Sparse Matrix Multiplication

Akshit Kumar (EE14B127)

29th November 2016

Abstract

This report contains the code and algorithm for the multiplication of sparse matrix with a vector using pointers and linked lists. The goal of this problem is to make the computation of Ax and $x^T A$ as cheap as it in the array approach. This is achieved by storing the elements in the a row major sparse representation to compute Ax and a column major sparse representation to compute $x^T A$.

1 Introduction

According to the book Numerical Recipes in C, to represent a matrix A of the dimension $N \times N$, the row indexed scheme sets up two one dimensional arrays - sa and ija. The first one of these stores matrix element values in single or double precision and second stores integer values according to certain storage rules which results in an elegant storage scheme. The method described in the book is ideal when the entire matrix is given to you or the inputs are in sorted order. However in the case of this problem, the inputs don't follow any of the above criterion and hence a different scheme has been used for storing the sparse matrix and computing Ax and $x^T A$.

2 Representation of Sparse Matrix Using Linked Lists

A sparse matrix can be represented in the following form :

Check out sparse-matrix.png

Here we have sparse matrix representation where we have an array of column headers and an array of row header which point to a Node element depending on the intersection of the row value and column. The Node elements contains the value of the row, column and element value and contains two pointers - one which points downwards and other which points rightwards. The drawback with using this representation is the complicated implementation required to keep this representation.

3 Simplified representation using Row-wise and Column-wise Linked Lists

The above sparse matrix representation can be simplified by making use of two matrices - row major representation and column major representation. Row major representation can be seen below :

Check out row-wise-sparse-matrix.png

In this we have an array of pointers where each pointer points to a linked list of row elements. Similarly we can have a column major representation of the matrix. In order to implement this, each element is an instantiation of the following struct type:

```
typedef struct Node{
int row;
int col;
double val;
struct Node* next;
}Node;
```

4 Calculation in Sparse Matrix Multiplication

The calculation of the matrix is pretty straight forward and uses the following formulae :

For the calculation of $p = A.x$ using the row-wise approach $p_i = \sum A_{ij}x_j$

For the calculation of $p = x^T A$ using the column-wise approach $p_j = \sum A_{ji}x_j$

Following this approach gives a time complexity of $O(mn)$ where m is the average number of non zero elements in each row or column and n is the dimension of the matrix.

5 Advantages and Disadvantages of Using Linked List

5.1 Advantages

This method is as fast as the method described in the book Numerical Recipes in C using arrays. In addition to having similar time complexity, this method doesn't assume any special manner in which the input should be obtained and can be used for any ordering of input data.

5.2 Disadvantages

The disadvantage of this method comes in the form of overhead and increased complexity due to the usage of pointers and linked lists to implement this sparse matrix representation.

6 Output of the Program

The output of the program is written in two file - output2a.dat and output2b.dat.

The resultant p vector of the dimension 500×1due to the computation of $p = A.x$ is written to the file output2a.dat.

The resultant p vector of the dimension 1×500 due to the computation of $p = x^T A$ is written to the file output2b.dat

