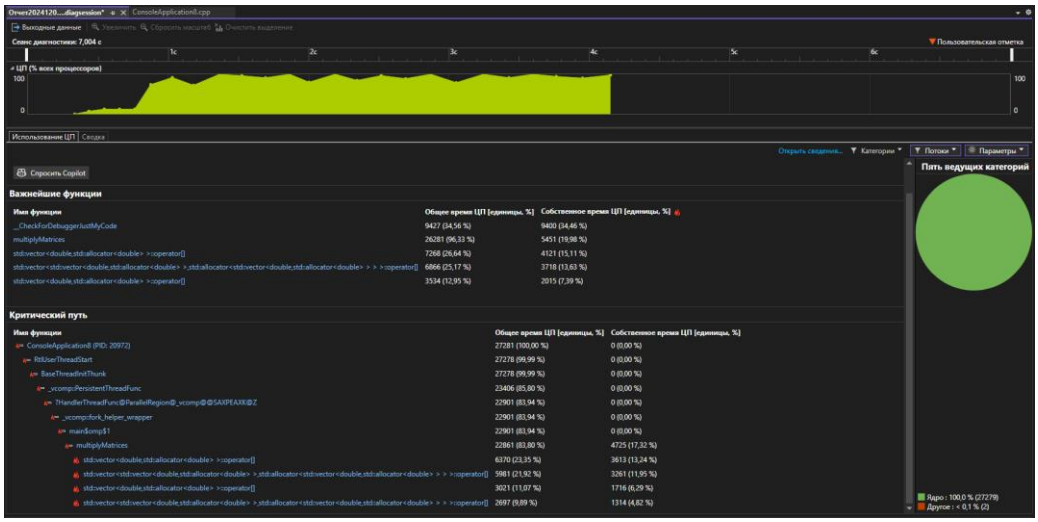
Лабораторная работа №6
“Тестирование производительности, профилирование приложений”
Перемножение N матриц размерности MxM.

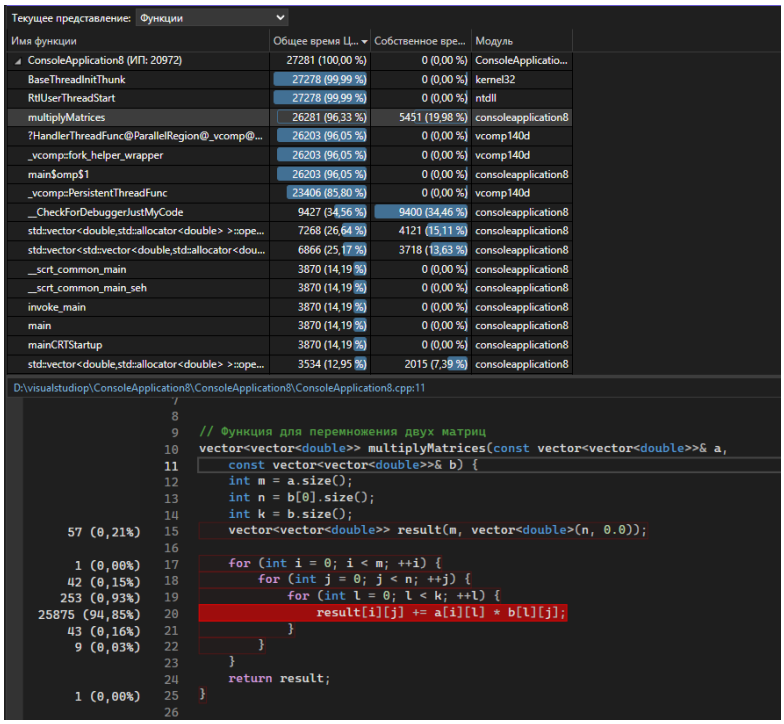
1. Написать многопоточную программу в соответствии с вариантом (количество потоков должно соответствовать возможному количеству потоков для вычислительной машины).

Код: https://pastebin.com/4PsqvLSk

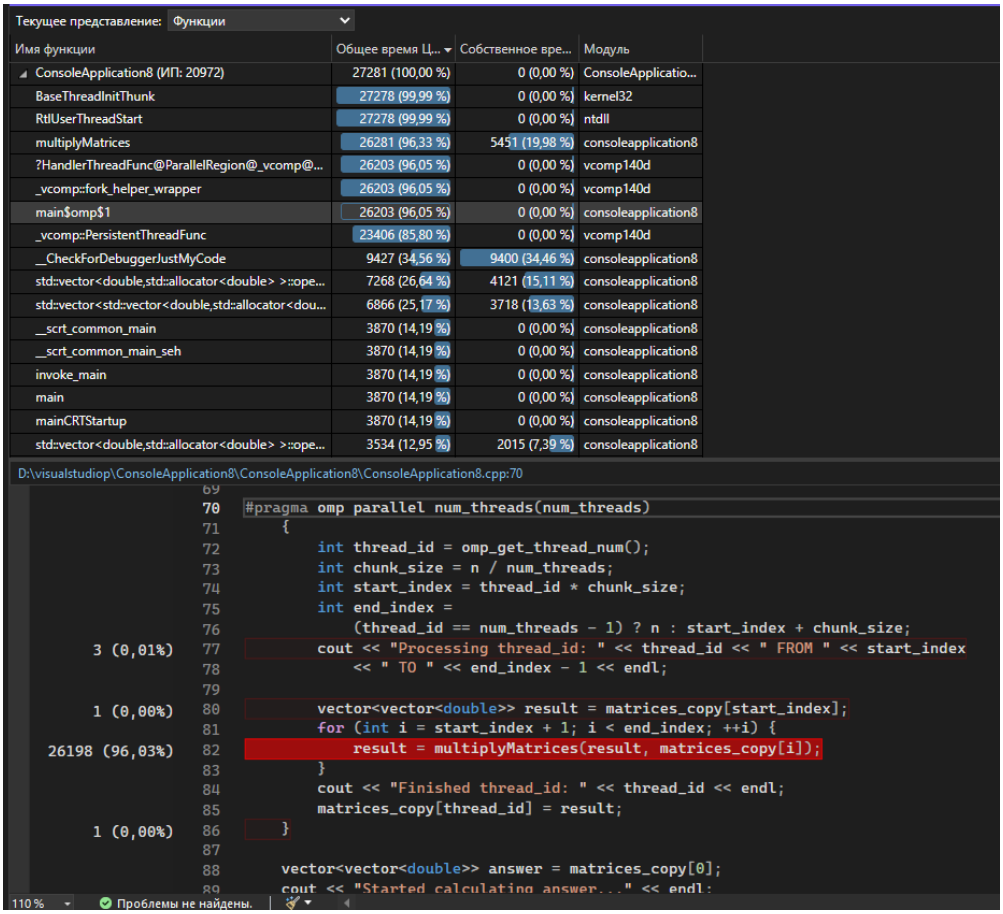
2. Провести профилирование с помощью дискретного профилировщика.



