Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

02.03.02 Фундаментальная информатика и информационные технологии

ОТЧЕТ
по научно-исследовательской работе

по направлению 02.03.02 «Фундаментальная информатика и информационные технологии», направленность (профиль) – «Системное программное обеспечение», квалификация – бакалавр, программа прикладного бакалавриата, форма обучения – очная, год начала подготовки (по учебному плану) – 2019

Выполнил:
студент гр. ИС-941
«1» июня 2021 г. _____ Степанищева А. В.

Оценка «_____»

Руководитель практики
от университета
доцент кафедры ВС
«29» мая 2021 г. _____ Пудов С. Г.

Новосибирск 2021

ПЛАН-ГРАФИК ПРОВЕДЕНИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ

Тип практики: учебная практика по получению первичных профессиональных умений и навыков, в том числе получение умений и навыков научно-исследовательской деятельности
Способ проведения практики: стационарная
Форма проведения практики: дискретно по периодам проведения практики

Тема: Реализация умножения разреженной матрицы на вектор, исследование влияния выбранного формата на производительность.

Содержание практики

Наименование видов деятельности	Дата (начало – окончание)
1.Общее ознакомление со структурным подразделением СибГУТИ, вводный инструктаж по технике безопасности	01.02.2021–13.02.2021
2.Выдача задания на практику, определение конкретной индивидуальной темы, формирование плана работ	15.02.2021–20.02.2021
3.Работа с библиотечными фондами Кафедры вычислительных систем, сбор и анализ материалов по теме практики	22.02.2021–20.03.2021
4.Выполнение работ в соответствии с составленным планом - Реализация чтения данных из файлов с расширением .mtx (Matrix Market format); - Перевод со способа хранения матрицы COO на CRS (CSC); - Реализация алгоритмов умножения матрицы на вектор для каждого способа хранения матрицы; - Замер производительности алгоритмов;	22.03.2021 – 22.05.2021
5.Анализ полученных результатов и произведенной работы, составление отчета по практике	24.05.2021–29.05.2021

Согласовано:

Руководитель практики
от университета
доцент кафедры ВС _____ Пудов С. Г.

ЗАДАНИЕ НА ПРАКТИКУ

Реализовать умножение разреженной матрицы на вектор в COO и CSR форматах, исследовать влияние выбранного формата хранения на производительность. Замерить время работы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Разреженная матрица — это матрица с преимущественно нулевыми элементами. В противном случае, если бóльшая часть элементов матрицы ненулевые, матрица считается **плотной**.

Вектор - в простейшем случае математический объект, характеризующийся величиной и направлением. Например, в геометрии и в естественных науках вектор есть направленный отрезок прямой в евклидовом пространстве (или на плоскости).

Существует несколько способов хранения (представления) разреженных матриц, отличающиеся:

- удобством изменения структуры матрицы (активно используется косвенная адресация) - это структуры в виде списков и словарей.
- скоростью доступа к элементам и возможной оптимизацией матричных вычислений (чаще используются плотные блоки - массивы, увеличивая локальность доступа к памяти).

Список координат (COO - Coordinate list) хранится список из элементов вида (строка, столбец, значение).

Сжатое хранение строкой (CSR - compressed sparse row). Формат хранения CSR (Compressed Sparse Rows) или CRS (Compressed Row Storage) призван устранить некоторые недоработки координатного представления.
Используются три массива:

– первый массив хранит значения элементов построчно

(строки рассматриваются по порядку сверху вниз),

– второй – номера столбцов для каждого элемента,

– третий заменяет номера строк, используемые в координатном формате, на индекс начала каждой строки.

Количество элементов массива RowIndex равно N + 1.

- *i*-ый элемент массива RowIndex указывает на начало i-ой строки.

- Элементы строки *i* в массиве Value находятся по индексам от RowIndex[i] до RowIndex[i + 1] – 1 включительно.

- Таким образом обрабатывается случай пустых строк, а также добавляется «лишний» элемент в массив RowIndex – устраняется особенность при доступе к элементам последней строки. RowIndex[N] = NZ. (NZ = non-zeros, ненулевые элементы матрицы)

По заданию требовалось реализовать умножение разреженной матрицы на вектор именно в форматах COO и CSR.

- При умножении разреженной матрицы на вектор, в качестве результата этой операции будет заполненный вектор, а не разреженная матрица.

В ходе выполнения работы использовались матрицы в координатном формате из открытого интернет-ресурса The SuiteSparse Matrix Collection .

ОСНОВНЫЕ ФУНКЦИИ

Для чтения матриц в координатном формате была использована библиотека **mmio.h**.

Для умножения разреженной матрицы на вектор в формате COO реализована функция **void spmv_coo(int *row, int *col, double *value, int matsize, double *vec, double *coo_res)**. На вход подаётся 6 параметров: массив строк, массив столбцов, массив значений, количество ненулевых элементов в матрице, вектор и массив для записи результата умножения.

Для умножения разреженной матрицы на вектор в формате CSR реализована функция **void spmv_csr(int *col, double *value, int rownum, double *vec, double *crs_res, int *rowoffsets)**. На вход подаётся 6 параметров: массив строк, массив значений, количество строк, вектор, массив для записи результата умножения и массив индексов строк.

