第一篇

并行性理论

概　要

本篇为理论篇，介绍计算机模型、程序行为、结构选择、可扩展性、可编程性和有关并行处理的性能等问题。这些课题组成了设计高性能计算机和开发支撑软件与应用系统的基础。

实际计算机模型包括共享存储型多处理机、消息传递型多计算机、向量超级计算机、同步处理机阵列和大规模并行处理机。本篇也要介绍理论并行随机存取机模型(PRAM)，并讨论 PRAM 模型与实际结构模型之间的区别。为了直接在集成电路中实现并行算法，还要介绍 VLSI 复杂性模型。

本篇还要讨论网络设计原理和并行程序特性。这包括相关性理论、计算粒度、通信时延、程序流机制、网络特性、性能定律和可扩展性等内容。从抽象模型到硬件机器、软件系统和性能评价这样来讨论并行性理论将会加深我们对并行计算机的理解。

第一章　并行计算机模型

并行处理已经成为现代计算机的关键技术，其推动力来自实际应用对高性能、低价格和持续生产力日益增长的要求。从常用的多道程序、多处理或多重运算也可看到，并发事件在当今高性能计算机中是极为常见的。

并行性在不同的处理级别中可表现为多种形式，如先行方式、流水方式、向量化、并发性、同时性、数据并行性、划分、交叉、重叠、多重性、重复、时间共享、空间共享、多任务处理、多道程序、多线程方式和分布式计算。

在这一章中我们将为并行计算机、向量超级计算机、多处理机、多计算机和大规模并行处理机的物理系统结构建立模型。同时也要介绍理论机器模型，其中包括并行随机存取机(PRAM)和 VLSI(超大规模集成)电路的复杂性模型。本书还将研究系统结构的沿革，介绍各种硬件和软件子系统，以便为详细论述后续章节铺平道路。

1.1　计算技术的现状

现代计算机是由性能很高的硬件设备和驱动这些设备的大量软件包构成的。为了评估目前的计算技术，我们首先要回顾一下计算机发展史上的里程碑，然后再重点考察现代计算机系统所用的关键硬软件子系统。最后，将探讨有里程碑意义的系统结构发展过程。当然，在谈到计算机性能时，一定会说明硬软件的基本作用。

1.1.1　计算机发展史上的里程碑

计算机经历了机械和电子两个主要发展阶段。1945 年前，计算机是由机械或机电部件构成的。最早的机械计算机可追溯到公元前 500 年中国所用的算盘。这种算盘用手工操作完成逐位进位的十进制运算。

1642 年，法国 Blaise Pascal 制造了一台机械加法器/减法器。1827 年，Charles Babbage 在英国设计了一台用于多项式计算的差分机。1941 年，德国的 Konrad Zuse 构造了第一台二进制机械计算机。Howard Aiken 最早提出十进制机电计算机，这在后来由 IBM 于 1944 年制成了 Havard Mark I。Zuse 和 Aiken 设计的两种机器都用于通用计算。

显然，用传动机件来实现计算和通信将大大地影响机械计算机的计算速度和可靠性，现代计算机的明显标志就是采用了电子元件。这样，机械计算机的传动部件已被电子计算机的具有高流动性的电子所取代，机械齿轮或杠杆传递信息也被传播速度几乎为光速的电信号所取代。

计算机的更新换代　过去 50 年，电子计算机已经历了五次更新换代。表 1.1 列出了它发展史上五次换代的概貌。前三代中，每一代持续大约十年，第四代的时间跨度约为十五年。我们现在恰好进入第五代，这一代用的是在一个硅芯片上具有一百多万个晶体管的

处理器和存储器件。

各代的划分主要是以硬件和软件技术的明显变化为标志的。表 1.1 所列的各项说明了每代新的硬件和软件特征。早先几代的大部分特征都反映到了后续各代中。换句话说，即最新一代计算机继承了前面各代的全部优点，而克服了它们的不足。

硬件的进展 就硬件技术而言，第一代计算机(1945—1954)将电子管和继电器存储器用绝缘导线互连在一起。第二代(1955—1964)的标志是采用分立式晶体管、二极管和铁氧体磁芯，用印刷电路将它们互连起来。

第三代(1965—1974)计算机开始在逻辑和存储器方面采用小规模或中规模集成(SSI 或 MSI)电路和多层印刷电路。第四代(1974—1991)采用的是大规模或超大规模集成(LSI 或 VLSI)电路。当计算机由第三代进入第四代时，磁芯存储器被半导体存储器所取代。

第五代(1991 至今)计算机的突出标志是采用 VLSI 工艺更加完善的高密度、高速度处理机和存储器芯片。例如，64 位 150MHz 微处理机现在可用具有 100 多万个晶体管的芯片制成，4M 位动态随机存储器(RAM)和 256K 位静态 RAM 也已广泛用于当今的高性能计算机中。

可以设想，在未来的十年中，可在一片 CMOS 芯片上用 5000 多万个晶体管制成 4 台微处理机，而对 64M 位动态 RAM 的需求量将十分可观。

表 1.1 电子计算机五次换代

代	技术和系统结构	软件和应用	代表性系统
第一代 (1945—1954)	电子管和继电器存储器，由程序计数器(PC)和累加器驱动 CPU，定点运算。	机器/汇编语言，单用户，无子程序链接，用 CPU 程序控制 I/O。	ENIAC， Princeton IAS， IBM 701
第二代 (1955—1964)	分立式晶体管和磁性存储器，浮点运算，I/O 处理机，多路存储器存取。	用有编译程序的高级语言、子程序库、批处理监控程序。	IBM 7090， CDC 1604， Univac LARC
第三代 (1965—1974)	集成电路(SSI/-MSI)，微程序设计、流水线、高速缓存和先行处理机。	多道程序设计和分时操作系统，多用户应用。	IBM 360/370， CDC 6600， TI-ASC，PDP-8
第四代 (1975—1990)	LSI/VLSI 和半导体存储器，多处理机，向量超级计算机，多计算机。	用于并行处理的多处理机操作系统、语言、编译器和环境。	VAX 9000， Cray X-MP， IBM 3090， BBN TC-2000
第五代 (1991—现在)	ULSI/VHSIC 处理机、存储器和开关，高密度封装技术，可扩展系统结构。	大规模并行处理，重大挑战性应用，异构处理。	Fujitsu VPP500， Cray/MPP， TMC/CM-5， Intel Paragon

