ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Итерационный метод решения обобщенной задачи поиска собственных значений и векторов на основе LU-разложения правой матрицы с использованием технологии CUDA

Выпускная квалификационная работа
обучающегося по направлению подготовки
02.03.02 Фундаментальная информатика и информационные технологии
очной формы обучения, группы 07001401
Макошенко Данила Михайловича

Научный руководитель
к.т.н., ст.пр. Лихошерстный А.Ю.

БЕЛГОРОД 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача поиска собственных значений и векторов часто предстает перед инженерами, которые проводят расчеты в таких промышленных сферах как строительство, машиностроение, авиастроение и т.п. Например, частоты колебаний выражаются через собственные значения. При обсчетах конструкций, большие части которых составляют пустоты, получаются разреженные матрицы. В связи с этим появляется проблема хранения и использования большого объёма данных в расчетах. Универсальных алгоритмов решения поставленной задачи не существует.

Наиболее высокой скоростью обсчета являются вычисления на графическом процессоре, т.к. подобные решения обладают значительно большим количеством вычислительных ядер по сравнению с центральным процессором. Однако объем памяти графического ускорителя, как правило, составляет всего несколько гигабайт.

Существующие методы решения собственных значений и векторов в процессе вычисления получают промежуточные плотные матрицы, что значительно увеличивает расход памяти. Предлагаемый метод сохраняет промежуточные результаты вычислений в разреженном виде, при этом не замедляя процесс вычислений.

Целью выпускной квалификационной работы является разработка и программная реализация итерационного алгоритма решения обобщенной задачи поиска собственных значений и векторов для разреженных матриц на основе LU-разложения правой части уравнения с использованием технологии CUDA

Поставленная цель достигается решением следующих задач:

- Обзор существующих методов решения задачи поиска собственных значений и векторов;

- Разработка алгоритма решения обобщенной задачи поиска собственных значений и векторов для разреженных матриц на основе LU-разложения правой части уравнения;
- Программная реализация разработанного алгоритма с использованием технологии CUDA;
- Оценка эффективности разработанного алгоритма на основе проведения вычислительных экспериментов.

В первой главе осуществлен обзор существующих методов решения задачи поиска собственных значений и векторов, дана базовая информация о собственных значениях и векторах, LU – разложении, формате CSR, а также технологии CUDA, которая использовалась при разработке параллельной версии алгоритма.

Во второй главе были описаны все аспекты разработанного алгоритма, а именно преобразование обобщенной задачи поиска собственных значений и векторов к классической задаче, где использовало неполное LU – разложение, которой посвящена отдельная подглава. Так же был описан алгоритм решения классической задачи при помощи степенного метода, который позволяет решить её минимально расходуя память.

В третьей главе описаны основные этапы разработки параллельной версии алгоритма с использованием технологии CUDA, а также приведены результаты испытаний программной реализации на матрицах различной размерности.

Данная выпускная квалификационная работа состоит из 46 страниц, 3 рисунков, 4 таблиц и 35 использованных литературных источников.

1. ОБЗОР СОВРЕМЕННЫХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ ПОИСКА СОБСТВЕННЫХ ЗНАЧЕНИЙ И ВЕКТОРОВ

1.1 Собственные значения и вектора матриц

Собственный вектор – понятие в линейной алгебре, определяемое для квадратной матрицы как вектор, умножение матрицы на который даёт тот же вектор, умноженный на некоторое скалярное значение, называемое собственным значением матрицы.

Таким образом, собственным вектором матрицы A называется такой ненулевой вектор *x*, что для некоторого λ выполняется равенство (1.1).

$$Ax = \lambda x \tag{1.1}$$

В свою очередь, собственным значением матрицы A называется такое число λ, для которого существует собственный вектор, т.е. равенство (1) имеет ненулевое решение *x* [1].

Обобщенной задачей поиска собственных значений и векторов называется задача вида (1.2).

$$Ax = \lambda Bx, \tag{1.2}$$

где A и B - матрицы, λ – собственное значение, а x – собственный вектор.

Для решения этой задачи, её необходимо свести к задаче поиска собственных значений и векторов матрицы общего вида (1.1).

Задача нахождения собственных значений и собственных векторов матрицы возникает при решении многих прикладных задач физики, механики, астрономии, в которых требуется определить нетривиальное решение однородной системы линейных алгебраических уравнений вида (1) и тех значение числового параметра λ, при которых такое решение существует [2]. Оценка критических нагрузок при расчете строительных конструкция

основана на собственных значения и векторах матриц. Собственные числа и собственные векторы являются важнейшими характеристиками, отражающими существенные стороны линейных моделей.

1.2 LU разложение матриц

LU-разложение – представление квадратной матрицы A в виде произведения матриц L и U, где L – нижняя треугольная матрица с единичной диагональю, а U – верхняя треугольная [5].

Такое представление удобно для решения системы алгебраических уравнений, т.к. оно позволяет перейти от решения исходной системы к последовательному решению систем с треугольными матрицами.

Будем использовать следующие обозначения для элементов матриц

$$L = (l_{ij}),\ U = (u_{ij}),\ i,j = 1,\dots,n, \tag{1.3}$$

где n – размерность матрицы.

Алгоритм нахождения матриц L и U выполняется последовательно, т.к. на каждом следующем шаге для вычислений используются элементы, найденные ранее. Всего алгоритм состоит из m шагов.

Для m = 1:

$$u_{1j} = a_{1j}, j = 1,\dots,n \tag{1.4}$$

$$l_{j1} = \frac{a_{j1}}{u_{11}}, j = 2,\dots,n (u_{11} \neq 0) \tag{1.5}$$

Для m = 2, …, n:

$$u_{mj} = a_{1j} - \sum_{k=1}^{m} l_{mk} u_{kj}, j = 1,\dots,n \tag{1.6}$$

$$l_{jm} = \frac{1}{u_{mm}} \left(a_{jm} - \sum_{k=1}^{m} l_{jk} u_{km}\right), j = m+1,\dots,n \tag{1.7}$$

В программной реализации в памяти компьютера LU разложение не расходует лишнюю память, т.к. занимает ровно столько места, сколько занимает исходная матрица. Размерность L и U матриц такая же, как у исходной, а т.к. U – это верхняя диагональная матрица, а L –нижняя диагональная с единицами на главной диагонали, которые можно не хранить в памяти, то обе матрицы можно хранить как одну, где ниже главной диагонали будут элементы L матрицы, а выше и на диагонали U матрицы.

Кроме того, на каждом шаге результаты вычислений можно сохранять на местах, освобождаемых последовательным затиранием элементов исходной матрицы.

1.3 Существующие алгоритмы нахождения собственных значений и векторов

1.3.1 Решения обобщенной задачи поиска собственных значений на основе разложения Холецкого

Данный алгоритм применяется для решения обобщенной симметричной положительно определенной задачей собственных значений вида (1.8).

$$Ax = \lambda Bx, \tag{1.8}$$

где матрица A - симметричная, а матрица B - симметричная положительно определенная [3].

Очевидно, что эта задача легко сводится к задаче поиска собственных значений несимметричной матрицы общего вида, достаточно умножить обе части системы на B^{-1}. Однако, в результате данного преобразования получится несимметричная задача поиска собственных значений, которая на порядок сложнее симметричной.

Поэтому для сведения к классической симметричной задаче воспользуется разложение Холецкого матрицы B (1.9).

$$B = LL^T, \tag{1.9}$$

где L – нижняя треугольная матрицы со строго положительными элементами на диагонали. Тогда получим равенство 1.10.

$$Ax = \lambda LL^T x \tag{1.10}$$

Т.к. при умножении матрицы L^T на обратную ей матрицу L^{-T} получится единичная матрицы, то предыдущая система эквивалента следующей:

$$AL^{-T}L^T x = \lambda LL^T x \tag{1.11}$$

Умножим обе части уравнения на L^{-1}:

$$L^{-1}AL^{-T}L^T x = \lambda L^T x \tag{1.12}$$

Для приведения задачи выполним следующие замены:

$$\mathrm{C} = L^{-1}AL^{-T} \tag{1.13}$$

$$y = L^T x \tag{1.14}$$

Тогда исходное уравнение сведется к следующему:

$$\mathrm{C}y = \lambda y, \tag{1.15}$$

Где C – симметричная матрица и т.о. получена хорошо известная и эффективно решаемая задача, где для собственные векторы исходной задачи могуть быть получены путем решения системы линейных уравнений с треугольной матрицей L^T.

