**ФГБОУ ВО   
Уфимский университет науки и технологий**

**Кафедра ВМиК**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Разработка параллельной версии операции перемножения разреженной матрицы на вектор

**ОТЧЕТ**

**по курсовому проекту**

**по** Технологиям параллельного программирования

(*наименование дисциплины*)

|  |
| --- |
| Курсовой проект |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия, И., О. | Подпись | Дата | Оценка |
| МО-325Б |  |
|  |  |
| Студент | | | Шарыгин М.С. |  |  |  |
| Преподаватель | | | Спеле В.В. |  |  |  |
| Принял | | |  |  |  |  |

**Уфа 2025 г****.**

Содержание

[1 Введение 3](#_Toc197035246)

[1.1 Цель работы 3](#_Toc197035247)

[1.2 Разреженные матрицы 4](#_Toc197035248)

[1.3 Форматы разреженных матриц 4](#_Toc197035249)

[2 Основная часть 8](#_Toc197035250)

[2.1 Установка программного обеспечения 8](#_Toc197035251)

[2.2 Разработка программы 9](#_Toc197035252)

[2.3 Расчёт производительности 22](#_Toc197035253)

[2.4 Профили выполнения 23](#_Toc197035254)

[3 Вывод 24](#_Toc197035255)

[4 Список литературы 25](#_Toc197035256)

[5 Приложение 26](#_Toc197035257)

[5.1 Код программы 26](#_Toc197035258)

# Введение

## Цель работы

В ходе курсового проекта необходимо выполнить следующие задания:

1. Скачать архив с матрицей в соответствии с вариантом в формате Matrix Market. Распаковать архив.
2. Реализовать считывание матрицы из mtx-файла с помощью вспомогательного кода.
3. Проанализировать структуру разреженной матрицы, выбрать наиболее эффективный формат хранения, реализовать конвертацию матрицы в выбранный формат.
4. Реализовать последовательную версию перемножения разреженной матрицы на вектор, состоящий из единиц, для выбранного формата хранения.
5. Установить нужное ПО в соответствии с вариантом.  
   («CPU»: «OneAPI BaseKit», «OneAPI HPCKit»; «GPU»: «CUDA»).
6. Реализовать параллельную версию программы, проверить корректность выполнения.
7. Снять и проанализировать профили выполнения программ. По возможности оптимизировать программу.
8. Подключить в проект библиотеку по варианту и реализовать версию перемножения матрицы на вектор с использованием этой библиотеки. Если библиотека не поддерживает наилучший формат хранения – использовать для этого пункта такой формат, который поддерживается библиотекой.
9. Провести замеры времени выполнения для собственного варианта и библиотечного метода. Рассчитать реальную производительность операции умножения матрицы на вектор.
10. Выгрузить векторы результата в файлы для своего и библиотечного варианта перемножения матрицы на вектор. Названия файлов должны указывать на ФИО студента и имя матрицы. Загрузить эти файлы для проверки в облако.

## Разреженные матрицы

В численном анализе и научных вычислениях разреженная матрица или разреженный массив – это матрица, в которой большинство элементов равны нулю. Не существует строгого определения относительно доли элементов с нулевым значением для того, чтобы матрица считалась разреженной, но общим критерием является, что количество ненулевых элементов примерно равно количеству строк или столбцов. Напротив, если большинство элементов не равны нулю, матрица считается плотной. Количество элементов с нулевым значением, деленное на общее количество элементов, иногда называют разреженностью матрицы [1]. На рисунке 1.1 представлен пример разреженной матрицы.

Изображение выглядит как Шрифт, белый, текст, типография

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Пример разреженной матрицы

## Форматы разреженных матриц

Рассмотрим следующие форматы разреженных матриц [2]:

1. **Координатный список** («COO» – «Coordinate Format») – формат хранения разреженных матриц, где явно указываются координаты каждого ненулевого элемента. Этот формат состоит из:

* «row\_indices» – номера строк (начинаются с 0 или 1);
* «col\_indices» – номера столбцов;
* «values» – значения ненулевых элементов.

Матрица из примера в данном формате будет выглядеть следующим образом:

* «row\_indices» = [0, 0, 1, 1, 2, 2, 2, 3, 4];
* «col\_indices = [0, 1, 1, 2, 2, 3, 4, 5, 6];
* «values» = [11, 22, 33, 44, 55, 66, 77, 88, 99].

