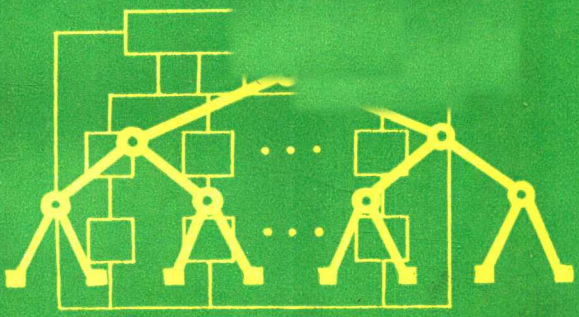
并行数值方法

陈景良



清华大学出版社

并行数值方法

陈景良

清华大学出版社

内 容 提 要

近二十年来，随着并行计算机的发展，并行算法的研究已有不少成果。本书主要篇幅用于有关一般算术表达式、多项式、递推问题、线性方程组及特征值的各类具体并行数值方法的介绍和讨论。在第1章中还对并行算法面向的机器模型，基本的概念，方法和问题作了分析与综合。

本书可作为计算数学，计算机软件与硬件等专业的工作者、研究生和学生的参考书或选修教材。

并行数值方法

陈景良

☆

清华大学出版社出版

北京　清华园

中国建筑工业出版社印刷厂排版

印刷厂印刷

新华书店北京发行所发行・各地新华书店经售

☆

开本：787×1092 1/32　印张：$4^{1}/_{8}$　字数：79 千字

1983年10月第一版　1983年10月第一次印刷

印数：1～20000

统一书号：15235・78　定价：0.50元

前　言

计算机传统结构的显著特征是单指令流单数据流，每一时刻按一条指令处理一个数据。通常的数值算法适于此类计算机，可称**串行算法**。六十年代开始发展含大量处理机的并行计算机，分单指令流多数据流与多指令流多数据流两类，每一时刻按一条或多条指令处理多个数据。所谓**并行算法**正是面向并行计算机的数值方法。

近二十年来，并行算法的研究已有不少成果。本书试图对并行算法从面向的机器模型，基本概念和一般问题，到已有的最基本的一些并行数值方法，进行较为系统的介绍与阐述。

全书共分六章。第1章并行算法概述，对并行算法从面向的硬件的类型到自身的基本的概念，方法和问题进行了分析与综合。占据本书主要篇幅的其余各章是关于算术表达式，多项式，递推问题，线性方程组及特征值的各类具体数值方法的讨论。为了揭示主要线索与实质，许多方法的推导和结论的证明是作者重新给出或加工的。为了系统描述，我们尽力避免术语和记号方面的不一致。

有关并行算法的文献较多，本书最后仅列出作者查阅过的被引用的部分文献。值得指出的是，关于并行数值方法的研究工作，*Miranker*[30]总结了六十年代后半期的情况，*Heller*[18]，*Sameh*[35]分别对直至1977年的最新发展作了叙述，这些综合述评很好地反映了进展概况。

并行数值方法正在发展中，作者用于这方面的研究时间不多且水平有限，错误和缺陷在所难免，切望读者提出宝贵意见。

李庆扬同志审阅了本书书稿，作者表示衷心感谢。

目　录

1.并行算法概述

1.1 引言

电子计算机从四十年代问世至整个五十年代的研制，基本上是依照*Von Neumann*的概念结构的（图1.1）。它们在每一时刻只能按一条指令对一个数据进行操作，可称**单（或串行）处理计算机**。由于电子信息传输速度以光速（每毫微秒1英尺即约30厘米）为极限，而且在此期间许多电路已达到在毫微秒的量级内动作，已感到单处理机速度提高的限制及不合算的代价。

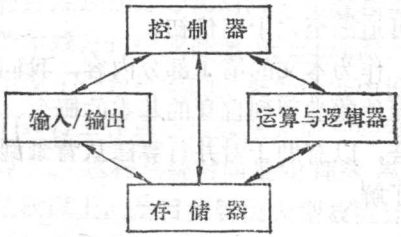


图 1.1

摆脱传统结构发展并行性具有提高速度的巨大潜力。**并行性**是指多于一个事件在同一时刻发生或在同一时间间隔内发生。六十年代随着硬件成本日趋下降，开始了计算机结构上发展并行性的革新，主要有（见[49]）：

