Федеральное агентство связи  
Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
 «Сибирский государственный университет  
телекоммуникаций и информатики»

Кафедра ВС

Лабораторная работа №1 по дисциплине

«Параллельные вычислительные технологии» по теме:

«Оптимизация доступа к памяти, циклов и ветвлений»

Выполнил:

ст. гр. ИВ-823

Шиндель Э. Д.

Проверил:

Заведующий

кафедрой ВС

Курносов М. Г.

Новосибирск, 2020

**Содержание:**

1. Задание 13

2. Задание 25

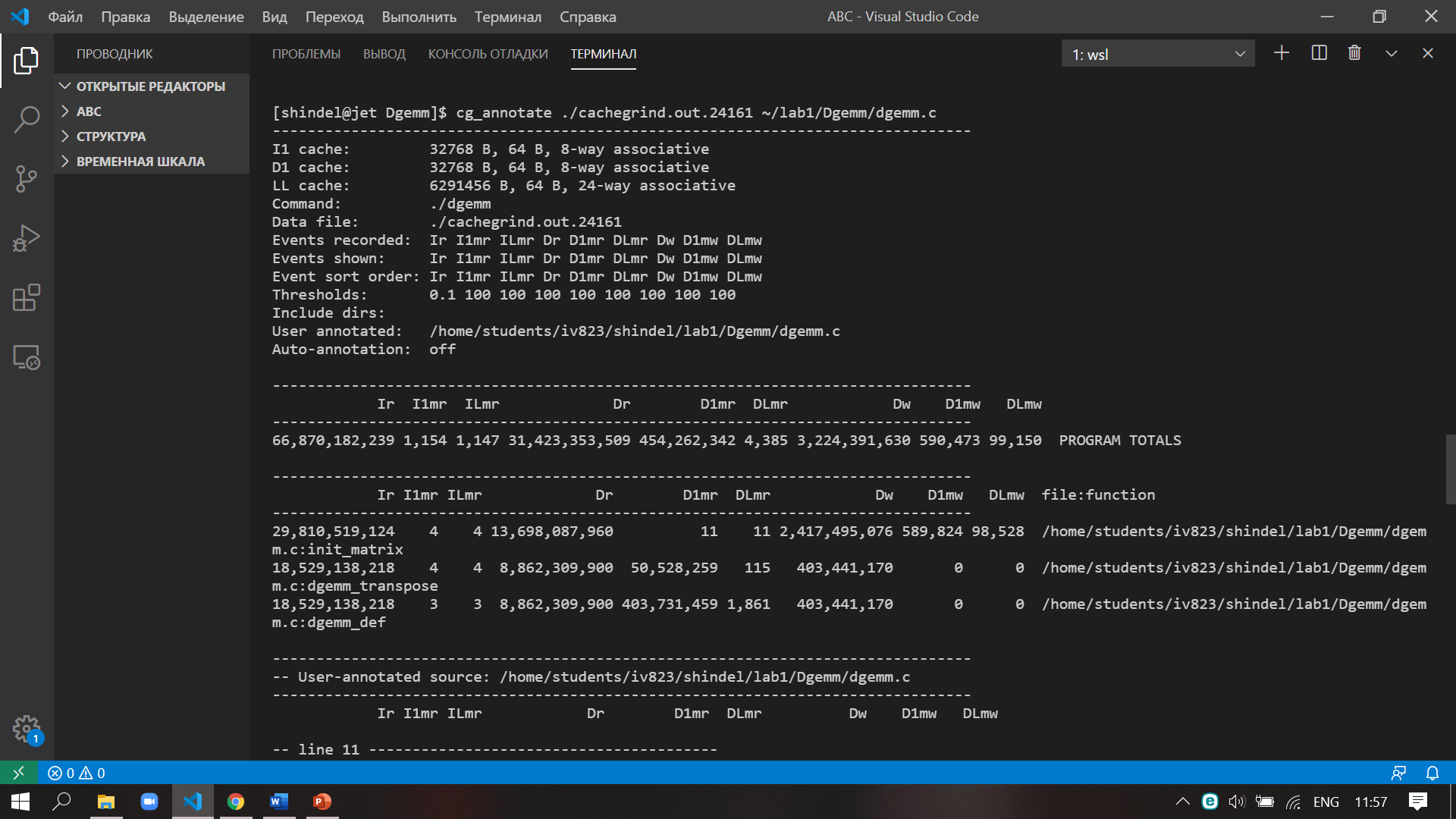
3. Задание 37

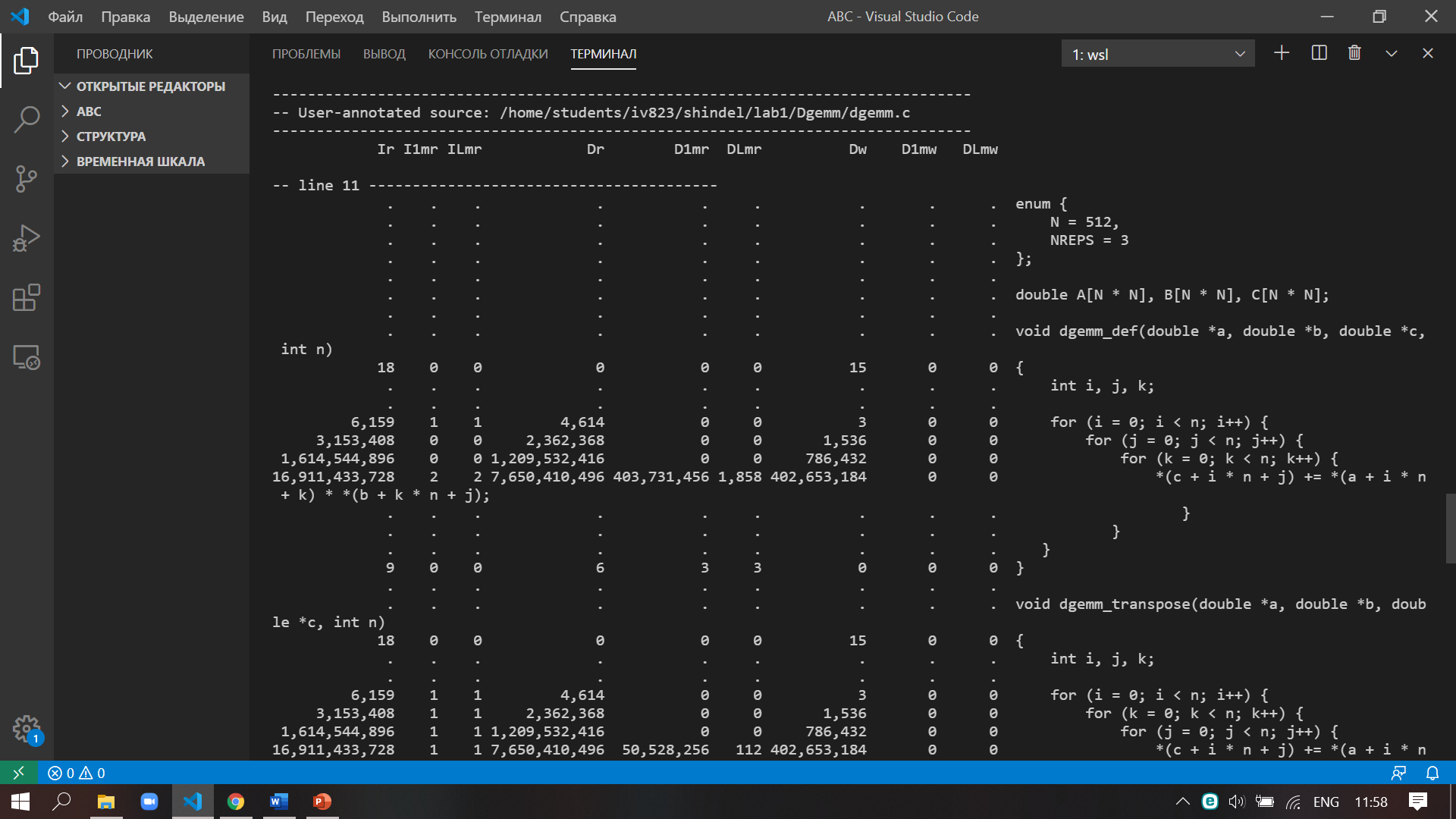
4. Листинг9

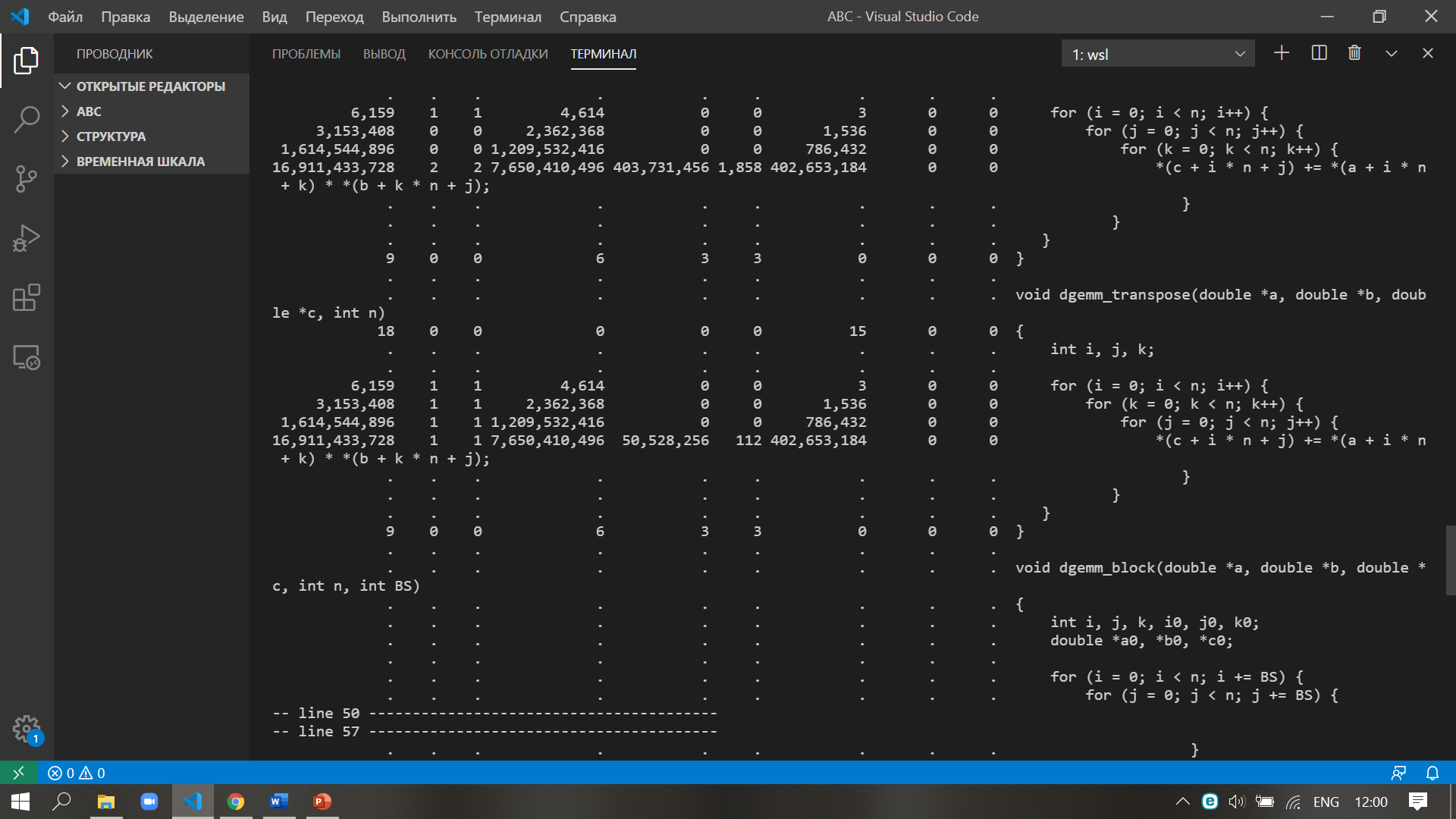
**Задание 1**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **dgemm\_def** | **dgemm\_transpose** | | **dgemm\_block** | | | | | | | |
| Time, s | Time, s | Speedup | BS = 2 | | BS = 4 | | BS = 8 | | BS = 16 | |
| Time, s | Speedup | Time, s | Speedup | Time, s | Speedup | Time, s | Speedup |
| **1,727** | **1,299** | **1,33** | **1,457** | **1,19** | **1,077** | **1,6** | **0,966** | **1,788** | **0,941** | **1,84** |

