Отчёт по лабораторной работе №3 "Обработка разреженных матриц", Вариант 3

Выполнил: Калашков Павел ИУ7-36Б, Вариант 7 (по журналу)

Описание условия задачи

Цель работы - реализовать алгоритмы обработки разреженных матриц, сравнить эффективность использования этих алгоритмов (по времени выполнения и по требуемой памяти) со стандартными алгоритмами обработки матриц при различном процентном заполнении матриц ненулевыми значениями и при различных размерах матриц.

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов: - вектор A содержит значения ненулевых элементов; - вектор JA содержит номера столбцов для элементов вектора A; - связный список IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A.

1. Смоделировать операцию умножения матрицы и вектора-столбца, хранящихся в этой форме, с получением результата в той же форме.
2. Произвести операцию умножения, применяя стандартный алгоритм работы с матрицами.
3. Сравнить время выполнения операций и объем памяти при использовании этих 2-х алгоритмов при различном проценте заполнения матриц

Техническое задание

Исходные данные

Исходными данными для программы является данные, введённые с консоли (см. ниже);

Данные, введённые с консоли, связаны с меню программы:

```
Возможные действия:
 1 - Ввести данные вручную
 2 - Сгенерировать данные
 3 - Умножить матрицу на столбец
 4 - Изменить размер матрицы и столбца
 5 - Исследовать время работы программы и используемый объём памяти
 0 - Выйти из программы
Введите номер действия:
```

Поэтому введённые данные являются **строкой**, которая может содержать в себе номер действия, а также входные данные для конкретных действий (см. ниже)

Результаты

Выходными данными являются данные в консоли (см. ниже);

Данными в консоли являются: - ответ программы на команду (сообщение об успешном / неуспешном выполнении, результаты действия команды); - меню программы;

Задача, реализуемая программой

Задачей программы является реализация операции умножения матрицы на столбец, при этом должна быть возможность: - вводить матрицу и столбец

вручную; - генерировать матрицу и столбец, задавая их заполненность; - вычислять результат умножения текущей матрицы на столбец; - исследовать время работы и объём используемой памяти;

Способ обращения к программе

Обращение к программе происходит посредством запуска исполняемого файла app.exe

Возможные аварийные ситуации и ошибки пользователя

Аварийной ситуацией является:

- несоблюдение формата входных данных (см. исходные данные);

Описание внутренних структур данных

Для реализации деления и обработки больших чисел были реализованы две структуры данных:

```
typedef struct
{
    short is_a_defined; // определена ли матрица в обычном виде
    short is_a_crs_defined; // определена ли матрица в разреженном виде
    short is_b_defined; // определён ли столбец в обычном виде
    short is_b_ccs_defined; // определён ли столбец в разреженном виде
    short is_c_defined; // определён ли результат в обычном виде
    short is_c_ccs_defined; // определён ли результат в разреженном виде
    int **matrix; // матрица в обычном виде
    int n; // размерность n матрицы
    int m; // размерность m матрицы
    int matrix_meaning; // количество значащих элементов в матрице
    int ia_length; // длина массива компонент ia
    int *a; // массив ненулевых значений в матрице
    int *ja; // массив, содержащий номера столбцов для элемента массива a
    node *ia; // связный список, в элемента nk которого находится номер компонент
    int *column; // масссив для элементов столбца
    int column_length; // длина этого массива
    int column_meaning; // количество значащих элементов строки
    int *b; // массив ненулевых значений в столбце
    int *jb; // массив, содержащий индексы соответствующих ненулевых значений
    int *c_column; // массив для элементов результата
    int c_column_length; // длина этого масссива
    int *c; // массив ненулевых значений столбца-результата
    int *jc; // массив, содержащий индексы соответствующих ненулевых значений
    int c_meaning; // количество ненулевых элементов в результате
} info;

typedef struct node
{
    int value; // значение
    struct node *next; // указатель на следующий
} node;
```

Данные структуры используются для обработки и хранения чисел, выходящих за разрядную сетку персонального комьютера.

Описанный алгоритм

Алгоритм делится на четыре главных части:

- 1 - вывод меню;
- 2 - запрос действия из меню;
- 3 - выполнение действия (если оно указано корректно);
- 4 - повтор, если действие - не выход из программы;

Вывод меню

Меню имеет формат:

```
Возможные действия:
 1 - Ввести данные вручную
 2 - Сгенерировать данные
 3 - Умножить матрицу на столбец
 4 - Изменить размер матрицы и столбца
 5 - Исследовать время работы программы и используемый объём памяти
 0 - Выйти из программы
Введите номер действия:
```

Запрос действия из меню

Программа запрашивает номер действия как строку, считывает и проверяет на корректность.

Если корректно, то переводит строку в число, иначе выводит сообщение об ошибке.

Выполнение действия

- 1 - Прочитать матрицу A и столбец B из консоли, при этом запрашивается количество ненулевых элементов, также их индексы и значение. Между индексами вводится пробел;
- 2 - Запросить у пользователя процент заполнения столбца, выполнить, запросить процент заполнения столбца, выполнить.
- 3 - Запросить у пользователя способ, которым он хочет произвести матричное умножение, умножить, вывести исходные данные и результат;
- 4 - Запросить у пользователя размеры матрицы и столбца, выделить память на матрицу и столбец;
- 5 - Произвести замеры времени и используемой памяти на введённых матрице и столбце, вывести результат;
- 0 - Выйти из программы: вывести сообщение о завершении программы;

Сравнение алгоритмов

Сравним время работы и объём используемой памяти для используемых алгоритмов при умножении матрицы A размерностью N*N на столбец длины N, заполненность равна p.

