CSE415: Introduction to Parallel Computing
Fall 2017, Homework 4
Deadline: April 2nd, Monday 11:59 p.m
*This homework is 15% of your **homework grade**.*

Important note: *Please use a word processing software (e.g., MS Word, Mac Pages, Latex, etc.) to type your homework. Follow the submission instructions at the end to turn in an electronic copy of your work.*

1) [60 pts] Leading kids in summer camp

You have joined a summer camp for kids as a camp leader. You need to explain some rules to the kids before starting the camp, so you need to arrange them in a line. The issue is not all the kids have same height. You have to make them stand in a line where the kids with less height stands in front and the taller kids stand behind so that everyone can see you. Given the heights of the kids, you need to arrange them in such a line. Imagine that the summer camp is in a place where the heights can be of any value.

Here is an example –
Before arranging

3	5	2	8	1	10	3

After arranging

1	2	3	3	5	8	10

Here is a code snippet for you to arrange the kids according to their heights -

```
void kids_line(int heights[], int kids_count) {
  int i, j, count;
  int *temp = malloc(kids_count*sizeof(int));

  for (i = 0; i < kids_count; i++) {
    count = 0;
    for (j = 0; j < kids_count; j++)
      if (heights[j] < heights[i])
        count++;
      else if (heights[j] == heights[i] && j < i)
        count++;
    temp[count] = heights[i];
  }
  memcpy(heights, temp, kids_count*sizeof(int));
  free(temp);
}
```

The basic idea is that for each element `heights[i]`, we count the number of kids in `heights` that are less than `heights[i]`. Then we insert `heights[i]` into the `temp` list using the subscript determined by `count`. There's a slight problem with this approach when the twi kids have same height, since they could get assigned to the same slot in `temp`. The code deals with this by incrementing `count` for equal

elements on the basis of the subscripts. If both `heights[i] == heights[j]` and `j < i`, then we count `heights[j]` as being "less than" `heights[i]`. After the algorithm has completed, we overwrite the original array with the temporary array using the string library function `memcpy`. A serial implementation is provided on D2L (kids_line_skeleton.c) that should be used for validation and speedup.

i) [20 pts] Implement two different OpenMP parallel versions of the count sort algorithm. First version should parallelizing `i`-loop, and second version should parallelize the `j`-loop. *In both versions, modify the code so that the* `memcpy` *part can also be parallelized.*

ii) [20 pts] On the **intel16** cluster, compare the running time of your OpenMP implementations using 1, 2, 4, 8, 14, 20 and 28 threads. Plot total running time vs. number threads (include curves for both the i-loop and j-loop versions in a single plot) for a fixed value of *n*. *A good choice for n would be in the range 50,000 to 100,000*.

iii) [15 pts] Try with different kinds of scheduling (eg – static, Dynamic) varying the chunk size and see if you see any changes in performances with respect to the default scheduling scheme.

iv) [5 pts] How does your best performing implementation (`i`-loop vs. `j`-loop) compare to the serial `qsort` library function's performance in C? Why? Explain your observations.

2) [30 pts] Iteration Space Dependency Graph(ISDG)

Make an ISDG for the following code snippet –

```
for(int i = 1; i <= M ; i++ )
   for(int j = 1; j <=N ; j++ )
         data[i][j] = data[i][j-1] + data[i-1][j-1] // imagine column 0 and row 0 have all zero entries
```

Is it possible to parallelize this loop? If so, show the updated code.

3) [25 pts] MPI split

Suppose you have 16 processors in a 2-d process grid with dimension 4*4. The processors are numbered though 0,1,2…..15. Split them in new communication where each new communicator will contain all the processors in a row. For example look at this example –

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	1	2	3

0	1	2	3
0	1	2	3
0	1	2	3

Give the necessary commands needed to split the processor grid in this way. Give appropriate color and key values needed for this split.

Instructions:

- **Accessing the git repo.** You can pull the skeleton codes from the git repo. Please refer to *"Homework Instructions"* under the *"Reference Material"* section on D2L.

- **Submission.** Your submission will include:
  - A pdf file named "HW4_*yourMSUNetID*.pdf", which contains your answers to the non-implementation questions, and report and interpretation of performance results. *If your assignment is written using a document processing software such as word, please export the document to PDF format for your homework submission: do not submit the *.doc file.*
  - All source files used to generate the reported performance results. Make sure to use the exact files names listed below:
    - countsort_paralleli.c
    - countsort_parallelj.c
    - countsort_paralleli_dynamic.c
    - countsort_parallelj_dynamic.c
    - countsort_paralleli_static.c
    - countsort_parallelj_static.c

    These are the default names in the git repository – essentially, do not change the directory structure, file names, or form of the function declarations.

  To submit your work, please follow the directions given in the "Homework Instructions" under the "Reference Material" section on D2L. Make sure to strictly follow these instructions; otherwise you may not receive proper credit.

- **Discussions.** For your questions about the homework, please use the Slack group for the class so that answers by the TA, myself (or one of your classmates) can be seen by others.

- **Compilation and execution.** You can use any compiler with OpenMP support. The default compiler environment at HPCC is GNU, and you need to use the –fopenmp flag to compile OpenMP programs properly. Remember to set the OMP_NUM_THREADS environment variable, when necessary.

- **Measuring your execution time properly.** The omp_get_wtime() command will allow you to measure the timing for a particular part of your program (see the skeleton codes). *Make sure to collect at least 3 measurements and take their averages while reporting a performance data point.*

- **Executing your jobs.** However, on the dev-nodes there will be several other programs running simultaneously, and your measurements will not be accurate. After you make sure that your program is bug-free and executes correctly on the dev-nodes, the way to get good performance data for different programs and various input sizes is to use the interactive or batch execution modes. *Note that jobs may wait in the queue to be executed for a few hours on a busy day, thus plan accordingly and do not wait for the last day of the assignment.*

  i) **Interactive queue.** Suggested options for starting an interactive queue on the **intel16** cluster is as follows:

  ```
  qsub -I -l nodes=1:ppn=28,walltime=00:30:00,feature=intel16 -N myjob
  ```

  The options above will allow exclusive access to a node for 30 minutes. If you ask for a long job, your job may get delayed. Note that default memory per job is 750 MBs, which should be

plenty for the problems in this assignment. But if you will need more memory, you need to specify it in the job script.

ii) **Batch job script.** Sometimes getting access to a node interactively may take very long. In that case, we recommend you to create a job script with the above options, and submit it to the queue (this may still take a couple hours, but at least you do not have to sit in front of the computer). Note that you can execute several runs of your programs with different input values in the same job script – this way you can avoid submitting and tracking several jobs.