Chapter 1

Practicalities

The subject of this book is the description, analysis, implementation, and application of collective communication algorithms on massively parallel parallel computers. In this chapter, we give an overview of the primary collective communications that will be treated. To facilitate analysis of the cost of the various operations, we introduce a simple parallel programming model to be used throughout the book. This will also allow us to put lower bounds on the cost of the operations, allowing us to evaluate the optimality of the various approaches. The cost of collective communications will be closely linked to the topology of the communication network. We will consider popular topologies, including hypercubes, linear arrays and rings, and higher dimensional meshes and tori. We have choosen to express our algorithms using the Message-Passing Interface, making the resulting codes portable among message-passing architectures. We give a rudimentary treatment of this interface at the end of this chapter.

1.1 A Model of Parallel Computation

The target architectures for our algorithms are distributed-memory multidimensional mesh multicomputers, including Multiple Instruction Multiple Data (MIMD) machines such as the Symult S2000, the Intel Touchstone Delta and Paragon systems, and the Cray T3D. For historic completeness, we also include hypercubes in our target architectures, which makes the results applicable to the Intel iPSC/860 and NCUBE/2.

Commercially available parallel systems change rapidly. Hence we will not explain our techniques as they are applied to particular machines, since the book would then be outdated before it was even published. Rather, we will present a simple programming model, which will be used to explain and evaluate the algorithms and issues.

1.1.1 Indexing of Processors

We assume the parallel architecture has p nodes. Often, we will index these nodes $\mathbf{P}_0, \ldots, \mathbf{P}_{p-1}$, regardless of the underlying architecture. Other times, it may become convenient to index them relative to a higher dimensional mesh. E.g., for two dimensional meshes, we may index the nodes $\mathbf{P}_{ij}$, where i and j indicate the row and column indices in the mesh, respectively.

1.1.2 Single-Ported Architectures

We assume that a node's network interface is single-ported. That is, at most one message can be sent and one message received at full node-to-network bandwidth. If k messages are received simultaneously, then the effective bandwidth into the node is reduced by a factor of k.

Many of the presented algorithms can be extended to multi-ported architectures, where more than one message can be sent of received simultaneously. Rather than treating these extensions in the book, we hope to present the techniques clearly enough that the reader can derive algorithms for multi-ported architectures.

1.1.3 Bidirectional Channels

Channels are assumed to be capable of transmitting a single message in each direction at full bandwidth. If k messages traverse a channel in the same direction, then the transmission time is increased by a factor of k.

1.1.4 Distance between two nodes

We will assume that messages between two nodes will always traverse the same prescribed path of physical links in the network. We will call the number of channels traversed between two nodes the *distance* between those two nodes.

1.1.5 Cost of Communication between Neighbors

We assume that the cost of sending/receiving a length n message to/from a physical neighbor can be modelled by:

$$\alpha + n\beta$$

where α is a fixed-cost message overhead or startup time independent of the length of the message, and β is the transmission cost per unit of data.

1.1.6 Pipelining of Messages

Let us assume that we wish to send a message between two nodes that are not physically neighbors, i.e., they are a distance s apart. Routing a message from between these two nodes could be accomplished by sending the message from neighboring node to neighboring node along the route prescribed by the routing algorithm, treating the message as a unit. The cost for a message of length n would become

$$s\alpha + sn\beta$$

For long message, we can do considerably better: First, the message is broken into k pieces where for our explanation it suffices to assume n is an integer multiple of k. Sending the message between the same two nodes can now be accomplished by *pipelining* the packets: The source node starts by sending the first n/k pieces of data to the first neighbor along the path. While this neighboring node passes the data to its neighbor, the root sends the next n/k pieces, etc. The time required for the first packet to reach the last node is $(s)(\alpha + n/k\beta)$, after which $k-1$ more packets must arrive, requiring time $\alpha + n/k\beta$ each, for a total time of

$$\begin{aligned} T_{\text{pipe}}(n,s,k) &= (s+k-1)\left(\alpha + \frac{n}{k}\beta\right) && (1.1) \\ &= (s+k-1)\alpha + \frac{s+k-1}{k}n\beta && (1.2) \end{aligned}$$

