**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

**КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ, НАХОДЯЩЕЕ ТРОЙКИ КОМПЛАНАРНЫХ ВЕКТОРОВ СРЕДИ ЗАДАННЫХ С ИСПОЛЬЗОВАНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ С ПРИМЕНЕНИЕМ OPENMP**

Пояснительная записка

**Исполнитель**:

Студентка группы БПИ195

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Зубарева Н.Д./

«17» ноября 2020 г.

**Оглавление**

[1. Текст задания 2](#_Toc56529473)

[2. Применяемые расчетные методы 3](#_Toc56529474)

[2.1. Теория решения задания 3](#_Toc56529475)

[2.2. Организация многопоточности 3](#_Toc56529476)

[2.3. Ввод входных данных 3](#_Toc56529477)

[2.4. Вывод данных 3](#_Toc56529478)

[3. Тестирование программы 4](#_Toc56529479)

[3.1. Корректные значения 4](#_Toc56529482)

[3.2. Некорректные значения 5](#_Toc56529483)

[4. Список литературы 7](#_Toc56529484)

[5. Приложение кода 8](#_Toc56529485)

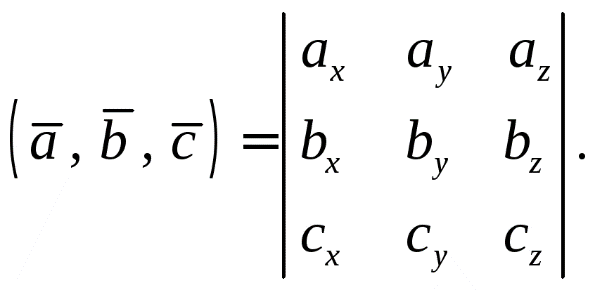
# Текст задания

Вариант 10: Найти все возможные тройки компланарных векторов. Входные данные: множество не равных между собой векторов (x, y, z), где x, y, z – числа. Оптимальное количество потоков выбрать самостоятельно. Использовать OpenMP.

# Применяемые расчетные методы

## Теория решения задания

По условию требуется находить компланарные тройки векторов среди данных. Согласно [2], для этого можно использовать значение смешанного произведения векторов, а именно, оно должно быть равно нулю. Также использована формула вычисления смешанного произведения по координатам трех данных векторов [3]:



## Организация многопоточности

Программа реализует модель итеративного параллелизма [4]: для каждого вектора создается поток, который далее перебирает все вторые и третьи элементы тройки из векторов с индексами после предыдущего в тройке и проверяет для полученной тройки равенство смешанного произведения нулю. Потоки, таким образом, фиксированы по индексам векторов и не вступают в конфликт, благодаря чему можно избежать использование блокировок и семафоров, вывод каждой тройки векторов в консоль также потокобезопасен. Программа написана на языке Си и использует OpenMP для создания потоков с помощью директивы #pragma omp parallel for, которая выделяет оптимальное количество потоков для реализации цикла, в котором вызывается функция, выполняемая одним потоком.

## Ввод входных данных

Ввод данных осуществлен через командную строку и чтение из файла. В командной строке задается путь к файлу, из которого нужно считать данные. Далее в файле должно быть указано число векторов. Было принято решение ограничить количество 3 векторами снизу (1 и 2 вектора всегда компланарны) и 50 векторами сверху (при большем количестве работа программы в среднем занимает больше 10 секунд). При нехватке векторов программа завершается, при избытке – считывание не осуществляется после 50 векторов. Далее в файле должны быть записаны векторы в указанном выше количестве. Считывание всех чисел осуществляется с помощью функции fscanf в формате %ld для числа векторов и %lf для элементов векторов.

## Вывод данных

Тройки компланарных векторов выводятся в консоль с помощью функции printf с форматированием %g.