第一代 从系统结构和软件的观点来看，第一代计算机由单个中央处理机(CPU)构成，CPU 用程序计数器、转移指令和累加器顺序完成定点运算，它还与所有的存储器存取

和输入/输出(I/O)操作有联系,所采用的是机器语言或汇编语言。

代表性系统有:ENIAC(电子数字积分计算机),它由宾夕法尼亚大学莫尔学院于1950年制成;IAS(高级研究院)计算机,它是由John von Neumann,Arthur Burks和Herman Goldstine于1946年在普林斯顿设计成功的;IBM 701,它是1953年由IBM制造的第一台电子存储-程序商用计算机。在早期的计算机中都没有实现子程序链接。

第二代 变址寄存器、浮点运算、多路存储器和I/O处理机都是从第二代计算机开始出现的。Fortran, Algol和Cobol等高级语言(HLL)是与编译器、子程序库和批处理监控程序一起被采用的。寄存器传送语言是由Irving Reed(1957)为数字计算机的系统设计开发的。

代表性系统有:1962年制成的IBM 7030(Stretch计算机),它具有指令先行和纠错存储器等特征;1959年制成的Univac LARC(利弗莫尔原子研究用计算机);还有60年代的CDC 1604。

第三代 第三代计算机的代表性产品是IBM/360—370系列、CDC 6600/7600系列、Texas(德克萨斯)仪表公司的ASC(高级科学计算机)、60年代中期至70年代中期的Digital Equipment公司的PDP-8系列。

微程序控制在这一代开始普及。为了缩小CPU和主存储器之间的速度差距,采用了流水线和高速缓存,同时为了使CPU和I/O操作在多用户程序之间交叉进行,还采用了多道程序的办法。这又促进了有虚拟存储器的分时操作系统(OS)的发展,使资源多路切换得到共享。

第四代 计算机发展到第四代时,出现了用共享存储器、分布存储器或向量硬件选件的不同结构的并行计算机,开发了用于并行处理的多处理操作系统专用语言和编译器。同时产生了用于并行处理或分布计算的软件工具和环境。

代表性系统包括:VAX 9000,Cray X-MP, IBM/3090VF, BBN TC-2000等。在这十五年间(1975—1990),并行处理技术逐渐成熟并进入生产主流。

第五代 第五代计算机刚开始出现。这些机器最重要的特点是进行大规模并行处理(MPP)。在MPP系统中,采用可扩展的和容许时延的系统结构,用的是VLSI硅片、砷化镓技术、高密度组装和光技术。

到90年代中期,第五代计算机的目标是要达到Teraflops(每秒10^{12}次浮点运算)。目前,用共享虚拟存储的异构计算机网络来解决大型问题的异构处理(heterogeneus processing)技术正在研究形成之中。最近发布的项目,如Fujitsu的VPP500,Cray Research的MPP,Thinking Machines公司的CM-5和Intel超级计算机系统Paragon即是第五代MPP系统的代表。

1.1.2 现代计算机的组成

下面我们将从并行处理的角度简要介绍现代计算机系统的硬件、软件和程序设计等组成部分。

计算问题 计算机系统结构不应只限于裸机硬件结构的概念,这一点早已为人们所认识。现代计算机是一种包括机器硬件、指令系统、系统软件、应用程序和用户接口的集成

系统。这些系统的组成如图 1.1 所示。现实生活中对问题要求快速而精确地求解推动了计算机的广泛使用。各种求解方法可能需要不同的计算资源，这与求解问题的性质有关。

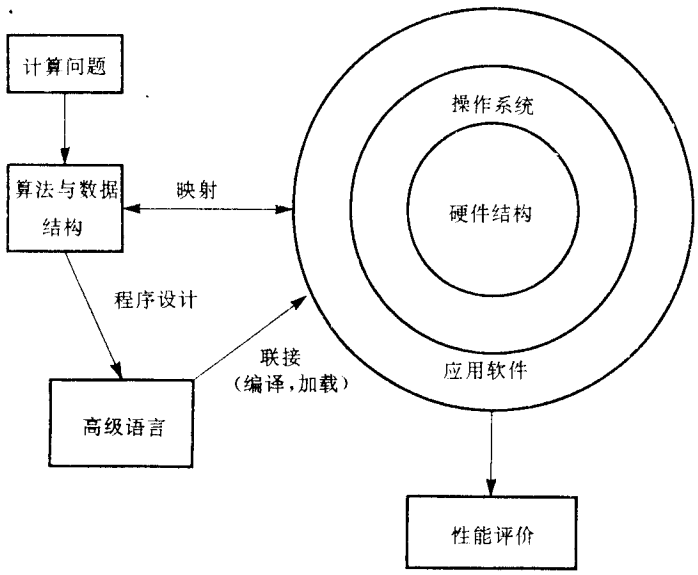

图 1.1　现代计算机系统的组成

对于科学技术中的数值计算问题，需要用复杂的数学公式进行枯燥乏味的整数或浮点运算来解决。对于商务和公务管理中的字母数字处理问题，就需要使用精确的事务处理、大型数据库管理和信息检索操作来解决。

对人工智能(AI)问题，则需要用逻辑推理和符号处理。这些计算问题分别叫做数值计算、事务处理和逻辑推理。求解某些复杂问题可能需要综合使用这些处理方法。

算法和数据结构　为了确切表示计算问题中的运算和通信，需要各种专门的算法和数据结构。大部分数值算法是确定性的，用的是规则结构数据。符号处理则用启发式或不确定性搜索方法对大型知识库进行操作。

