第 1 章 并行计算方法

1.1 并行计算概述

1.1.1 计算平台的变革

在我国，土木工程领域大规模科学与工程计算越来越重要，如奥运工程设计中火灾模拟技术应用，杭州湾大桥和浦东机场二期航站楼等大跨工程设计中数值风工程技术应用，央视大楼等重大结构设计中抗震性能分析，以及上海磁悬浮工程温度应力分析，等等。由这些工程案例可以看到，计算需处理的信息量越来越庞大，要解决的问题越来越复杂，因而计算量剧增。对于大型工程问题来说，保证数值解的精度非常不容易，即使使用目前速度最快的串行计算机也难以满足要求。这就是说，依靠单机结构几乎不可能大幅度提高计算规模与计算速度以赶上工程计算需求的增长。因此，要实现超大规模计算，国内外普遍认为，必须充分发掘计算并行性 [1-4]。提高计算并行性的主要途径，除充分利用单处理器内部可执行的并行性之外，是充分利用处理机之间的并行处理。这种途径的优点是可扩展性很强，并且不受投资昂贵的微电子工艺的限制。不管是过去还是现在，采用并行和集中式计算技术来构建的高性能计算系统（High-Performance Computing，HPC）的应用在大规模科学计算中都占据主要地位。但在今天，随着互联网络通信速度与带宽的大幅提高，普遍的计算趋势是平衡共享网络资源，采用并行和分布式计算技术构建的高吞吐量计算系统（High-Throughput Computing，HTC）及其支持的云计算将有可能成为大规模工程计算的又一助力 [5]。图 1-1 阐述了 HPC 系统与 HTC 系统的演化。

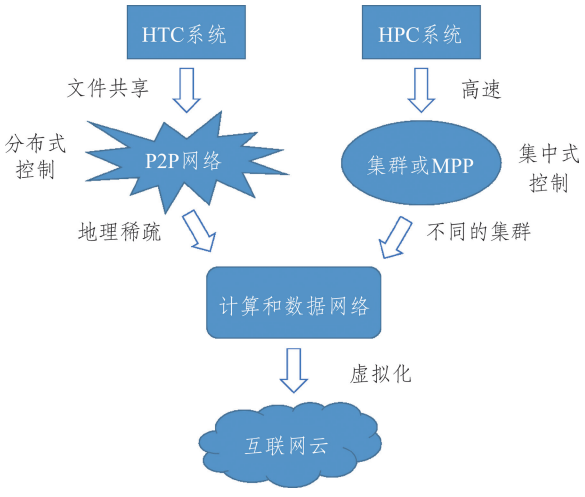

图 1-1 HPC 系统与 HTC 系统的演化 [5]

关于集中式计算、并行计算、分布式计算、云计算的精确定义，一些高科技组织已争论了多年。通常来说，它们互有关联，其简单的基本概念表述如下：

集中式计算：在集中式计算中，所有计算资源都集中在一个物理系统之内。所有资源包括处理器、内存、存储器是全部共享的，并且他们紧耦合在一个集成式的操作系统中。许多数据中心和超级计算机都是集中式系统，但它们都被用于并行计算、分布式计算和云计算中。

分布式计算：一个分布式系统由众多自治的计算机组成，各自拥有其私有内存，通过计算机网络通信。分布式系统中的信息交换通过消息传递的方式完成。运行在分布式系统上的程序称为分布式程序。

并行计算：在并行计算中，所有处理器或是紧耦合于中心共享内存或是松耦合于分布式内存。处理器间的通信通过共享内存或通过消息传递完成。通常称有并行计算能力的计算系统为并行计算机，运行在并行计算机上的可并行运行的程序称为并行程序。

云计算：一个互联网云的资源可以是集中式的也可以是分布式的。云可以在集中的或者分布式的大规模数据中心之上，由物理或虚拟的计算资源构建。云采用分布式计算或者并行计算，或两者兼有。云计算常被认为是一种效用计算或者服务计算形式。

