

Name: \_\_\_\_\_

Directions: **Work only on this sheet** (on both sides, if needed). MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

**IMPORTANT NOTE:** If you believe that nothing needs to be placed into a blank, simply give **Nothing** as your answer.

1. (50) Consider the mutual outlinks example, beginning on p.93. Below is a different version of **dowork()**. One of the differences is that it uses dynamic loop scheduling. Another difference is that it doesn't use **atomic** or **critical**. Fill in the blanks.

```
1 float dowork()
2 {
3     #pragma omp parallel
4     { int i;
5         #pragma omp for BLANKa
6         for (BLANKb) {
7             tot += procpairs(i);
8         }
9     }
10    BLANKc
11    BLANKd
12    int divisor = n * (n-1) / 2;
13    return ((float) tot)/divisor;
14 }
```

2. (50) Below is an MPI program that removes 0s from an array. The strategy is that first the manager node breaks the original array into equal-sized chunks, sending one for each worker. Each worker removes the 0s and sends back the nonzero elements. The manager collects these into an array **no0s**. Fill in the blanks below:

```
1 #include <mpi.h>
2 #include <stdlib.h>
3 #define MAXN 100000
4 #define MAXNPROCS 100
5 #define DATAMSG 0
6 #define NEWDATAMSG 1
7
8 int nnodes, // number of MPI processes
9     n, // size of original array
10    me, // my MPI ID
11    has0s[MAXN], // original data
12    no0s[MAXN], // 0-free data
13    nno0s; // number of non-0 elements
14 int debug;
15
16 // not shown
17 init(int argc, char **argv)
18
19 void managernode()
20 {
21     MPI_Status status;
22     int i;
23     int lenchunk;
24     // assumed divides evenly
25     lenchunk = n / nnodes;
26     for (i = 1; i < nnodes; i++) {
27         BLANKa
28     }
29     int k = 0;
30     for (i = 1; i < nnodes; i++) {
31         BLANKb
```

```
32         BLANKc
33         BLANKd
34     }
35     nno0s = k;
36 }
37
38 // not shown
39 void remov0s(int *oldx, int n, int *newx, int *nnewx)
40
41 void workernode()
42 {
43     int lenchunk;
44     MPI_Status status;
45     BLANKe
46     BLANKf
47     remov0s(has0s, lenchunk, no0s, &nno0s);
48     BLANKg
49 }
50
51 // not shown
52 int main(int argc, char **argv)
```

## Solutions:

### 1.

```
1 float dowork()
2 {
3     #pragma omp parallel
4     { int i;
5         #pragma omp for reduction(+:tot) schedule(dynamic)
6         for (i = 0; i < n-1; i++) {
7             tot += procpairs(i);
8         }
9     }
10    int divisor = n * (n-1) / 2;
11    return ((float) tot)/divisor;
12 }
```

### 2.

```
1 #include <mpi.h>
2 #include <stdlib.h>
3
4 #define MAX_N 100000
5 #define MAX_NPROCS 100
6 #define DATA_MSG 0
7 #define NEWDATA_MSG 1
8
9 int nnodes, // number of MPI processes
10    n, // size of original array
11    me, // my MPI ID
12    has0s[MAX_N], // original data
13    no0s[MAX_N], // 0-free data
14    nno0s; // number of non-0 elements
15
16 int debug;
17
18 init(int argc, char **argv)
19 {
20     int i;
21     MPI_Init(&argc,&argv);
22     MPI_Comm_size(MPLCOMM_WORLD,&nnodes);
23     MPI_Comm_rank(MPLCOMM_WORLD,&me);
24     n = atoi(argv[1]);
25     if (me == 0) {
26         for (i = 0; i < n; i++)
27             has0s[i] = rand() % 4;
28     } else {
29         debug = atoi(argv[2]);
30         while (debug);
31     }
32 }
33
34 void managernode()
35 {
36     MPI_Status status;
37     int i;
38     int lenchunk;
39     lenchunk = n / (nnodes-1); // assumed divides evenly
40     for (i = 1; i < nnodes; i++) {
41         MPI_Send(has0s+(i-1)*lenchunk, lenchunk,
42                 MPI_INT, i, DATA_MSG, MPLCOMM_WORLD);
43     }
44     int k = 0;
45     for (i = 1; i < nnodes; i++) {
46         MPI_Recv(no0s+k, MAX_N,
47                 MPI_INT, i, NEWDATA_MSG, MPLCOMM_WORLD, &status);
48         MPI_Get_count(&status, MPI_INT, &lenchunk);
49         k += lenchunk;
50     }
51     nno0s = k;
52 }
53
54 void remov0s(int *oldx, int n, int *newx, int *nnewx)
```

```

55 { int i, count = 0;
56   for (i = 0; i < n; i++)
57     if (oldx[i] != 0) newx[count++] = oldx[i];
58   *nnewx = count;
59 }
60
61 void workernode()
62 {
63   int lenchunk;
64   MPI_Status status;
65   MPI_Recv(has0s, MAX_N,
66            MPLINT, 0, DATA_MSG, MPLCOMM_WORLD, &status);
67   MPI_Get_count(&status, MPLINT, &lenchunk);
68   remov0s(has0s, lenchunk, no0s, &nno0s);
69   MPI_Send(no0s, nno0s,
70            MPLINT, 0, NEWDATA_MSG, MPLCOMM_WORLD);
71 }
72
73 int main(int argc, char **argv)
74 {
75   int i;
76   init(argc, argv);
77   if (me == 0 && n < 25) {
78     for (i = 0; i < n; i++) printf("%d ", has0s[i]);
79     printf("\n");
80   }
81   if (me == 0) managernode();
82   else workernode();
83   if (me == 0 && n < 25) {
84     for (i = 0; i < n; i++) printf("%d ", no0s[i]);
85     printf("\n");
86   }
87   MPI_Finalize();
88 }

```