Стоит заметить, что перед тем, как выполнять умножение матрицы в формате CSR необходимо выполнить конвертацию: из формата хранения COO в формат CSR. Данная конвертация формата реализована в функции **void coo_csr(int* rowoff, int* row, int size)**. На вход подаётся 3 параметра: массив индексов строк, массив строк и размер матрицы. По завершении работы функции в массив **rowoffsets** записываются индексы строк разреженной матрицы.

Для того, чтобы конвертация работала корректно, необходимо было отсортировать матрицу. Для сортировки был реализован алгоритм сортировки Quick Sort **void quickSort(int *row, int *col, double *value, int left, int right)**. На вход подаётся 5 параметров: массив строк, массив столбцов, массив значений, а также левая и правая границы.

Для работы со временем используется библиотека **time.h**, а именно функция **time()**.

ВЫПОЛНЕНИЕ

1. **Объявление переменных.**

```
FILE *f = NULL;
MM_typecode matcode;
int ret_code;
int vector, outputsize, rownum, colnum, rowoffsize, *rowoffsets;
int *row, *col;
double *value, *crs_res, *coo_res, *vec;
int matsize;
clock_t timer_coo, timer_csr;
```

Объявляем переменные для работы с матрицей и вектором. Также объявляем две переменные типа **clock_t** для работы со временем (замер времени работы программы).

2. **Выполнение проверки на корректность аргумента (файла) с помощью библиотеки mmio.h.**

```
if (argc < 2) {
    fprintf(stderr, "Usage: %s [martix-market-filename]\n", argv[0]);
    exit(1);
} else {
    if ((f = fopen(argv[1], "r")) == NULL) exit(1);
}

if (mm_read_banner(f, &matcode) != 0) {
    printf("Could not process Matrix Market banner.\n");
    exit(1);
}

if (mm_is_complex(matcode) && mm_is_matrix(matcode) &&
    mm_is_sparse(matcode)) {
    printf("Sorry, this application does not support ");
    printf("Market Market type: [%s]\n", mm_typecode_to_str(matcode));
    exit(1);
}

if ((ret_code = mm_read_mtx_crd_size(f, &rownum, &colnum, &matsize)) != 0)
    exit(1);
```

3. **Динамическое выделение памяти. Проверка на корректное выделение памяти.**

```
row = (int *)malloc(matsize * sizeof(int));
assert(row != NULL);
col = (int *)malloc(matsize * sizeof(int));
assert(col != NULL);
value = (double *)malloc(matsize * sizeof(double));
assert(value != NULL);
vector = rownum * sizeof(double);
vec = (double*)malloc(vector);
assert(vec != NULL);
rowoffsize = (rownum + 1) * sizeof(int);
rowoffsets = (int*)malloc(rowoffsize);
assert(rowoffsets != NULL);
outputsize = rownum * sizeof(double);
coo_res = (double*)malloc(outputsize);
assert(coo_res != NULL);
crs_res = (double*)malloc(outputsize);
assert(crs_res != NULL);
```

С помощью функции **malloc()** из библиотеки **stdlib.h** выделили память для всех переменных, с которыми будем работать далее.

4. **Считывание значений.**

```
for (int i = 0; i < matsize; i++) {
    fscanf(f, "%d %d %lg", &row[i], &col[i], &value[i]);
}
fclose(f);
```

В цикле считываем значения из файла и записываем их в массив строк, столбцов и значений, после чего закрываем поток.

5. **Заполнение вектора рандомными значениями.**

```
srand((int)time(NULL));
for (int i = 0; i < rownum; i++) {
    vec[i] = rand() / (double)RAND_MAX;
}
```

6. **Умножение разреженной матрицы на вектор, формат хранения COO**

```
timer_coo = clock();
spmv_coo(row, col, value, matsize, vec, coo_res);
timer_coo = clock() - timer_coo;

for (int i = 0; i < rownum; i++) {
    printf("Result vector element %d = %lf\n\n", i, coo_res[i]);
}

double total_coo = (((double)timer_coo)/CLOCKS_PER_SEC);
printf("# Elapsed time coo (sec): %.6f\n\n", total_coo);
```

Первым делом запускаем таймер, для того, чтобы вычислить время работы для этого способа хранения разреженной матрицы. Далее вызываем функцию умножения, после чего выводим в терминал результат работы программы и время выполнения.

7. **Сортировка матрицы и конвертация в формат CSR**

```
quickSort(row, col, value, 0, matsize - 1);
coo_csr(rowoffsets, row, matsize);
```

Перед конвертацией матрицы из формата хранения COO в формат хранения CSR необходимо отсортировать матрицу по неубыванию. После сортировки вызываем функцию конвертации.

8. **Умножение разреженной матрицы на вектор, формат хранения CSR.**

```
timer_csr = clock();
spmv_csr(col, value, rownum, vec, crs_res, rowoffsets);
timer_csr = clock() - timer_csr;

for (int i = 0; i < rownum; i++) {
    printf("Result vector element %d = %lf\n\n", i, crs_res[i]);
}

double total_csr = (((double)timer_csr)/CLOCKS_PER_SEC);
printf("# Elapsed time csr (sec): %.6f\n\n", total_csr);
```

Первым делом запускаем таймер, для того, чтобы вычислить время работы для этого способа хранения разреженной матрицы. Далее вызываем функцию умножения, после чего выводим в терминал результат работы программы и время выполнения.