问题的形式化和并行算法的开发常常需要理论家、实验工作者和计算机程序设计员之间相互配合地进行学科交叉。已有许多著作论述了确定性算法和启发式算法在并行计算机上的设计和映射问题。在本书中，相对于并行算法设计和分析来说，我们将更加关注资源的映射问题。

硬件资源　计算机的系统结构可如图 1.1 用三个同心圆来表示。现代计算机系统通常通过硬件资源、操作系统和应用软件的综合效果展示其功能。处理机、存储器和外围设备组成了计算机系统的硬件核心。我们将要讨论的是指令系统处理机、存储器组织、多处理机、超级计算机、多计算机和大规模并行计算机。专用硬件接口常常安装在例如终端机、工作站、光学页式扫描器、磁性墨水字符识别器、调制解调器、文件服务器、语音数据输入、打印机和绘图机等 I/O 设备中。这些外围设备可以直接或通过局域网和广域网与主机相连。

此外，还需要各种软件接口程序。这类软件接口包括文件传送系统、编辑程序、字处理器、设备驱动程序、中断处理程序、网络通信程序等。有了这些程序，用户程序就能很方便地移植到不同的机器结构上去。

操作系统 一个有效的操作系统能管理用户程序执行过程中的资源分配和再分配。我们将在第十二章中讨论多处理机和多计算机的扩展 UNIX。对 Mach/OS 内核和 OSF/1 我们将讨论多线程的内核功能、虚拟存储器管理、文件子系统和网络通信服务等问题。除操作系统外，开发的应用软件必须对用户有用，设计的基准程序必须有利于性能评价。

映射是一种算法结构与硬件结构相匹配的双向过程。有效的映射会使程序设计者得益，并产生较好的源代码。算法和数据结构到机器结构的映射包括处理机调度、存储器映象、处理器间的通信等。这些问题通常都与系统结构有关。

人们一直在为各种计算机系统结构寻求理想的映射方法。这些映射方法的实现取决于高效的编译器和操作系统支持。并行性可以在算法设计时、编写程序时、编译时和运行时进行开发。在这些级别上开发并行性的技术构成了并行处理技术的核心。

系统软件支持 用高级语言开发有效的程序需要软件的支持。用 HLL 编写的源代码必须先用优化编译器翻译成目标代码。编译器将变量分配给寄存器或存储单元，并给操作符保留功能部件。汇编程序用来将编译后的目标代码译成机器硬件能识别的机器代码。加载程序则通过操作系统内核启动程序执行。

资源的联用是通过编译器、汇编程序、加载程序和操作系统内核将实际机器资源提交给程序执行。这一过程的有效与否决定了硬件的利用率和计算机的可编程性。目前，由于现有的语言是早先为顺序计算机开发的，因此并行程序设计对大多数程序员来说仍是非常困难的。程序员常常不得不考虑与硬件有关的特性去编程，而不能以通用和可移植方式进行并行程序设计。比较理想的是，需要我们去开发一种与系统结构无关的语言、编译器和软件工具的并行编程环境。

对于开发并行语言，我们将着眼点放在语言执行的效率、对不同机器的可移植性、与现有的顺序语言的兼容性、并行性的表达和编程的简便性等上面。我们可以设计一种新的语言，也可以尝试逐步扩展现有的顺序语言。新语言有用显式高级结构描述并行性的优点，但是新语言往往与现有语言不兼容，而需要新的编译器或者通过新的步骤才能利用现有的编译器。大部分系统选用的是语言扩展方式。

编译器支持 改进编译器有三种途径：预处理程序、预编译器和并行化编译器。预处理程序采用顺序编译器和目标计算机的低层程序库实现高级并行结构。预编译器需要程序流分析、相关性检查和有限的优化来检测并行性。第三种途径则要求彻底开发一种并行化或向量化的编译器，使之能自动检测源代码的并行性，并能将顺序代码变换成并行结构。这些方法将在第十章详细讨论。

联接过程的效果取决于预处理程序、预编译器、并行化编译器、加载程序和操作系统支持的功效。由于程序行为的不可预测，现有的编译器在检测所有类型的并行性时都不是完全自动或完全智能进行的。通常，是将编译器命令插入源代码，帮编译器做出较好的结果。这样，用户可与编译器进行交互重构程序，这已被证明对提高并行计算机性能是十分有用的。

1.1.3 计算机系统结构的发展

计算机系统结构的研究涉及硬件组织和程序设计/软件需求两方面。从汇编语言程序员看，计算机系统结构可由其指令系统来抽象。它包含操作码、寻址方式、寄存器、虚拟存储器等。

从硬件实现的观点来看，抽象机由 CPU、高速缓存、总线、微代码、流水线、物理存储器等组成。因此，系统结构的研究覆盖了指令系统结构和机器组织实现两个方面。

过去 40 年间，计算机系统结构在发展，但并没有发生革命性的变化。可以证明，持续特性才是系统提供的实际性能。在图 1.2 中，我们先从 von Neumann(冯·诺依曼)结构的顺序机执行标量数据开始谈起。顺序计算机是从位串行操作到字并行操作、从定点运算到浮点运算改进过来的。由于 von Neumann 系统结构的程序中的指令是顺序执行的，所以速度很慢。