Notice that *if* the message is very large, the β term will start dominating and *if* k is picked large enough, $(p+k-1)/k \rightarrow 1$, so that asymptotically the cost of broadcasting the message becomes $n\beta$. We can compute the optimal number of packets, k_{opt} by pretending that the above equation is a continuous function of k, and differentiating with respect to that variable.

$$\frac{dT(n,s,k)}{dk} = \alpha - \frac{s-1}{k^2} n\beta$$

Observing that for $k > 0$ the function is convex, and hence has a unique minimum, allows us to solve for the optimal k by setting the derivative to zero:

$$\alpha - \frac{s-1}{k_{\text{opt}}^2} n\beta = 0 \Rightarrow k_{\text{opt}} = \sqrt{(s-1)n\frac{\beta}{\alpha}}$$

Naturally, $k_{\text{opt}} < 1$ or $k_{\text{opt}} > n$ doesn't make sense, so that the complete formula becomes

$$k_{\text{opt}}(n,s) = \begin{cases} 1 & \text{if } \sqrt{(s-1)n\frac{\beta}{\alpha}} < 1 \\ n & \text{if } \sqrt{(s-1)n\frac{\beta}{\alpha}} > n \\ \sqrt{(s-1)n\frac{\beta}{\alpha}} & \text{otherwise} \end{cases}$$

This gives a total time for sending the message a distance s of

$$T_{\text{pipe}}(n,s,k_{\text{opt}}(n,s)) = \begin{cases} s\alpha + sn\beta & \text{if } \sqrt{(s-1)n\frac{\beta}{\alpha}} < 1 \\ (s+n-1)(\alpha+\beta) & \text{if } \sqrt{(s-1)n\frac{\beta}{\alpha}} > n \\ (\sqrt{(s-1)\alpha} + \sqrt{n\beta})^2 & \text{otherwise} \end{cases}$$

Finally, if the distance s is fixed,

$$\lim_{n\to\infty} T_{\text{pipe}}(n,s,k_{\text{opt}}(n,s)) = n\beta$$

Thus for long messages, the cost of sending a message between two nodes using this kind of pipelining approach is essentially independent of the distance between the two nodes.

1.1.7 Model of Communication Cost

This is a good time to note that in current generation networks, there is a cost for introducing a message into the network, and a cost for forwarding a message at each hop, which is handled by hardware and is typically several orders of magnitude smaller. As a result, a more accurate model for sending a message a distance s using k packets is

$$T(k,n,s) = \alpha + (s+k-1)(\alpha_{\text{net}} + \frac{n}{k}\beta)$$

with $\alpha_{\text{net}} \approx 0$. In this case, $k_{\text{opt}} \approx 0$, and, since typically $\alpha >> \beta$ the time for sending a message a distance s becomes essentially $\alpha + n\beta$. This allows us to claim that if no network conflicts occur, we can model the time for sending a message between any two nodes in the network by

$$\alpha + n\beta$$

This assumption models networks that include hardware cut-through routing or worm-hole routing can be models quite accurately by this simple model.

1.1.8 Network Conflicts

Notice that since messages are assumed to be pipelined, leading to the cost model $\alpha + n\beta$, our model essentially assumes that the whole path between two nodes is occupied during the communication. Thus, if a single link accomodates more than one messages, e.g. if two paires of nodes are communicating, and the paths share this link, network conflicts occur. Thus, in the presence of network conflict, we will model the cost of sending a message between two nodes as

$$\alpha + kn\beta$$

where k equals the maximum over all links in the path of the number of conflicts on those links.

1.1.9 Cost of Computation

We will denote the time required for a typical arithmetic operation, e.g. a multiply or add, as γ.

