# 并行计算及其进展

# PARALLEL COMPUTING AND ITS EVOLUTION

彼得·J. 丹宁

1985年2月Intel公司宣布了ipsc机，这是一种具有新型结构的计算机，它具备有超过当今最快的超级计算机的潜力，而它的成本却只有后者的几分之一。这机器是根据加利福尼亚计算学院Chuck Seitz领导的称作"宇宙方"的课题制造的。它是由2n个用特殊的高速网络连结起来的处理器板组成的(Intel公司现在提供n=5,6或7的机器)。在这种机器里该网络把处理器连结起来就象这些处理器是在n维空间立方体的角上(所以称为"超立方")，这意味着每一个处理器直接地与别的n个处理器相连，而任意两个处理器之间的最长通路是n条链的跨度。一个含有128个处理器的机器对于最快的算法预计可保持10 MFLOPS的处理率(即每秒1千万次浮点运算)。这种处理率大概是Cray X-MP-2X机实用程序中最快持续处理率的十五分之一，但它的成本只是后者的二十分之一。(Cray X-MP机的最高瞬时运算速度可超过六亿次，但这种速度在实际程序中无法持续)。

1986年初春，Floating Point Systems公司宣布了一个超立方体超级计算机，这种机器的最初的格局是由四个称为四维立方体的柜组成的。每一个四维立方体含有十六个节点。节点用的是Inmos公司的Transputer，其处理能力是16MFLOPS；于是最初的格局具有1GFLOPS的能力(即每秒10亿次浮点运算)。与此同时，在1986年春末，Thinking Machine公司宣布了"联结计算机"(connection machine)，这种机器是将$2^{14}$个(16 384)至$2^{16}$个(65 536)处理器安排在一个高速的网络中组成的。那些节点的单个处理能力并不强而且不包含浮点运算的硬件，但它们集合在一起的时候却能以相当高的速度进行符号运算。

我对这些发展感到非常高兴。这些新的结构已经抓住了许多大规模运算用户的想象力，而另外一些制造厂家不久亦将提供有竞争力的产品。然而，这种概念：即一台机器里包含许许多多的处理器，每一个处理器同时地解决同一个问题的不同部分，是在电子计算机时代出现之初就有的了。

19世纪20年代，麻省理工学院的Vanevar Bush演示了一个能解任意微分方程的通用模拟计算机，它由许多并行工作的部件组成的。19世纪40年代冯诺伊曼(Von Neumann)的论文考虑了在一个离散网络中解微分方程的办法。所有网络中的节点用微分方程的方法去确定相邻结点间的相互影响并并行地进行更新。可能的智能抽象运算的许多模型是以自动机的正则网络为基础的。60年代许多研究者考虑了一种称为并行程序图式的模型。他们力求找出并行系统的行为特征，并探求如何消除并行处理中的一些不利的行为，如，由于器件速度的无法预知而导致运算结果的不确定性。60年代后期在伊利诺大学建的ILLIAC IV计算机是由64个以锁步方式工作的处理机组成的。尽管它的内存有限，加之硬件昂贵，使它无法推广，但它引发了并行算法方面的许多极有意义的工作。70年代的超大规模集成电路(VLSI)技术的出现又激发了用并行计算机组成电路的兴趣以及使用它的算法的兴趣。

尽管人们在并行计算方面的研究一直持续不断，但许多并行计算机仍仅是实验室里的猎奇，许多

彼得·J. 丹宁 (Peter J. Denning) 美国《Communication of the ACM》杂志主编。

并行算法也还是纸上谈兵。是什么东西在这样长时间地阻碍了这种技术的通用化和商业化呢？为什么在长达40年中一直无声无息的想法现在又得到如此重视呢？我想，答案在于串行存储程序计算机在概念上的简单性；在于处理器技术的成本的不断降低；还在于有一种看起来似乎有理的论点：即“大些总是好些”。

串行计算机是由单个处理器，内存和通信子系统组成的。处理器从内存中取出一条一条指令，逐一解码并对标准寄存器或指令中所指明的内存空间执行指定的操作。这种机器的组织方法是很符合“一步接一步算法”的直观想法的。这种算法是大部分通用算法语言和计算理论的基础。它的简单性和广义性使它成为符合绝大部分人胃口的计算模型。

直到70年代微电子技术才趋于成熟，使计算机的结构设计者们能够认真考虑创建由许许多多的处理器所组成的商业性计算机。象ILLIAC IV这样的原型机，其当时所基于的技术对于广泛应用来说，是引不起任何兴趣的。而且成本问题现在对于多处理机系统来说也已不再是个障碍。

