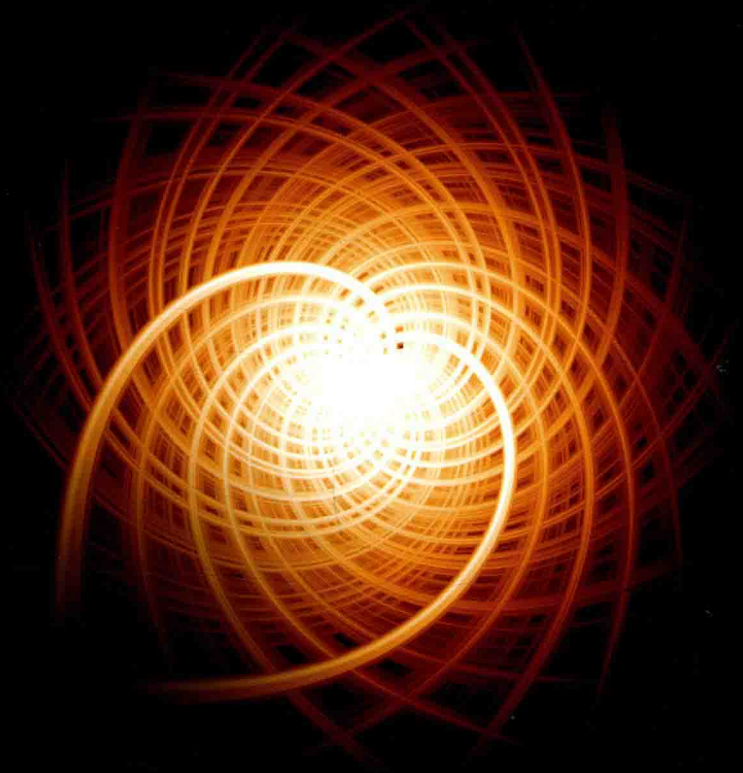
高性能计算技术丛书

高性能计算专家潜心撰写，融合作者多年并行计算技术研究成果
计算模型丰富，计算复杂度、通信复杂度、存储复杂度面面俱到
直指算法设计精髓，综合考虑技术的先进性和实用性



Parallel Computing Models and Algorithms

并行计算
模型与算法

张云泉 袁良◎著

机械工业出版社
China Machine Press

Parallel Computing Models and Algorithms

并行计算

模型与算法

张云泉 袁良◎著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

并行计算：模型与算法 / 张云泉，袁良著 . —北京：机械工业出版社，2016.4
（高性能计算技术丛书）

ISBN 978-7-111-53340-5

I. 并… II. ①张… ②袁… III. 计算模型 IV. TP301

中国版本图书馆 CIP 数据核字（2016）第 062994 号

并行计算：模型与算法

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：余 洁 责任校对：殷 虹
印 刷：三河市宏图印务有限公司 版 次：2016 年 6 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：13
书 号：ISBN 978-7-111-53340-5 定 价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

前　言

处理器是计算机系统的核心。由于摩尔定律的作用，对于体系结构的设计可以利用更多数量的晶体管来开发并行性，这使得处理器性能在2005年之前一直保持指数级增长，并且程序无须改变即可享用“免费的午餐”。但是，首先，时钟频率的提升已逼近物理极限，无法像以前一样从升级的高频单核处理器中获得免费的性能提升；其次，指令级并行，例如超标量发射和流水线技术对性能提升的潜力已基本发挥到极致，多年来没有更为显著的技术突破；最后，数据级并行，例如处理器的单指令多数据(SIMD)指令集，可以在一定程度上继续增加处理器的绝对峰值性能，但受限于带宽和片上硬件的资源，以及算法和编译的软件优化难度。总之，多核、众核处理器等利用线程级并行继续提升片上计算能力的方法已成为硬件发展的唯一道路。

算法是计算机科学的核心。由上所述，处理器体系结构的趋势是更多地利用线程级并行性，这一转变使得程序员必须设计过去只有在大型并行系统上才用到的并行算法，才能充分利用硬件计算能力。串行算法教科书通常使用RAM作为串行计算模型，并多以具体问题(排序、图算法、字符串等)或算法设计技巧(分而治之、动态规划、随机方法等)来组织内容。但并行算法的设计与底层并行机制紧密相关，因此我们以并行计算模型组织本书内容。首先，不同的模型适用于不同的问题；其次，对于同一问题在不同模型下从不同角度研究算法特征。