效用计算（Utility Computing），简单地说就是通过互联网资源来实现企业用户的数据处理、存储和应用等问题，企业不必再组建自己的数据中心，改变目前传统数据库软件侧重于离线和后台应用的局面。效用计算的具体目标是结合分散各地的服务器、存储系统以及应用程序来立即提供需求数据的技术，使得用户能够像把灯泡插入灯头一样来使用计算机资源。效用计算理念发展的进一步延伸，使云计算技术正在逐步成为技术发展的主流。

需要指出，虽然在解决问题时都是将大任务化为小任务，但分布式计算和并行算法是不同的。分布式的任务包互相之间有独立性，上一个任务包的结果未返回或者是结果处理错误，对下一个任务包的处理几乎没有什么影响。因此，分布式的实时性要求不高，而且允许存在计算错误。分布式要处理的问题一般是基于“寻找”模式的。而另一方，并行程序并行处理的任务包之间有很大的联系，而且并行计算的每一个任务块都是必要的，没有浪费的分割的，就是每个任务包都要处理，而且计算结果相互影响。这就要求每个计算结果都要绝对正确，而且在时间上要尽量做到同步。由此可见，大规模工程计算使用的是并行计算。

1.1.2 并行计算的基本概念

并行计算的发展得益于人类的两方面认识[6]：一是如上提到的，单机性能不可能满足大规模科学与工程计算的需要，而并行计算机是实现超大规模计算的唯一途径；二是同时性与并发性是客观存在的普遍属性，具有实际物理背景的计算问题在许多情况下都可划分为相互独立，但又彼此存在一定联系的若干个能够并行的子任务，即这些计算问题多具有内在并行性。

如果将进程定义为顺序执行的一组操作或一段程序，那么并行算法则是一些可同时执行的进程的集合，这些进程互相作用、协调工作，从而完成一个问题的求解。那么要用并行机求解问题，必须把这个大问题分解成若干能够并行执行的小问题，对这些小问题的结果进行有效组合可以得到原问题的最后结果。但不幸的是，一个大问题的分解是非常困难的，通常一个问题中存在待求因素互相依赖的情况，这将可能导致小问题之间存在数据相关性。数据相

关性越大，“沟通”与“等待”的时间需求越大，这时需要很好地规划并行方案、设计并行算法以获得较好的效率。

为了更浅显地解释并行算法里的概念，另以两个简单的例子来说明一个大问题的分解和并行执行 [7]。

范例 1：假设一个房屋开发商要建造 6 幢相同的房子，而一个建筑承包商能在 5 个月内造出一幢，并且不能同时建造另一幢。如果房屋开发商只雇用一个建筑承包商，完成 6 幢房子的建造需要 30 个月（如图 1-2）；如果将 6 幢房子分别交给 2 个建筑承包商负责，则完成建造只需要 15 个月（如图 1-3）；而如果将 6 幢房子分别交给 6 个建筑承包商负责，则完成建造只需要 5 个月。由于每幢房子的建造是独立的，因而这是一个工作容易分解的例子，整个问题具有直观的并行性。