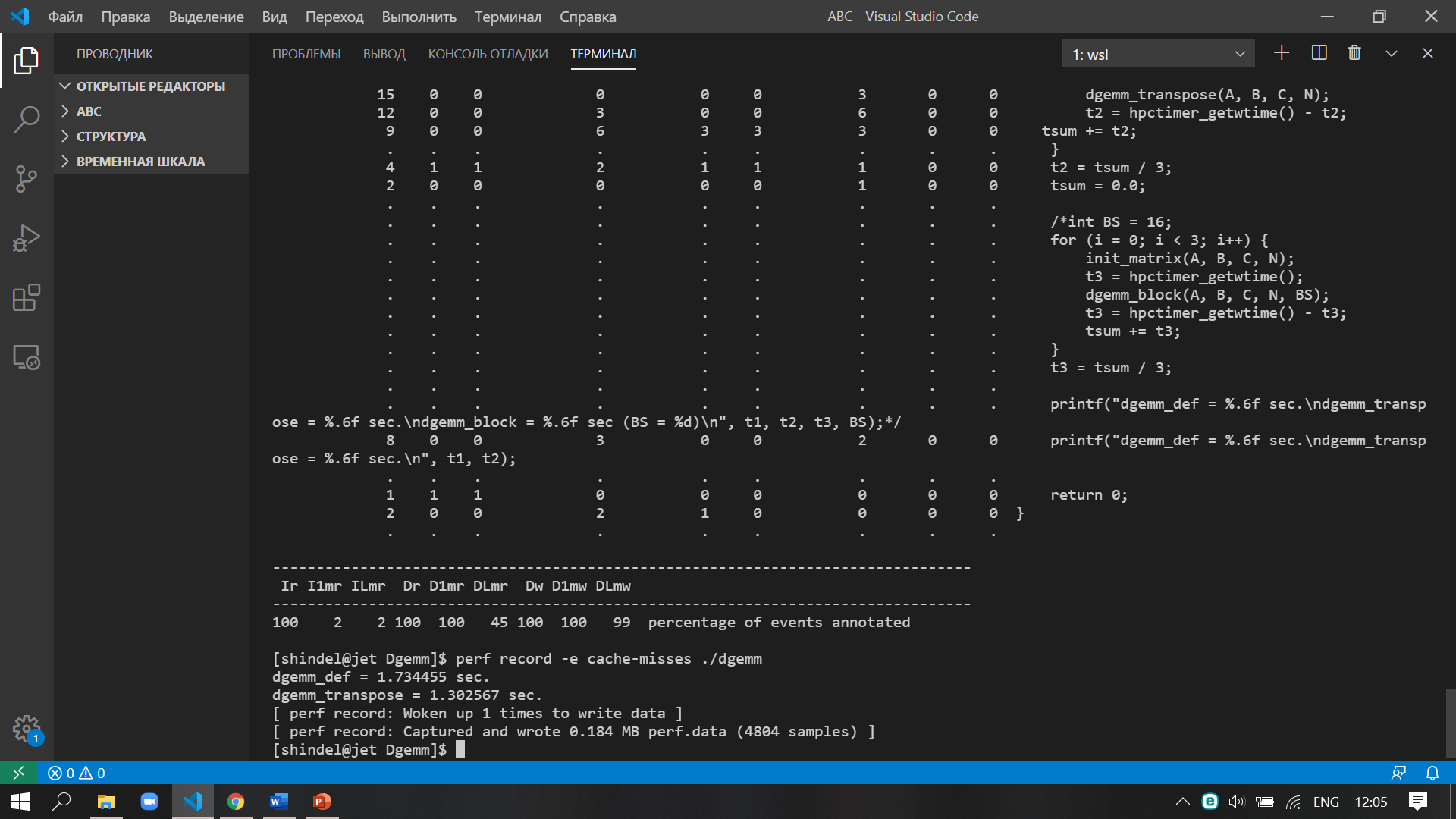
Эксперимент проводился на кластере Jet

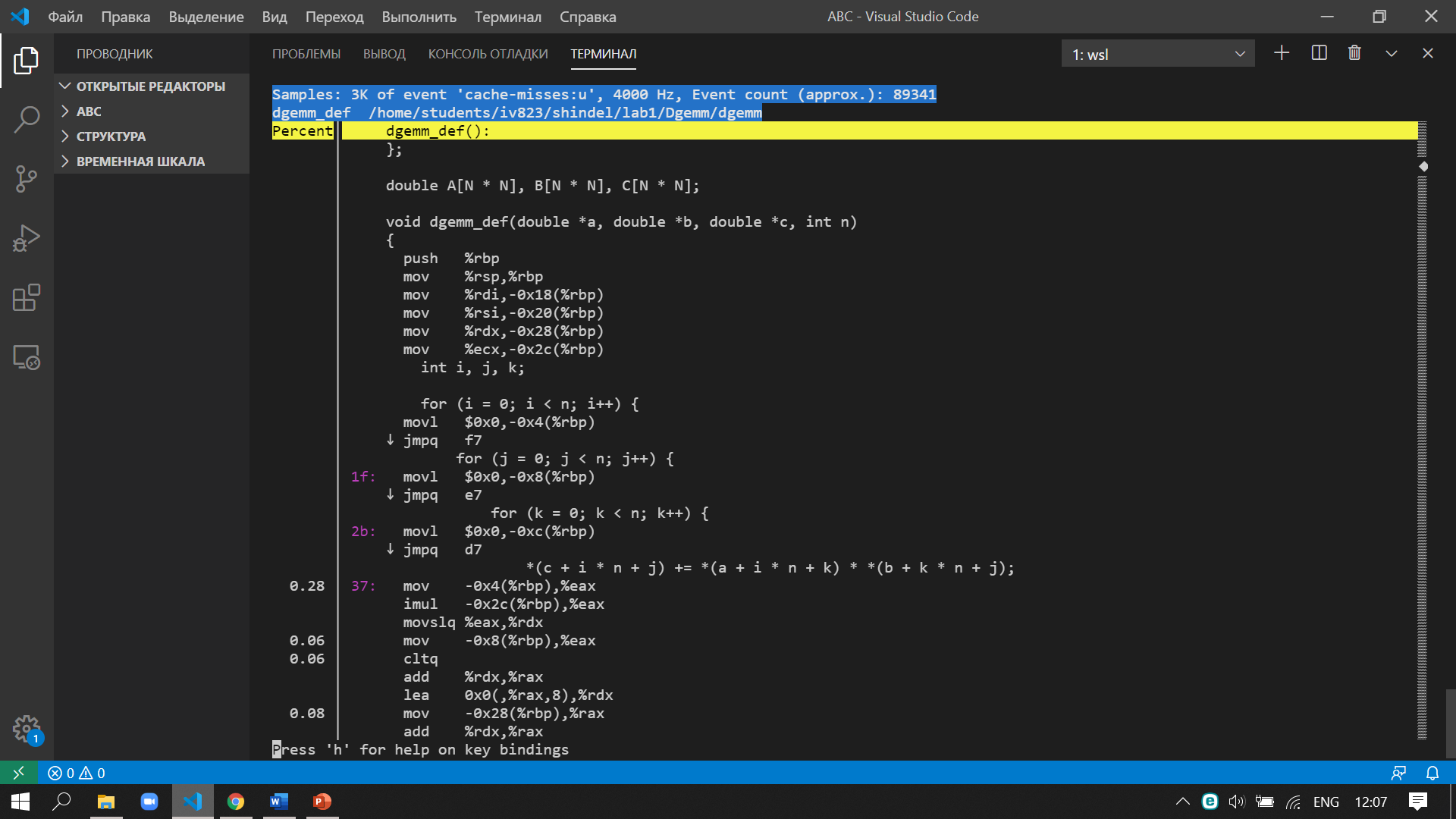


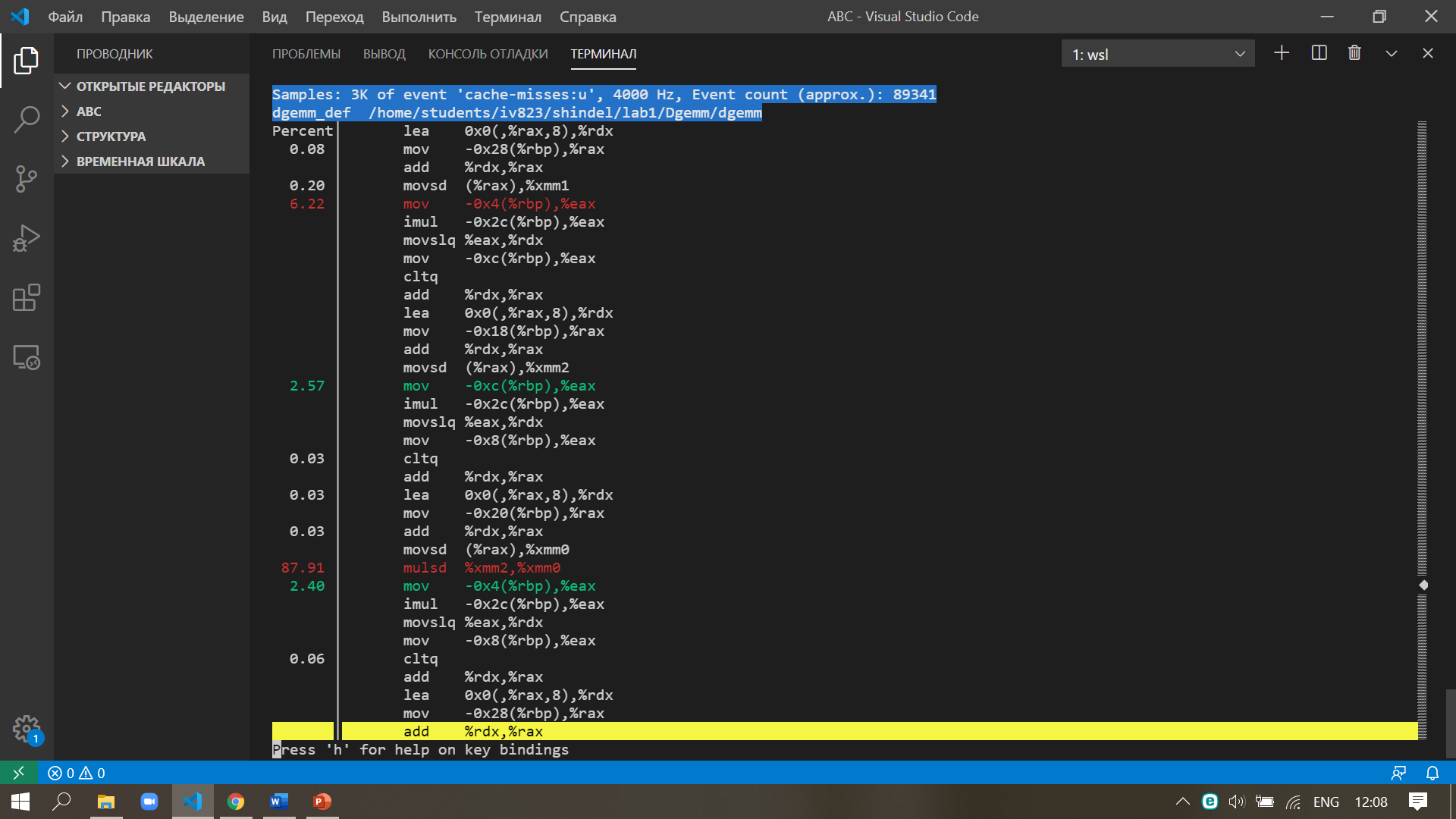


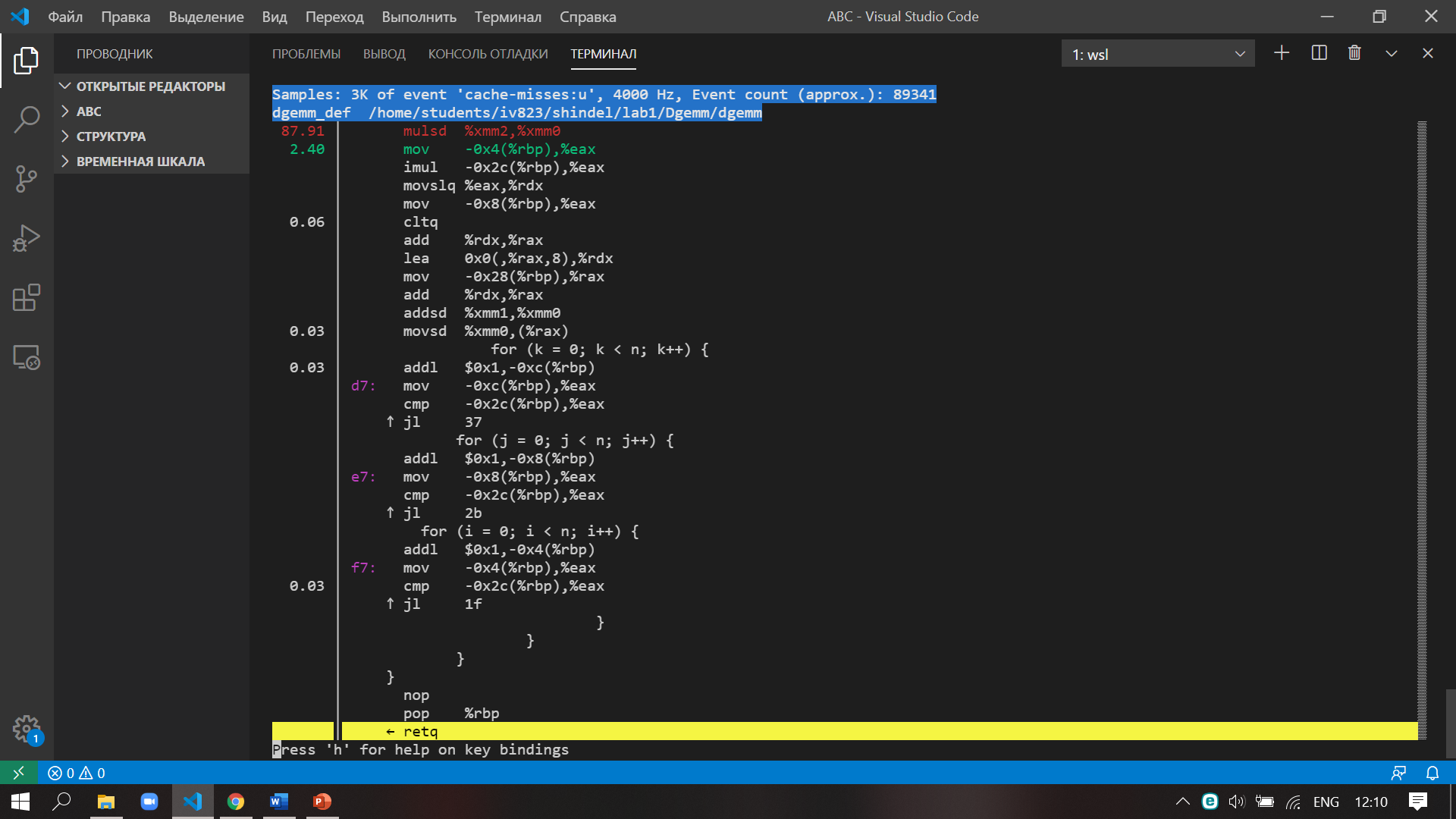