好算法不言自明，这是撰写本书时的核心思想。Knuth认为计算机程序设计是一门艺术而非科学，因此每个好算法都是一件精美的艺术品。相比于文字解释，作者相信使用精致优美的伪码表述的算法能进行自我说明，是艺术品最原始、最真实的展现。我们鼓励读者以算法伪码为中心阅读本书，而将文字看作对伪码的简要说明。然而，限于作者水平，加之时间仓促，书中定有不少欠妥和错误之处，恳请读者批评指正。

目　录

第1章 绪 论

本章前三节对模型、计算模型和并行计算模型进行简要介绍。1.1 节列举数学和物理中的两个模型，以说明模型的意义和分类方法；1.2 节介绍几种计算能力模型和算法设计模型，进一步介绍模型在计算机科学中的作用；1.3 节介绍并行计算的基本度量参数，以及三类基本的并行计算模型，后续几章均面向某类并行计算模型展开。并行计算领域中其他一些相关概念统一放在 1.4 节做简要介绍，更为详细的讲解请参考 1.6 节中列举的相关文献。最后，1.5 节包含本书所用到的并行算法表述方法和问题定义。

1.1 模型

模型是所研究对象(包括实物系统或动态过程等事物)的一种抽象表述形式，其目的是描述对象的结构、功能、过程等本质特征，通常需要对研究对象进行一定的简化，并用一定的形式或规则描述其主要特征。简而言之，这种对研究对象的模拟称为模型。对所研究问题建立抽象模型是一种重要的研究方法，有利于深入理解问题本质并提供简化的设计和分析平台。

建立模型的方法可分为白盒方法和黑盒方法。在白盒方法中，需要分析事物内部的运行机制，利用数学工具进行抽象，形成符号化、形式化的表示。然而，由于实际情况的复杂性或不可知性等原因，有时无法或者没有必要抽象问题的所有因素，此时可以利用黑盒方法，即忽略、简化或假设问题内部机制，或从事物对外的接口层面出发，通过实测数据分析建立统计分析模型或利用问题接口特征建立唯象模型等方式进行分析。下面分别介绍白盒的数学模型和黑盒的物理模型，以便体会两种方法的区别。

1.1.1 白盒模型

考虑如下问题，为了将一幅画挂到墙上，可以首先将一条绳子的两端固定在画框的上侧，再将绳子挂于固定在墙面的钉子上。假设墙上有两个并排的钉子，如何将画稳定地挂在两个钉子上，使得当其中任意一个钉子被拔掉时，画将跌落到地上。

为了解决这个问题，需要建立一个数学模型。记两个钉子为 x_1 和 x_2，假设已存在某

种满足条件的解决方案，此时从绳子的一端遍历到另一端，当其沿顺时针方向从一个钉子(x_i)上方绕过时，记为 x_i，而沿逆时针绕过时记为 x_i^{-1}。完成遍历时会得到一个 x_i 或 x_i^{-1} 相乘且不满足交换律的序列 S，因此安排绳子的方法即使得该序列 S 满足，当任意一个 x_i 从该序列中删除时，S 可以消为 1。一个简单的解决方案为 $x_1x_2x_1^{-1}x_2^{-1}$。

该问题的数学模型并不复杂，抽象了实际问题的两个特征：首先，仅当绳子从钉子上方绕过时才会起到支撑作用，且仅存在两个这样的方向，即顺时针和逆时针方向；其次，遍历绳子得到的序列不满足交换律。

建立数学模型，使得问题的本质暴露无遗，并给出了更为形式化的描述方法，提供了解决问题的设计平台。利用数学模型，还可方便地进行问题的扩展研究。例如更一般的问题为：假设墙上有 n 个并排的钉子，给定 n 个钉子集合幂集的一个子集 D，如何设计绕绳方案使得仅当 D 中任意一个元素所包含的钉子被拔掉时，画将跌落到地上。

