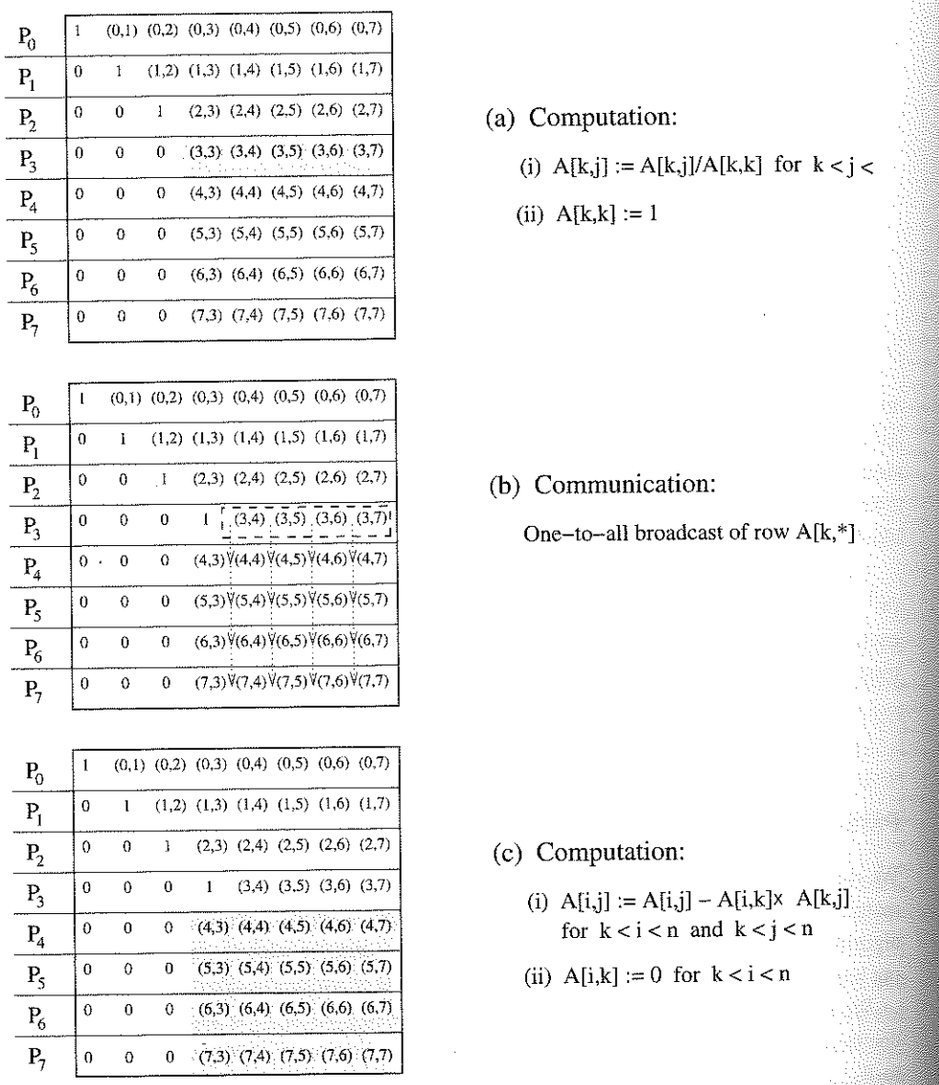

Figure 8.6 Gaussian elimination steps during the iteration corresponding to $k = 3$ for an 8×8 matrix partitioned rowwise among eight processes.

belong to the same process. So this step does not require any communication. In the second computation step of the algorithm (the elimination step of line 12), the modified (after division) elements of the kth row are used by all other rows of the active part of the matrix. As Figure 8.6(b) shows, this requires a one-to-all broadcast of the active part of the kth row to the processes storing rows $k + 1$ to $n - 1$. Finally, the computation $A[i, j] := A[i, j] - A[i, k] \times A[k, j]$ takes place in the remaining active portion of the matrix, which is shown shaded in Figure 8.6(c).

The computation step corresponding to Figure 8.6(a) in the kth iteration requires $n - k - 1$ divisions at process P_k. Similarly, the computation step of Figure 8.6(c) involves $n - k - 1$ multiplications and subtractions in the kth iteration at all processes P_i, such that $k < i < n$. Assuming a single arithmetic operation takes unit time, the total time spent in computation in the kth iteration is $3(n - k - 1)$. Note that when P_k is performing the divisions, the remaining $p - 1$ processes are idle, and while processes $P_{k+1}, \ldots, P_{n-1}$ are performing the elimination step, processes $P_0, \ldots, P_k$ are idle. Thus, the total time spent during the computation steps shown in Figures 8.6(a) and (c) in this parallel implementation of Gaussian elimination is $3\Sigma_{k=0}^{n-1}(n - k - 1)$, which is equal to $3n(n - 1)/2$.