9. **Освобождение памяти.**

```
free(col);
free(row);
free(value);
free(vec);
free(rowoffsets);
free(crs_res);
free(coo_res);
```

ЗАКЛЮЧЕНИЕ

В результате учебной практики разработаны и исследованы алгоритмы умножения разреженной матрицы на вектор с использованием библиотеки ANSI C для ввода - вывода Matrix Market. Мной были получены знания о способах хранениях разреженной матрицы, а именно список координат **(COO - Coordinate list)** и сжатое хранение строкой **(CSR - compressed sparse row, CRS - compressed row storage, Йельский формат)**.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

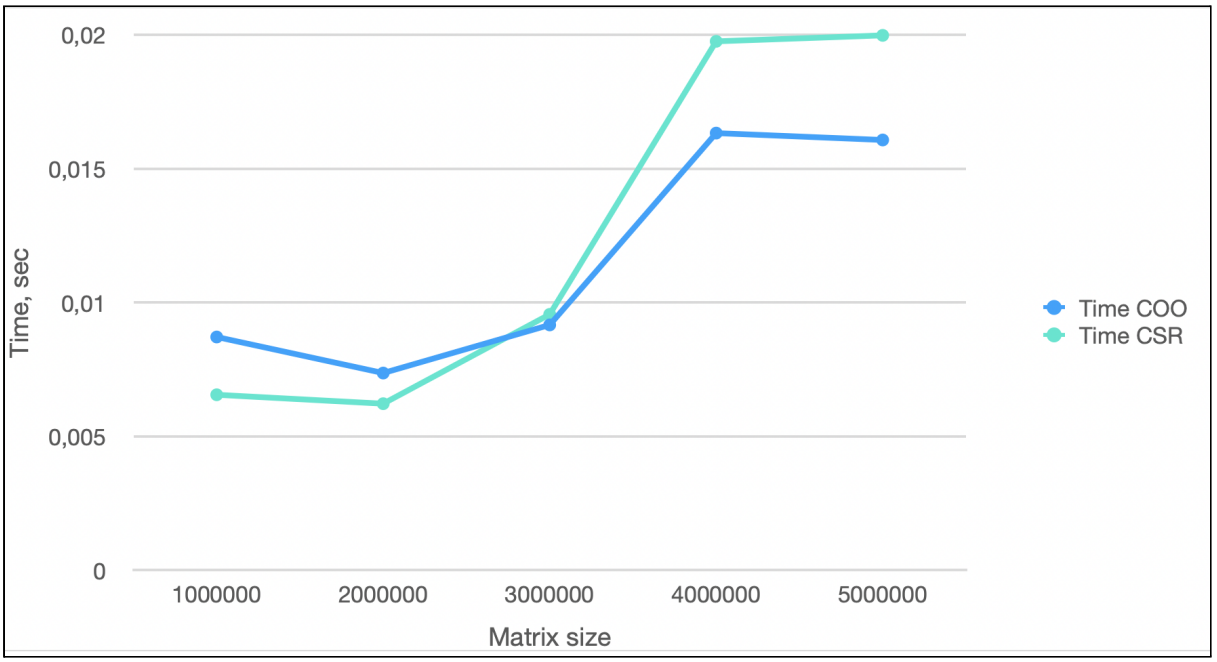
1. Reginald P. Tewarson. Sparse Matrices. — Academic Press, 1973. — 160 с. — ISBN 0126856508. перевод: Тьюарсон Р. Разреженные матрицы = Sparse Matrices. — М.: Мир, 1977. — 191 с.

2. Писсанецки С. Технология разреженных матриц = Sparse Matrix Technology. — М.: Мир, 1988. — 410 с. — ISBN 5-03-000960-4.

3. Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений = Computer Solution of Large Sparse Positive Definite Systems. — М.: Мир, 1984. — 333 с.

ПРИЛОЖЕНИЯ

1. График измерений



2. Листинг программы

main. c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include "quicksort.h"
#include "mmio.h"
#include "functions.h"


int main(int argc, const char * argv[]) {

    FILE *f = NULL;
    MM_typecode matcode;
    int ret_code;
    int vector, outputsize, rownum, colnum, rowoffsize, *rowoffsets;
    int *row, *col;
    double *value, *crs_res, *coo_res, *vec;
    int matsize;
    clock_t timer_coo, timer_csr;

    if (argc < 2) {
        fprintf(stderr, "Usage: %s [martix-market-filename]\n", argv[0]);
        exit(1);
    } else {
        if ((f = fopen(argv[1], "r")) == NULL) exit(1);
```