（1）多用户分享资源的分时系统。实现在时间上轮流使用快速中央处理器，而让慢速外围设备的使用在时间上重叠起来。

（2）时间上重叠使用中央处理器的流水线系统。可供采用的流水线计算机有美国的控制数据公司的*STAR*-100

（1973），它有两条流水线，*Texas*仪器公司的*ASC*（先进科学计算机）(1972)，它至多可有四条流水线，以及属于目前为最高水平的*CRAY-1*.

（3）资源重复设置的并行处理系统。最早的努力是从1962年*SOLOMON* I 阵列处理机开始，以1972年完成*Illiac* Ⅳ系统作为成果。此外，在关联处理机的基础上建造了*PEPE*（1971）及*Goodyear*宇航中心*STARAN*系统。以上这些系统利用至多32～288个处理机并行工作，几十倍地提高了处理速度。

随着并行处理计算机的研究与发展，出现了对于数值方法的一种新的分类：通常所说的数值方法适用于串行处理计算机，可称**串行算法**；而新的面向并行处理计算机的数值方法则称**并行算法**。并行算法的研究已有不少成果，有关的文献可追溯至六十年代初。

作为本文的第1部分内容，我们试图对并行算法从面向的硬件的类型到自身的基本的概念、方法和问题作一分析与综合，以有助于对并行算法从背景到实质的比较具体和清晰的了解。

1.2 并行处理计算机及其分类

关联处理机（associative processor） 实现存储器操作并行。具有关联存储器（图1.2），这是一种按内容寻址的具有信息并行处理功能的存储器。比如读数，不是按地址读出相应存储单元的数码，而是按给定信息内容的某些或全部特征，把与之相符的所有数码一次判别出来。关联处理机的中央处理器的操作也是并行的，擅长信息探索任务，还能求解一系列数学计算问题，获得比串行计算机较高的效率。

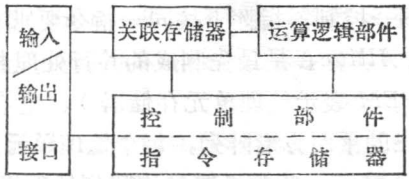


图 1.2

流水线处理机（*pipeline processor*） 实现处理器操作步骤并行。将一种操作按功能划分成一系列子操作，利用功能部件分离与时间重叠实现子操作并行，是流水线结构的普遍原则。这正是生产装配线的设计思想，它使每个被操作对象在整个操作流程的不同的功能部件中保持在不同的完成阶段，从而使吞吐量大为增加。这种原则首先用于指令操作，在功能划分（例如将浮点加法划分为对阶、尾数相加、规格化等）基础上，以流水线方式组成高速中央处理器。这种原则的进一步应用，出现了一台机器的中央处理器中设置多条专用流水线，让它们并行工作或连接起来完成一些复合操作（如计算向量内积等）。这种系统对向量处理效率很高，可达每秒浮点操作亿次以上，是目前解决大型数值计算问题为中心的巨型计算机的主要型式。图 1.3 是一个操作被划分为 3 个子操作组成的流水线的示意图。三个大框图表示将操作数 A 与 B 变换为量 $O(A, B)$ 所经过的三个子操作部件，三个小框相应地表示中间结果 $O(A, B)_1$ 与 $O(A, B)_2$ 以及最终结果 $O(A, B)$ 的存储部件。显然，流水线在接收三组操作数对 (A_i, B_i) 后，将以三个子操作部件中最慢的一个子操作时间为每组操作数得到结果所需的时间。

