

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ М. В. ЛОМОНОСОВА  
ФИЛИАЛ МОСКОВСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА  
ИМЕНИ М. В. ЛОМОНОСОВА В ГОРОДЕ САРОВЕ



## **Параллельные методы решения задач**

Лабораторная работа:

«Генерация разреженной матрицы в формате CSR, используя сеточные  
данные на неструктурированной смешанной сетке»

Выполнил студент  
1-го курса магистратуры  
Козлов Н.М.  
Группа: ВМ-124  
Дата подачи: 12.03.2025  
Вариант: А2

Саров 2024

## 2.1. Описание задачи

Цель первого задания: освоение работы с CSR форматом.

Поэтому нужно реализовать:

- 1) генерацию тестовой смешанно-элементной сетки и построение смежности ячеек в формате CSR.
- 2) заполнение матрицы

### 1. Генерация сетки и построение смежности.

Двумерная сетка строится на основе решетки, размер  $N_x \times N_y$ . Нумерация узлов и клеток слева направо, сверху вниз. Сеточные элементы - треугольники или четырехугольники. Клетки решетки делятся или не делятся на треугольники в соотношении, определяемом параметрами  $k_1, k_2$ .  $k_1$  клеток не делим на треугольники (оставляем четырехугольниками),  $k_2$  клеток делим, и так по всем клеткам. Итак, мы получили топологию сетки, теперь на основе топологии элементов-узлов надо построить топологию смежности через узлы или элементы в зависимости от варианта.

На выходе должны получить описание смежности сетки + диагональ в формате CSR (то есть массивы IA и JA).

### 2. Заполнение матрицы.

Для ненулевых недиагональных элементов матрицы по некоторой формуле задаем значения, например, по такой:  $a_{ij} = \cos(i*j+i+j)$ . Для диагональных элементов значение задается суммированием недиагональных элементов строки, умноженных на константу 1.234. Также создаем вектор правой части  $b$ . На выходе получаем к уже имеющимся IA и JA третий массив значений  $A$  и плотный вектор  $b$ .

## 2.2 Описание программной реализации

Для решения задания были созданы следующие функции:

```
//Красивый вывод IA, JA
void print_csr(int* IA, int* JA, int N
//Постный вывод IA, JA
//Вдруг потом в файл надо будет вывести и дальше как-то с данными работать)
void print_clear_csr(int* IA, int* JA, int N)
//Красивый вывод матрицы, хранящейся в CSR формате с правой частью для решения СЛАУ
void print_full_csr(int* IA, int* JA, int N, double* A, double* b)
//Постный вывод матрицы, хранящейся в CSR формате с правой частью для решения СЛАУ
//Вдруг потом в файл надо будет вывести и дальше как-то с данными работать)
void print_clear_full_csr(int* IA, int* JA, int N, double* A, double* b)
//Вывод информации о расходе памяти
int memoryUsage()
//Вывод информации о всей существующей оперативной памяти
```

```

int memoryExist()
//Генерация JA, IA массивов по заданным параметрам сетки
void generate(int Nx, int Ny, int k1, int k2, int& N, int*& IA, int*& JA)
//Заполнение матрицы A и правой части b по заданным формулам в CSR формате
void fill(int* IA, int* JA, int N, double*& A, double*& b)

```

Первые 4 функции представляют собой различные вариации вывода информации, чтобы было красиво/практично. Для вывода информации при запросе пользователя в итоговом варианте использована функция `print_full_csr()`. Входными параметрами этих функций являются данные CSR матрицы, сгенерированные при создании массивов JA/IA или A/b.

Функция `memoryUsage()` использована для определения расхода памяти и работает только на UNIX системах. Она парсит данные из файла `/proc/self/status`, а именно 23 строчку – `VmRSS`, размер резидентной памяти в данный момент.

Функция `memoryExist()` использована для определения всей существующей оперативной памяти. Это сделано для того, чтобы предотвратить вызов `SGKILL` при вводе слишком больших параметров сетки, настолько, что они не влезают во всю доступную память. Пример применения защиты от супердурака представлен ниже:

```

JA = new int[SIZE]; //номера столбцов ненулевых элементов

if(memoryUsage() > (memoryExist() / 2)){
    delete[] JA;
    std::cout << "Too much memory allocated during malloc JA\n";
    exit(1);
}

```

Функция `generate(int Nx, int Ny, int k1, int k2, int& N, int*& IA, int*& JA)` реализует первую часть задания, а именно по заданным параметрам сетки создание JA и IA массивов матрицы смежности полученного графа.

