据局部性（Cache、Memory、TLB等）
  - 内存使用：80%为宜
  - 任务加载
  - OS、网络的调整
  - …… ……

# 课程小结

- 工作负载
  - 执行时间，Mflops, MIPS
- 加速比
  - 加速比的性质
  - 不同约束条件下的加速比
- 可扩展性
  - 可扩展性的含义和目标
  - 测度：等效率测度，等速度测度，平均时延测度
- 基准测试
  - 程序级性能评测基准测试
  - LINPACK测试

# 推荐读物和网站

- 《并行计算—结构、算法、编程》
  - 第3章：并行计算性能评测
- 可扩展并行计算（“Scalable Parallel Computing”）
  - 第1章：可扩展计算机平台和模型
  - 第3章：性能指标和基准程序

- X. H. Sun and L. Ni, “Scalable Problems and Memory-Bounded Speedup”, Journal of Parallel and Distributed Computing, Vol.19, Sept. 1993, pp27-37
- LINPACK Benchmark: http://www.netlib.org/benchmark
- 曹振南，如何做Linpack测试及性能优化，2004
- 高性能计算机测试基准测试程序，http://www.hpctest.org.cn/resources/index.html

# 下一讲

- 并行计算模型及并行算法设计策略
  - 《并行计算—结构、算法、编程》第4，5章