科学通报
第41卷 增刊 CHINESE SCIENCE BULLETIN 1996年5月

高性能计算机与计算科学*

唐志敏
（中国科学院计算技术研究所，北京 100080）

关键词 高性能计算机 计算科学 大规模并行处理 超级计算 计算模型

1 引言

在本世纪40年代末50年代初，电子数字计算机刚刚问世的时候，科学和工程计算问题是它唯一的应用领域．50年后的今天，计算机无论在性能还是应用方面都有了突飞猛进的发展，但科学与工程计算仍然是它的重要应用领域之一，尤其是高性能计算机的主要应用领域．

70年代中期以后，出现了以Cray计算机系列为代表的超级计算机．随着可解问题规模的增大，计算本身成了科学研究工作中不可分割的一个重要部分，从而形成了独立于理论科学和实验科学之外的第三类科学，即计算科学（computational science）．计算科学与理论和实验科学的相互作用是很强的：理论科学提出数据模型供计算科学用数值方法解决，而这种数值解又可能导致建立新的理论；实验科学则为计算科学提供数据，而计算科学又可模拟那些难以在实验室中实施的过程[1]．对于给定的问题，计算科学的责任是通过采用先进的数值建模技术、硬件工程和软件机制，提供最好的程序代码，它们能以最小的用户（即理论科学家和实验科学家）努力产生精确的结果．

不难看出，计算科学的内容包括了数值计算、高性能计算机体系结构、软件技术等多个方面，而这几个方面也正好是高性能计算机系统的研究中涉及的最主要的内容．虽然各自的侧重点有所不同，但其中无疑有许多共同或相通的地方．因此，将这两方面的研究放在一起讨论，有助于双方的交融，有利于高性能计算机系统更好地服务于超级计算科学．

2 高性能计算机的现状及存在的问题

鉴于单处理机系统在性能提高上的局限性，所有现代高性能计算机都或多或少地采用了各种形式的并行处理机制，所以现代的高性能计算机都是并行计算机[2]．

如果说并行处理是实现高性能计算的必由之路，那么大规模并行处理则是实现高性能计算的主要途径．处理单元越多，计算能力越强，这在直观上是很明显的．同时，因为可以使用大量的处理单元，所以每个处理单元的设计不必追求最能提高速度的方案，其实现也不一

* 中国科学院科技政策局资助项目

定要使用速度最快的工艺. 这样，合理设计的大规模并行处理系统（以下简称 MPP 系统或 MPP 计算机）不仅可以有极高的运算速度，而且具有很高的性能价格比.

但实际上，不仅 MPP 计算机的设计没有想象的那么容易，甚至 MPP 的思想也是经历了许多年才逐渐为大多数研究人员所接受. 这里当然有器件和工艺技术方面的因素，但本质的原因是对应用问题中是否存在可利用的大规模并行性的怀疑. 著名的 Amdahl 定律为这类怀疑提供了理论依据.

Amdahl 定律认为，并行处理的加速比受限于串行部分（即不可并行部分）在解题算法中所占比例的倒数. 这自然是对的. 但问题在于，并行处理的终极目标是解大型或超大型问题，即超出已有计算机能力许多倍的大问题. 而许多问题都有这样的特性，即串行部分所占的比例基本固定，从而随着问题规模的增大，串行部分所占的比例越来越小，可实现的加速比不断增加.

一大类迫切需要超高性能计算机的应用问题，即所谓的大挑战（grand challenge）问题，就有上述的特征. 最近的应用实例表明，具有大规模并行性的问题不仅仅限于科学和工程应用领域. 事实上，许多事务和数据处理工作中也包含了大规模并行的因素. 大规模数据库中资料的查询就是一个典型的例子.

尽管多机并行是提高运算速度的关键，但并不是处理机越多性能就越好. 并行处理有多种形式，判断哪种形式更好，需考虑多方面的因素. 除了通用性、价格、使用方便性等因素外，另一个必须重视的因素就是可伸缩性.

