



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
(ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 Программная инженерия

Отчет по лабораторной работе №4 по дисциплине анализ алгоритмов

Тема: Многопоточные вычисления

Студент: Серова М.Н.

Группа: ИУ7-55Б

Оценка (баллы): _____

Преподаватель: Волкова Л.Л.

Москва, 2021

Содержание

| | |
|---|-----------|
| Введение | 2 |
| 1 Аналитическая часть | 3 |
| 1.1 Нахождение определителя матрицы | 4 |
| 1.1.1 Матрица 1×1 | 5 |
| 1.1.2 Матрица 2×2 | 5 |
| 1.1.3 Матрица 3×3 | 5 |
| 1.1.4 Матрица $n \times n$ | 5 |
| 1.2 Вывод | 6 |
| 2 Конструкторская часть | 7 |
| 2.1 Требования к программному обеспечению | 7 |
| 2.2 Схемы алгоритмов | 7 |
| 2.3 Вывод | 11 |
| 3 Технологическая часть | 13 |
| 3.1 Средства реализации | 13 |
| 3.2 Реализация алгоритмов | 13 |
| 3.3 Тестирование | 15 |
| 3.4 Выводы | 16 |
| 4 Экспериментальная часть | 17 |
| 4.1 Пример работы программы | 17 |
| 4.2 Технические характеристики | 18 |
| 4.3 Время выполнения алгоритмов | 18 |
| 4.4 Выводы | 20 |
| Заключение | 21 |
| Список литературы | 22 |

Введение

Цель: сравнить производительность последовательной и многопоточной версий алгоритма вычисления определителя матрицы.

Задачи:

1. Изучить алгоритм вычисления определителя квадратной матрицы.
2. Реализовать и протестировать:
 - (а) последовательный алгоритм нахождения определителя матрицы;
 - (б) многопоточный алгоритм нахождения определителя матрицы.
3. Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (процессорному времени работы).
4. Определить, всегда ли при задании количества потоков, равному удвоенному количеству логических ядер процессора, достигается наибольшая эффективность.

1 Аналитическая часть

Многопоточность — способность центрального процессора (CPU) или одного ядра в многоядерном процессоре одновременно выполнять несколько процессов или потоков, соответствующим образом поддерживаемых операционной системой.

Этот подход отличается от многопроцессорности, так как многопоточность процессов и потоков совместно использует ресурсы одного или нескольких ядер: вычислительных блоков, кэш-памяти ЦПУ или буфера перевода с преобразованием (TLB).

В тех случаях, когда многопроцессорные системы включают в себя несколько полных блоков обработки, многопоточность направлена на максимизацию использования ресурсов одного ядра, используя параллелизм на уровне потоков или на уровне инструкций.

Поскольку эти два метода являются взаимодополняющими, их иногда объединяют в системах с несколькими многопоточными ЦП и в ЦП с несколькими многопоточными ядрами. Смысл многопоточности — квазимногозадачность на уровне одного исполняемого процесса.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Достоинства:

- облегчение программы посредством использования общего адресного пространства;
- меньшие затраты на создание потока в сравнении с процессами;
- повышение производительности процесса за счёт распараллеливания процессорных вычислений;
- если поток часто теряет кэш, другие потоки могут продолжать использовать неиспользованные вычислительные ресурсы.

Недостатки:

- несколько потоков могут вмешиваться друг в друга при совместном использовании аппаратных ресурсов [1];
- с программной точки зрения аппаратная поддержка многопоточности более трудоемка для программного обеспечения [2];
- проблема планирования потоков;
- специфика использования. Вручную настроенные программы на ассемблере, использующие расширения MMX или AltiVec и выполняющие предварительные выборки данных, не страдают от потерь кэша или неиспользуемых вычислительных ресурсов. Таким образом, такие программы не выигрывают от аппаратной многопоточности и действительно могут видеть ухудшенную производительность из-за конкуренции за общие ресурсы.

Однако несмотря на количество недостатков, перечисленных выше, многопоточная парадигма имеет большой потенциал на сегодняшний день, и при должном написании кода позволяет значительно ускорить однопоточные алгоритмы.

Определитель матрицы или просто определитель играет важную роль в решении систем линейных уравнений. Определитель матрицы A обозначается как $\det A$, ΔA , $|A|$

Сформулировать определение определителя матрицы можно на основе его свойств. Определителем вещественной матрицы называется функция $\det : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$, обладающая следующими свойствами:

1. $\det(A)$ - кососимметрическая функция строк (столбцов) матрицы A
2. $\det(A)$ - полилинейная функция строк(столбцов) матрицы A
3. $\det(E) = 1$, где E - единичная $n \times n$ -матрица

1.1 Нахождение определителя матрицы

Ниже описаны способы нахождения определителя матрицы размеров 1×1 , 2×2 , 3×3 , $n \times n$.

