**Introduction**

The purpose of this assignment is to become familiar with pthreads and OpenMP by parallelizing a serial program that computes the k-most similar documents of a set of documents.

I have provided a serial version of the program ([serial.tar.gz](https://moodle2.umn.edu/file.php/16303/assignments/assignment2/serial.tar.gz)), which takes as input a matrix representation of a set of documents, and for each document it computes the k most similar documents. Download and build the program. You can run it by executing the following command: 'sd -nnbrs=20 -minsim=.3 sports.mat sports.nbrs', which will generate the sports.nbrs file, listing the nearest neighbors.

**Requirements/Guidelines**

* Your task is to develop a pthreads and an OpenMP version of the sd program that should scale well all the way up to 8 threads. That means that the speedup that you should be getting should be greater than 7. That speedup should be realized for the following input datasets:
  + sports.mat, sports5.mat, sports10.mat, sports20.mat, sports30.mat.
* The above input files can be accessed on any of the parallel machines at the directory '/export/scratch/csci5451\_s13/hw2\_data/sports\*.mat'
* Use the programs that you developed to find the 50 and 200 neighbors of each document. That is, run the program using '-nnbrs=50' and '-nnbrs=200'.
* The output produced by your parallel program should be identical to that produced by the serial program. That is, for each document its similar documents will be output in decreasing similarity order, and the set of neighbors for each document will be output starting for the neighbors of document 0, followed by the neighbors of document 1, and so. Note that during development of your program, you can output the results into a file to make sure that they are similar. Once you know that they are similar, there is no point providing the output file name (especially for the larger files).
* Keep the source trees for the pthread and openMP version of your program separately. Name the respective directories `pthreads-sd' and `openmp-sd'. Put all the parallel logic of each implementation into the file `sd.c'. This is a requirement, as this will be the only file that the TA will look while grading your program. The binaries for both source trees should still be 'sd'.
* Use the define NTHREADS to specify the number of threads to both your openMP and pthread versions. To ensure the TA will be able to run your code using different numbers of threads, have the following at the top of 'sd.c':  
  #ifndef NTHREADS  
  #define NTHREADS 8  
  #endif
* **NOTE: You can see an example of the assignment file organization at '/export/scratch/csci5451\_s13/example' on any of the parallel machines**

**What you need to turn in**

1. The source files of your programs.
2. A short writeup describing how you went about parallelizing the algorithm.
3. Timing results on 1, 2, 4, and 8 processors for your programs for each one of the above version of the sports.mat file. You can compile these timing results into nice-looking charts. Include them in the above report.
4. How to submit your programs/report
   * Create a directory that has the following structure:
     + umn-user-name/pthreads-sd/your-source-files-go-here
     + umn-user-name/openmp-sd/your-source-files-go-here
     + umn-user-name/report/the-pdf-of-your-report
   * Make sure that each of your source file directories can be built by simply executing 'make'.
   * Do not put any test files in the above directories.
   * Create a tar file from the above directory structure and compress it. That is the file should be" umn-user-name-.tar.gz.
   * Upload that single file in the moodle website by the due date.
   * You can find an example of how the assignment files should be organized at '/export/scratch/csci5451\_s13/example' on any of the parallel machines
     + Feel free to copy the Makefile and modify it to include any extra files which you may have.
5. Please submit reports in .pdf format, not .doc or .docx

**Evaluation criteria**

1. Follow the assignment directions.
2. Correct execution.
3. Achieve the desired speedup levels.