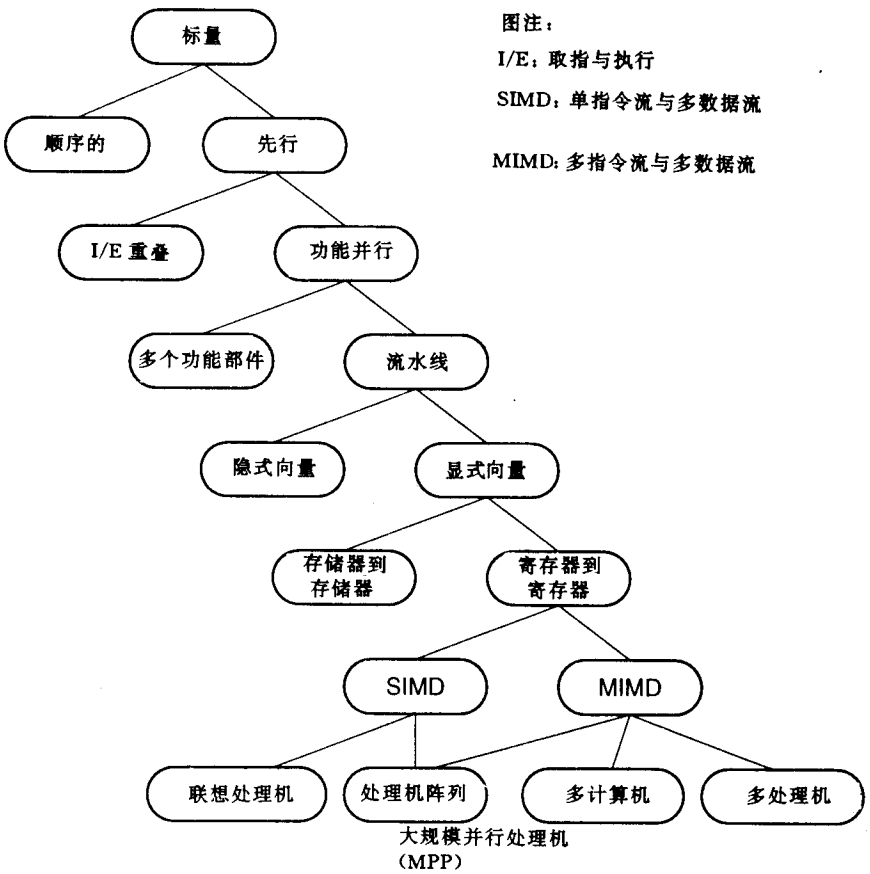

图 1.2　从顺序标量计算机到向量处理机和并行计算机的系统结构的演变

先行、并行性和流水线技术　用先行技术预取指令可使 I/E(指令读取/译码和执行)

操作重叠起来，从而能实现功能并行性。支持功能并行性的方法有两种：一种是同时使用多个功能部件，另一种是在不同处理级别实施流水线技术。

后一种方法包括流水线指令执行、流水线算术计算和存储器存取操作。流水线技术已被证明对向量数据串重复执行相同的操作颇具吸引力。向量操作原先是标量流水线处理机用软件控制循环隐式实现的。

Flynn 分类法 Micheal Flynn(1972)根据指令和数据流概念提出了不同计算机系统结构的分类法。如图 1.3a 所示，传统的顺序机被称为 SISD(单指令流单数据流)计算机。向量计算机用标量和向量硬件装备，或以 SIMD(单指令流多数据流)机的形式出现(图 1.3b)。并行计算机则属 MIMD(多指令流多数据流)机。

MISD(多指令流单数据流)机表示于图 1.3d。在执行不同的指令流时，同一数据流通过处理机线性阵列。这种系统结构也就是所谓流水线执行特定算法的搏动阵列(systolic arrays)。

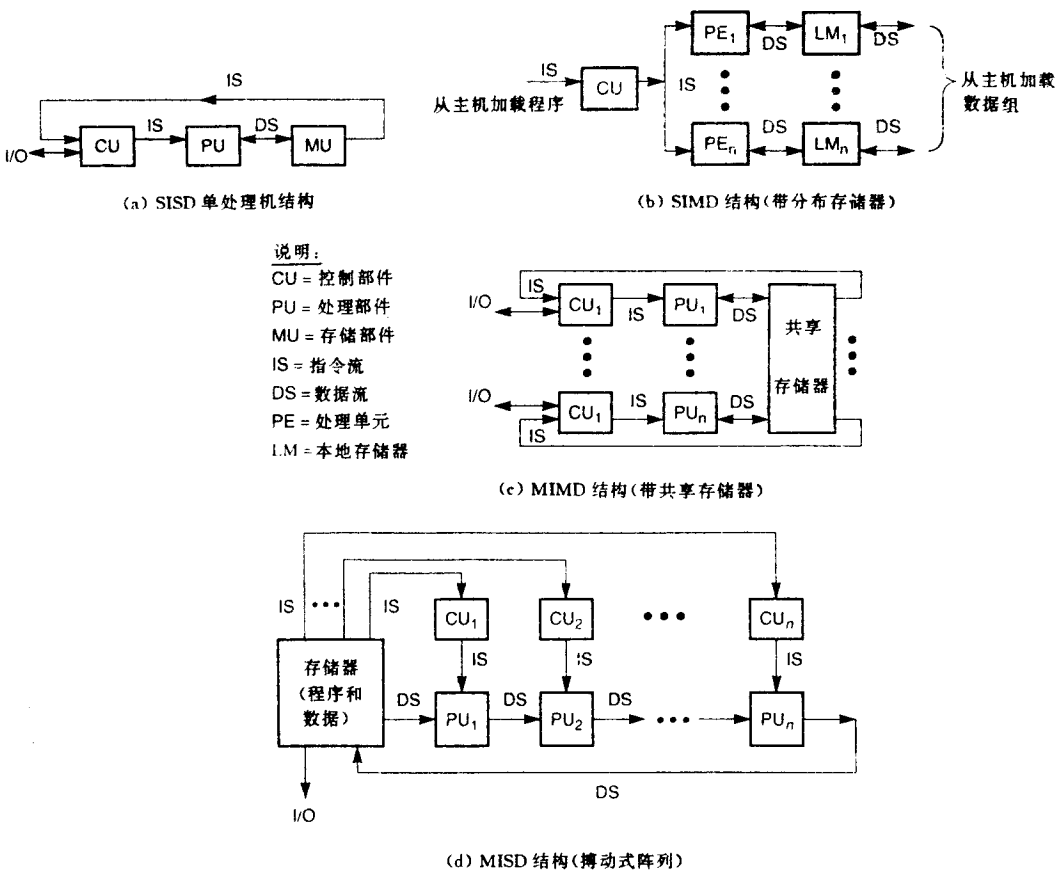

图 1.3 计算机系统结构的 Flynn 分类(引自 Micheal Flynn，1972)