Преимущества:

* простота создания и модификации;
* подходит для поэлементного заполнения (например, при чтении из файла или записи в него).

Недостатки:

* неэффективен для арифметических операций;
* отсутствие быстрого доступа к строкам / столбцам.

Таким образом, координатный список – это простой, но неоптимальный формат для вычислений.

1. **Сжатое хранение строкой** («CSR» – «Compressed Sparse Row») – оптимальный формат для хранения разреженных матриц, ориентированный на быстрый доступ по строкам и эффективное умножение матрицы на вектор. Структура такого формата:

* «values» – массив ненулевых элементов (по порядку слева направо, сверху вниз);
* «col\_indices» – номера столбцов для каждого элемента;
* «row\_ptr» – указатели на начало каждой строки (размер = количество строк исходной матрицы + 1).

В «CSR» исходная матрица будет выглядеть так:

* «values» = [11, 22, 33, 44, 55, 66, 77, 88, 99];
* «col\_indices» = [0, 1, 1, 2, 2, 3, 4, 5, 6];
* «row\_ptr» = [0, 2, 4, 7, 8, 9].

Преимущества:

* быстрый доступ к строкам – идеален для умножения матрицы на вектор.

Недостатки:

* сложное добавление и удаление элементов;
* медленный доступ к столбцам в противопоставление к быстрому доступу к строкам.

1. **Сжатое хранение столбцом** («CSC» – «Compressed Sparse Column») – аналог «CSR», оптимизированный для доступа по столбцам; состоит из:

* «values» – массив ненулевых элементов (по порядку сверху вниз, слева направо);
* «row\_indices» – номера строк для каждого элемента;
* «col\_ptr» – указатели на начало каждого столбца (размер = количество столбцов исходной матрицы + 1).

В «CSC» исходная матрица будет выглядеть так:

* «values» = [11, 22, 33, 44, 55, 66, 77, 88, 99];
* «row\_indices» = [0, 0, 1, 1, 2, 2, 2, 3, 4];
* «col\_ptr» = [0, 1, 3, 5, 6, 7, 8, 9].

Преимущества и недостатки аналогичны формату «CSR», но только по отношению к столбцам.

1. **Список списков** («LIL» – «List of Lists») – гибкий формат для хранения разреженных матриц, где каждая строка представлена отдельным списком ненулевых элементов и их столбцов. Структура «LIL»:

* «rows» – массив списков, где каждая ячейка содержит номера ненулевых элементов в строке;
* «data» – массив списков, где каждая ячейка содержит значения этих ненулевых элементов.

Матрица из примера в формате «LIL» выглядит так:

* «rows» = [[0, 1], [1, 2], [2, 3, 4], [5], [6]];
* «data» = [[11, 22], [33, 44], [55, 66, 77], [88], [99]].

Преимущества:

* простота модификации – легко добавлять и удалять элементы;
* эффективен для построчного заполнения.

Недостатки:

* медленный доступ для арифметических операций как, например, перемножение матрицы на вектор;
* высокие накладные расходы из-за хранения списков.

1. **Словарь ключей** («DOK» – «Dictionary of Keys») – формат для хранения разреженных матриц, где каждый ненулевой элемент задаётся парой координат (строка, столбец) и хранится в словаре.

В формате «DOK» исходная матрица будет выглядеть так:

* «dok» = {(0, 0): 11, (0, 1): 22, (1, 1): 33, (1, 2): 44, (2, 3): 55, (2, 4): 66, (2, 5): 77, (3, 5): 88, (4, 6): 99}.

Преимущества:

* мгновенный доступ к элементам матрицы;
* простота модификации – легко добавлять и удалять элементы.

Недостатки:

* медленный доступ для арифметических операций как, например, перемножение матрицы на вектор;
* высокие накладные расходы из-за хранения ключей.

# Основная часть

Для начала необходимо выбрать матрицу, с которой будем работать в ходе курсового проекта. Пусть это будет «Serena». Параллельную программу реализуем с помощью «CUDA» на видеокарте «NVidia GeForce GTX 1660». Для библиотечного варианта применим «cuSPARSE», а для снятия профилей – «Nsight Systems» и «Nsight Compute».

