1. General Remarks

You are asked to benchmark memory access time for computers in Phillips 314 Lab running Linux OS.

- There are two benchmarks.
- Your benchmarks must be written in standard C language and compiled by the `gcc` compiler in the 314 Phillips computer lab.
- Your code must be well documented so any person with limited knowledge of C could understand what the code is doing.
- The first line in the code must show your name and net id, and specify how the code should be compiled.
- Each code must be saved in a separate file named `your_net_id_problem_number.c`.
- Results from the benchmarks need to be described in a file `your_net_id_benchmark_number.pdf`.
- All files need to be archive with the `tar` or `gzip` facilities. The archive must have the name `your_net_id_assignment_1.suffix` where `suffix` is either `tar` or `zip`. Please submit your work to Blackboard and Ewmail a copy to `awb8@cornell.edu`

ECE5720 has a priority access to 314 Phillips Monday-Wednesday 4pm-8pm. The lab is reserved by other classes at other times. If some of the workstations are not occupied it is OK to use them. However you may be asked to vacate the station if students from the reserved classs show up.

Once your programs compile and run get "clean" time measurements. For that make sure that you are the only user on the workstation.

2. Timing C codes

You will need to use a fine resolution clock to time your codes. You can find examples how to do it at

`https://www.cs.rutgers.edu/~pxk/416/notes/c-tutorials/gettime.html`

Check also

`http://linux.die.net/man/3/clock_gettime`

3. Problem 1

Given points $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k}) \in R^k$, $i = 1, ..., N$, find the centroid $c = (c_1, c_2, \ldots, c_k)$,

$$c = \frac{1}{N} \sum_{i=1}^{N} x_i$$

by the following methods.

1. Store a 2D array $X = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})_{i=1}^{N}$. Sum the coordinates $x_{i,1}$, $x_{i,2}, \ldots, x_{i,k}$ in the following double loop,

```
Initialize all c(j) to zero
for i = 1 to N
  for j = 1 to k
    c(j) := c(j) + x(i,j)
```

2. Store a 2D array $X = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})_{i=1}^{N}$. Sum the coordinates $x_{i,1}$, $x_{i,2}, \ldots, x_{i,k}$ in the following double loop,

```
Initialize all c(j) to zero
for j = 1 to N
  for i = 1 to k
    c(j) := c(j) + x(i,j)
```

Which strategy will comput the centroid faster and why?

Things to do.

- For this experiment use $k = 15$, $k = 17$ and $k = 64$.
- Set $N = 2^{20}$ (or as large as possible). Allocate arrays dynamically.
- Populate your array with random numbers,

```
float x = ((float)rand()/(float)(RAND_MAX));
```

- For timing purpose average 10 runs for each case.
- Time only the part where the centroids are computed. (Do not time parts where arrays are created.)

4. Problem 2

In this assignment you will test memory access time by touching elements of a linear array `A` with a progressively growing stepsize (stride). You will work with arrays of a `float` type.

Define an array of length `MAX_LENGTH` $= 2^{26}$ (decrease the length if you get segmentation faults). Probe the access time to subarrays `A[0:n]` staring from `n = MIN_SIZE` $= 2^{10}$, then doubling `n` until `n = MAX_LENGTH`. For each length, touch elements which are `STRIDE` $= 1$ apart, again doubling `STRIDE` until `STRIDE` becomes half of the array length. Runs for each combination of array length and stride length should be repeated 10 times and the timing for each combination should be averaged (by 10). A pseudo code might look as follows

```
for (n := 2*n)                                      (double the length of n)
  for array A[0:n-1]
    for (s := 2*s)                                  (double the stride)
      Tstart = clock_gettime()                      (start the timer)
      for (k = 0 to s*K)                            (repeat s*K times)
        for (i = 0:s:n-1)
          load A(i)                                 (load A[0], A[s], A[2s],..)
      T(n,s) = (clock_gettime() - Tstart)/(n*K)   (average the time)
```

All (averaged) results should be poltted on the same graphs with the x-axis being the stride length (on logarithmic scale), and the y-axis being the (averaged) time.

If plots are "croweded" split them so trends are visible.

Use different colors (and/or different markers) for each length of the array. You may want to graph your results in MATLAB.

Once you have your (combined) plots ready, find (if possible) from the plots

1. L1 caache size
2. L1 cache line length,
3. L1 associativity.
4. Explain how you figured out answers to (1)-(3).

Is there enough information to find the above for L2 cache? If so, do it. If no, justify why no.

Is there anything else that you can find out from this experiment?

Each answer must be fully explained.

Your grade will depend on

- clarity of your codes
- clarity of your discussion

This assignment is an individual assignment. You may discuss the assignment with your classmate but the work must be your own (no code sharing). If you use outside sources (like books, articles, internet) for this assignment you must clearly cite them.