并行处理机（parallel processor） 实现处理器操作并行。重复设置中央处理单元 *PE*(processing element),

所有 PE 在同一控制器指挥下按同一指令要求对一组数据同时进行操作。*Illiac*Ⅳ是最先制成的并行处理机，图 1.4 是原理框图（PEM 表示处理单元存储器），它有 64 个处理单元，排成8×8的平面方形阵列，每个处理单元只与其四个相邻的处理单元相连。此类机器又称阵列处理机，对有限差分、矩阵、快速富氏变换（FFT）等计算具有很高效率，而且在图象信息处理等一些专门领域中得到成功的应用（见[3]）。

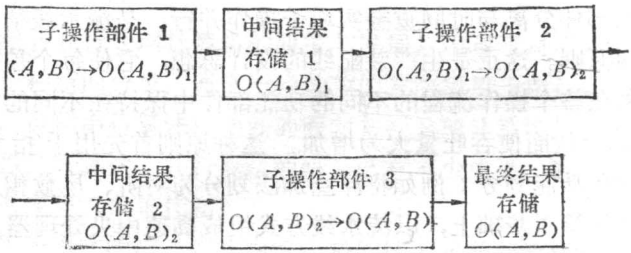


图 1.3

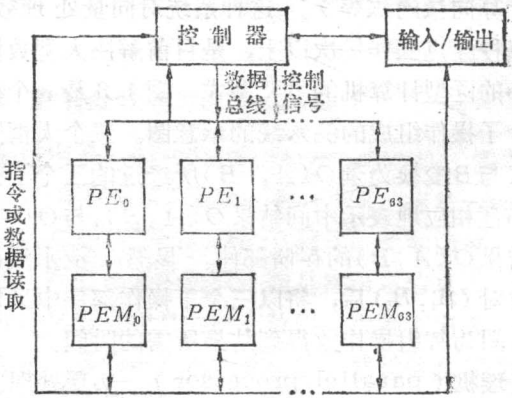


图 1.4

多处理机（multiprocessor） 实现指令、任务并行。指令级并行就是同时对多条指令及其各自有关数据进行处理。

美国M.J.Flynn于1966年将计算机系统分为四种类型：

（1）单指令流单数据流（*SISD*-Single Instruction Stream Single Data Stream）。此类系统即传统串行处理计算机。

（2）单指令流多数据流（*SIMD*-Single Instructior Stream Multiple Data Stream）。以并行处理机为代表，流水线处理机与关联处理机也属此类系统。

（3）多指令流单数据流（MISD）。此类系统在实际上以何种计算机为代表尚为存疑。

（4）多指令流多数据流（MIMD）。多处理机属此类系统，是并行处理的理想结构，但在算法、程序和硬件等方面尚面临许多复杂问题。

1.3 假设与定义

目前并行算法的研究都是面向SIMD或MIMD两类系统，而且大多倾向SIMD系统，这反映了SIMD系统的固有的简单性，即比较容易得到产生可靠且有效的程序的算法（见[18]）。

SIMD机或MIMD机中可供使用的处理机的数目p当然是重要参数。面向SIMD机的并行算法是以“p个处理机在同一时间对各自的数据执行同一指令”为依据而建立的有效算法。面向MIMD机的并行算法建立的依据则是“p个处理机在同一时间对各自的数据执行不同的指令”。

但是算法的选择往往还取决于非算术的考虑。事实上，

所有并行算法的实践必须处理当前的并行计算机上存在的数据结构、存储分配和存取冲突，以及处理机间通讯等复杂问题。

因此，为了避免局限于个别机器特征而造成并行算法的研究不能放开手脚，需要进一步提出具有若干理想化假设的机器模型，作为建立各种并行算法的面向的一般依据，从而使精力能集中于数值方法方面的研究。常用的一些主要假设归结起来有如下几种（见[25]，[24]）：

（1）处理机数目的假设：在任何时间可用任意多个处理机。

（2）存储器的假设：在任何时间有任意多个单元的主存储可供每个处理机存取；不存在任何存取冲突。

（3）时间的假设：处理机与存储器间传递数据不需时间；每个运算都花同样的时间量，称作**一单位步**。

SIMD 系统和 MIMD 系统加上如上假设构成两种机器模型：SIMD模型和MIMD模型。

理想的情形是一个算法使用 p 个处理机应当比使用 1 个处理机快 p 倍，但除极个别情形（如造函数表）外，实际速度提高要少得多。因此建立度量并行算法有效性的概念是重要的（见[24]，[35]）。