Аннотированный исходный текст программы, сформированный с помощью Valgrind





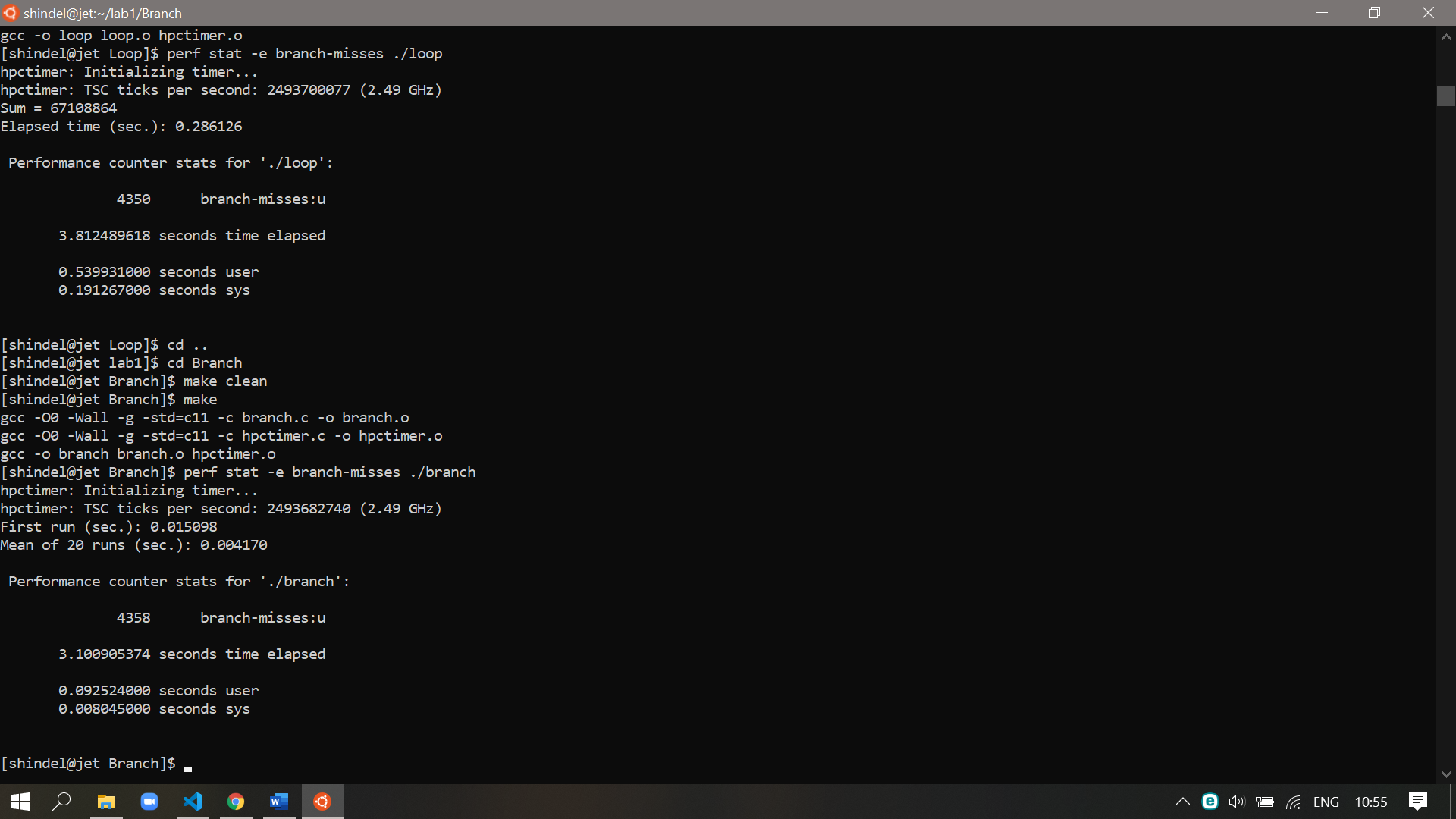




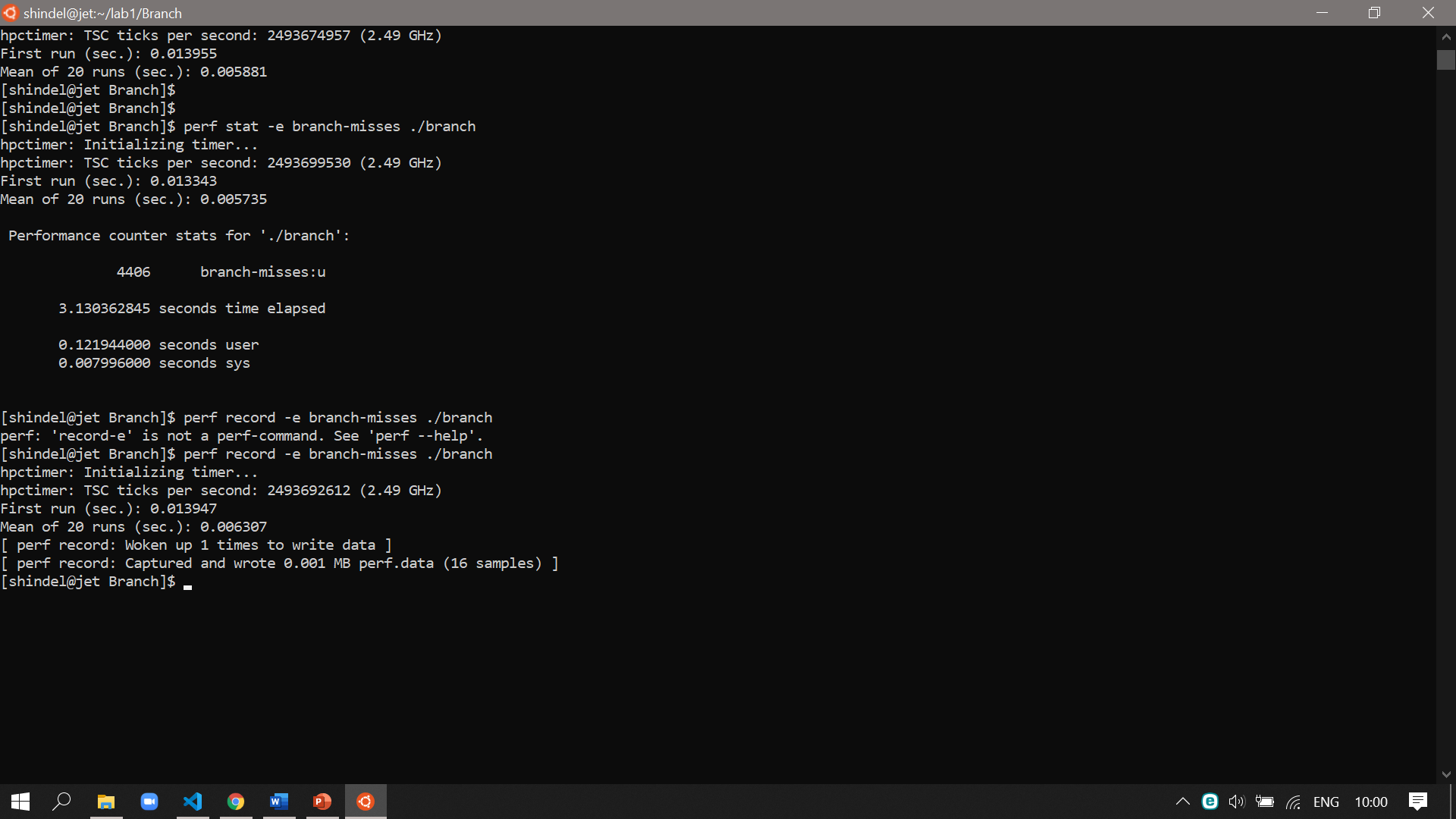
Аннотированный исходный текст программы, сформированный с помощью профилировщика perf