# Тестирование программы

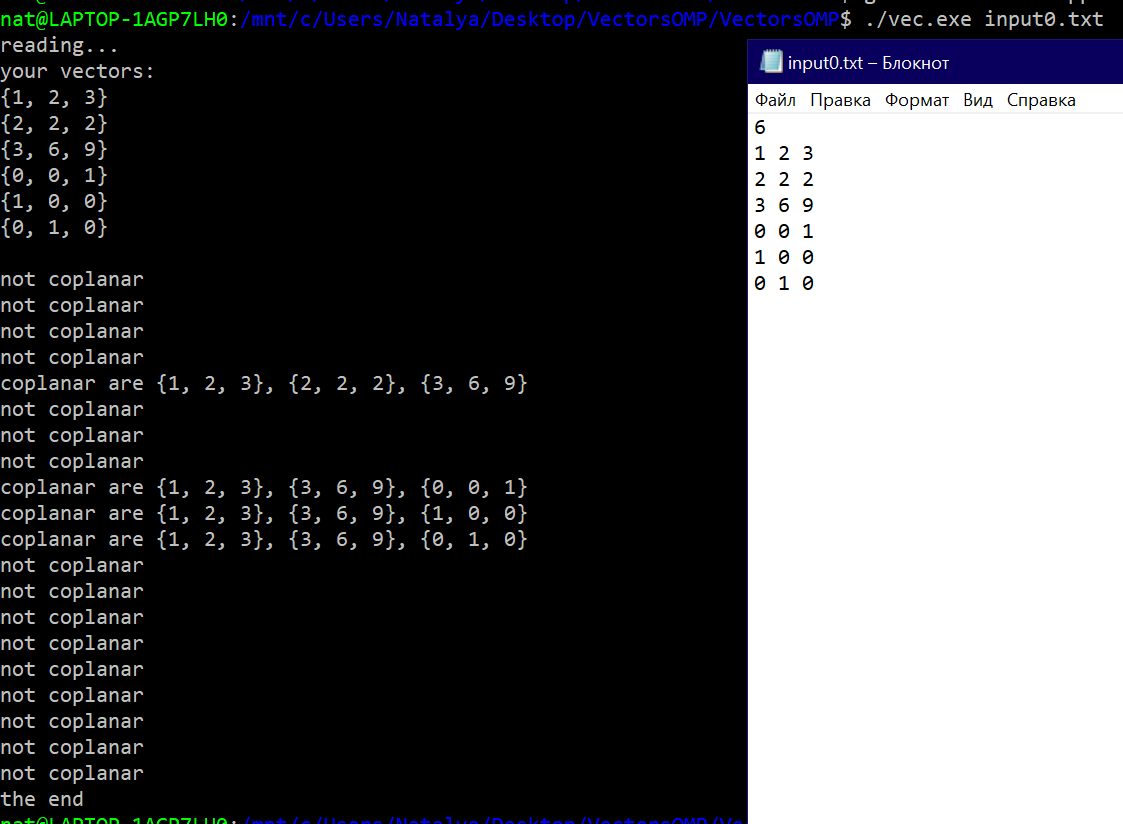
Программа компилируется и запускается следующим образом из командной строки (рисунок 1).



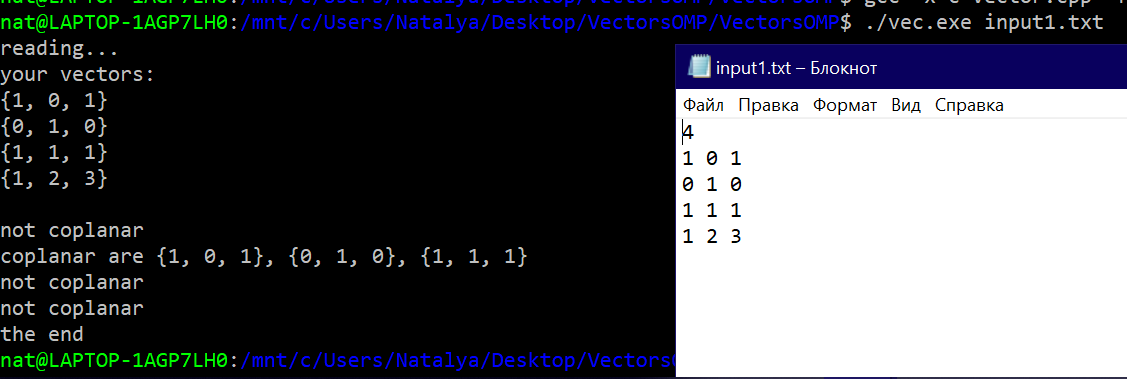
Рисунок 1 Команды компиляции и запуска программы