“大些总是好些”的论点有好几种形式：一种是所谓的Grosch定律；这是一种40年代凭经验得出的“相关”说：在给定的技术条件下，计算机的成本是与它速度的平方根成反正的。根据这种推算，一个快四倍的计算机，成本仅增加1倍。那么，为什么不寻求尽可能快的计算机呢？

问题在于能装进一个箱子的计算机的能力总是有限的。其解释（有时称为光速论）是这样说的：电子管中光的速度是$3\times10^8$米/秒，而硅片中的信息传输率在考虑开关延迟的因素之后至多是$3\times10^7$米/秒。一个直径比一英寸（约3厘米）略大的硅片能用$10^{-9}$秒的时间传递一个信号。由于非并行处理的芯片在单个传递中至多只能执行一个浮点运算，这样的芯片能支持大约$10^9$FLOPS（10亿次浮点运算/秒）。由于这种原因，电子专家们并不期望用当代技术制造的单处理器会大大超过每秒10亿次的浮点运算率。当代的超级计算机的处理速度均约为这种速度的十分之几。

由于题目的大小和所需的计算能力呈非线性关系，这种限制是一个严重的问题。 例如，一个$n\times n$矩阵的乘法要大约$n^3$次运算。一个增至二倍大小的题目则需要一个能增快八倍的计算机才能在相同的时间内完成。另一种说法是，一个二倍快的处理器在原来相同的时间内只能做$n\times2^{1/3}$大小的矩阵乘法，这就是说大约只能处理25%的问题。

换句话说，要在同样的时间内解决线性增长的问题必须引出为解这一问题所必需的超线性增长。总有一个时候，我们会耗尽现有机器的能力，最终我们将需要远远大于单处理器计算机的能力。

“大些总是好些”的另一种论据是以对一个给定的处理器和n个具有n分之一速度处理器作比较为依据的。假定一个工作可分成n个相等的步骤。而且这个工作在这机器上的执行时间是T秒，在这一组小处理器中将需nT的时间完成。因为每一步是以原速度的1/n完成的。由于其工作步是串行的，每一时刻的每一作业只能使用一个小处理器。于是，n个作业可以并发地使用这一组小型处理器，合起来计算，每T秒完成一个作业。这种多处理器其实和一个单个处理器的处理能力一样，但应答时间要快得多。那么，为什么不寻求尽可能大的处理器呢？

这种论据的弊病在于它假定作业是内在地串行的。反之，若假定每一作业能分成n个相等的独立的组成部分。用这种方法（称为并行分解），则每一作业的组成部分可在它自己的小处理器中运行。假定在处理器之间的分配输入时间和综合结果输出的时间是可以忽略不计的，这n个部分能并行地在T秒中完成。如果每个小处理器的成本不超过单个处理器的1/n，这样的作法就有好处（如果应用Grosch定律，这n个慢处理器的总成本将是单个快处理器的$n^{1/2}$倍。但因为使用不同的生产技术，Grosch定律在这里不适用：快处理器的生产一般是小量的、特制的，而便宜的慢处理器一般是大量生产的）。

所以，结论是，作业的并行分解能够使多处理器和单个处理器有相同的处理能力和应答时间，但其成本只要后者的n分之一。

这个结论表明，除了硬件的速度障碍之外还有另一个障碍：软件的障碍。我们善于理解串行算法，然而对指挥并行处理器的算法没有多少经验。我们还不知道如何为这种新机器编程序。多数通用语言源本就是串行的，程序总是一个一个指令地执行。

这种软件障碍，即那些软件技术中限制我们有效地使用并行硬件技术的东西并不是一推即跨的一堵砖墙，却是又深又密的一大片丛林。想要通过它，就得坚持不懈地劈荆斩棘，逐步前进。这里，我想描述一下，在这个道路崎岖的旅程中，大约要通过四个阶段：第一阶段已将近完成，第二阶段目前正在进行着，第三阶段尝试性的工作已经开始，第四阶段则还甚为遥远。

在第一阶段中，人们把并行概念引入单台计算机这类硬件中来。这方面包括处理器中的多功能单元，多排存储，流水线式的执行指令和将一个指令应用到一个数据流的向量流水线等一些老例子。这些老的概念眼前正在对一种称之为简化结构的独特机器（RISCs）进行着一轮新的改进。RISCs是很快的，其原因一方面

