Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 4

по курсу «Анализ алгоритмов»

на тему: «Параллельные вычисления на основе нативных потоков»

Студент ИУ7-54Б (Группа) _____ (Подпись, дата) Писаренко Д. П. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Кластеризация данных является важным инструментом в области машинного обучения. Она позволяет группировать данные на основе их сходства и отделить их от остальных [1].

Целью данной лабораторной работы является получение навыков организации параллельного выполнения операций.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать алгоритм кластеризации k-средних;
2) разработать параллельную версию алгоритма;
3) определить средства программной реализации;
4) реализовать данные алгоритмы;
5) выполнить замеры процессорного времени работы различных реализаций алгоритма и произвести анализ полученных данных.

1 Аналитическая часть

В данном разделе приведена информация о понятии кластеризации и нечетком алгоритме кластеризации с-средних.

1.1 Нечеткий алгоритм кластеризации с-средних

Кластерный анализ — это ряд математических методов интеллектуального анализа данных, предназначенных для разбиения множества исследуемых объектов на компактные группы, называемые кластерами. Под объектами кластерного анализа подразумеваются предметы исследования, нуждающиеся в кластеризации по некоторым признакам. Признаки объектов могут иметь как непрерывные, так и дискретные значения [**c-means**].

Метод с-средних — итеративный нечеткий алгоритм кластеризации. В данном методе кластеры являются нечеткими множествами, и каждый объект из выборки исходных данных относится одновременно ко всем кластерам с различной степенью принадлежности. Таким образом, матрица принадлежности объектов к кластерам (или матрица разбиения) содержит не бинарные, а вещественные значения, принадлежащие отрезку $[0; 1]$ [**c-means**].

Пусть X — исходный набор данных размера N. Обновление матрицы принадлежности и списка центров кластеров производится в 5 этапов:

1) инициализация матрицы центров кластеров W случайными значениями;

2) инициализация матрицы разбиения $U = (\mu_{ij})$ следующим образом:

$$\mu_{ij}^{(t)} = \frac{1}{\sum_{l=1}^{C} \left(\frac{d_{ij}}{d_{il}}\right)^{\frac{2}{m-1}}}, i = 1, \ldots, N; j, l = 1, \ldots C$$
$$\mu_{ij} = \begin{cases} 1, & \text{если } d_{ij} = 0 \\ 0, & \text{для } l \neq j, \end{cases} \tag{1.1}$$

где t — номер итерации,
C — количество кластеров,
m — показатель нечеткости, регулирующий точность разбиения,
d_{ij} — расстояние от x_i до w_j, $d_{ij} = \|x_i - w_j^{(t)}\|$;

3) увеличить t на 1 и рассчитать матрицу $W^{(t)}$ по формуле (**??**)

$$W_j^{(t)} = \frac{\sum_{i=1}^{N} \left(\mu_{ij}^{(t-1)}\right)^m x_i}{\sum_{i=1}^{N} \left(\mu_{ij}^{(t-1)}\right)^m}, j = 1, \ldots, C; \tag{1.2}$$

4) вычислить матрицу разбиения $U^{(t)}$ согласно соотношению (**??**);

5) если $\|U^{(t)} < U^{(t-1)}\| \geq \varepsilon$ перейти на шаг 3 [**c-means-steps**].

1.2 Использование потоков

В данной задаче возможно использование потоков при заполнении матрицы принадлежности: матрица разбивается на n групп строк, где n — количество потоков. Каждая такая группа обрабатывается параллельно. Поскольку элементы матрицы вычисляются независимо друг от друга (см. (**??**)) в использовании средств синхронизации (мьютекс, семафор) нет необходимости.

Вывод

В данном разделе было рассмотрено понятие кластеризации. Также был описан нечеткий алгоритм с-средних.

2 Конструкторский раздел

В этом разделе будет представлено описание используемых типов данных, а также схематические изображения алгоритмов сортировок: блочной, быстрой и выбором.

2.1 Требования к программному обеспечению

Программа должна поддерживать два режима работы: режим массового замера времени и режим сортировки введенного массива.

Режим массового замера времени должен обладать следующей функциональностью:

— генерировать массивы различного размер для проведения замеров;

— осуществлять массовый замер, используя сгенерированные данные;

— результаты массового замера должны быть представлены в виде таблицы и графика.

К режиму сортировки выдвигается следующий ряд требований:

— возможность работать с массивами разного размера, которые вводит пользователь;

— наличие интерфейса для выбора действий;

— на выходе программы массив, отсортированный тремя алгоритмами по возрастанию.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

— целое число представляет количество элементов в массиве;

— список целых чисел;

2.3 Разработка алгоритмов

2.4 Оценка трудоемкости алгоритмов

Модель для оценки трудоемкости алгоритмов состоит из шести пунктов:

1) $+, -, =, + =, - =, ==, ||, \&\&, <, >, <=, >=, <<, >>, []$ — считается, что эти операции обладают трудоемкостью в 1 единицу;

2) $*, /, * =, / =, \%$ — считается, что эти операции обладают трудоемкостью в 2 единицы;

3) трудоемкость условного перехода принимается за 0;

4) трудоемкость условного оператора рассчитывается по следующей формуле

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай} \\ max(f_1, f_2), & \text{худший случай} \end{cases}, \tag{2.1}$$

где f_1 — трудоемкость блока, который вычисляется при выполнении условия, а f_2 — трудоемкость блока, который вычисляется при невыполнении условия;

5) трудоемкость цикла рассчитывается по следующей формуле

$$\begin{aligned} f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}); \end{aligned} \tag{2.2}$$

6) вызов подпрограмм и передача параметров принимается за 0.