```
}

if (mm_read_banner(f, &matcode) != 0) {
   printf("Could not process Matrix Market banner.\n");
   exit(1);
}

if (mm_is_complex(matcode) && mm_is_matrix(matcode) &&
   mm_is_sparse(matcode)) {
   printf("Sorry, this application does not support ");
   printf("Market Market type: [%s]\n", mm_typecode_to_str(matcode));
   exit(1);
}

if ((ret_code = mm_read_mtx_crd_size(f, &rownum, &colnum, &matsize)) != 0)
   exit(1);

row = (int *)malloc(matsize * sizeof(int));
assert(row != NULL);
col = (int *)malloc(matsize * sizeof(int));
assert(col != NULL);
value = (double *)malloc(matsize * sizeof(double));
assert(value != NULL);
vector = rownum * sizeof(double);
vec = (double*)malloc(vector);
assert(vec != NULL);
rowoffsize = (rownum + 1) * sizeof(int);
rowoffsets = (int*)malloc(rowoffsize);
assert(rowoffsets != NULL);
outputsize = rownum * sizeof(double);
coo_res = (double*)malloc(outputsize);
assert(coo_res != NULL);
crs_res = (double*)malloc(outputsize);
assert(crs_res != NULL);


for (int i = 0; i < matsize; i++) {
   fscanf(f, "%d %d %lg", &row[i], &col[i], &value[i]);
}
fclose(f);

srand((int)time(NULL));
for (int i = 0; i < rownum; i++) {
   vec[i] = rand() / (double)RAND_MAX;
}

timer_coo = clock();
spmv_coo(row, col, value, matsize, vec, coo_res);
timer_coo = clock() - timer_coo;

for (int i = 0; i < rownum; i++) {
   printf("Result vector element %d = %lf\n\n", i, coo_res[i]);
```

```
    }

    double total_coo = (((double)timer_coo)/CLOCKS_PER_SEC);
    printf("# Elapsed time coo (sec): %.6f\n\n", total_coo);

    quickSort(row, col, value, 0, matsize - 1);
    coo_csr(rowoffsets, row, matsize);

    timer_csr = clock();
    spmv_csr(col, value, rownum, vec, crs_res, rowoffsets);
    timer_csr = clock() - timer_csr;

    for (int i = 0; i < rownum; i++) {
        printf("Result vector element %d = %lf\n\n", i, crs_res[i]);
    }

    double total_csr = (((double)timer_csr)/CLOCKS_PER_SEC);
    printf("# Elapsed time csr (sec): %.6f\n\n", total_csr);


    free(col);
    free(row);
    free(value);
    free(vec);
    free(rowoffsets);
    free(crs_res);
    free(coo_res);
    return 0;
    }
```

quicksort.h

```
#ifndef quicksort_h
#define quicksort_h

void swap(int* a, int* b);
void swap_val(double* a, double* b);
void quickSort(int *row, int *col, double *value, int left, int right);

#endif /* quicksort_h */
quicksort.c
#include "quicksort.h"

void swap(int* a, int* b)
{
  int t = *a;
  *a = *b;
  *b = t;
}
void swap_val(double* a, double* b)
```

```
{
  double t = *a;
  *a = *b;
  *b = t;
}

void quickSort(int *row, int *col, double *value, int left, int right)
{
  int i = left, j = right;
  int pivot = row[(left + right) / 2];
  int pivot_col = col[(left + right) / 2];

  while(i <= j) {
    while(row[i] < pivot || (row[i] == pivot && col[i] < pivot_col))
      i++;
    while(row[j] > pivot || (row[j] == pivot && col[j] > pivot_col))
      j--;
    if(i <= j) {
      swap(&row[i], &row[j]);
      swap(&col[i], &col[j]);
      swap_val(&value[i], &value[j]);
      i++;
      j--;
    }
  }
  if(left < j)
    quickSort(row, col, value, left, j);
  if (i < right)
    quickSort(row, col, value, i, right);
}
```

functions.h

```
#ifndef functions_h
#define functions_h

#include <stdio.h>

void spmv_coo(int *row, int *col, double *value, int matsize, double *vec, double *coo_res);
void coo_csr(int* rowoff, int* row, int size);
void spmv_csr(int *col, double *value, int rownum, double *vec, double *crs_res, int *rowoffsets);

#endif /* functions_h */
```

functions.c

```
#include "functions.h"
```

```
void coo_csr(int* rowoff, int* row, int size){

   rowoff[0] = 0;
   int prev = 0, accurate = 1, j = 1;

   for (int i = 1; i < size; i++) {
      if (row[i] - row[prev] > 1) {
         for (int k = 0; k < row[i] - row[prev]; k++) {
            rowoff[j++] = accurate;
         }
         prev = i;
      } else if (row[prev] != row[i]) {
         rowoff[j++] = accurate;
         prev = i;
      }
      accurate+=1;
      rowoff[j] = accurate;

   }
}

void spmv_coo(int *row, int *col, double *value, int matsize, double *vec, double *coo_res) {

   for (int j = 0; j < matsize; j++) {
      coo_res[row[j]] += value[j] * vec[col[j]];
   }
}

void spmv_csr(int *col, double *value, int rownum, double *vec, double *crs_res, int *rowoffsets) {

   double temp;
   for (int i = 0; i < rownum; i++) {
      temp = crs_res[i];
      for (int j = rowoffsets[i]; j < rowoffsets[i+1]; j++) {
         temp += value[j] * vec[col[j]];
      }
      crs_res[i] = temp;
   }
}
```

mmio.h