1.3.2 Метод итераций для нахождения собственных значений и векторов

Для решения частичной проблемы собственных значений и собственных векторов в практических расчетах часто используется метод итераций (степенной метод) [6]. На его основе можно определить приближенно собственные значения матрицы A.

Алгоритм метода итераций состоит из следующих шагов:

1. Выбрать произвольное начальное приближение собственного вектора $X^{1(0)}$ (индекс в скобках здесь и далее указывает номер приближения, а индекс без скобок соответствует номеру собственного значения или вектора);
2. Найти первые приближения собственных векторов (1.16) и значения (1.17) и положить k = 0.

$$X^{1(1)} = AX^{1(0)} \tag{1.16}$$

$$\lambda_1^{(1)} = \frac{x_i^{1(1)}}{x_i^{1(0)}}, \tag{1.17}$$

где i – любой номер $1 \le i \le n$;

3. Вычислить (1.18)

$$X^{1(k+1)} = AX^{1(k)} \tag{1.18}$$

4. Найти следующее приближение собственного значения:

$$\lambda_1^{(k+1)} = \frac{x_i^{1(k+1)}}{x_i^{1(k)}}, \tag{1.19}$$

где $x_i^{1(k+1)}$, $x_i^{1(k)}$ – соответствующие координаты векторов $X^{1(k+1)}$ и $X^{1(k)}$. При этом может быть использована любая координата с номером i, $1 \le i \le n$;

5. Если $\Delta = \left|\lambda_1^{(k+1)} - \lambda_1^{(k)}\right| \le \varepsilon$, процесс завершить и положить

$$\lambda_1 \cong \lambda_1^{(k+1)}. \quad (1.20)$$

Если $\Delta > \varepsilon$, положить k=k+1 и перейти к шагу 3.

Вместо применяемой в пункте 4 алгоритма формы для $\lambda_1^{(k+1)}$ можно взять среднее арифметическое соответствующих отношений для разных координат.

Используя λ_1, можно определить следующее значение λ_2 по формуле 1.21.

$$\lambda_2 = \frac{x_i^{1(k+1)} - \lambda x_i^{1(k)}}{x_i^{1(k)} - \lambda x_i^{1(k-1)}} \quad (1.21)$$

где i = 1, 2, …, n. Эта формула даёт грубые значения для λ_2, так как значение λ_1 является приближенным. На основе (1.21) можно вычислять остальные λ_j (j = 3, 4, …, n).

1.3.3 Метод А.М. Данилевского

Метод А.М. Данилевского заключается в преобразовании исходной матрицы

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (1.22)$$

в подобную ей матрицу Фронбениуса

$$P = \begin{pmatrix} p_1 & p_2 & \dots & p_{n-1} & p_n \\ 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & 1 \end{pmatrix} \quad (1.23)$$

по формуле

$$P = B^{-1}AB \qquad (1.24)$$

с помощью матрицы подобия B. Переход от матрицы A к подобной ей матрице P осуществляется с помощью n-1 преобразований подобия, последовательно преобразующих строки матрицы A, начиная с последней, в соответствующие строки матрицы P.

На первом этапе, предполагая что $a_{n,n-1} \neq 0$, построим матрицу B_1 (1.26), заменив в единичной матрицу порядка n элементы n-1 строки на значения

$$b_{n-1,j} = -\frac{a_{nj}}{a_{n,n-1}} \; j \neq n-1;$$
$$b_{n-1,n-1} = \frac{1}{a_{n,n-1}} \qquad (1.25)$$

$$B_1 = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ b_{n-1,1} & b_{n-1,2} & \dots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \qquad (1.26)$$

Умножим справа матрицу A на матрицу B1

$$A \cdot B_1 = C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1,n-1} & c_{1,n} \\ c_{21} & c_{22} & \dots & c_{2,n-1} & c_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ c_{n-1,1} & c_{n-1,2} & \dots & c_{n-1,n-1} & c_{n-1,n} \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}, \qquad (1.27)$$

где

$$\mathrm{c}_{ij} = a_{ij} + a_{i,n-1} \cdot b_{n-1,j} \;\; 1 \leq i \leq n, \;\; j \neq n-1;$$
$$\mathrm{c}_{i,n-1} = a_{i,n-1} \cdot b_{n-1,n-1} \;\; 1 \leq i \leq n. \qquad (1.28)$$

Тогда обратная матрица B_1^{-1} имеет вид

$$B_1^{-1} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & a_{nn} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \quad (1.29)$$

Пусть $D_1 = B_1^{-1} \cdot C$. Т.к., очевидно, умножение слева матрицы C на матрицу B_1^{-1} не изменяет последнюю строку C, то матрицы D_1 имеет вид

$$D_1 = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1,n-1} & d_{1,n} \\ d_{11} & d_{22} & \dots & d_{2,n-1} & d_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ d_{n-1,1} & d_{n-1,2} & \dots & d_{n-1,n-1} & d_{n-1,n} \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}, \quad (1.30)$$

где

$$d_{ij} = c_{ij} \quad 1 \le i \le n-2$$

$$d_{n-1,j} = \sum_{k=1}^{n} a_{nk} c_{kj} \quad 1 \le j \le n. \quad (1.31)$$

Полученная матрица D_1 подобна матрице A и имеет одну преобразованную строку. Этим заканчивается первый этап процесса.

На втором этапе, предполагая, что $d_{n-1,n-2} \neq 0$, построим матрицу B_2, заменив в единичной матрице порядка n элементы n-2 строки на значения

$$b_{n-2,j} = -\frac{d_{n-1,j}}{d_{n-1,n-2}} \quad j \neq n-2\,;$$

$$b_{n-2,n-2} = \frac{1}{d_{n-1,n-2}} \tag{1.32}$$

$$B_2 = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ d_{n-2,1} & d_{n-2,2} & \dots & d_{n-2,n-1} & d_{n-2,n} \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \tag{1.33}$$

Далее, взяв в качестве матрицы A матрицу D_1 и проведя вычисления по формулам (1.28) и (1.31), получим матрицу $D_2 = B_2^{-1} \cdot D_1 \cdot B_2$ с двумя преобразованными строками. Над матрицей D_2 производим те же операции. Продолжая этот процесс, мы получим матрицу Фронбениуса

$$P = B_{n-1}^{-1} \cdot B_{n-2}^{-1} \cdot \dots \cdot B_2^{-1} \cdot B_1^{-1} \cdot A \cdot B_1 \cdot B_2 \cdot \dots \cdot B_{n-2} \cdot B_{n-1}\,, \qquad (1.34)$$

Пусть $\vec{y}$ – собственный вектор матрицы P, отвечающий собственному значению λ. Тогда $P\vec{y} = \lambda\vec{y}$ или в координатном виде

$$\begin{cases} p_1 y_1 + p_2 y_2 + \dots + p_{n-1} y_{n-1} + p_n y_n = \lambda y_1 \\ y_1 = \lambda y_2 \\ y_2 = \lambda y_3 \\ \dots \\ y_{n-1} = \lambda y_n \end{cases} \tag{1.35}$$

Полагая $y_n = 1$, мы получим, используя последовательно эти уравнения с низу вверх, $y_{n-1} = \lambda$, $y_{n-2} = \lambda^2$, …, $y_1 = \lambda^{n-1}$.

$$\vec{y} = (\lambda^{n-1}, \lambda^{n-2}, \dots, \lambda, 1) \tag{1.36}$$

Так как матрицы A и P подобны, то $B^{-1}AB\vec{y} = \lambda\vec{y}$ или $AB\vec{y} = \lambda B\vec{y}$. Это означает, что вектор

$$\vec{x} = B\vec{y} = B \cdot \begin{pmatrix} \lambda^{n-1} \\ \lambda^{n-2} \\ \dots \\ \lambda \\ 1 \end{pmatrix} \tag{1.37}$$

является собственным вектором матрица A, отвечающим собственному значению λ.