1.1.1 Матрица 1×1

Для матрицы первого порядка значение детерминанта равно единственному элементу этой матрицы:

$$\delta = |a_{11}| = a_{11} \quad (1.1)$$

1.1.2 Матрица 2×2

Для матрицы 2×2 определитель вычисляется следующим образом:

$$\Delta = \begin{vmatrix} a & c \\ b & d \end{vmatrix} = ad - bc \quad (1.2)$$

Абсолютное значение определителя $|ad - bc|$ равно площади параллелограмма с вершинами $(0, 0)$, (a, b) , $(a + c, b + d)$, (c, d) .

1.1.3 Матрица 3×3

Определитель матрицы 3×3 можно вычислить по формуле:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \cdot \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} = a_{11}a_{22}a_{33} - a_{12}a_{23}a_{31} - a_{13}a_{21}a_{32} + a_{13}a_{22}a_{31} + a_{12}a_{21}a_{33} - a_{11}a_{21}a_{32} \quad (1.3)$$

Определитель матрицы, составленной из векторов a, b, c представляет собой объём параллелепипеда, натянутого на вектора a, b, c .

1.1.4 Матрица $n \times n$

В общем случае, для матриц $n \times n$, где $n > 2$ определитель можно вычислить, применив следующую рекурсивную формулу:

$$\Delta = \sum_{j=1}^n (-1)^{1+j} \cdot a_{1j} \cdot M_j^{-1}, \text{ где } M_j^{-1} - a_{ij} \quad (1.4)$$

В данной лабораторной работе стоит задача распараллеливания алгоритма нахождения определителя матрицы. Так как каждое слагаемое для вычисления итогового определителя вычисляется независимо от других и матрица не изменяется, для параллельного вычисления определителя было решено распределять задачу вычисления слагаемых между потоками.

1.2 Вывод

Был рассмотрен алгоритм нахождения определителя квадратной матрицы размера $n \times n$, он независимо вычисляет слагаемые для нахождения итогового определителя, что дает возможность для реализации параллельного варианта алгоритма.

2 Конструкторская часть

Данный раздел содержит требования к разрабатываемому ПО, схемы реализуемых в работе алгоритмов (стандартного рекурсивного алгоритма вычисления определителя матриц и алгоритма с использованием потоков в виде схем потока-диспетчера и рабочего потока) и теоретический расчет повышения эффективности исполнения алгоритма по времени.

2.1 Требования к программному обеспечению

Требования, выдвигаемые к разрабатываемому ПО:

- входные данные - размер матрицы (целое число), её элементы (вещественные числа);
- выходные данные - определитель матрицы (вещественное число).

2.2 Схемы алгоритмов

В данном пункте раздела представлены схемы реализуемых в работе алгоритмов.

На рисунке 2.1 представлена схема рекурсивного алгоритма нахождения определителя.