所谓的可伸缩性（scalability），实际上包括如下几个方面的内容[3]：

（1）规模可伸缩性：当处理机数量增加时，系统性能可线性增长；

（2）技术可伸缩性：当结点处理机升级换代时，系统性能可相应增长；

（3）问题可伸缩性：对给定的并行系统，当问题规模变化时，运行效率变化不大，而当问题规模与系统规模线性增长时，总运行时间保持不变；

（4）算法可伸缩性：当处理机数量增加时，原来的并行算法仍然具有很好的计算效率.

其中，规模可伸缩性对于通过并行处理实现超高性能计算机是至关重要的：只有在规模可伸缩性较好的系统中，增加处理机才可能获得性能上的改善. 因此，MPP 计算机必然具有很好的规模可伸缩性. 体现在互连网络上，主要是采用伸缩性较好的点对点网络，如超立方体、网格和胖树（fat-tree）等.

当然，对规模可伸缩性的理解也不是绝对的. MPP 计算机应该有较高的性能价格比和极高的系统性能，但考虑到用户的接受能力，其实际价格不能过高. 从目前的技术水平和用户需求来看，允许数千个结点的 MIMD 系统就是伸缩性较好的系统，而含上万个结点的系统则由于价格过于昂贵，实际上是不可实现的.

鉴于集中式的共享存储器系统在总存储带宽上的限制和实现上的复杂度，高度并行的计算机系统总是将物理存储器分散在各处理单元附近，以便为每个处理单元提供足够的存储器带宽. 从而所有的 MIMD 型 MPP 系统在总体结构上都是相似的，即由处理单元结点集和互连网络组成. 每个处理结点可以是一个由 CPU、cache、局部存储器和互连网络接口组成的单处理机（当然结点处理单元还可以有 I/O 接口）. 不同系统的差别主要体现在网络接口

中. 例如:

(a) 早期的消息传递型多计算机系统采用存储转发的通信方式, 互连网络采用超立方体(hypercube) 结构;

(b) 新一代的消息传递型系统在二或三维网格 (mesh) 上使用蠕虫 (wormhole) 机制进行通信;

(c) 分布式共享存储器系统中, 网络接口实现对远程数据的直接访问并维持数据的正确性;

(d) 在基于高性能工作站的网络计算环境中, 网络接口实际上就是网卡; 等等.

分散存储带来的主要问题是任一处理单元都不能直接访问非本地存储器. 目前的大部分系统都通过硬件提供的消息传递机制和在软件或编程模型上显式采用通信原语来帮助解决这个问题. 消息传递模型与传统单处理机系统和共享存储器多处理机系统的统一地址空间概念是不一致的, 从而导致串行程序的自动并行化工作在消息传递系统上难以进行, 也大大增加了为使系统发挥高性能而必须进行的算法改造工作的难度.

为了避免消息传递模型带来的低效率和不方便, 并增强系统的通用性, 有必要提供逻辑上统一的地址空间. 因此, 出现了共享虚存 (shared virtual memory) 和分布式共享存储器(distributed shared memory) 的概念, 即在具有分布式物理存储器的系统中, 通过统一的逻辑或物理地址空间来实现存储器共享.

完全由软件实现的共享虚存适合在现有的消息传递型系统和基于高性能工作站的局域网上实现. 此时, 每个处理单元的局部存储器被看作是整个虚拟地址空间的一部分, 地址转换类似于典型的虚拟存储系统. 只是缺页中断的处理过程有所不同: 新页面不但可能来自外存, 而且更有可能来自其他处理单元的局部存储器. 在后者的情况下, 页面的调入是通过消息传递来完成的. 在出现同一数据的多个拷贝时, 借助类似于保持多 cache 内数据一致性的目录机制来保证数据的正确性. 这种方法对用户 (程序员) 完全透明, 且实现简单, 但实际效率很低. 把整个页面作为同步互斥和数据一致性保护的基本单位会大大增加通信量, 由操作系统处理缺页中断并发出通信原语也是一个漫长的过程.

与此相反, 硬件实现的分布式共享存储器系统中, 对非本地数据的访问是通过将远程数据由硬件自动取入本地 cache 而实现的. 因为数据传输、共享和同步、以及数据一致性操作的单位都缩小到了 cache 块, 这种方式具有较共享虚存方式高得多的系统效率.