1.3.4 Метод А.Н. Крылова

Согласно тождеству Гамильтона-Кели, матрица A удовлетворяет своему характеристическому уравнению, поэтому

$$A^n - p_1A^{n-1} - p_2A^{n-2} - \dots - p_{n-1}A - p_nE = 0 \tag{1.38}$$

Возьмем произвольный ненулевой вектор $\overrightarrow{y_0} = \begin{pmatrix} y_{10} \\ \vdots \\ y_{n0} \end{pmatrix}$ и умножим обе части равенства (1.38) справа на $\overrightarrow{y_0}$

$$A^n\overrightarrow{y_0} - p_1A^{n-1}\overrightarrow{y_0} - p_2A^{n-2}\overrightarrow{y_0} - \dots - p_{n-1}A\overrightarrow{y_0} - p_nE\overrightarrow{y_0} = 0 \tag{1.39}$$

Положим

$$\overrightarrow{y_k} = A^k \overrightarrow{y_0} \quad k = 1,2,\dots,n, \tag{1.40}$$

тогда равенство (1.39) приобретает вид

$$\overrightarrow{y_n} - p_1\vec{y}_{n-1} - p_2\vec{y}_{n-2} - \cdots - p_{n-1}\vec{y}_1 - p_n\vec{y}_0 = \vec{0}, \tag{1.41}$$

где $\overrightarrow{y_k} = \begin{pmatrix} y_{1k} \\ \vdots \\ y_{nk} \end{pmatrix}$ k = 0, 1, …, n.

Векторы $\overrightarrow{y_k}$ удобно находить с помощью рекуррентной формулы

$$\overrightarrow{y_k} = A\vec{y}_{k-1} \quad \text{k = 0, 1, …, n.} \tag{1.42}$$

Следовательно, векторное равенство (1.41) эквивалентно системе линейных алгебраических уравнений

$$p_1\vec{y}_{i,n-1} - p_2\vec{y}_{i,n-2} - \cdots - p_{n-1}\vec{y}_{i1} - p_n\vec{y}_{i0} = \vec{0} \quad \text{i= 1, 2,…,n} \tag{1.43}$$

из которой можно найти неизвестные значений p1, p2, …, pn, определяющие коэффициенты характеристического уравнения. Если векторы $\overrightarrow{y_k}$ k = 0, 1, …, n-1 линейно независимы, то система (1.43) будет иметь единственное решение, если эти векторы окажутся линейно зависимы, то можно изменить начальный вектор $\overrightarrow{y_0}$.

Пусть $\lambda_1, \lambda_2, \dots, \lambda_n$ – корни характеристического уравнения. Разложим вектор $\overrightarrow{y_0}$, использовавшийся при вычислении собственных значений, по собственным векторам $\overrightarrow{x_1}, \overrightarrow{x_2}, \dots, \overrightarrow{x_n}$ матрицы A

$$\overrightarrow{y_0} = \mathrm{c}_1\overrightarrow{x_1} + \mathrm{c}_2\overrightarrow{x_2} + \cdots + \mathrm{c}_n\overrightarrow{x_n} \tag{1.44}$$

где $c_1 \neq 0$ i = 1, 2, …, n – некоторые коэффициенты. Отсюда, учитывая, что

$$\begin{gathered} A\vec{x}_i = \lambda_i \vec{x}_i \\ A^2\vec{x}_i = \lambda_i^2 \vec{x}_i \\ \dots \\ A^{n-1}\vec{x}_i = \lambda_i^{n-1}\vec{x}_i \quad \text{i = 1, 2, …, n} \end{gathered} \tag{1.45}$$

получим

$$\begin{gathered} \vec{y}_1 = c_1\lambda_1\vec{x}_1 + c_2\lambda_2\vec{x}_2 + \cdots + c_n\lambda_n\vec{x}_n \\ \vec{y}_2 = c_1\lambda_1^2\vec{x}_1 + c_2\lambda_2^2\vec{x}_2 + \cdots + c_n\lambda_n^2\vec{x}_n \\ \vec{y}_{n-1} = c_1\lambda_1^{n-1}\vec{x}_1 + c_2\lambda_2^{n-1}\vec{x}_2 + \cdots + c_n\lambda_n^{n-1}\vec{x}_n \end{gathered} \tag{1.46}$$

Пусть

$$Q_i(\lambda) = \lambda^{n-1} + q_{1i}\lambda^{n-2} + \cdots + q_{n-1,l} \quad \text{i = 1, 2, …, n} \tag{1.47}$$

произвольная система многочленов. Составляя линейную комбинацию векторов $\vec{y}_{n-1}$, $\vec{y}_{n-2}$, …,$\vec{y}_0$ с коэффициентами из (1.47) в силу соотношений (1.44) и (1.46) находим

$$\begin{gathered} \vec{y}_{n-1} + q_{1i}\vec{y}_{n-2} + \cdots + q_{n-1,i}\vec{y}_0 = \\ = c_1Q_i(\lambda_1)\vec{x}_1 + c_2Q_i(\lambda_2)\vec{x}_2 + \cdots + c_nQ_1(\lambda_n)\vec{x}_n \\ \text{i = 1, 2, …, n} \end{gathered} \tag{1.48}$$

Если положить

$$Q_i(\lambda) = \frac{D(\lambda)}{\lambda - \lambda_i} \quad \text{i = 1, 2, …, n} \tag{1.49}$$

то, очевидно

$$Q_i(\lambda_j) = \begin{cases} 0, & j \neq i \\ D`(\lambda_i) \neq 0, & j = i \end{cases} \quad (1.50)$$

Формула (1.48) при этом принимает вид

$$c_i Q_i(\lambda_i)\vec{x}_i = \vec{y}_{n-1} + q_{1i}\vec{y}_{n-2} + \cdots + q_{n-1,i}\vec{y}_0 \text{ i = 1, 2, …, n} \quad (1.51)$$

Коэффициенты q_{ji} j = 1, 2, …, n-1 могут быть легко определены по схеме Горнера

$$q_{0i} = 1 \quad , \quad q_{ji} = \lambda_i q_{j-1,i} - p_j \quad j = 1, 2, \dots, n-1 \quad (1.52)$$

Таким образом формула (1.51), в которой коэффициенты вычисляются по формуле (1.52), определяет собственный вектор $\vec{x}_i$, отвечающий собственному значению λ_i, с точностью до числового множителя.

1.3.5 Метод Леверрье-Фаддеева

Этот метод получения характеристического уравнения матрицы основан на формулах Ньютона для сумм степеней корней алгебраического уравнения.

Пусть

$$D(\lambda) = \lambda^n - p_1\lambda^{n-1} - p_2\lambda^{n-2} - \cdots - p_n \quad (1.53)$$

характеристический многочлен матрицы A и $\lambda_1, \lambda_2, \dots, \lambda_n$ – полная совокупность его корней, где каждый корень повторяется столько раз, какова его кратность.

Обозначим $S_k = \lambda_1^k + \lambda_2^k + \cdots + \lambda_n^k$, k = 1, 2, …, n. Тогда при k ≤ n справедливы формулы Ньютона:

$$kp_k = S_k - p_1 S_{k-1} - \cdots - p_{k-1} S_1, \quad k = 1,2, \ldots, n \tag{1.54}$$

Если числа Sk известны, то, решая рекуррентную систему (1.54), можно найти нужные коэффициенты pk:

$$\begin{cases} p_1 = S_1 \\ p_2 = -\frac{1}{2}(S_2 - p_1 S_1) \\ \ldots \\ p_n = -\frac{1}{n}(S_n - p_1 S_{n-1} - \cdots - p_{n-1} S_1) \end{cases} \tag{1.55}$$

Из формул (1.45) следует, что числа λ_i^k, I = 1, 2, …, n являются собственными значениями матрицы Ak, а из формул вытекает, что

$$S_k = SpA^k, \; k = 1, 2, \ldots, n, \tag{1.56}$$

где SpA^k – сумма элементов главной диагонали (след) матрицы Ak.

Степени $A^k = A^{k-1} \cdot A$ находятся непосредственным перемножением.

Таким образом, схема развертывания определителя по методу Леверрье состоит в следующем: сначала вычисляются Ak, k = 1, 2, …, n – степени данной матрицы A, затем находятся соответствующие Sk – суммы элементов главных диагоналей матриц Ak и, наконец, по формулам (1.55) определяются искомые коэффициенты pk, k = 1, 2, …, n.

Д.К.Фаддеев предложил видоизменение метода Леверрье, которое кроме упрощений при вычислении коэффициентов характеристического многочлена позволяет определить обратную матрицу и собственные векторы матрицы.

Будем вместо последовательности A, A2, …, An вычислять последовательность A1, A2, …, An, построенную следующим образом:

$$\begin{array}{lll} A_1 = A, & SpA_1 = q_1 & B_1 = A_1 - q_1 \cdot E \\ A_2 = AB_1 & \dfrac{SpA_2}{2} = q_2 & B_2 = A_2 - q_2 \cdot E \\ \dots & \dots & \dots \\ A_{n-1} = AB_{n-2} & \dfrac{SpA_{n-1}}{n-1} = q_{n-1} & B_{n-1} = A_{n-1} - q_{n-1} \cdot E \\ A_n = AB_{n-1} & \dfrac{SpA_n}{n} = q_n & B_n = A_n - q_n \cdot E \end{array} \qquad (1.57)$$

где E – единичная матрицы того же порядка, что и матрица A.