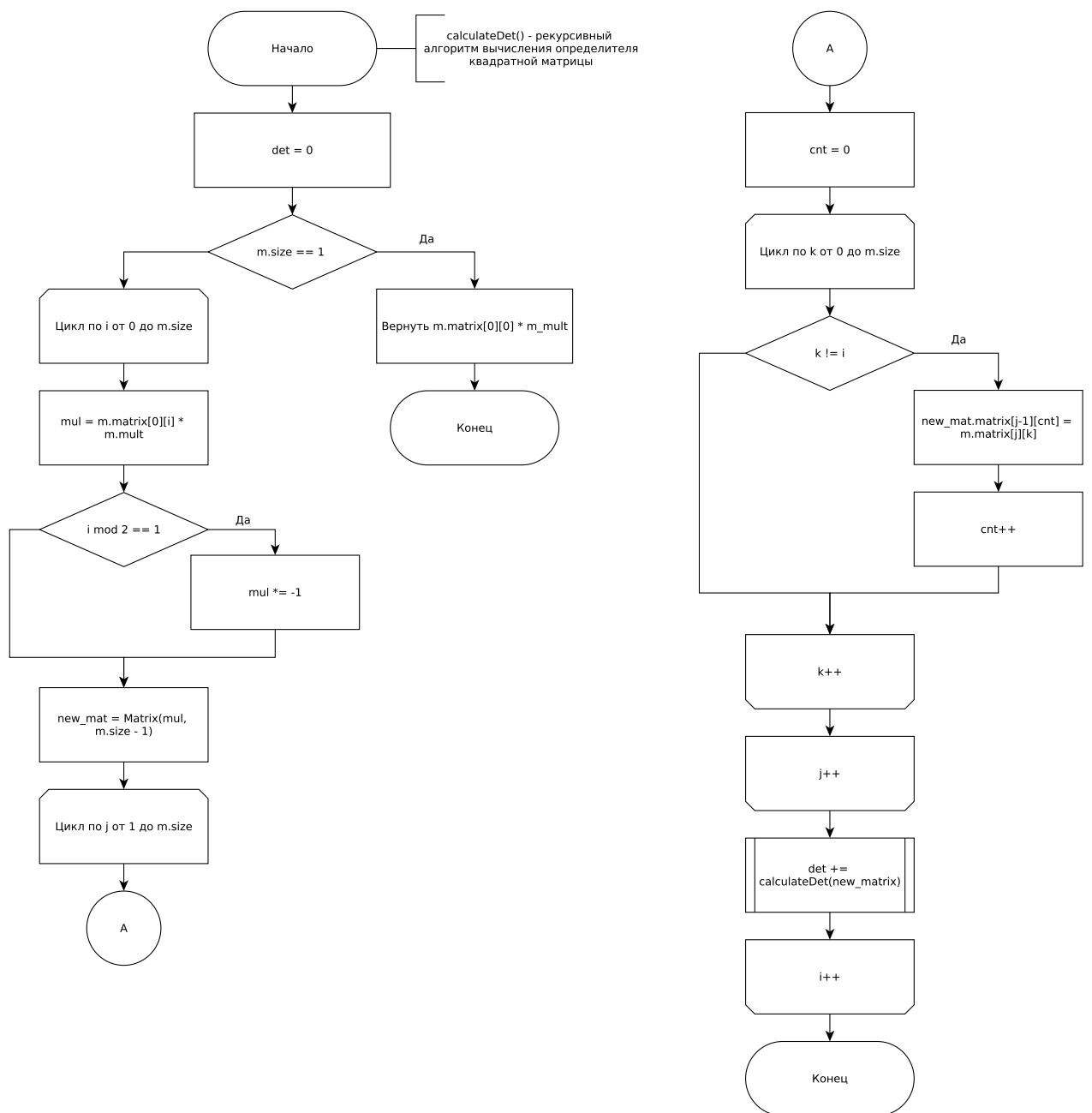


Рис. 2.1: Схема рекурсивного алгоритма нахождения определителя

На рисунке 2.2 представлена схема алгоритма подсчета слагаемых итогового определителя матрицы при использовании потоков.