四种机器模型中，过去制造的大部分并行计算机都采用了适于通用计算的 MIMD 模型。SIMD 和 MISD 模型更适合于专用计算。因此，在商用机中 MIMD 模型最通用，

SIMD 次之，MISD 最少用。

并行/向量计算机 真正的并行计算机是那些以 MIMD 模式执行程序的计算机。并行计算机有两大类，即共享存储型多处理机和消息传递型多计算机。多处理机和多计算机之间的主要差别就在于存储器共享和处理机间通信机制的不同。

多处理机系统中的处理机通过公用存储器的共享变量实现互相通信。多计算机系统的每个计算机结点有一个与其它结点不共享的本地存储器。处理机之间的通信通过结点间的消息传递来实现。

显式向量指令是随向量处理机的问世而出现的。一台向量处理机可以装备有用硬件或固件并发控制的多条向量流水线。流水线向量处理机有两种类型。

存储器-存储器结构将向量操作数流直接从存储器取至流水线，然后再送回存储器。寄存器-寄存器结构利用向量寄存器作为存储器与功能流水线之间的接口。向量处理机结构将在第八章进行讨论。

系统结构树（图 1.2）的另一重要分支由同步进行向量处理的 SIMD 计算机组成。SIMD计算机开发的是空间并行性，而不是象流水线计算机那样的时间并行性。SIMD 计算是用同一种控制器同步控制处理单元（PE）阵列来实现的。联想存储器可用来制造 SIMD 联想处理机。SIMD 机将在第八章中与流水线向量计算机一起进行讨论。

开发层次 根据 Lionel Ni 的最新分类法（1990），并行计算机的分层开发可表示于图 1.4。硬件配置在不同的机器中都不会一样，即使模型相同的机器也是如此。计算机系统中处理机的地址空间在不同系统结构之间也不会相同，这主要取决于与机器相关的存储器组织。这些性能应该考虑与目标应用领域相适应，由设计人员决定。

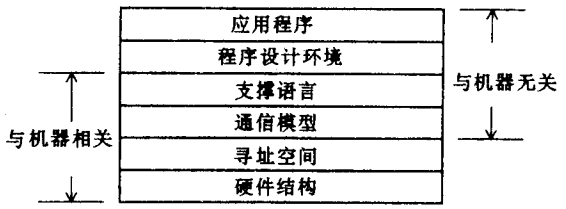

图 1.4 计算机系统并发的六个层次（承 Lionel Ni 许可，1990）

另一方面，我们也要开发与机器无关的应用程序和程序设计环境。由于与机器结构无关，用户程序就可以最小的代价移植到许多计算机上。高级语言和通信模型与计算机系统的结构选择有关。从程序员的观点出发，这两层对系统结构应该是透明的。

目前，Fortran、C、Pascal、Ada、Lisp 已为大多数计算机所支持。但是，通信模型、共享变量与消息传递几乎都是与机器相关的。用元组空间（tuple space）的 Linda 方法可提供一种对并行计算机的结构透明通信模型。这些语言特性将在第十章中进行讨论。

应用程序设计员对系统结构透明性比较有兴趣，但是内核程序设计员们不得不要更多地利用硬件支持。作为一位优秀的计算机结构设计师，他必须从两方面去考虑问题。编译器和操作系统应该设计成能为程序员消除尽可能多的结构限制。

新的挑战 并行处理技术是 40 年来在微电子、印刷电路、高密度封装技术、高性能处

理机、存储系统、外围设备、通信通道、语言开发、编译技术、操作系统、程序设计环境和应用问题等研究和工业发展的产物。

硬件技术的迅速发展已使建造适于并行处理的新一代计算机的经济可行性显著增加。但是，阻碍并行处理进入生产、成为主流技术的主要问题还是在软件和应用方面。

到目前为止，对并行和向量计算机编程仍是非常困难和费力的事。我们必须在软件领域中努力争取较大的进展，以便为高性能计算机开发出友好的用户环境。我们必须对整个新一代的程序员进行高效开发并行性编程方面的训练，使高性能计算机能用来快速而准确地解决科学、工程、管理、社会及国防等问题。

实际生活中具有代表性的问题有：天气预报建模、VLSI 电路的计算机辅助设计、大型数据库管理、人工智能、犯罪控制和国防战略研究等，还可以举出一些新的例子。并行计算机的应用领域正在不断扩大。读者要很好了解可扩展的计算机系统结构并熟悉并行程序设计技术，才能更好地面对未来的计算挑战。

1.1.4 性能的系统属性

理想的计算机系统的性能要求机器功能和程序行为之间有良好的匹配。机器功能可用较好的硬件技术、改进的系统结构特性和有效的资源管理等措施来提高。但是，程序行为很难预测，因为它与应用及运行时的条件有密切关系。

还有其它许多影响程序行为的因素，其中包括算法设计、数据结构、语言效率、程序员的技能以及编译技术等。仅仅只对上述少量方面有所改进而不去触及其它部分，要达到硬件和软件之间良好的匹配是不可能的。

此外，机器性能会随程序而变化。这就说明在实际应用中要达到峰值性能的目标是不可能的。另一方面，又不能说机器具有某种平均性能。实际上，所有的性能指标和测定结果都一定是由执行综合性的程序产生的。因此，应该在一定范围内或按调和分布来描述性能。

下面我们介绍一下能反映计算机性能的基本因素。这些性能指标决不是所有应用中一成不变的。但是，它们对系统结构设计师设计较好的机器、对培养程序员或对编译器设计者优化代码以提高硬件的执行效率都有指导意义。

