3.6 并行算法模型

在讨论了分解、映射和最小化交互开销的技术之后，我们现在介绍一些常用的并行算法模型。算法模型通常是通过选择分解和映射技术以及应用适当的策略将交互最小化来构建并行算法的一种方法。

3.6.1 数据并行模型

数据并行模型是最简单的算法模型之一。在这种模型中，任务被静态或半静态地映射到进程上，每个任务对不同的数据执行类似的操作。这种在不同数据项上同时执行相同操作的并行方式称为**数据并行（Data Parallelism）**。工作可以分阶段进行，不同阶段操作的数据可能不同。通常情况下，数据并行计算阶段会穿插交互，以同步任务或为任务获取新数据。由于所有任务都执行类似的计算，因此将问题分解为任务通常基于数据分区，因为统一的数据分区和静态映射足以保证负载平衡。

数据并行算法可以在共享地址空间和消息传递范式中实现。不过，消息传递范式中的分区地址空间可以更好地控制位置，从而更好地处理定位问题。另一方面，共享地址空间可以减轻编程工作，尤其是在算法的不同阶段数据分布不同的情况下。

数据并行模型中的交互开销可通过以下方式降至最低：选择一种保持局部性的分解方式；在适用情况下，重叠计算和交互；使用优化的集体交互例程。数据并行问题的一个主要特点是，对于大多数问题而言，数据并行程度会随着问题规模的增大而增大，因此可以使用更多进程来有效解决更大的问题。

第 3.1.1 节中描述的密集矩阵乘法就是数据并行算法的一个例子。在图 3.10 所示的分解中，所有任务都是相同的，但它们应用于不同的数据。

3.6.2 任务图模型

如第 3.1 节所述，任何并行算法中的计算都可以看作一个任务依赖图。任务依赖图可能是琐碎的（如矩阵乘法），也可能是非琐碎的。不过，在某些并行算法中，任务依赖图被明确用于映射。在**任务图模型（Task Graph Model）**中，任务间的相互关系被用来提高局部性或降低交互成本。这种模型通常用于解决与任务相关的数据量相对于计算量较大的问题。通常，任务是静态映射的，以帮助优化任务间的数据移动成本。有时也会使用分散式动态映射，但即便如此，映射也会使用任务依赖图结构和任务交互模式的相关信息，以尽量减少交互开销。在具有全局可寻址空间的范例中，工作更容易共享，但也有机制可以共享不相连的寻址空间中的工作。

适用于这一模式的减少交互的典型技术包括：在根据任务的交互模式映射任务的同时，通过促进局部性来减少交互的数量和频率；使用异步交互方法将交互与计算重叠。

基于任务图模型的算法实例包括并行排序（第 9.4.1 节）、稀疏矩阵因式分解，以及许多通过分而治之分解法衍生的并行算法。这种由任务依赖图中的独立任务自然表达的并行性被称为**任务并行（Task Parallelism）**。

3.6.3 工作池模型

工作池（Work Pool）或**任务池（Task Pool）**模型的特点是将任务动态映射到进程上，以实现负载平衡，其中任何任务都可能由任何进程执行。任务与进程之间没有理想的预先映射。这种映射可以是集中式的，也可以是分散式的。任务指针可以存储在物理共享列表、优先级队列、哈希表或树中，也可以存储在物理分布式数据结构中。工作在开始时可能是静态可用的，也可能是动态生成的；也就是说，进程可以生成工作并将其添加到全局（可能是分布式）工作池中。如果工作是动态生成的，并且使用的是分散映射，那么就需要终止检测算法（第 11.4.4 节），以便所有进程都能实际检测到整个程序的完成（即所有潜在任务的耗尽），并停止寻找更多工作。

在消息传递范式中，当与任务相关的数据量相对于与任务相关的计算量较少时，通常会使用工作池模型。因此，任务可以随时移动，而不会造成过多的数据交互开销。任务的粒度可以调整，以便在负载不平衡与为添加和提取任务而访问工作池的开销之间取得理想的平衡。

当任务静态可用时，通过分块调度（第 3.4.2 节）或相关方法对循环进行并行化处理，就是使用集中映射的工作池模型的一个例子。并行树形搜索的工作由集中式或分布式数据结构表示，是使用工作池模型的一个例子，其中的任务是动态生成的。

3.6.4 主-从模型

在 "主-从"或 "管理者-工作进程"模式中，一个或多个主进程生成工作并将其分配给工作进程。如果管理者能估算出任务的大小，或者随机映射能充分平衡负载，那么任务可以事先分配好。在另一种情况下，工作者会在不同时间被分配较小的工作。如果主程序生成工作需要耗费大量时间，因此不希望让所有工人等待主程序生成所有工作，那么后一种方案更可取。在某些情况下，工作可能需要分阶段进行，每个阶段的工作必须在下一阶段的工作生成之前完成。在这种情况下，管理者可以让所有工人在每个阶段结束后同步。通常情况下，不需要将工作预先映射到进程中，任何工作者都可以完成分配给它的任何工作。管理者-工人模式可扩展为分级或多级管理者-工人模式，在这种模式中，最高级别的管理者将大块任务分配给第二级别的管理者，第二级别的管理者再将任务细分给自己的工人，并可能亲自完成部分工作。这种模式一般同样适用于共享地址空间或消息传递范式，因为交互自然是双向的；也就是说，管理者知道自己需要分配工作，而工人也知道自己需要从管理者那里获得工作。

在使用主从模式时，应注意确保主模式不会成为瓶颈，如果任务太小（或工人速度相对较快），就可能出现这种情况。在选择任务粒度时，应使执行工作的成本优先于传输工作的成本和同步成本。异步交互可以帮助重叠交互和与主站生成工作相关的计算。如果工作者的请求具有非确定性，异步交互还可以减少等待时间。

3.6.5 流水线模型或生产者-消费者模型

在流水线模型中，数据流通过一连串的进程传递，每个进程都对数据流执行一些任务。这种在数据流上同时执行不同程序的现象被称为**数据流并行（Stream Parallelism）**。除了启动流水线的进程外，新数据的到达会触发流水线中的进程执行新任务。进程可以以线性或多维数组、树或有循环或无循环的一般图的形式组成流水线。管道是由生产者和消费者组成的链条。管道中的每个进程都可以被视为管道中前一个进程的数据项序列的消费者，以及管道中后一个进程的数据生产者。流水线不一定是线性链，也可以是有向图。流水线模型通常涉及任务到流程的静态映射。

负载平衡是任务粒度的函数。粒度越大，填满管道所需的时间就越长，即链中第一个进程产生的触发会传播到最后一个进程，从而使一些进程继续等待。然而，过细的粒度可能会增加交互开销，因为在较小的计算片段之后，进程需要进行交互以接收新数据。适用于该模型的最常见的交互减少技术是与计算重叠交互。

二维流水线的一个例子是并行 LU 因子分解算法，第 8.3.1 节将对此进行详细讨论。

3.6.6 混合模型

在某些情况下，可能会有不止一种模型适用于当前问题，从而形成混合算法模型。混合模型可以由分层应用的多个模型组成，也可以由顺序应用于并行算法不同阶段的多个模型组成。在某些情况下，算法表述可能具有不止一种算法模型的特征。例如，数据可以在任务依赖图的指导下以流水线方式流动。在另一种情况下，主要计算可由任务依赖图描述，但图中的每个节点可代表由多个子任务组成的超任务，这些子任务可能适合数据并行或流水线并行。并行快速排序（第 3.2.5 节和第 9.4.1 节）就是非常适合混合模型的应用之一。