1.1.10 Summary of Assumptions

1.2 Common Collective Communications

To describe the common collective communications that we will treat in this book, we start by introducing some notation.

Let x, $x^{(j)}$, and y be vectors of data of length n. For many of the operations, we will need to partition these vectors into p pieces, where p equals the number of nodes. Thus

$$x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{p-1} \end{array} \right), \quad x^{(j)} = \left(\begin{array}{c} x_0^{(j)} \\ \hline x_1^{(j)} \\ \hline \vdots \\ \hline x_{p-1}^{(j)} \end{array} \right) \quad \text{and} \quad y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{p-1} \end{array} \right)$$

where x_i, $x_i^{(j)}$, and y_i are of length $n_i \geq 0$, with $\sum_{i=0}^{p-1} n_i = n$. Also, given a associative and commutative *element-wise* operator $\oplus$, we will assume that

$$y = x^{(0)} \oplus x^{(1)} \oplus \cdots \oplus x^{(p-1)}$$

Given this notation, the various collective operations can now be defined.

The common collective communications are now given by

Broadcast: Initially, one node, $\mathbf{P}_k$, owns a vector of data, x. After the broadcast, all nodes own a copy of x. This communication is often referred to as a one-to-all broadcast. The MPI call is `MPI_Bcast`.

Scatter: Initially, one node, $\mathbf{P}_k$, owns a vector of data, x. After the scatter, each node, $\mathbf{P}_i$, owns a piece of the data x_i, $i = 0, \ldots, p-1$. This communication is ofter referred to as a one-to-all personalized broadcast. The MPI call is **MPI_Scatter**.

Gather: Before the gather, each node, $\mathbf{P}_i$, owns a piece of the data x_i, $i = 0, \ldots, p-1$. Afterwards, one node, $\mathbf{P}_k$, owns the entire vector of data, x. The MPI call is **MPI_Gather**.

Collect: Initially, all nodes have a piece of the data: x_i is owned by $\mathbf{P}_i$. After the collect, all nodes own all of the date, x. This communication is often referred to as an all-to-all (simultaneous) broadcast, or a simultanious gather. The MPI call is **MPI_Allgather**.

Operation	BEFORE						AFTER					
	$\mathbf{P}_0$	$\mathbf{P}_1$	$\cdots$	$\mathbf{P}_k$	$\cdots$	$\mathbf{P}_{p-1}$	$\mathbf{P}_0$	$\mathbf{P}_1$	$\cdots$	$\mathbf{P}_k$	$\cdots$	$\mathbf{P}_{p-1}$
Broadcast				x			x	x	x	x	x	x
Reduce-to-one	$x^{(0)}$	$x^{(1)}$	$\cdots$	$x^{(k)}$	$\cdots$	$x^{(p-1)}$				y		
Scatter				x_0 x_1 $\vdots$ x_k $\vdots$ x_{p-1}			x_0	x_1	$\ddots$	x_k	$\ddots$	x_{p-1}
Gather	x_0	x_1	$\ddots$	x_k	$\ddots$	x_{p-1}				x_0 x_1 $\vdots$ x_k $\vdots$ x_{p-1}		
Reduce-to-all	$x^{(0)}$	$x^{(1)}$	$\cdots$	$x^{(k)}$	$\cdots$	$x^{(p-1)}$	y	y	y	y	y	y
Collect	x_0	x_1	$\ddots$	x_k	$\ddots$	x_{p-1}	x_0 x_1 $\vdots$ x_k $\vdots$ x_{p-1}	x_0 x_1 $\vdots$ x_k $\vdots$ x_{p-1}	$\cdots$ $\cdots$ $\cdots$ $\cdots$	x_0 x_1 $\vdots$ x_k $\vdots$ x_{p-1}	$\cdots$ $\cdots$ $\cdots$ $\cdots$	x_0 x_1 $\vdots$ x_k $\vdots$ x_{p-1}
Distr. Red.	$x_0^{(0)}$ $x_1^{(0)}$ $\vdots$ $x_k^{(0)}$ $\vdots$ $x_{(0)p-1}$	$x_0^{(1)}$ $x_1^{(1)}$ $\vdots$ $x_k^{(1)}$ $\vdots$ $x_{(1)p-1}$	$\cdots$ $\cdots$ $\cdots$ $\cdots$	$x_0^{(k)}$ $x_1^{(k)}$ $\vdots$ $x_k^{(k)}$ $\vdots$ $x_{(k)p-1}$	$\cdots$ $\cdots$ $\cdots$ $\cdots$	$x_0^{(p-1)}$ $x_1^{(p-1)}$ $\vdots$ $x_k^{(p-1)}$ $\vdots$ $x_{p-1}^{(p-1)}$	y_0	y_1	$\ddots$	y_k	$\ddots$	y_{p-1}