是它们的指令集合是很小的，另一方面是它们的编译程序能够仔细地分析程序，以便产生使指令流水线经常保持最忙状态的目的码。

大多数计算机的操作系统都备有准备就绪队，它能列出所有正在等待处理器执行的所有进程的名称。若加上一点变化，操作系统就能扩充到允许许多个而不是一个处理器为等待就绪队服务。这种原理正在一种称之为对称多处理器的计算机(如Sequent Balomce 21000 或Encove Multimax小型计算机)上作探索。早在60年代后期，有些制造厂家就曾提供过多至四个处理器的机器。今天市场上已有包含多达三打(36个)程序处理器的多处理器。只要插进更多的处理器板这种机器的能力还能得到加强。

第一阶段对于用户来说是比较容易的。因为对所有可见到的软件技术几乎不需做任何改变。比如说，一旦Fortran语言升级为可以处理向量数据，用户就可以通过很小的变化，重新用Fortran语言组写新的算法来探索Cray-1向量计算机硬件。同样，一些关键子程序，象快速富里叶变换程序也可以新编成矩阵或向量计算机所用的码，而不必改变原来的程序调用规则。一旦UNIX的核心得以升级，允许多处理器可同时运行，那就可以试探用一个多处理机而不必改变UNIX操作系统的命令语言。所以，并行处理第一阶段所需的变化可以只限于编译程序、子程序库和操作系统，它对高级语言的用户来说绝大部分是觉察不到的。第一阶段目前正在顺利进行，几年之内就会成熟。

绝大多数程序总是被假定是串行执行的。正是这种人为的假定，而不是硬件本身限制了我们去充分应用并行计算机的能力。但我们对计算能力的强烈需求将迫使我们从第一阶段走向并行处理的第二阶段。

在第二阶段，在不同的机器上并行地执行合同程序的情况变得非常明朗。程序之间通过高速通讯键交换数据而不是传递共享的数据地址。超立方体系结构就属于这一类。

在第二阶段，改变编程语言是需要的。必须在建立过程中和各过程间增加交换信息用的语句。牛津大学的C. A. R. Hoare曾建议过一种很简单的过程间通讯的表达方法，即，将一种可变的变量(称为'口'，port)增加到算法语言中去，在需要从别的过程得到数据值的地方，让程序中包含一个'口'的名字，后面再跟一个问号。在一个值需被送出去的地方，则令程序中包含有一个'口'的名字，在其前面有一个感叹号。送者和收者必须在一个'口'会合，然后数据才能真正在它们之间传送。例如，一个从输入口读入X，Y的值并把它们的和传递到口Z的过程可能包含语言!Z=X?+Y?。在计算这种表达式的时候，过程将等待直到X和Y的口都有输入值，然后将计算结果送到口Z。这些表达方法被用在一种称为Occam的派生Pascal语言中。

这种被Hoare称为CSP(互相通讯的串行过程)的模型，比它的概念上的表兄弟——串行过程模型更难编程。其原因是程序员必须指明许多个(而不只是一个)串行过程以及它们之间的通讯。例如超立方计算机的程序员就面临着写2n个程序(每一个处理器一个)和理解(2n)n这样潜在的通讯通路的任务。过程间互相交叉的执行和它们之间的相互作用是非常难以对付的：不但存在着许多可能性，而且会产生新形式的错误，例如，结果的不确定性与死锁等会在并发过程中产生。需要有新的编程环境去管理这些新的复杂性。Larry Snyder的Poker编程系统就是这方面的一个例子。

操作系统正在发生更深刻的变化。操作系统正被扩充来管理计算机网络而不单是单台计算机。接口仍将非常简单，同一个操作能够处理各种资源，不论它是近处的或远程的。这种分布式操作系统最终将使需要跨跃不同类型计算机的运算成为可能。

这些变化，尽管现在还不广泛，但一直在顺利进行着。第二阶段的进展，在计算技术方面遇到的限制会比编程技术方面更多些。例如，如何将逻辑模拟、流体动力学，化学特性，有限元结构分析，地震模拟等程序的核心部分分解成能在不同计算机上运行的多个组成部分？解线性方程或偏微分方程的算法在分解以后仍将保持它的数值特性和稳定性吗？我们的基础数学和统计学软件需要做多大多深入的返工才能在分区式的机器上运行？总之，将来需以很大的努力向这些已编成程序的基本算法去探求新的知识。

因此我预期在第二阶段会有很多算法方面的研究。我预期会发现大量的关于应用于新的划分式计算机的算法的细节信息。但是我们已经知道的繁杂性将驱使我们设法把细节藏存在简单的接口程序的后面，并去发现能自动执行算法划分的编译程序和操作系统。这一切，将迫使我们走向第三阶段。