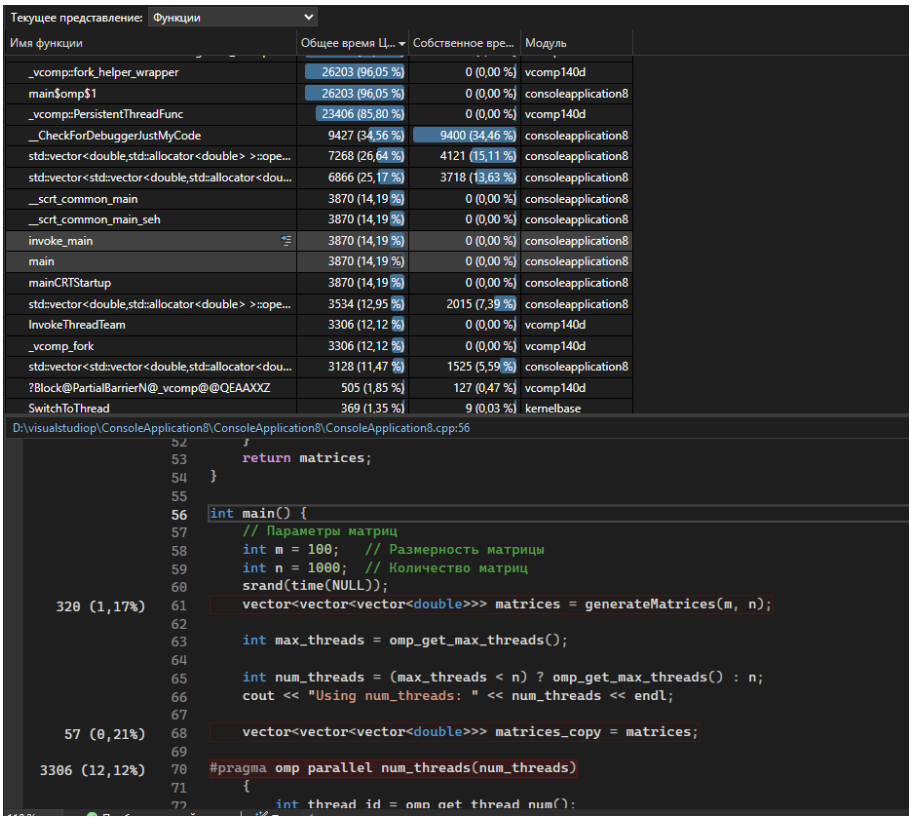
83% времени ЦП: функция multiplyMatrices.



Самое узкое место функции multiplyMatrices – расчёт ij значения.

