Name: _____

Directions: Work only on this sheet (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, SHOW YOUR WORK.

1. (10) Fill in the blank with a three- or four-letter abbreviation for a general software category, or else the name of a specific software package: If we have a hypercube machine and wish to use the shared-memory programming paradigm, we should use _____.

2. (15) In the PVM program on p.52 of our text, which PVM library function is analogous to MPI's MPI_Comm_rank()?

3. (15) Suppose the sorting algorithm described in Section 5.3.2 of our text is implemented in MPI. Show how to code the first recv() at the top of p.148 in MPI. Assume the numbers are integers and the message type is stated in a #define as NUMBER_TYPE. Make sure to write not only the call to MPI_Recv() itself but also the code which determines the value of i.

4. (15) Consider our example MPI program which solves systems of linear equations. Rewrite the line

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

using MPI_Send() and/or MPI_Recv().

5. (15) Look at the sample MulSim program in our printed lecture notes. Suppose we had forgotten to include calls to LOCK() and UNLOCK(). Let wrong_cc denote the final value of CompositeCount under this circumstance, and let true_cc denote the correct value. Which of the following statements is correct? (a) wrong_cc $\leq$ true_cc, (b) wrong_cc $\geq$ true_cc, (c) wrong_cc could be either larger than, smaller than or equal to true_cc.

6. (15) Write UCTuplets code for a use-once barrier. Make your code as efficient as possible.

7. (15) In the text and in class, it was mentioned that the butterfly barrier in Section 6.1.4 could be used to implement an all-gather operation. Show the complete MPI code for implementing MPI_AllGather() for in this manner for the case of P_2 on p.166, using MPI_Send() and MPI_Recv(). Show only the code executed by P_2 not the code executed by the other P_i.

Solutions:

1. SDSM or DSM.

2. pvm_mytid().

3.

```
MPI_Comm_rank(MPI_COMM_WORLD,&Me);
...
MPI_Recv(&Number,1,MPI_INT,Me-1,MPI_ANY_TAG,MPI_COMM_WORLD
   &Status);
```

4.

```
if (my_rank == 0)
   for (m=1; m < p; m++)
       MPI_Send(&n,1,MPI_INT,m,N_MSG,MPI_COMM_WORLD);
else MPI_Recv(&n,1,MPI_INT,0,N_MSG,MPI_COMM_WORLD,
   &Status);
```

5. (a)

6. Note that in the code below, we do not use a **while** loop, and the first field in our call to rd() is "si", not "sp".

```
// initialize
if (UCTNodeNum == 0)
   out("si","barrier",0);
...
// barrier action starts here
in("sp","barrier",&Count);
out("si"."barrier",++Count);
if (Count < UCTNWorkNodes)
   rd("si","barrier",UCTNWorkNodes);
```

7. Suppose each node will contribute K **int**s, held in LocalArray, with the result of the gather going into to the 8K-element FullArray.

In the first stage, P_2 must send its K numbers to P_3 and receive K numbers from the latter, putting them in the proper place, which is starting at FullArray[3*K]:

```
MPI_Send(LocalArray,K,MPI_INT,3,AG_MSG,MPI_COMM_WORLD);
MPI_Recv(FullArray+3*K,K,MPI_INT,3,AG_MSG,MPI_COMM_WORLD,
   &Status)
```

In the second stage, P_2 will send 2K numbers to P_0, P_2's own K numbers, plus P_3's K numbers, which P_2 had received during the first stage:

```
MPI_Send(LocalArray,K,MPI_INT,0,AG_MSG,MPI_COMM_WORLD);
MPI_Send(FullArray+3*K,K,MPI_INT,0,AG_MSG,MPI_COMM_WORLD);
```

Then P_2 must receive 2K numbers from P_0, P_0's own K numbers, plus P_1's K numbers, which P_0 had received during the first stage. We will not show the rest of the code here, but it would continue along these lines.

Note that we would need to write the code so that sends and receives do not produce deadlock, say by having lower-numbered partners send first during stages 1 and 3,

and higher-numbered partners sending first during stage 2. We would also have to have the code coordinate correctly; the code for P_0 during stage 2, for instance, would consist of two receives, the first being to FullArray+2*K and the second to FullArray+3*K.