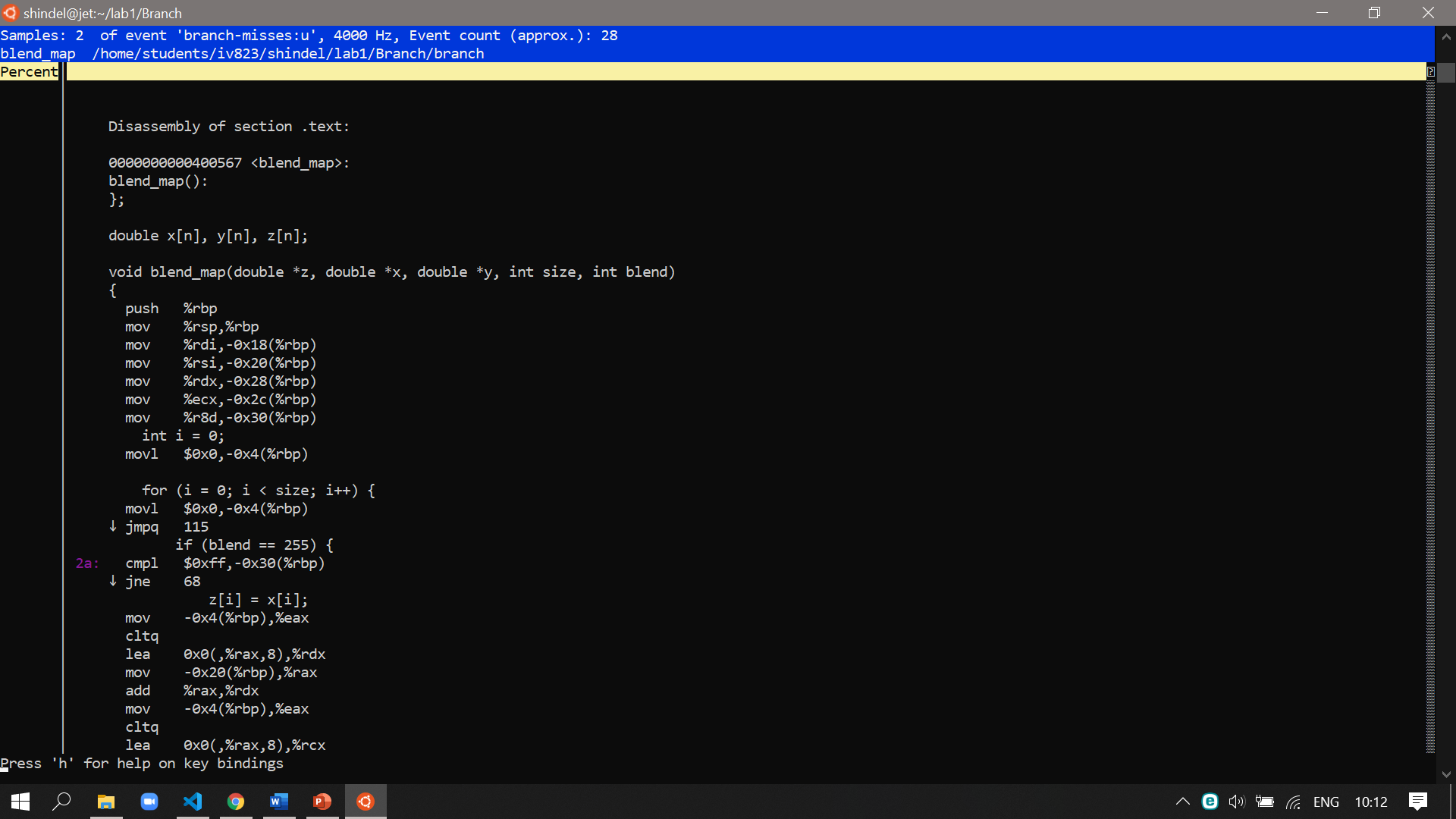
**Задание 2**

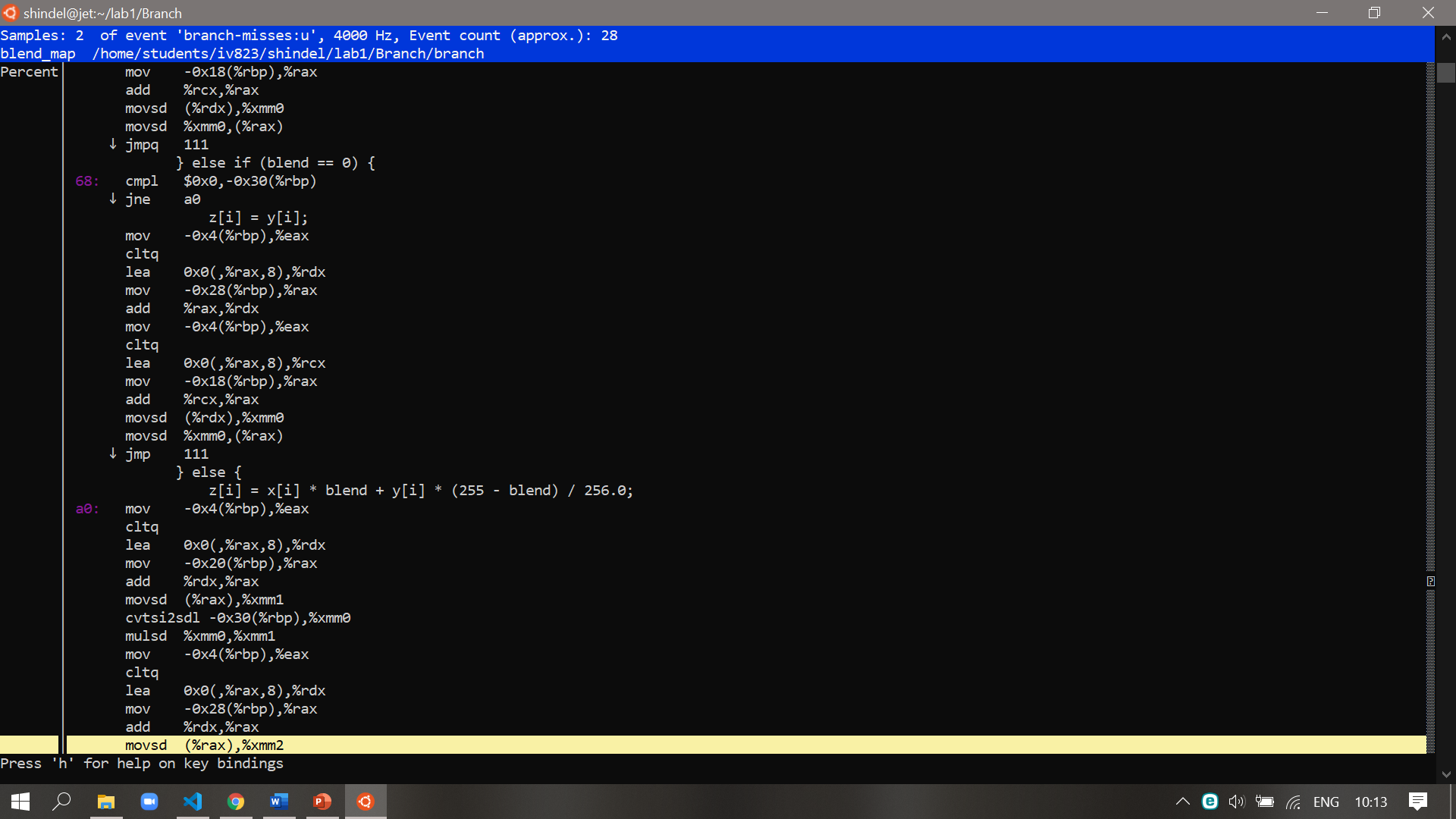
|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **Время выполнения функции blend\_map** | **Время выполнения функции blend\_map\_opt** | **Ускорение (speedup)** |
| 100 000 | 0,000557 | 0,00039 | **1,43** |
| 1 000 000 | 0,00603 | 0,004227 | **1,43** |

