Problem 1: Sort Integers Saved in a File

Implement an MPI program that sorts a set of 4-byte integer values in ***ascending order***. The values are saved in a file. When the program finishes, the sorted values should be saved in the same file. The values are saved in binary format. Thus, library functions such as `fread()` and `fwrite()` are used to read and write the file. File pathname can be provided to the program via user input (`scanf` ()).

Use count sort. You can assume that the smallest value is 0 and the largest value is 999. So it is safe to set counter array size to 1000. Use the similar methods to split the computation and maximize parallelism (local counter array) as what was used in the pthread version, which has been introduced in the class and the textbook.

To test your program, you can compile and use gendatai.c and checkdatai.c attached with this assignment. You can run gendatai to generate the file containing random integer values. Then, you run your program to sort the values in the file. After that, you run checkdatai to check whether the values have been sorted in ascending order.

Gendatai needs you to provide the number of values and the file pathname in the command line. Checkdatai only needs the file pathname. For example, to generate 5000 random values and save them into ./file5kvalues, you can use the following command

./gendatai 5000 ./file5kvalues

To check whether the values are sorted in the file, you run command

./checkdatai ./file5kvalues.

Both tools report the sum of the values in the file. You need to compare the sums, in case your program misses some values or includes extra values.

You can assume that all integers in the file can fit into memory. So you can malloc a buffer and read all integers into the buffer before doing countsort.

You can assume that the number of integers is divisible by the number of processes used to do contsort. So you don't need to consider the remainders when you try to evenly distribute the data and workload.

Problem 2: Calculating the approximation of π

Write a pthread program to calculate the approximation of π. The method of calculating of the approximation was introduced in the class in the MPI part. You need to implement this method using pthreads. The number of threads and the number of terms required to calculate the approximation should be specified in the command line.

To test your program, run it with different number of terms and different number of threads. You should be able to see that the approximation is closer to the real value of π when you increase the number of terms, and the execution time reduces when you increase the number of threads. Note that the server only has 4 CPU cores. Using more than 4 threads cannot increase speed. Your program usually finishes very quickly when the number of terms is not large. It may take a while only when the number of terms is large enough (e.g., greater than 1 billion).