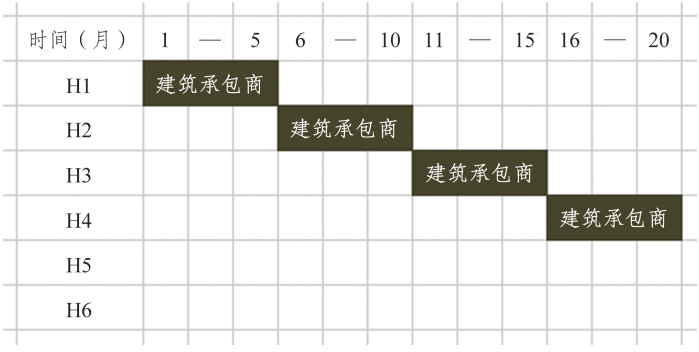

图 1-2　一个建筑承包商建造 6 幢相同的房子

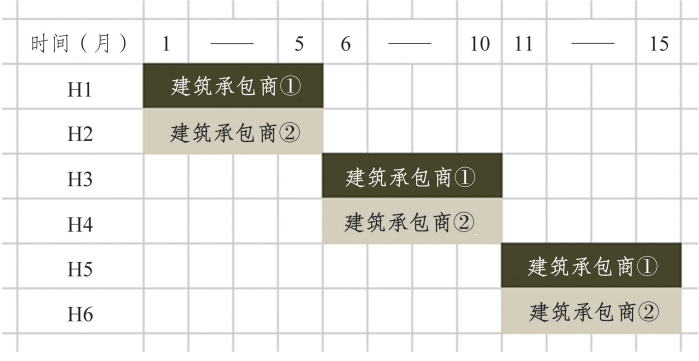

图 1-3　两个建筑承包商建造 6 幢相同的房子

范例 2：假设一个建筑承包商要建造 6 幢相同的房子，建造能力同上例。如将一幢房子的建造工作分为 5 个部分 —— ①地基、②地上结构、③木工部分（如安装门窗等）、④安装电气系统、⑤装饰，显然每部分的工作都是在前一部分的基础上进行的，5 个部分必须顺序执行。在这种情况下，把工作分解且并行执行是困难的，但是经过有效的规划亦可以实现。流水操作就是一个很好的并行规划：假设建筑承包商将施工人员分为 5 组，每组可单独在一个月内完成一部分任务。当第一组完成第一幢房子的第一部分时，第二组可立即开始第一幢房子的第二部分。与此同时，第一组将开始第二幢房子的第一部分工作。整个流水线如图 1-4 所示，第 5 个月结束时，第一幢房子已经完成，而后每个月完成一幢房子，工期从原来的 30 个月缩短到 10 个月。由于各步骤之间存在依赖性，因而这是一个可分解性较差的例子。

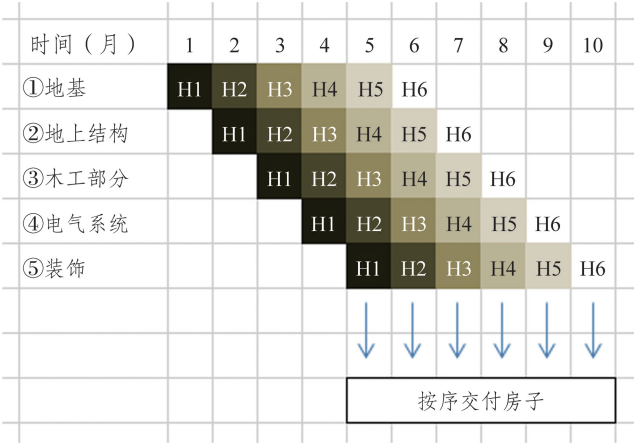

图 1-4 一个建筑承包商建造 6 幢相同的房子的流水线 [6]

1.2 并行计算的性能分析

理想情况下，应如范例 1，在一个处理器上用 T 时间完成求解的问题，在 P 个处理器上应能用 T/P 时间完成。换句话说，P 倍并行性至少是存在的。然而情况并不会如理想情况一般，并行计算通常会引入串行计算中不存在的开销，即使有良好设计的程序，满足 T/P 目标的挑战随着 P 的增加也会变得越加困难。

1.2.1 性能损失的原因

以下是 4 个导致并行性能损失的基本原因，本节将对它们进行简单的介绍 [7]。

（1）顺序计算不需要付出的开销。

（2）不可并行化的计算部分的比例。

（3）对资源的竞争。

（4）闲置的处理器。

1.2.1.1 开 销

并行计算中，建立线程和进程以并行执行，以及撤销线程和进程都存在开销。

除此之外，通常认可的并行开销来源有以下 4 种。

通信：进程之间的通信是开销的主要部分。由于在串行计算中，处理器并不需要与其他处理器进行通信，因此在并行计算中所有通信都是一种开销。

同步：当一个进程必须等待另一个进程中出现的事件时就存在同步开销。

计算：并行计算几乎总是要完成一些额外的计算，这些计算在串行求解时是不需要的。

存储器：并行计算常常导致存储器的开销，而计算的规模受制于存储器的容量。

并行的开销通常影响了处理器数 P 无限制增长所带来的好处。即使计算理论上可以分配一个处理器专门对一个数据点进行计算，但通常在 $P=n$ 之前也会因为开销过大而被迫放弃。