The communication step of Figure 8.6(b) takes time $(t_s + t_w(n-k-1)) \log n$ (Table 4.1). Hence, the total communication time over all iterations is $\Sigma_{k=0}^{n-1}(t_s + t_w(n - k - 1)) \log n$, which is equal to $t_s n \log n + t_w(n(n - 1)/2) \log n$. The overall parallel run time of this algorithm is

$$T_P = \frac{3}{2}n(n - 1) + t_s n \log n + \frac{1}{2}t_w n(n - 1) \log n. \tag{8.18}$$

Since the number of processes is n, the cost, or the process-time product, is $\Theta(n^3 \log n)$ due to the term associated with t_w in Equation 8.18. This cost is asymptotically higher than the sequential run time of this algorithm (Equation 8.17). Hence, this parallel implementation is not cost-optimal.

Pipelined Communication and Computation We now present a parallel implementation of Gaussian elimination that is cost-optimal on n processes.

In the parallel Gaussian elimination algorithm just presented, the n iterations of the outer loop of Algorithm 8.4 execute sequentially. At any given time, all processes work on the same iteration. The $(k + 1)$th iteration starts only after all the computation and communication for the kth iteration is complete. The performance of the algorithm can be improved substantially if the processes work asynchronously; that is, no process waits for the others to finish an iteration before starting the next one. We call this the ***asynchronous*** or ***pipelined*** version of Gaussian elimination. Figure 8.7 illustrates the pipelined Algorithm 8.4 for a 5×5 matrix partitioned along the rows onto a logical linear array of five processes.

During the kth iteration of Algorithm 8.4, process P_k broadcasts part of the kth row of the matrix to processes $P_{k+1}, \ldots, P_{n-1}$ (Figure 8.6(b)). Assuming that the processes form a logical linear array, and P_{k+1} is the first process to receive the kth row from process P_k. Then process P_{k+1} must forward this data to P_{k+2}. However, after forwarding the kth row