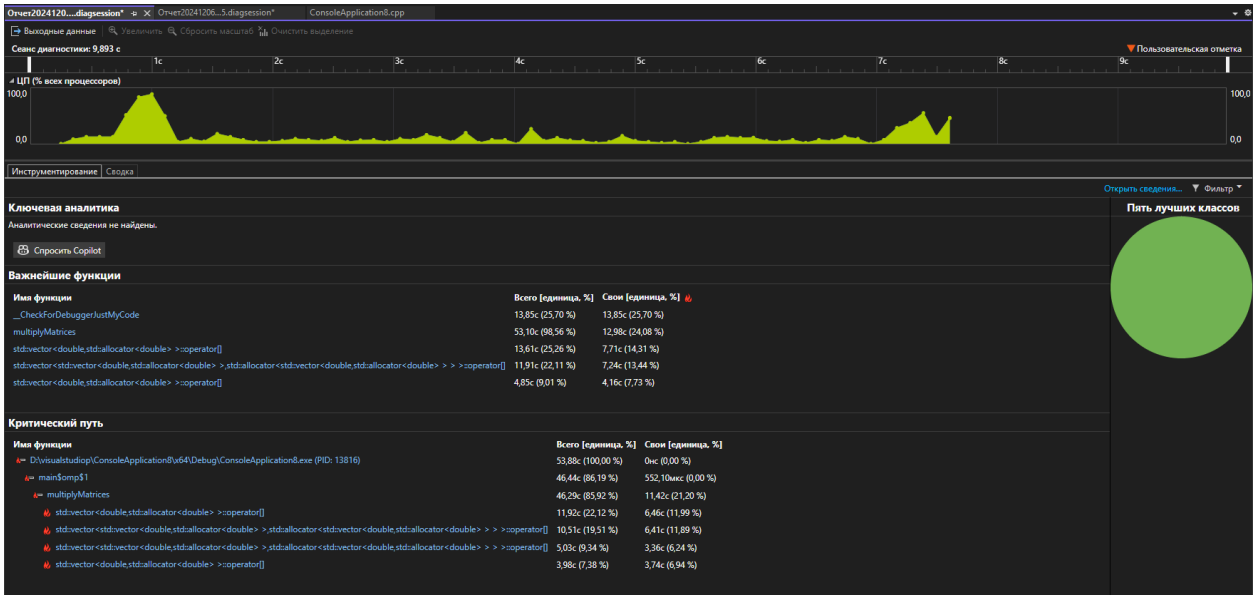


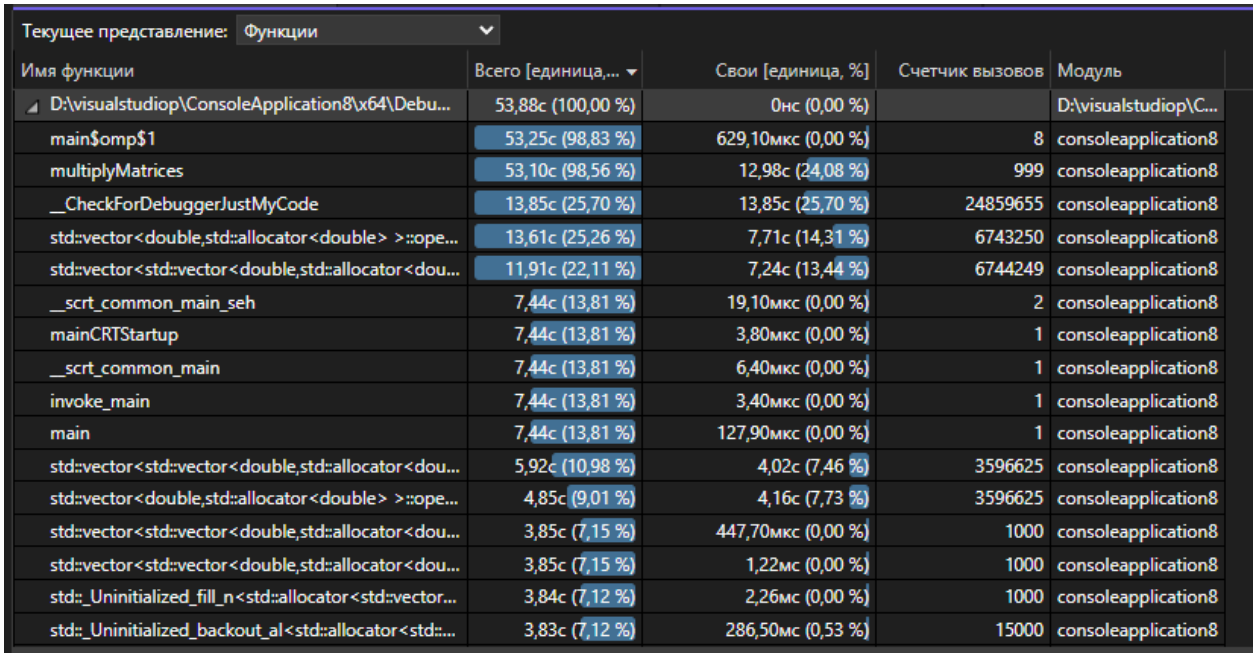
Самое узкое место omp – вызов функции multiplyMatrices.

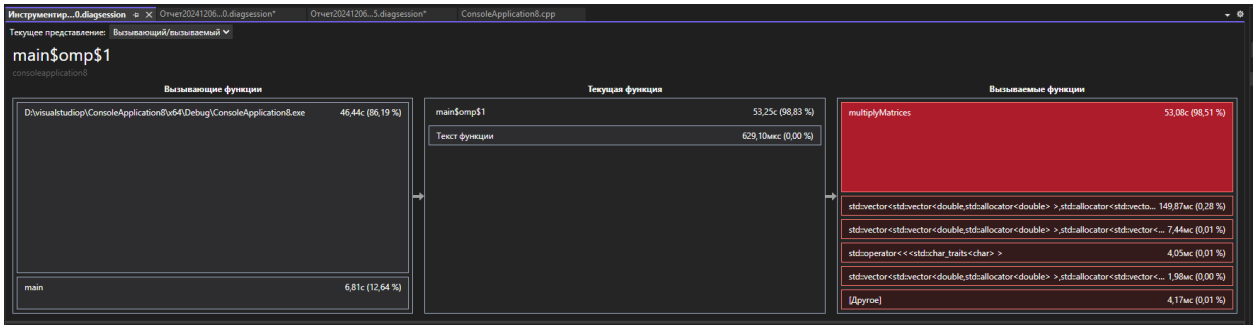


3. Провести профилирование с помощью инструментированного профилировщика.

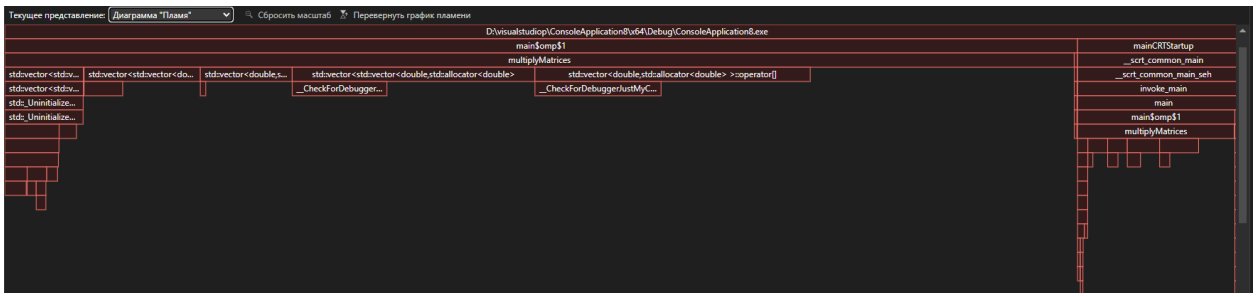
Рассчитано произведение 1000 матриц 15x15