我们来考虑在一给定计算机上执行一给定的程序。程序性能最简单是用解题时间来量度，它包括磁盘和存储器访问、输入和输出操作、编译时间、操作系统开销和 CPU 时间等。为了缩短解题时间，必须减少所有这些时间因素。

在多道程序计算机中，一个给定程序的 I/O 和系统开销可与其它程序所需要的 CPU 时间重叠起来。因此，在这种情况下，去比较执行程序所需的总 CPU 时间更为合理。CPU 是用来同时执行系统程序和用户程序的，用户最关心的事情就是用户 CPU 时间。

时钟频率和 CPI 目前，数字计算机的 CPU（或简称处理机）是由一个恒定周期（τ，以 ns 表示）的时钟驱动的。周期的倒数是时钟频率（$f=1/\tau$，以 MHz 表示）。程序的规模是由其指令数（I_c），也就是程序串要执行的机器指令数来决定的。执行不同的机器指令所需要的时钟周期数也是不一样的。因此，一条指令的周期数（CPI）就成为衡量执行每条指令所需时间的重要参数。

对给定的指令系统来说，如果我们知道指令在程序中出现的频率，那么就可计算所有指令类型的平均CPI。精确估算平均CPI需要对大量的程序代码在长时期内跟踪测试。如果并不专门针对某种指令类型，这时我们可简单地用CPI来表示给定的指令系统和综合程序的平均值。

性能因子 设 I_c 为已知程序的指令条数或指令的计数。执行程序所需的CPU时间（T，以秒/程序 表示）可用三个主要因素的乘积来计算：

$$T = I_c \times \mathrm{CPI} \times \tau \tag{1.1}$$

执行一条指令需要经历取指令、译码、取操作数、执行和存储结果等过程。在这过程中，只有指令译码和执行两步在CPU中进行，其余三个操作可能需要访问存储器。我们将完成一次存储器访问所需要的时间定义为存储周期。通常，存储周期比处理机周期 τ 大 k 倍。k 值与存储器技术以及所用的处理机-存储器互连方法有关。

一种指令类型的CPI可分为完成指令所需的处理机周期数和存储器周期数两部分。完整的指令执行过程可能包含一至四次存储器访问（一次用于取指令，两次用于取操作数，一次用于存储结果），这与指令的类型有关。这样，我们即可将式(1.1)重写成如下形式：

$$T = I_c \times (p + m \times k) \times \tau \tag{1.2}$$

式中 p 为指令译码和执行所需的处理机周期数，m 为所需的存储器访问次数，k 为存储周期与处理机周期之比，I_c 为指令条数，τ 为处理机周期。如果考虑整个指令系统对CPI各分量（p，m，k）加权，式(1.2)还可进一步细化。

系统属性 上面五个性能因子（I_c，p，m，k，τ）与四种系统属性有关：指令系统结构、编译技术、CPU实现和控制技术、高速缓存与存储器层次结构，见表1.2所列。

表1.2 性能因子与系统属性的关系

系统属性	性能因子				
	指令条数 I_c	平均周期/指令CPI			处理机周期 τ
		处理机周期数/指令 p	存储器访问次数/指令 m	存储器访问时延 k	
指令系统结构	X	X			
编译技术	X	X	X		
处理机实现和控制技术		X			X
高速缓存和存储器层次结构				X	X

指令系统结构对指令条数（I_c）和所需的处理机周期数（p）都有影响，编译技术则影响 I_c、p 值和存储器访问次数（m）。CPU实现和控制技术决定总的处理机所需时间（$p \cdot \tau$），而存储器技术和层次结构的设计则影响存储器访问时延（$k \cdot \tau$）。上述CPU时间可作为估算处理机执行速率的基础。

MIPS 速率 设 C 为执行已知程序所需的时钟周期总数。于是式(1.2)中的 CPU 时间可由 $T = C \times \tau = C/f$ 来计算。此外，CPI$=C/I_c$，则 $T = I_c \times \text{CPI} \times \tau = I_c \times \text{CPI}/f$。处理机的速率通常用 10^6 指令/秒 (MIPS)来表示。我们将它简称为某处理机的 MIPS 速率。应当指出，MIPS 速率与许多因素有关，其中包括时钟速率(f)、指令条数(I_c)和给定机器的 CPI，如下式所示：

$$\text{MIPS 速率} = \frac{I_c}{T \times 10^6} = \frac{f}{\text{CPI} \times 10^6} = \frac{f \times I_c}{C \times 10^6} \tag{1.3}$$

根据式(1.3)，式(1.2)中 CPU 时间也可写成 $T = I_c \times 10^{-6}/\text{MIPS}$。在表 1.2 所列系统属性和上述所得各式的基础上，我们可以得出结论，一台计算机的 MIPS 速率与时钟速率成正比，与 CPI 成反比。所有四种系统属性：指令系统、编译器、处理机和存储技术对 MIPS 速率都有影响，MIPS 还随程序变化而变化。

吞吐率 另一个重要概念是系统在单位时间内能执行多少个程序，这称为系统的吞吐率 W_s(单位为程序数/秒)。在多道程序系统中，系统吞吐率常低于 CPU 吞吐率 W_p。W_p 可用下式表示：

$$W_p = \frac{f}{I_c \times \text{CPI}} \tag{1.4}$$

注意，从式(1.3)可得 $W_p = (\text{MIPS}) \times 10^6/I_c$。$W_p$ 的单位是程序数/秒。CPU 吞吐率是根据 MIPS 速率和程序的平均长度(I_c)来衡量机器每秒钟能执行多少个程序的尺度。为什么 $W_s < W_p$，其原因是由于用多道程序或分时操作在 CPU 上交叉执行多个程序时，I/O、编译器和操作系统产生的额外系统开销所造成的。如果 CPU 能以完善的程序交叉方式保持无空闲状态，则 $W_s = W_p$。这大概是从来不会发生的，因为系统开销常常产生额外的延迟，而 CPU 会在某些周期是空闲的。