## Установка программного обеспечения

Файлы «mmio.h» и «mmio.cpp», а также матрицу «Serena» необходимо скопировать в файл проекта, затем нужно добавить «mmio.cpp» к проекту так, как показано на рисунке 2.1.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Добавление «mmio.cpp»

Установка «CUDA», «Nsight Systems» и «Nsight Compute» осуществляется с помощью установщика, в котором после скачивания достаточно выбрать «экспресс-установку», благодаря чему всё необходимое программное обеспечение установится правильно.

## Разработка программы

Для начала необходимо подключить все библиотеки, которые будут использоваться в проекте, и стандартное пространство имен, а также задать количество нитей в блоке для выполнения программы на графическом процессоре (рисунок 2.2).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Подключение библиотек

Затем с помощью функции «mm\_read\_unsymmetric\_sparse(…)», которая реализована во вспомогательном коде [3], необходимо считать заданную матрицу, как показано на рисунке 2.3.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Считывание матрицы

В исходном файле могут быть строки, значение которых равно нулю, то подсчитаем количество ненулевых [строк], чтобы при переводе разреженной матрицы в другой формат не сохранять первые [т.е. нулевые строки] (рисунок 2.4).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Подсчёт количества ненулевых строк

Так как нам необходимо реализовать перемножение матрицы на вектор, то исходя из теоретической части для этой операции лучше всего использовать формат «CSR», а, следовательно, нужно перевести матрицу из формата «COO». Для этого:

1. Сначала создадим векторы «values», «col\_indices» и «row\_ptr», а также «position», в котором будет храниться количество ненулевых значений в одной строке.
2. Затем с помощью цикла «for» будем добавлять в «row\_ptr» количество ненулевых значений на каждой строке.
3. После ещё одним циклом «for» просуммируем количество ненулевых значений.
4. Наконец очередным циклом «for» запишем:

* в «values» не равные нулю значения;
* в «col\_indices» номера столбцов этих значений.

1. В конце выполним проверку: совпадает ли количество ненулевых значений «count\_0», посчитанное ранее, с числом, находящимся в конце массива «row\_ptr».

Реализация описанного выше алгоритма представлена на рисунке 2.5.

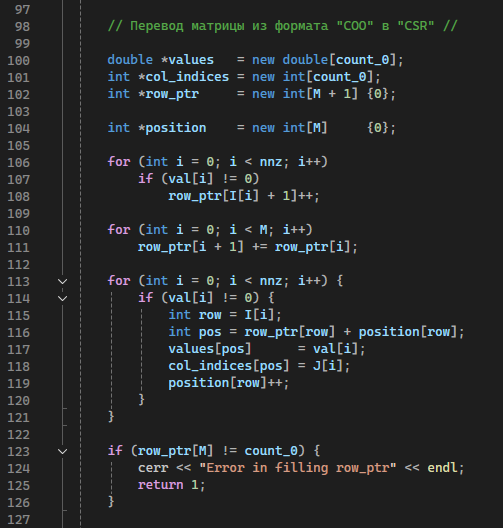


Рисунок . – Перевод матрицы из формата «COO» в «CSR»

Чтобы убедиться в корректности работы алгоритма выведем первые 10 элементов «CSR» матрицы, как показано на рисунке 2.6. На рисунке 2.7 изображен вывод этих элементов в консоли.

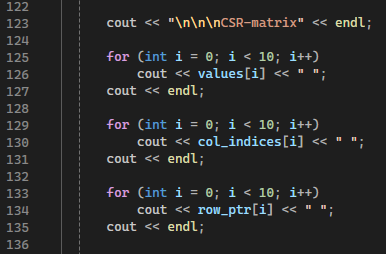


Рисунок . – Вывод элементов «CSR» матрицы

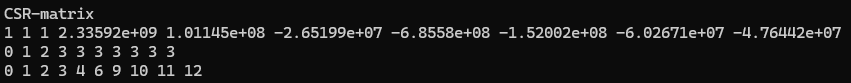


Рисунок . – Пример вывода элементов «CSR» матрицы

Для начала реализуем последовательную версию программы перемножения матрицы на вектор. Создадим результирующий вектор «res\_cpp» размерностью «M[[1]](#footnote-1)» и единичный вектор «vec» размерностью «N[[2]](#footnote-2)», который инициализируем единицами циклом «for» (рисунок 2.8).