1.2.1.2　不可并行计算

如果一个计算在本质上是串行的，那么适用再多的处理器也无法改进性能。换句话说，不可并行计算部分的存在，必将限制并行计算的性能。在固定应用规模的前提下，阿姆达尔定律（Amdahl's law）描述了程序执行时间中串行和并行两部分的关系。如果一个计算在一个处理器上串行执行需要花费的时间是 T_s，则当有 p 个处理器时，T_p 可表示为

$$T_p = xT_s + \frac{(1-x)T_s}{p} \tag{1-1}$$

式中：x 为不可并行计算部分的耗时在计算串行总时间中的比例。

1.2.1.3　竞　争

为争夺共享资源而引起的竞争常会降低系统的性能，使得多处理器的计算性能甚至比单处理器的还要差。

1.2.1.4　闲置的处理器

理想情况时，所有处理器在所有时间都忙于工作，但实际上并非如此。一个进程由于缺少工作或因为正在等待某个外部事件的发生，如来自其他进程的数据，将无法继续执行。因此，闲置时间通常是同步和通信造成的。但从程序设计层面来说，闲置时间的一个常见来源是处理器负载的不均匀分布，另一个常见来源是存储器的限制。

1.2.2　相关性与粒度

1.2.2.1　相关性

相关性的概念提供了一种推断低效来源的方法。

相关性是指两个计算间的排序关系。在不同的场合，相关性以不同的形式表现出来。例如两个进程间，当一个进程等待来自另一进程的消息到达时，就出现了相关性。通常，相关性也以读写操作加以定义，对于线程计算来讲就相应于存储器的装载（读）和存储（写）。数据相关性是指对一对存储器操作的顺序，为了保证正确性，必须保持这种顺序关系。

遵从所有相关性的任何执行顺序的排列都将产生最初程序所指定的相同结果。因此，相关性的概念允许我们描述和区别哪些执行顺序必须保证而哪些不必，从而保证程序执行的正确性。另外，它还提供了一种方法用以推断性能损失的可能原因。例如，跨进程的数据相关性就要求两个进程间必须进行同步或通信。

1.2.2.2　粒　度

并行的粒度由线程或者进程间的交互频率所决定，即跨越线程或进程边界的相关性频率。因此，粗粒度是指线程和进程依赖于其他线程或进程的数据或事件的频率是较低的，而细粒度计算则是那些交互频繁的计算。每次交互必将引入通信和同步，因此粒度的概念需要引起重视。

减少相关性的一种方法是增加交互的粒度。最佳的粒度常常依赖于算法的特征和硬件的特征。在极端的情况下，最粗粒度的计算含有巨量的计算而无交互。

1.2.3 性能的评价

1.2.3.1 时间性能

评价并行算法性能的因素主要包括：①计算时间；②处理器个数；③机器模型[8]。加速比和并行效率就是两个常用的并行算法时间性能和处理器利用性能的度量。

加速比：加速比反映的是并行执行后，程序运行时间减少的比率，能最为直观地反映并行带来的好处。实际加速比（Real Speedup）的定义[8]表达为

$$R.S_{\mathrm{p}}(n,p)=\frac{\text{实际使用的串行算法执行时间}}{\text{用 } p \text{ 个处理器解决问题的时间}}=\frac{T_{\mathrm{s}}}{T_{\mathrm{p}}} \tag{1-2}$$

