Prof. Jingke Li (lij@pdx.edu); Classes/Labs: TR 1400-1550 @FAB 150, 88-10; Office Hours: TR 1300-1400 @FAB 120-06

Assignment 4: Programming with MPI (Due Thursday, 6/2/22)

This last assignment is to practice programming with file-IO and with message-passing. You are to implement several versions of the bucket sort.

1 A Sequential Program with File IO (`bsort-file.c`)

The file `bsort.c` contains a sequential version of the bucket sort program. It has the following user interface:

```
linux> ./bsort B [N]   -- use B buckets to sort N numbers (B must be a power of 2)
```

The "power of 2" assumption on `B` is to enable easy bucket distribution based on leading bits of each number.

Read and understand the program.

- Note that the buckets are declared with a size `2*N/B`. This is to safe guard the perturbations in the data distribution, since on average, there should be `N/B` numbers in each bucket.
- Also pay attention to the macro definition for deciding which bucket a number should be placed in:

  ```
  // bucket index for integer x of b bits (B is #buckets)
  #define BktIdx(x,b,B) ((x) >> ((b) - (int)log2(B)))
  ```

  For this program, data are 13-bit integers (i.e., with a value range [0, 8191]). If we use 4 buckets, then the leading 2 bits are used as the bucket index. For example, for the number 4734 (= 1001001111110_2), the leading 2 bits are 10_2, so it is to be placed in `bucket[2]`.

Your Task Your first task is to write a new version, `bsort-file.c`, which reads input data from a file, and writes sorted result to another file. The new program's user interface is:

```
linux> ./bsort-file B <infile> <outfile>  (B must be a power of 2)
-- use B buckets to sort data in <infile> and write result to <outfile>
```

The data size `N` is to be derived from the input file size, which can be obtained by the `fseek` and `ftell` routines:

```
fseek(fin, 0L, SEEK_END);          // go to the end of file
int size = ftell(fin);             // this gives the file size
rewind(fin);                       // reset the file pointer
```

Note that you need to reset the file pointer after `fseek`, so that you can read the data from the start of the file. Use `fread` and `fwrite` routines to read and write the files. Look up on the Internet for usage examples if you are not familiar with them.

2 An MPI Version (`bsort-mpi.c`)

Your second task is to write an MPI version of the bucket sort program, also with file IO, `bsort-mpi.c`. The program has the following user interface:

```
linux> mpirun -n P bsort-mpi <infile> <outfile>  (P must be a power of 2)
-- use P buckets to sort data in <infile> and write result to <outfile>
```

The program should follow the standard SPMD-style, *i.e.* writing a single program, which is to be replicated over the participating processes. The program consists of the following steps:

1. Initialize MPI, and get the parameters, `P` and `rank`. Verify that `P` is a power of 2. (If not, prompt the user and terminate.)

2. Open the input file and get the file size. Derive the data size (i.e. parameter `N`) from the file size, and verify that `P` evenly divides `N`. (If not, prompt the user and terminate.)
3. Declare a data array of size `N/P`. Compute an unique file-view offset based on `rank`, and read `N/P` integers from the input file into the data array.
4. Declare an array of `P` buckets of size `max(2*(N/P)/P, 4)`. The first term is of the same idea of safe-guarding distribution perturbations as in the sequential version (since the average bucket size is `(N/P)/P`), and the value 4 is to guarantee a minimum size to cover some corner cases, since `(N/P)/P` can be 0 when `N` is small.
5. Distribute the data in data array to the buckets. (*Hint:* You may copy corresponding code from `bsort.c`.)
6. (Communication) Send `bucket[k]` to process `k`. Since the buckets are of different sizes, use a sequence of individual `send/recv` routines to implement this. (It is easier than using collective routines.) You need to be careful in organizing your code to avoid deadlock, since every process is both a sender and a receiver.
7. After receiving `p-1` buckets from other processes, merge them with own `bucket[rank]` into a buffer (of size `2*N/P`), and sort them with bubble sort.
8. Figure out a way to collectively compute all processes' file-view offsets for writing to the output file. (*Hint:* A scan operation over bucket size should help.)
9. Open the output file and write the data in sorted buffer to the proper location in the file. Note that you should select both the `CREATE` and the `WRONLY` modes when opening the output file. This would allow the file be created if it's not already there. (Note that if the file already exists, new data will be written over the old data, but only for the section the new data covers, i.e. if there is more old data, part of that will remain.)

Note that the program should not allocate more than $O($`N/P`$)$ amount of memory, and both file input and output should use the parallel collective routines.

3 A Second MPI Version* (`bsort-mpi2.c`)

[* For CS415 students, this part is optional; if you correctly implement it, you will earn an extra 15% of points.]

This version is the same as `bsort-mpi.c` except that in the Communication step (Step 6), the buckets are sent using the v-version of the collective routines, instead of a sequence of individual `send/recv`s.

Additional Information

- The program `datagen.c` can be used to generate new input data. It takes an integer argument, `N`, and generates `N` random integers with value in the range [0, 8191].

```
linux> ./datagen 1024 > data1k
```

- The program `verify.c` can be used to verify that the data in a file are sorted in an ascending order:

```
linux> ./verify out16
Data in 'out16' are sorted (total 16 items).
```

- To see the content of an input or output file, use the Linux utility, `od`:

```
linux> od -i in32   -- display binary content of file in32 as integers
```

Summary and Submission

Write a summary covering (1) status of your programs; and (2) your experience with this assignment. Make a zip file containing all your programs and your write-up. Submit it through the "Assignment 4" folder on Canvas.