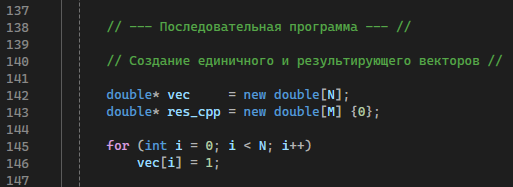


Рисунок . – Создание и инициализация векторов

Теперь перейдем непосредственно к перемножению (рисунок 2.9):

* с помощью функции «clock()» зафиксируем время начала и завершения операции;
* двумя циклами «for» реализуем перемножение: первый цикл будет проходить по количеству строк исходной матрицы, а второй – по массивам преобразованной матрицы;
* после выведем первые 10 элементов произведения и время выполнения в миллисекундах.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Последовательная программа

На рисунке 2.10 представлен пример результата выполнения последовательной программы.

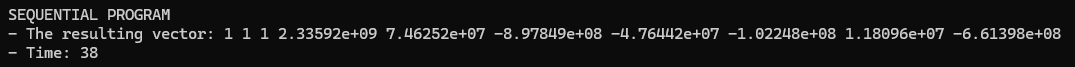


Рисунок . – Пример результата выполнения последовательной программы

Далее создадим код для параллельной программы на «CUDA». Сначала создадим переменные для замера времени (рисунок 2.11):

* «time\_transaction1», «start\_transaction1» и «stop\_transaction1» – время копирования с «CPU» на «GPU»;
* «time\_transaction2», «start\_transaction2» и «stop\_transaction2» – время копирования с «GPU» на «CPU»;
* «time\_calculation», «start\_calculation» и «stop\_calculation» – время выполнения перемножения.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Создание переменных для замера времени

Теперь создадим (рисунок 2.12):

* массивы:
  + - «res» – для сохранения результата параллельного перемножения в программе на «CPU»;
    - «values\_cu», «vec\_cu», «res\_cu», «col\_indices\_cu» и «row\_ptr\_cu» – копии массивов «values», «vec», «res», « col\_indices » и « row\_ptr » на «CPU»;
* переменные:
  + - «size\_double» и «size\_int» – объемы памяти, занимаемый переменными типа «double» и «int» соответственно (необходимы для того, чтобы не получать этот объем каждый раз при выделении памяти).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Создание массивов и переменных

Далее с помощью «cudaMalloc(…)» выделим память на «GPU» для массивов, описанных ранее, а функцией «cudaMemset(…)» инициализируем «res\_cu» нулями, как показано на рисунке 2.13.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Выделение памяти

Наконец скопируем массивы, применив функцию «cudaMemcpy(…)», а с помощью «cudaEventRecord(…)» зафиксируем время начала и завершения копирования (рисунок 2.14).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Копирование с «CPU» на «GPU»

Далее вызовем функцию перемножение матрицы на вектор «SpMV» (рисунок 2.15), которая работает так же, как и последовательная версия, за исключением того, что «threadIdx.x + blockIdx.x \* blockDim.x» распределяет строки матрицы между потоками графического процессора. Реализация функции «SpMV» приведена на рисунке 2.16.

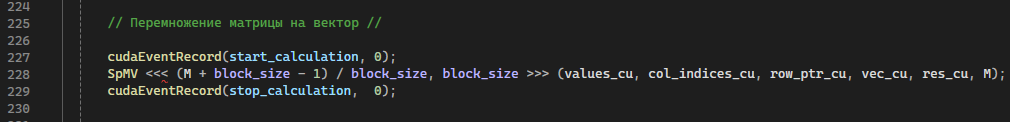


Рисунок . – Вызов функции перемножения

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Функция «SpMV»

После выполнения перемножения скопируем полученную матрицу с графического процессора на центральный, а также замерим время этой операции (рисунок 2.17).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Копирование с «GPU» на «CPU»

