4.7 提高通信速度的方法

本章到目前为止，我们已经推导出了各种通信操作的程序及其通信时间，前提是原始信息不能被分割成更小的部分，并且每个节点都有一个用于发送和接收数据的端口。在本节中，我们将简要讨论放宽这些假设对某些通信操作的影响。

4.7.1 消息分割与部分路由

在第 4.1-4.6 节描述的程序中，我们假设整个 m 字数据包在源节点和目的节点之间沿同一路径传输。如果我们把大信息分成较小的部分，然后通过不同的路径传输这些部分，有时就能更好地利用通信网络。我们已经证明，除了One-to-All Broadcast、All-to-One Reduction、All Reduce等少数例外情况外，本章讨论的通信操作对于大型报文来说几乎都是最优的；也就是说，这些操作的成本中与 t_w 相关的项无法渐进地减少。在本节中，我们将介绍三种全局通信操作的渐进最优算法。

需要注意的是，本节的算法依赖于 m 足够大，可以分割成 p 个大致相等的部分。因此，前面的算法对于较短的信息仍然有用。将本节算法的成本与本章前面介绍的相同操作的成本进行比较，会发现当信息被分割时，与 t_s 相关的项会增加，与 t_w 相关的项会减少。因此，根据 t_s、t_w 和 p 的实际值，报文大小 m 有一个截止值，只有长度大于截止值的报文才会从本节的算法中受益。

One-to-All Broadcast

考虑从一个源节点向一个 p 节点集合中的所有节点广播一条大小为 m 的信息 M。如果 m 足够大，以至于可以将 M 分成大小分别为 m/p 的 p 个部分 M_0、M_1、...、M_{p-1}，那么Scatter操作（第 4.4 节）可以在 $t_s \log p + t_w(m/p)(p-1)$ 的时间内将 M_i 发送到节点 i。请注意，One-to-All Broadcast的理想结果是将 $M = M_0 \cup M1 \cup \ldots \cup M_{p-1}$ 放在所有节点上。这可以通过在Scatter操作后对每个节点上大小为 m/p 的报文进行All-to-All Broadcast来实现。在超立方体上，完成这种All-to-All Broadcast的时间为 $t_s \log p + t_w(m/p)(p-1)$。因此，在超立方体上，One-to-All Broadcast可在以下时间内完成

$$T = 2 \times (t_s \log p + t_w(p-1)\frac{m}{p}) \tag{1}$$

$$\approx 2 \times (t_s \log p + mt_w) \tag{2}$$

与One-to-All Broadcast和All-to-One Reduction的开销公式相比，该算法的启动成本增加了一倍，但因 t_w 项引起的成本降低了 $(\log p)/2$ 倍。同样，在线性阵列和网状互连网络上，One-to-All Broadcast也可以得到改进。

All-to-One Reduction

All-to-One Reduction是One-to-All Broadcast的对偶。因此，可以通过颠倒One-to-All Broadcast的通信方向和顺序来获得All-to-One Reduction算法。上文我们展示了如何通过先执行Scatter操作，再执行All-to-All Broadcast来获得最佳的One-to-All Broadcast算法。因此，利用对偶性的概念，我们应该能够通过先执行All-to-All Reduction（All-to-All Broadcast的对偶），再执行Gather操作（Scatter的对偶），来实现All-to-One Reduction。我们把这种算法的细节留给读者练习。

All Reduce

由于All Reduce操作在语义上等同于All-to-One Reduction和One-to-All Broadcast，因此我们可以利用上面介绍的这两种操作的渐近最优算法来构建All Reduce操作的类似算法。将All-to-One Reduction和One-to-All Broadcast分解为不同的操作，可以证明All Reduce操作可以通过All-to-All Reduction、Gather、Scatter和All-to-All Broadcast来完成。由于中间的Gather和Scatter操作会使彼此的效果相抵消，所以All Reduce操作只需要一个All-to-All Reduction和All-to-All Broadcast。首先，将 p 个节点上的 m 个字的信息从逻辑上拆分成大小大致为 m/p 个字的 p 个部分。然后，将 p_i 上的第 i 个分量进行All-to-All Reduction。经过这一步骤后，每个节点上的最终结果就只剩下一个不同的 m/p 字分量。All-to-All Broadcast可以在每个节点上构建这些分量的连接。

一个 p 节点超立方体互连网络允许在 $t_s \log p + t_w(m/p)(p-1)$ 时间内，对大小为 m/p 的报文进行All-to-One Reduction和One-to-All Broadcast。因此，All Reduce操作可在以下时间内完成

$$T = 2 \times (t_s \log p + t_w (p-1)\frac{m}{p}) \tag{3}$$

$$\approx 2 \times (t_s \log p + t_w m) \tag{4}$$

4.7.2 全端口通信

在并行结构中，单个节点可能有多个通信端口，并与集合中的其他节点相连。例如，二维环绕网格中的每个节点有四个端口，而 d 维超立方中的每个节点有 d 个端口。在本书中，我们通常假设**单端口通信（Single-Port Communication）**模型。在单端口通信中，一个节点每次只能在其一个端口上发送数据。同样，节点每次也只能在一个端口上接收数据。但是，节点可以同时发送和接收数据，可以在同一个端口，也可以在不同的端口。与单端口模型相反，**全端口通信（All-Port Communication）**模型允许在连接节点的所有通道上同时进行通信。

在具有全端口通信的 p 节点超立方体上，One-to-All、All-to-All Broadcast 通信和 Personalized Communication的通信时间表达式中的 t_w 系数都比其对应的单端口系数小 $\log p$。

尽管速度明显加快，但全端口通信模型有一定的局限性。例如，它不仅难以编程，而且要求报文足够大，以便在不同通道之间有效分割。在一些并行算法中，报文大小的增加意味着节点计算粒度的相应增加。当节点处理大型数据集时，如果算法的计算复杂度高于通信复杂度，节点间通信时间就会被计算时间所支配。例如，在矩阵乘法的情况下，节点间传输 n^2 个字的数据需要进行 n^3 次计算。如果通信时间只占整个并行运行时间的一小部分，那么通过使用复杂的技术来改善通信，对并行算法的整体运行时间而言并无太大优势。

全端口通信的另一个限制是，只有当数据在内存中的获取和存储速度足以支持所有并行通信时，全端口通信才能有效。例如，要在 p 节点超立方体上有效利用全端口通信，内存带宽必须比单信道通信带宽至少大对数 p 倍；也就是说，内存带宽必须随着节点数的增加而增加，以支持所有端口的同时通信。一些现代并行计算机，如 IBM SP，对这一问题有一个非常自然的解决方案。分布式内存并行计算机的每个节点都是一个 NUMA 共享内存多处理器。多个端口由不同的内存库提供服务，如果将发送和接收数据的缓冲区适当放置在不同的内存库中，就可以充分利用内存和通信带宽。