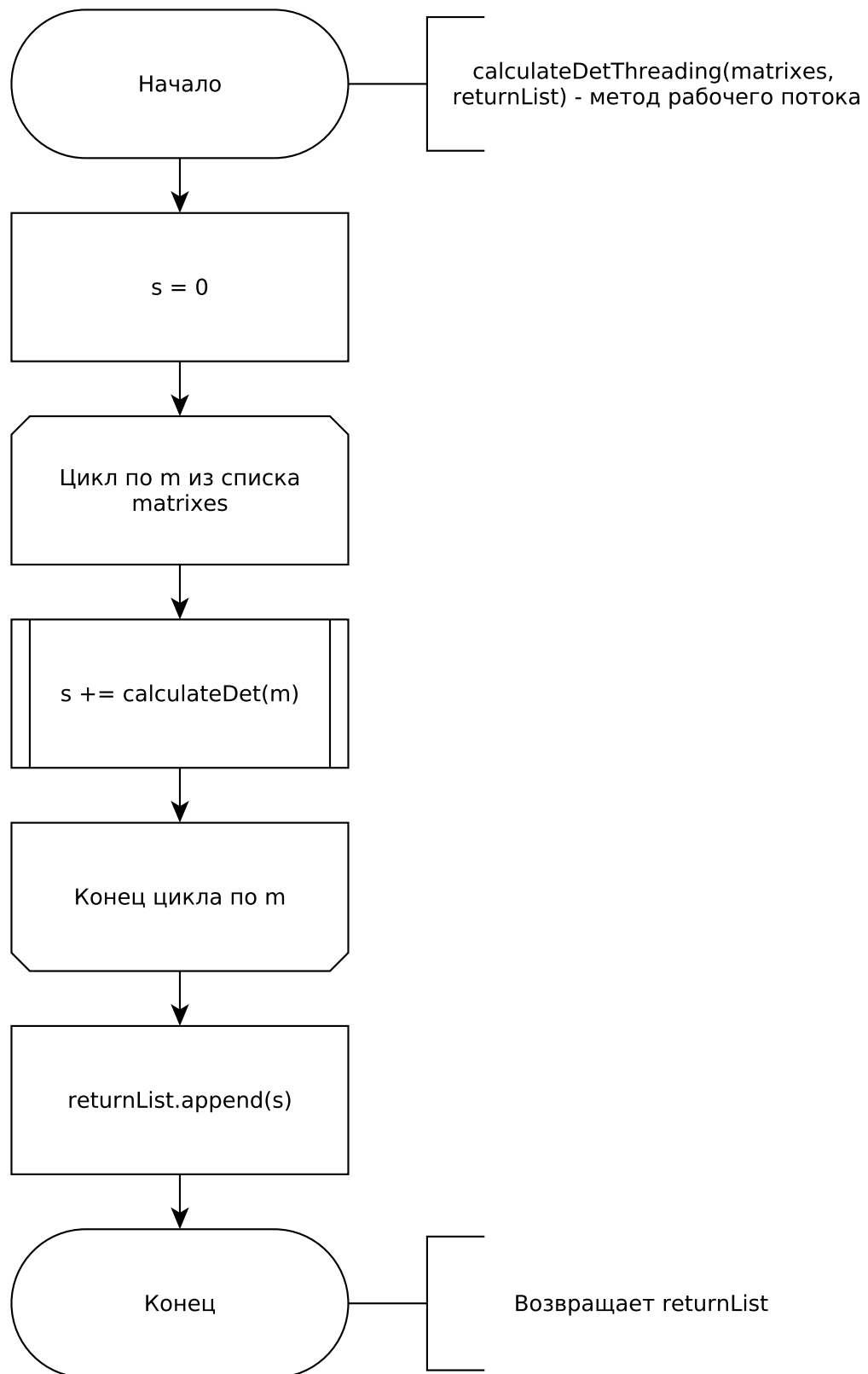


Рис. 2.2: Схема рекурсивного алгоритм нахождения определителя

На рисунках 2.3 - 2.4 представлена схема алгоритма создания потоков, разделения задач между ними и нахождения итогового определителя матрицы.