Выполним синхронизацию функцией «cudaDeviceSynchronize()» (рисунок 2.18). Затем применим «check\_results» для проверки корректности: сравним векторы, полученные последовательной и параллельной программами, с помощью относительной погрешности, как показано на рисунке 2.19.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Синхронизация и проверка корректности

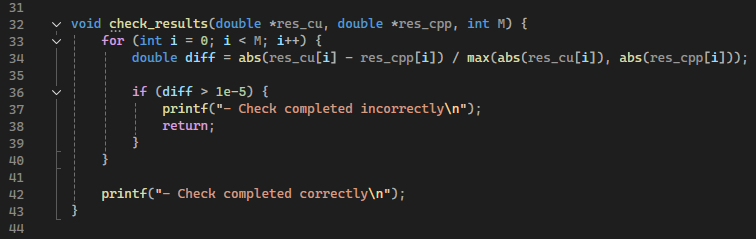
****

Рисунок . – Функция «check\_results»

«cudaEventElapsedTime(…)» позволяет вычислить разницу между событиями во времени в миллисекундах, которую затем выведем в консоль (рисунок 2.20). Далее применим функцию «save\_vector», чтобы сохранить полученный после перемножения вектор в формате «mtx». Реализация этой функции представлена на рисунке 2.21.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Время выполнения

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Функция «save\_vector»

Результат, получаемый в консоли после параллельного перемножения, приведен на рисунке 2.22.

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Пример результата выполнения параллельной программы

Наконец реализуем параллельную программу с использованием библиотеки «cuSPARSE». Сначала также необходимо создать переменные, которые будут нужны для замера времени (рисунок 2.23).

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Создание переменных для замера времени

Затем, как показано на рисунке 2.24, создадим дескрипторы:

* первый («handle») нужен для вызова функций библиотеки,
* остальные («matA», «vecX» и «vecY»)

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Создание дескрипторов

Далее зададим коэффициенты и выберем алгоритм – перемножение матрицы на вектор (рисунок 2.25).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Задание коэффициентов и выбор алгоритма

Создадим буфер, как показано на рисунке 2.26. Он [буфер] необходим для работы алгоритма «cuSPARSE».

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Создание буфера

Теперь функцией «cusparseSpMV» выполним перемножение матрицы на вектор, как приведено на рисунке 2.27.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Перемножение матрицы на вектор

После этого аналогичному тому, как делали для параллельной программы без «cuSPARSE», выполним копирование полученного вектора на «CPU», проверим корректность, выведем время и сохраним полученный вектор (рисунок 2.28).

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Копирование, проверка корректности и сохранение

Результат, получаемый в консоли после параллельного перемножения с использованием «cuSPARSE», приведен на рисунке 2.29.

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Пример результата выполнения библиотечной программы

В конце, когда все программы выполнены, необходимо освободить память (рисунок 2.30).

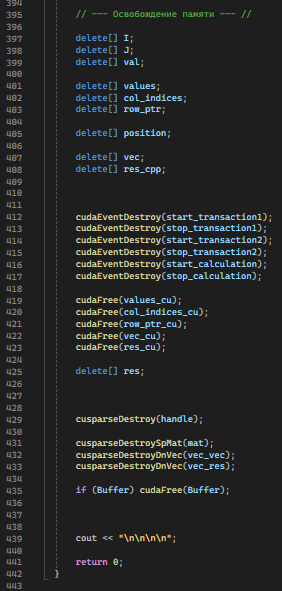


Рисунок . – Освобождение памяти

## Расчёт производительности

Рассчитаем производительность по следующим формулам:

* производительность расчета (гигафлопс) ;
* пропускная способность = .

После расчета получим следующие результаты (рисунок 2.31).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Полученные результаты

## Профили выполнения

Снимем профиль выполнения программы в «Nsight Systems», после чего получим график, представленный на рисунке 2.32.

Изображение выглядит как текст, снимок экрана, Параллельный, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок . – Результат профилировки в «Nsight Systems»

# Вывод

В ходе курсового проекта мы изучили разреженные матрицы и их форматы, а также реализовали последовательную и параллельную (собственную и с использованием библиотеки) программы перемножения матрицы на вектор.