При этом эмпирически выявлена формулы определения размеров этих массивов, которые определяются следующим образом:

```

int SIZE = (Nx + 1) * (Ny + 1) + //самовхождения (число ячеек_x+1)*(число ячеек_y+1) +...
2 * (Nx * (Ny + 1) + //горизонтальные рёбра (число ячеек_x)*(число ячеек_y+1) +...
(Nx + 1) * Ny + //вертикальные рёбра (число ячеек_x+1)*(число ячеек_y) +...
(Nx * Ny) / (k1 + k2) * k2 + //количество целых (полных по k2) косых рёбер
((Nx * Ny) % (k1 + k2) > k1 ? (Nx * Ny) % (k1 + k2) - k1 : 0)); //количество
последних косых рёбер

JA = new int[SIZE]; //номера столбцов ненулевых элементов
N = (Ny + 1) * (Nx + 1);
IA = new int[N + 1]; //информация о позиции начала списка столбцов данной строки

```

Определяется это так: размер массива JA – это самовхождения узлов (главная диагональ матрицы смежности) + 2\*количество рёбер (поскольку матрица симметрична), размер массива IA для заданного варианта равен количеству узлов сетки + 1 (для хранения информации о всех ненулевых элементах (`IA[N] = SIZE;`))

Далее идёт инициализация значений обеих матриц. Цикл построчно пробегается по всей матрице смежности. Последний индекс столбца, записанный в массив JA, будет сохранён в массив IA для заданного узла. Поскольку порядок слева

направо и сверху вниз, то ненулевые столбцы в каждой строке упорядочены по следующему правилу:  $\nwarrow$ ,  $\uparrow$ ,  $\leftarrow$ ,  $\bullet$ ,  $\rightarrow$ ,  $\downarrow$ ,  $\searrow$ , то есть, если есть сосед, находящийся по направлению стрелки, то индекс столбца попадает в массив JA. Код, соответствующий логике описанного алгоритма, представлен ниже:

```
int I_node, I;
int indecies = 0, kk = k1 + k2;
for (int i = 0; i <= Ny; ++i) {
    for (int j = 0; j <= Nx; ++j) {
        I_node = i * (Nx+1) + j; //Индекс узла в сетке
        I = i * Nx + j; //Абсолютный индекс ячейки в сетке
        IA[I_node] = indecies;
        if (i && j && (I - Nx - 1) % kk >= k1)
            JA[indecies++] = I_node - Nx - 2; //  $\nwarrow$  влево-вверх
        if (i) //Верхний сосед есть у всех узлов, кроме первого слоя сетки по i
            JA[indecies++] = I_node - Nx - 1; //  $\uparrow$  вверх
        if (j) //Левый сосед есть у всех узлов, кроме первого слоя сетки по j
            JA[indecies++] = I_node - 1; //  $\leftarrow$  влево
        JA[indecies++] = I_node; // самовхождение
        if (Nx - j) //Правый сосед есть у всех узлов, кроме последнего слоя сетки по j
            JA[indecies++] = I_node + 1; //  $\rightarrow$  вправо
        if (Ny - i) //Нижний сосед есть у всех узлов, кроме последнего слоя сетки по i
            JA[indecies++] = I_node + Nx + 1; //  $\downarrow$  вниз
        if (Nx - j && Ny - i && (I % kk >= k1))
            JA[indecies++] = I_node + Nx + 2; //  $\searrow$  вправо-вниз
    }
}
```

На вход функция требует параметры сетки Nx, Ny, K1, K2 и ссылки на данные, которые она получит в ходе работы. В результате работы функции в созданные в main() переменные N, JA, IA запишутся корректные данные о CSR формате хранения матрицы.

Функция `void fill(int* IA, int* JA, int N, double*& A, double*& b)` реализует вторую часть задания.

Обход матрицы A реализован аналогично функциям вывода. Инициализация данных происходит согласно формулам, указанным в задании.

Важным замечанием является то, что поскольку заранее неизвестно, где находится диагональный элемент в строке, значение которого зависит от всех остальных элементов строки, его инициализация происходит только после обхода всей строки, остальные же инициализируются по ходу обхода.

Раскрутки циклов так и не сделал, руки не дошли, однако однозначно можно её проверить с инициализацией вектора b и, возможно, и массива A.

Для определения времени использовалась функция `omp_get_wtime()` из пакета OpenMP.