1.1.2　黑盒模型

再来考虑统计物理中的一个例子。对于一个微观粒子，其能量只能取一组离散能级值 e_1，e_2，e_3，…，且能级为 e_i 的微观粒子可处于 w_i 个不同状态。对于一个包含 N 个微观粒子的宏观平衡系统，称 $\{n_i\}$ 为粒子能级分布，即处于能级 e_i 的粒子数为 n_i（$N=\sum n_i$），且每个粒子均可处于 w_i 种状态之一；系统内能 E（忽略粒子间相互作用）为所有 N 个粒子能量之和，即有 $E=\sum e_in_i$。

统计物理的基础问题之一为求解能量 E 固定的平衡态系统中粒子分布。由于系统由大量粒子组成，无法准确测量每个粒子的初始状态，也无法利用数学方程求解每个状态，因此，只能基于某些简化的假设条件，利用统计方法进行建模。一个合理的假设为，每个粒子满足上述两个约束等式，即固定的粒子总数和系统能量等式的粒子状态分布都是等概率出现的，因此微观状态数目 Ω 最多的分布 $\{n_i\}$ 为系统平衡时的分布。

具体而言，位于能级 e_i 的 n_i 个粒子可有 $w_i^{n_i}$ 种状态，由于粒子是可以区别的，因此可得所有能级上粒子分布的微观状态数目为：

$$\Omega=\frac{N!}{\prod n_i!}\prod w_i^{n_i}$$

结合两个约束等式，利用拉格朗日乘子法可得 $n_i=w_i\mathrm{e}^{-\alpha-\beta e_i}$，其中 α 和 β 为拉格朗日法中的乘子。该模型没有利用宏观系统的内部机制，仅基于一个简单合理的假设，利用统计方法进行求解。该模型依然可以适用其他不同假定的系统，如系统中的粒子不可区

分或每个能级的 w_i 状态上只能容纳一个粒子等。

1.2 计算模型

模型在计算机科学中也具有基础性作用，本节介绍计算理论层次上的理论计算能力模型，以及算法设计层次的串行算法设计模型，而并行计算模型在 1.3 节介绍。

计算理论层次主要指计算能力模型，从文字意义上理解，包括模型的“能”和“力”两方面。

对于模型的“能”，即能识别或判定的语言类或能计算函数类等，其中以 Church-Turing 论题为核心给出了算法的定义，称为可计算性理论，研究模型之间的等价或包含关系，从而解释什么是计算。

对于模型的“力”，即计算的速度，问题的复杂度与计算能力模型的选择相关。例如单带图灵机上判定语言 $L=\{0^k1^k \mid k\in \mathbf{N}\}$ 的时间复杂度为 $\Theta(n \log n)$，但在双带图灵机上时间复杂度为 $\Theta(n)$。这似乎与下面介绍的算法设计模型中的研究内容一致，但由于计算能力模型与真实计算机器之间的差距，通常更着重研究模型整体的“力”，如可以证明确定型计算模型(例如双带和单带图灵机)都是多项式等价的，而研究单一问题的不同算法更偏向基于算法设计模型，如 RAM 或电路模型。

算法设计层次主要指算法设计模型，从文字意义上理解，包括“设”和“计”两方面。“设”即给出具体算法，“计”为求解相应算法复杂度即问题上界。算法设计和分析需要计算模型支持，对实际硬件执行过程建立抽象建模，在其上进行计算机算法设计，并分析算法计算时间复杂度、空间复杂度以及访存复杂度。

计算能力模型中，一类自动机定义了一类语言，但自动机本身也有算法设计的问题，如设计最简单的自动机识别一种语言，或者将在不同自动机间转化。同样，在算法设计模型中，也存在模型间等价性定理，这属于计算能力问题。因此，计算模型均包括计算能力和算法设计两个属性，只是某些模型更偏重某个属性，并不存在严格界限。

1.2.1 计算能力模型

计算模型的直观意义就是描述计算过程的模型，然而对究竟什么是计算、什么是算法这一问题，目前公认的是 Church-Turing 论题，即任何可计算函数均可由图灵机以及等价的计算模型实现，这些等价的模型从不同角度出发定义什么是可计算，包括部分递归