# Список литературы

1. <https://en.wikipedia.org/wiki/Sparse_matrix>
2. <https://python-school.ru/blog/python/sparse-matrix/>
3. <https://math.nist.gov/MatrixMarket/mmio-c.html>

# Приложение

## Код программы

#define block\_size 256

#include <iostream>

#include <fstream>

#include <vector>

#include "mmio.h"

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <cusparse.h>

using namespace std;

\_\_global\_\_ void SpMV(double \*values\_cu, int \*col\_indices\_cu, int \*row\_ptr\_cu, double \*vec\_cu, double \*res\_cu, int M) {

int row = threadIdx.x + blockIdx.x \* blockDim.x;

if (row < M) {

double sum = 0;

int row\_start = row\_ptr\_cu[row];

int row\_stop = row\_ptr\_cu[row + 1];

for (int j = row\_start; j < row\_stop; j++)

sum += values\_cu[j] \* vec\_cu[col\_indices\_cu[j]];

res\_cu[row] = sum;

}

}

void check\_results(double \*res\_cu, double \*res\_cpp, int M) {

for (int i = 0; i < M; i++) {

double diff = abs(res\_cu[i] - res\_cpp[i]) / max(abs(res\_cu[i]), abs(res\_cpp[i]));

if (diff > 1e-5) {

printf("- Check completed incorrectly\n");

return;

}

}

printf("- Check completed correctly\n");

}

void save\_vector(const string &filename, const double \*vec, int size) {

ofstream file(filename);

if (!file.is\_open()) {

cerr << "Error: Cannot open file " << filename << endl;

return;

}

file << "%%MatrixMarket matrix array real general\n";

file << size << " 1\n";

for (int i = 0; i < size; ++i)

file << vec[i] << "\n";

file.close();

cout << "Vector saved to " << filename << endl;

}

int main() {

// Считывание матрицы //

const string fpath = "Serena.mtx";

int M = 0;

int N = 0;

int nnz = 0;

int \*I;

int \*J;

double \*val;

int rcode = mm\_read\_unsymmetric\_sparse(fpath.c\_str(), &M, &N, &nnz, &val, &I, &J);

cout << "M = " << M << " N = " << N << " nnz = " << nnz << endl;

for (int i = 0; i < 10; i++)

cout << "elem " << i << " row = " << I[i] << " column = " << J[i] << " val = " << val[i] << endl;

// Подсчёт количества ненулевых строк //

int count\_0 = 0;

for (int i = 0; i < nnz; i++)

if (val[i] != 0)

count\_0++;

// Перевод матрицы из формата "COO" в "CSR" //

double \*values = new double[count\_0];

int \*col\_indices = new int[count\_0];

int \*row\_ptr = new int[M + 1] {0};

int \*position = new int[M] {0};

for (int i = 0; i < nnz; i++)

if (val[i] != 0)

row\_ptr[I[i] + 1]++;

for (int i = 0; i < M; i++)

row\_ptr[i + 1] += row\_ptr[i];

for (int i = 0; i < nnz; i++) {

if (val[i] != 0) {

int row = I[i];

int pos = row\_ptr[row] + position[row];

values[pos] = val[i];

col\_indices[pos] = J[i];

position[row]++;

}

}

if (row\_ptr[M] != count\_0) {

cerr << "Error in filling row\_ptr" << endl;

return 1;

}

cout << "\n\n\nCSR-matrix" << endl;

for (int i = 0; i < 10; i++)

cout << values[i] << " ";

cout << endl;

for (int i = 0; i < 10; i++)

cout << col\_indices[i] << " ";

cout << endl;

for (int i = 0; i < 10; i++)

cout << row\_ptr[i] << " ";

cout << endl;

// --- Последовательная программа --- //

// Создание единичного и результирующего векторов //

double\* vec = new double[N];

double\* res\_cpp = new double[M] {0};

for (int i = 0; i < N; i++)

vec[i] = 1;

// Перемножение матрицы на вектор //

double start\_time = clock();

for (int i = 0; i < M; i++)

for (int j = row\_ptr[i]; j < row\_ptr[i + 1]; j++)

res\_cpp[i] += values[j] \* vec[col\_indices[j]];

double end\_time = clock();

cout << "\n\n\nSEQUENTIAL PROGRAM" << endl;

cout << "- The resulting vector: ";

for (int i = 0; i < 10; i++)

cout << res\_cpp[i] << " ";

cout << endl;

cout << "- Time: " << (end\_time - start\_time) / CLOCKS\_PER\_SEC \* 1000 << endl;

