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. (25) Consider line 36, p.91 in the mutual outlinks example. Suppose we were to write this program in MPI, and we wanted each node to know the end grand total. Give the MPI call for this. (You'll need an MPI keyword not shown in the book, which is MPI_SUM.)

2. (25) Consider a scan of the set union operation. We'll represent subsets by vectors, so that for instance (0,1,1,1) would represent the subset consisting of objects 2, 3 and 4 of a 4-object full set, while (1,1,1,0,1) would mean the subset consisting of objects 1, 2, 3 and 5 in a 5-object full set. So in general for an n-element full set, each x_i in Equation (7.1) would be an n-component vector. Define $u \otimes v$ in terms of u, v and a function in the parallel matrix algorithms chapter. (No English, just symbols.)

3. (50) The CUDA code below implements the first stage of the parallel scan method in the book, for sums. It is presumed that after the kernel returns, the host will perform the final step, which is to update the individual scans found by the threads to compute the overall scan. The call will be

```
dim3 dimGrid(nblks,1);
dim3 dimBlock(1,1,1);
psum<<<dimGrid,dimBlock>>>(dx,ds,n);
```

Fill the gaps:

```
// places scan of this thread's section of x into
// this thread's section of s; assumes n divisible
// by block size; (not assumed efficient)
__global__ void psum(int *x, int *s, int n)
{  int i, bnum = blockIdx.x,
       bgn =     ;  // index of start of section
       fin =     ;  // index of end of section
                    // fill gap with one statement
                    // fill gap with loop
}
```

Solutions:

1.

```
MPI_Allreduce(&sum,&tot,1,MPI_INT,MPI_SUM,MPI_COMM_WORLD);
```

2. $u \otimes v = b(u + v)$

3.

```
__global__ void psum(int *x, int *s, int n)
{  int i, bnum = blockIdx.x,
       npb = n / gridDim.x,
       bgn = bnum * npb,
       fin = bgn + npb - 1;
   s[bgn] = x[bgn];
   for (i = bgn+1; i <= fin; i++)
      s[i] = s[i-1] + x[i];
}
```