函数、Post 可计算函数、λ 可定义函数、一般递归函数和马尔科夫算法等。

本小节介绍几种计算模型，包括函数、机器、语言、文法和程序，每个模型都包括不同能力的子类型。表 1-1 列出了不同模型中一种能力较弱的实例以及定义什么是算法的实例。

表 1-1　计算能力模型

模型	弱	强
机器	有限自动机	图灵机
文法	正则文法	句构造文法
函数	原始递归函数	部分递归函数
程序	BlooP	FlooP
语言	正则表达式(语言)	递归可枚举语言
应用	模式查找、词法分析	计算能力

1.2.1.1　程序

以函数形式定义的可计算类，以及后面利用其他模型构建的可计算类，均可以利用程序设计实现。程序即一组指令的集合，因此指令种类确定了程序可实现的函数。下面介绍两种程序设计语言。

BlooP 程序包括三条指令：$V \leftarrow V+1$，$V' \leftarrow V$，以及 LOOPV，V 为变量。其中 LOOP 为有限循环，即执行到 LOOP 时的 V 值即为循环次数。可以证明，BlooP 程序与下面将要介绍的原始递归函数等价，即包含相同的函数类。

FlooP 程序在 BlooP 基础上增加了无界限循环 μ-LOOP，即包含三条指令：$V \leftarrow V+1$，$V \leftarrow V-1$，if $V \neq 0$，goto A。可以证明，FlooP 语言可表示所有可计算函数。几乎所有程序设计语言都包含 FlooP 语言的三条指令，因此利用这些程序设计语言均可设计实现可计算函数。

1.2.1.2　函数

函数模型从数学函数角度出发，定义不同的基础函数和操作算子。不同初始函数集加不同的算子集可生成不同的函数类，函数类的层次包含关系是数理逻辑分支之一递归论的主要研究内容。

原始递归函数包括三个基本函数，即零函数 $n(x)=0$、后继函数 $s(x)=x+1$和投影函数 $u_i^n(x_1, \cdots, x_n)=x_i$。由基本函数出发通过合成和原始递归两个运算得到的函数依然属于原始递归函数。**合成**即函数组合；若 g 是可计算的，则如下定义的函数 h 称为 g

的**原始递归**：

$$h(t)=\begin{cases}k & t=0\\ g(t,h(t-1)) & t>0\end{cases}$$

每个原始递归函数都是可计算的，并且可以证明，它与 BlooP 程序设计语言能力等价。与 FlooP 等价的函数模型称为**递归函数**，即在原始递归函数定义基础上增加极小化 μ 算子。Ackerman 函数是全函数，它不是原始递归函数(无法用 BlooP 程序计算)，但属于部分递归函数(可由 FlooP 程序计算)。因此部分递归函数的模型计算更“能”。

1.2.1.3 机器

下面三种模型，即机器、文法和语言，都需要字母表、字符串以及语言等概念。**字母表** Σ 为至少包含两个元素的有限集合。**字符串** $s(|s|=k)$ 为字母表笛卡儿积 Σ^k 中的元素。空串用 $\varepsilon\in\Sigma^0$ 表示。**语言** L 为字母表上所有字符串集合 $\Sigma^*=\Sigma^0\cup\Sigma^1\cup\cdots$ 的子集，即 $L\subseteq\Sigma^*$。

有限自动机(Finite State Machine)是一个五元组$(Q,\ \Sigma,\ \delta,\ q_0,\ F)$：

- Q 是一个有限状态集合。
- Σ 是字母表。
- δ 是转移状态集合，$Q\times\Sigma\to\Sigma$。
- $q_0\in Q$ 是初始状态。
- $F\subseteq Q$ 是接受状态集合。

字母表 Σ 上使得有限自动机 M 进入一个接受状态 $q\in F$ 的串集合 A 组成机器 M 的语言，即 $L(M)=A$。

在计算理论方面，特别是算法的定义中，图灵机是一种更为重要的机器模型，如果一个计算模型能计算所有通用图灵机可计算函数，称为**图灵完全**(Turing completeness)。如果一个图灵完全的模型中所有可计算函数均为图灵可计算函数，则称其为**图灵等价**(Turing equivalence)。目前所有的图灵完全模型都是图灵等价模型，这一事实支持了 Church-Turing 论题。

