```
            cost[low][high] = bestcost;
            root[low][high] = bestroot;
         }
      }

      /* Print structure of binary search tree */

      print_root(root, 0, n-1);


/* Print the root of the subtree spanning keys
   'low' through 'high' */

void print_root (int **root, int low, int high) {
   printf ("Root of tree spanning %d-%d is %d\n",
      low, high, root[low][high+1]);
   if (low < root[low][high+1]-1)
      print_root (root, low, root[low][high+1]-1);
   if (root[low][high+1] < high-1)
      print_root (root, root[low][high+1]+1, high);
}

/* Allocate a two-dimensional array with 'm' rows and
   'n' columns, where each entry occupies 'size' bytes */

void alloc_matrix (void ***a, int m, int n, int size)
{
   int i;
   void *storage;
   storage = (void *) malloc (m * n * size);
   *a = (void **) malloc (m * sizeof(void *));
   for (i = 0; i < m; i++) {
      (*a)[i] = storage + i * n * size;
   }
}
```

Figure 8.22 (contd.) C program implementing dynamic programming algorithm to find an optimal binary search tree.

Use the partitioning-communication-agglomeration-mapping design methodology to implement a parallel version of this program. (Hint: You need to find an agglomeration that will allow multiple processes to be computing concurrently.)