## Корректные значения

Программа осуществляет перебор троек векторов для нахождения компланарных. При некомпланарности тройки выводится сообщение без указания номера тройки, при компланарности – векторы, входящие в нее (рисунок 2, рисунок 3).

*Рисунок 2 Работа программы при корректных данных*

**

*Рисунок 3 Работа программы при корректных данных*

## Некорректные значения

Программа также обрабатывает случаи ввода некорректных данных, например когда число векторов меньше указанного в файле числа (в этом случае считывается максимально возможное число векторов) (рисунок 4).

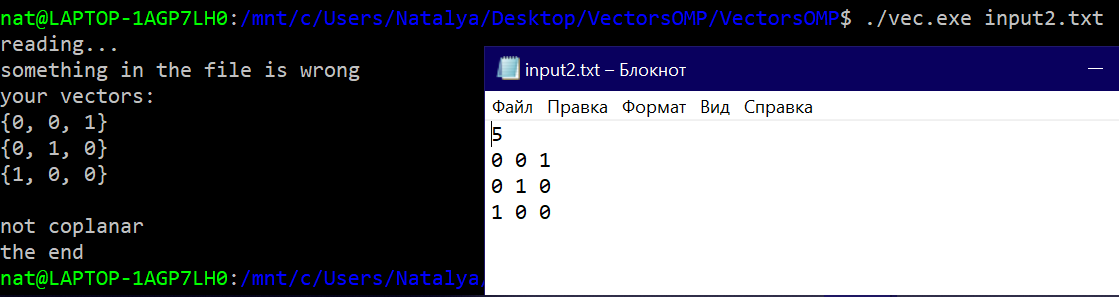


Рисунок 4 Работа программы при количестве векторов меньше указанного

Если указанное число не считывается верно, выводится сообщение об ошибке, и работа программы завершается (рисунок 5).

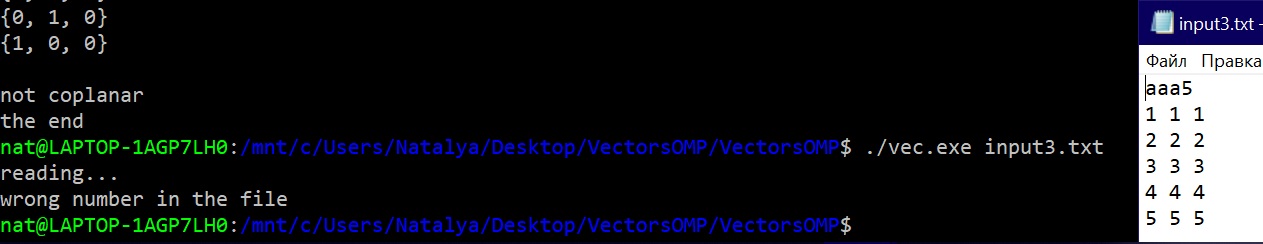


Рисунок 5 Работа программы при некорректном количестве векторов

Если какой-то из элементов векторов задан не числом, работа осуществляется со считанными до этого векторами (рисунок 6).

