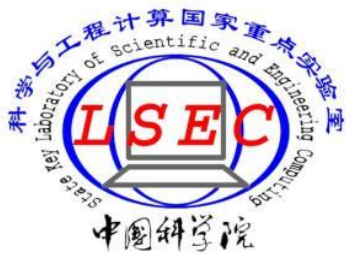
程序性能评价与优化

崔 涛
tcui@lsec.cc.ac.cn

国家数学与交叉科学中心
科学与工程计算国家重点实验室
中科院数学与系统科学研究院



大纲

- 基本概念
- 性能评价方法
  - 浮点性能
  - 加速比性能定律
    - ✓Amdahl定律
    - ✓Gustafson定律
  - 可扩展性评价标准
- 性能优化

基本概念

并行程序执行时间（execution time）等于从并行程序开始执行，到所有进程执行完毕，墙上时钟走过的时间，也称之为墙上时间（wall time）,包括：

计算CPU 时间 进程指令执行所花费的CPU 时间，它可以分解为两个部分，一个是程序本身指令执行占用的CPU 时间，即通常所说的用户时间（user time），主要包含指令在CPU 内部的执行时间和内存访问时间，另一个是为了维护程序的执行，操作系统花费的CPU 时间，即通常所说的系统时间（system time），主要包含内存调度和管理开销、I/O 时间、以及维护程序执行所必需要的操作系统开销等。通常地，系统时间可以忽略。

通信CPU 时间 包含进程通信花费的CPU 时间。

同步开销时间 包含进程同步花费的时间。

基本概念

进程空闲时间 当一个进程阻塞式等待其他进程的消息时，CPU 通常是空闲的，或者处于等待状态。进程空闲时间是指并行程序执行过程中，进程所有这些空闲时间的总和。

在处理器资源独享的前提下，假设某个串行应用程序在某台并行机单处理器上的执行时间为T_S，而该程序并行化后，P 个进程在P 个处理器并行执行所需要的时间为T_P.，则该并行程序在该并行机上的加速比S_P 可定义为：

$$S_P = \frac{T_S}{T_P}$$

效率E_P定义为:

$$E_P = \frac{S_P}{P} = \frac{T_S}{T_P * P}$$

这里，需要说明的是，T_1 指处理器个数为1 时，并行程序的执行时间。通常情形下，T_1 大于T_S，因为并行程序往往引入一些冗余的控制和管理开销。

基本概念

将并行程序的墙上时间分解为:

$$T_P = C_i + D_i \tag{1}$$

其中，C_i为第i个进程花费的CPU 时间，D_i 为第i 个进程的空闲时间。

$$C_i = L_i + O_i \qquad i = 1, 2, \cdots, P \tag{2}$$

其中，L_i 为第i 个进程数值计算指令执行花费的CPU 时间，O_i 为第i 个进程通信、同步花费的CPU 时间。

并行计算粒度进程 指令数值计算时间与墙上时间的比值，即：

$$\gamma_i = \frac{L_i}{T_P}$$

非数值冗余 由于并行引入的额外非数值计算开销,

$$W_i = D_i + O_i \qquad i = 1, 2, \cdots, P$$

基本概念

负载平衡 为了减少无谓的空闲时间，各个进程分配的CPU 时间尽量相等，为此，定义负载平衡效率如下：

$$\eta_P = \frac{\sum_{i=0}^{P-1} C_i}{P \times \max_{i=1,2,\cdots,P} C_i}$$

根据$C_T = \sum_i C_i, D_T = \sum_i D_i$, $C_T + D_T = T_P \times P$,则效率公式

$$E_P = \frac{S_P}{P} = \frac{T_S}{C_T} \times \frac{C_T}{C_T + D_T}$$

分别定义：

数值效率$NE_P = \frac{T_S}{C_T}$

并行效率$PE_P = \frac{C_T}{C_T + D_T}$

基本概念

数值效率反映了并行计算引入的额外CPU 时间开销，这种开销来自两个方面。一方面，并行执行时，随着处理器个数的增长，各个进程的cache 命中率将提高，有助于缩短总的计算CPU 时间，从而提高数值效率；另一方面，并行计算可能引入额外的开销，例如并行算法增加计算量、并行通信CPU 时间和同步开销等，它们将延长计算CPU 时间，从而降低数值效率。

并行效率反映了并行程序具体执行的并行性能，它依赖于并行机网络的通信性能，以及并行程序的负载平衡等方面，并行效率总是小于1的。

如果数值效率大于1，则可能效率将大于1，也就是，并行程序的加速比将大于处理器的个数，此时，称之为超线性加速比。由以上的分析可知，需要辩证地看待超线性加速比。如果串行程序能够很好地发挥单处理器的峰值性能，则并行程序几乎不可能获得超线性加速比。反之，如果出现超线性加速比，说明串行程序需要进一步的性能优化。