Reduce-to-one: Initially, all nodes, $\mathbf{P}_j$, own a vector on length n, $x^{(j)}$. After the reduce-to-one, the result $y = x^{(0)} \oplus \cdots \oplus x^{(p-1)}$ is owned by one node, $\mathbf{P}_k$. The MPI call is **MPI_Reduce**.

Reduce-to-all: Identical to reduce-to-one, except that the result is duplicated on all nodes. The MPI call is **MPI_Allreduce**.

Distributed reduce: Identical to reduce-to-one, except that each node, $\mathbf{P}_i$ owns a portion of the final result, y_i.

All-scatter: This operation is equivalent to performing a scatter from every node. It is often called a all-to-all personalized broadcast. More on this operation in Chapter ??. The MPI call is **MPI_Alltoall**.

1.3 Lower bounds

Before launching into the discussion of collective communication algorithms, it is very useful to present lower bounds on the time required in order to better gauge the algorithms. In this section, we give handwaving arguments to derive these lower bounds. We will distinguish between three components of communication cost: terms due to latency, bandwidth, and computation. A table of lower bounds is given in Table 1.1.

Communication	Latency	Bandwidth	Computation
Broadcast	$\lceil log_2(p) \rceil \alpha$	$n\beta$	–
Reduction-to-one	$\lceil log_2(p) \rceil \alpha$	$n\beta$	$\frac{p-1}{p} n\gamma$
Scatter	$\lceil log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–
Gather	$\lceil log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–
Collect	$\lceil log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–
Distributed Reduction	$\lceil log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$
Reduction-to-all	$\lceil log_2(p) \rceil \alpha$	$2\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$
All-scatter	$\lceil log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	–

Table 1.1: Lower bounds for the different components of communication cost. Pay particular attention to the conditions for the lower bound, given in the text

Latency. The lower bound on latency is derived by the simple observation that for all collective communications, at least one nodes has data that must somehow arrive at all other nodes. Under our model, at each step, we can at most double the number of nodes that get that data.

Computation. Only the reduction communications require computation. The computation involved would require $(p-1)n$ computations if executed on a single node, or time $(p-1)n\gamma$. Distributing this computation perfectly among the nodes reduces the time to $(p-1)n\gamma/p$ under ideal circumstances. Hence the lower bound.

Bandwidth. For the broadcast and reduction-to-one, the root node must either send or receive all n item, repectively. The cost of this is bounded below by $n\beta$. For the scatter and gather, the root node must either send or receive $\frac{p-1}{p}n$ items, with a cost of at least $\frac{p-1}{p}n\beta$. The same is the case for all nodes during the collect and distributed reduction.

The reduction-to-all is somewhat more complicated: *If the lower bound on computation is to be achieved,* one can argue that $\frac{p-1}{p}n$ items must leave each node, and $\frac{p-1}{p}n$ items must be received by each node after the computation is completed, for a total cost of at least $2\frac{p-1}{p}n\beta$.

We will discuss lower bounds for the all-scatter in Chapter ??.