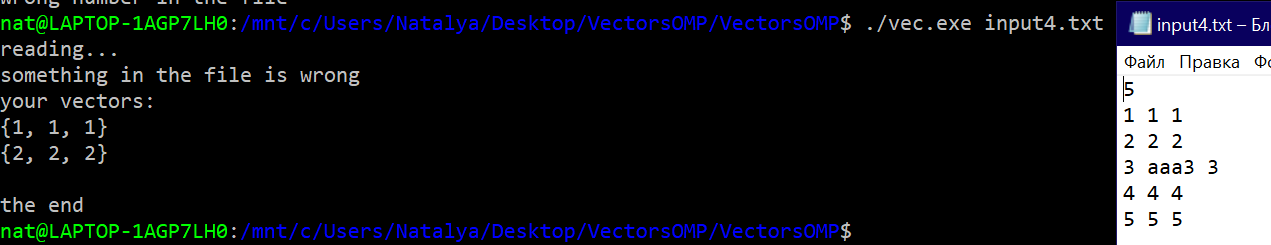


Рисунок 6 Работа программы при некорректном элементе вектора

В случае, когда векторов меньше трех, выводится сообщение о нехватке векторов и работа программы завершается (рисунок 7).

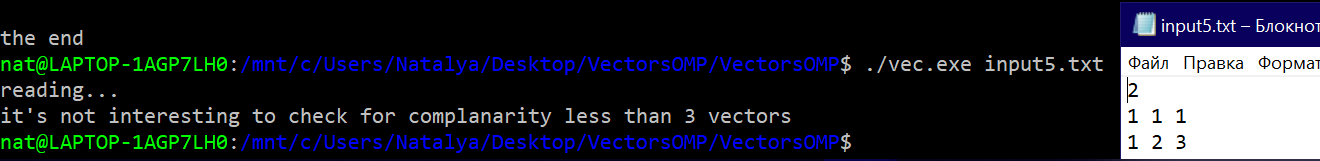


Рисунок 7 Сообщение об ошибке и завершение работы при недостаточном количестве векторов

Если векторов задано слишком много (больше 50), выводится сообщение об избытке и работа осуществляется с 50 векторами (рисунок 8, рисунок 9).