例 1.1 MIPS 和性能测量

假定用 VAX/780 和以 IBM RS/6000 为基础的工作站执行一假设基准程序。机器的特性和性能如下所列：

机器	时钟	性能	CPU 时间
VAX11/780	5MHz	1MIPS	12x 秒(s)
IBM RS/6000	25MHz	18MIPS	x 秒(s)

这些数据表明，在 VAX11/780 上测得的 CPU 时间比在 RS/6000 上测得的长 12 倍。这是因为机器不同和所用的编译器不同、在两种机器上运行的目标代码长度不同的缘故。所有其它的时间开销均忽略不计。

由式(1.3)可知，在 RS/6000 上运行的目标代码比在 VAX 机器上运行的代码长 1.5 倍。此外，执行同一基准程序时，VAX/780 的平均 CPI 为 5，而 RS/6000 的平均 CPI 为 1.39。

VAX11/780 具有典型的 CISC(复杂指令系统计算机)结构，而 IBM 机器则为典型的 RISC(精简指令系统计算机)结构，这些将在第四章中叙述。这个例子只在单一程序运行基础上对两种机器的性能简单地作了比较。当不同的程序运行时，结论可能就不一样。

除非我们知道程序长度和每段代码的平均CPI，否则是无法计算CPU吞吐率W_p的。系统吞吐率W_s应该通过长期监察大量的程序才能测得，它们不应该只根据一个或少量程序的运行就得出有关机器性能的全盘结论。

程序设计环境 计算机的可编程性与给用户提供的程序设计环境有关。目前，大多数计算机环境对用户并不很友好。实际上，任何新计算机系统的可销售性取决于能否创造一种对用户友好的环境，使程序设计成为愉快的任务，而不是头痛的事。下面我们简要地介绍一下现代计算机所要求的环境性能。

传统的单处理机计算机是在顺序环境下编制程序的，指令一条接一条地依次执行。实际上，早期的UNIX/OS内核设计成能响应某一时刻从用户进程来的系统调用，接连的系统调用必须顺序通过内核程序。

大多数现有的编译器都能产生顺序目标代码在顺序计算机上运行。换句话说，也就是传统计算机可以用所有为单处理机计算机开发的语言、编译器和操作系统在顺序程序设计环境中运行。

在应用并行计算机时，人们都希望有一个自动发掘并行性的并行环境。为了在不同粒度级别上用智能更强的编译器来描述并行性或便于检测并行性，就必须扩展语言或开发新的语言结构。

除并行语言和编译器之外，操作系统也必须扩展成能支持并行操作以及能管理可并行工作的资源。其中重要的问题有：并发事件的并行调度、共享存储器分配、共享外围设备和通信链路等。

隐式并行性 隐式方法常用传统的C、Fortran、Lisp或Pascal等语言来编写源程序。顺序编码的源程序可用并行化编译器译成并行目标码。如图1.5a所示。此编译器必须能检测并行性，并能分配机器资源。这种编译方法已经在共享存储型多处理机中得到应用。

由于并行性是隐式的，成功与否很大程度上依靠并行化编译器的"智能"，这种方法对程序员来说不须花费很大力气。伊利诺依大学的David Kuck和Rice大学的Ken Kennedy以及他们的合作者都已采用这种隐式并行性方法。

显式并行性 第二种方法(图1.5b)需要程序员付出较大努力才能开发出用C、Fortran、Lisp或Pascal等并行方言所写成的源程序。并行性在用户程序中以显式说明，这将大大减轻编译器检测并行性的负担，而编译器只需保存并行性，并将资源分配给目标机器。加州理工学院的Charles Seitz和麻省理工学院的William Dally在开发多计算机时采用了这种显式方法。

要使一个环境对用户更加友好，必须要有专用软件工具。有些工具是传统高级语言的并行扩展，另一些则是集成环境，其中包括提供不同级别的程序抽象、验证、测试、查错和调试等各种工具；性能预测和监控；辅助程序开发的可视化支持、性能测量以及计算结果的图形显示及动画表示。

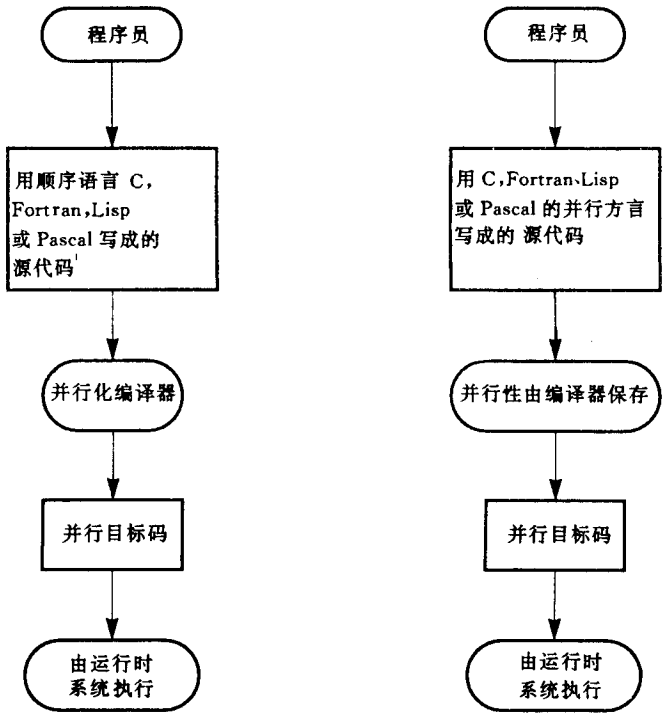

图 1.5　两种并行程序设计方法(承 Charles Seitz 许可,翻印自 Suaya 和 Birtwistle 编《VLSI and Parallel Computation》中的"Concurrent Architectures", P. 51 和 P. 53, Morgan Kaufmann 出版,1990)