```
#ifndef MM_IO_H
#define MM_IO_H

#define MM_MAX_LINE_LENGTH 1025
#define MatrixMarketBanner "%%MatrixMarket"
#define MM_MAX_TOKEN_LENGTH 64
#include <stdio.h>
```

```
typedef char MM_typecode[4];

char *mm_typecode_to_str(MM_typecode matcode);

int mm_read_banner(FILE *f, MM_typecode *matcode);
int mm_read_mtx_crd_size(FILE *f, int *M, int *N, int *nz);
int mm_read_mtx_array_size(FILE *f, int *M, int *N);

int mm_write_banner(FILE *f, MM_typecode matcode);
int mm_write_mtx_crd_size(FILE *f, int M, int N, int nz);
int mm_write_mtx_array_size(FILE *f, int M, int N);


/********************* MM_typecode query fucntions ***************************/

#define mm_is_matrix(typecode)    ((typecode)[0]=='M')

#define mm_is_sparse(typecode)    ((typecode)[1]=='C')
#define mm_is_coordinate(typecode)((typecode)[1]=='C')
#define mm_is_dense(typecode)    ((typecode)[1]=='A')
#define mm_is_array(typecode)    ((typecode)[1]=='A')

#define mm_is_complex(typecode)    ((typecode)[2]=='C')
#define mm_is_real(typecode)        ((typecode)[2]=='R')
#define mm_is_pattern(typecode)    ((typecode)[2]=='P')
#define mm_is_integer(typecode) ((typecode)[2]=='I')

#define mm_is_symmetric(typecode)((typecode)[3]=='S')
#define mm_is_general(typecode)    ((typecode)[3]=='G')
#define mm_is_skew(typecode)    ((typecode)[3]=='K')
#define mm_is_hermitian(typecode)((typecode)[3]=='H')

int mm_is_valid(MM_typecode matcode);        /* too complex for a macro */


/********************* MM_typecode modify fucntions ***************************/

#define mm_set_matrix(typecode)    ((*typecode)[0]='M')
#define mm_set_coordinate(typecode)    ((*typecode)[1]='C')
#define mm_set_array(typecode)    ((*typecode)[1]='A')
#define mm_set_dense(typecode)    mm_set_array(typecode)
#define mm_set_sparse(typecode)    mm_set_coordinate(typecode)

#define mm_set_complex(typecode)((*typecode)[2]='C')
#define mm_set_real(typecode)    ((*typecode)[2]='R')
#define mm_set_pattern(typecode)((*typecode)[2]='P')
#define mm_set_integer(typecode)((*typecode)[2]='I')


#define mm_set_symmetric(typecode)((*typecode)[3]='S')
#define mm_set_general(typecode)((*typecode)[3]='G')
#define mm_set_skew(typecode)    ((*typecode)[3]='K')
```

```
#define mm_set_hermitian(typecode)((*typecode)[3]='H')

#define mm_clear_typecode(typecode) ((*typecode)[0]=(*typecode)[1]= \
                        (*typecode)[2]=' ',(*typecode)[3]='G')

#define mm_initialize_typecode(typecode) mm_clear_typecode(typecode)


/********************* Matrix Market error codes ***************************/


#define MM_COULD_NOT_READ_FILE    11
#define MM_PREMATURE_EOF        12
#define MM_NOT_MTX              13
#define MM_NO_HEADER            14
#define MM_UNSUPPORTED_TYPE       15
#define MM_LINE_TOO_LONG        16
#define MM_COULD_NOT_WRITE_FILE    17


/********************* Matrix Market internal definitions ********************

   MM_matrix_typecode: 4-character sequence

                ojbect        sparse/       data        storage
                              dense         type        scheme

   string position:     [0]       [1]           [2]         [3]

   Matrix typecode:  M(atrix)  C(oord)       R(eal)       G(eneral)
                             A(array)    C(omplex)   H(ermitian)
                                         P(attern)   S(ymmetric)
                                         I(nteger)    K(kew)

 ***********************************************************************/

#define MM_MTX_STR        "matrix"
#define MM_ARRAY_STR    "array"
#define MM_DENSE_STR    "array"
#define MM_COORDINATE_STR "coordinate"
#define MM_SPARSE_STR    "coordinate"
#define MM_COMPLEX_STR    "complex"
#define MM_REAL_STR       "real"
#define MM_INT_STR        "integer"
#define MM_GENERAL_STR  "general"
#define MM_SYMM_STR       "symmetric"
#define MM_HERM_STR       "hermitian"
#define MM_SKEW_STR       "skew-symmetric"
#define MM_PATTERN_STR  "pattern"


/*  high level routines */
```

```
int mm_write_mtx_crd(char fname[], int M, int N, int nz, int I[], int J[],
        double val[], MM_typecode matcode);
int mm_read_mtx_crd_data(FILE *f, int M, int N, int nz, int I[], int J[],
       double val[], MM_typecode matcode);
int mm_read_mtx_crd_entry(FILE *f, int *I, int *J, double *real, double *img,
          MM_typecode matcode);

int mm_read_unsymmetric_sparse(const char *fname, int *M_, int *N_, int *nz_,
              double **val_, int **I_, int **J_);