硬件实现的分布式共享存储又有两种形式, 即部分 cache 方式和全 cache 方式. 在部分 cache 的情况下, 处理单元既有为维持其高速运行所必需的高速 cache 和局部存储器, 又有专为访问远程数据而设置的大容量 cache 和有关的一致性控制部件. 在处理机访存时, 一旦发现物理地址超出局存空间且远程访问 cache 也不命中, 就立即启动远程访问逻辑, 将远地的正确数据取入远程访问 cache, 并执行有关的数据一致性操作. 这样, 只有当所有处理单元的局存中都没有所需的数据时才会引发缺页中断, 用磁盘页面替换内存页面, 同步和一致性操作的单位也缩小到了一个 cache 存取块.

在全 cache 的系统内, 所有处理单元的局部存储器构成共享存储器, 而每个处理单元所对应的局存则充当该共享内存的一个 cache. 与部分 cache 的系统相比, 全 cache 的系统中处

理单元有很大的 cache，从而有利于开拓程序中的局部性因素，提高系统性能．同时，由于处理单元可访问的所有存储器都被当作 cache 使用，传统意义下的局存或主存也就不再存在，这在逻辑上消除了对物理存储器的寻址．当然全 cache 系统的硬件复杂度也是很大的．目前这种类型的系统有 Kendall Square Research 公司的 KSR1 和 KSR2．

在可伸缩性方面，共享存储器 MPP 系统需考虑为保证数据一致性而采用的目录机制的规模和一致性机制的效率．目录中必须记下共享地址空间内每一个存储页面或 cache 存取块与每一个处理单元的关系，从而目录存储器的大小将正比于处理机数与数据存储器容量之乘积．在处理单元很多时，这种需求是难以承受的．实际上，共享同一数据项的处理单元数并不太多，即登记处理单元与数据块关系的关联矩阵是稀疏的，从而可以大大压缩所需的存储空间．例如对每个存储页面或 cache 存取块，在目录项中只登记若干个处理单元号，当实际共享它的处理单元多于这个数目时，就采用对应于每个处理单元的位向量（bit vector）方式．IEEE 关于可伸缩的一致性接口（SCI，scalable coherence interface）标准也很有特色：不同 cache 中的同一个行被串在一个双向链表中，而对应于此 line 的目录项则维持一个指向该链表头部的指针．

并行计算机的应用程序开发主要有两种形式，即串行程序的自动并行化和基于并行语言的并行程序设计．自动并行化的目标在于开发循环迭代间的并行性因素，目前虽已取得了一些进展，但效果仍不理想，且主要针对 FORTRAN 进行，适合于共享存储器的多处理机．基于并行语言的并行程序设计有多种方式，如共享变量、消息传递、数据并行、面向对象、函数式、数据流式等，但真正适合 MPP 计算机的主要是数据并行方式．

数据并行指的是将程序所要处理的数据域分割成许多小区域，从而可用区域的划分来代替计算的划分．这样，只要给每个处理机分配一个子区域，整个计算就能完全并行地进行．适合数据并行的应用都具有这样的特点，即对数据域中的每个元素都实施相同的计算，所以这类应用也适合在 SIMD 系统上实现．

MIMD 系统上的数据并行是通过所谓的 SPMD（single program multiple data）方式实现的：程序员只要准备一个单结点的程序，这个程序实际上将运行在每一个处理结点上．当然，每个结点处理的数据是各不相同的．必要的通信可通过 send/receive 原语实现．为了减轻程序员在数据划分和通信控制方面的难度，出现了以 HPF（high performance FORTRAN）为代表的直接支持数据划分的并行语言．HPF 是 FORTRAN 90 的扩展，它通过 DECOMPOSITION，ALIGN 和 DISTRIBUTE 语句实现数据的自动分配，并生成必要的通信语句．

