CMPS109 Spring 2018 : Lab 5

In this lab you will implement an embarrassingly parallel C++ multi-threaded most significant digit radix sorter that satisfies complimentary functional and non-functional requirements.

This lab is worth 7% of your final grade.

Submissions are due NO LATER than 23:59, Sunday May 6, 2018.

Late submissions will not be graded.

Background information

From https://en.wikipedia.org/wiki/Radix_sort :

"Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value."

"A most significant digit (MSD) radix sort can be used to sort keys in lexicographic order."

For example, given a vector of unsigned integers [33, 54, 3, 135, 644, 3, 5, 13, 53, 502, 99] the MSD radix sorted output would be [13, 135, 3, 3, 33, 5, 503, 53, 54, 644, 99].

Setup

SSH in to any of the CMPS109 teaching servers using your CruzID Blue credentials:

```
$ ssh <cruzid>@<server>.soe.ucsc.edu
```
(use Putty http://www.putty.org/ if on Windows)

Authenticate with Kerberos and AFS: ***(do this EVERY time you log in)***

```
$ kinit
$ aklog
```

Create a suitable place to work: ***(only do this the FIRST time you log in)***

```
$ mkdir -p ~/CMPS109/Lab5
$ cd ~/CMPS109/Lab5
```

Install the lab environment: ***(only do this once)***

```
$ tar xvf /var/classes/CMPS109/Spring18/Lab5.tar.gz
```

Build the skeleton system:

```
$ make
```

Check your implementation's functionality:

```
$ make check
```

Check your implementation's performance:

```
$ make perf
```

Calculate your expected grade:

```
$ make grade
```

 Copyright © 2018 David C. Harrison. All rights reserved.

What to submit

In a command prompt:

```
$ cd ~/CMPS109/Lab5
$ make submit
```

This creates a gzipped tar archive named `CMPS109-Lab5.tar.gz` in your home directory.

UPLOAD THIS FILE TO THE APPROPRIATE CANVAS ASSIGNMENT.

Requirements

Your completed radix sorter must do two things:

1. Correctly MSD radix sort large vectors of unsigned integers
2. Exhibit an approximately super-linear speedup as more CPU cores are used

The first being a functional requirement, the second be a non-functional (performance) requirement.

These requirements are equally weighted - see Grading Scheme below. It is recommended you work on the functional requirement first, and only then work on the non-functional requirement.

Remember: *"make it work"* always comes before *"make it fast"*.

What you need to do

The class `RadixSort` is provided, but unimplemented. You need to implement it without changing the existing signature. You can add member variable and functions, but do not change existing signatures.

Basic steps are as follows:

1. Investigate how an MSD radix sort is supposed to work
2. Implement a single threaded version of `RadixSort::msd()`
3. Implement a multi-threaded version of `RadixSort::msd()`

Note that the multi-threaded version is simply required to run multiple instances of the single threaded version in parallel.

To execute your MSD radix sorter after running `make`:

```
$ ./radix 9999 2 1 -d
```

Where "9999" is the number of random unsigned integers that the test harness will generate, "2" is the number of times it will generate that many random unsigned integers, and "1" is the maximum number of CPU cores to use when sorting the lists.

The `-d` flag requests a sampled dump of the sorted vectors to demonstrate (hopefully) correct ordering.

A more strenuous test might be:

```
$ ./radix 5555555 16 8
```

As in previous labs, if there's something you don't understand, do this, in this order:

1. Google it
2. Post a discussion on Canvas
3. Come along to a section and/or office hours and ask questions

 Copyright © 2018 David C. Harrison. All rights reserved.

Grading scheme

The following aspects will be assessed by executing your code on a machine with an identical configuration to the CMPS109 teaching servers:

1. (70 Marks) **Does it work?**

    a. Functional Requirement (30 marks)
    b. Non-Functional Requirement (30 marks)
    c. Code free of warnings (10 marks)

For a, marks are deducted for any sort operations failing to produce the correct answer.

For b, marks are deducted for any multi-core sorts failing to perform as required.

2. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%)

    b. Your submission is determined to be a copy of another CMPS109 student's submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 50% copied code No deduction
        - 50% to 75% copied code (-50%)
        - > 75% (-100%)

 Copyright © 2018 David C. Harrison. All rights reserved.