Name: _____

Directions: MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

1. (15) Suppose we are running a matrix application in C on a single-core system. We might be doing various operations with a matrix A, such as calculating row and column sums, products of A with vectors (say, both pre- and post-multiplying), and so on. Let n denote the number of rows and columns of the matrix.

Of course, the elements of A are individual variables, and thus they may be subject to *false sharing* problems. Which of the following is true?

(i) With larger values of n, we probably won't have false sharing problems.

(ii) With smaller values of n, we probably won't have false sharing problems.

(iii) The value of n is irrelevant.

(iv) False sharing is not an issue if we have just one core.

2. (30) Consider the Quicksort example using the OpenMP **task** facility, Sec. 4.5.1. As we know, the smaller the granularity in parallel computation, the worse the adverse impact of overhead. So, here we might add another argument to **qs()**, named **k**. If a task is given a chunk (not in the first call) that is of size smaller than **k**, this thread will NOT create new tasks.

In addition to adding the argument **k** to the declaration of **qs()** and to calls to that function, two lines of the original code must be changed. For each one, state the line number and what the new contents of that line will be.

NOTE CAREFULLY: In your electronic file, treat the two changes as part (a) and part (b) of this problem. Also, you may assume that any nonparallel function used earlier in the book is available to you, callable without function itself being in your code.

Sample answer:

```
change line 88 to if (!me) you
```

3. Consider the example on finding the maximal burst in a time series, Sec. 4.14. Suppose we wish to find the maximal sum rather than the maximal mean. So, in line 51 **sum()** will be called instead of **mean()**. (Note: The x_i can be negative.) For convenience, we will in general make minimal changes to the code, for example using the same variable names.

(a) (10) Give the names of any variables that will no longer be needed. If there are none, just write None.

(b) (15) There is actually just one other line that needs to be changed. State which one, and how it should change.

4. (30) The Rdsm package has a barrier facility. Below is the code, with some blanks. Fill them.

```
> barr
function ()
{
    realrdsmlock(brlock)
    count <- barrnumleft[1]
    sense <- barrsense[1]
    if (count == 1) {
        barrnumleft[1] <- blank (a)
        barrsense[1] <- blank (b)
        realrdsmunlock(brlock)
        return()
    }
    else {
        barrnumleft[1] <- barrnumleft[1] - 1
        realrdsmunlock(brlock)
        repeat {
            if (barrsense[1] != sense)
                blank (c)
        }
    }
}
```

Solutions:

1. (iv) With multiple cores, a write to a variable by one core may unnecessarily invalidate other variables in the same cache line at other cores. With a single core, there is no such problem.

2.

```
void swap(int *yi, int *yj)
{  int tmp = *yi;
   *yi = *yj;
   *yj = tmp;
}

int separate(int *x, int low, int high)
{  int i,pivot,last;
   pivot = x[low];  // would be better to take, e.g., median of 1st 3 elts
   swap(x+low,x+high);
   last = low;
   for (i = low; i < high; i++) {
      if (x[i] <= pivot) {
         swap(x+last,x+i);
         last += 1;
      }
   }
   swap(x+last,x+high);
   return last;
}

int cmpints(int *u, int *v)
{  if (*u < *v) return -1;
   if (*u > *v) return 1;
   return 0;
}

void qs(int *z, int zstart, int zend, int firstcall, int k)
{
   #pragma omp parallel
   {  int part;
      if (firstcall == 1) {
         #pragma omp single nowait
         qs(z,0,zend,0,k);
      } else {
         if (zstart + k < zend) {
            part = separate(z,zstart,zend);
            #pragma omp task
            qs(z,zstart,part-1,0,k);
            #pragma omp task
            qs(z,part+1,zend,0,k);
         } else qsort(z+zstart,zend-zstart+1,sizeof(int),cmpints);
      }
   }
}
```

3.a Line 33.

3.b Line 55:

```
xbar = xbar + x[perend];
```

4.

```
> barr
function ()
{
    realrdsmlock(brlock)
    count <- barrnumleft[1]
    sense <- barrsense[1]
    if (count == 1) {
        barrnumleft[1] <- myinfo$nwrkrs
        barrsense[1] <- 1 - barrsense[1]
        realrdsmunlock(brlock)
        return()
    }
    else {
```

```
        barrnumleft[1] <- barrnumleft[1] - 1
        realrdsmunlock(brlock)
        repeat {
            if (barrsense[1] != sense)
                break
        }
    }
}
```