// --- Параллельная программа --- //

// Создание переменных для замера времени //

float time\_transaction1 = 0;

float time\_transaction2 = 0;

float time\_calculation = 0;

cudaEvent\_t start\_transaction1, stop\_transaction1, start\_transaction2, stop\_transaction2, start\_calculation, stop\_calculation;

cudaEventCreate(&start\_transaction1);

cudaEventCreate(&stop\_transaction1 );

cudaEventCreate(&start\_transaction2);

cudaEventCreate(&stop\_transaction2 );

cudaEventCreate(&start\_calculation );

cudaEventCreate(&stop\_calculation );

// Создание массивов //

double \*res = new double[M];

double \*values\_cu, \*vec\_cu, \*res\_cu;

int \*col\_indices\_cu, \*row\_ptr\_cu;

int size\_double = sizeof(double);

int size\_int = sizeof(int);

// Выделение памяти //

cudaMalloc((void\*\*)&values\_cu, count\_0 \* size\_double);

cudaMalloc((void\*\*)&col\_indices\_cu, count\_0 \* size\_int );

cudaMalloc((void\*\*)&row\_ptr\_cu, (M + 1) \* size\_int );

cudaMalloc((void\*\*)&vec\_cu, N \* size\_double);

cudaMalloc((void\*\*)&res\_cu, M \* size\_double);

// Инициализация результирующего массива нулями //

cudaMemset(res\_cu, 0, M \* size\_double);

// Копирование с "CPU" на "GPU" //

cudaEventRecord(start\_transaction1, 0);

cudaMemcpy(values\_cu, values, count\_0 \* size\_double, cudaMemcpyHostToDevice);

cudaMemcpy(col\_indices\_cu, col\_indices, count\_0 \* size\_int, cudaMemcpyHostToDevice);

cudaMemcpy(row\_ptr\_cu, row\_ptr, (M + 1) \* size\_int, cudaMemcpyHostToDevice);

cudaMemcpy(vec\_cu, vec, N \* size\_double, cudaMemcpyHostToDevice);

cudaEventRecord(stop\_transaction1, 0);

// Перемножение матрицы на вектор //

cudaEventRecord(start\_calculation, 0);

SpMV <<< (M + block\_size - 1) / block\_size, block\_size >>> (values\_cu, col\_indices\_cu, row\_ptr\_cu, vec\_cu, res\_cu, M);

cudaEventRecord(stop\_calculation, 0);

// Копирование с "GPU" на "CPU" //

cudaEventRecord(start\_transaction2, 0);

cudaMemcpy(res, res\_cu, M \* size\_double, cudaMemcpyDeviceToHost);

cudaEventRecord(stop\_transaction2, 0);

// Синхронизация и проверка корректности //

cudaDeviceSynchronize();

cout << "\n\n\nPARALLEL PROGRAM" << endl;

check\_results(res, res\_cpp, M);

// Время выполнения //

cudaEventElapsedTime(&time\_transaction1, start\_transaction1, stop\_transaction1);

cudaEventElapsedTime(&time\_calculation, start\_calculation, stop\_calculation );

cudaEventElapsedTime(&time\_transaction2, start\_transaction2, stop\_transaction2);

printf("- Time copying from CPU to GPU: %.2f \n", time\_transaction1);

printf("- Time calculation: %.2f \n", time\_calculation );

printf("- Time copying from GPU to CPU: %.2f \n", time\_transaction2);

// Сохранение вектора //

save\_vector("Sharigin\_MS\_\_\_Serena\_\_\_Without\_cuSPARSE.mtx", res, M);

// --- Параллельная программа с использованием "cuSPARSE" --- //

// Создание переменных для замера времени //

float time\_transaction\_SP = 0;

float time\_calculation\_SP = 0;

cudaEvent\_t start\_transaction\_SP, stop\_transaction\_SP, start\_calculation\_SP, stop\_calculation\_SP;

cudaEventCreate(&start\_transaction\_SP);

cudaEventCreate(&stop\_transaction\_SP );

cudaEventCreate(&start\_calculation\_SP);

cudaEventCreate(&stop\_calculation\_SP );

