TÉCNICO LISBOA

Parallel and Distributed Computing

Project Assignment

RECOMMENDER SYSTEM

Version 1.0 (03/03/2020)

2019/2020

2nd Semester

Contents

Revisions

Version 1.0 (March 3, 2020)	Initial Version

1 Introduction

The purpose of this class project is to give students hands-on experience in parallel programming on both UMA and multicomputer systems, using OpenMP and MPI, respectively. For this assignment you are to write a sequential and two parallel implementations of a matrix factorization routine used in some recommender systems.

2 Problem Description

There are many services now that make recommendations for the user, be it songs, movies, books, etc. One approach is to match the previous activity of different users, and then suggest items that users with similar profiles have selected and given a high evaluation.

Given a set of items, I_0 through I_m, and a set of users, U_0 through U_n, we can build a matrix with all evaluations users have given to items. For a large number of items and/or users, this will typically be a sparse matrix, holding the subset of items that users have evaluated. For example, with three users and five items, the matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{array}{c|ccccc} & I_0 & I_1 & I_2 & I_3 & I_4 \\ U_0 & 2 & 0 & 3 & 0 & 0 \\ U_1 & 0 & 0 & 3 & 0 & 1 \\ U_2 & 0 & 5 & 3 & 4 & 0 \end{array}$$

represents a situation where user U_0 has evaluated two items, I_0 and I_2, with 2 and 5 (let's say on a 1 to 5 scale, 0s indicate no evaluation). Similarly for the other users.

In order to guess how appealing an item not yet selected by a user is, we are going to estimate values for the zeros in this matrix. To this end, we assume that the items have features, however we do not know what they are nor how many we should consider. Let us say that for the example above we have two features, F_0 and F_1. Then we can consider two auxiliary matrices, one that associates the importance of a feature to each user, $\boldsymbol{L}$, the other how the feature is present in each item, $\boldsymbol{R}$:

$$\boldsymbol{L} = \begin{array}{c|cc} & F_0 & F_1 \\ U_0 & & \\ U_1 & & \\ U_2 & & \end{array} \qquad \boldsymbol{R} = \begin{array}{c|ccccc} & I_0 & I_1 & I_2 & I_3 & I_4 \\ F_0 & & & & & \\ F_1 & & & & & \end{array}$$

The approach is then to find the matrices $\boldsymbol{L}$ and $\boldsymbol{R}$ such that the matrix $\boldsymbol{B} = \boldsymbol{L} \times \boldsymbol{R}$ has all the same non-zero entries as $\boldsymbol{A}$.

For the current example we get:

$$\boldsymbol{L} = \begin{bmatrix} 1.48 & 1.29 \\ 1.36 & 1.38 \\ 1.98 & 0.92 \end{bmatrix} \qquad \boldsymbol{R} = \begin{bmatrix} 0.72 & 2.02 & 0.93 & 1.49 & 0.25 \\ 0.73 & 1.09 & 1.26 & 1.14 & 0.48 \end{bmatrix}$$

$$\boldsymbol{B} = \boldsymbol{L} \times \boldsymbol{R} = \begin{bmatrix} 2.00 & 4.38 & 3.00 & 3.70 & 0.99 \\ 1.98 & 4.23 & 3.00 & 3.60 & 1.00 \\ 2.09 & 5.00 & 3.00 & 4.00 & 0.94 \end{bmatrix}$$

This result suggests that user U_0 is highly advised to try item I_1 and should avoid item I_4.

3 Matrix Factorization

We will use an iterative method to find the auxiliary matrices $\boldsymbol{L}$ and $\boldsymbol{R}$. We start by initializing these matrices with random values, obtaining an initial value for the approximate solution, $\boldsymbol{B}^{(0)} = \boldsymbol{L}^{(0)} \times \boldsymbol{R}^{(0)}$. We then go through each non-zero value of matrix $\boldsymbol{A}$ and improve this initial estimate by minimizing the sum of the squared errors:

$$\min \sum_{i,j} \Delta_{ij}, \qquad \Delta_{ij} = (A_{ij} - B_{ij})^2$$

The new estimates for the elements of $\boldsymbol{L}$ and $\boldsymbol{R}$ are obtained as follows. For each non-zero element A_{ij} of the matrix $\boldsymbol{A}$, update the values L_{ik} and R_{kj} using the following equations:

$$L_{ik}^{(t+1)} = L_{ik}^{(t)} - \alpha \sum_{j} \frac{\partial \Delta_{ij}}{\partial L_{ik}^{(t)}} \qquad\qquad R_{kj}^{(t+1)} = R_{kj}^{(t)} - \alpha \sum_{i} \frac{\partial \Delta_{ij}}{\partial R_{kj}^{(t)}}$$

where

$$\frac{\partial \Delta_{ij}}{\partial L_{ik}^{(t)}} = 2(A_{ij} - B_{ij}^{(t)})(-R_{kj}^{(t)}) \qquad\qquad \frac{\partial \Delta_{ij}}{\partial R_{kj}^{(t)}} = 2(A_{ij} - B_{ij}^{(t)})(-L_{ik}^{(t)})$$

The parameter α allows the tuning of the convergence rate. To simplify, we will use a fixed value for α for each problem instance.

The method can iterate until the error is below a given threshold, or by defining a maximum number of iterations, which is what we will be using.