2.4.1 Трудоемкость алгоритма блочной сортировки

В данной реализации размер блока обозначается как k, трудоемкость операции добавления и удаления элемента из вектора равна 2.

Лучший случай: массив отсортирован, элементы распределены равномерно (все блоки содержат одинаковое число элементов), расчет трудоемкости данного случая приведен в следующей формуле

$$
\begin{aligned}
f_{best} = 1 + 1 + \frac{n}{k} \cdot (1 + 2 + f_{shaker} + 2 + 1 + 4 + \\
+ k \cdot (3 + 1 + 4)) + 1 + 1 + \\
+ \frac{n}{k} \cdot (1 + 4 + 1 + 1 + 5 + 1 + 4 + 1 + 1 + n \cdot (5)) = \\
= 4 + \frac{29 \cdot n + n \cdot f_{shaker} + 5 \cdot n^2}{k} + 8 \cdot n = \\
= 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + n \cdot (14.5 + \frac{k}{2}) + \frac{5 \cdot n^2}{k} = \\
= 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + 14.5 \cdot n + \frac{n \cdot k}{2} + \frac{5 \cdot n^2}{k} = \\
= \frac{5 \cdot n^2}{k} + 22.5 \cdot n + 29 \cdot \frac{n}{k} + \frac{n \cdot k}{2} + 4.
\end{aligned}
\tag{2.3}
$$

Худший случай: большое количество пустых блоков, массив отсортирован в обратном порядке (худший случай сортировки перемешиванием, которая используется в блочной сортировке), расчет трудоемкости приведен в следующей формуле

$$
\begin{aligned}
f_{worst} = 1 + 1 + \frac{n}{k} \cdot (1 + 2 + f_{shaker} + 2 + 1 + 4 + \\
+ k \cdot (3 + 1 + 4)) + 1 + 1 + \\
+ \frac{n}{k} \cdot (1 + 4 + 1 + 1 + 5 + 1 + 4 + 1 + 1 + k \cdot (6)) = \\
= 4 + \frac{29 \cdot n + n \cdot f_{shaker} + 6 \cdot n^2}{k} + 8 \cdot n = \\
= 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + n \cdot (19.5 + \frac{k}{2}) + \frac{6 \cdot n^2}{k} = \\
= 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + 19.5 \cdot n + \frac{n \cdot k}{2} + \frac{6 \cdot n^2}{k} = \\
= \frac{6 \cdot n^2}{k} + 27.5 \cdot n + 29 \cdot \frac{n}{k} + \frac{n \cdot k}{2} + 4.
\end{aligned}
\tag{2.4}
$$

2.4.2 Трудоемкость алгоритма быстрой сортировки

Чтобы вычислить трудоемкость алгоритма быстрой сортировки, нужно учесть следующее:

— трудоемкость условного оператора на проверку *pivot* в теле цикла вычисляется по следующей формуле

$$f_{if\_pivot} = 1 + \begin{cases} 2 + 1, \\ 0 \end{cases} \tag{2.5}$$

— трудоемкость цикла, в теле которого условный оператор на проверку $pivot$, вычисляется по следующей формуле

$$f_{for\_low\_high} = 1 + M \cdot f_{if\_pivot} \tag{2.6}$$

— трудоемкость условного оператора на проверку low в теле цикла вычисляется по следующей формуле

$$f_{if\_low} = 1 + \begin{cases} 1, \\ 0 \end{cases} \tag{2.7}$$

— трудоемкость цикла, в теле оператор ветвления f_{if_low} и $f_{for\_low\_high}$, вычисляется по следующей формуле

$$f_{while\_stack} = 1 + N \cdot (2 + f_{if\_low} + 2 + f_{for\_low\_high} + 2 + 2) \tag{2.8}$$

— трудоемкость условного оператора на проверку длины массива вычисляется по формуле

$$f_{if\_len} = 1 + \begin{cases} 1, \\ 1 + f_{while\_stack} \end{cases} \tag{2.9}$$

В итоге, трудоемкость быстрой сортировки вычисляется по формуле

$$\begin{gathered} f_{quick\_sort} = 1 + f_{if\_len} = 1 + 1 + N \cdot (2 + f_{if\_low} + 2 + f_{for\_low\_high}) = \\ 1 + 1 + N \cdot (2 + 2 + 2 + 1 + M \cdot f_{if\_pivot} + 2 + 2) = \\ 1 + 1 + N \cdot (2 + 2 + 2 + 1 + M \cdot (3 + 2 + 2) + 1 + 3 + 2 + 1 = \\ 2 + 7N + 7MN + 7N = 14N + 7MN \end{gathered} \tag{2.10}$$

2.4.3 Трудоемкость алгоритма сортировки выбором

Трудоемкость сортировки выбором в худшем случае $O(N^2)$.
Трудоемкость сортировки выбором в лучшем случае $O(N^2)$.
Худший и лучший случаи совпадают [**selectionsort**].

Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов. Была введена модель оценки трудоемкости алгоритма, были рассчитаны трудоемкости алгоритмов в соответствии с этой моделью.

В результате теоретической оценки трудоемкостей алгоритмов выяснилось, что лучшей асимптотической оценкой во всех случаях обладает быстрая сортировка. Худшей асимптотической оценкой во всех случаях обладает сортировка выбором $O(N^2)$.

3 Технологический раздел

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

Для реализации данной работы был выбран язык *Python* [**python**]. Данный выбор обусловлен следующим:

— язык поддерживает все структуры данных, которые выбраны в результате проектирования;

— язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;

— язык позволяет замерять процессорное время с помощью модуля *time*.

Процессорное время было замерено с помощью функции *process_time()* из модуля *time* [**python-time**].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— *main.py* — файл, содержащий функцию *main*;

— *functions.py* — файл, содержащий вспомогательные функции (ввод массива с клавиатуры, рандомно, с файла и т.д.)

— *sorts.py* — файл, содержащий код реализаций всех алгоритмов сортировок;

— *tests.py* — файл, в котором содержатся функции для замера и вывода времени выполнения реализаций алгоритмов.

3.3 Реализация алгоритмов

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для разработанных алгоритмов сортировок. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Массив	**Ожидаемый результат**	**Фактический результат**
[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]	[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]
[ ]	[ ]	[ ]
[1]	[1]	[1]
[4, 1, 2, 3]	[1, 2, 3, 4]	[1, 2, 3, 4]
[2, 1]	[1, 2]	[1, 2]
[31, 57, 24, -10, 59]	[-10, 24, 31, 57, 59]	[-10, 24, 31, 57, 59]

Вывод

Были разработаны и протестированы спроектированные алгоритмы сортировок: блочная, быстрая и выбором.

4 Исследовательский раздел

блаблабла

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута, а именно были исследованы алгоритмы сортировок.

Для достижения поставленной цели были выполнены следующие задачи.

— описаны алгоритмы блочной, быстрой сортировок и сортировки выбором;

— разработано программное обеспечение, реализующее алгоритмы сортировок;

— выбраны инструменты для реализации алгоритмов и замера процессорного времени их выполнения;

— проведен анализ затрат реализаций алгоритмов по времени и памяти.

В результате исследования реализаций различных алгоритмов было получено, что для массивов длиной 1000, отсортированных в обратном порядке, реализация алгоритма быстрой сортировки по времени оказалась в 2.59 раз лучше, чем реализация блочной сортировки, и в 15.49 раз лучше реализации сортировки выбором. В свою очередь, реализация блочной сортировки оказалась лучше в 5.99 раз по времени выполнения, чем реализация сортировки выбором.

Для отсортированных массивов длиной 1000 реализация быстрой сортировки оказалась лучше по времени в 2.39 раз, чем реализация блочной сортировки, и в 15.6 раз лучше, чем реализация сортировки выбором. В свою очередь, реализация блочной сортировки оказалась лучше в 6.52 раза по времени выполнения, чем реализация сортировки выбором.

Для случайно упорядоченных массивов длиной 1000 реализация быстрой сортировки оказалась лучше по времени в 2.49 раз, чем реализация блочной сортировки, и в 15.01 раза лучше, чем реализация сортировки выбором. В свою очередь, реализация блочной сортировки на случайно упорядоченных массивах оказалась лучше в 6.03 раза по времени выполнения, чем реализация сортировки выбором.

Для массивов длиной менее 300, реализация блочной сортировки была лучше или такой же по времени выполнения по сравнению с быстрой сортировкой, но на массивах длиной более 300 быстрая сортировка становилась лучше по времени выполнению.

В результате теоретической оценки алгоритмов по памяти можно сделать вывод о том, что алгоритм сортировки выбором является наименее ресурсозатратным. Алгоритм быстрой сортировки, напротив, требует больше всего памяти, что объясняется тем, что алгоритм рекурсивный, и на каждый вызов функции требуется выделение памяти на стеке для сохранения информации, связанной с этим вызовом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Каримов К. Х.*, *Василий Е. А.* Теоретические основы кластеризации данных // Актуальные вопросы фундаментальных и прикладных научных исследований : Сборник научных статей по материалам II Международной научно-практической конференции, Уфа, 19 мая 2023 года. Том Часть 2. — Уфа : Общество с ограниченной ответственностью “Научно-издательский центр «Вестник науки»”, 2023. — С. 242—247. — EDN XNJGQS.