CSC4005 Project 1 Template

Usage

```
git clone https://github.com/bokesyo/CSC4005_2022Fall_Demo.git
cd CSC4005_2022Fall_Demo/project1_template
```

Test data generator

`test_data_generator.cpp` is a test data generator. Please do not modify it. To use it, you should first compile it.

```
g++ test_data_generator.cpp -o gen
```

this operation will produce an executable named as `gen`.

Then, specify the number of elements to be sorted, and the file name.

```
./gen $number_of_elements_to_sort $save_file_name # Replace $variable with your own value.
```

For example, to generate a dataset with `10000` elements and name it as `./test_data/10000a.in`,

```
./gen 10000 ./test_data/10000a.in
```

Then you will find `10000a.in` in `./test_data` directory.

You can generate many datasets and use them to test your program.

Sequential Odd Even Transposition Sort

Please implement Parallel Odd Even Transposition Sort in `odd_even_sequential_sort.cpp`.

```
g++ odd_even_sequential_sort.cpp -o ssort
```

To run it, use

```
./ssort $number_of_elements_to_sort $path_to_input_file # Replace $variable with your own value.
```

For example,

```
./ssort 10000 ./test_data/10000a.in
```

The program will generate an output file called `10000a.in.seq.out` in `./test_data`.

Parallel Odd Even Transposition Sort

Please implement Parallel Odd Even Transposition Sort in `odd_even_parallel_sort.cpp`.

Compile

Use mpic++ to compile it .

```
mpic++ odd_even_parallel_sort.cpp -o psort
```

Run

```
salloc -n8 -p Debug # allocate cpu for your task
mpirun -np 8 ./psort $number_of_elements_to_sort $path_to_input_file # Replace $variable
with your own value.
```

For example, to sort `./test_data/10000a.in` generated before, we can use

```
salloc -n8 -p Debug # allocate cpu for your task
mpirun -np 8 ./psort 10000 ./test_data/10000a.in
```

The program will generate an output file called `10000a.in.parallel.out` in `./test_data`.

Check the correctness of your program

`check_sorted.cpp` is a tool for you to check if your sorting reuslt is correct.

To use it, first you should compile it,

```
g++ check_sorted.cpp -o check
```

Then you can utilize it by

```
./check $number_of_elements_to_sort $path_to_output_file
```

For example, if we want to check the output file `./test_data/10000a.in.parallel.out`, you can use

```
./check 10000 ./test_data/10000a.in.parallel.out
```

The output will be like (but not identical):

```
Not Sorted. 4983 errors.
```

Submit your job to sbatch

Firstly write a shell script called `sbatch_template.sh`,

```
#!/bin/bash
#SBATCH --job-name=your_job_name    # Job name
#SBATCH --nodes=1                   # Run all processes on a single node
#SBATCH --ntasks=20                 # number of processes = 20
#SBATCH --cpus-per-task=1           # Number of CPU cores allocated to each process
#SBATCH --partition=Project         # Partition name: Project or Debug (Debug is default)

cd /nfsmnt/119010355/CSC4005_2022Fall_Demo/project1_template/
mpirun -np 4 ./psort 10000 ./test_data/10000a.in
```

Then use

```
sbatch xxx.sh
```

to submit your job. No need to use `salloc` here.

You can use `squeue` command to see the status of your job, when it terminates, sbatch will create a file called `slurm-4748.out` in `/nfsmnt/119010355/CSC4005_2022Fall_Demo/project1_template/`.

If you have any suggestions, please email TA.