1.2.1.4 文法

文法是一个四元组 $G=(N,\ T,\ P,\ S)$，其中：

- N 为变量的非空有穷集合，也称为非终结符号(nonterminal)。
- T 为终结符的非空有穷集合(terminal)。

- P 为产生式的非空有穷集合，定义语法规则，每个产生式左边至少有 N 中的一个元素。
- S 为文法开始符号，$S \in N$。

从字符串 S 开始，利用一系列产生式规则将所有非终结符替换为终结符，生成的最终字符串 ω 称为文法生成的串。文法 G 定义的语言为 G 能生成的只包含终结符的字符串集合。

句构造文法(Phrase-Structure Grammar)中的产生式 $R=a \rightarrow b$，$a \in (N \cup T)^+$，$b \in (N \cup T)^*$，且 a 至少包含一个非终结符。

上下文相关文法(Context-Sensitive Grammar)中的产生式 $R=a \rightarrow b$，$a \in (N \cup T)^+$，$b \in (N \cup T)^*$，a 至少包含一个非终结符且 $|a| \leqslant |b|$。

上下文无关文法(Context-Free Grammar)中的产生式 $R=a \rightarrow b$，$a \in N$，$b \in (N \cup T)^+$，即满足 $|a|=1$ 且 $|a| \leqslant |b|$。

正则文法(Regular Grammar)中的产生式 $R=a \rightarrow b$，$a \in N$，$b \in T$，即满足 $|a|=|b|=1$，或 $R=a \rightarrow cb$，$a \in N$，$b \in N$，$c \in T$。

以上四种文法即定义了一组层次，其接受的语言类能力依次递减，并存在能力相同的机器，分别为图灵机、线性有界自动机、下推自动机和有限状态机。

1.2.1.5 语言

如前所述，某种机器模型接受的字符串集合，或者某种文法生成的字符串集合，称为一种语言，因此定义了正则语言、上下文无关语言、上下文相关语言、递归可枚举语言等。

也可以不依赖机器或文法等外部方式，直接从字符串出发定义语言，如下面定义的正则表达式，即从基本的几种字符串集合出发，利用一些操作生成更为丰富的字符串集合种类。

正则表达式 R 可为三类基本集合：只包含字母表中一个元素 $a \in \Sigma$ 的集合；只包含空串 ε 的集合；空集 $\varnothing$。如果 R_1 和 R_2 是正则表达式，则 $R_1 \cup R_2$、$R_1 \times R_2$ 和 R_1^* 依然是正则表达式。

一个正则表达式生成的串集合称为正则语言，正则表达式与有限自动机或正则文法

是等价的：给定接受语言为 L 的一个有限自动机或正则文法，可以找到一个正则表达式，其生成的串集合为 L；反之亦然。

从字符串以及集合的角度出发可以得出，机器或文法等模型的定义是字符串的有穷集合，因此机器的集合和文法的集合为可数集合。类似地，语言可以是字符串的无穷集合，因此语言的集合为不可数集合，如表 1-2 所示。从而可知，语言集合的势严格大于机器集合的势，因此存在不可识别的语言。

表 1-2 字母表、串及语言

	字母表	串的集合	语言
定义	Σ	Σ^*	Σ^* 的一个子集
势	有限	可数	不可数

1.2.2 算法设计模型

解决某一问题的算法最终要在硬件平台上执行，因此抽象各种具体计算机的基本特征，形成一个面向算法的计算模型，为算法设计和分析提供简化平台。这样可首先在计算模型上设计算法，然后将算法映射到硬件平台上。

由于计算机执行过程复杂，并且影响算法性能的参数较多，参数间也相互制约，因此计算模型应具有以下特性：简洁性，即只抽象硬件最关键特征，为算法设计提供一个简化平台；通用性，即算法设计可移植到所有具备计算模型特征的硬件上，并保证性能。下面介绍两种基本的算法设计模型。

1.2.2.1 随机访存模型

随机访存模型(RAM 模型)抽象串行计算机。RAM 模型忽略了冯·诺依曼模型五个构件中的输入、输出和控制器，只保留运算器和存储器，并且假定任何操作的开销为单位时间，包括读写任意内存位置字节和执行任意乘法、加法、比较等运算，其中读写开销与内存位置无关正是 RAM 模型中随机访问的意义所在。

