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.

1. (10) Fill in the blank with a Unix command name: The **mpirun** command we've been using with MPICH invokes _____ to get the application program started on other nodes.

2. Consider the Dijkstra program in Section 3.1.1 of the OpenMP PLN.

(a) (10) Which OpenMP pragma line should be deleted in order to improve performance?

(b) (10) Why was the **for** in line 61 (with index variable **step**) not implemented as a parallel **for**?

(c) (10) Which OpenMP pragma line should be modified in order to improve performance? How should it be modified?

3. (15) Consider the pthreads prime-finding program in our inroductory PLN, Section 2.2.2. Supose we wish to improve performance of this program via "chunking," which we'll do by adding 10 instead of 2 to **nextbase** in line 49. Show how to modify the rest of the code accordingly. Your answer should use language like "Change line 8888 to..." "Between lines 3000 and 3001 add..." etc.

4. (10) Suppose we have a NUMA Omega network with 8 PEs. Say we use low-order interleaving, and that consecutive words of memory have consecutive addresses (as opposed to consecutive addresses differing by 4, as in PCs). It allows fetch-and-add operations but with no combining. We have a variable **z** in our program which is in location 164. Suppose PEs 0, 1 and 7 each do a fetch-and-add operation on **z**, with operand 6, and the original value of **z** is 20. There are no other memory transactions in progress at this time. What values do the three nodes get back?[1]

5. Suppose we are reading someone's pthreads code, and see this code

```
#define PADDING (some number here)
...
int tmp,nlocks;
pthread_mutex_t *lockarray;
...
tmp = nlocks * (sizeof(pthread_mutex_t)+PADDING);
lockarray = (pthread_mutex_t *) malloc(tmp);
```

(a) (10) Fill in with a term from our course: The programmer here is trying to avoid _____.

(b) (10) The value of PADDING should be set to _____.

6. (15) Write OpenMP code in the form of a function **fs(x,n)** which does a right-shift of the first **n** elements of the **int** array **x**. Element i gets shifted to i+1, with element 0's new value being 0. You must have an OpenMP **parallel** directive either just before the function or at the beginning of its code. Aim for reasonably good performance, but assume that this function will be used on a variety of platforms.

Solutions:

1. ssh

2.a Line 87, the barrier, is redundant, since the parallel **for** has an implied barrier at the end.

2.b The iterations of the loop are not independent, so they must be done sequentially.

2.c On line 67, we can add a **nowait** clause. This won't harm the correct operation of the program, and it will help reduce the penalty we incur in the critical section, by spreading out the times at which threads enter it.

3. Line 49 becomes

```
nextbase += 10;
```

We need to have each thread check values of **base**, **base+2**, ..., **base+8**. So, place a **for** loop around lines 52-57,

[1] There is an old joke about the newscaster who said, "And here are today's baseball scores: 6-5, 3-0 and 4-3." No team names! In our case here, my point is, don't just give the set of values. State which node receives which value.

```
for (i = 0; i < 10; i += 2)
```

and replace **base** by **base+i**.

4. Call the messages from PEs 0, 1 and 7 A, B and C. A and B route through switching nodes 00, 11 and 22, while C goes through 03, 13 and 22.

A and B clash at 00, with A getting priority. A then clashes with C at 22, with A getting priority again. Then B comes along and beats C too. So A, B and C get the values 20, 26 ande 32, respectively.

5. false sharing; a cache line

6. First of all, to get good performance, e.g. with respect to cache coherency, break the array into blocks, with each thread working on its own block. Great care must be taken, however, at the boundaries of the blocks. When copying from the end of the previous block to the beginning of my block, I must ensure that I am getting the original value, not the copied one. For example:

```
void rs(int *x, int n)
{  int chunk,nth;
   #pragma omp parallel
   {  int me = omp_get_thread_num(),
           i,tmp,mystart;
      #pragma omp single
      {  nth = omp_get_num_threads();
         printf("there are %d threads\n",nth);  }
      chunk = n / nth;
      mystart = me * chunk;
      if (me > 0) tmp = x[mystart-1];
      else tmp = 0;
      #pragma omp barrier
      for (i = mystart+chunk-1; i >= mystart; i--)
         x[i] = x[i-1];
      x[mystart] = tmp;
   }
}
```