尽管数据并行方式在一定程度上简化了在消息传递型多计算机上编程的难度，但仍然存在程序移植和并行化方面的困难．因为基于统一地址空间的共享存储器模型往往更适合于人们的思路，随着分布式共享存储器多处理机的进一步发展，基于共享变量的程序设计将逐步主导 MPP 系统的应用软件开发．虽然全新的非过程型语言在并行性的描述和开发方面有着 FORTRAN 和 C 无法比拟的优点，实现效率和已有软件资源的负担决定了至少下世纪初的并行程序设计语言仍将是基于 FORTRAN 和 C 的．

MPP 计算机的程序开发环境特别需要提供对程序调试、性能调整（performance tuning）

和软件移植的支持. 为了了解程序中的故障或影响性能的因素, 需要一系列的分析工具, 包括可视化工具.

目前 MPP 计算机的系统软件实际上从事着与单处理机系统软件类似的管理工作, 而对大规模并行处理带来的新问题 (如前述的局部性、通信效率和负载平衡等问题) 几乎无能为力. 纯粹依赖精心的算法设计或程序设计来提高 MPP 系统的性能并不是长久之计. 因此, MPP 系统本身的并行化算法 (即管理 MPP 计算机的算法) 的研究将成为今后 20 年的一项重要工作. 这类算法直接关系到并行处理系统的工作效率, 它们自身虽然不一定并行或分布地执行, 但它们的执行结果却指示了一系列需并行执行或可并行执行的任务或动作.

3 计算科学对高性能计算环境的要求

在超级计算方面获得的已有经验主要来自向量处理机和向量多处理机. 现在已经发现, 在向量超级计算机中可以忽略的许多问题, 如问题分割、负载平衡、通信效率等, 对 MPP 计算机的性能起着关键的作用. 因为这些问题及其作用形式随应用领域、体系结构和系统软件的不同而有较大的差异, 很难找到系统性的方法来消除它们对性能的影响. 因此, 一些关键理论和技术的进一步研究和发展已成为 MPP 系统能否进一步发展的关键.

尽管 MPP 系统的峰值速度比向量多处理机高出一个数量级. 但是, 向量超级计算机经过 10 多年的发展, 已积累了大量高效率的应用软件, 其系统软件, 尤其是向量化编译程序, 也已基本成熟. 因此, 许多实际计算问题在这类系统上可获得 60% 以上的系统效率. 相比之下, 许多用户反映, 开发 MPP 系统的性能是他们最感困难的事情. MPP 系统的效率很难发挥出来, 除非对解题算法作彻底的改进, 并且对某些关键程序段作人工优化处理 (如采用汇编代码), 而这样的改进和优化工作只有少数计算数学或超级计算机专家才能完成. 所以, 从系统所能达到的持续性能出发, 并综合考虑花费在算法改进和程序编制方面的人力和时间因素, MPP 的性能价格比优势有时并不明显.

造成 MPP 系统的峰值速度与实际速度相差很大的因素很多, 如局部存储器带宽不够, 通信速度与计算速度不匹配, I/O 成为影响性能的瓶颈, 系统软件效率不高, 等等. 当前的 MPP 系统都是基于高性能微处理器的, 而大多数高性能微处理器即使在片内 cache 完全命中的情况下每个时钟周期也只能进行一次存储器访问, 这个数字与它们每个周期能执行多条指令或完成多次浮点运算的峰值速度是不相称的. 通信速度与计算速度的不匹配, 主要原因不在于传输得不快, 而在于通信本身的开销大.

对于并行处理系统, 下述几个部分是决定整个系统性能的关键: 高速的单元处理机、存储系统、通信机制、同步机制和 I/O 系统. 为了提高 MPP 系统的实际速度, 一方面要改进这几个部分的结构设计; 另一方面必须改善系统软件的效率, 加强静态和动态优化的能力. 例如, 设法提高程序的访存局部性, 有效地利用预取和后存技术, 对提高 MPP 系统的性能极有帮助. 所有这些都需要体系结构、系统软件和算法设计等 3 个方面的紧密配合.