4 Implementation Details

4.1 Input Data

Your program should allow **a single** command-line parameter, a filename with all the information of the instance to run.

The format of this text file is the following:

1. the first line is an integer, indicating the number of iterations to run;
2. the second line is a floating-point number, the value of α;
3. the third line is an integer, the number of latent features to consider;
4. the fourth line has three integers separated by a space, the number of rows and columns, respectively, and the number of non-zero elements of the input matrix;
5. the remaining lines have the matrix, one matrix element per line, each line with a set of three values: two integers indicating the row and column of the element; and the actual element, a floating-point value in the range 1 to 5.

4.2 Output Data

The output of the program should be the item recommended for each user. For each row of matrix $\boldsymbol{B}$, the program should output the index (an integer) of the item with highest value not taking into account those that had been evaluated in the give matrix $\boldsymbol{A}$. Hence, the output consists of `nU` lines with a single integer, where `nU` is the number of users.

For example, considering the matrix $\boldsymbol{A}$ given in Section 2, the program should output:

```
$ matFact inst0.in
1
1
0
```

The submitted programs should send these output lines (and **nothing else!**) to the standard output, so that the result can be validated against the correct solution.

The project **cannot be graded** unless you follow strictly these input and output rules!

4.3 Implementation Notes

Please consider the following set of notes:

- To minimize numerical errors due to rounding, use the type `double` for the floating-point numbers.
- Use the routine in Appendix A for the initialization of matrices $\boldsymbol{L}$ and $\boldsymbol{R}$.
- Note that you will need two copies of both $\boldsymbol{L}$ and $\boldsymbol{R}$, as you will need a stable copy while computing the new iteration.

The program must adhere to these rules so that we can validate your results!

5 Part 1 - Serial implementation

Write a serial implementation of the algorithm in C (or C++). Name the source file of this implementation `matFact.c`. As stated, your program should expect a single input parameter.

This version will serve as your base for comparisons and must be as efficient as possible.

6 Part 2 - OpenMP implementation

Write an OpenMP implementation of the algorithm, with the same rules and input/output descriptions. Name this source code `matFact-omp.c`. You can start by simply adding OpenMP directives, but you are free, and encouraged, to modify the code in order to make the parallelization more effective and more scalable. Be careful about synchronization and load balancing!

Important note: in order to test for scalability, we will run this program assigning different values to the shell variable `OMP_NUM_THREADS`. If you override this value in your program, we will not be able to properly evaluate the scalability of your program.

7 Part 3 - MPI implementation

Write an MPI implementation of the algorithm as for OpenMP, and address the same issues. Name this source code `matFact-mpi.c`.

For MPI, you will need to modify your code substantially. Besides synchronization and load balancing, you will need to create independent tasks, taking into account the minimization of the impact of communication costs. You are encouraged to explore different approaches for the problem decomposition.

Note that this distributed version should permit running larger instances, namely instances that do not fit in the memory of a single machine.

Extra credits will be given to groups that present a combined MPI+OpenMP implementation.

8 What to Turn in, and When

You must eventually submit the sequential and both parallel versions of your program (**please use the filenames indicated above**), and a table with the times to run the parallel versions on input data that will be made available (for 1, 2, 4 and 8 parallel tasks for both OpenMP and MPI, and additionally 16, 32 and 64 for MPI).

You must also submit a short report about the results (2 pages) that discusses:

- the approach used for parallelization
- what decomposition was used
- what were the synchronization concerns and why
- how was load balancing addressed
- what are the performance results, and are they what you expected

You will turn in the serial version and OpenMP parallel version at the first due date, with the short report, and then the serial version again (hopefully the same) and the MPI parallel version at the second due date, with an updated report. Both the code and the report will be uploaded to the Fenix system in a zip file. **Name these files** as `g<n>omp.zip` and `g<n>mpi.zip`, where `<n>` is your group number.

1st due date (serial + OMP): **April 3rd**, until 5pm.
Note: your project will be tested in the lab class after the Easter break.

2nd due data (serial + MPI): **May 15th**, until 5pm.
Note: your project will be tested in the lab class right after the due date.

A Routine `random_fill_LR()`

```
#include <stdlib.h>

#define RAND01 ((double)random() / (double)RAND_MAX)

void random_fill_LR(int nU, int nI, int nF)
{
    srandom(0);

    for(int i = 0; i < nU; i++)
        for(int j = 0; j < nF; j++)
            L[i][j] = RAND01 / (double) nF;
    for(int i = 0; i < nF; i++)
        for(int j = 0; j < nI; j++)
            R[i][j] = RAND01 / (double) nF;
}
```

To make sure your implementation is correct, the initial matrices $\boldsymbol{L}^{(0)}$ and $\boldsymbol{R}^{(0)}$ for the example in Section 2 should be:

$$\boldsymbol{L} = \begin{bmatrix} 0.420094 & 0.197191 \\ 0.391550 & 0.399220 \\ 0.455824 & 0.098776 \end{bmatrix} \qquad \boldsymbol{R} = \begin{bmatrix} 0.167611 & 0.384115 & 0.138887 & 0.276985 & 0.238699 \\ 0.314435 & 0.182392 & 0.256700 & 0.476115 & 0.458098 \end{bmatrix}$$