性能评价方法

- 浮点性能
- 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
- 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

浮点性能

- 理论浮点峰值性能

CPU主频 ×每个时钟周期执行浮点运算的次数 × CPU核数目

- 实际浮点性能
  - 串行程序一般只能发挥峰值性能的几个到十多个百分点
    - ✓影响因素： cache访问失效、访存性能
  - 并行程序的实际浮点性能=总浮点运算次数 / 执行时间
  - 衡量程序的绝对指标

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 $f = \frac{W_s}{W}$；
  - W_p：应用程序中可并行化部分，$1 - f$为并行分量比例；
- T_s：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

- 固定负载的加速公式：

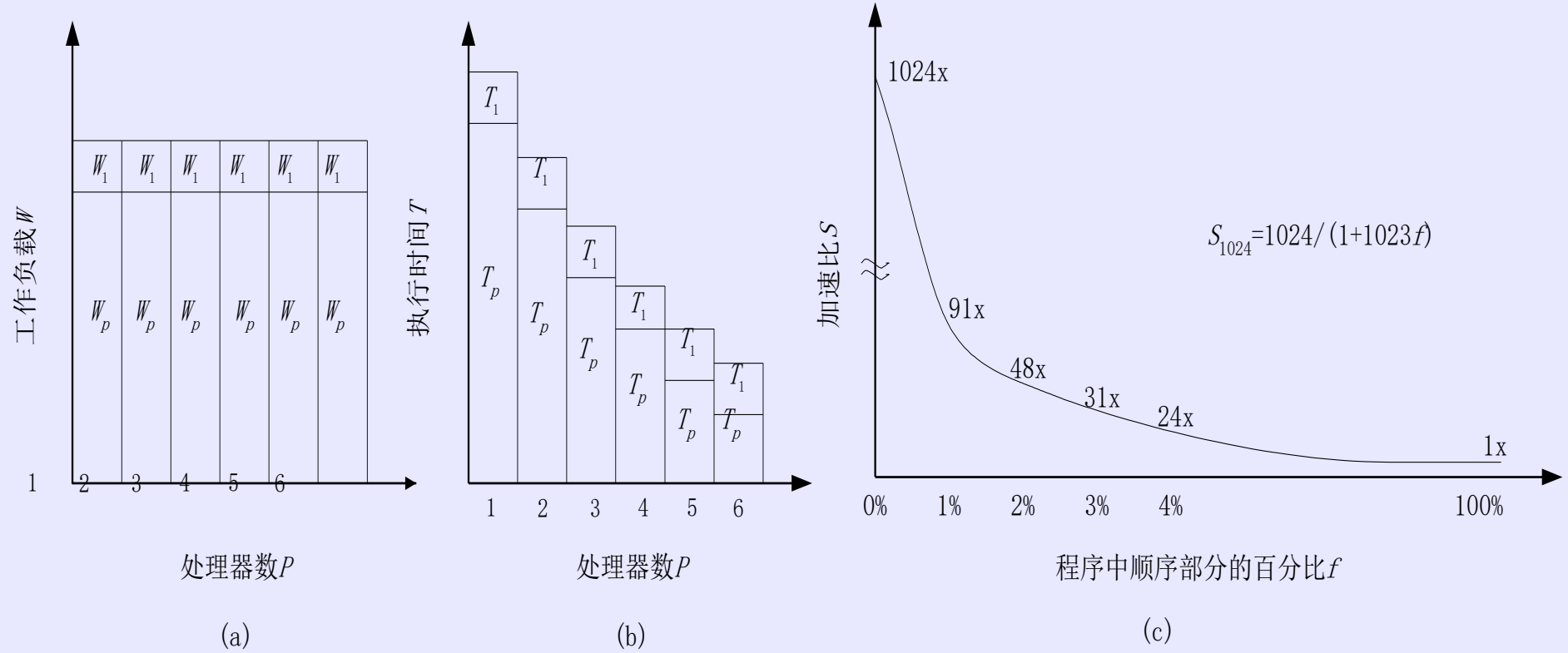
$$S = \frac{W_s + W_p}{W_s + \frac{W_p}{p}} = \frac{f + (1-f)}{f + \frac{1-f}{p}} = \frac{p}{1 + f(p-1)}$$

- $p \to \infty$时， $S = \frac{1}{f}$

若考虑额外开销W_o：

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

Amdahl's law (cont'd)



