Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

MSc. in HPC

Introduction to MPI (55611)

Programming Exercises 1

Darach Golden

November 17, 2023

1 Notes

- **To complete these exercises, submit a document containing results for each question and a separate tarball of the all code used to obtain the results. Details of how to compile and run the code must be provided in a "README" file. You *must* use a makefile to compile the code. The code must compile on `seagull`**
- **Use C to implement all exercises**

2 Exercises

1. Read vector from file.

   Write code to:

   - read a vector from file on rank 0. The file format should be:
     - length of the vector on the first line
     - entries of the vector, separated by spaces or carriage returns, on the remaining line or lines
   - Evenly divide the vector among the processes (you can assume that the number of processors divides the length of the vector). On each processor you should only allocate enough memory for the local subset of the vector assigned to that processor
   - Add 1.0 to each element of the local vector on each processor
   - gather the local vectors back into a vector on rank 0 and write this vector to a file of the same format as the input file
   - Run this code on 4 processors and for a vector of length $n = 16$. Run the code on at least 10 processors for a vector length at $n = 10000$
   - Provide two implementations of this code: one which uses `MPI_Send` / `MPI_Recv` and a second which uses MPI collective calls

2. One dimensional decomposition of a list of items of length `n` over `p` processors

   Write a function to decompose an array of doubles of size `n` across `p` processors. The code should return for each process the local starting point into the global array and the local end point into the global array. You cannot assume that `p` divides into `n` without remainder. The interface in C must be

   `int decomp1d(int n, int p, int myid, int *s, int *e)` ,

   where `myid` is the rank of the processor; `n`, `p`, `myid` are inputs and `s` and `e` are outputs. The return value should be 0 on success.

   For example on rank 0, the output should be: $s = 1,\ e = e_0$. On rank 1, the result should be: $s = e_0 + 1,\ e = e_1$. On rank $p - 1$ the result should be: $s = e_{p-2} + 1, e = n$.

- Ensure that the decomposition is *load balanced* – numbers of elements assigned to different processes should differ by at most one
- Note that this function does not need to contain any MPI code – it just returns values for `s` and `e` based on values of `n,p,myid`
- Use your function in a basic MPI code which calls the decomposition function and prints out the results for each rank. Demonstrate that the decomposition produces consistent results on up to 9 processors for $n = 25$. Do the same on 10 processors for $n = 100000$. You do not need to allocate any memory for the vector, just print the output of the function
- Demonstrate that the function also produces correct results when run on *one* processor

3. Read a vector from file and calculate the square of the Euclidean (l_2) norm of the vector ($||x||_2^2$ = the sum of squared entries of the vector)

   Using a similar file format used in Q1 (see the attached files `q3file_16.txt` and `q3file_97.txt`)

   - read a vector from file on rank 0
   - distribute the vector among p processors. You *cannot* assume the number of processors divides the length of the vector without remainder. Use the function from Q2 or otherwise to divide the vector among the processors in a load balanced way. Different processors may have different local vector sizes
   - on each processor calculate $||x_{local}||_2^2$
   - calculate the full sum of squares by combining all local results onto rank 2 and print the answer
   - calculate the full sum of squares by combining all local results onto all processes and print the answer
   - Using the files `q3file_16.txt` and `q3file_97.txt`, demonstrate that your answers obtained using the parallel code are correct when run on:
     - `q3file_16.txt` for 4 processors
     - `q3file_16.txt` for 9 processors
     - `q3file_16.txt` for 1 processor
     - `q3file_97.txt` for 11 processors
     - `q3file_97.txt` for 6 processors