|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 №** | 04 |

**Название:**

***Параллельная реализация алгоритма yмножения матриц***

**Дисциплина:  *Анализ алгоритмов***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-56Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Волокова Л. Л.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2020*

**Оглавление**

[Введение 3](#_Toc57225228)

[I. Аналитическая часть 4](#_Toc57225229)

[1. Описание алгоритмов 4](#_Toc57225230)

[*i.* *Классический алгоритм умножения* 4](#_Toc57225231)

[*ii.* *Алгоритм Винограда* 4](#_Toc57225232)

[*iii.* *Оптимизированный алгоритм Винограда* 5](#_Toc57225233)

[2. Модель вычислений 6](#_Toc57225234)

[II. Конструкторская часть 7](#_Toc57225235)

[1. Схемы алгоритмов 7](#_Toc57225236)

[2. Оценка трудоёмкости 12](#_Toc57225237)

[*i.* *Классический алгоритм:* 12](#_Toc57225238)

[*ii.* *Алгоритм Винограда:* 12](#_Toc57225239)

[*iii.* *Оптимизированный алгоритм Винограда* 13](#_Toc57225240)

[III. Технологическая часть 14](#_Toc57225241)

[1. Требования к программному обеспечению: 14](#_Toc57225242)

[2. Средства реализации 14](#_Toc57225243)

[3. Реализации алгоритмов 14](#_Toc57225244)

[*i.* *Классический алгоритм* 14](#_Toc57225245)

[*ii.* *Алгоритм Винограда* 14](#_Toc57225246)

[*iii.* *Оптимизированный алгоритм Винограда* 15](#_Toc57225247)

[4. Тесты 16](#_Toc57225248)

[IV. Экспериментальная часть 17](#_Toc57225249)

[1. Примеры работы 17](#_Toc57225250)

[2. Сравнение работы алгоритмов при чётных размерах матрицы 17](#_Toc57225251)

[3. Сравнение работы алгоритмов при нечётных размерах матрицы 18](#_Toc57225252)

[Заключение 20](#_Toc57225253)

# Введение

**Цель лабораторной работы:** изучение возможности параллельных вычислений и использование такого подхода на практике. Реализация парралельного алгоритма Винограда умножения матриц.

Для того чтобы добиться этой цели, были поставлены следующие задачи:

* изучить понятие параллельный вычислений;
* реализовать последовательный и параллельный алгоритм Винограда;
* сравнить временные характеристики реализованных алгоритмов экспериментально;

1. **Аналитическая часть**
   1. **Описание алгоритмов**
      1. *Алгоритм Винограда*

Рассматривая результат умножения двух матриц очевидно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора:

*где U = Ai - i-ая строка матрицы A,*

*V = Bj - j-ый столбец матрицы B*

Их скалярное произведение равно:

Это равенство можно переписать в виде:

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного.

* + 1. Параллельный алгоритм Винограда:
* Прежде всего предварительные вычисления MulH и MulV не зависят друг от друга, значит, их можно вычислить параллельно;
* Распараллелить ту часть алгоритма, которая содержит 3 вложенных цикла: Вычисление результата для каждой строки не зависит от результата выполнения умножения для других строк. Поэтому можно распараллелить часть кода, где происходят эти действия. Каждый поток будет выполнять вычисления определенных строк результирующей матрицы.
  1. **Модель вычислений**

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

* +, -, /, \%, ==, !=, <, >, <=, >=, [], ++, \-- -- имеют трудоемкость 1
* трудоемкость оператора выбора

if (условие) then

A

Else

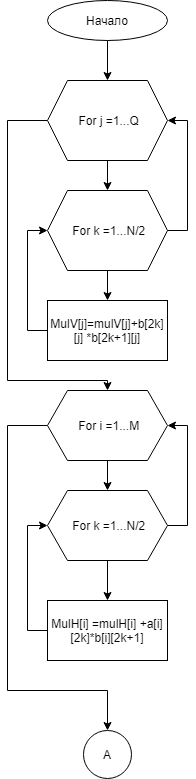
B

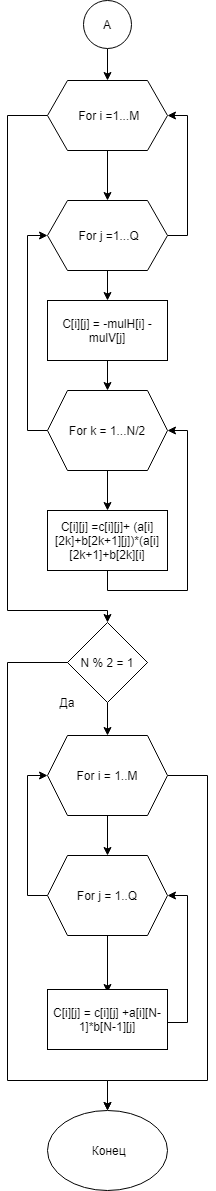
рассчитывается, как

* трудоемкость цикла рассчитывается, как
* трудоемкость вызова метода равна 0
* трудоемкость вызова функции равна 1