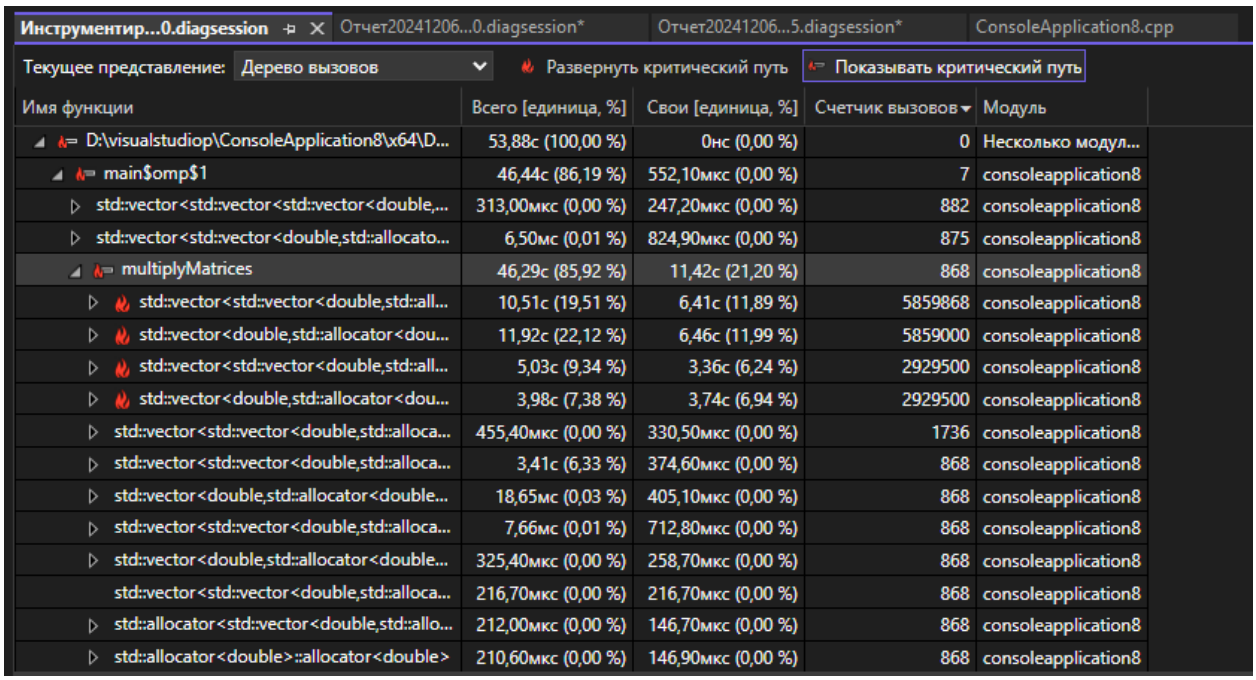




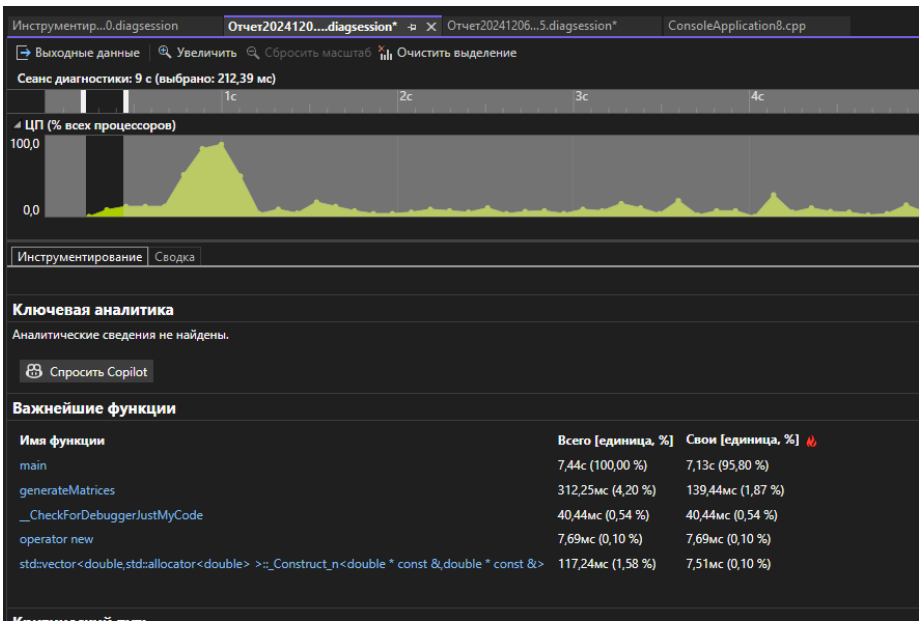
Убеждаемся, например, что функция multiplyMatrices вызывается 999 раз – 999 перемножений необходимо для расчёта 1000 матриц.



Как видно из результатов профилирования – больше всего времени в функциях omp, multiplyMatrices, обращениях по индексу в векторе.