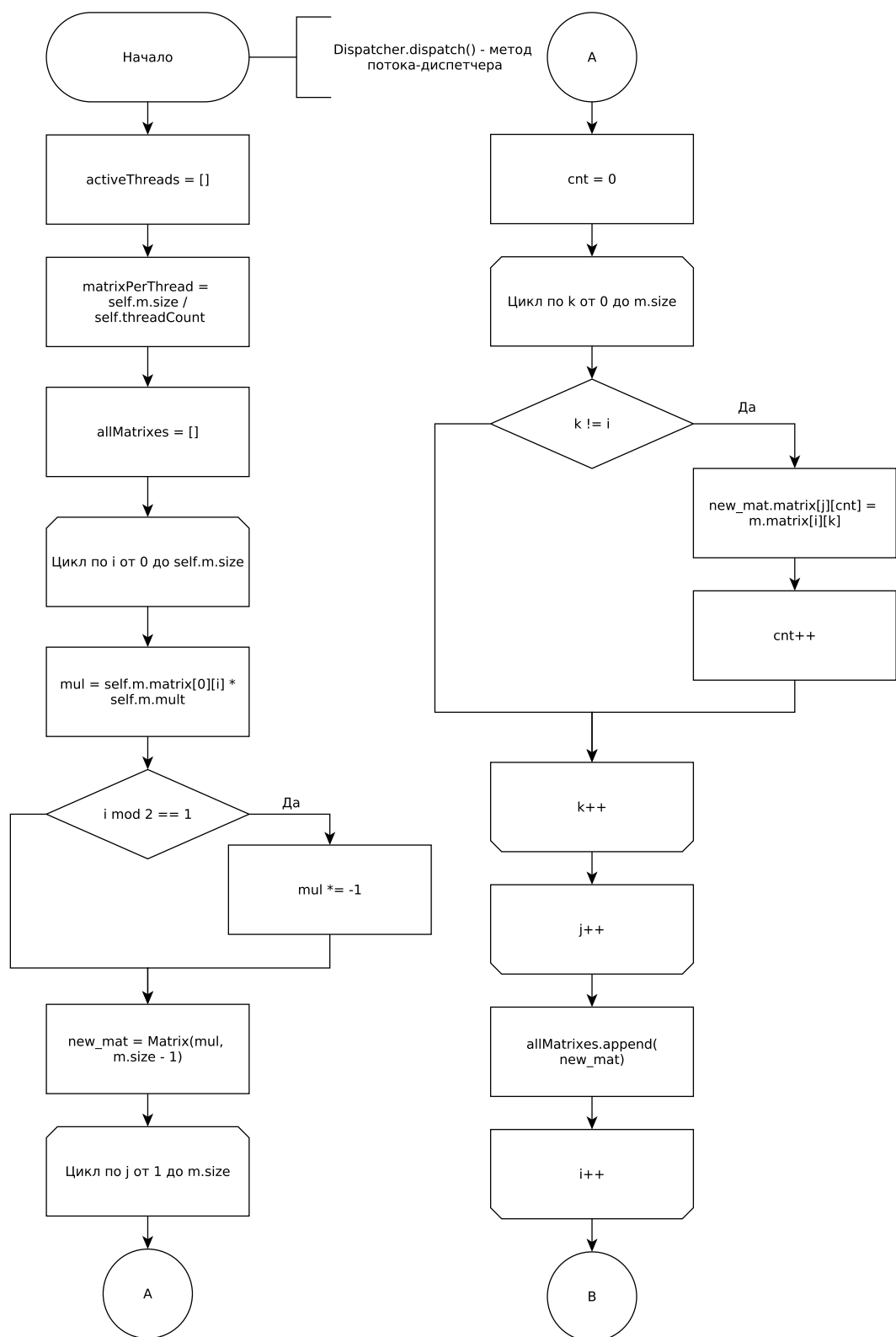


Рис. 2.3: Схема рекурсивного алгоритм нахождения определителя

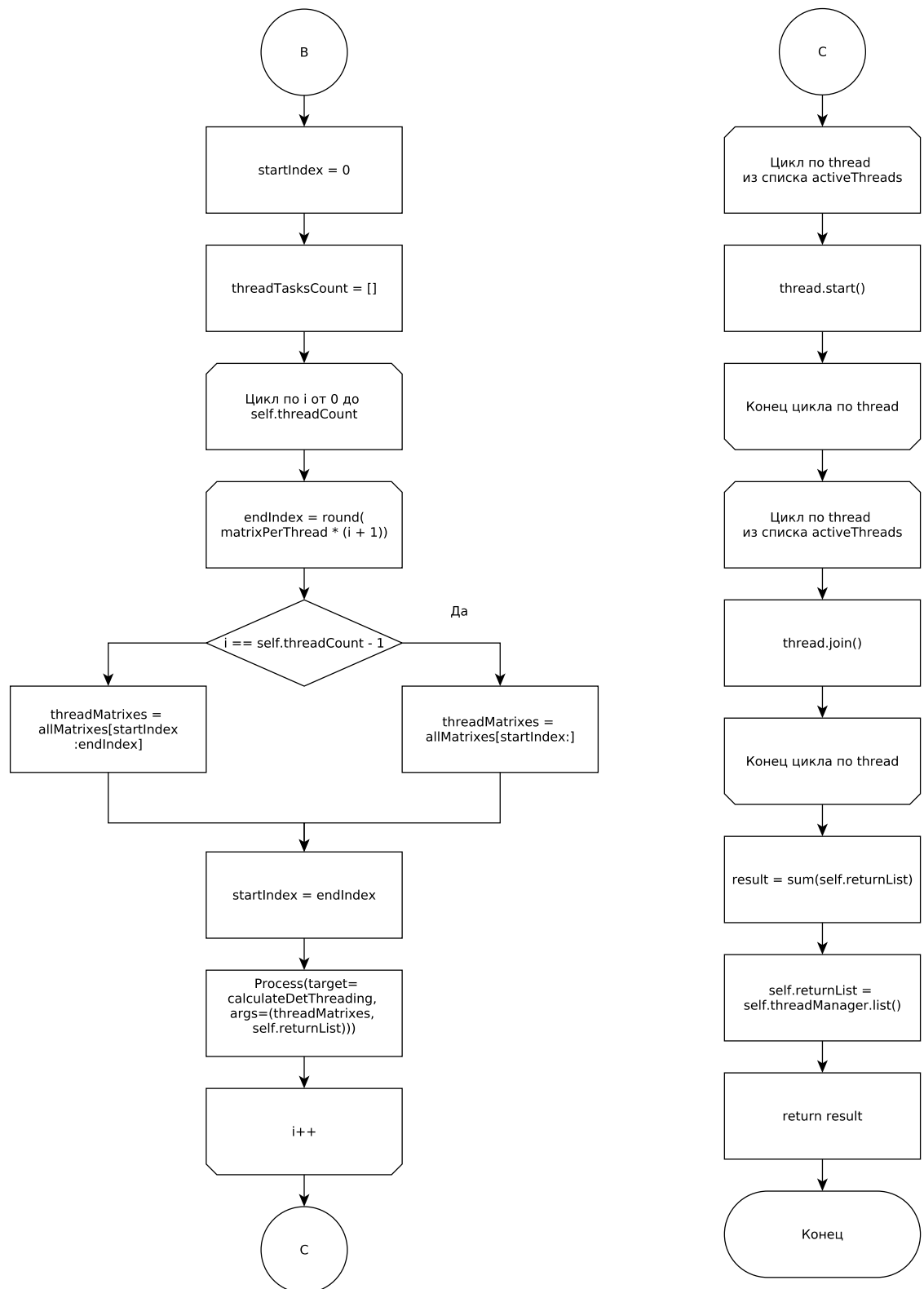


Рис. 2.4: Схема рекурсивного алгоритм нахождения определителя

2.3 Вывод

В данном разделе на основе приведенных в аналитическом разделе теоретических данных были составлены схемы алгоритмов для реализации в

технологической части, в том числе, разделение вычисления определителя на потоки.

3 Технологическая часть

Данный раздел содержит обоснование выбора языка и среды разработки, реализацию алгоритмов.