对于一个给定的问题，设T_p是所讨论的并行算法使用 p 个处理机的运行时间，T_1是已知最快的串行算法在单处理机上的运行时间。定义所得**加速**为

$$S_p=T_1/T_p,$$

用以度量算法并行性对计算时间改进的程度。定义其**效率**为

$$E_p=S_p/p=T_1/pT_p,$$

用来度量处理能力发挥的程度。显然任何并行算法总可串行

处理，即可列入串行算法之列，因此T_1作为最快串行算法的运行时间必定有

$$T_1 \leqslant pT_p.$$

由此

$$1 \leqslant S_p \leqslant p,\quad 0 < E_p \leqslant 1.$$

另一重要参数是**冗余度**

$$R_p = O_p / O_1,$$

其中O_p与O_1分别是上述所说的并行算法与串行算法的总运算次数。显然，$R_p \geqslant 1$.

为了便于并行有效性的分析，在前面的假设(3)下，可将运行时间按单位步折合成运算的步数，并且直接将T_1与T_p理解作运算步数。这时便有$O_1 = T_1$. 但p个处理机总共相当于执行了pT_p次运算，一般地$O_p \leqslant pT_p$. 因此很自然地还可定义一个参数，即并行处理机的**利用率**

$$U_p = O_p / pT_p.$$

由定义本身知$U_p \leqslant 1$，且

$$U_p = R_p O_1 / pT_p = R_p (T_1 / pT_p) = R_p E_p.$$

一个目标是构造具有对p为线性加速的算法，从而有效地利用处理机。即希望加速形如

$$S_p(N) = cp - g(p, N),$$

其中N表示问题的尺度(例如线性方程组的阶)，$0 < c \leqslant 1$，$0 \leqslant g(p,N) = O(1)$当$N \to \infty$；$c$应当与$p$无关且接近于1；但并非经常能得到线性加速。线代数许多重要问题最好的加速是

$$S_p(N) = c \cdot \frac{p}{\log p} - g(p, N)(*),$$

(*) $\log p$表示$\log_2 p$，以后不再重申。

虽然低于线性加速，但却可接受。如果最大加速$S_p(N)<d$，d是与p无关的常数，这样的算法很难利用并行性。例如*Horner*法则与连分数可作为“最坏情形”。

1.4 并行算法计算树

首先定义带标号的双权树组$BT(p)$：

（1）在$BT(p)$中具有一个节点的树（树根和树叶相重，标号为0）且其深度为0；

（2）在$BT(p)$中给出一个深度为n的树，如果将所有标号加1，且至多将p个叶子用树  代替，则新的树亦在$BT(p)$中且深度为$n+1$；

（3）$BT(p)$中所有的树均由（1）与（2）两款组成。

注意，对于确定的p，$BT(p)$是树的一个集合，包含深度n不同的和相同的树的集合。例如，图1.5中(*a*)与(*b*)的树都属于$BT(2)$且$n=4$，(*c*)的树属于$BT(3)$但仍有$n=4$.

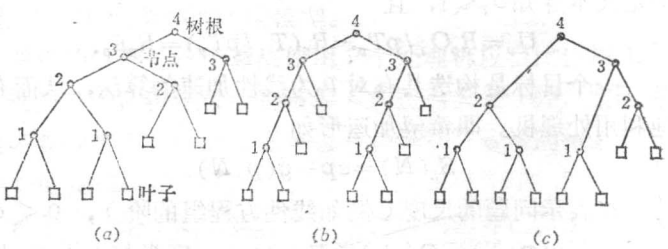


图 1.5

用叶子代表操作数，节点代表双目运算（如+，−，×，/等），因此根据1.3节中每个运算都只需一个单位步的假设，$BT(p)$中深度为n的所有的树对应于使用p个处理机的并行计算的n步。