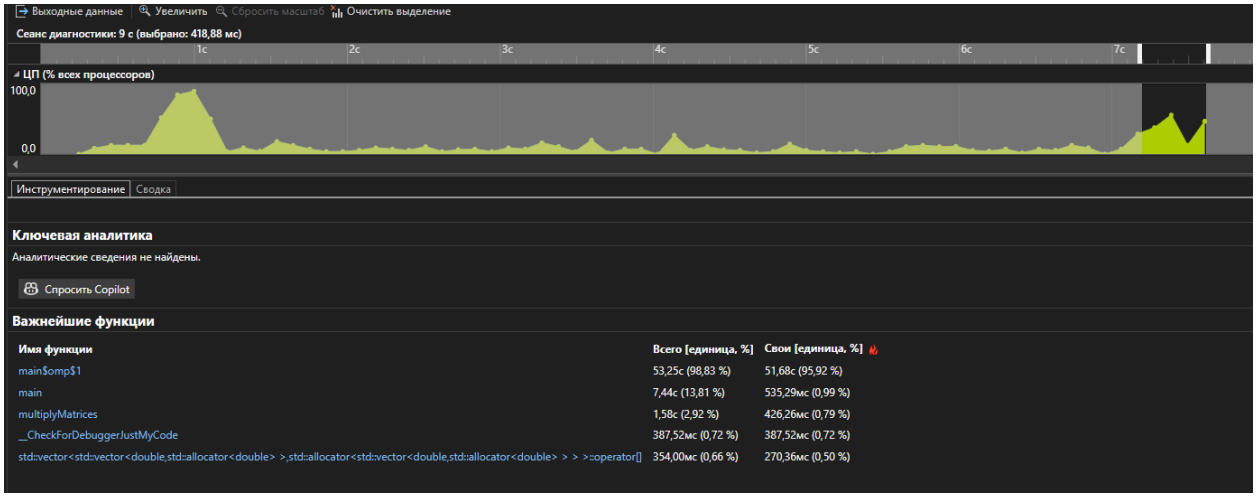
Имя функции	Всего [единица, %]	Свои [единица, %]	Счетчик вызовов	Модуль
D:\visualstudiop\ConsoleApplication8\x64\D...	53,88с (100,00 %)	0нс (0,00 %)	0	Несколько модул...
mainomp1	46,44с (86,19 %)	552,10мкс (0,00 %)	7	consoleapplication8
std::vector<std::vector<std::vector<double,...	313,00мкс (0,00 %)	247,20мкс (0,00 %)	882	consoleapplication8
std::vector<std::vector<double,std::allocato...	6,50мс (0,01 %)	824,90мкс (0,00 %)	875	consoleapplication8
multiplyMatrices	46,29с (85,92 %)	11,42с (21,20 %)	868	consoleapplication8
std::vector<std::vector<double,std::all...	10,51с (19,51 %)	6,41с (11,89 %)	5859868	consoleapplication8
std::vector<double,std::allocator<dou...	11,92с (22,12 %)	6,46с (11,99 %)	5859000	consoleapplication8
std::vector<std::vector<double,std::all...	5,03с (9,34 %)	3,36с (6,24 %)	2929500	consoleapplication8
std::vector<double,std::allocator<dou...	3,98с (7,38 %)	3,74с (6,94 %)	2929500	consoleapplication8
std::vector<std::vector<double,std::alloca...	455,40мкс (0,00 %)	330,50мкс (0,00 %)	1736	consoleapplication8
std::vector<std::vector<double,std::alloca...	3,41с (6,33 %)	374,60мкс (0,00 %)	868	consoleapplication8
std::vector<double,std::allocator<double...	18,65мс (0,03 %)	405,10мкс (0,00 %)	868	consoleapplication8
std::vector<std::vector<double,std::alloca...	7,66мс (0,01 %)	712,80мкс (0,00 %)	868	consoleapplication8
std::vector<double,std::allocator<double...	325,40мкс (0,00 %)	258,70мкс (0,00 %)	868	consoleapplication8
std::vector<std::vector<double,std::alloca...	216,70мкс (0,00 %)	216,70мкс (0,00 %)	868	consoleapplication8
std::allocator<std::vector<double,std::allo...	212,00мкс (0,00 %)	146,70мкс (0,00 %)	868	consoleapplication8
std::allocator<double>::allocator<double>	210,60мкс (0,00 %)	146,90мкс (0,00 %)	868	consoleapplication8



Имя функции	Всего [единица, %]	Свои [единица, %]
main	7,44с (100,00 %)	7,13с (95,80 %)
generateMatrices	312,25мс (4,20 %)	139,44мс (1,87 %)
__CheckForDebuggerJustMyCode	40,44мс (0,54 %)	40,44мс (0,54 %)
operator new	7,69мс (0,10 %)	7,69мс (0,10 %)
std::vector<double,std::allocator<double> >::_Construct_n<double * const &,double * const &>	117,24мс (1,58 %)	7,51мс (0,10 %)

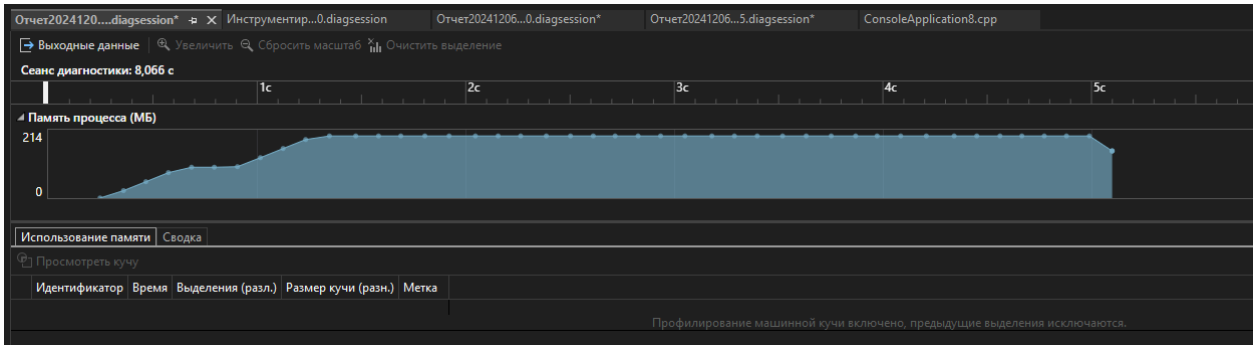
В начале работы – вызов функции генерации матриц.

Пиковая нагрузка – вычисление omp, multiplyMatrices, обращение к значению вектора.