#endif
```

mmio.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include "mmio.h"

int mm_read_unsymmetric_sparse(const char *fname, int *M_, int *N_, int *nz_,
              double **val_, int **I_, int **J_)
{
   FILE *f;
   MM_typecode matcode;
   int M, N, nz;
   int i;
   double *val;
   int *I, *J;

   if ((f = fopen(fname, "r")) == NULL)
         return -1;


   if (mm_read_banner(f, &matcode) != 0)
   {
      printf("mm_read_unsymetric: Could not process Matrix Market banner ");
      printf(" in file [%s]\n", fname);
      return -1;
   }



   if ( !(mm_is_real(matcode) && mm_is_matrix(matcode) &&
         mm_is_sparse(matcode)))
   {
      fprintf(stderr, "Sorry, this application does not support ");
```

```
        fprintf(stderr, "Market Market type: [%s]\n",
                mm_typecode_to_str(matcode));
        return -1;
    }

    /* find out size of sparse matrix: M, N, nz .... */

    if (mm_read_mtx_crd_size(f, &M, &N, &nz) !=0)
    {
        fprintf(stderr, "read_unsymmetric_sparse(): could not parse matrix size.\n");
        return -1;
    }

    *M_ = M;
    *N_ = N;
    *nz_ = nz;

    /* reseve memory for matrices */

    I = (int *) malloc(nz * sizeof(int));
    J = (int *) malloc(nz * sizeof(int));
    val = (double *) malloc(nz * sizeof(double));

    *val_ = val;
    *I_ = I;
    *J_ = J;

    /* NOTE: when reading in doubles, ANSI C requires the use of the "l"  */
    /*   specifier as in "%lg", "%lf", "%le", otherwise errors will occur */
    /*  (ANSI C X3.159-1989, Sec. 4.9.6.2, p. 136 lines 13-15)            */

    for (i=0; i<nz; i++)
    {
        fscanf(f, "%d %d %lg\n", &I[i], &J[i], &val[i]);
        I[i]--;  /* adjust from 1-based to 0-based */
        J[i]--;
    }
    fclose(f);

    return 0;
}

int mm_is_valid(MM_typecode matcode)
{
    if (!mm_is_matrix(matcode)) return 0;
    if (mm_is_dense(matcode) && mm_is_pattern(matcode)) return 0;
    if (mm_is_real(matcode) && mm_is_hermitian(matcode)) return 0;
    if (mm_is_pattern(matcode) && (mm_is_hermitian(matcode) ||
            mm_is_skew(matcode))) return 0;
    return 1;
}
```

```
int mm_read_banner(FILE *f, MM_typecode *matcode)
{
    char line[MM_MAX_LINE_LENGTH];
    char banner[MM_MAX_TOKEN_LENGTH];
    char mtx[MM_MAX_TOKEN_LENGTH];
    char crd[MM_MAX_TOKEN_LENGTH];
    char data_type[MM_MAX_TOKEN_LENGTH];
    char storage_scheme[MM_MAX_TOKEN_LENGTH];
    char *p;


    mm_clear_typecode(matcode);

    if (fgets(line, MM_MAX_LINE_LENGTH, f) == NULL)
        return MM_PREMATURE_EOF;

    if (sscanf(line, "%s %s %s %s %s", banner, mtx, crd, data_type,
        storage_scheme) != 5)
        return MM_PREMATURE_EOF;

    for (p=mtx; *p!='\0'; *p=tolower(*p),p++);  /* convert to lower case */
    for (p=crd; *p!='\0'; *p=tolower(*p),p++);
    for (p=data_type; *p!='\0'; *p=tolower(*p),p++);
    for (p=storage_scheme; *p!='\0'; *p=tolower(*p),p++);

    /* check for banner */
    if (strncmp(banner, MatrixMarketBanner, strlen(MatrixMarketBanner)) != 0)
        return MM_NO_HEADER;

    /* first field should be "mtx" */
    if (strcmp(mtx, MM_MTX_STR) != 0)
        return  MM_UNSUPPORTED_TYPE;
    mm_set_matrix(matcode);


    /* second field describes whether this is a sparse matrix (in coordinate
            storgae) or a dense array */


    if (strcmp(crd, MM_SPARSE_STR) == 0)
        mm_set_sparse(matcode);
    else
    if (strcmp(crd, MM_DENSE_STR) == 0)
            mm_set_dense(matcode);
    else
        return MM_UNSUPPORTED_TYPE;


    /* third field */

    if (strcmp(data_type, MM_REAL_STR) == 0)
        mm_set_real(matcode);
```

```
    else
    if (strcmp(data_type, MM_COMPLEX_STR) == 0)
        mm_set_complex(matcode);
    else
    if (strcmp(data_type, MM_PATTERN_STR) == 0)
        mm_set_pattern(matcode);
    else
    if (strcmp(data_type, MM_INT_STR) == 0)
        mm_set_integer(matcode);
    else
        return MM_UNSUPPORTED_TYPE;


    /* fourth field */

    if (strcmp(storage_scheme, MM_GENERAL_STR) == 0)
        mm_set_general(matcode);
    else
    if (strcmp(storage_scheme, MM_SYMM_STR) == 0)
        mm_set_symmetric(matcode);
    else
    if (strcmp(storage_scheme, MM_HERM_STR) == 0)
        mm_set_hermitian(matcode);