观察 1.1.2 小节范例 1 及范例 2，由上式可以计算得到：范例 1 的实际加速比为 6；范例 2 的实际加速比为 3。式中：T_{s} 又称为串行执行的时间复杂度，显然，它是问题的规模 n 的函数；T_{p} 是并行执行的时间复杂度，可以看到它也是一个问题规模 n 的函数，同时它亦是处理器个数 p 的函数。固定 n，画出以 p 为 x 轴，以实际加速比为 y 轴的实际加速比曲线，可以用于在处理器数目增加时分析算法的性能；假如固定 p，画出以 n 为 x 轴的曲线，则可用于在问题规模增加时分析算法的性能。

对于固定规模问题，则引入阿姆达尔加速比模型[8]：对于一个给定的并行系统和一个固定规模的问题，设该问题的串行部分所占比例为 f，并行部分所占比例为 N，且 $f+N=1$，并行系统的规模即参与并行计算的处理器个数为 p，则固定规模问题的并行加速比定义为

$$S_{\mathrm{p}}=\frac{f+N}{f+N/p}=\frac{1}{f+N/p} \tag{1-3}$$

显然，对同一规模的问题而言，理想情况下（排除并行带来的必要计算时间和通信时间等开销），并行部分所占比例越大，则加速比越大，并行算法的时间性能越好。

效率：效率（efficiency）是和加速比关系密切的时间性能度量，它是加速比和处理器个数 p 的比值[8]，写作：

$$E=\frac{T_{\mathrm{s}}}{pT_{\mathrm{p}}} \tag{1-4}$$

加速比最大不超过处理器个数，因此效率最大为 1。显然，1.1.2 小节中，范例 1 的效率为 1，而范例 2 的效率为 0.6。一般来说，只有很有限的问题的加速比与效率能达到范例 1 的水平，而设计一个并行算法的最终目的则是使算法的加速比与效率尽量向这一水平靠拢。

1.2.3.2 可扩展性能

首先要说明，尽管可扩展性能最理想的情况是使用越多的处理器就越能改善性能，然而这种想法是天真的，因为当增加 p 时，要达到高的并行效率就越加困难。用一个例子来说明这个问题。

范例 3[7]：假设一个计算在进行串行计算时所需的时间为 T_{s}，且全部可以并行。同时假设并行时存在 $0.2T_{\mathrm{s}}$ 的开销，并且乐观地假设开销的总量不随处理器的增加而增加。因此在两个处理器上的并行求解将需时 T_2：

$$T_2 = \frac{T_s}{2} + 0.2T_s = \frac{7}{10}T_s$$

使用两个处理器的效率为：

$$E_2 = \frac{T_s}{2T_2} + 0.71$$

使用 10 个处理器时，执行时间和效率分别为：

$$T_{10} = \frac{T_s}{10} + 0.2T_s = \frac{3}{10}T_s$$

$$E_{10} = \frac{T_s}{10T_{10}} + 0.33$$

对于 100 个处理器则有：

$$T_{100} = \frac{T_s}{100} + 0.2T_s = \frac{21}{100}T_s$$

$$E_{100} = \frac{T_s}{100T_{100}} + 0.047$$

可以看到，处理器由 10 个增大到 100 个，计算用时并没有显著减少，但在 100 个处理器的情况下，每个处理器只有 4.7% 的时间在进行有效的工作。这一极低的效率表明增加更多处理器的好处随处理器数量的增加而减少，处理器数目不是越多越好的。