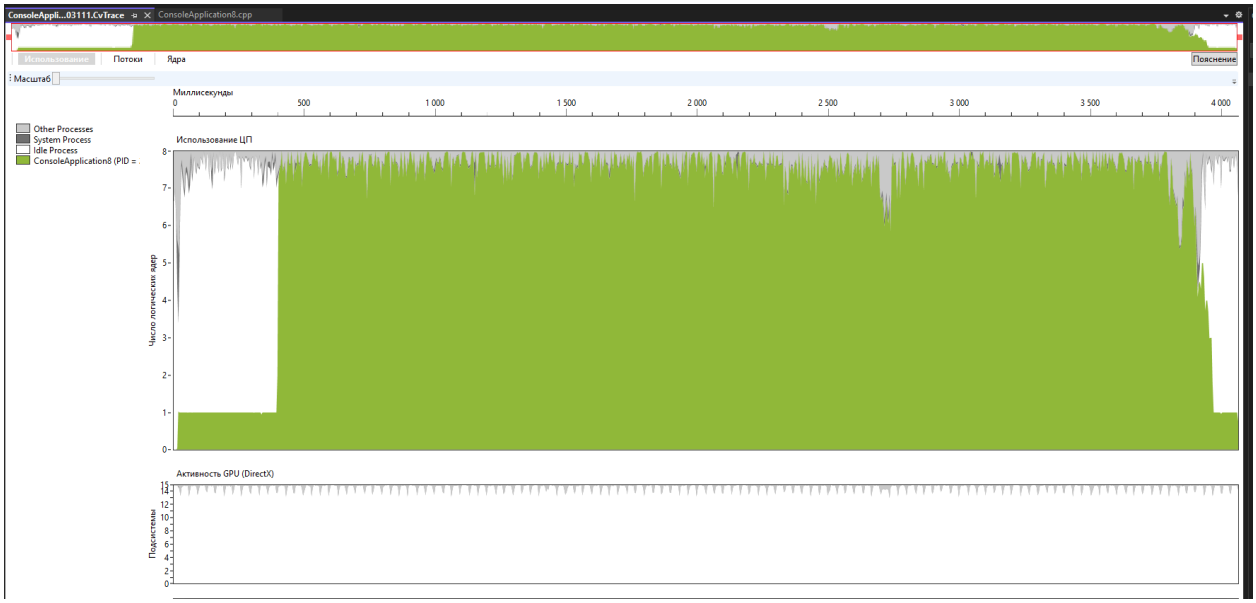
В конце – работа omp.

4. Провести профилирование с помощью профилировщика выделения памяти.

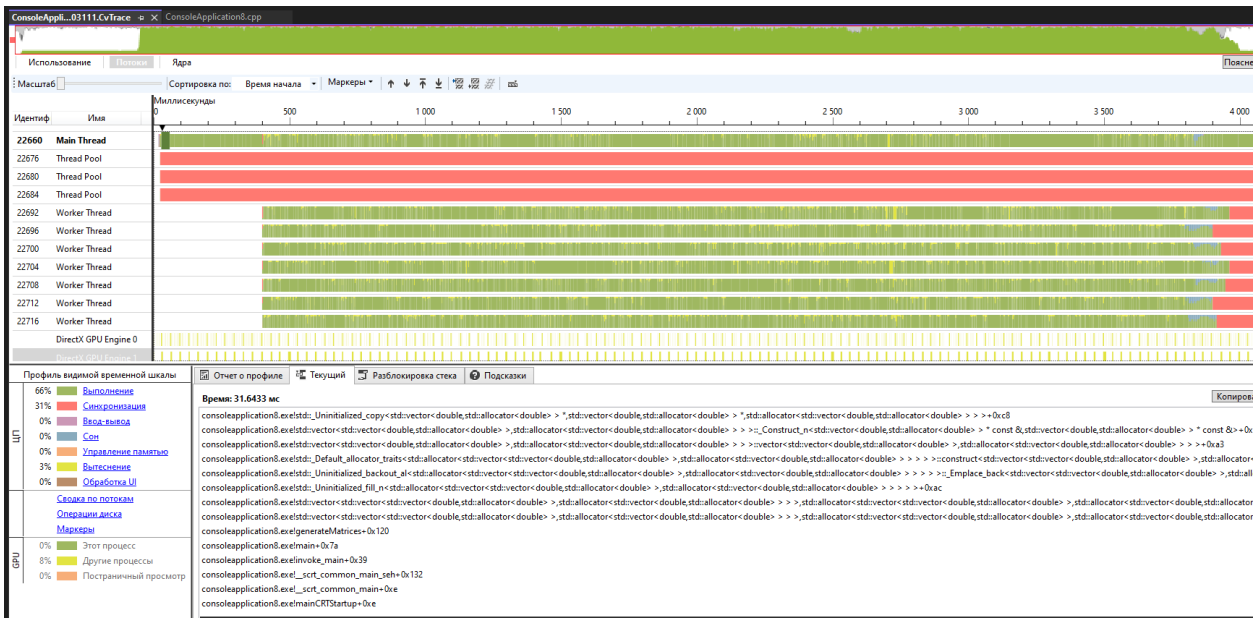


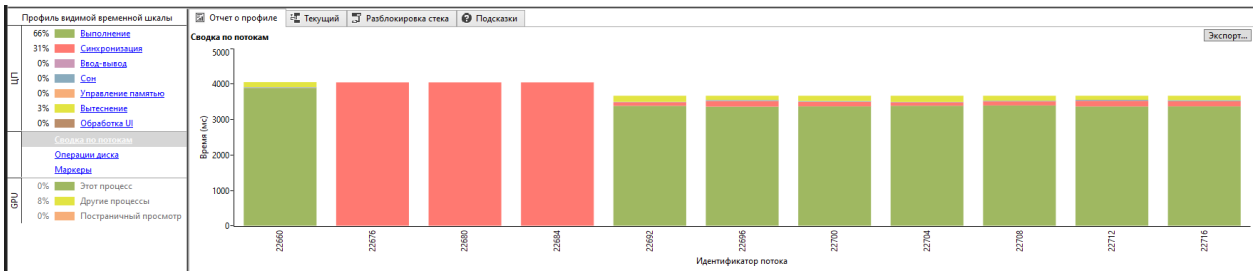
Память из кучи не выделялась.

5. Провести профилирование с помощью профилировщика конкуренции.

