CS 395T **Scalable Machine Learning** **Fall 2014**

Homework 2

Lecturer: Inderjit Dhillon Date Due: Sep 22, 2014

Keywords: *OpenMP, Netflix, Parallel ALS*

Note:

1. The datasets for this homework can be downloaded from the Assignments tab in Canvas (called `hw2-data.zip`).
2. You will run your code and collect timing results on TACC machines. See `https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide` to know about using the Stampede cluster.
3. Submit your code through Canvas. Submit a hard copy of your answers in class.

Problem 1: Parallelizing Matrix Multiplication

You are given a skeleton sequential program for matrix multiplication below:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define M 500
#define N 500
int main(int argc, char *argv) {
 int i, j, k;
 double sum;
 double **A, **B, **C;

 A = malloc(M*sizeof(double *));
 B = malloc(M*sizeof(double *));
 C = malloc(M*sizeof(double *));
 for (i = 0; i < M; i++) {
   A[i] = malloc(N*sizeof(double));
   B[i] = malloc(N*sizeof(double));
   C[i] = malloc(N*sizeof(double));
 }
 for (i = 0; i < M; i++) {
   for (j = 0; j < N; j++) {
     A[i][j] = i+j;
     B[i][j] = i*j;
     C[i][j] = 0;
   }
 }
 for (i = 0; i < M; i++) {
   for (j = 0; j < N; j++) {
     sum = 0;
     for (k=0; k < M; k++) {
       sum += A[i][k]*B[k][j];
     }
     C[i][j] = sum;
   }
 }
}
```

You are to parallelize this algorithm in three different ways, using OpenMP:

(a) Add the necessary `pragma` to parallelize the outer **for** loop.

(b) Remove the pragma for the outer **for** loop and create a `pragma` for the middle **for** loop.

(c) Add the necessary `pragma`s to parallelize both outer and middle **for** loops.

Now:

1. (4 pt) For each of the above cases, collect timing data given 1 thread, 4 threads, 8 threads and 16 threads and two matrix sizes 50 x 50 and 500 x 500. For each set of conditions, collect ten data values and average them (timing values can vary significantly). How do the timing results compare with the sequential version (above given code)?
   Note: You can use `omp_get_wtime()` to measure runtime and `lscpu` command to get the machine configuration.
2. (1 pt) How long does Matlab's matrix-matrix multiplication take for the two matrix sizes? What do you observe?

Problem 2. Parallel Alternating Least Squares

Write a C/C++ program which implements the ALS solver for the Netflix problem and parallelize it using OpenMP. Below are some guidelines for your implementation.

- Solve the ridge regression sub-problems using any numerical library (e.g., LAPACK, Eigen, Armadillo)
- Fix the number of alternating iterations to 10 and initialize each variable with a uniform random number among $[0, 1]$.
- Write a Makefile to compile your code using `icc`. The name of the output binary file is `omp-als`.
- Usage of the binary file: `./omp-als rank lambda nr_threads data_dir`
- Datasets:
  - `hw2-data.zip` contains the text-format version of the three datasets used in HW1
  - For each dataset, there is a file called *meta*, which contains three lines. The first line contains the numbers of rows and cols of the rating matrix. The second line contains the number of observed entries and the filename of the training set. The third line contains the number of observed entries and the filename of the test set.
  - Each line in either the training/test set contains a rating (i, j, R_{ij}).

- Output the wall time (only for ALS computation) and test RMSE for each iteration. A sample output is as follows.

```
$ ./omp-als 10 1 8 ../data/small/
start with RMSE 1.70056 nr_threads 8
iter 1 walltime 0.025339 rmse 1.072345
iter 2 walltime 0.048888 rmse 1.086753
iter 3 walltime 0.072393 rmse 1.094055
iter 4 walltime 0.095570 rmse 1.097710
iter 5 walltime 0.118840 rmse 1.099892
iter 6 walltime 0.142379 rmse 1.100797
iter 7 walltime 0.165937 rmse 1.100877
iter 8 walltime 0.189532 rmse 1.100535
iter 9 walltime 0.212874 rmse 1.100090
iter 10 walltime 0.236576 rmse 1.099594
```

(a) (5 pt) Write the parallel ALS solver as required above. Give the exact command you used in Stampede to compile your code. Submit (in Canvas) a zip file `hw2sol.zip` containing your code and Makefile. Executing `make test` in your `hw2sol` directory should start running: '`./omp-als 20 1 16 ../data/large`'. Do *not* zip the `data` folder; your submission must be self-contained otherwise.

(b) (5 pt) Report the 10-iteration running time for `large` dataset with 1,2,4,8,16 threads using $k = 20$ and $\lambda = 1$.

(c) (5 pt) What is the speedup you observe with `omp-als` using 16 threads compared to the Matlab implementation provided to you as part of HW1 solutions (available in Canvas)? You will get full credit if you get at least 2x speedup.

(d) (5 pt) **Extra credit**: For every 1x speedup above 2x in part (c), you will get an extra point, and you can get up to maximum of 5 extra points.

Problem 3. Logistic Regression

Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, $\boldsymbol{x}_i \in \mathbf{R}^d$, $y_i \in \{+1, -1\}$, L2-regularized logistic regression estimates the model $\boldsymbol{w}$ by solving the following optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n} \log\left(1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)\right) := f(\boldsymbol{w})$$

(a) (3 pt) Derive the gradient and the Hessian of $f(\boldsymbol{w})$.

(b) (1 pt) Gradient descent is one of the classical first order methods for minimizing $f(\boldsymbol{w})$. The update rule for gradient descent is:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f(\boldsymbol{w}), \qquad (1)$$

where $\eta > 0$ is the step size. What is the time complexity of one gradient descent update (eq. (1)) for L2-regularized logistic regression?

(c) (1 pt) Newton method is a classical second order method for minimizing $f(\boldsymbol{w})$. The update rule for Newton method is:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta(\nabla^2 f(\boldsymbol{w}))^{-1}\nabla f(\boldsymbol{w}). \tag{2}$$

Assume we first form the Hessian matrix $\nabla^2 f(\boldsymbol{w})$ and then compute the Newton direction $(\nabla^2 f(\boldsymbol{w}))^{-1}\nabla f(\boldsymbol{w})$. What is the time complexity of one Newton update (eq. (2)) for L2-regularized logistic regression?