3.1 Средства реализации

Для реализации программы был выбран язык программирования Python [3]. Такой выбор обусловлен следующими причинами:

- имеется большой опыт разработки;
- имеет большое количество расширений и библиотек, в том числе библиотеку для работы с потоками, измерения времени, построения графиков;
- обладает информативной документацией;

3.2 Реализация алгоритмов

В листингах 3.1 - 3.4 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Рекурсивный алгоритм вычисления определителя матрицы

```
1 def calculateDet(m: Matrix):
2     s = 0
3     if m.size == 1:
4         s = m.matrix[0][0] * m.mult
5     else:
6         for i in range(m.size):
7             mul = m.matrix[0][i] * m.mult
8             if i % 2 == 1:
9                 mul *= -1
10            size = m.size - 1
11            new_mat = Matrix(size, mul)
12            for j in range(1, m.size):
13                cnt = 0
14                for k in range(m.size):
15                    if k != i:
16                        new_mat.matrix[j - 1][cnt] = m.matrix[j][k]
17                        cnt += 1
18                s += calculateDet(new_mat)
19     return s
```

Листинг 3.2: Поток-диспетчер (часть 1)

```
1 class Dispatcher:
2     def __init__(self, size: int = 0, matrix: Matrix = None):
3         self.m = matrix
4         if size == 0:
5             size = 9
6         if self.m is None:
7             self.m = Matrix(size).randomize()
8         self.threadCount = 1
9         self.threadManager = Manager()
10        self.returnList = self.threadManager.list()
11
12    def dispatch(self):
13        activeThreads = []
14        matrixPerThread = self.m.size / self.threadCount
15        allMatrixes = []
16        for i in range(self.m.size):
17            mul = self.m.matrix[0][i] * self.m.mult
18            if i % 2 == 1:
19                mul *= -1
20            size = self.m.size - 1
21            matrix = []
22            for j in range(1, self.m.size):
23                matrix.append([])
24                for k in range(self.m.size):
25                    if k != i:
26                        matrix[-1].append(self.m.matrix[j][k])
27            allMatrixes.append(Matrix(size, mul, matrix))
28        startIndex = 0
29        threadTasksCount = []
30        for i in range(self.threadCount):
31            endIndex = round(matrixPerThread * (i + 1))
32            if i == self.threadCount - 1: # last thread
33                threadMatrixes = allMatrixes[startIndex:]
34            else:
35                threadMatrixes = allMatrixes[startIndex:endIndex]
36            startIndex = endIndex
37            activeThreads.append(
38                Process(target=calculateDetThreading, args=(threadMatrixes,
39                                                            self.returnList)))
39            threadTasksCount.append(len(threadMatrixes))
```

Листинг 3.3: Поток-диспетчер (часть 2)

```
1     for thread in activeThreads:
2         thread.start()
3     for thread in activeThreads:
4         thread.join()
5     result = sum(self.returnList)
6     self.returnList = self.threadManager.list()
7     return result
```

Листинг 3.4: Рабочий поток

```
1 def calculateDetThreading(matrixes, returnList: list):
2     s = 0
3     for m in matrixes:
4         s += calculateDet(m)
5     returnList.append(s)
```

3.3 Тестирование

В таблице 3.1 представлены использованные для тестирования методом "черного ящика" данные, были рассмотрены все возможные тестовые случаи. Все тесты пройдены успешно.

Таблица 3.1: Проведенные тесты

| Матрица | Определитель |
|---|--------------|
| (9) | 9 |
| $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | 0 |
| $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ | 1 |
| $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 12 \end{pmatrix}$ | -9 |
| $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$ | 0 |

3.4 Выводы

В данном разделе были реализованы и протестированы алгоритмы рекурсивного вычисления определителя и вычисления определителя с использованием многопоточности.

4 Экспериментальная часть

В данном разделе сравниваются реализованные алгоритмы, дается сравнительная оценка затрат на время.

4.1 Пример работы программы

Пример работы программы представлен на рисунках 4.1 - 4.2.

```
Введите любой символ для генерации матрицы заданного размера, нажмите Enter для ввода матрицы:

Введите размеры квадратной матрицы (целое число): 9
Введите матрицу:
1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1
Рекурсивно вычисленный определитель 1.0
Без потоков, время выполнения = 0.4965999126434326
Количество потоков: 1, время выполнения = 0.5147125720977783, результат = 1.0
Количество потоков: 2, время выполнения = 0.2974381446838379, результат = 1.0
Количество потоков: 4, время выполнения = 0.18285012245178223, результат = 1.0
Количество потоков: 8, время выполнения = 0.14126157760620117, результат = 1.0
Количество потоков: 16, время выполнения = 0.14821529388427734, результат = 1.0
Количество потоков: 32, время выполнения = 0.1662731170654297, результат = 1.0
```