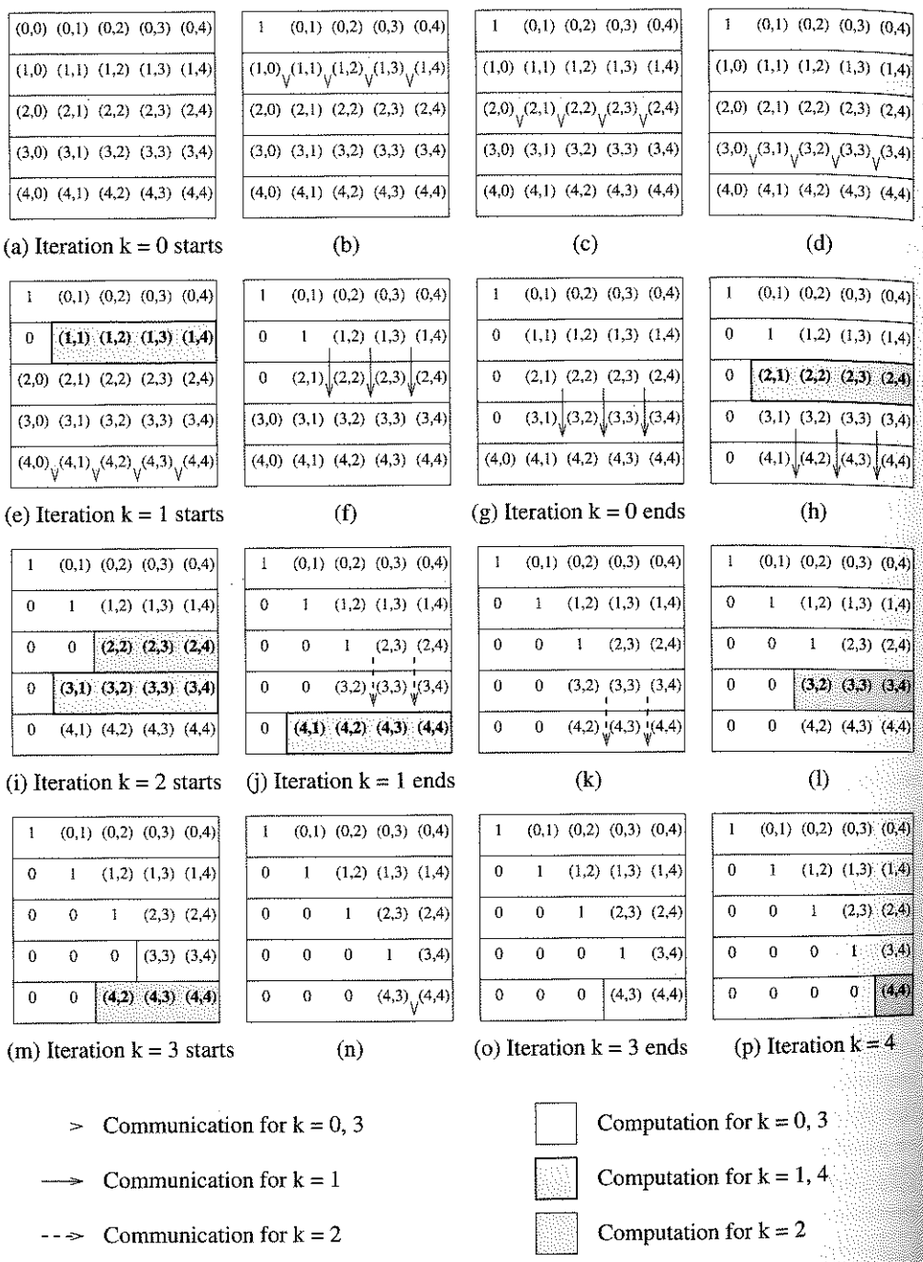

Figure 8.7 Pipelined Gaussian elimination on a 5 × 5 matrix partitioned with one row per process.

to P_{k+2}, process P_{k+1} need not wait to perform the elimination step (line 12) until all the processes up to P_{n-1} have received the kth row. Similarly, P_{k+2} can start its computation as soon as it has forwarded the kth row to P_{k+3}, and so on. Meanwhile, after completing the computation for the kth iteration, P_{k+1} can perform the division step (line 6), and start the broadcast of the $(k+1)$th row by sending it to P_{k+2}.

In pipelined Gaussian elimination, each process independently performs the following sequence of actions repeatedly until all n iterations are complete. For the sake of simplicity, we assume that steps (1) and (2) take the same amount of time (this assumption does not affect the analysis):

1. If a process has any data destined for other processes, it sends those data to the appropriate process.
2. If the process can perform some computation using the data it has, it does so.
3. Otherwise, the process waits to receive data to be used for one of the above actions.

Figure 8.7 shows the 16 steps in the pipelined parallel execution of Gaussian elimination for a 5 × 5 matrix partitioned along the rows among five processes. As Figure 8.7(a) shows, the first step is to perform the division on row 0 at process P_0. The modified row 0 is then sent to P_1 (Figure 8.7(b)), which forwards it to P_2 (Figure 8.7(c)). Now P_1 is free to perform the elimination step using row 0 (Figure 8.7(d)). In the next step (Figure 8.7(e)), P_2 performs the elimination step using row 0. In the same step, P_1, having finished its computation for iteration 0, starts the division step of iteration 1. At any given time, different stages of the same iteration can be active on different processes. For instance, in Figure 8.7(h), process P_2 performs the elimination step of iteration 1 while processes P_3 and P_4 are engaged in communication for the same iteration. Furthermore, more than one iteration may be active simultaneously on different processes. For instance, in Figure 8.7(i), process P_2 is performing the division step of iteration 2 while process P_3 is performing the elimination step of iteration 1.

We now show that, unlike the synchronous algorithm in which all processes work on the same iteration at a time, the pipelined or the asynchronous version of Gaussian elimination is cost-optimal. As Figure 8.7 shows, the initiation of consecutive iterations of the outer loop of Algorithm 8.4 is separated by a constant number of steps. A total of n such iterations are initiated. The last iteration modifies only the bottom-right corner element of the coefficient matrix; hence, it completes in a constant time after its initiation. Thus, the total number of steps in the entire pipelined procedure is $\Theta(n)$ (Problem 8.7). In any step, either $O(n)$ elements are communicated between directly-connected processes, or a division step is performed on $O(n)$ elements of a row, or an elimination step is performed on $O(n)$ elements of a row. Each of these operations take $O(n)$ time. Hence, the entire procedure consists of $\Theta(n)$ steps of $O(n)$ complexity each, and its parallel run time is $O(n^2)$. Since n processes are used, the cost is $O(n^3)$, which is of the same order as the sequential complexity of Gaussian elimination. Hence, the pipelined version of parallel Gaussian elimination with 1-D partitioning of the coefficient matrix is cost-optimal.