理想 MPP 计算机系统在各方面的性能都应该是匹配的和均衡的. 这些性能包括处理单元的标量和向量计算速度、处理单元存储器容量和带宽、处理单元的通信速度和 I/O 带宽、整个互连网络的通信带宽和网络中半数结点同另一半进行通信的最大带宽, 以及整个系统的

计算能力和 I/O 能力. 其中, 由于技术方面的原因, I/O 带宽与运算速度的匹配是最难实现的.

MPP 计算机都有大规模并行的计算单元, 但大都缺乏大规模并行的输入输出单元. 从而输入输出成为系统最大的瓶颈. 早期的消息传递型系统把 I/O 工作都交给系统主机 (host) 去完成, 造成了严重的瓶颈现象, 也增加了通信网络的开销. 给部分或所有处理单元提供 I/O 能力和设备可以大大缓解瓶颈问题, 但由此带来的资源管理工作 (如并行文件系统等) 仍是一个没有得到很好解决的问题.

4 超级计算理论

并行计算模型是 MPP 体系结构和 MPP 算法之间的接口界面. 在这一界面的约定下, 并行系统的设计者可以设计对并行性的支持机构, 以提高系统的性能; 算法设计者可以发展高效率的计算方法以充分利用并行系统的计算能力. 一个成功的并行模型不宜对硬件和软件结构的细节作过多的限制, 从而保证它在相当范围上的通用性, 便于算法和程序的移植; 但它又要能很好地反映出不同结构的主要特征, 从而为结构和算法的设计提供启发和依据.

程序设计模型在很大程度上体现了计算模型. 例如, 顺序计算已有很成功的计算模型, 即 RAM, 它与传统计算机的 von Neumann 结构是直接对应的. RAM 的并行版本是 PRAM, 它实际上对应着共享存储的编程模型. 但由于 PRAM 模型对存储器的访问时间、存储体的分布和存储体的访问冲突未作任何限制, 并不能直接指导基于共享存储模型的并行程序设计. 与 Hoare 的 CSP (通信的顺序进程) 模型对应的是消息传递型程序设计方式 (数据并行是其特例). 尽管数据并行是一种有效的方式, 但它毕竟只适合有限形式的数据结构, 且不能用于功能 (任务) 级的并行. 一般的消息传递方式原则上是可用的, 但实际使用时遭遇到了编程的巨大问题. 另外, CSP 模型本身不提供任何解决诸如通信开销、网络延迟等问题的手段.

Culler 等提出的 LogP 模型对并行系统中的开销给予了较多的重视, 从而可作为 CSP 的一个重要补充. 但 MPP 系统的程序设计方便性 (programmability) 仍未得到很好的解决. 数据流模型便于程序设计, 且具有彻底的并行性, 但具体实现和效率上的困难经过近 30 年的努力尚未克服. Dennis 最近提出了一种具有数据流思想的通用语义模型, 希望它能有助于全新的 MPP 系统的设计.

研究应用领域并行计算方法的主要目的是开发应用问题的计算中潜在的并行性, 从而使这种计算便于高效地在并行计算机系统上实施. 高效并行算法的研究不但要求对应用背景有充分的认识, 而且要求对计算机系统的结构有足够的了解. 缺少通用、有力的并行计算模型导致了并行算法设计上的盲目性, 许多算法的设计往往因“机”制宜, 造成大量的重复投资.

并行算法研究的目标是把计算的时间复杂性尽量转化为空间复杂性, 基本做法是加宽算法树的宽度, 压低算法树的高度. 合理的排序与分解 (如区域分解、算子分解、数据分级等) 有助于负载平衡和减少通讯量, 这是大规模并行计算提高效率的两个重要手段. 目前通行的将串行算法 (程序) 人工或自动地改为并行算法 (程序) 的做法有很大的局限性. 从并

行处理的角度重新考察应用问题的数学或物理模型才是根本的出路所在.

值得注意的是，高并行度与高效率不是同一个概念，在并行算法研究中不能单纯追求并行度高. 以已有的并行迭代算法为例，极高并行度的算法一般不是高效率的. 高并行度往往只考虑局部的最优并行化，而高效率并行算法追求的则是整个算法效率的优化，这就必须结合计算问题本身内在的模型背景统一考虑. 因此，计算效率或实际计算时间（而不是处理机效率）应是评价不同算法性能的唯一标准.