Рис. 4.1: Пример работы программы для вводимой матрицы

```
Введите любой символ для генерации матрицы заданного размера, нажмите Enter для ввода матрицы:
d
Введите размер матрицы для автоматической генерации: 9
Generated matrix [[290, -462, -36, -561, -312, -123, -696, -148, -973], [-700, -749, -439, 692, -626, 864, 58
6, 992, 903], [-706, 716, -476, -341, 381, -291, -451, 23, -706], [961, -436, 443, -191, -280, -228, 745, -528
, 287], [-702, -553, -617, -32, -4, 537, -867, -606, -709], [146, -242, -468, -738, -298, 962, -817, 917, -874
], [864, -938, -886, -60, 591, 940, 517, -131, 949], [-737, 363, -934, 206, -635, 759, 485, 374, -24], [303, 3
67, 907, 457, -557, -349, -634, -376, 290]]
Рекурсивно вычисленный определитель 954348099616103725872662116
Без потоков, время выполнения = 0.8421685695648193
Количество потоков: 1, время выполнения = 0.8674774169921875, результат = 954348099616103725872662116
Количество потоков: 2, время выполнения = 0.4965970516204834, результат = 954348099616103725872662116
Количество потоков: 4, время выполнения = 0.3104851245880127, результат = 954348099616103725872662116
Количество потоков: 8, время выполнения = 0.21962666511535645, результат = 954348099616103725872662116
Количество потоков: 16, время выполнения = 0.22811508178710938, результат = 954348099616103725872662116
Количество потоков: 32, время выполнения = 0.27892494201660156, результат = 954348099616103725872662116
```

Рис. 4.2: Пример работы программы для вводимой матрицы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- операционная система: Ubuntu 20.01 Linux x86_64 [4];
- оперативная память: 8 Гб;
- процессор: AMD Ryzen5 4500U [5]:
 - количество физических ядер: 6;
 - количество логических ядер: 6.

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов измерялось на автоматически генерируемых квадратных матрицах необходимого размера (элементы которых - вещественные числа в диапазоне $[-1000, 1000]$) с использованием функции `time` библиотеки `time`. Усредненные результаты 10 замеров реального времени работы приведены в таблице ниже.

На рисунке 4.3 представлена зависимость времени вычисления определителя матриц от размеров на основе таблицы 4.1. Для матриц, размерность которых меньше 6, вычисление с использованием потоков занимает минимум в 1.3 раз (с увеличением коэффициента при увеличении количества потоков) больше времени в связи с превышением затратами на создание потоков затрат на последовательное вычисление определителя. Однако с увеличением размеров матрицы становится заметным преимущество использования параллельных вычислений. Таким образом, использование двух потоков дает выигрыш от 1.2 до 1.7 раз (для матриц, больших 6×6), четырех - от 1.2 до 2.8 раз (для матриц, больших 6×6), восьми - от 1.6 до 3.5 раз (матрицы от 7×7), шестнадцати - от 2.7 до 4.4 раз, для двадцати четырех - от 2.3 до 4.28 раз (матрицы от 8×8). Таким образом, с увеличением размерности матрицы, выигрыш от использования потоков увеличивается, при этом наибольшим приростом, несмотря на получение преимуществ для матриц больших, чем те, на которых начинает выигрывать использование двух потоков (6×6), обладает использование шестнадцати потоков, причем с ростом размерности матрицы

выигрыш увеличивается соразмерно количеству потоков. Исключением является использование 24 потоков, который уступает в эффективности 16-и потокам из-за увеличения затрат на содержание потоков и управления ими.

Таблица 4.1: Время вычисления определителя матриц разных размеров в миллисекундах

| Размер | 1 поток | 2 потока | 4 потока | 8 потоков | 16 потоков | 24 потока |
|--------|---------|----------|----------|-----------|------------|-----------|
| 4 | 12 | 14 | 16 | 22 | 36 | 51 |
| 5 | 14 | 15 | 16 | 23 | 37 | 51 |
| 6 | 22 | 18 | 19 | 23 | 37 | 50 |
| 7 | 50 | 33 | 29 | 31 | 46 | 58 |
| 8 | 251 | 138 | 80 | 83 | 94 | 108 |
| 9 | 2102 | 1212 | 756 | 599 | 476 | 491 |