// Создание дескриптора //

cusparseHandle\_t handle;

cusparseCreate(&handle);

// Дескрипторы матрицы и векторов (единичного и результирующего) //

cusparseSpMatDescr\_t mat;

cusparseDnVecDescr\_t vec\_vec, vec\_res;

cusparseCreateCsr(

&mat,

M, N, count\_0,

row\_ptr\_cu,

col\_indices\_cu,

values\_cu,

CUSPARSE\_INDEX\_32I,

CUSPARSE\_INDEX\_32I,

CUSPARSE\_INDEX\_BASE\_ZERO,

CUDA\_R\_64F

);

cusparseCreateDnVec(&vec\_vec, N, vec\_cu, CUDA\_R\_64F);

cusparseCreateDnVec(&vec\_res, M, res\_cu, CUDA\_R\_64F);

// Коэффициенты перед матрицей и вектором //

double alpha = 1.0;

double beta = 0.0;

// Выбор алгоритма //

cusparseSpMVAlg\_t alg = CUSPARSE\_SPMV\_CSR\_ALG1;

// Буфер //

size\_t bufferSize = 0;

void \*Buffer = nullptr;

cusparseSpMV\_bufferSize(

handle, CUSPARSE\_OPERATION\_NON\_TRANSPOSE,

&alpha, mat, vec\_vec, &beta, vec\_res, CUDA\_R\_64F, alg, &bufferSize

);

cudaMalloc(&Buffer, bufferSize);

// Перемножение матрицы на вектор //

cudaEventRecord(start\_calculation\_SP, 0);

cusparseSpMV(

handle, CUSPARSE\_OPERATION\_NON\_TRANSPOSE,

&alpha, mat, vec\_vec, &beta, vec\_res, CUDA\_R\_64F, alg, Buffer

);

cudaEventRecord(stop\_calculation\_SP, 0);

// Копирование с "GPU" на "CPU" //

cudaEventRecord(start\_transaction\_SP, 0);

cudaMemcpy(res, res\_cu, M \* sizeof(double), cudaMemcpyDeviceToHost);

cudaEventRecord(stop\_transaction\_SP, 0);

// Проверка корректности //

cudaDeviceSynchronize();

cout << "\n\n\nPARALLEL PROGRAM (cuSPARSE)" << endl;

check\_results(res, res\_cpp, M);

// Время выполнения //

cudaEventElapsedTime(&time\_calculation\_SP, start\_calculation\_SP, stop\_calculation\_SP);

cudaEventElapsedTime(&time\_transaction\_SP, start\_transaction\_SP, stop\_transaction\_SP);

printf("- Time copying from CPU to GPU: %.2f \n", time\_transaction1 );

printf("- Time calculation: %.2f \n", time\_calculation\_SP);

printf("- Time copying from GPU to CPU: %.2f \n", time\_transaction\_SP);

// Сохранение вектора //

save\_vector("Sharigin\_MS\_\_\_Serena\_\_\_With\_cuSPARSE.mtx", res, M);

// --- Освобождение памяти --- //

delete[] I;

delete[] J;

delete[] val;

delete[] values;

delete[] col\_indices;

delete[] row\_ptr;

delete[] position;

delete[] vec;

delete[] res\_cpp;

cudaEventDestroy(start\_transaction1);

cudaEventDestroy(stop\_transaction1);

cudaEventDestroy(start\_transaction2);

cudaEventDestroy(stop\_transaction2);

cudaEventDestroy(start\_calculation);

cudaEventDestroy(stop\_calculation);

cudaFree(values\_cu);

cudaFree(col\_indices\_cu);

cudaFree(row\_ptr\_cu);

cudaFree(vec\_cu);

cudaFree(res\_cu);

delete[] res;

cusparseDestroy(handle);

cusparseDestroySpMat(mat);

cusparseDestroyDnVec(vec\_vec);

cusparseDestroyDnVec(vec\_res);

if (Buffer) cudaFree(Buffer);

cout << "\n\n\n\n";

return 0;

}

1. «M» – количество строк в исходной матрице [↑](#footnote-ref-1)
2. «N» – количество столбцов в исходной матрице [↑](#footnote-ref-2)