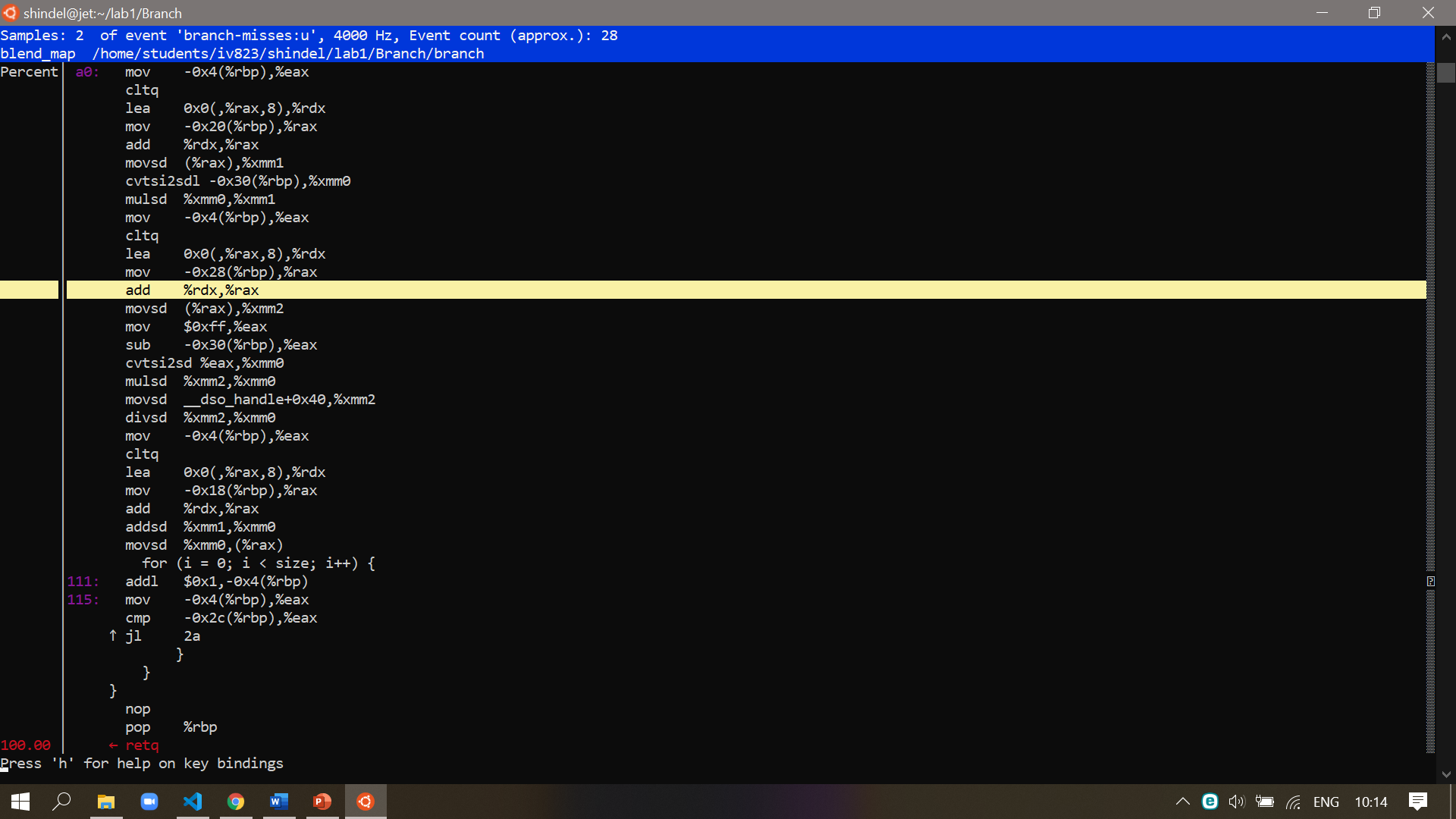


Количество ветвлений в bland\_map

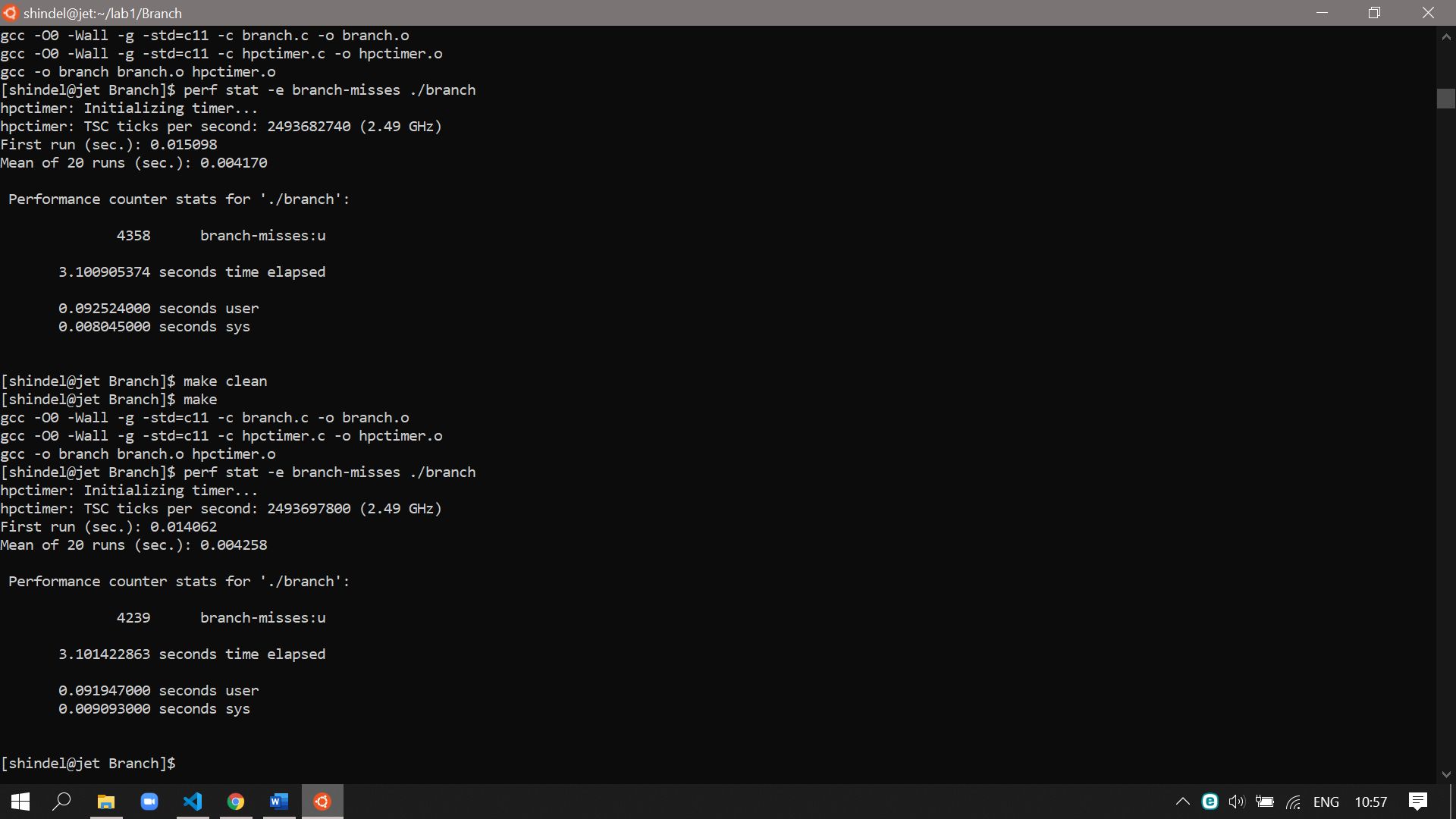








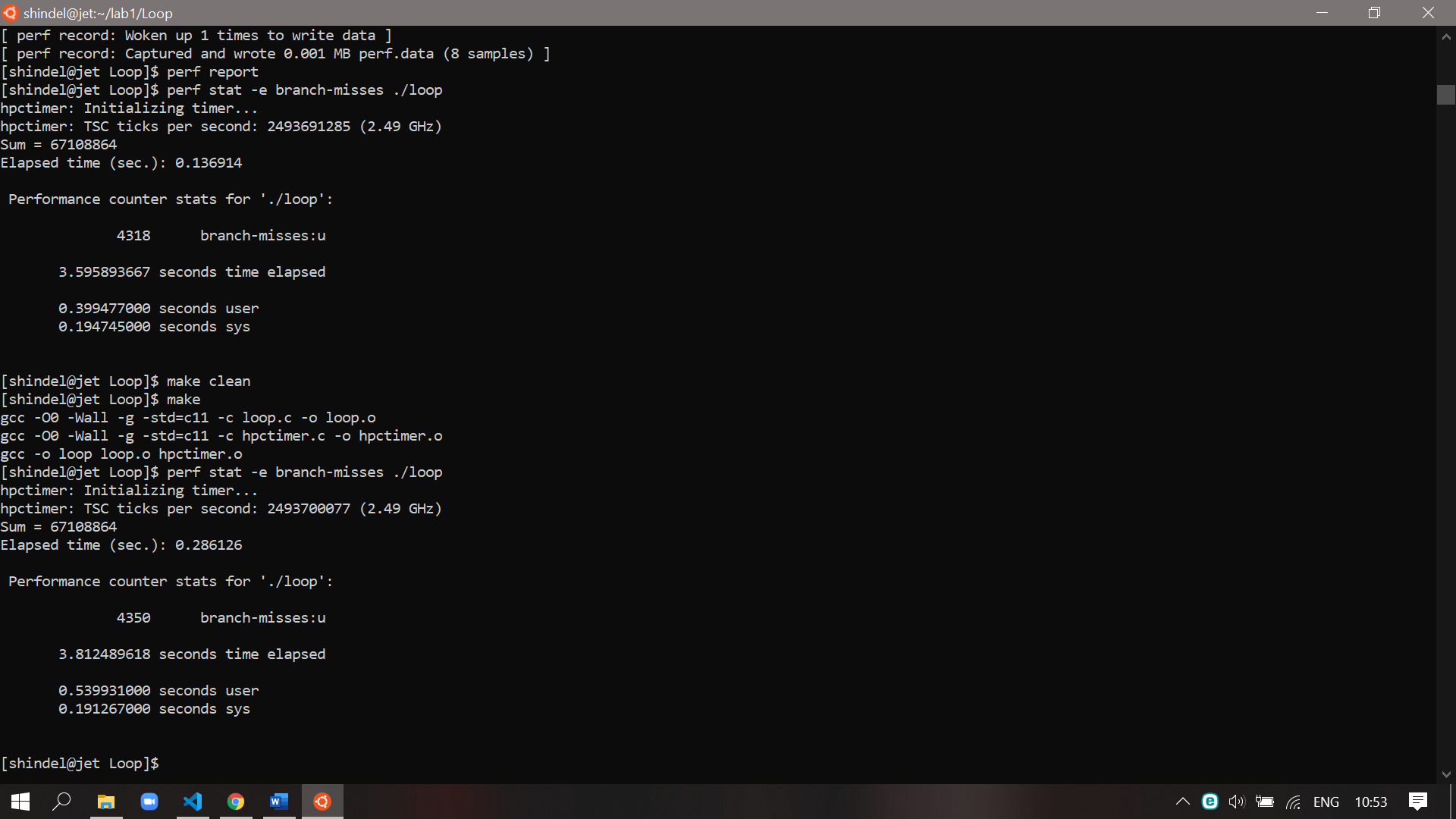
Аннотированный исходный код программы bland\_map