一个树从根到每个叶子所经过的包括根在内的节点数叫做分枝数。不同的叶子分枝数可以不同，其中最大的分枝数定义作树的高度。高度是树的纵向的一种度量，深度则不仅与纵向有关，而且考虑到横向的结构，显然总有高度≤深度。例如图1.5中的树深度都等于4，(a)与(b)的高度也是4，但(c)的高度是3．

所有高度为h的树中，叶子数与节点数的最大值分别为2^h与2^h-1（包括根在内），从而有深度$\leqslant 2^h-1$．

现设$L(p,n)$是$BT(p)$中深度为n的树的叶子数的最大值，可看成p个处理机用n步算出一个结果所容许的操作数输入的最多个数；$m(p,N)$是$BT(p)$中具有N个叶子的树的深度的最小值，即利用p个处理机从N个输入算出一个结果至少需要的步数。我们有如下基本结果：

1° $L(p,0)=1$.

2° $L(p,n+1)=L(p,n)+min\{p,L(p,n)\}$.

3° $L(p,n-1)<N\leqslant L(p,n)\Longleftrightarrow m(p,N)=n$.

4° 设$k=\lceil \log p\rceil$.(*)则有

$$L(p,n)=2^{min(k,n)}+p\cdot max\{0,n-k\}.$$

5° $$m(p,N)=min\{k,\lceil \log N\rceil\}+max\{0,\lceil (N-2^k)/p\rceil\}.$$

6° 若计算数Q至少要求q次运算，则使用p个处理机计算Q的任一种算法至少需要用$m(p,q+1)$步。

1°与3°显然成立．现在我们给出其它结果的证明。

2°的证明：若$p\geqslant L(p,n)$，则$BT(p)$中深度为n且叶子数达最大值$L(p,n)$的树的所有$L(p,n)$个叶子可用树 

代

(*) $\lceil x\rceil$表示不小于x的最小整数，有$x\leqslant\lceil x\rceil<x+1$；
$\lfloor x\rfloor$表示不大于x的最大整数，有$x-1<\lfloor x\rfloor\leqslant x$.

替，得到$BT(p)$中深度为$n+1$叶子数取最大值$L(p,n+1)$的树，显然$L(p,n+1)=2L(p,n)$；若$p<L(p,n)$，则$L(p,n)$所对应的树标号为1的节点个数应为p个，属这p个节点的叶子数为$2p$，将其中p个叶子用树 

代替，得到深度为$n+1$叶子数取最大值$L(p,n+1)$的树，显然$L(p,n+1)=L(p,n)+p$. ▲

4°的证明： 由$k=\lceil\log p\rceil$有$\log p\leqslant k<\log p+1$，由此$k-1<\log p\leqslant k$或$2^{k-1}<p\leqslant 2^k$.

因深度为n的树的叶子数的最大值是2^n（见图1.6）。如果$2^n\leqslant 2p$，$BT(p)$中存在这种深度为n叶子数为2^n的树，故$L(p,n)=2^n$. 这与4°的结论相符，因为由$2^n\leqslant 2p$有$n\leqslant\log p+1\leqslant k+1$，当$n\leqslant k$时显然，当$n=k+1$时$p=2^k$从而

$$L(p,n)=2^{min(k,k+1)}+2^k\cdot max\{0,k+1-k\}$$
$$=2^k+2^k=2^n.$$

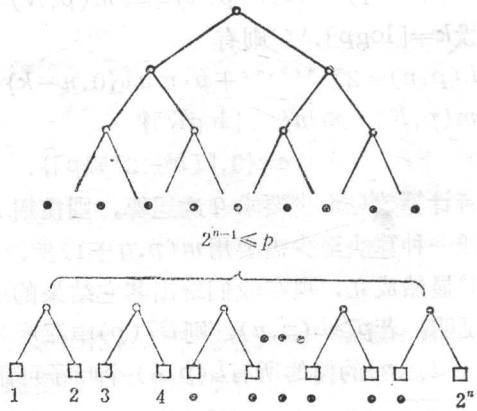


图 1.6