Также, согласно требованиям задания, была проделана защита от дурака на входных параметрах и даже оформлена вежливая просьба с объяснением,

как использовать программу верно. На рисунке 1 представлен результат запуска программы без входных аргументов.

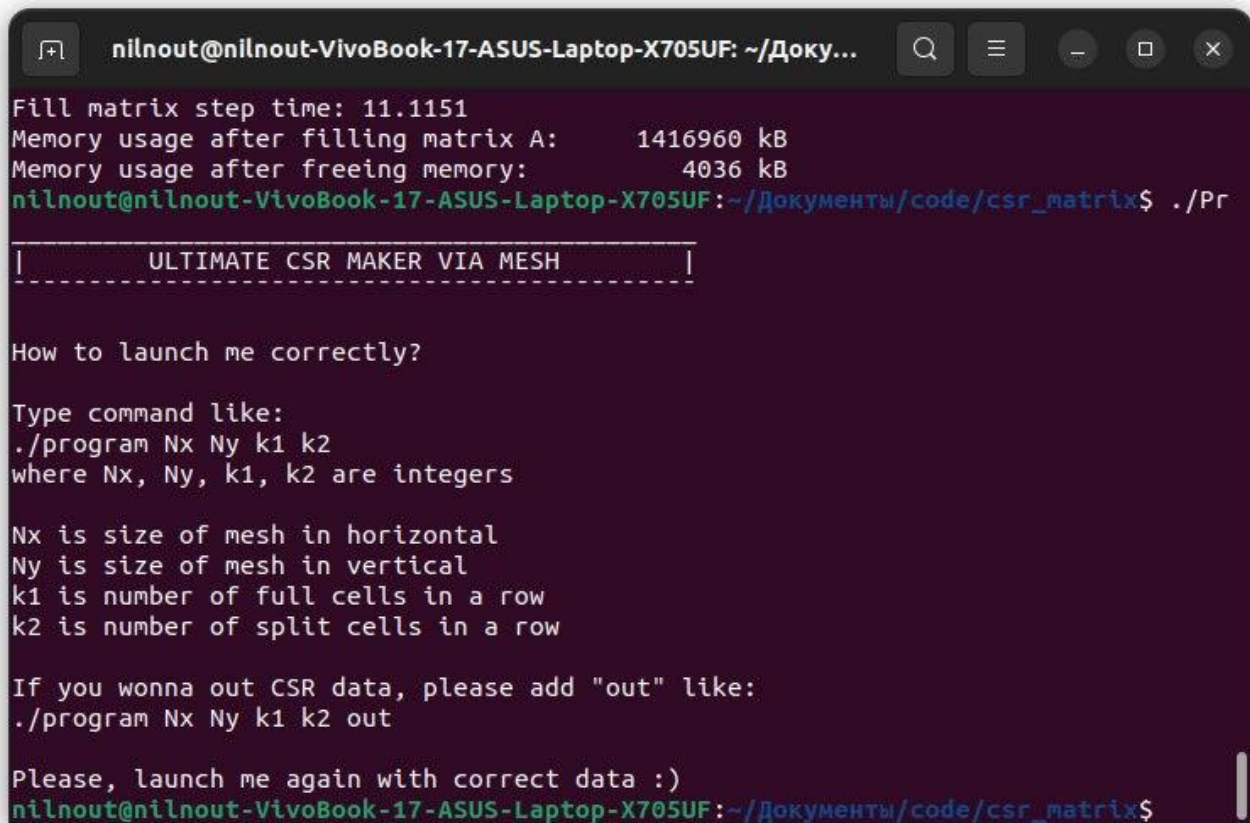
При корректно введенных данных программа выдаёт отчёт, содержащий данные о потраченном на оба задания времени и резидентной памяти, что продемонстрировано на рисунке 2. А если пользователь добавит флаг 'out', то выведется вся CSR матрица, что продемонстрировано на рисунке 3.

При некорректно введенных данных программа выдаёт разного рода предупреждения с просьбой опомниться и сделать всё правильно. Прошу не сильно мучать мой код тестами, поскольку функция `stoi`, а также её родители и прадеды, `strtol` и `atoi`, неидеальны в своих попытках корректно сконвертировать некорректные данные, и отбрасывают некорректность после чисел, а писать свою пользовательскую функцию не хватило сил, но на что-то `stoi` честно выбрасывает исключения, которые я честно обрабатываю.

Для компиляции программы необходимо иметь компилятор `g++` (устанавливается следующей командой: `sudo apt install build-essential`).