Примечание: CRS - Compressed Row Storage, разреженный строчный формат

Результат каждого измерения усреднён, было проведено 1000 измерений при каждом значении N и p;

Возьмём N = 50:

N	p, %	Время выполнения обычного умножения, мкс	Память, затраченная при обычном умножении, байты	Время выполнения умножения методом CRS, мкс	Память, затраченная при выполнении умножения методом CRS, байты
50	10	21	10000	12	1408
50	20	25	10000	14	4400
50	30	29	10000	30	6400
50	40	31	10000	42	8400
50	50	35	10000	63	10500
50	60	35	10000	77	12520
50	70	36	10000	91	14540
50	80	34	10000	103	16560
50	80	35	10000	117	18580
50	100	34	10000	132	20600

При N = 100:

N	p, %	Время выполнения обычного умножения, мкс	Память, затраченная при обычном умножении, байты	Время выполнения умножения методом CRS, мкс	Память, затраченная при выполнении умножения методом CRS, байты
100	10	11	40000	5	8840
100	20	61	40000	61	16880

N	p, %	Время выполнения обычного умножения, мкс	Память, затраченная при обычном умножении, байты	Время выполнения умножения методом CRS, мкс	Память, затраченная при выполнении умножения методом CRS, байты
100	30	62	40000	93	24920
100	40	70	40000	133	32960
100	50	55	40000	163	41000
100	60	58	40000	184	49040
100	70	56	40000	219	57080
100	80	46	40000	209	65120
100	90	47	40000	222	73160
100	100	51	40000	293	81200

При N = 500:

N	p, %	Время выполнения обычного умножения, мкс	Память, затраченная при обычном умножении, байты	Время выполнения умножения методом CRS, мкс	Память, затраченная при выполнении умножения методом CRS, байты
500	1	1171	1000000	172	24020
500	2	1018	1000000	422	44040
500	3	1043	1000000	426	64060
500	4	1042	1000000	621	84080
500	5	994	1000000	706	104100
500	6	943	1000000	797	124120
500	7	938	1000000	914	144140
500	8	956	1000000	1025	164160
500	9	959	1000000	1174	184180
500	10	912	1000000	1273	204200
500	100	923	1000000	13434	2006000

Видим, что использование разреженного строчного формата помогает оптмизировать работу с матрицами, если количество их ненулевых элементов, выраженное в форме N^(1 + g), не превышает N^(1.5), т.е g <= 0.5. Так, в среднем случае использование разреженного строчного формата помогает получить прирост к времени выполнения в размере 600 - 700 %, уменьшая при этом затраты исползуемоей памяти в 2000 - 4000 %.

Заметим однако, что при g > 0.5 использование разреженного строчного формата становится неэффективно по времени, а начиная с процента заполненности матрицы, равного 50%, использование CRS становится невыгодно также и по памяти.

Теоретическая часть

1. Что такое разреженная матрица, какие схемы хранения таких матриц Вы знаете?

Разреженная матрица - матрица с преимущественно нулевыми элементами. Обычно матрица считается разреженной, если количество её ненулевых элементов представимо в виде n^(1 + g), где n - порядок матрицы, g < 1

Известные способы хранения разреженных матриц:

Связная схема Кнута: хранить в массиве в произвольном порядке сами элементы, индексы строк и столбцов соответствующих элементов, номер следующего ненулевого элемента, расположенного в матрице по строке (и по столбцу), а также номера элементов, с которых начинается строка (и столбец).

Разреженный строчный формат (CRS, метод Чанга и Густавсона): хранить в массиве значения ненулевых элементов, соответствующие им столбцовые индексы, а также массив указателей, отмечающих позиции элемента, с которого начинается описание очередной строки.

2. Каким образом и сколько памяти выделяется под хранение разреженной и обычной матрицы?

Объём памяти, выделяемой под хранение разреженной матрицы, пропорционально числу ненулевых элементов данной матрицы, в то время как объём памяти, выделяемой под хранение обычной матрицы пропорционален общему числу элементов. Так, в случае представления разреженной матрицы при помощи разреженного строчного формата, выделяется память под значения элементов, индексы их столбцов, а также индексы элементов, являющихся первыми в своей строке.

3. Каков принцип обработки разреженной матрицы?

Алгоритмы обработки разреженных матриц подразумевают действия только с ненулевыми элементами, таким образом, время работы данных алгоритмов пропорционально количеству ненулевых элементов данных матриц.

4. В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?

Стандартные алгоритмы обработки матриц следует применять в случае, когда матрицы не являются разреженными, т.е. при представлении числа их ненулевых элементов в виде n^(1 + g), где n - порядок матрицы, g > 1. Таким образом, выбор алгоритма зависит от порядка матрицы и от количества её ненулевых элементов.

Выводы по проделанной работе

В случае, когда обрабатываемые матрицы являются разреженными, а их обработка затрагивает только ненулевые элементы, матрицы стоит хранить и обрабатывать при помощи разреженного строчного формата или одного из других способов, предназначенных для обработки разреженных матриц. Данные методы помогут не только сократить время работы программы, но и сократить объём используемой памяти. Однако при использовании таких алгоритмов стоит быть осторожнее, т.к. если применить их к обычным матрицам, затраты по времени и по памяти возрастут многократно.