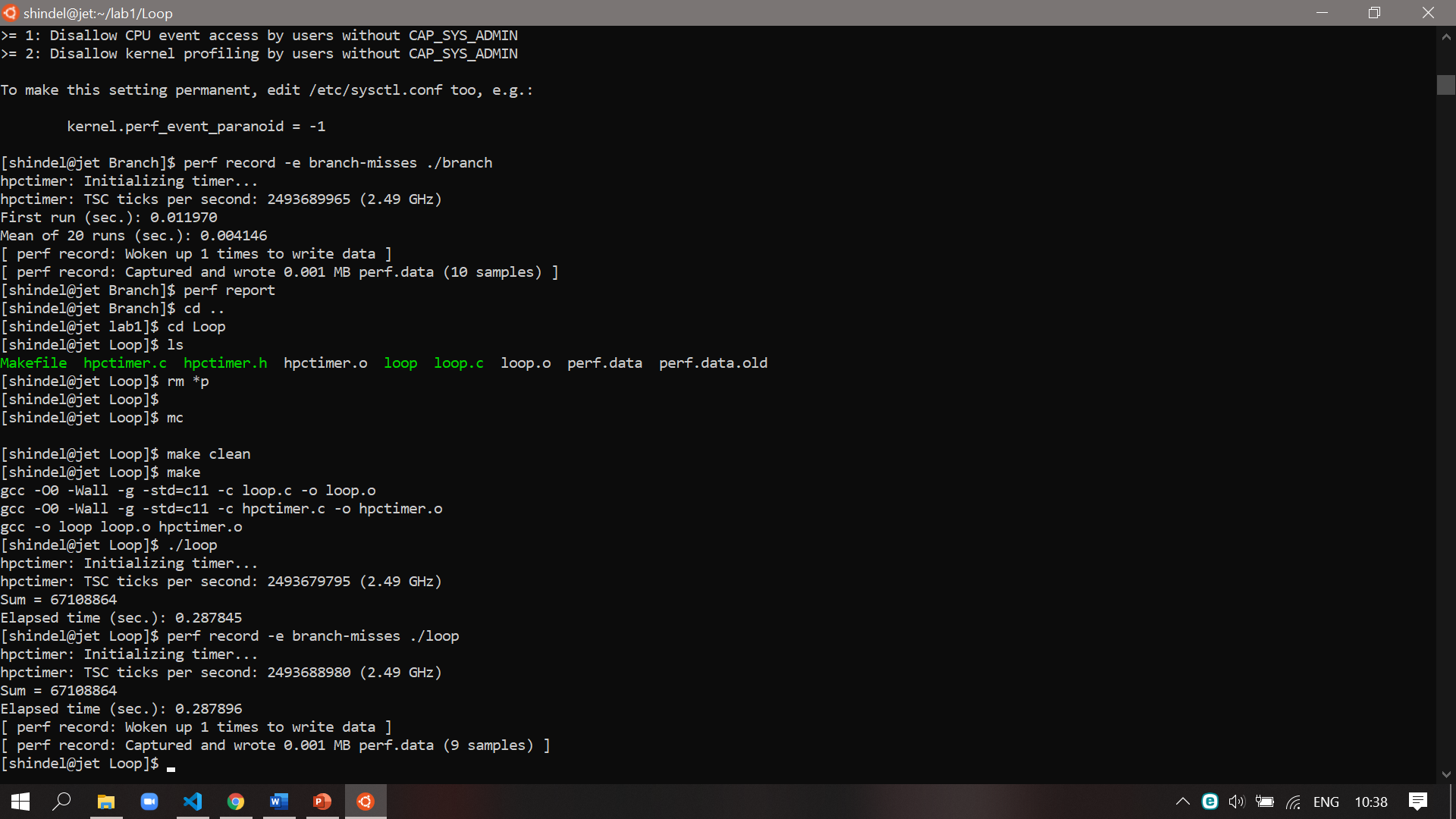
Количество ветвлений в bland\_map\_opt

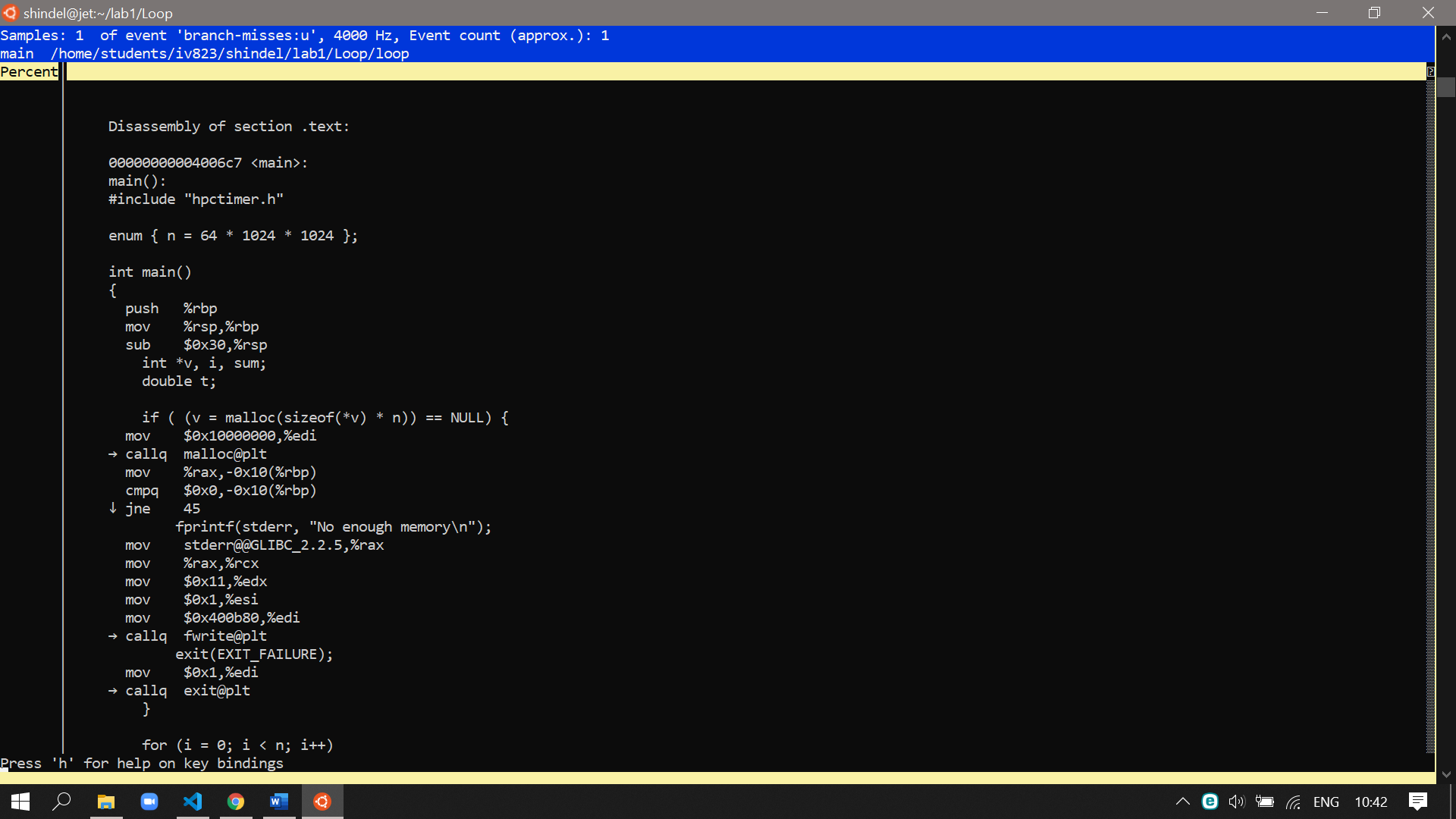
**Задание 3**

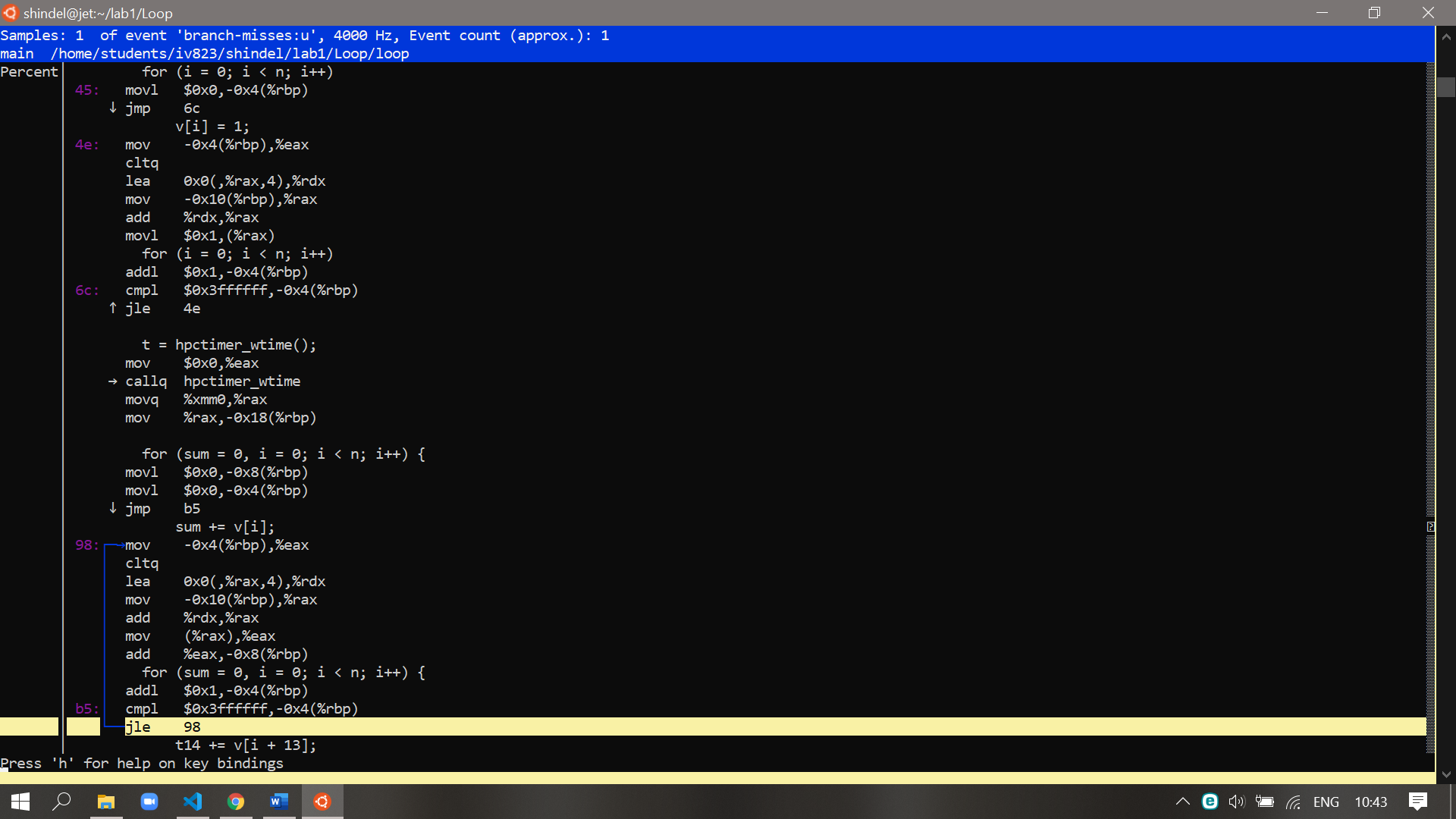
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | **Глубина раскрутки цикла** | | | | | | | | | |
| **--** | **2** | | **4** | | **8** | | **16** | | |
| Time | Time | Speedup | Time | Speedup | Time | Speedup | Time | Speedup |
| 16 MiB | 0,073 | 0,048 | 1,52 | 0,039 | 1,87 | 0,037 | 1,97 | 0,035 | 2,09 |
| 64 MiB | 0,285 | 0,189 | 1,51 | 0,157 | 1,82 | 0,147 | 1,94 | 0,138 | 2,07 |