Компиляция происходит с помощью следующей команды (при терминале, открытом в папке с файлом программы):

```
g++ main.cpp -o program -fopenmp
```



```
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF: ~/Доку...
Fill matrix step time: 11.1151
Memory usage after filling matrix A:      1416960 kB
Memory usage after freeing memory:        4036 kB
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$ ./Pr

|-----|
|  ULTIMATE CSR MAKER VIA MESH  |
|-----|

How to launch me correctly?

Type command like:
./program Nx Ny k1 k2
where Nx, Ny, k1, k2 are integers

Nx is size of mesh in horizontal
Ny is size of mesh in vertical
k1 is number of full cells in a row
k2 is number of split cells in a row

If you wanna out CSR data, please add "out" like:
./program Nx Ny k1 k2 out

Please, launch me again with correct data :)
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$
```

Рисунок 1 – Запуск программы без аргументов



```
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF: ~/Доку...
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$ g++
main.cpp -o Pr -fopenmp
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$ ./Pr
1000 3000 300 4
Memory usage before generating:          3456 kB
Generation step time: 0.0974217
Memory usage after generating:          74496 kB
Fill matrix step time: 1.64523
Memory usage after filling matrix A:      216064 kB
Memory usage after freeing memory:        4116 kB
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$ ./Pr
10000 3000 300 4
Memory usage before generating:          3456 kB
Generation step time: 0.961477
Memory usage after generating:          710144 kB
Fill matrix step time: 16.5813
Memory usage after filling matrix A:      2123264 kB
Memory usage after freeing memory:        4084 kB
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$
```

Рисунок 2 – Запуск программы с параметрами

```
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF: ~/Докум...
Memory usage after freeing memory:          3968 kB
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$ ./Pr
2 2 3 4 out
Memory usage before generating:          3456 kB
Generation step time: 1.564e-06
Memory usage after generating:          3712 kB
Fill matrix step time: 2.6615e-05
Memory usage after filling matrix A:      3968 kB

CSR FORMAT MATRIX
N: 9

A Matrix: {1.88838, 0.540302, -0.989992 | 0.540302, 2.14111, 0.283662, -0.91113 |
0.283662, 0.689591, -0.275163 | -0.989992, 2.80221, 0.988705, -0.292139 | -0.911
13, 0.988705, 4.83034, -0.748058, 0.266643, 0.999843 | -0.275163, -0.748058, 2.39
582, -0.918283 | -0.292139, 0.387804, 0.0221268 | 0.266643, 0.0221268, 0.737676,
-0.309023 | 0.999843, -0.918283, -0.309023, 2.7483}

JA Matrix: {0, 1, 3 | 0, 1, 2, 4 | 1, 2, 5 | 0, 3, 4, 6 | 1, 3, 4, 5, 7, 8 | 2, 4
, 5, 8 | 3, 6, 7 | 4, 6, 7, 8 | 4, 5, 7, 8}

IA Matrix:
{0, 3, 7, 10, 14, 20, 24, 27, 31, 35}

b:
{0, 0.841471, 0.909297, 0.14112, -0.756802, -0.958924, -0.279415, 0.656987, 0.989
358}

Memory usage after freeing memory:          3968 kB
nilnout@nilnout-VivoBook-17-ASUS-Laptop-X705UF:~/Документы/code/csr_matrix$
```

Рисунок 3 – Запуск программы с флагом ‘out’

### 3.1 Описание компьютерной платформы

Intel(R) Core(TM) i3-7100U CPU @ 2.40GHz, 2 ядра, RAM 12 ГБ, Windows 10 Pro.

TPP=2,4 GHz \*2 cores \*8 IPS (4 via AVX2 \* 2 via FMA3) =38.4GFLOPS

BW=2,4 GHz \*8Б\*1=19.2GB/c

### 3.2 Результат вычислительных экспериментов

Для проверки линейного изменения расхода памяти и времени закрепим один из параметров сетки и будем варьировать второй, поскольку варьирование сразу двух параметров приведёт к квадратичной зависимости.