    else
    if (strcmp(storage_scheme, MM_SKEW_STR) == 0)
        mm_set_skew(matcode);
    else
        return MM_UNSUPPORTED_TYPE;


    return 0;
}

int mm_write_mtx_crd_size(FILE *f, int M, int N, int nz)
{
    if (fprintf(f, "%d %d %d\n", M, N, nz) != 3)
        return MM_COULD_NOT_WRITE_FILE;
    else
        return 0;
}

int mm_read_mtx_crd_size(FILE *f, int *M, int *N, int *nz )
{
    char line[MM_MAX_LINE_LENGTH];
    int num_items_read;

    /* set return null parameter values, in case we exit with errors */
    *M = *N = *nz = 0;

    /* now continue scanning until you reach the end-of-comments */
    do
    {
```

```
    if (fgets(line,MM_MAX_LINE_LENGTH,f) == NULL)
        return MM_PREMATURE_EOF;
  }while (line[0] == '%');

  /* line[] is either blank or has M,N, nz */
  if (sscanf(line, "%d %d %d", M, N, nz) == 3)
    return 0;

  else
  do
  {
    num_items_read = fscanf(f, "%d %d %d", M, N, nz);
    if (num_items_read == EOF) return MM_PREMATURE_EOF;
  }
  while (num_items_read != 3);

  return 0;
}


int mm_read_mtx_array_size(FILE *f, int *M, int *N)
{
  char line[MM_MAX_LINE_LENGTH];
  int num_items_read;
  /* set return null parameter values, in case we exit with errors */
  *M = *N = 0;

  /* now continue scanning until you reach the end-of-comments */
  do
  {
    if (fgets(line,MM_MAX_LINE_LENGTH,f) == NULL)
        return MM_PREMATURE_EOF;
  }while (line[0] == '%');

  /* line[] is either blank or has M,N, nz */
  if (sscanf(line, "%d %d", M, N) == 2)
    return 0;

  else /* we have a blank line */
  do
  {
    num_items_read = fscanf(f, "%d %d", M, N);
    if (num_items_read == EOF) return MM_PREMATURE_EOF;
  }
  while (num_items_read != 2);

  return 0;
}

int mm_write_mtx_array_size(FILE *f, int M, int N)
{
  if (fprintf(f, "%d %d\n", M, N) != 2)
```

```
        return MM_COULD_NOT_WRITE_FILE;
    else
        return 0;
}



/*-------------------------------------------------------------------------*/

/************************************************************************/
/* use when I[], J[], and val[]J, and val[] are already allocated */
/************************************************************************/

int mm_read_mtx_crd_data(FILE *f, int M, int N, int nz, int I[], int J[],
        double val[], MM_typecode matcode)
{
    int i;
    if (mm_is_complex(matcode))
    {
        for (i=0; i<nz; i++)
            if (fscanf(f, "%d %d %lg %lg", &I[i], &J[i], &val[2*i], &val[2*i+1])
                != 4) return MM_PREMATURE_EOF;
    }
    else if (mm_is_real(matcode))
    {
        for (i=0; i<nz; i++)
        {
            if (fscanf(f, "%d %d %lg\n", &I[i], &J[i], &val[i])
                != 3) return MM_PREMATURE_EOF;

        }
    }

    else if (mm_is_pattern(matcode))
    {
        for (i=0; i<nz; i++)
            if (fscanf(f, "%d %d", &I[i], &J[i])
                != 2) return MM_PREMATURE_EOF;
    }
    else
        return MM_UNSUPPORTED_TYPE;

    return 0;

}

int mm_read_mtx_crd_entry(FILE *f, int *I, int *J,
        double *real, double *imag, MM_typecode matcode)
{
    if (mm_is_complex(matcode))
    {
            if (fscanf(f, "%d %d %lg %lg", I, J, real, imag)
```

```
            != 4) return MM_PREMATURE_EOF;
    }
    else if (mm_is_real(matcode))
    {
        if (fscanf(f, "%d %d %lg\n", I, J, real)
            != 3) return MM_PREMATURE_EOF;

    }

    else if (mm_is_pattern(matcode))
    {
        if (fscanf(f, "%d %d", I, J) != 2) return MM_PREMATURE_EOF;
    }
    else
        return MM_UNSUPPORTED_TYPE;

    return 0;

}


/************************************************************************
    mm_read_mtx_crd()  fills M, N, nz, array of values, and return
                        type code, e.g. 'MCRS'

                        if matrix is complex, values[] is of size 2*nz,
                            (nz pairs of real/imaginary values)
************************************************************************/

int mm_read_mtx_crd(char *fname, int *M, int *N, int *nz, int **I, int **J,
        double **val, MM_typecode *matcode)
{
    int ret_code;
    FILE *f;

    if (strcmp(fname, "stdin") == 0) f=stdin;
    else
    if ((f = fopen(fname, "r")) == NULL)