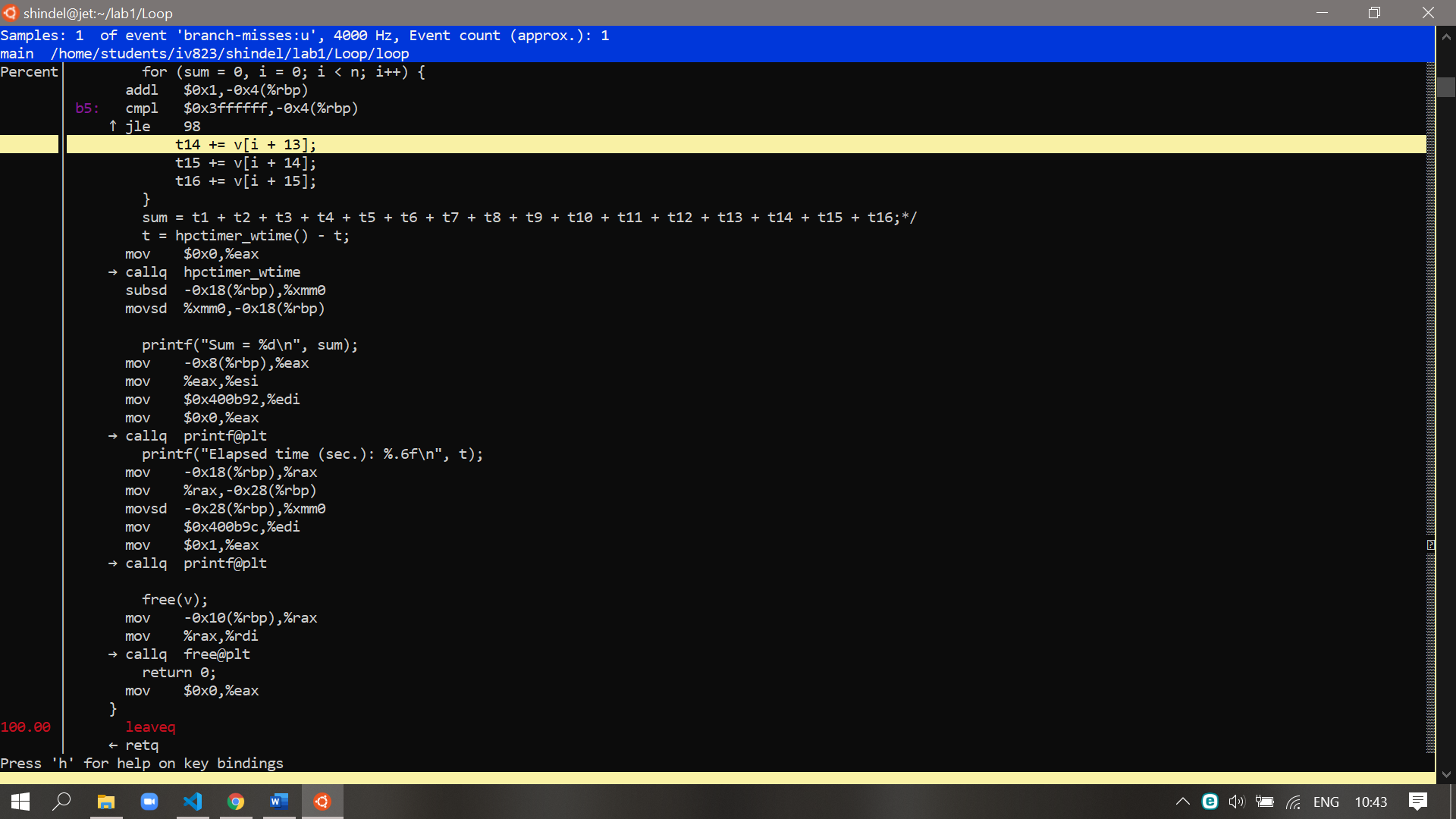


Количество ветвлений в исходной программе

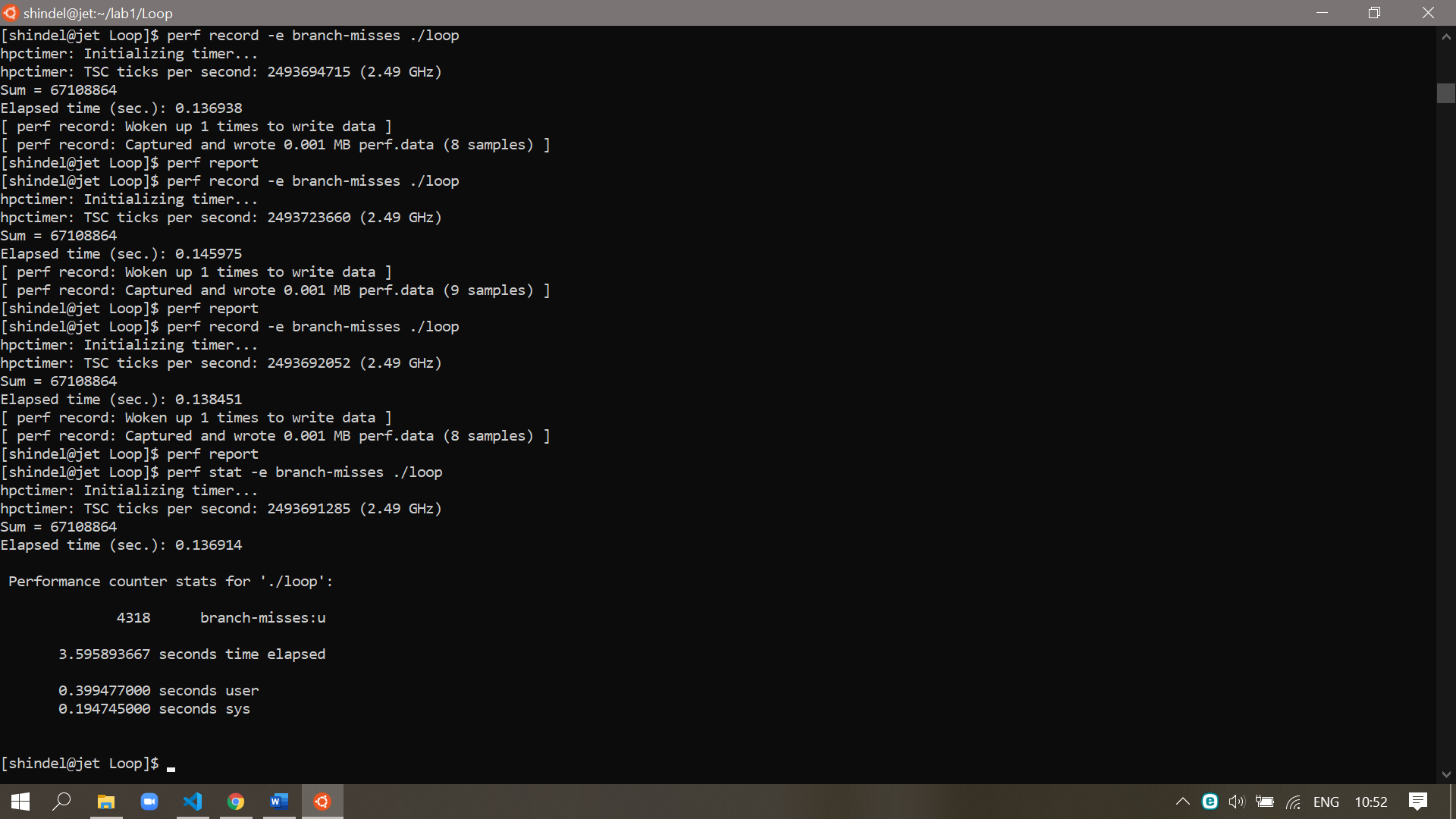








Аннотированный исходный код программы



Количество ветвлений в программе с развёрнутым циклом

**Листинг**

/\*

 \* dgemm.c: DGEMM - Double-precision General Matrix Multiply.

 \*

 \*/

#include <stdio.h>

#include <stdlib.h>

#include "hpctimer.h"

enum {

    N = 512,

    NREPS = 3

};

double A[N \* N], B[N \* N], C[N \* N];

void dgemm\_def(double \*a, double \*b, double \*c, int n)

{

    int i, j, k;

    for (i = 0; i < n; i++) {

        for (j = 0; j < n; j++) {

            for (k = 0; k < n; k++) {

                \*(c + i \* n + j) += \*(a + i \* n + k) \* \*(b + k \* n + j);

            }

        }

    }

}

void dgemm\_transpose(double \*a, double \*b, double \*c, int n)

{

    int i, j, k;

    for (i = 0; i < n; i++) {

        for (k = 0; k < n; k++) {

            for (j = 0; j < n; j++) {

                \*(c + i \* n + j) += \*(a + i \* n + k) \* \*(b + k \* n + j);

            }

        }

    }

}

void dgemm\_block(double \*a, double \*b, double \*c, int n, int BS)

{

    int i, j, k, i0, j0, k0;

    double \*a0, \*b0, \*c0;

    for (i = 0; i < n; i += BS) {

        for (j = 0; j < n; j += BS) {

            for (k = 0; k < n; k += BS) {

                for (i0 = 0, c0 = (c + i \* n + j), a0 = (a + i \* n + k);

i0 < BS; ++i0, c0 += n, a0 += n) {

                    for (k0 = 0, b0 = (b + k \* n + j);

k0 < BS; ++k0, b0 += n) {

                        for (j0 = 0; j0 < BS; ++j0) {

                            c0[j0] += a0[k0] \* b0[j0];

                        }

                    }

                }

            }

        }

    }

}

void init\_matrix(double \*a, double \*b, double \*c, int n)

{

    int i, j, k;

    for (i = 0; i < n; i++) {

        for (j = 0; j < n; j++) {

            for (k = 0; k < n; k++) {

                \*(a + i \* n + j) = 1.0;

                \*(b + i \* n + j) = 2.0;

                \*(c + i \* n + j) = 0.0;

            }

        }

    }

}

void print\_matrix(double \*a, int n)

{

    int i, j;

    printf("Matrix:\n");

    for (i = 0; i < n; i++) {

        for (j = 0; j < n; j++) {

            printf("%12.2f", \*(a + i \* n + j));

        }

        printf("\n");

    }

}

int main(int argc, char \*\*argv)

{

    int i;

    double t1, t2, tsum = 0.0;

    for (i = 0; i < 3; i++) {

        init\_matrix(A, B, C, N);

        t1 = hpctimer\_getwtime();

        dgemm\_def(A, B, C, N);

        t1 = hpctimer\_getwtime() - t1;

    tsum += t1;

    }

    t1 = tsum / 3;

    tsum = 0.0;

    for (i = 0; i < 3; i++) {

        init\_matrix(A, B, C, N);

    t2 = hpctimer\_getwtime();

        dgemm\_transpose(A, B, C, N);

        t2 = hpctimer\_getwtime() - t2;

    tsum += t2;

    }

    t2 = tsum / 3;

    tsum = 0.0;

    /\*int BS = 16;

    for (i = 0; i < 3; i++) {

        init\_matrix(A, B, C, N);

        t3 = hpctimer\_getwtime();

        dgemm\_block(A, B, C, N, BS);

