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. Look at the program on pp.74-75.

(a) () Instead of the current form line 16, it could have been

```
if (m[rownum*n+k] == 1) sum++;
```

Fill the blank with a *term from our course*: This would reduce unnecessary multiplications, but would increase ____________________________.

(b) () Using the official terms, state what kind of memory each of the following variables is likely stored in:

variable	mem. type
sum	
(device matrix)	
(device vector)	

2. () In the program on pp.91-92 give the line number at which the number of row pairs (r,s) with s > r is computed for some fixed r.

3. () The program below is an OpenMP version of the mutual outlinks program. However, it uses a different way of apportioning work to threads. Suppose we have a 24x24 matrix, with 2 threads. Then thread 0 will be responsible for rows 0-6 and 17-23, while thread 1 will be responsible for rows 7-12 and 13-16. When I say that a thread is "responsible" for row i, it means that this thread will check all row pairs (i,j), $i < j < n$. The motivation here is that the larger i is, the fewer row pairs there are for that i, and the "mirror image" scheme here (the range 17-23 is obtained from 0-6 by subtracting from 23) is meant to produce better load balancing. Fill in the blanks.

```
#include <omp.h>
#include <stdio.h>

// OpenMP example:  finds mean number of mutual outlinks, among all
// pairs of Web sites in our set

int n,  // number of sites (will assume n is even)
    nth,  // number of threads (will assume n/2 divisible by nth)
    *m, // link matrix
    tot = 0; // grand total of matches

// processes row pairs (i,i+1), (i,i+2), ...
int procpairs(int i)
{  int j,k,sum=0;
   for (j = i+1; j < n; j++) {
      for (k = 0; k < n; k++)
         sum += m[n*i+k] * m[n*j+k];
   }
   return sum;
}

float dowork()
{
   #pragma omp parallel
   {  int pn1,pn2,i;
      int id = omp_get_thread_num();
      nth = omp_get_num_threads();
      int n2 = n / 2;
      int chunk = n2 / nth;
      // in checking (i,j) pairs, j > i, this thread will
      // process i from pn1 to pn2, inclusive, and the
      // "mirror images" of those i
      pn1 =                        // fill blank here
      pn2 = pn1 + chunk - 1;
      int mysum = 0;
      for (i = pn1; i <= pn2; i++) {
                                   // put in 0-4 lines here
      }
                                   // put in 0-4 lines here
   }
   int divisor =                   // fill blank here
   return ((float) tot)/divisor;
}

int main(int argc, char **argv)
{   int n2 = n/2,i,j;
    n = atoi(argv[1]);  // number of matrix rows/cols
    int msize = n * n * sizeof(int);
    m = (int *) malloc(msize);
    // as a test, fill matrix with random 1s and 0s
    for (i = 0; i < n; i++) {
       m[n*i+i] = 0;
       for (j = 0; j < n; j++) {
          if (j != i) m[i*n+j] = rand() % 2;
       }
    }
    if (n < 10) {
       for (i = 0; i < n; i++) {
          for (j = 0; j < n; j++) printf("%d  ",m[n*i+j]);
          printf("\n");
       }
    }
    tot = 0;
    float meanml = dowork();
    printf("mean = %f\n",meanml);
}
```

Solutions:

1. *thread divergence*

2. register, global, global

3.

```
#include <omp.h>
#include <stdio.h>

// OpenMP example:  finds mean number of mutual outlinks, among all
// pairs of Web sites in our set

int n,  // number of sites (will assume n is even)
    nth,  // number of threads (will assume n/2 divisible by nth)
    *m, // link matrix
    tot = 0; // grand total of matches

// processes row pairs (i,i+1), (i,i+2), ...
int procpairs(int i)
{  int j,k,sum=0;
   for (j = i+1; j < n; j++) {
      for (k = 0; k < n; k++)
         sum += m[n*i+k] * m[n*j+k];
   }
   return sum;
}

float dowork()
{
   #pragma omp parallel
   {  int pn1,pn2,i;
      int id = omp_get_thread_num();
      nth = omp_get_num_threads();
      int n2 = n / 2;
      int chunk = n2 / nth;
      // in checking (i,j) pairs, j > i, this thread will
      // process i from pn1 to pn2, inclusive, and the
      // "mirror images" of those i
      pn1 = id * chunk;
      pn2 = pn1 + chunk - 1;
      int mysum = 0;
      for (i = pn1; i <= pn2; i++) {
         mysum += procpairs(i);
         mysum += procpairs(n-1-i);
      }
      #pragma omp atomic
      tot += mysum;
      #pragma omp barrier
   }
   int divisor = n * (n-1) / 2;
   return ((float) tot)/divisor;
}

int main(int argc, char **argv)
{   int n2 = n/2,i,j;
    n = atoi(argv[1]);  // number of matrix rows/cols
    int msize = n * n * sizeof(int);
    m = (int *) malloc(msize);
    // as a test, fill matrix with random 1s and 0s
    for (i = 0; i < n; i++) {
       m[n*i+i] = 0;
       for (j = 0; j < n; j++) {
          if (j != i) m[i*n+j] = rand() % 2;
       }
    }
    if (n < 10) {
       for (i = 0; i < n; i++) {
          for (j = 0; j < n; j++) printf("%d  ",m[n*i+j]);
          printf("\n");
       }
    }
    tot = 0;
    float meanml = dowork();
    printf("mean = %f\n",meanml);
}
```