大规模并行计算的对象是求解大型计算问题，并行计算与超级计算（supercomputing）密不可分. 大量实践和分析都表明，很多数学物理问题的潜在并行性体现在当规模增大时，并行加速比呈线性增长. 由于大规模并行计算机目前尚处于专家级应用水平，中、小型计算问题不必使用并行计算机.

可伸缩性是研制并行算法的另一个重要指标. 只有可伸缩的算法才能发挥可伸缩大规模并行计算机的高效率. 适当的分块技术是实现可伸缩算法的重要手段，也是提高带有 cache 的高性能计算机的性能所必需的.

5 发展方向和对策

传统单处理机迅猛发展的一个重要原因是其编译技术允许应用程序员用高级语言进行与机器结构无关的程序设计，从而有效地保护了应用软件开发方面的巨大投资. 相比之下，并行计算机（尤其是大规模并行计算机）的编译系统还远没有做到这一点. 每个新的 MPP 机型的问世都面临着改写所有应用程序的艰巨任务. 因此，为了进一步扩大 MPP 计算机的应用范围，提高 MPP 计算机的实际可用性，必须实现与机器无关的 MPP 程序设计环境.

分布式共享存储模型可以方便地移植已有应用程序，便于开展自动并行化工作，因此具有一定的通用性. 但是，它仍不能解决 MPP 系统实际速度远低于峰值速度的通病. 相反，由于目前的系统软件和优化技术在增强局部性、降低通信量等方面缺乏有效的手段，直接基于消息传递方式编程更利于发挥系统性能. 因此，在今后的若干年中，这两类模型都会有一个不断完善的过程，但从长远观点看，只要解决了实际效率问题，共享虚存模型将更具有优势，而消息传递模型则可能成为一种重要的辅助方式.

随着高速信息网络的逐步建立并投入使用，基于高速通信网络的并行计算将成为高性能计算的一个重要方面. 国内外已开始重视这方面的研究工作. 这里的主要问题是使用的方便性和实际效率问题. 但是，另一方面，从我国目前的技术状况，尤其是高性能计算的普及状况来看，网络计算的实用化为时尚早. 由于国内高性能计算机数量太少，真正能接触到并使用高性能计算机的用户很少. 可以说，我国科学计算领域的大部分用户并没有并行计算的经验. 因此，从现在直接跨入网络计算环境是不现实的，诸如算法改造、程序调试、实际效率等方面的问题（它们可能比常规的 MPP 系统复杂得多）很容易使用户丧失信心或兴趣.

因此，我们还是应该主要发展各种形式的共享存储多处理机系统和 MPP 多计算机系统. 在开展并行处理研究的同时，国家应投资建立若干个高性能计算机（或超级计算机）中心，为那些需要高性能计算机的科学家们提供工具，使他们逐步熟悉并接受大规模并行计算的思想和方法，为从通过网络使用高性能计算机过渡到真正的网络计算作好准备.

另外，需要注意加强高性能计算机和计算科学的教育工作[4]．国外自90年代以来已有许多大学增设了计算科学（computational science and engineering，简称CSE）等方面的系科[5]．我们的条件虽然还不够成熟，但在有条件的院校开设计算科学方面的研究生课程还是完全可以的．在大学阶段，应该增设并行算法或并行程序设计方面的课程，并以之逐步替代传统的算法分析和程序设计课程．

参 考 文 献

1 Rodrigue G, Giroux E D, Pratt M. Numerical modeling in large-scale scientific computation. IEEE Computer, 1980, 13(10): 65～69

2 Hwang K. Advanced Computer Architecture: Parallelism, Scalability, Programmability. New York: McGraw-Hill Inc, 1993

3 Bell G. Scalahle, parallel computers: alternatives, issues, and challenges. International Journal of Parallel Programming, 1994, 22(1): 3～46

4 Miller R. The status of parallel processing education. IEEE Computer, 1994, 27(8): 40～43

5 Rice J R. Academic programs in computational science and engineering. IEEE Computational Science and Engineering, 1994, 1(1): 13～21