        t3 = hpctimer\_getwtime() - t3;

        tsum += t3;

    }

    t3 = tsum / 3;

    printf("dgemm\_def = %.6f sec.\ndgemm\_transpose = %.6f sec.\ndgemm\_block = %.6f sec (BS = %d)\n", t1, t2, t3, BS);\*/

    printf("dgemm\_def = %.6f sec.\ndgemm\_transpose = %.6f sec.\n", t1, t2);

    return 0;

}

\*

 \* branch.c:

 \*

 \*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "hpctimer.h"

enum {

    n = 1000000,

    nreps = 20

};

double x[n], y[n], z[n];

void blend\_map(double \*z, double \*x, double \*y, int size, int blend)

{

    int i = 0;

    for (i = 0; i < size; i++) {

        if (blend == 255) {

            z[i] = x[i];

        } else if (blend == 0) {

            z[i] = y[i];

        } else {

            z[i] = x[i] \* blend + y[i] \* (255 - blend) / 256.0;

        }

    }

}

void blend\_map\_opt(double \*dest, double \*a, double \*b, int size, int blend)

{

    int i = 0;

    if (blend == 255) {

        for (i = 0; i < size; i++) {

            z[i] = x[i];

    }

    } else if (blend == 0) {

        for (i = 0; i < size; i++) {

        z[i] = y[i];

    }

    } else {

    for (i = 0; i < size; i++) {

            z[i] = x[i] \* blend + y[i] \* (255 - blend) / 256.0;

        }

    }

}

int main()

{

    double tfirst, t;

    int i;

    /\* First run: warmup \*/

    tfirst = hpctimer\_wtime();

    //blend\_map(z, x, y, n, 0);

    blend\_map\_opt(z, x, y, n, 0);

    tfirst = hpctimer\_wtime() - tfirst;

    /\* Measures \*/

    t = hpctimer\_wtime();

    for (i = 0; i < nreps; i++) {

        //blend\_map(z, x, y, n, 0);

        blend\_map\_opt(z, x, y, n, 0);

    }

    t = (hpctimer\_wtime() - t) / nreps;

    printf("First run (sec.): %.6f\n", tfirst);

    printf("Mean of %d runs (sec.): %.6f\n", nreps, t);

    return 0;

}

/\*

 \* loop.c:

 \*

 \*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "hpctimer.h"

enum { n = 64 \* 1024 \* 1024 };

int main()

{

    int \*v, i, sum;

    double t;

    if ( (v = malloc(sizeof(\*v) \* n)) == NULL) {

        fprintf(stderr, "No enough memory\n");

        exit(EXIT\_FAILURE);

    }

    for (i = 0; i < n; i++)

        v[i] = 1;

    t = hpctimer\_wtime();

    for (sum = 0, i = 0; i < n; i++) {

        sum += v[i];

    }

    /\*int t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15,

t16;

t1 = t2 = t3 = t4 = t5 = t6 = t7 = t8 = t9 = t10 = t11 = t12 = t13 = t14

= t15 = t16 = 0;

    for (sum = 0, i = 0; i < n; i += 16) {

        t1 += v[i];

        t2 += v[i + 1];

     t3 += v[i + 2];

     t4 += v[i + 3];

     t5 += v[i + 4];

     t6 += v[i + 5];

     t7 += v[i + 6];

     t8 += v[i + 7];

     t9 += v[i + 8];

     t10 += v[i + 9];

     t11 += v[i + 10];

     t12 += v[i + 11];

     t13 += v[i + 12];

     t14 += v[i + 13];

     t15 += v[i + 14];

     t16 += v[i + 15];

    }

    sum = t1 + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13

+ t14 + t15 + t16;\*/

    t = hpctimer\_wtime() - t;

    printf("Sum = %d\n", sum);

    printf("Elapsed time (sec.): %.6f\n", t);

    free(v);

    return 0;

}