RAM 模型上的算法可用时间和空间复杂度两种方法度量。时间复杂度为算法执行运算的语句个数，空间复杂度为算法需求的内存空间大小。将时间和空间看做资源，则 RAM 模型上的主要研究内容是分析某一问题消耗某种资源的上下界。

问题的资源消耗上界的证明通常通过设计一个算法实现，存在几种通用算法设计思路，包括分治、动态规划、贪心、回溯和分支定界等。

问题的资源消耗下界的证明不仅指出无法设计一个更低复杂度的算法，还提供了更深入理解问题的本质，并且通常复杂性下界证明过程利用的技术可用来设计最优算法。通常找到问题的下界并进行证明比设计一个达到下界复杂度的算法更难，但是也存在一些证明问题下界的通用方法，如利用比较树方法、对手参数方法、势能方法和图论方法。

1.2.2.2　电路模型

逻辑电路是计算机硬件设计基础，可以表示为DAG(Directional Acyclic Graph，有向无环图)。图中没有入度的节点为输入节点，输入为布尔变量，即0或1；非输入节点表示一个操作，操作输入变量并输出结果；没有出度的节点为输出节点。逻辑电路的DAG中的节点均为布尔逻辑操作，若为代数操作则称为代数电路，若为比较操作称为比较电路。总之，这些能表示为DAG的电路统称为**电路模型**。

一种与电路等价的模型称为**非转移程序**(Straight-Line Program)，即删除分支和循环功能的RAM模型。由于没有分支和循环语句，非转移程序中的每条语句均可看做图中的一个节点，若某个语句依赖另一条语句的输出数据，则图中设置一条有向边表示依赖，可得非转移程序等价的DAG，因此非转移程序可用电路实现，同样，电路模型、非转移程序模型和DAG模型是等价的。

电路模型的复杂度可用DAG的非输入节点数目表示，称为电路大小表示；或DAG深度，即DAG所有输入节点到输出节点长度最大值表示。

1.3　并行计算模型

并行计算即利用多个处理部件同时执行计算解决某个问题，IEEE电子工程名词标准词典里对并行的定义为“分别使用独立设备并且在同一时刻转移、产生或者处理一个整体的几个独立部分”。并行计算的主要目标是提升问题求解速度，同时，并行环境的存储容量也支持处理规模更大的问题。

串行计算的复杂度度量通常包括时间复杂度和空间复杂度，前者考虑算法执行的指令数目，后者为算法的空间需求大小。在并行计算中，由于不同计算同时执行，需要对数据和计算进行划分和控制，同时由于数据传输速度落后于处理速度，因此并行计算的性能还要考虑数据的通信和移动开销，即数据的横向通信发生在多个处理器利用消息传递数据和控制信息上，数据的纵向移动发生在具有多层存储层次的处理器上。据此可将并行计算模型分为如下三代：

- 第一代并行计算模型主要关注问题的计算方面，即问题的计算并行性。
- 第二代并行计算模型主要关注问题的通信方面，即数据的横向局部性。
- 第三代并行计算模型主要关注问题的访存方面，即数据的纵向局部性。

本书的内容和目标即从计算、通信和访存三个层面介绍并行计算模型，通过在这些模型上设计算法以及分析下界，理解问题在计算、通信和访存这三个方面的特性。因此基于不同底层计算模型进行理论分析和算法设计，都是挖掘问题本身在某一方面的特性，而忽略了其他方面的开销，这虽然降低了算法的实用性，但提高了模型的可用性。

1.3.1 基本度量参数

如前所述，在RAM模型中，时间复杂度和空间复杂度是两个主要性能指标，而对电路模型而言，主要指标为深度和大小。对于实际程序以及并行算法而言，还有更多更为实际的参数，本小节进行简要介绍。

