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. (20) Suppose we are writing MPI code for the hyperquicksort pseudocode on p.5 of our PLN on sorting. In this problem, implement only the two lines that begin with “low-numbered” and “with high-numbered.”

Assume you have already set these variables: **d**, the dimension of the cube; **me**, this node’s ID number; **mydata**, the data at this node, an array of **int**s); **nmydata** is the number of elements in **mydata**; and **pivot**, the pivot element.

Take “smaller” in the pseudocode to me “smaller than or equal to.”

You must use **MPI_Sendrecv()**. To see how its arguments work, see the function **OddEvenSort()** in the TA’s handouts.

Make sure your handwriting is legible. Write your code on scratch paper first, then copy to this sheet.

2. (20) Write an MPI function to do low-pass filtering via cosine transform, with prototype

```
void lowpass(int *x, int n, int *c, float prop)
```

Here we have a one-dimensional data set **x** of length **n**. The array **c** is for temporary storage of the spectrum of **x**. We calculate all but the **r = floor(prop*n)** highest-frequency components, then transform back to the time domain, overwriting **x**. The time-domain data are originally in **x** at node 0, and will again be placed in **x** at node 0 when we finish. Assume that **n-r** is evenly divisible by **p**, the number of nodes; call the quotient **s**.

Do *not* use parallel matrix methods. Instead, each node will calculate a chunk of size **s** in **c**, and then each node will calculate a chunk of size **s** in the new **x**. Make sure to make good use of MPI’s collective operations.

Here are the formulas for the one-dimensional cosine transform and its inverse:

$$c_k = \sum_{j=0}^{n-1} x_j \; cos \left[\frac{\pi}{n} (j + 0.5) k \right] \tag{1}$$

$$x_k = 0.5 c_0 + \sum_{j=1}^{n-1} c_j \; cos \left[\frac{\pi}{n} j (k + 0.5) \right] \tag{2}$$

Make sure your handwriting is legible. Write your code on scratch paper first, then copy to this sheet.

3. This problem concerns the function **SampleSort()** in the TA’s handouts.

(a) (10) In our PLN presentation on this type of parallel sorting, we spoke in terms of deciles, i.e. the 10r percentiles, r = 1,...,10. The code here deals with cr percentiles. State the value of c in terms of variables in the program.

(b) (15) Fill in the blanks: After the call to **MPI_Gather()**, the cr percentile found by node j will be stored in **allpicks[**_____________**]** at node _____________.

4. This problem concerns the Apriori algorithm for data mining.

In the notation of our PLN, **F[i]**, the list of frequent subsets of size **i**, is a two-dimensional array whose jth row lists the members of the jth subset of size i. The number of rows in that array is **NF[i]**. For instance, if the third member of F_3 is $\{$5,12,13$\}$, then we have **F[3][2][0]** = 5, **F[3][2][1]** = 12 and **F[3][2][2]** = 13.

Suppose also that our code which calculated the F_i placed additional information at the end of the itemset’s row. First, it tacked on the count of that itemset, at position i, so that if for instance $\{$5,12,13$\}$ has count 292, then **F[3][2][3]** would be 292. Second, starting with position i+1, it placed pointers to the rows of the frequent itemsets which are subsets of this one, and their sizes. For example, there could be a pointer to the row in F for $\{$5,12,13,16,22$\}$, followed by 5. After the last pointer/size pair, it placed a 0.

(a) (25) Write OpenMP code which uses the F_i (which are already computed using the algorithm in the PLN) to find the number of association rules $I \rightarrow J$ that meet or exceed both the support and confidence thresholds.[1] The confidence threshold proportion is in the variable **confthresh**.

Keep your code simple, not worrying too much about maximizing performance. **Make sure your handwriting is legible. Write your code on scratch paper first, then copy to this sheet.**

[1]Normally we would record the itemsets themselves, but for simplicity’s sake we will just output the count.

(b) (10) Suppose we are doing an OpenMP parallel version of the Apriori algorithm. Since F_1 never changes after it is first set, we may wish to copy it to a separate **private** variable, **LocalF1** so as to reduce interconnect traffic. Why is this not really necessary?

Solutions:

1. Basic idea:

- node 0 broadcasts **x** to all nodes
- each node calculates a chunk of the nonzero portion of **c**
- everyone does an all-gather operation so that all nodes have all of the nonzero portion of **c**
- each node calculates its chunk of **x** (don't throw out any)
- then gather to node 0

```
s = (n-r)/p;  // chunk size for calculating c
MPI_Comm_rank(MPI_COMM_WORLD,&me);
MPI_Bcast(x,n,MPI_FLOAT,0,MPI_COMM_WORLD);
for (k = me*s; k < (me+1)*s; k++)  {
   tot = 0.0;
   for (j = 0; j < n; j++)  // note n
      tot += x[j]*cos((pi/n)*(j+0.5)*k);
   myc[k-me*s] = tot;
MPI_Allgather(myc,s,MPI_FLOAT,c,s,MPI_FLOAT,MPI_COMM_WORLD);
q = n/p;  // chunk size for calculating x
for (k = me*q; k < (me+1)*q; k++)  {
   tot = 0.5 * c[0];
   for (j = 1; j < n-r; j++)  // note n-r
      tot += c[j]*cos((pi/n)*(k+0.5)*j);
   myx[k-me*q] = tot;
}
MPI_Gather(myx,q,MPI_FLOAT,x,q,MPI_FLOAT,0,MPI_COMM_WORLD);
```

2.a $\frac{100}{p} \cdot r$

2.b allpicks[(p-1)*j+r], 0

3.a Basic idea:

```
nconf = 0
for each itemset size i
   // start parallel for
   for each potential I (as in an association I => J) of size i
      count = 0
      for each possible J which may be associated with this I
         if confidence higher than threshold
            count++
   atomic increment of nconf by count

nconf = 0;
#pragma omp parallel private(count)
for (i = 0; i < t-1; i+1)  {  // note t
   #pragma omp for
   for (j = 0; j < NF[i]; j++)  {  // note NF[i]
      k = i+1;
      count = 0;
      while (F[i][j][k] != 0)  {
         Jsetsize = F[i][j][k+1];
         Jsupport = F[i][j][k][Jsetsize];
         if (float(Jsupport)/F[i][j][i] > confthresh
            count++;
         k += 2;
      }
      #pragma omp atomic
      nconf += count;
   }
}
```

3.b F_1 would be almost immediately cached, and since it never changes, the cached copy remains intact and useable efficiently.