1. **Конструкторская часть**
   1. **Схемы алгоритмов**

**Алгоритм Винограда**

****

****

**Выводы:**

В результате работы над конструкторским разделом была разработана схема алгоритма Винограда и выбран способ распараллеливания алгоритма.

1. **Технологическая часть**
   1. **Требования к программному обеспечению:**

На вход подаются размеры двух матриц. Матрицы генерируются случайным образом и выводятся на экран. На выход программа выдаёт 2 матрицы, которые являются результатами работы алгоритмов умножения.

* 1. **Средства реализации**

Для реализации программы был использован язык Python.

Для замера процессорного времени была использована функция time() из библиотеки time.

Многопоточное программирование было реализовано с помощью библиотеки Threading.

* 1. **Реализации алгоритмов**
     1. *Алгоритм Винограда*

1. **def** matrix\_mult\_winograd(A, B):
2. **if** A.col != B.row:
3. **return** Mat()
5. M = A.row
6. N = A.col
7. Q = B.col
8. # I
9. mulV = [0 **for** i **in** range(Q)]
10. **for** j **in** range(Q):
11. **for** k **in** range(N // 2):
12. mulV[j] = mulV[j] + B.mat[2 \* k][j] \* B.mat[2 \* k + 1][j]
14. #II
15. mulH = [0 **for** i **in** range(M)]
16. **for** i **in** range(M):
17. **for** k **in** range(N // 2):
18. mulH[i] = mulH[i] + A.mat[i][2 \* k] \* A.mat[i][2 \* k + 1]
20. #III
21. C = matrix\_create(M, Q)
22. **for** i **in** range(M):
23. **for** j **in** range(Q):
24. C.mat[i][j] = -mulH[i] - mulV[j]
25. **for** k **in** range(N // 2):
26. C.mat[i][j] = C.mat[i][j] + (A.mat[i][2 \* k] + B.mat[2 \* k + 1][j]) \* (A.mat[i][2 \* k + 1] + B.mat[2 \* k][j])
28. #IV
29. **if** (N % 2 == 1):
30. **for** i **in** range(M):
31. **for** j **in** range(Q):
32. C.mat[i][j] = C.mat[i][j] + A.mat[i][N - 1] \* B.mat[N - 1][j]
34. **return** C
    * 1. *Fdfdf*
35. **def** parallel(func, args, M, num\_thread):
36. count = M // num\_thread
37. threads = []
38. tmp = list(args)
39. start, end = 0, 0
40. **for** i **in** range(num\_thread):
41. **if** i == num\_thread - 1:
42. end = M
43. **else**:
44. end = start + count
45. tmp.append(start)
46. tmp.append(end)
48. f\_args = tuple(tmp)
49. t = Thread(target = func, args = f\_args)
50. threads.append(t)
51. t.start()
53. tmp.pop()
54. tmp.pop()
55. start += count
57. **for** t **in** threads:
58. t.join()