1.3.1.1 串行参数

指令计算性能是机器的运行速度，可用MIPS(Million Instructions Per Second)表示，即每秒钟可执行的百万条指令数。**硬件性能**指硬件**峰值性能**，可根据时钟频率 f 和每周期完成的指令数(Instructions Per Cycle，IPC)得出；**程序性能**指软件执行速度，即程序负载与程序执行时间之比。**程序负载** W 是程序执行的指令数，可为所有指令数或仅为所有浮点指令数。对于一般程序用程序指令数作为负载，因此性能用MIPS即每秒百万条指令数表示，但该指标受指令集和编译器的影响，因此对于科学计算程序，常用浮点操作数作为负载，满足硬件无关性，对应的性能指标用MFLOPS(Million Floating-point Operations Per Second)即每秒百万浮点操作执行次数表示。程序负载是**问题规模**即输入数据大小 n 的函数，且对大部分问题负载与输入的函数关系是确定的；**程序执行时间**也称为执行时间(wall-clock)，是程序开始到结束的时间，包括计算、访存、通信以及操作系统时间之和。通常程序性能小于硬件峰值性能，因此二者之比可定义为**利用率**。

数据传输性能可分为通过网络的横向数据传输性能和通过存储层次的纵向数据移动性能，通常用**带宽**(Bandwidth)和**延迟**(Latency)两个参数描述。延迟为传输单位个数据所花时间，带宽为连续数据发送间隔时间的倒数。数据传输性能同样也可分为硬件性能和程序性能。硬件性能可通过频率和字长计算得出。对程序性能，其带宽通常称为**吞吐率**(Throughput)，即固定时间处理数据量的大小。对于存储层次而言，其**容量**也是重要参数。

1.3.1.2 并行参数

为了方便介绍并行程序基本参数，首先给出一个具体事例，即求 n 个数之和问题的串行和并行算法。串行算法顺序执行 $n-1$ 次加法，时间复杂度为 $O(n)$。并行算法中假设 n 为 2 的幂且有 n 个处理器排成一维线性阵列，每个处理器顺序编号并存储相应编号的输入值，在每一步中，奇数编号的处理器 i 将其值发送到左邻居 $i-1$，偶数编号的处理器将收到的值与其存储的值相加，下一步中只有偶数编号的处理器参与计算。重复此步骤直到只剩 0 号处理器输出最终结果。并行算法执行类似一棵完全二叉树，时间复杂度为 $O(\log n)$。下面分别介绍常用的并行算法复杂性度量，包括问题规模 n、并行粒度、处理器个数、并行执行时间、加速比、成本以及效率。

处理器个数 P 通常为问题规模 n 的函数。在并行算法设计和分析时，处理器个数常用的有 n^2、n、$\log n$ 和 $n^{\varepsilon}(0<\varepsilon<1)$。在上述并行求和算法例子中，并行处理器个数 $P=n$。

并行执行时间 T_P 为并行算法运行的时间。但对于不同并行计算模型，时间衡量标准也不同。例如在第 3 章的共享存储模型 PRAM 上，由于假定访存为单位时间，所以运行时间可由运算步骤衡量，不考虑访存时间；在第 4 章关注网络通信性能的分布式层次模型上，如 LogP 模型，运行时间由通信开销衡量，而没有考虑计算时间；在第 5 章关注存储层次间数据流动的模型上，如 HMM 模型，运行时间由数据移动次数衡量，也没有考虑计算时间。在第 2 章介绍的一些固定互连结构的并行计算模型中，如线性互连模型，需要同时考虑数据传输开销和运算性能，但大部分算法的运行由计算和通信交替执行，因此可以简化分析只考虑计算开销。并行求和算法的并行时间为 $\log n$。

加速比 S 为串行执行时间与并行执行时间之比，$S=T_1/T_P$。并行求和算法的加速比为 $n/\log n$。若 $S=P$，称该并行算法具有线性加速。若 $S>P$，称该并行算法具有超线性加速。有两种情况会出现超线性加速比：在并行搜索算法中，某一处理器可能提前求出解，从而减少其他不必要的分支，降低了问题的开销；或者在并行执行时，由于缓存的增加，提高了数据的读取速度。

并行成本 C 定义为并行执行时间乘以处理器个数 $C=P\times T_P$。如果 $C=T_1$，则称并行算法是成本最优的。并行求和算法的成本为 $n\log n$，因此该算法非成本最优。

有的问题无法同时达到最大加速比和成本最优。在追求最大加速比时需要考虑问题的最大并行度，然而并非在算法执行期间的每个时刻均存在如此多可并行指令，因此会