在第三阶段，新的算法语言将使并行算法变成内含的，而它们的编译程序将担负起划分程序的重担。传统的算法语言是祈使式的，它的语句均被视为指导处理器的命令。而新的语言却是功能式的。这意味着它们的语句是一种能表述组合功能的表述式。这方面的例子很多，如：LISP(用于符号字符串的处理，这些符号串是用于表述表达式和值)；VAL和LUCID(用于数据流运算，在数据流运算中，每当所有的操作数都

就绪时，操作就立即被触发）；REDUCE 和MACSYMA（用于符号代数表达式的处理——微分，积分并化简为最简式）；SQL（用于关系数据库的查询）；APL（用于数量向量的函数运算）和PROLOG（用于运算演译逻辑系统中的表达式）。这里并没有包含所有的例子。这些语言的共同特点是，它们只表达运算的结果而不是表达获得这结果的方法，这就为编译程序和操作系统提供了很大的灵活性，可以把这些运算分配到很多处理元件中去处理。

目前，计算机学科外的人，对这些语言的经验还很少。有充分的理由相信这些语言在解决别的领域中的问题时是不完备的。很自然，许多问题会被分解成许多部份，很多块，以求单独地、分开得到解答，因此在算法形式化的时候必须明白地做这种划分。不同的部分可能需要在不同的机器上，用不同的语言去解答。流体动力学计算是一个例子，那里一个测试区域可以被分成许多小区，每一个小区用不同的算法去解决。而且，不仅每个小区是一个块，它可能分成更小的 不同的小块。例如，某一个块是从一个区域描述中导出的符号方程式，另一个块含有符号处理程序以便把符号方程式约简并生成对应的Fortran 程序，第三个块则去执行这个程序。但，第三阶段，在不均称小块的语言处理方面，以及对它们之间的边界作出定义方面所表现的无能将驱使我们向第四阶段迈进。

第四阶段将会有非常高水平的用户接口。它能够和科学家和工程师相互应答，其抽象的水平就象科学家们和工程师们之间相互应答的水平一样。这接口将有助于我们使用自然语言、图形，讲话和以形式表达式给出特定领域中诸问题的精确的公式化表达。这些接口系统将把这些描述转换为自然的块块，并建立这些块块的功能式的表达，把这功能式的表达转化为特定的程序，把每一程序再转变成可执行码，请求执行程序码，最后收集运算结果并用该领域高水平的抽象方式显示出来。第四阶段的系统将采用今天的基于知识的系统（指专家系统：译者）技术做为建造的模块。

我对计算技术发展的大略阶段和科学问题算法形成过程中抽象级别关系之间的类似性有很深的印象。从最抽象的级别到最具体的级别是：

4.精确描述。建立问题模型和解问题时观察到的限定条件的精确描述。这种描述可以使用自然语言，数学表达式，和各学科用的特殊的技术术语和特殊的表达方法。

3.抽象的算法。根据给定的描述指定解问题的一般策略。例如，如何将问题划分成块；使用什么迭代策略；数据交换和收敛策略。（在这一级，我们并不关心数据表达的细节和机器的能力等等）。

2.具体可行的算法。把解问题的程序模块建立起来或联接起来。每一模块可能需用不同的语言表达。

1.机器编码。用机器的指令集建立程序编码以及指派操作系统去执行已指定的运算细节。

这种解问题的过程是一系列变换的过程，从各学科抽象水平上的精确表达直到硬件抽象水平上的算法的精确描述。

上述的软件技术发展的这四个阶段与解问题的四个原理的抽象级别一一对应。四阶段中发展的每一个阶段就是上述编程工具开发级别关系中的一步。当今计算机科学并没提供多少工具去支持这种在较高级别上的转换：这正是为什么目前尚没有第三和第四阶段的系统存在的原因。尽管并行算法在较高的级别上较不容易看到，但它的重要性并不亚于其他级别。它将提供工具去实现操作第三阶段和第四阶段的系统所需要的功能。

（李堂秋译）

---

（上接33页）

对于零部件工业尤其要采取优惠政策，在税收、分配、外汇留成方面给予照顾，鼓励直接出口。

汽车工业利用外资的过程还刚开始，由于缺乏经验和完整的支持系统，因而在合营企业中所出现的种种问题是在所难免的，我们的责任是正视问题，剖析问题，找出问题滋生的因素，对症下药，切莫讳疾忌医。

行业内外要努力创造条件，对利用外资、发展民族汽车工业之举给予充分支持。中国汽车工业是振兴中华，加速现代化民族工业的建立，并使之立于世界之林是历史赋予我们的责任。

历史的经验和实践告诉我们：

在国际合作中，竞争往往是主要方面，谁首先能赢得时间，谁就可能赢得所期待的一切。