Ниже приведены доказательства следующих утверждений:

а) q1=p1, q2=p2, …, qn=pn;

б) Bn – нулевая матрица;

в) если A – неособенная матрица, то $A^{-1} = \frac{B_{n-1}}{p_n}$.

а) Используем метод математической индукции. Пусть n = 1, тогда $p_1 = SpA = q_1$. Предположим, что при n = k q1=p1, q2=p2, …, qn=pn, и возьмем n = k+1. Согласно (1.57)

$$A_{k+1} = A^{k+1} - q_1 A^k - \cdots - q_k A = A^{k+1} - p_1 A^k - \cdots - p_k A. \qquad (1.58)$$

Следовательно

$$SpA_{k+1} = (k+1)q_{k+1} = SpA^{k+1} - p_1 SpA^k - \cdots - p_k SpA = \qquad (1.59)$$

$$= S_{k+1} - p_1 S_k - \cdots - p_k S_1.$$

Отсюда в силу формул Ньютона $(k+1)q_{k+1} = (k+1)p_{k+1}$ и, следовательно, $q_{k+1} = p_{k+1}$, что доказывает а).

б) В силу тождества Гамильтона-Кели

$$B_n = A^n - p_1 A^{n-1} - \cdots - p_1 E = 0 \qquad (1.60)$$

в) Из формул (1.59) и (1.60) следует, что

$$AB_{n-1} = A_n = B_n + p_n E = p_n E \qquad (1.61)$$

поэтому $A^{-1} = \frac{B_{n-1}}{p_n}$.

Таким образом коэффициенты характеристического многочлена матрицы А определяются с помощью формул (1.57).

Посмотрим матрицу

$$Q_i = \lambda_i^{n-1} E + \lambda_i^{n-2} B_1 + \cdots + \lambda_i B_{n-2} + B_{n-1}, \qquad (1.62)$$

где B_k – матрицы, вычисленные по формулам (1.57), а λ_i – i-е собственное значение матрицы А.

Можно доказать, в предположении, что все $\lambda_1, \lambda_2, \ldots, \lambda_n$ различны, что матрица Qi ненулевая.

Покажем, что каждый столбец матрицы Qi, состоит из компонент собственного вектора, отвечающего собственному значению λ_i.

Действительно,

$$\begin{gathered}(\lambda_i E - A)Q_i = (\lambda_i E - A)\left(\lambda_i^{n-1} E + \lambda_i^{n-2} B_1 + \lambda_i^{n-3} B_2 + \cdots + \lambda_i B_{n-2} + B_{n-1}\right) = \\ = \lambda_i^n E + \lambda_i^{n-1} B_1 + \lambda_i^{n-2} B_2 + \cdots + \lambda_i^2 B_{n-2} + \lambda_i B_{n-1} - \lambda_i^{n-1} A - \lambda_i^{n-2} A B_1 - \cdots - \\ -\lambda_i A B_{n-2} - A B_{n-1} = \lambda_i^n E + \lambda_i^{n-1}(B_1 - A) + \\ +\lambda_i^{n-2}(B_2 - A B_1) + \cdots + \lambda_i (B_{n-1} - A B_{n-2}) - A B_{n-1} = \\ = \lambda_i^n E - p_1 \lambda_i^{n-1} E - p_2 \lambda_i^{n-2} E - \cdots - p_{n-1} \lambda_i E - p_n E = 0\end{gathered} \qquad (1.63)$$

Отсюда следует, что для любого столбца $\vec{y}$ построенной матрицы Qi $(\lambda_i E - A) \cdot \vec{y} = \vec{0}$, т.е. $\vec{y}$ – собственный вектор матрицы A, отвечающий собственному значению λ_i.

Вычисляя собственные векторы описанным образом, нет необходимости находить все столбцы матрицы Qi. Следует ограничиться вычислением одного (любого) столбца, для чего удобно пользоваться рекуррентной формулой:

$$\vec{y}_0 = \vec{e}, \;\; \vec{y}_k = \lambda_i \vec{y}_{k-1} + \vec{b}_k \;\; k = 1,2,\dots,n-1, \qquad (1.64)$$

где $\vec{b}_k$ – одноименный с вычисляемым столбцом матрицы Qi столбец матрицы Bk, а $\vec{e}$ – одноименный столбец единичной матрицы.

Тогда собственный вектор $\vec{x}_i$ матрицы A, отвечающий собственному значению λ_i, есть $\vec{x}_i = \vec{y}_{n-1}$.

1.4 Обзор технологии NVIDIA CUDA

CUDA – это архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров) [31].

В состав CUDA, кроме самой видеокарты, входят программные компоненты, объединённые в пакет CUDA Toolkit. В ранних версиях так же требовался дополнительный драйвер CUDA, но в настоящее время он включен в стандартные пакет драйверов для видеокарты NVIDIA. CUDA Toolkit содержит компилятор nvcc, транслирующий исходный код программ в промежуточный ассемблерный код, библиотеки, необходимые для работы с платформой, а так же дополнительное программного обеспечение, например Nsight Monitor.

Многие разработчики программного обеспечения, ученые и исследователи широко применяют CUDA в различных областях: физика

и химия, обработка видео и изображений, в медицине. С помощью нее моделируют динамику жидкостей, восстанавливают изображения, которые были получены при помощи компьютерной томографии, проводят сейсмический анализ, трассировку лучшей и многое другое.

Платформа параллельных вычислений CUDA обеспечивает набор расширений для языков C и C++, позволяющих выражать как параллелизм данных, так и параллелизм задач на уровне мелких и крупных структурных единиц. Такой подход позволяет писать код, исполняемый на GPU, так и код, выполняющийся на CPU. Кроме того, программист может выбирать между средствами разработки: использовать высокоуровневые языки или же открытые стандарты, такие как директивы OpenACC.

1.4.1 Модель памяти CUDA

В спецификациях к архитектуре CUDA выделяют 6 видов памяти:

- Регистровая;
- Разделяемая;
- Локальная;
- Глобальная;
- Константная;
- Текстурная.

Регистровая память представляет собой сверхоперативное запоминающее устройство в мультипроцессоре. Данный вид памяти располагается на мультипроцессоре, не кэшируется и обладает максимальной скоростью среди других видов. Для большей эффективности надо стараться занимать как можно меньше регистров, т.к. на каждый поток выделяется некоторое количество регистров и при их нехватке количество потоков будет ограничено.

Константная память физически не отделена от глобальной памяти. Но в отличии от глобальной может кэшироваться и в таком случае скорость работы с данными достаточно высока. Однако, если необходимые данные отсутствуют в кэше, то их чтение будет выполнено с задержками в 400-600 тактов.

Локальная память занимает часть DRAM (динамическая память с произвольным доступом) и используется, когда локальные данные процедур занимают слишком большой размер. Этот вид памяти так же не кэшируется и имеет низкую скорость доступа.

Глобальная память занимают основную часть DRAM. Обладает высокой пропускной способностью (более 100Гб/с) и возможностью произвольной адресации глобальной памяти. Но также, как и локальная память не кэшируется и обладает низкой скоростью работы.

Разделяемая память обладает скоростью, сравнимой с регистровой памятью. Основное назначение — это обеспечение взаимодействия между потоками, в связи с чем эта память доступна на запись и на чтение из всех потоковых процессоров в мультипроцессоре.

Текстурная память – блок памяти, доступный только на чтение всеми мультипроцессорами. Выборка из этого типа памяти происходит при помощи текстурных блоков видеочипа. За счёт этого возможно выполнение линейной интерполяции без каких-либо дополнительных затрат.

1.4.2 Математические библиотеки CUDA

В состав CUDA Toolkit входит набор таких математических библиотек, как cuBLAS, cuSPARSE и другие [30]. Общей особенностью этих пакетов является то, что все они предназначены для произведения параллельных вычислений на GPU.

CuBLAS представляет собой реализацию BLAS (базовых подпрограмм линейной алгебры) поверх среды CUDA. Для использования API cuBLAS программист должен предварительно выделить и заполнить память видеокарты для матриц, векторов, а также, при необходимости, других возможных параметров функций. Ранние версии библиотеки имели собственные функции для выделения памяти и копирования данных, но в последних версиях платформы от данной практики было решено отказаться и теперь для этих целей используются общие функции CUDA.