1.2　多处理机和多计算机

下面我们从系统结构角度介绍两类并行计算机。这些物理模型可以用共享公用存储器、还是用非共享分布存储器加以区别。1.2 和 1.3 两节我们只介绍结构组织模型，并行计算机的理论模型和复杂性模型在 1.4 节中再进行介绍。

1.2.1　共享存储型多处理机

下面我们将介绍三种共享存储型多处理机模型:均匀存储器存取(Uniform-Memory-Access,简称 UMA)模型、非均匀存储器存取(Nonuniform-Memory-Access,简称 NUMA)模型和只用高速缓存的存储器结构(Cache-Only Memory Architecture,简称 COMA)模型,这些模型的区别在于存储器和外围资源如何共享或分布。

UMA 模型　在 UMA 多处理机模型(图 1.6)中,物理存储器被所有处理机均匀共享。所有处理机对所有存储字具有相同的存取时间,这就是为什么称它为均匀存储器存取

的原因。每台处理机可以有私用高速缓存，外围设备也以一定形式共享。

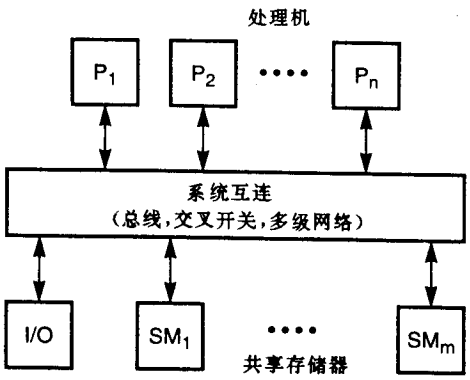

图 1.6　UMA 多处理机模型（例如 Sequent Symmetry S-81）

多处理机由于高度资源共享而被称为紧耦合系统（tightly coupled system）。系统的互连采用总线、交叉开关、或多级网络形式，这些都将在第七章中讨论。

大多数计算机制造厂都有它们单处理机（UP）生产线的多处理机（MP）扩展部分。UMA 模型比较适合于描述多用户的一般应用和分时应用，它可在限时应用中用来加快单个大程序的执行。为了协调并行事件，各处理机之间的同步和通信可通过公用存储器的共享变量来实现。

当所有处理机都能同样访问所有外围设备时，系统被称为对称（symmetric）多处理机。在这种情况下，所有处理机都能同样地运行执行程序，如操作系统内核程序和 I/O 服务程序等。

在不对称（asymmetric）多处理机中，只有一台或一组处理机是可执行的，执行处理机或主处理机（MP）能执行操作系统并操纵 I/O。其余的处理机没有 I/O 能力，因而被称为附属处理机（AP）。附属处理机在主处理机的监督下执行用户代码。在 MP 和 AP 的配置情况下，主处理机和附属处理机之间的共享存储器还照常存在。

例 1.2　多处理机的近似性能

这个例子可使读者看到在共享存储型多处理机系统上并行程序的执行情况。假设下面的 Fortran 程序可在单处理机上顺序执行，并假定所有数组 A(I)，B(I)，C(I)都有 N 个元素。

```
L1:      Do 10 I = 1,N
L2:      A(I) = B(I) + C(I)
L3: 10   Continue
L4:      SUM = 0
L5:      Do 20 J = 1,N
L6:      SUM = SUM + A(J)
L7: 20   Continue
```

假设执行代码行L2,L4和L6，每行要用一个机器周期。为简化分析，执行程序控制语句L1,L3,L5和L7所需的时间可以忽略。假定经过共享存储器的处理机之间的每次通信操作需要k个周期。

开始，假定所有数组已经装入主存储器，并且短程序段已经装入高速缓冲存储器。换句话说，取指令和加载数据的开销可以忽略不计。同时，我们还忽略总线争用或存储器存取冲突问题。这样我们就能集中分析CPU的要求了。

在以上的假定条件下，可以用2N个周期在顺序机器上执行上述程序。在I循环中执行N次独立迭代需要N个周期，同样对于含有N次递归迭代的J循环也需要N个周期。

为了在M-处理机系统上执行程序，我们将循环操作划分成M段，每段有L=N/M个元素。下面的并行程序中，Doall表示所有M段在M台处理机上并行执行：

```
      Doall    k = 1,M
               Do 10 I = L(k-1)+1, kL
                    A(I) = B(I) + C(I)
10             Continue
               SUM(k) = 0
               Do 20 J = 1,L
                    SUM(k) = SUM(k) + A(L(k-1)+J)
20             Continue
      Endall
```

对于M-路并行执行来说，分段的I循环可以在L个周期中完成。分段的J循环在L个周期中产生M个部分和。这样，产生所有的M个部分和共需2L个周期。之后我们还是要将这些M个部分和合并才能得到N元素的最终和。

每一对部分和的相加需要通过共享存储器，用k个周期。为了合并所有的部分和，可以设计一个l层二进制加法树，其中$l=\log_2 M$。加法树用$l(k+1)$个周期从树叶到树根顺序合并M个部分和。这样，多处理机需要$2L+l(k+1)=2N/M+(k+1)\log_2 M$个周期才能得到最终和。

假定数组中有$N=2^{20}$元素，顺序执行原先的程序需用$2N=2^{21}$个机器周期。假设每次机间通信IPC的同步开销的平均值$k=200$个周期，则在$M=256$台处理机上的并行执行需要$2^{13}+1608=9800$个机器周期。

比较一下上述计时结果可以看到，多处理机给出的加速因子最大值为256，而实际值为214。因此，所得到的效率为214/256=83.6%。加速比和效率问题我们将在第三章中进行讨论。

以上结果是在有利的开销假设条件下得到的。实际上，在考虑所有的软件开销和潜在的资源冲突后，最后得到的加速比可能较低。但不管怎样，如果处理机之间的通信开销能降到足够低的水平，则此例已能展示出并行处理的良好前景。

NUMA模型 NUMA多处理机是一种共享存储器系统，其访问时间随存储字的位置不同而变化。图1.7给出了两种NUMA机模型，其共享存储器物理上是分布在所有