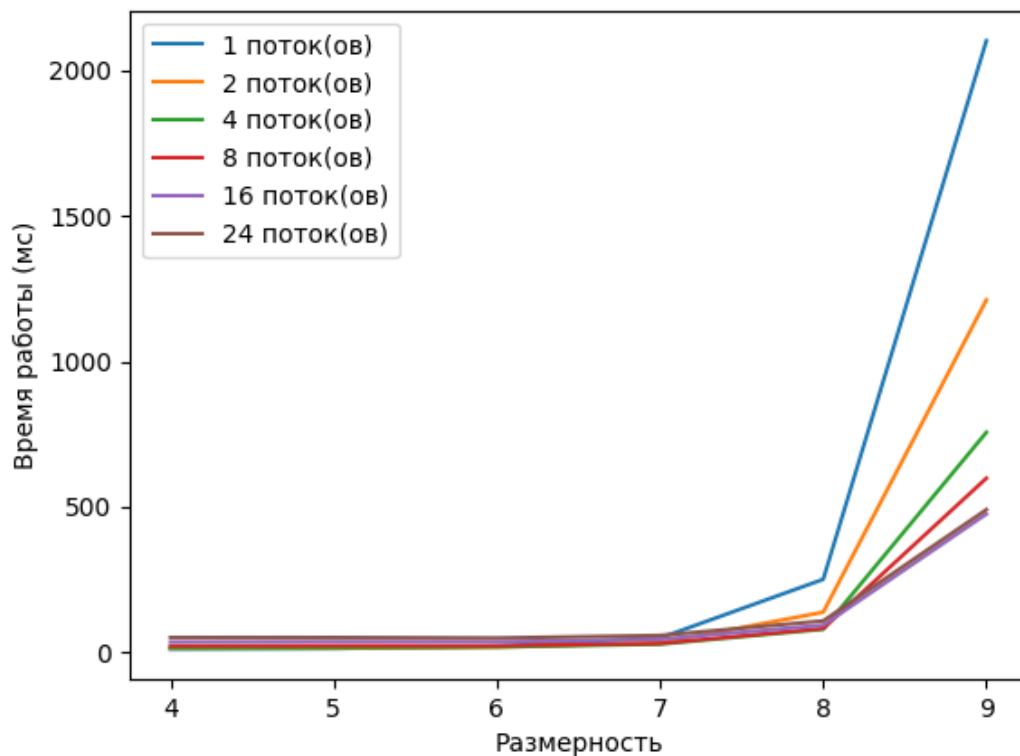


Рис. 4.3: Зависимость времени работы алгоритмов от размера квадратной матрицы

4.4 Выводы

В данном разделе были проведены измерения времени, затрачиваемого на вычисление определителя матрицы с использованием параллельных вычислений и без них. На матрицах, размер которых не превышает 5, параллельные версии алгоритма оказались неэффективны. С увеличением размеров увеличивается эффективность использования параллельных вычислений, причем чем больше количество процессов, тем позже появляется выигрыш, но тем более существенным он оказывается. Исключением является использование 24 потоков в связи с большими затратами на обслуживание потоков в ядрах по сравнению с меньшим количеством потоков. Таким образом, выигрыш для 4 и 8 потоков проявляется на матрицах, больших 6×6 и составляет от 1.2 до 1.7 раз и от 1.2 до 2.8 раз соответственно, для восьми потоков - на матрицах, больших 7×7 , от 1.5 до 3.5 раз, для 16 и 24 потоков - на матрицах, больших 8×8 , от 2.7 до 4.4 раз и от 2.3 до 4.28 раз соответственно.

Заключение

В процессе выполнения лабораторной работы были изучены и реализованы алгоритмы последовательного и параллельного вычисления определителя квадратной матрицы.

Было исследовано время выполнения выше обозначенных алгоритмов. В результате было выявлено, что на матрицах, размер которых не превышает 5, использование параллельных вычислений нецелесообразно из-за превышения затратами на содержание потоков затрат на последовательное вычисление определителя. С увеличением размеров увеличивается эффективность использования параллельных вычислений, причем чем больше количество процессов, тем позже появляется выигрыш, но тем более существенным он оказывается. Исключением является использование 24 потоков в связи с увеличением затрат на обслуживание потоков в ядрах по сравнению с 16 потоками. Таким образом, выигрыш для 4 и 8 потоков проявляется на матрицах, больших 6×6 и составляет от 1.2 до 1.7 раз и от 1.2 до 2.8 раз соответственно, для восьми потоков - на матрицах, больших 7×7 , от 1.5 до 3.5 раз, для 16 и 24 потоков - на матрицах, больших 8×8 , от 2.7 до 4.4 раз и от 2.3 до 4.28 раз соответственно. Наиболее эффективным для матриц, больших 8×8 , является использование 16 потоков, для матриц 7×7 - 8 потоков, 6×6 - четырех.

Список литературы

- [1] Mario Nemirovsky D. M. T. Multithreading Architecture // Morgan and Claypool Publishers. 2013.
- [2] Olukotun K. Chip Multiprocessor Architecture — Techniques to Improve Throughput and Latency // Morgan and Claypool Publishers. 2007. p. 154.
- [3] About Python [Электронный ресурс]. Режим доступа: <https://www.python.org/about/>. Дата обращения: 20.10.2021.
- [4] Ubuntu по-русски [Электронный ресурс]. Режим доступа: <https://ubuntu.ru/>. Дата обращения: 20.10.2021.
- [5] AMD Ryzen™ 5 4500U. Режим доступа: <https://www.amd.com/ru/products/apu/amd-ryzen-5-4500u>. Дата обращения: 20.10.2021.