ECE5720 Programming Assignment 1, due February 6 Spring 2020

1. General Remarks

You are asked to write 3 standard C codes

1. a benchmark for memory access time
2. a simple "row-by-colum" matrix matrix multiplication
3. a "tiled" matrix-matrix multiplication

Your codes must compile by a standard `gcc` compiler and benchmarked on `ecelinux.ece.cornell.edu`.

- 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.
- Codes must be saved in separate text files named `your_net_id_problem_number.c`.
- Results from the benchmarks need to be described in a file `your_net_id_benchmark_number.pdf` (please DO NOT submit `*.docx` files)
- All files need to be archive with `tar` or `gzip`. The archive must have the name `your_net_id_HW_1.suffi` where `suffix` is either `tar` or `zip`. Please submit your work to Canvas.

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

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).

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.

4. Matrix-matrix multiplication

Recall the problem of matrix-matrix multiplication discussed in Lecture 1. There were two algorithms presented

- an elementary "row-by-column" approach, and
- a blocked (or tiled) approach.

You are asked to write a C code implementing these two sequential algorithms.

5. General instructions

- there should be a single file for each code
- name the files as follows
  - `mynetid_mm_rbyc.c` for the row-by-column method
  - `mynetid_mm_tile.c` for the blocked method
  - `mynetid` is your Cornell net ID.
- the first few lines in each file should say how the code is to be compiled and executed
- your codes must be written in standard C language and compiled by the `gcc` compiler (use the `-O3` optimization flag), please use only standard C directives (OS independent)
- if your codes generate compile errors on Linux machines they will be considered as non-compliant and will not earn any credit
- your code must be well documented so any person with limited knowledge of C could understand what the code is doing
- please clean the code so all spurious and debugging commands are removed
- your codes and findings (discussion of performance) must be described in a file `mynetid_hw1.pdf`, for clarity and to save space you can refer to relevant lines in the codes,

- all files need to be archived with `tar` or `gzip` , the archive file should be named `mynetid_hw1.suffix` where `suffix` is either `tar` or `zip`,
- submit your work on Canvas.

3. Code specific instructions

- `mynetid_mm_rbyc.c` should take from the command line user supplied parameter `dim_n` which defines the dimension of two square matrices to be multiplied
- `mynetid_mm_tile.c` should take from the command line user supplied parameters
  - `dim_n` which is the dimension of two square matrices to be multiplied,
  - `tile_size` is the dimension of tiles
  - the user should be informed that `dim_n` has to be divisible by `tile_size`,
- `mynetid_mm_pt.c` should take from the command line user supplied parameters
  - `dim_n`, the dimension of two square matrices to be multiplied,
  - `nthreads`, the number of threads to be used
  - you can use any work assignment to threads you find applicable but try to make the code as efficient as possible
- the memory for the matrices should be allocated dynamically, for example by the following commands

```
/* using a single pointer */
     int *mat = (int *)malloc(r * c * sizeof(int));
     /* loop to fill-in the array */
     *(mat + i*c + j) = (read from file or set to f(i,j));

/* using an array of pointers */
     int *mat[r];
     for (i=0; i<r; i++)
       mat[i] = (int *)malloc(c * sizeof(int));
     /* loop to fill-in the array */
     mat[i][j] = (read from file or set to f(i,j));
```

- populate matrices by calls to the library function `drand48()`, set the seed by calling `srand48(1)`
- you have to time your codes, for a high resolution clock consult
  - `https://www.cs.rutgers.edu/~pxk/416/notes/c-tutorials/gettime.html`
  - `http://linux.die.net/man/3/clock_gettime`

- if you relay on resources outside lecture notes but publically available, you need to cite sources in your write-up,
- follow the above to the letter

6. Benchmarking.

In this assignment you want to investigate the influence of various parameters on the performance (speed) of the codes. Please discuss the following points.

- How does the performance of the sequential row-by-column compare to the sequential blocked matrix-matrix multiplication?
- How does perfomance vary with different dimensions of tiles? What is the speed-up (or slow down) when tiling is used?

Parameters:

(a) To assess the quality of your codes you need to banchmark your code for a range of dimensions and thread numbers.

(b) Please graph and tabulate your results and discuss your findings in the file `mynetid_hw1.pdf`.

Please present your tables and graphs in a way so they are easily readable. For example, if certain combinations of (number o threads, matrix dimension) do not bring any new information, you may omit them from your graphs. But then explain why you are omitting them.