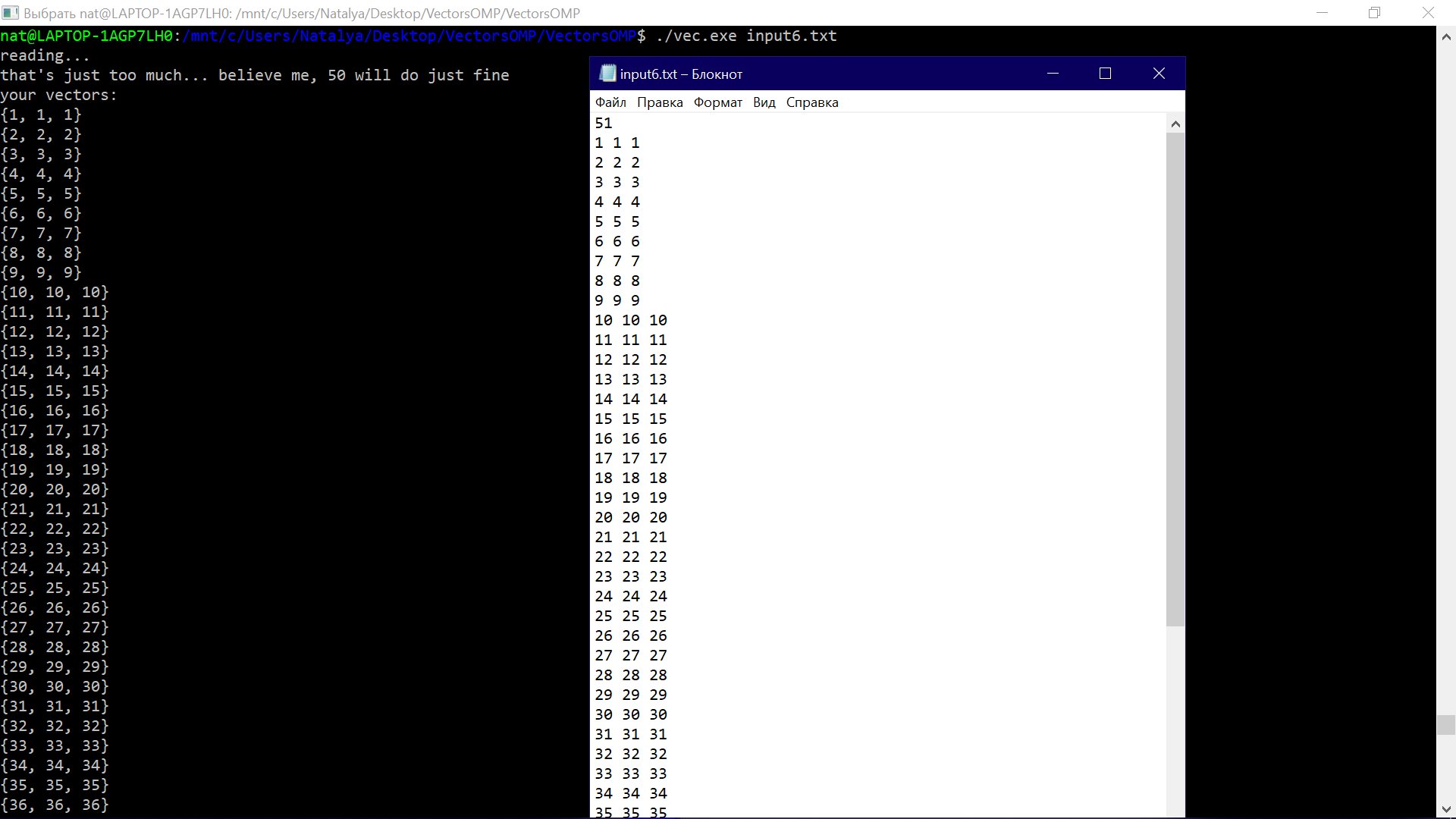


Рисунок 8 Работа программы при числе векторов больше 50 – обрезка количества

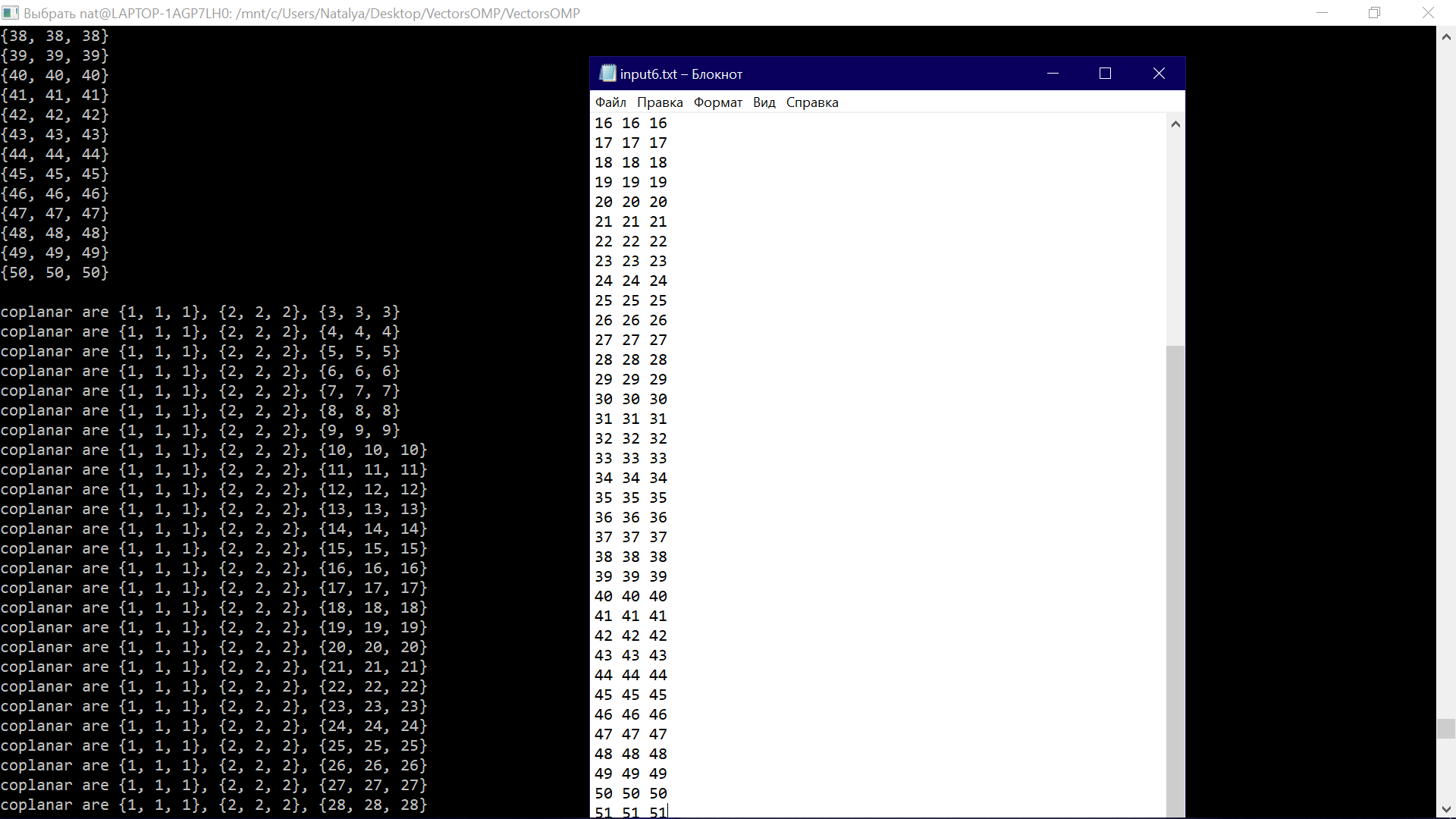


Рисунок 9 Работа программы при числе векторов больше 50

# Список литературы

[1] Инструкция по составлению пояснительной записки [Электронный ресурс]. //URL: http://softcraft.ru/edu/ comparch/tasks/mp01/ (Дата обращения: 30.10.2020, режим доступа: свободный)

[2] Статья «Coplanarity» Wikipedia.org //URL: https://en.wikipedia.org/wiki/Coplanarity (Дата обращения: 15.11.2020, режим доступа: свободный)

[3] Статья «Triple product» Wikipedia.org //URL: https://en.wikipedia.org/wiki/Triple\_product#Scalar\_triple\_product (Дата обращения: 15.11.2020, режим доступа: свободный)

[4] Практические приемы построения многопоточных приложений [Электронный ресурс].  
//URL: [http://softcraft.ru/edu/comparch/tasks/t03/](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fsoftcraft.ru%2Fedu%2Fcomparch%2Ftasks%2Ft03%2F) (Дата обращения: 15.11.2020, режим доступа: свободный)