Ещё одной особенностью cuBLAS является индексация массивов. Для максимальной совместимости с существующими средами Fortran библиотека cuBLAS использует индексирование массивов, начинающее с единицы. Но, т.к. в C и C++ используется индексация с 0, то для совместимости с этими языками в некоторые функции библиотеки передаётся дополнительный параметр, который указывает, с какой позиции начинается индексация.

В отличии от cuBLAS, библиотека cuSPARSE предназначена для обработки разреженных матриц. Библиотечные функции можно разделить на четыре категории:

- Операции между вектором в разреженном формате и вектором в плотном формате;
- Операции между матрицей в разреженном формате и вектором в плотном формате;
- Операции между матрицей в разреженном формате и набором векторов в плотном формате;
- Операции, которые допускают преобразование между различными матричными форматами.

1.5 Формат Compressed Sparse Row

Формат CSR предназначен для компактного хранения разреженных матриц [16]. Он хранит разреженную матрицу M размером m x n в виде трех одномерных массивов:

- Массив A, который имеет длину, равную количеству ненулевых элементов матрицы M и, соответственно, хранящий все не нулевые элементы в порядке слева направо и сверху вниз;
- Массив JA, размерности аналогичной массиву A, хранит номера столбцов каждого элемента A;
- Массив IA, размером m + 1, в котором i-й элемент указывает, с какой позиции в массивах A и JA начинается i-я строка.

Для примера возьмем следующую разреженную матрицу:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 5 & 0 \\ 0 & 0 & 6 & 8 \\ 7 & 0 & 0 & 1 \end{pmatrix}$$

Для нее m = 4 и массивы имеют вид:

A = [1, 2, 3, 4, 6, 8, 7, 1]

JA = [1, 4, 2, 3, 3, 4, 1, 4]

IA = [1, 3, 5, 7, 9]

Данный формат имеет следующий преимущества: требует меньше памяти, по сравнению с классическим вариантом хранения матрицы в памяти, обеспечивает быстрый доступ к строке и умножение матрицы на вектор.

2. РАЗРАБОТКА АЛГОРИТМА РЕШЕНИЯ ОБОБЩЕННОЙ ЗАДАЧИ ПОИСКА СОБСТВЕННЫХ ЗНАЧЕНИЙ И ВЕКТОРОВ ДЛЯ РАЗРЕЖЕННЫХ МАТРИЦ НА ОСНОВЕ LU-РАЗЛОЖЕНИЯ ПРАВОЙ ЧАСТИ УРАВНЕНИЯ

2.1 Преобразование обобщенной задачи поиска собственных значений и векторов к классической задаче

Данный алгоритм применяется для решения обобщенной задачи поиска собственных значений вида

$$Ax = \lambda Bx \tag{2.1}$$

где матрицы A и B не вырожденные общего вида (могут быть не положительно определенными, не симметричными), λ – собственное значение, x – собственный вектор.

Для решения этой задачи, необходимо свести её к задаче вида

$$Cx = \lambda x \tag{2.2}$$

В качестве первого шага, используем LU разложение матрицы B

$$B = LU \tag{2.3}$$

Тогда

$$Ax = \lambda LUx \tag{2.4}$$

Произведем замену произведения матрицы U на вектор x:

$$y = Ux \tag{2.5}$$

$$x = U^{-1}y \tag{2.6}$$

Тогда

$$AU^{-1}y = \lambda Ly \tag{2.7}$$

Умножим обе части уравнения слева на матрицу обратную L

$$L^{-1}AU^{-1}y = \lambda y \tag{2.8}$$

Перемножим матрицы L^{-1}, A и U^{-1} и обозначим результат как матрицу C:

$$C = L^{-1}AU^{-1} \tag{2.9}$$

Тогда исходное уравнение сведется к следующему:

$$Cy = \lambda y \tag{2.10}$$

откуда, зная y, можно вычислить собственный вектор x умножив матрицу обратную U на y.

2.2 Использование неполного LU-разложения матриц

Одним из широко известных способов разложения матриц на множители является LU-факторизация, позволяющая представить матрицу A в виде

$$A = LU, \tag{2.11}$$

где L и U – нижне- и верхнетреугольные матрицы соответственно.

Однако, алгоритм факторизации непригоден для разреженных матриц, так как ведет к заполнению портрета, т.е. появлению в матрицах L и U ненулевых элементов в тех позициях, для которых $a_{ij} = 0$, - и как следствие, резкому увеличению объёма памяти, требуемой для хранения матриц.

Поэтому, вместо задачи нахождения факторизации (2.11) сформулируем другую задачу. Для заданной матрицы A потребуем представить её в виде

$$A = LU + R, \tag{2.12}$$

где матрицы в правой части удовлетворяют следующим свойствам:

- Матрицы L и U являются нижнетреугольной и верхнетреугольной соответственно;
- $P_L \subset P_A$ и $P_U \subset P_A$;
- $P_A \cap P_R = \emptyset$.

Тогда приближенное представление $A \approx LU$ называется неполной LU-факторизацией матрицы A или коротко её ILU-разложением.

Использование ILU – разложение на CSR матрицей позволит эффективно использовать память. Т.к. портрет матрицы не изменится, то не потребуется выделять дополнительную память для хранения новой матрицы, а результат можно будет записать на место исходной матрицы. Массивы, содержащие номера столбцов элементов и позиции, с которых начинаются новые строки, не потребует изменения. Таким образов будет обновлен только массив, который содержит ненулевые элементы.

Алгоритм ILU-разложения представлен на рисунке 2.1.

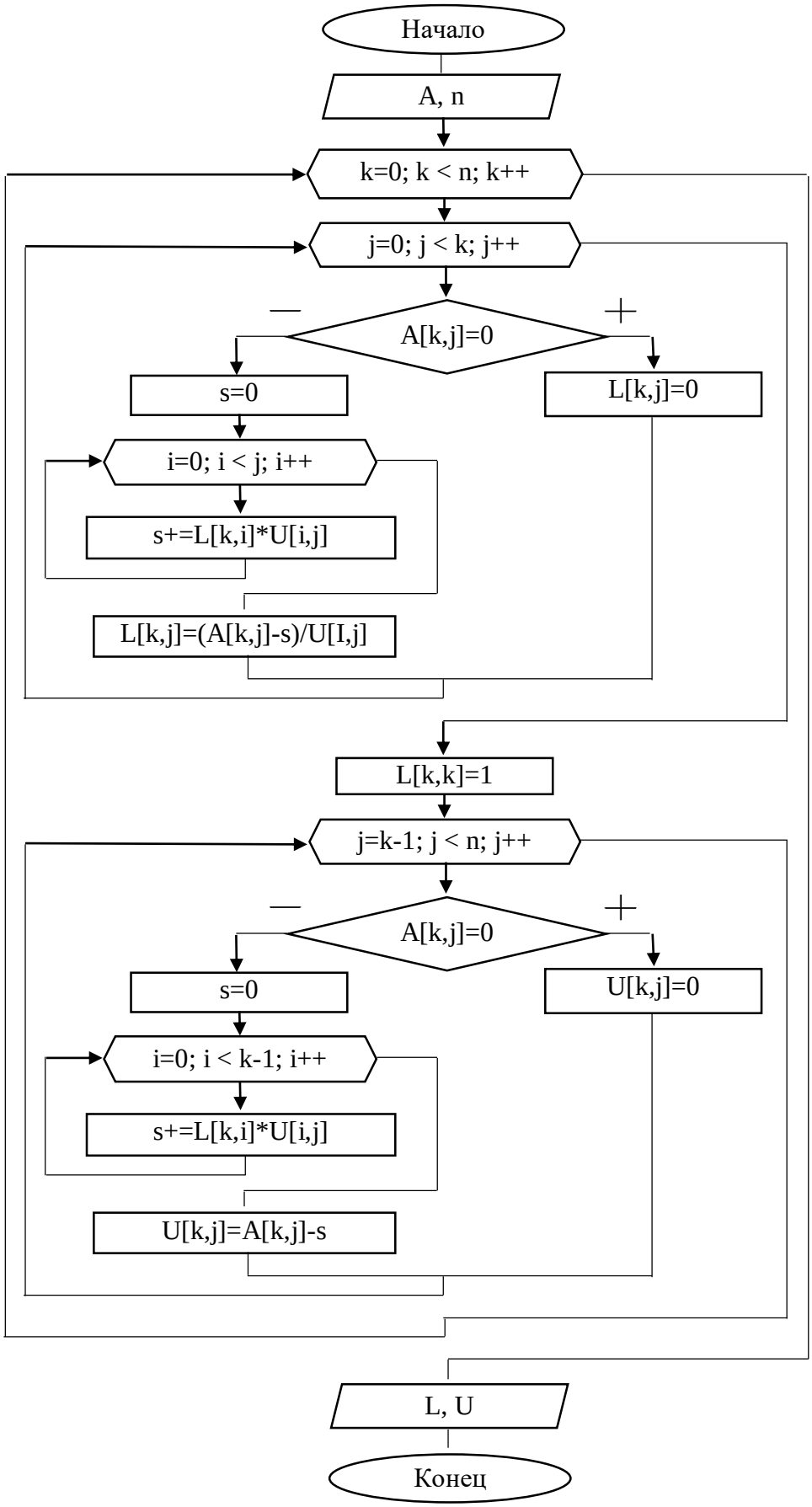

Рисунок 2.1. Алгоритм ILU-разложения

2.3 Степенной метод со сдвигами нахождения собственных значений

Степенной метод позволяет найти наибольшее по модулю собственное значение и собственный вектор матрицы A.

