Name: ____________________________

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

1. (20) Fill in the blank (your answer should have the word *and* in it): According to class discussion, in developing a parallel program, the hardest sections to write are __________________.

2. (20) Suppose we have a symmetric matrix A, written in partitioned form

$$\left(\begin{array}{cc} A_1 & A_2 \\ A_2' & A_3 \end{array} \right) \tag{1}$$

where ' indicates transpose, and m, the number of rows of A_1 is half the number of rows of A. We have a column vector

$$u = \left(\begin{array}{c} u_1 \\ u_2 \end{array} \right) \tag{2}$$

with the number of elements in u_1 being m. We wish to compute the *quadratic form*

$$q = u'Au \tag{3}$$

by exploiting the partitioning (probably in parallel, but not relevant here). Show the algebraically simplified form of q. Note: In your electronic file, write A_1 as A1, and so on.

3. (50) Here we will store many long arrays in one big array. We will store array i in row i of the big array. Anticipating having a great many large arrays, we will use OpenMP to build our big array. For convenience here, assume the number of arrays will be a multiple of the number of threads. Our function is

```
#include <omp.h>

void fillimage(float **arrs, int r, int narr,
      float *a) {
   blank (a)
   {
      int arr; float *arrstart;
      int me = omp_get_thread_num();
      int nth = omp_get_num_threads();
      int block = narr / nth;
      for (arr = blank (b) ) {
         arrstart = blank (c)
         memcpy( blank (d));
      }
   }
}
```

Here **arrs** is the input arrays, each of length **r**, with there being **narr** arrays in all. The big array to be filled is **a**.

Here is a test example:

```
int main() {
   float x[4] = {1,2,3,4}, y[4] = {5,6,7,8};
   float *xy[2] = {x,y};
   float z[8];
   int i;
   fillimage(xy,4,2,z);
   // results should be 1,2,...,8
   for (i = 0; i < 8; i++)
      printf("%f  ",z[i]);
   printf("\n");
}
```

Fill in the blanks.

4. (10) In our NMF tutorial, the approximating matrix can actually turn out to be of rank larger than the targeted value k. Explain why. Remember, you are limited to a single line, though it can be rather long.

Solutions:

1. The start and finish.

2.

$$
\begin{aligned}
u'Au &= && (4)\\
&= (u_1', u_2') \begin{pmatrix} A_1 & A_2 \\ A_2' & A_3 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} && (5)\\
&= (u_1'A_1 + u_2'A_2',\ u_1'A_2 + u_2'A_3) \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} && (6)\\
&= (u_1'A_1u_1 + u_2'A_2'u_1) + (u_1'A_2u_2 + u_2'A_3u_2) && (7)\\
&= u_1'A_1u_1 + 2u_1'A_2u_2 + u_2'A_3u_2 && (8)
\end{aligned}
$$

Note the fact from linear algebra (and our book's review) that $(VW)' = W'V'$.

3.

```
#include <omp.h>

void fillimage(float **arrs, int r, int narr,
      float *a) {
   #pragma omp parallel
   {
      int arr; float *arrstart;
      int me = omp_get_thread_num();
      int nth = omp_get_num_threads();
      int block = narr / nth;
      for (arr = me*block; arr < (me+1)*block;
            arr++) {
         arrstart = arrs[arr];
         memcpy(a+r*arr, arrstart,
            r*sizeof(float));
      }
   }
}
```

4. Since pixel brightness is in [0,1], we truncate values greater than 1. This perturbs some of the data. So, even though we have set things up so that no linear combination of more than k colums of the matrix can be nonzero, that property will be ruined. It doesn't change the effectiveness of the operation, though.

Note by the way that rank(AB) $\leq$ min(rank(a),rank(B), and that W and H have ranks at most k at any iteration, due to number of columns/rows.