Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Classes: M 16:40-18:30, W 16:40-17:20, Labs: W 17:25-18:55, all @ FAB 88-10.

Lab 8: Parallel Sorting

1 Odd-Even Sort

The file `oddevenSort.c` contains a sequential odd-even sort program. Compile this program with the `DEBUG` flag, and run it several times. Do you observe the array being sorted before all iterations are executed?

1. Write a new version of this program, `oddevenSort1.c`, to include termination detection. The new program should terminate as soon as the array is sorted. Compile and run it to verify.
2. Write a second new program, `oddevenSort2.c`. It's similar to `oddevenSort.c` except that it reads data from a file and writes result to another file. Both the input and output files contain byte-coded integers. The following is the program's interface:

   ```
   linux> ./oddevenSort2 <infile> <outfile>
   ```

   The array size `N` is to be derived from the input file size, which can be obtained:

   ```
   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
   ```

   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.
3. Write a third new program, `oddevenSort3.c`. This time, the program is a MPI program. It uses only two processes, rank 0 and rank 1, each handles half of the array. To make things simpler, you could base this program on the original version, *i.e.* without early termination and file I/O. So the initial data is generated locally by each process, and the final result is printed out separately as well. Use send and receive to facilitate necessary data exchanges.
4. (Optional) Further improve the above MPI program to include early termination, file I/O, and/or unlimited number of processes. (You may want to complete Part 2 (below) first, then come back.)

Input Data The provided 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
```

2 Bucket Sort

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

```
linux> ./bucketSort N B   -- use B buckets to sort N numbers (B's value is a power of 2)
```

The assumption on `B`'s value is to enable bucket distribution to be based on leading bits of each number.

1. Read and understand the program. Pay special attention to the routine for deciding which bucket an array element should be placed in:

   ```
   // find bucket idx for an int value x of b bits
   int bktidx(int x, int b, int B) {
     return x >> (b - (int)log2(B));
   }
   ```

In the program, the data value range is set to 13 bits, or [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, so it is to be placed in `bucket[2]`.

Compile and run this program with different parameter combinations.

2. Now write a simplified MPI version of bucket sort, `bucketSort2.c`, with the following user interface:

```
linux> mpirun -n P bucketSort2 N <outfile>   -- use P buckets to sort N numbers
```

Parameter `P`'s value is to be used as the number of buckets, so it is assumed to be a power of 2. The program works as follows:

   (a) Process 0 runs the original program (with `P` serving the role of `B`) up until the buckets are generated.

   (b) Process 0 keeps `bucket[0]` to itself, and sends the rest `P-1` buckets to their corresponding processes, *i.e.* `bucket[i]` to process `i`.

   (c) Every process sorts its bucket using bubble sort.

   (d) The processes use a collective write routine to write their results to the output file, at the right place based on process rank order.

3. (Optional) Extend this program to include file input.

Submission

As usual, write a short report, in plain text or pdf, summarize your work with this lab. Submit the new programs, `oddevenSort[123].c` and `bucketSort2.c`, through the "Lab8" submission folder on D2L (under the "Activities/Assignments" tab). You should submit your work before the week-end, *i.e.* Sunday 3/2.