Основным достоинством данного метода является возможность эффективного использования разреженной матрицы, т.к. векторы получаются только с помощью умножения матрицы на вектор, а исходная матрица остаётся разреженной даже после выполнения сдвига.

В качестве нулевого приближения берут произвольный вектор $x^{(0)}$ и последовательно строят следующие приближения:

$$x^{(1)} = A \cdot x^{(0)}$$

$$x^{(2)} = A \cdot x^{(1)} \qquad (2.13)$$

$$\dots$$

$$x^{(k)} = A \cdot x^{(k-1)}$$

Для соответствующих координат векторов данной последовательности буду выполняться выражение

$$\lim_{k\to\infty} \frac{x^{(k)}}{x^{(k-1)}} = \lambda^{(k)} \qquad (2.14)$$

Для того, чтобы степенной метод можно было использовать для нахождения не только максимального собственного значения применяется следующее преобразование

$$A' = A - \sigma E \qquad (2.15)$$

где A – исходная матрица, E – единичная матрица, σ – некоторое число.

Обозначим λ^* как следующее собственное значение после λ. Тогда, если $\sigma = \lambda$, то число $\lambda - \lambda^*$станет минимальным по модулю, а максимальным по модулю окажется сдвиг другого собственного значения.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ РАЗРАБОТАННОГО АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ CUDA

3.1 Программная реализация параллельной версии алгоритма

Для изоляции вычисления собственных значений и векторов от остального кода, все с этим связанные операции удобно вынести в отдельную функцию. Данная функция могла бы иметь только следующие входные значения:

- Матрица A в CSR формате;
- Матрица B в CSR формате;
- Точность вычислений.

Однако, ввиду особенностей языка программирования C++, к этим данным необходимо добавить следующие:

- Размерность матрицы;
- Количество ненулевых элементов в матрице A;
- Количество ненулевых элементов в матрице B;
- Ссылка на массив, куда будут записаны вычисленные собственные значения;
- Ссылка на двумерный массив, куда будут записаны вычисленные собственные вектора.

Так как вычисления будут производиться на GPU, то данные, над которыми будут проводиться операции необходимо скопировать в память видеокарты, предварительно выделив для них достаточное количество места. Пример выделения памяти для массива ненулевых элементов матрицы A представлен на листинге 3.1.

Листинг 3.1. Выделение памяти для матрицы

```
double *devA;
cudaStat = cudaMalloc((void**)&devA, iNNZA * sizeof(double));
if (cudaStat != cudaSuccess)
    return -1;
```

Пример копирования данных из оперативной памяти в память GPU представлен на рисунке 3.2.

Листинг 3.2. Копирование данных из ОЗУ и в память ГПУ

```
cudaStat = cudaMemcpy(devA, a, iNNZA * sizeof(double), cudaMemcpyHostToDevice);
if ((cudaStat != cudaSuccess)) {
	return -2;
}
```

Согласно алгоритму, первым этапом вычислений является ILU разложение. В библиотеке cuSPARSE, которая входит в состав Cuda Toolkit, есть специальный метод, который произведет необходимую операцию над данными. Однако перед тем, как передать в него данные, необходимо выделить достаточно памяти буферной памяти. Для этого предусмотрены специальные функции (листинг 3.3), куда необходимо передать информация о размере матрицы, количестве ненулевых элементов матрицы B и ссылку, куда будут записаны прогнозируемые размеры буферов.

Листинг 3.3. Выделение буфера для ILU-разложения

```
	cusparseDcsrilu02_bufferSize(cusparseHandle, iEqNum, iNNZB, descr_B, devB, devIBcsr, devJB, info_B, &pBufferSize_M);
	cusparseDcsrsv2_bufferSize(cusparseHandle, trans_L, iEqNum, iNNZB, descr_L, devB, devIBcsr, devJB, info_L, &pBufferSize_L);
	cusparseDcsrsv2_bufferSize(cusparseHandle, trans_U, iEqNum, iNNZB, descr_U, devB, devIBcsr, devJB, info_U, &pBufferSize_U);
	pBufferSize = std::max(pBufferSize_M, std::max(pBufferSize_L, pBufferSize_U));
	cudaMalloc((void**)&pBuffer, pBufferSize);
```

После того как память для буфера была выделена, вызывается функция, которая произведет неполное lu разложение. Матрица L и матрица U будут записаны на место матрицы B. Таким образом, кроме временно выделенного места для буфера, больше память не расходуется.

Следующий этап заключается в вычислении матрицы C, которая получается в результате произведения матрицы обратной L на матрицу A и на матрицу обратную U. Однако, предварительно требуется получить обратные матрицы. После чего необходимо произвести две операции перемножения матриц. В библиотеке cuSPARSE для этих целей предусмотрены два метода: cusparseXcsrgemmNnz и cusparseScsrgemm. Что бы осуществить умножение необходимо использовать их последовательно. Так cusparseXcsrgemmNnz заполнит массив, который содержит позиции начала строк матрицы в CSR формате и количество ненулевых элементов. А метод cusparseScsrgemm основываясь на этих данных, заполнит два оставшихся массива, которые хранят ненулевые элементы и их номера столбцов:

Листинг 3.4. Умножение матриц

```
cusparseXcsrgemmNnz(cusparseHandle, trans_L, trans_L, iEqNum, iEqNum, iEqNum, descr_Linv, iNNZB, devIBcsr, devJB, descr_A, iNNZA, devIAcsr, devJA, descr_C, devICcsr, nnzTotalDevHostPtr);
cusparseDcsrgemm(cusparseHandle, trans_L, trans_L, iEqNum, iEqNum, iEqNum, descr_Linv, iNNZB, devLinv, devIBcsr, devJB, descr_A, iNNZA, devA, devIAcsr, devJA, descr_C, devC, devICcsr, devJC);
```

Данная операция производится два раза. В первый раз умножая матрицу A слева на матрицу обратную L, а во второй, матрицу- результат предыдущей операции на матрицу обратную U. В результате чего в память будет помещена матрица C, которая является результатом вычислений.

Следующим этапом является вычисление собственных значений для матрицы C. Метод для решения данной задачи уже предусмотрен в библиотеке cuSOLVER, но для того, чтобы им воспользоваться, требуется вычислить приближенные собственные значения. Воспользуемся степенным методом со сдвигом. В начале выделяется память для массива собственных векторов во время работы метода и заполняется единицами (листинг 3.5).

Листинг 3.5. Выделение памяти для массива собственных векторов и заполнение его единицами

```
double *devX;
cudaMalloc((void**)&devX, iEqNum * sizeof(double));
setXone << <(iEqNum - 1 + BLOCK_SIZE) / BLOCK_SIZE, BLOCK_SIZE >> >(devX, iEqNum);
cudaDeviceSynchronize();
```

Теперь в цикле выполняется вычисление собственных значений. Для этого сначала умножается матрица A на приближенный собственный вектор, который на первой итерации заполнен единицами, и записывается результат как собственный вектор.

Листинг 3.6. Умножение матрицы на вектор

```
cusparseDcsrmv(cusparseHandle, trans_L, iEqNum, iEqNum, nnzA, &alpha, descr_A, devAA, devIAAcsr, devJAA, devX, &gamma, devX);
```

После этого выполняется нормировка вектора (листинг 3.7), это необходимо для того, чтобы не допустить выхода за диапазон типа данных double.