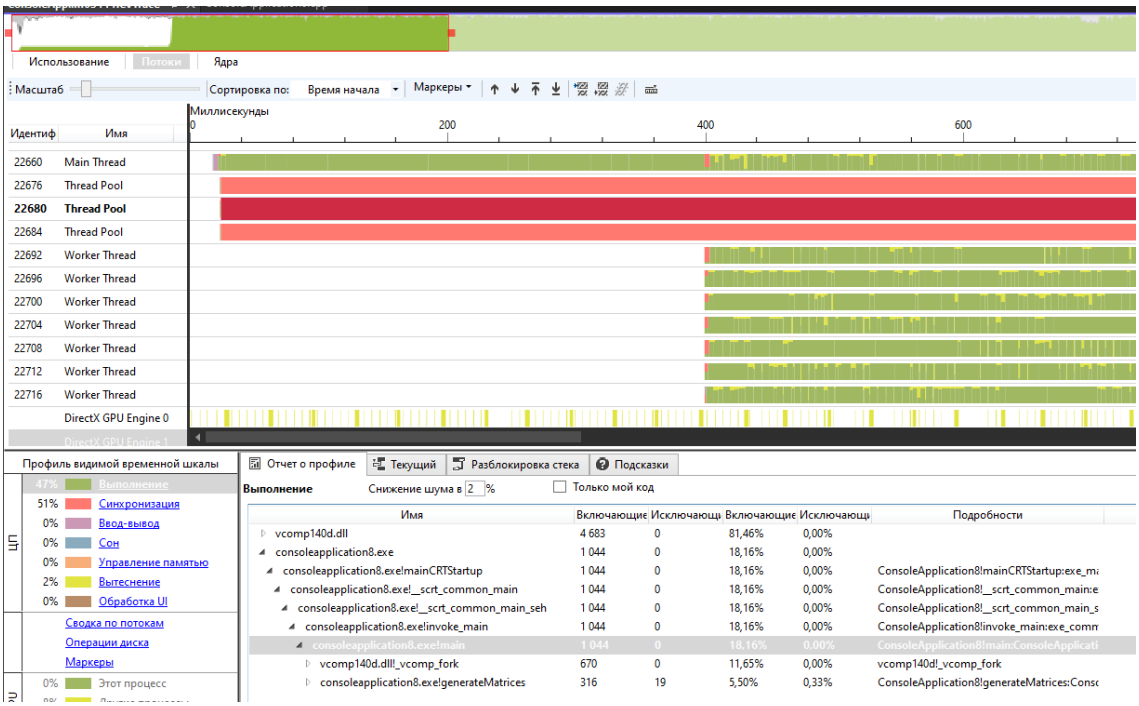


На графике хорошо видно момент начала и окончания работы многопоточного расчёта multiplyMatrices. До этого и после стартовый поток работает один.