# Приложение кода

1. #include <omp.h>
2. #include <stdlib.h>
3. #include <stdbool.h>
4. #include <stdio.h>
5. /// <summary>
6. /// Структура для вектора, состоящая из трех координат.
7. /// </summary>
8. typedef struct Vector {
9. double x, y, z;
10. } Vector;
11. /// <summary>
12. /// Глобальные переменные для массива считанных векторов и их количества.
13. /// </summary>
14. Vector\* vectors;
15. size\_t numberOfVectors;
16. /// <summary>
17. /// Метод для чтения из файла, в котором должно быть записано
18. /// число векторов, а далее векторы покоординатно.
19. /// В случае неверного формата происходит обработка ошибок.
20. /// </summary>
21. void read(char\* filename) {
22. FILE\* file = fopen(filename, "r");
23. if (!file) {
24. perror(filename);
25. exit(1);
26. }
27. if (fscanf(file, "%ld", &numberOfVectors) != 1) {
28. printf("wrong number in the file\n");
29. fclose(file);
30. exit(1);
31. }
32. /// Проверка на то, что количество векторов не меньше 3.
33. /// Если меньше, то искать компланарные тройки бесполезно.
34. /// Завершаем программу.
35. if (numberOfVectors < 3)
36. {
37. printf("it's not interesting to check for complanarity less than 3 vectors\n");
38. fclose(file);
39. exit(0);
40. }
41. /// Проверка на то, что количество векторов не превышает 50.
42. /// Если превышает, мы все же не будем считать больше 50 векторов.
43. if (numberOfVectors > 50)
44. {
45. printf("that's just too much... believe me, 50 will do just fine\n");
46. numberOfVectors = 50;
47. }
48. /// Выделяем память для векторов в глобальной переменной и считываем.
49. vectors = (Vector\*)calloc(numberOfVectors, sizeof(Vector));
50. for (size\_t i = 0; i < numberOfVectors; i++) {
51. if (fscanf(file, "%lf", &vectors[i].x) != 1) {
52. numberOfVectors = i;
53. printf("something in the file is wrong\n");
54. break;
55. }
56. if (fscanf(file, "%lf", &vectors[i].y) != 1) {
57. numberOfVectors = i;
58. printf("something in the file is wrong\n");
59. break;
60. }
61. if (fscanf(file, "%lf", &vectors[i].z) != 1) {
62. numberOfVectors = i;
63. printf("something in the file is wrong\n");
64. break;
65. }
66. }
67. fclose(file);
68. }
69. /// <summary>
70. /// Метод для печати считанных векторов в консоль.
71. /// </summary>
72. void printVectors(Vector\* vectors) {
73. printf("your vectors:\n");
74. for (size\_t i = 0; i < numberOfVectors; i++)
75. {
76. printf("{%g, %g, %g}\n", vectors[i].x, vectors[i].y, vectors[i].z);
77. }
78. printf("\n");
79. }
80. /// <summary>
81. /// Метод для проверки, является ли тройка векторов компланарной через
82. /// равенство смешанного произведения нулю.
83. /// </summary>
84. bool coplanar(Vector a, Vector b, Vector c) {
85. int value = (a.x \* b.y \* c.z) + (a.y \* b.z \* c.x) + (a.z \* b.x \* c.y) -
86. (a.z \* b.y \* c.x) - (a.x \* b.z \* c.y) - (b.x \* a.y \* c.z);
87. return value == 0;
88. }
89. /// <summary>
90. /// Функция, выполняемая потоком. Каждый поток прикреплен к
91. /// первому вектору в тройке и внутри него происходит подбор
92. /// второго и третьего векторов, проверка их на компланарность
93. /// и вывод результата.
94. /// </summary>
95. void threadFunction(size\_t i) {
96. for (size\_t j = i + 1; j < numberOfVectors; ++j) {
97. for (size\_t k = j + 1; k < numberOfVectors; ++k) {
98. if (coplanar(vectors[i], vectors[j], vectors[k])) {
99. printf("coplanar are {%g, %g, %g}, {%g, %g, %g}, {%g, %g, %g}\n",
100. vectors[i].x, vectors[i].y, vectors[i].z,
101. vectors[j].x, vectors[j].y, vectors[j].z,
102. vectors[k].x, vectors[k].y, vectors[k].z);
103. }
104. else { printf("not coplanar\n"); }
105. }
106. }
107. }
108. /// <summary>
109. /// Метод для организации потоковой работы. Потоки создаются
110. /// с помощью директивы параллельного цикла, в котором вызывается
111. /// функция. Программа сама выбирает оптимальное количество потоков
112. /// для распараллеливания цикла.
113. /// </summary>
114. void threadWork() {
115. #pragma omp parallel for
116. for (size\_t i = 0; i < numberOfVectors; i++) {
117. threadFunction(i);
118. }
119. }
120. /// <summary>
121. /// Точка входа, если аргументы входной строки верны,
122. /// отсюда вызываются методы чтения, вывода
123. /// считанных векторов, проверки компланарности.
124. /// </summary>
125. int main(int argc, char\*\* argv) {
126. if (argc != 2) {
127. printf("wrong number of args %d\n", argc);
128. return 1;
129. }
130. char\* input = argv[1];
131. printf("reading...\n");
132. read(input);
133. printVectors(vectors);
134. threadWork();
135. printf("the end\n");
136. /// Освобождение памяти, выделенной для векторов.
137. free(vectors);
138. return 0;
139. }