对于渐进时间复杂性为 $O(n^x)$ 的串行计算，时间与计算规模 n 的关系为

$$T = cn^x \tag{1-5}$$

如果我们希望得到的理想的可扩展性能表现为计算规模增加 m 倍，使用 p 个处理器可得到相同的计算时间，那么所需处理器数目 p 与计算规模增加的倍数 m 的关系为

$$p = m^x \tag{1-6}$$

由此可见，处理器数目 p 应由计算规模决定。

计算复杂性：以搜索这个计算任务为例。在搜索问题中，给定了一个具体的数 s 和长度为 n 的数组 A（数组中数的位置用 1 到 n 作标记），任务是当 s 在 A 中时，找到 s 的位置，而 s 不在 A 中时，需要报告“未找到”。这时输入的长度即为 n+1。下面的过程即是一个最简单的算法：我们依次扫过 A 中的每个数，并与 s 进行比较，如果相等即返回当前的位置，如果扫遍所有的数而算法仍未停止，则返回“未找到”。如果我们假设 s 在 A 中每个位置都是等可能的，那么算法在找到 s 的条件下需要 $(1+2+\cdots+n)/n=n(n+1)/2n=(n+1)/2$ 的时间。如果 s 不在 A 中，那么需要 $(n+1)$ 的时间。由大 O 表达式的知识，我们知道算法所需的时间即为 $O(n)$。

1.2.4　认识并行计算机

并行机存在相当大的差异，一个优质的并行程序不应过多受到硬件差异的影响。但作为使用者，应对硬件差异有所了解，以更好地分析计算性能，尤其是性能损失的原因。

这里比较 5 类并行计算机 [7]。

芯片多处理器：多核体系。每个处理器所见到的都是一个一致的共享存储器。

对称多处理器（Symmetric Multiprocessor, SMP）：一种所有处理器访问单一逻辑存储器的并行计算机，存储器的一部分在物理上邻近每一个处理器。

异构芯片设计：用一个或多个专用计算引擎来扩展一个标准的处理器，这些专用计算引擎称为附属处理器。其思想是由标准处理器完成通用、难以并行化的计算部分，而由附属处理器完成密集计算部分。比较熟悉的附属部件有图形处理部件（GPU）、为视频游戏设计的细胞处理器（cell）。异构芯片系统不为协同处理单元提供一致性存储器，但主处理器可对所有存储器进行全局访问。

机群：机群是由商品部件构成的并行计算机，如刀片服务器。每一个机群结点在刀片服务器中称为刀片，一个刀片含有一个或几个处理器芯片、一个 RAM 存储器、磁盘存储器和若干通信口以及几个冷却风扇。机群的一个主要特性是存储器不为各机器所共享，处理器只访问自己刀片的存储器，当要与其他处理器通信时，需要借助消息传递机制。

超级计算机：传统上，超级计算机由国家大型实验室和大公司所使用，具有许多不同的体系结构，包括机群。它拥有大规模并行处理器（Massively Parallel Processor, MPP），各并行处理器间同样需要借助消息传递机制通信。

由上可见：芯片多处理器与对称多处理器实现的是一个共享地址空间，是所有处理器可访问的一致性存储器，可实现低时延通信，支持较细粒度的计算；机群和超级计算机实现的是一个分布式地址空间，每个处理器只能访问整个存储器的一部分，在分布地址中，不共享存储器的各个处理器通过消息传递进行相互之间的通信，适宜于粗粒度计算。异构芯片系统具有很高的性能价格比，是目前的“热门”。在异构芯片系统中，附属处理器可以视为当今求解密集计算任务的专用引擎。主处理器与附属处理器并发运行并不是并行性的主要来源，通常大量的并行性嵌入在附属处理器中。需小心处理主处理器与附属处理器之间的通信耗时。

1.3 有限元法中并行计算的基本框架

为实现并行计算，针对计算问题建立的数值算法应当充分利用其内在并行性。研究传统数值算法本身的并行性，并对其进行并行化改造，是实现并行计算行之有效的手段（如图1-5所示）：如果传统数值算法本身具备良好的、明显的并行性，则可直接进行并行算法设计；如果算法不具备良好的并行性，则应从数值分析的角度进行并行化改造。然而，要发展并行数值方法，将传统方法并行化并不是唯一手段，更应该鼓励为了并行计算的目的而建立全新的计算算法，以达成更有效快速的计算。

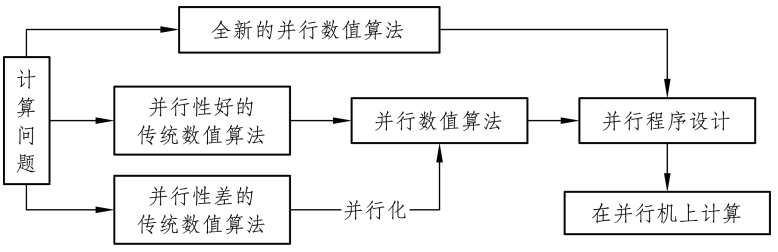

图 1-5 实现并行计算 [6]

传统有限元算法采用串行计算。但它是否可以改造成并行算法呢?