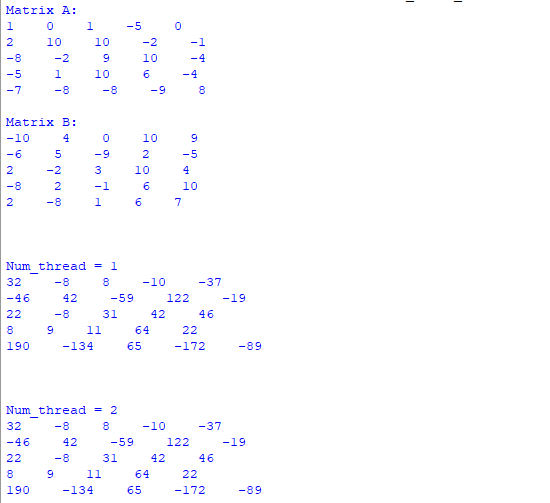
61. **def** calc\_MulV(B, mulV, start = 0, end = None):
62. **if** end == None:
63. end = start + 1
64. Nmin1 = B.row - 1
65. **for** j **in** range(start, end):
66. **for** k **in** range(0, Nmin1, 2):
67. mulV[j] += B.mat[k][j] \* B.mat[k + 1][j]
69. **def** calc\_MulH(A, mulH, start = 0, end = None):
70. **if** end == None:
71. end = start + 1
72. Nmin1 = A.col - 1
73. **for** i **in** range(start, end):
74. **for** k **in** range(0, Nmin1, 2):
75. mulH[i] += A.mat[i][k] \* A.mat[i][k + 1]
77. **def** part3\_parallel(A, B, C, mulH, mulV, start, end):
78. Nmin1 = A.col - 1
79. Q = B.col
80. **for** i **in** range(start, end):
81. **for** j **in** range(Q):
82. C.mat[i][j] = -mulH[i] - mulV[j]
83. **for** k **in** range(0, Nmin1, 2):
84. C.mat[i][j] += (A.mat[i][k] + B.mat[k + 1][j]) \* (A.mat[i][k + 1] + B.mat[k][j])
86. **def** part4\_parallel(A, B, C, mulH, mulV, start, end):
87. Nmin1 = A.col - 1
88. Q = B.col
89. Nmin1 = A.col - 1
90. **for** i **in** range(start, end):
91. **for** j **in** range(Q):
92. C.mat[i][j] = C.mat[i][j] + A.mat[i][Nmin1] \* B.mat[Nmin1][j]
94. **def** mult\_parallel(A, B, num\_threads = None):
96. **if** A.col != B.row:
97. **return** Mat()
98. M = A.row
99. N = A.col
100. Q = B.col
101. Nmin1 = N - 1
102. #Get Thread\_Count
103. **if** num\_threads == None:
104. **try**:
105. num\_threads = multiprocessing.cpu\_count()
106. **except** AttributeError:
107. num\_threads = 4
109. mulV = [0 **for** i **in** range(Q)]
110. mulH = [0 **for** i **in** range(M)]
112. **if** (num\_threads >= 2):
113. #I
114. t1 = Thread(target = parallel, args = (calc\_MulV, (B, mulV), Q, num\_threads // 2))
115. t1.start()
116. #II
117. t2 = Thread(target = parallel, args = (calc\_MulH, (A, mulH, ), M, num\_threads - num\_threads // 2))
118. t2.start()
119. t1.join()
120. t2.join()
121. **else**:
122. # I
123. **for** j **in** range(Q):
124. **for** k **in** range(0, Nmin1, 2):
125. mulV[j] += B.mat[k][j] \* B.mat[k + 1][j]
127. #II
128. **for** i **in** range(M):
129. **for** k **in** range(0, Nmin1, 2):
130. mulH[i] += A.mat[i][k] \* A.mat[i][k + 1]
132. #III
133. C = matrix\_create(M, Q)
134. parallel(part3\_parallel, (A, B, C, mulH, mulV), M, num\_threads)
136. #IV
137. **if** (N % 2 == 1):
138. parallel(part4\_parallel, (A, B, C, mulH, mulV), M, num\_threads)
139. **return** C
     1. **Тесты**

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице:

|  |  |  |
| --- | --- | --- |
| A | B | C = AxB |
| [3] | [-5] | [-15] |
|  |  |  |
|  |  |  |
|  |  |  |

В результате проверки реализации всех алгоритмов умножения прошли все поставленные функциональные тесты.

1. **Экспериментальная часть**
   1. **Примеры работы**



* 1. **Сравнение работы алгоритмов**

# Заключение

* Реализован классический алгоритм умножения матриц и алгоритм Винограда
* Оптимизированна работа алгоритма Винограда
* Выполнил сравнительный анализ трудоёмкостей алгоритмов
* Сравнил эффективность алгоритмов по времени

В ходе лабораторной работе были изучены и реализованы три алгоритма умножения матриц: классический алгоритм, алгоритм Винограда и его оптимизированный вариант. Сравнительный анализ алгоритмов показал, что алгоритмы Винограда, введя дополнительные векторы, добились уменьшения времени выполнения умножения за счёт уменьшения трудоёмких операций: неоптимизированный и оптимизированный варианты работают быстрее классического алгоритма.