Name: _____

Directions: **Do NOT turn in this sheet of paper (unless you lack a laptop or have a laptop failure during the Exam). You will submit electronic files to handin.**

1. (70) A method for compressing data is to store only repeat counts in *runs*, where the latter means a set of consecutive, identical values. For instance, the sequence 2,2,2,0,0,5,0,0 would be compressed to 3,2,2,0,1,5,2,0, meaning that the data consist of first three 2s, then two 0s, then one 5, and finally two 0s. Note that the compressed version consists of alternating *run counts* and *run values*, respectively 2 and 0 at the end of the above example.

Here you will write a function to uncompress an array that uses this type of storage, in both OpenMP and Snow, submitting files named **uncomprle.c** and **uncomprle.R**, respectively.

In both cases, you are required to first call a parallel sum scan, using the functions in **prfx.c** from the (top level of) our class Web site or in **ParSum.R** in the **Extras** directory in our Web site. You will use the output of that operation to decide where to place the runs in your overall output. You will then do further parallel processing.

You may assume that the number of threads divides the number of run counts, i.e. half of 4 in the above example.

The signature for a C version must be

```
void uncomprle(int *x,int nx,int *tmp,
   int *y,int *ny)
```

where **x** is the input array, of length **nx**, **y** is the output array, and **tmp** is an array for temporary storage to be explained shortly. It is assumed for convenience that space has been preallocated for **y** and **tmp** before the call. The function fills in **y** and also sets ***ny**.

The test code I will use for OpenMP is

```
// test code
int main()
{  int i;
   int x[12] = {2,3,1,9,3,5,2,6,2,88,1,12};
   int nx = 12;
   int tmp[100],y[100];
   int ny;
   uncomprle(x,nx,tmp,y,&ny);
   // y should be {3,3,9,5,5,5,6,6,88,88,12};
   for (i = 0; i < ny; i++) printf("%d\n",y[i]);
}
```

In the case of R, your function will have two parameters: **cls**, the cluster name, and **x**, the input vector. The function will return **y**. My test call will be

```
uncomprle(c(2,3,1,9,3,5,2,6,2,88,1,12))
```

2. (30) Give code to do the parallel scan operation in Problem 1 using Thrust. (Use only Thrust functions from our course materials.) You will fill the blanks:

```
// possibly put material here

int main()
{  int i;
   int x[12] = {2,3,1,9,3,5,2,6,2,88,1,12};
   int nx = 12;
   int tmp[100];
   // compute and print out tmp here
   // (should be 2,3,6,...)
}
```

Turn in a file **findtmp.cu** (for compiling with gcc for OpenMP, use **.c** suffix). **IT SHOULD BE THE ENTIRE CODE, INCLUDING main() ABOVE, COMPILABLE AND RUNNABLE.**

Solutions:

1. OpenMP code:

```
1  void uncomprle(int *x,int nx,int *tmp,int *y,int *ny)
2  {
3     int i,nx2 = nx/2;
4     int z[MAXTHREADS];
5     for (i = 0; i < nx2; i++) tmp[i+1] = x[2*i];
6     parprfsum(tmp+1,nx2+1,z);
7     tmp[0] = 0;
8     #pragma omp parallel
9     {  int j,k;
10       int me=omp_get_thread_num();
11       #pragma omp for
12       for (j = 0; j < nx2; j++) {
13          // where to start the j-th run?
14          int start = tmp[j];
15          // what value is in the run?
16          int val = x[2*j+1];
17          // how long is the run?
18          int nrun = x[2*j];
19          for (k = 0; k < nrun; k++)
20             y[start+k] = val;
21       }
22    }
23    *ny = tmp[nx2];
24 }
```

2.

```
1  #include <stdio.h>
2  #include <thrust/host_vector.h>
3  #include <thrust/scan.h>
4  #include <thrust/sequence.h>
5  #include <thrust/remove.h>
6
7  struct iseven {
8     bool operator()(const int i)
9     {  return (i % 2) == 0;
10    }
11 };
12
13 int main()
14 {  int i;
15    int x[12] = {2,3,1,9,3,5,2,6,2,88,1,12};
16    int nx = 12;
17    int tmp[100];
18    int seq[nx];
19    thrust::host_vector<int> out(nx);
20    thrust::sequence(seq,seq+nx,0);
21    thrust::host_vector<int> hx(x,x+nx);
22    thrust::host_vector<int>::iterator newend =
23       thrust::copy_if(hx.begin(),hx.end(),seq,out.begin(),iseven());
24    thrust::inclusive_scan(out.begin(),out.end(),out.begin());
25    // compute and print out tmp here
26    // (should be 2,3,6,...)
27    for (int i = 0; i < newend-out.begin(); i++) printf("%d\n",out[i]);
28 }
```