**Parallel Computing COP6616**

**Fall 2024**

**Instructor Scott Piersall  
Homework 2  
Due: 10/15/2024, 1:00 pm**

1. **(25 points)** **Please implement an MPI program in C, C++, OR Python to calculate the multiplication of a matrix and a vector (Using MPI Scatter and Gather).** Specifically, the MPI program can be implemented in the following way:

1.Implement a Serial code solution to compute the multiplication of a matrix and a vector. This is the basis for your computation of Speedup provided by parallelization. (You should lookup the definition of Speedup in parallel computing carefully)  
2. According to the input argument (the size of the vector) from main() function, generate a matrix and a vector with random integer values, where the column size of matrix should be equal to the size of the vector.

3. **SCATTER**: According to the number of processes from the input argument, split the matrix into chunks (row-wise) with roughly equal size, then distribute chunks to all processes using “scatter”. Additionally, the vector can be broadcasted to all processes.   
4. Conduct product for the chunk of matrix and vector.

5. **GATHER**: The final result is collected on the master node using “gather”.

6. **VERIFY CORRECTNESS**: Make sure the result of your MPI code matches the results of your serial code.

7. **SPEEDUP**: What is the speedup S of your approach? (Speedup is a specific measurement and you should report it correctly) Combine the answer to this with your experiments in the next step

8. **EXPERIMENTS**: You should run experiments consisting of running problem set sizes over varying number of compute nodes. Discuss the relationship between increasing the number of nodes to Speedup S in your submitted PDF

Note:

* You should develop your code on the MPI cluster hpclab.unfcsd.unf.edu.
* I have placed a PowerPoint Presentation into the files section in canvas which contains a walkthrough and connection information for the compute cluster
* You can login to this cluster using SSH and your UNF N#
* For assistance with setup/compilation, you may ask members of the MPI Introduction Chapter Team in Discord

**2. (4 points)** For each of the following code segments, use OpenMP pragmas to make the loop parallel, or explain why the code segment is not suitable for parallel execution. You do not need to write executable C programs, please just add valid openMP pragmas. The code can be modified with keeping the same semantics for adding openMP pragmas.  
  
**(1)**for (i=0;i<(int)sqrt(x);i++){

    a[i] = 2.3 \* i;

    if (i<10) b[i] = a[i];

}

**(2)**

flag = 0;

for (i=0; (i<n) && (!flag); i++){

   a[i] = 2.3\*i;

   if (a[i] < b[i]) flag = 1;

}

**(3)**

for (i=0; i<n; i++)

   a[i] = foo(i);

**(4)**

for (i=0; i<n; i++){

    a[i] = foo(i);

    if (a[i] < b[i]) a[i] = b[i];

}

**(5)**

for (i=0; i<n; i++){

    a[i] = foo(i);

    if (a[i] < b[i]) break;

}

**(6)**

p = 0;

for (i=0; i<n; i++){

    p += a[i] \* b[i];

}

**(7)**

for (i=k; i<2\*k; i++){

    a[i] = a[i] + a[i-k];

}

**(8)**

for (i=k; i<n; i++){

    a[i] = b \* a[i-k];

}

## 3. (35 points) Computing Euclidean Distance Using OpenMP or MPI

In N-dimensional space, the Euclidean distance between two points P(p1, p2, p3,…, pn) and Q(q1, q2, q3,…, qn) is calculated as follows,

Please write **an OpenMP or MPI program** to compute the Euclidean distance between two *N*-dimensional points, where ***N is at least 1 million.*** Please initialize your vectors (denotes the two points) with random integers within a small range (0 to 99). This computation consists of a fully data-parallel phase (computing the square of the difference of the components for each dimension), a reduction (adding together all of these squares), and finally taking the square root of the reduced sum.

Note: You may want to use gcc/g++ directly, since gcc/g++ already supports OpenMP. To use it, you can compile your code with a special flag “-fopenmp”.

**4. (35 points)** **Find prime numbers using MPI.** Develop an MPI program to find and print the first n prime numbers in the range [0…n]. Run your code using 2,4,8, and 16 nodes. The MPI communication methods you choose are up to you, however….you MUST distribute the work evenly among the nodes, not perform redundant or unnecessary calculations AND you should automatically skip even numbers. For your experiment, use the value of 2500000 for n. Your program should print out each prime number found (and the node that found it), and the master node (rank 0) should output the time elapsed for your program to run. Run the experiment for 2,4,8, and 16 nodes. Graph your results. Include the graph and a paragraph interpreting and explaining your experimental results in the PDF file report. Include your source code in the ZIP file for turn in.

Hints: There are a variety of ways to implement a solution to question 4. My hints are

* Do not assign contiguous blocks of numbers to each node. That does NOT distribute the work evenly among the nodes. Think about this hint ***carefully***. If you do not understand why assigning a contiguous block of numbers to each node is incorrect for this problem, please come see me to discuss….seriously. Solutions submitted which just assign contiguous blocks of numbers to nodes will receive zero credit
* MPI\_Reduce is your friend for this problem……..

**What to submit:**

1. For all code assignments, please write a README file explaining how to run and test your code. In your code, please add appropriate comments and make your code structured.
2. PDF File: You should create a report and convert its contents to PDF format. The report should answer the following questions:
   1. Answers to the Speedup portion of Question 1
   2. Answers to the 8 problems in Question 2
3. Zip all code and PDF files into a single file to upload.

**NOTE: Because you are all sharing the compute nodes and cluster resources, each experiment should be run multiple times, and your runtimes averaged over each run for varying node sizes. This should reduce any large variances in your experimental results resulting from resource contention.**