Листинг 3.7. Нормировка вектора

```
cublasDnrm2(cublasHandle, iEqNum, devX, 1.0, &normX);
normX = 1.0 / normX;
cublasDscal(cublasHandle, iEqNum, &normX, devX, 1.0);
```

Далее повторно выполним аналогичную операцию умножения и запишем результат в другой массив. Проведем скалярное умножение (листинг 3.8) полученного результата с ранее вычисленным приближенным значением собственного вектора. В итоге мы получим приближенное собственное значение:

Листинг 3.8. Скалярное умножение векторов

```
cublasDdot(cublasHandle, iEqNum, devC, 1.0, devX, 1.0, &lambda[k]);
```

После того, как работа цикла будет завершена, необходимо произвести сдвиг, чтобы перейти к вычислению следующего собственного значения.

Когда будут найдены все приближенные собственные значения, можно воспользоваться функцией cusolverSpDcsreigvsi, которая вычислит собственное значение и собственный вектор с заданной точностью. Данный метод одновременно вычисляет значения только одного собственного значения и собственного вектора, поэтому её необходимо вызывать в цикле, передавая на вход каждый раз следующее приближенное собственное значение, вычисленное ранее:

Листинг 3.9. Нахождение собственных значений и векторов с помощью функции cusolverSpDcsreigvsi

```
for (int k = 0; k < iEqNum; k++) {
        cusolverSpDcsreigvsi(cusolverHandle, iEqNum, NNZA, descr_A, devA,
devIAcsr, devJA, lambda[k], devX0, maxite, eps, devMu, devX);
        cudaMemcpy(egval + k, devMu, sizeof(double), cudaMemcpyDeviceToHost);
        cudaMemcpy(eigvec[k], devX, iEqNum * sizeof(double),
cudaMemcpyDeviceToHost);
}
```

По окончанию работы цикла, достаточно освободить память от временных переменных. Собственные значения и вектора будут доступны по ссылкам, которые передавались из управляющего метода.

3.2 Оценка эффективности разработанного алгоритма на основе проведения вычислительных экспериментов.

Для проведения оценки эффективности проводился ряд испытаний при вычислении собственных значений и векторов матриц различной размерности. Ввиду того, что персональный компьютер, на котором производятся испытания, не обладает достаточной мощностью, чтобы обрабатывать большие объёмы данных (Intel Core i5-4210u, Nvidia GeForce gt840m 2gb), в качестве оптимальных и достаточных для демонстрации эффективности были выбраны следующие размеры матриц: 500, 1000, 2000, 3000 и 4000. Все вычисления производились над различными наборами данных и для каждой размерности проводилось по 3 испытания для того чтобы получить в результате более объективные результаты. Результаты, полученные в результате испытаний, приведены в таблице 3.1. Все значения были округлены до целых секунд.

Таблица 3.1

Результаты испытаний параллельной версии алгоритма

Порядок матрицы	Испытание №1, с	Испытание №2, с	Испытание №3, с	Среднее время 3 испытаний, с
500	10	9	10	10
1000	35	31	29	32
2000	189	186	193	189
3000	420	431	433	428
4000	644	631	633	636

Для наглядной демонстрации эффективности работы программы, наиболее рациональным способом было выбрано сравнение с

последовательной версией программы. Были проведены аналогичные испытания, результаты которых приведены в таблице 3.2

Таблица 3.2

Результаты испытаний последовательной версии алгоритма

Порядок матрицы	Испытание №1	Испытание №2	Испытание №3	Среднее время 3 испытаний
500	4	4	4	4
1000	42	36	44	41
2000	382	394	384	386
3000	1328	1300	1318	1316
4000	3520	3543	3537	3533

Для более удобной демонстрации результатов испытаний параллельной и последовательной версии, они представлены в виде графика на рисунке 3.1.

Таблица 3.3

Объединенные результаты испытаний программ

	500	1000	2000	3000	4000
Параллельная	10с	32с	189с	428с	636с
Последовательная	4с	41с	386с	1316с	3533с

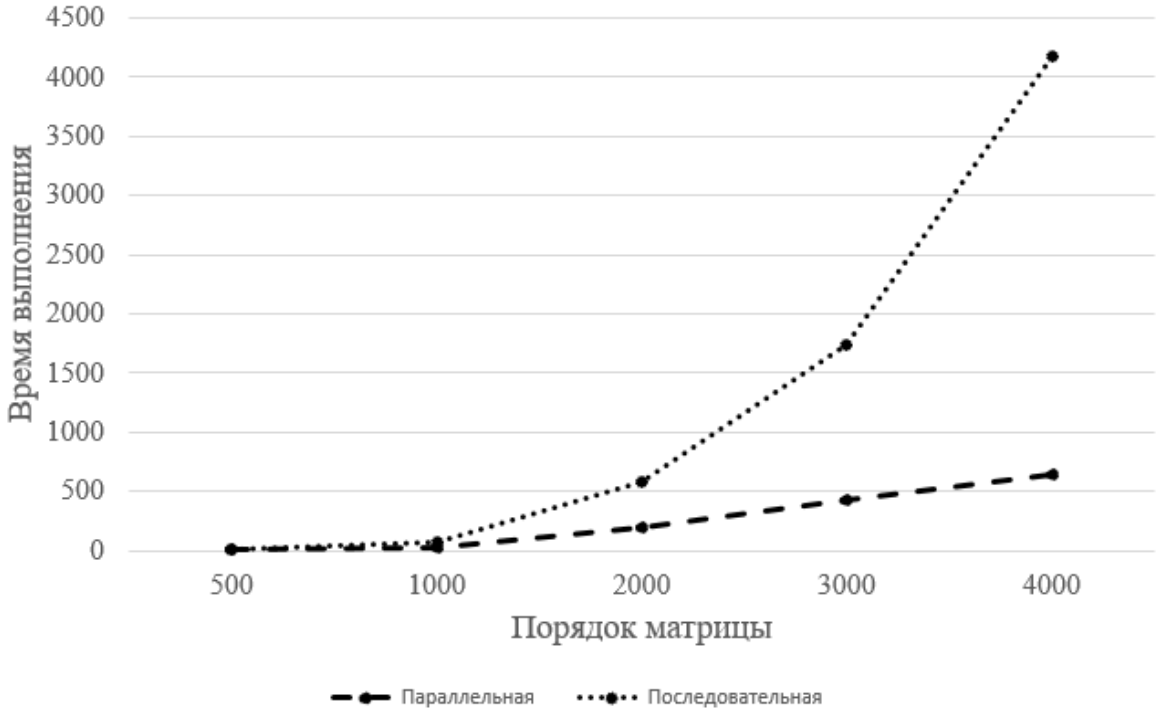

Рисунок 3.1. Сравнительный график скорости выполнения последовательной и параллельной версии программы

Для получения значения ускорения воспользуемся следующей формулой:

$$S_p(n) = \frac{T_1(n)}{T_p(n)}, \tag{3.1}$$

где n – размерность матрицы, S_p – ускорение параллельной версии, T_1 – время выполнения последовательной версии, T_p – время выполнения параллельной версии.

Данные об ускорении, полученные по формуле 3.1 представлены в таблице 3.4 и на рисунке 3.2.