        return MM_COULD_NOT_READ_FILE;


    if ((ret_code = mm_read_banner(f, matcode)) != 0)
        return ret_code;

    if (!(mm_is_valid(*matcode) && mm_is_sparse(*matcode) &&
            mm_is_matrix(*matcode)))
        return MM_UNSUPPORTED_TYPE;

    if ((ret_code = mm_read_mtx_crd_size(f, M, N, nz)) != 0)
        return ret_code;
```

```
    *I = (int *)  malloc(*nz * sizeof(int));
    *J = (int *)  malloc(*nz * sizeof(int));
    *val = NULL;

    if (mm_is_complex(*matcode))
    {
        *val = (double *) malloc(*nz * 2 * sizeof(double));
        ret_code = mm_read_mtx_crd_data(f, *M, *N, *nz, *I, *J, *val,
                *matcode);
        if (ret_code != 0) return ret_code;
    }
    else if (mm_is_real(*matcode))
    {
        *val = (double *) malloc(*nz * sizeof(double));
        ret_code = mm_read_mtx_crd_data(f, *M, *N, *nz, *I, *J, *val,
                *matcode);
        if (ret_code != 0) return ret_code;
    }

    else if (mm_is_pattern(*matcode))
    {
        ret_code = mm_read_mtx_crd_data(f, *M, *N, *nz, *I, *J, *val,
                *matcode);
        if (ret_code != 0) return ret_code;
    }

    if (f != stdin) fclose(f);
    return 0;
}

int mm_write_banner(FILE *f, MM_typecode matcode)
{
    char *str = mm_typecode_to_str(matcode);
    int ret_code;

    ret_code = fprintf(f, "%s %s\n", MatrixMarketBanner, str);
    free(str);
    if (ret_code !=2 )
        return MM_COULD_NOT_WRITE_FILE;
    else
        return 0;
}

int mm_write_mtx_crd(char fname[], int M, int N, int nz, int I[], int J[],
        double val[], MM_typecode matcode)
{
    FILE *f;
    int i;

    if (strcmp(fname, "stdout") == 0)
        f = stdout;
    else
```

```
    if ((f = fopen(fname, "w")) == NULL)
        return MM_COULD_NOT_WRITE_FILE;

    /* print banner followed by typecode */
    fprintf(f, "%s ", MatrixMarketBanner);
    fprintf(f, "%s\n", mm_typecode_to_str(matcode));

    /* print matrix sizes and nonzeros */
    fprintf(f, "%d %d %d\n", M, N, nz);

    /* print values */
    if (mm_is_pattern(matcode))
        for (i=0; i<nz; i++)
            fprintf(f, "%d %d\n", I[i], J[i]);
    else
    if (mm_is_real(matcode))
        for (i=0; i<nz; i++)
            fprintf(f, "%d %d %20.16g\n", I[i], J[i], val[i]);
    else
    if (mm_is_complex(matcode))
        for (i=0; i<nz; i++)
            fprintf(f, "%d %d %20.16g %20.16g\n", I[i], J[i], val[2*i],
                    val[2*i+1]);
    else
    {
        if (f != stdout) fclose(f);
        return MM_UNSUPPORTED_TYPE;
    }

    if (f !=stdout) fclose(f);

    return 0;
}


/**
*  Create a new copy of a string s.  mm_strdup() is a common routine, but
*  not part of ANSI C, so it is included here.  Used by mm_typecode_to_str().
*
*/
char *mm_strdup(const char *s)
{
    int len = strlen(s);
    char *s2 = (char *) malloc((len+1)*sizeof(char));
    return strcpy(s2, s);
}

char  *mm_typecode_to_str(MM_typecode matcode)
{
    char buffer[MM_MAX_LINE_LENGTH];
    char *types[4];
    char *mm_strdup(const char *);
```

```
    int error =0;

    /* check for MTX type */
    if (mm_is_matrix(matcode))
        types[0] = MM_MTX_STR;
    else
        error=1;

    /* check for CRD or ARR matrix */
    if (mm_is_sparse(matcode))
        types[1] = MM_SPARSE_STR;
    else
    if (mm_is_dense(matcode))
        types[1] = MM_DENSE_STR;
    else
        return NULL;

    /* check for element data type */
    if (mm_is_real(matcode))
        types[2] = MM_REAL_STR;
    else
    if (mm_is_complex(matcode))
        types[2] = MM_COMPLEX_STR;
    else
    if (mm_is_pattern(matcode))
        types[2] = MM_PATTERN_STR;
    else
    if (mm_is_integer(matcode))
        types[2] = MM_INT_STR;
    else
        return NULL;


    /* check for symmetry type */
    if (mm_is_general(matcode))
        types[3] = MM_GENERAL_STR;
    else
    if (mm_is_symmetric(matcode))
        types[3] = MM_SYMM_STR;
    else
    if (mm_is_hermitian(matcode))
        types[3] = MM_HERM_STR;
    else
    if (mm_is_skew(matcode))
        types[3] = MM_SKEW_STR;
    else
        return NULL;

    sprintf(buffer,"%s %s %s %s", types[0], types[1], types[2], types[3]);
    return mm_strdup(buffer);

}
```