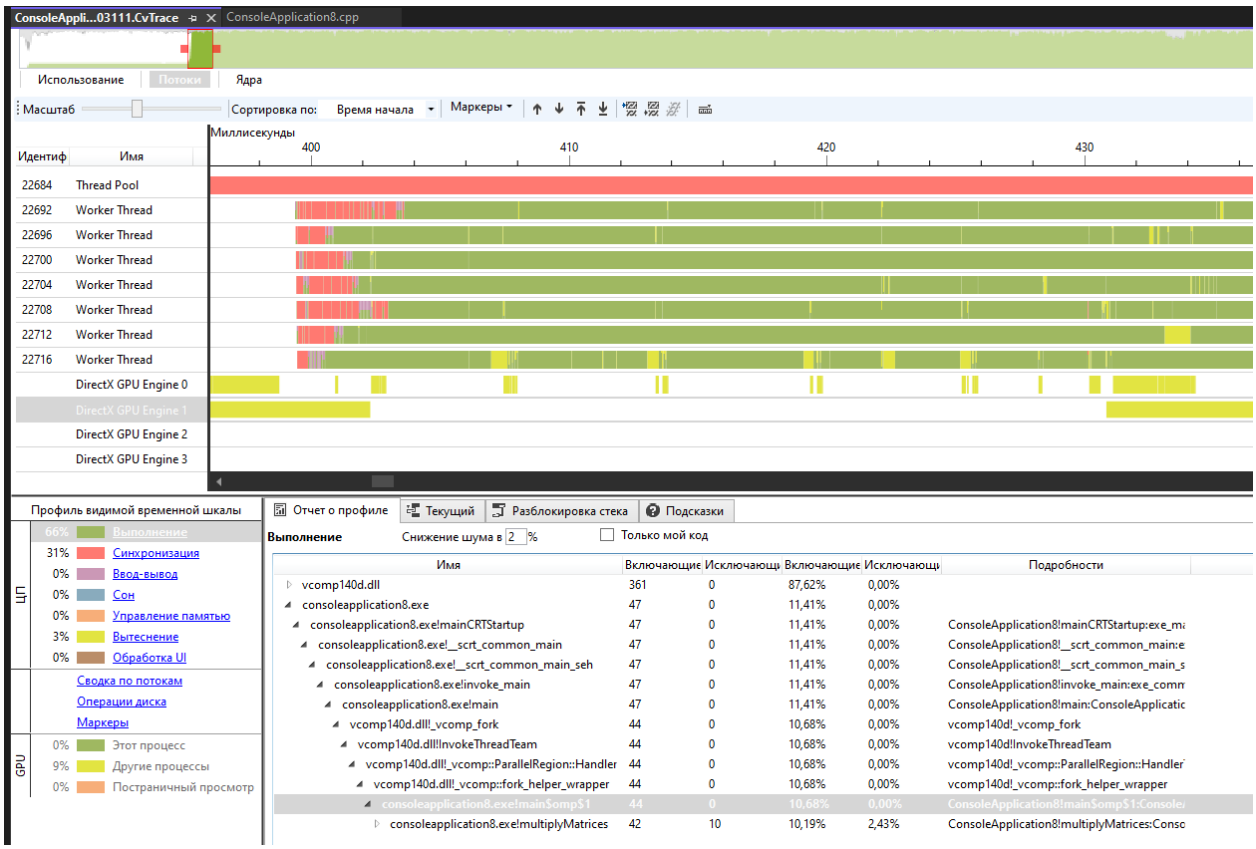




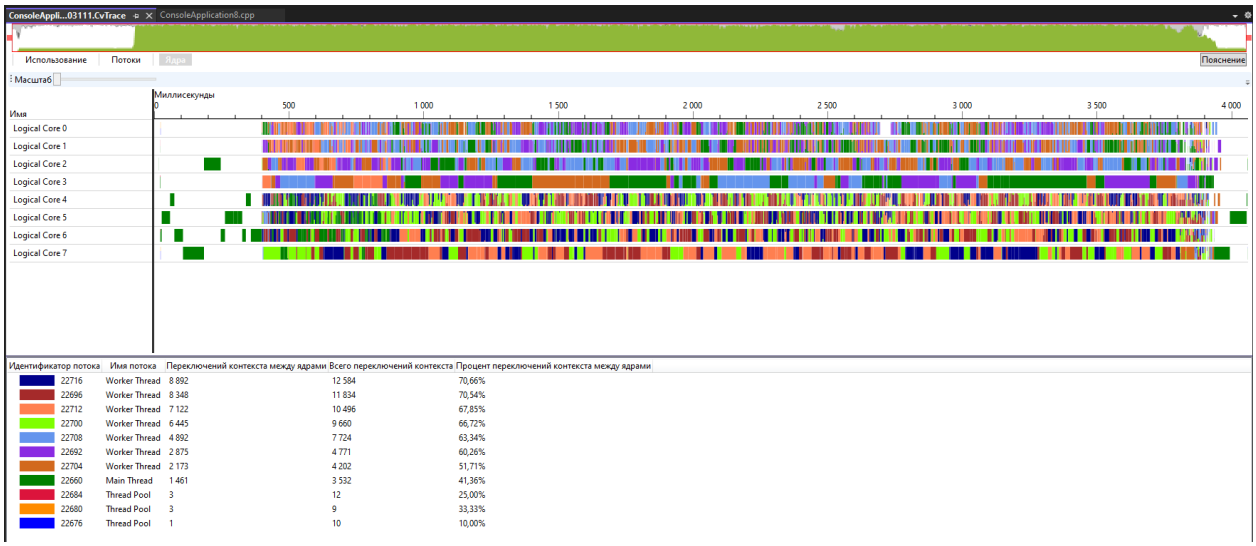
Аналогично, хорошо видно момент начала и окончания работы многопоточного расчёта multiplyMatrices. До этого и после стартовый поток работает один.



Убеждаемся, что стартовый поток генерирует матрицы в начале работы.



Убеждаемся, что в момент начала работы потоков вызывается функция multiplyMatrices.



Приложение. Код программы

```
#include <omp.h>


#include <chrono>
#include <iostream>
#include <vector>


using namespace std;
using namespace chrono;


// Функция для перемножения двух матриц
vector<vector<double>> multiplyMatrices(const vector<vector<double>>& a,
                                        const vector<vector<double>>& b) {
  int m = a.size();
  int n = b[0].size();
  int k = b.size();
  vector<vector<double>> result(m, vector<double>(n, 0.0));


  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) {
      for (int l = 0; l < k; ++l) {
        result[i][j] += a[i][l] * b[l][j];
      }
    }
  }
  return result;
}


void outputMatrix(vector<vector<double>> matrices_copy) {
  int m = matrices_copy.size();
  int n = matrices_copy[0].size();
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) {
      cout << matrices_copy[i][j] << " ";
    }
    cout << endl;
  }
  cout << endl;
}


float randomNumber(float Min, float Max) {
  return ((float(rand()) / float(RAND_MAX)) * (Max - Min)) + Min;
}


vector<vector<vector<double>>> generateMatrices(int m, int n) {
  vector<vector<vector<double>>> matrices(
```

```
      n, vector<vector<double>>(m, vector<double>(m)));
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      for (int k = 0; k < m; ++k) {
        matrices[i][j][k] = randomNumber(-1.0, 1.0);
      }
    }
  }
  return matrices;
}

int main() {
  // Параметры матриц
  int m = 100;   // Размерность матрицы
  int n = 1000;  // Количество матриц
  srand(time(NULL));
  vector<vector<vector<double>>> matrices = generateMatrices(m, n);

  int max_threads = omp_get_max_threads();

  int num_threads = (max_threads < n) ? omp_get_max_threads() : n;
  cout << "Using num_threads: " << num_threads << endl;

  vector<vector<vector<double>>> matrices_copy = matrices;

#pragma omp parallel num_threads(num_threads)
  {
    int thread_id = omp_get_thread_num();
    int chunk_size = n / num_threads;
    int start_index = thread_id * chunk_size;
    int end_index =
        (thread_id == num_threads - 1) ? n : start_index + chunk_size;
    cout << "Processing thread_id: " << thread_id << " FROM " << start_index
         << " TO " << end_index - 1 << endl;

    vector<vector<double>> result = matrices_copy[start_index];
    for (int i = start_index + 1; i < end_index; ++i) {
      result = multiplyMatrices(result, matrices_copy[i]);
    }
    cout << "Finished thread_id: " << thread_id << endl;
    matrices_copy[thread_id] = result;
  }

  vector<vector<double>> answer = matrices_copy[0];
  cout << "Started calculating answer from chu" << endl;
  for (int i = 1; i < n; ++i) {
```

```
    answer = multiplyMatrices(answer, matrices_copy[i]);
  }

  if (m <= 10) {
    cout << "\n\nAnswer:" << endl;
    outputMatrix(answer);
  } else {
    cout << "\n\Finished. Answer is not shown due to big matrices size."
         << endl;
  }

  return 0;
}
```