Таблица 3.4

Ускорения параллельной версии программы

Порядок матрицы	500	1000	2000	3000	4000
Ускорение	0,4	1,28	2,04	3,08	5,56

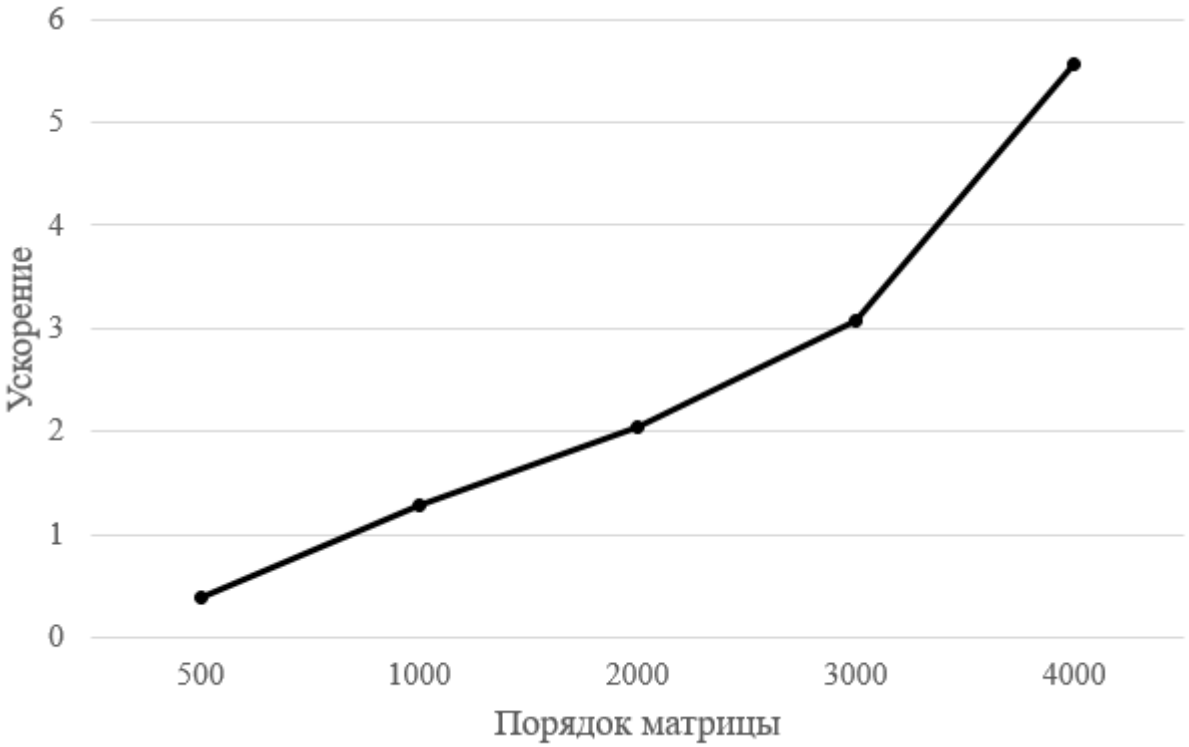

Рисунок 3.2. График, демонстрирующий ускорение параллельной версии

Исходя из полученных результатов можно сделать вывод, что при вычислении собственных значений и векторов для матриц размером менее 1000 использование параллельной версии не имеет никаких значимых преимуществ. Однако ситуация меняется при размерности матрицы более 2000. Уже при 3000 ускорение составляет 3 единицы, а при 4000 превышает 5.

ЗАКЛЮЧЕНИЕ

Целью данной выпускной квалификационной работы была разработка и программная реализация итерационного метод решения обобщенной задачи поиска собственных значений и векторов на основе LU-разложения правой матрицы с использованием технологии CUDA.

Все поставленные задачи были достигнуты, а именно:

- Произведен обзор существующих методов решения задачи поиска собственных значений и векторов и сделан вывод, что большинство существующих методов предназначены только для нахождения собственных значений симметричный и положительно определенных матриц, а остальные не эффективно расходуют память;
- Разработан алгоритм решения обобщенной задачи поиска собственных значений и векторов для разреженных матриц на основе LU-разложения правой части уравнения, который может решать несимметричные и не положительно определенные матрицы;
- Выполнена программная реализация разработанного алгоритма с использованием технологии CUDA, в которой используется формат CSR для хранения матриц и минимально расходуется память;
- Осуществлена оценка эффективности разработанного алгоритма на основе проведения вычислительных экспериментов, по результатам которых сделан вывод, что параллельный алгоритм позволяет ускорить вычисления в 5,5 раз.

По результатам проведенных вычислительных экспериментов можно сделать вывод, что параллельная версия реализованного приложения выполняет задачу поиска собственных значений и векторов быстрее последовательной версии при размерности матриц более 1000. Кроме того, во время выполнения расходуется минимальное количество памяти, за счет чего можно осуществлять вычисления над большими объемами данными, по

сравнению с существующими реализациями подобных задач. Исходя из этого, можно заключить, что разработанное приложение можно использоваться для решения различных прикладных задач.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. NVIDIA Developer Documentation. URL: https://docs.nvidia.com/
2. Абаффи Й. Математические методы для линейных и нелинейных уравнений: проекционные ABS-алгоритмы / Й. Абаффи, Э. Спедикато. — Москва: Мир, 1996.
3. Баландин М.Ю. Методы решения СЛАУ большой размерности / М.Ю. Баландин, Э.П. Шурина. - Новосибирск: НГТУ, 2000. — 70 с.
4. Бахвалов Н.С. Практикум по алгебре / Н.С. Бахвалов, А.И. Кострикин. – Москва: МГУ, 1983.
5. Березин И.С. Методы вычислений / И.С. Березин, Н.П. Жидков. – Москва: Физматгиз, 1962. – 635 с.
6. Богачев К.Ю. Практикум на ЭВМ. Методы решения линейных систем и нахождения собственных значений / К.Ю. Богачев. – Москва: МГУ, 1998. – 137с.
7. Боглаев Ю.П. Вычислительная математика и программирование. Москва: Высшая школа, 1990. – 544 с.
8. Бордовицина Т.В. Современные численные методы в задачах небесной механики. – Москва: Наука, 1984. – 136 с.
9. Воеводин В.В. Математические модели и методы в параллельных процессах. – Москва: Наука, 1986. – 259 с.
10. Волков Е.А. Численные методы. – Москва: Наука, 1987. – 248 с.
11. Вычисления на графических процессорах (GPU) в задачах математической и теоретической физики / Перепёлкин Е.Е., Садовников Б.И., Иноземцева Н.Г. – Москва: URSS, 2014. – 176 с.
12. Вычислительные методы для инженеров / А.А. Амосов, Ю.А. Дубинский, Н.В. Копченова — Москва: Высшая школа, 1994. — 544 с.
13. Годунов С.К. Современные аспекты линейной алгебры. — Новосибирск: Научн. книга, 1997.

14. Демидович Б.П. Основы вычислительной математики / Б.П. Демидович, И.А. Марон. – Москва: Наука, 1966. – 664 с.
15. Долгополов Д.В. Методы нахождения собственных значений и собственных векторов матриц / Д.В. Долгополов. -Санкт-Петербург: СПбГТИ(ТУ), 2005. – 39с
16. Ильин В.П. Методы неполной факторизации для решения линейных систем. — Москва: Физматлит, 1995.
17. Калиткин Н.Н. Численные методы / Н.Н. Калиткин. -Санкт-Петербург: БХВ-Петербург, 2011. – 592 с.
18. Киреев В.И. Численные методы в примерах и задачах: Учеб. пособие / Киреев В.И., Пантелеев А.В. – Москва: Высшая школа, 2006. – 480 с.
19. Курош А.Г. Курс высшей алгебры. – Москва: Наука, 1975. – 431 с.
20. Марчук Г.И. Методы вычислительной математики, 3-е изд. – Москва: Наука, 1989.
21. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. — Москва: Мир, 1991.
22. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учебное пособие / Боресков А.В., Марковский Н.Д., Микушин Д.Н. и др. – Москва: МГУ, 2012. – 336 с.
23. Пирумов У. Г. Численные методы: Учеб. пособие. Москва: Дрофа, 2004. – 224 с.
24. Писсанецки С. Технология разреженных матриц. — Москва: Мир,1988.
25. Плис А.И. Лабораторный практикум по высшей математике / Плис А.И. Сливина Н.А. – Москва: Высшая школа, 1994. – 416 с.
26. Плохотников К.Э. Вычислительные методы. Теория и практика в среде MATLAB (курс лекций). – Москва: Телеком, 2009. – 496 с.
27. Самарский А.А. Введение в численные методы. – Москва: Наука, 1987. – 286 с.
28. Современные информационные технологии в задачах навигации и наведения беспилотных маневренных летательных аппаратов / К.К.

Веремеенко, С.Ю. Желтов, Н.В. Ким и др. – Москва: ФИЗМАТЛИТ, 2009. – 562с.

29. Тихонов А.Н. Вводные лекции по прикладной математике / Тихонов А.Н., Костомаров Д.П. – Москва: Наука, 1984. – 160 с.
30. Уилкинсон Дж. Алгебраическая проблема собственных значений / Дж. Уилкинсон. -Москва: Наука, 1970. – 565 с.
31. Фаддеев Д.К. Вычислительные методы линейной алгебры / Д.К. Фаддеев, В.Н. Фаддеева. – Москва: Физматгиз, 1963. – 734с.
32. Хемминг Р.В. Численные методы. – Москва: Наука, 1972. – 400 с.
33. Хорн. Р. Матричный анализ / Р. Хорн, Ч. Джонсон. – Москва: Мир, 1989.
34. Численные методы / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельков. – Москва: Наука, 1987.