可以看到，求解一个大型复杂的连续体结构是一个分解相当困难的大问题。一个连续体结构内部各点之间不仅需要满足力的平衡关系还要满足变形的协调关系，这样一来，从空间域上看，各点紧密联系，不可分割。这种联系将导致出现阻碍问题分解的数据相关性。

用传统有限元方法求解时，这种数据相关性出现的原因为各单元节点之间必须满足平衡与协调。对于给定的静力线弹性问题，传统有限元方法的计算可以分为两个部分：①单元局部级的计算；②系统方程组的求解。单元局部级的计算包括：$\boldsymbol{B}$ 矩阵的计算；利用 Gauss 积分（或 Hammer 积分）计算单元刚度矩阵及单元荷载向量；由单元刚度矩阵集成系统刚度矩阵及由单元荷载向量集成系统荷载向量。很显然，各单元的计算是相互独立的，与范例 1 一样，单元局部级的计算具有良好的可分解性及直观的空间域并行性，可以改造。而集中了主要计算量的系统方程组的求解是一个线性方程组的求解问题。从数值计算的角度上看，这种无法分解的联系使得系统刚度矩阵中非零元素的分布为具有一定带宽的带状，而不是都分布在对角线上的块状，在对刚度矩阵求逆时很难分解、并行计算。为了将位移法有限元方法并行化，研究者研发出了一种有效的处理方法即子结构法 [9]，它将一个大型结构分成若干个小结构，这些小结构即为大型结构的子结构。从数值计算的角度看，划分子结构后，特殊的编码方式使得在集成的系统刚度矩阵中，与子结构的边界条件相关的元素集中在大型矩阵的第一行块及第一列块，而其他元素以方块状分布在对角线上。采用静力凝聚的思想求解，对角线上方块状分布的元素可以并行求逆，这就成功地把大型矩阵求逆的问题分解了。而从另一个方面解释，子结构法用静力凝聚法求解时实际上是把待求未知量分为了两部分：一部分是对问题进行分解得到的子结构内部的内变量，各子结构的内变量之间相互无关；另一部分是反映各子结构之间相互的平衡、协调关系的边界变量。两个部分分别求解，并行考虑内变量的同时，在边界变量上考虑了问题的内在联系。这一部分亦可以改造。

参考文献

[1] 陈康，郑纬民 . 云计算 : 系统实例与研究现状 [J]. 软件学报，2009, 5: 1337-1348.

[2] 李三立 . 超级计算：人类认识世界的又一次革命 [J]. 中国计算机用户，1996, 1: 10-11.

[3] MAJUMDAR J. Development of parallel algorithms for computer vision[J]. Defence Science Journal, 1996, 46(4): 243-251.

[4] SCOY F L V. Developing software for parallel computing systems [J]. Computer Physics Communications, 1996, 97(1-2): 36-44.

[5] KAI HWANG, GEOFFREY C FOX, JACK J DONGARRA. 云计算与分布式系统：从并行处理到物联网 [M]. 武永卫，等，译 . 北京：机械工业出版社，2013.

[6] 张宝琳，谷同祥，莫克尧 . 数值并行计算原理与方法 [M]. 北京：国防工业出版社，1999.

[7] XAVIER C, IYENGAR S S. 并行算法导论（中译本）[M]. 北京：机械工业出版社，2004.

[8] LIN C, SNYDER L. Principles of Parallel Programming [M]. NewYork: Addison Wesley, 2008.

[9] 周绥平，刘西拉 . 结构矩阵析及 SMIS-PC 程序 [M]. 北京：人民交通出版社，1989.