7 Source Code of the Program

```c
/*
 * Name : Akshit Kumar
 * Roll No. : EE14B127
 * Solution to the problem of storing sparse matrices and multiplying
     them with vectors
 */
// Inclusion of necessary libraries
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <limits.h>

/*
 * Defining the structure for Node
 * row contains the row of the non zero element
 * col contains the col of the non zero element
 * val contains the value of the non zero element in double precision
 * next holds the pointer to the next non zero element in row wise or column wise saving
 */
typedef struct Node{
  int row;
  int col;
  double val;
  struct Node* next;
}Node;

struct Node* rows[500] = {NULL}; // Array of pointers each pointing to a row linked list
struct Node* cols[500] = {NULL}; // Array of pointers each pointing to a column linked list

int x[500]; // Holds the vector x

/*
 * Function to insert an element in the row major fashion.
 * First in the array of row pointers, we go to relevant row head, then traverse the linked list to find
 * the appropriate place to add the particular node such that the columns are in ascending order in each row
 */
void insert_node_row_wise(Node *head, int row, int col, double val){
  // Make a new node
  Node *new_node;
  new_node = (Node *) malloc(sizeof(Node));
  new_node->row = row;
  new_node->col = col;
  new_node->val = val;
  new_node->next = NULL;
  // If the head points to NULL, make the head point to the new node
  if(rows[row] == NULL){
    rows[row] = new_node;
  }
  else{
    // if the col is less than the first col, new node needs to be placed right after the head
    if(col < rows[row]->col){
      new_node->next = rows[row];
      rows[row] = new_node;
    }
    // else,keep traversing using the two pointers - prev and temp to find the appropriate place to insert the node
    else{
    Node *temp,*prev;
```

```
    temp = rows[row];
    while(temp != NULL && temp->col < col){
      prev = temp;
      temp = temp->next;
    }
    new_node->next = temp;
    prev->next = new_node;
    }
  }
}

/*
 * Function to insert an element in the column major fashion.
 * First in the array of column pointers, we go to relevant column head, then traverse the linked list to find
 * the appropriate place to add the particular node such that the rows are in ascending order in each column
 */
void insert_node_col_wise(Node *head, int row, int col, double val){
  // Make a new node
  Node *new_node;
  new_node = (Node *) malloc(sizeof(Node));
  new_node->row = row;
  new_node->col = col;
  new_node->val = val;
  new_node->next = NULL;
  // If the head points to NULL, make the head point to the new node
  if(cols[col] == NULL){
    cols[col] = new_node;
  }
  else{
    // if the row is less than the first row, new node needs to be placed right after the head
    if(row < cols[col]->row){
      new_node->next = cols[col];
      cols[col] = new_node;
    }
    // else, keep traversing using the two pointers - prev and temp to find the appropriate place to insert the node
    else{
    Node *temp,*prev;
    temp = cols[col];
    while(temp != NULL && temp->row < row){
      prev = temp;
      temp = temp->next;
    }
    new_node->next = temp;
    prev->next = new_node;
    }
  }
}

/*
 * Function to multiply the A matrix with x vector. This follows the simple row-column multiplication
 * Each row element is multiplied by the x vector - ie only the non zero elements are considered in multiplication and taking sum
```

```
 */
void multiply_a_into_x(){
  double result[500] = {0.00};
  for(int i = 0; i < 500; i++){
    Node*temp = rows[i];
    // traverse the entire row and multiply each row element by its
        corresponding column element in vector and add
    while(temp != NULL){
      result[i] = result[i] + (temp->val * (double) x[temp->col]);
      temp = temp->next;
    }
  }
  // Writing out the solution to the output file - output2a.dat
  FILE *file = fopen("output2a.dat","w");
  fprintf(file,"Solution of A.x is :\n");
  printf("Solution of A.x is :\n");
  for(int i = 0; i < 500; i++){
    printf("%lf\n",result[i]);
    fprintf(file,"%lf\n",result[i]);
  }
  fclose(file);
}

/*
 * Function to multiply x' row vector with A matrix. This again
     follows the simple row-column multiplication.
 * Each column element of A matrix is multiplied with row element of x
     and sum is taken
 */
void multiply_x_transpose_into_a(){
  double result[500] = {0.00};
  for(int i = 0; i < 500; i++){
    Node* temp = cols[i];
    // traverse the entire column and multiply each column element by
        its corresponding row element in vector and add
    while(temp != NULL){
      result[i] = result[i] + (temp->val * (double)x[temp->row]);
      temp = temp->next;
    }
  }
  // Writing out the solution to the output file - output2b.dat
  FILE *file = fopen("output2b.dat","w");
  fprintf(file,"Solution of x'.A is : \n");
  printf("Solution of x'.A is :\n");
  for(int i = 0; i < 500; i++){
    printf("%lf ",result[i]);
    fprintf(file,"%lf ",result[i]);
  }
  printf("\n");
  fclose(file);
}

int main(int argc,char** argv){
  if(argc != 2){
    printf("Usage ./a.out <filename>\n");
    exit(1);
  }
```

```
  FILE *file = fopen(argv[1],"r");
  if(file == NULL){
    printf("Unable to open file\n");
    exit(1);
  }
  int i = 0;
  char line[256];
  // Read the data from the input file
  while(fgets(line,sizeof(line),file) != NULL){
    char *pos;
    if ((pos=strchr(line, '\n')) != NULL)
      *pos = '\0';
    if(line[0] == '#' || line[0] == '\n'){
      continue;
    }
    else{
      int row,col,x_val;
      double val;
      if(sscanf(line,"%d %d %lf",&row,&col,&val) == 3){
        insert_node_row_wise(rows[row],row,col,val); // make sparse
            matrix using row-major approach
        insert_node_col_wise(cols[col],row,col,val); // make sparse
            matrix using column-major approach
      }
      else if(sscanf(line,"%d",&x_val) == 1){
        x[i++] = x_val; // get the values of the x vector
      }
    }
  }
  multiply_a_into_x(); // perform A.x
  multiply_x_transpose_into_a(); // perform x'.A
  return 0;
}
```

sparse–matrix.c