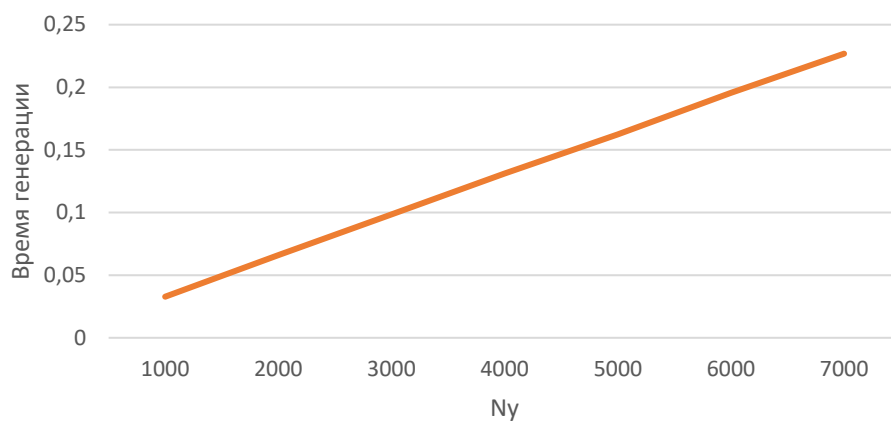
Пусть  $N_x=1000$ . Тогда зависимость времени и расхода памяти в обоих заданиях от значения параметра  $N_y$  указана в таблице 1.

Таблица 1 – Зависимость времени и расхода памяти в обоих заданиях от значения параметра  $N_y$

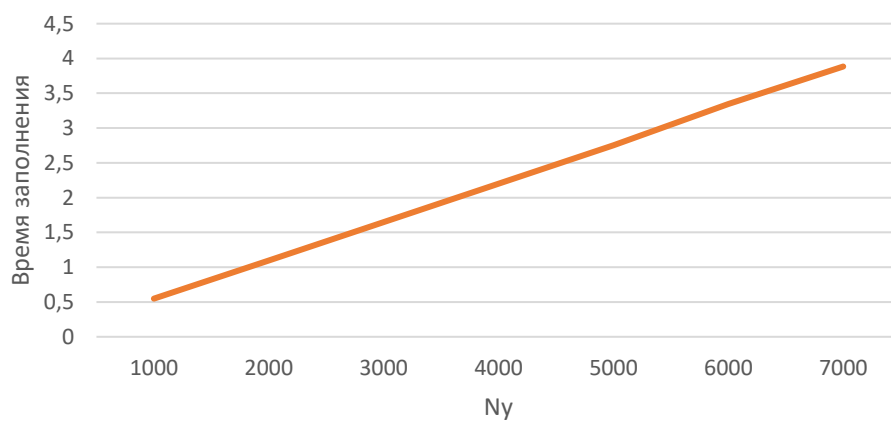
$N_y$	Generation time	Fill time	Generation memory (MB)	Fill memory (MB)
1000	0,0328084	0,548261	26,625	73,125
2000	0,0660009	1,09651	49,625	142,125
3000	0,0985329	1,64745	72,75	210,875
4000	0,131173	2,19989	95,75	280,125
5000	0,162288	2,75078	118,75	349
6000	0,195623	3,34684	141,625	418,125
7000	0,226828	3,88415	164,75	487,125

Эта зависимость продемонстрирована следующими графиками.

Зависимость времени генерации от размеров  
сетки при  $N_x=1000$

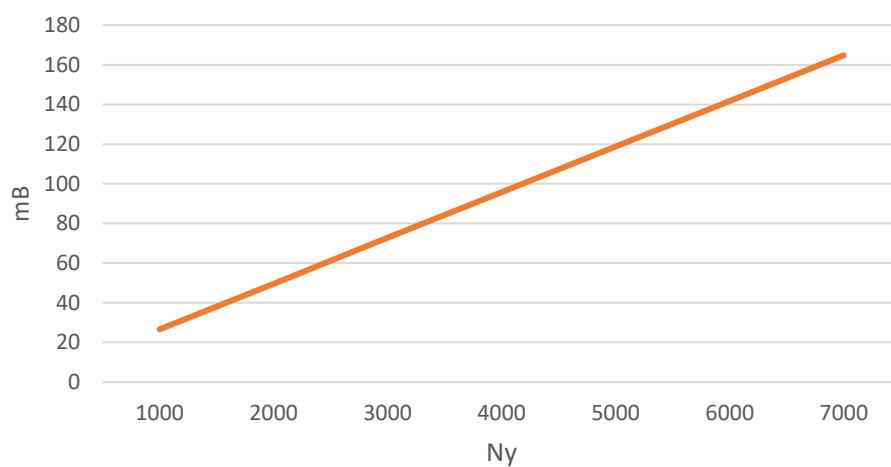


Зависимость времени заполнения от размеров  
сетки при  $N_x=1000$





Расход памяти при генерации при Nx=1000



Расход памяти при заполнении при Nx=1000