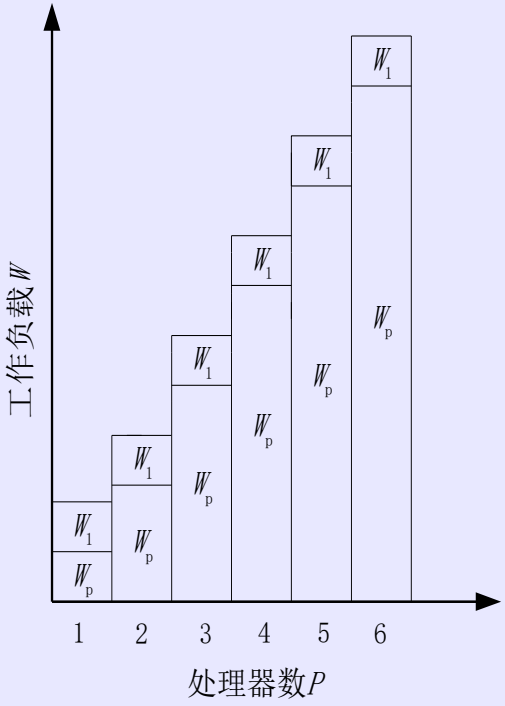
Gustafson定律

- 出发点：
  - 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： $S' = \dfrac{W_S + pWp}{W_S + p \cdot Wp / p} = \dfrac{W_S + pWp}{W_S + W_P}$

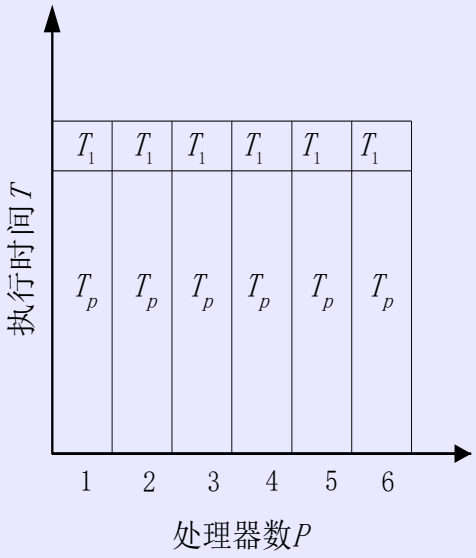
$$S' = f + p\,(1\text{-}f) = p + f\,(1\text{-}p) = p\text{-}f\,(p1)$$

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

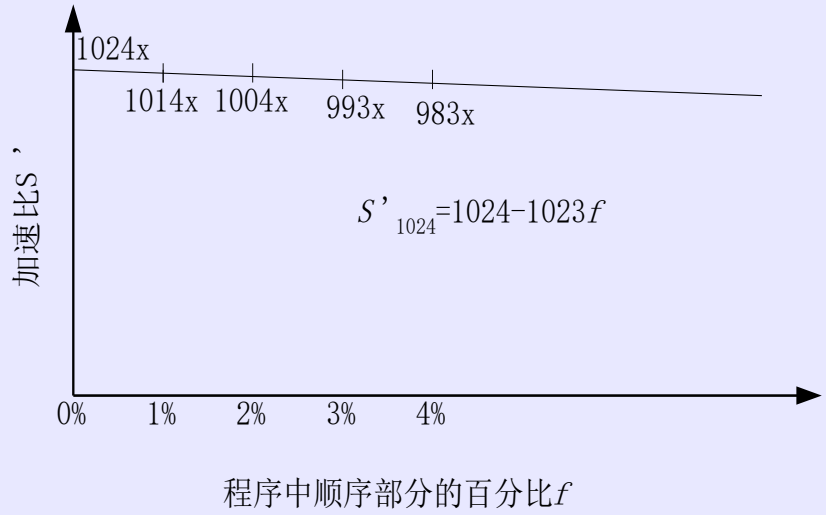
Gustafson定律（cont'd)



(a)



(b)



(c)

可扩展性评价标准

- 并行计算的可扩展性（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
- 影响加速比的因素：处理器数与问题规模
  - 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。

可扩展性评价标准

- 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

性能优化

串行程序性能优化

一个好的并行程序首先应该拥有良好的单机性能。串行程序性能优化是并行程序性能优化的基础。

- 调用高性能库
- 选择适当的编译器优化选项：“-O2” “-O3”
- 合理定义数组维数，连续数据访问避免地址增量为2的幂次
- 提高数据访问局部性
  - ✓ 时间局部性： 对同一地址的多次访问应尽可能再相邻时间
  - ✓ 空间局部性：尽可能访问当前地址的邻居
  - ✓ 注意嵌套循环顺序
  - ✓ 数据分块
  - ✓ 循环展开
- 其他优化方法： 针对指令调度、分支预测等

并行程序性能优化

并行程序优化主要是选择好的并行算法及通讯模式：

- 减少通讯量、提高通讯粒度
- 全局通讯尽量用高效的聚合通讯算法
- 挖掘算法并行度、减少CPU空闲等待
- 负载平衡
- 通信、计算重叠
- 冗余计算减少通信