Теория параллелилзма

# Отчёт

# Уравнение теплопроводности

Выполнил Грищенко Александр Михайлович, 21932

03.2023

# 1 Цели работы

Реализовать решение уравнение теплопроводности (пятиточечный шаблон) в двумерной области на равномерных сетках.
Перенести программу на GPU используя директивы OpenACC.
Произвести профилирование программы и оптимизацию кода.
Сравнить время работы на CPU и GPU.

# 2 Используемый компилятор

pgc++

# 3 Используемый профилировщик

nsys (NVIDIA Nsight Systems) с OpenACC trace.

# 4 Как проводился замер времени работы

Для замера времени работы использовалась библиотека chrono.
Замер времени производился несколько раз, затем бралось среднее время.

# 5 Выполнение на CPU

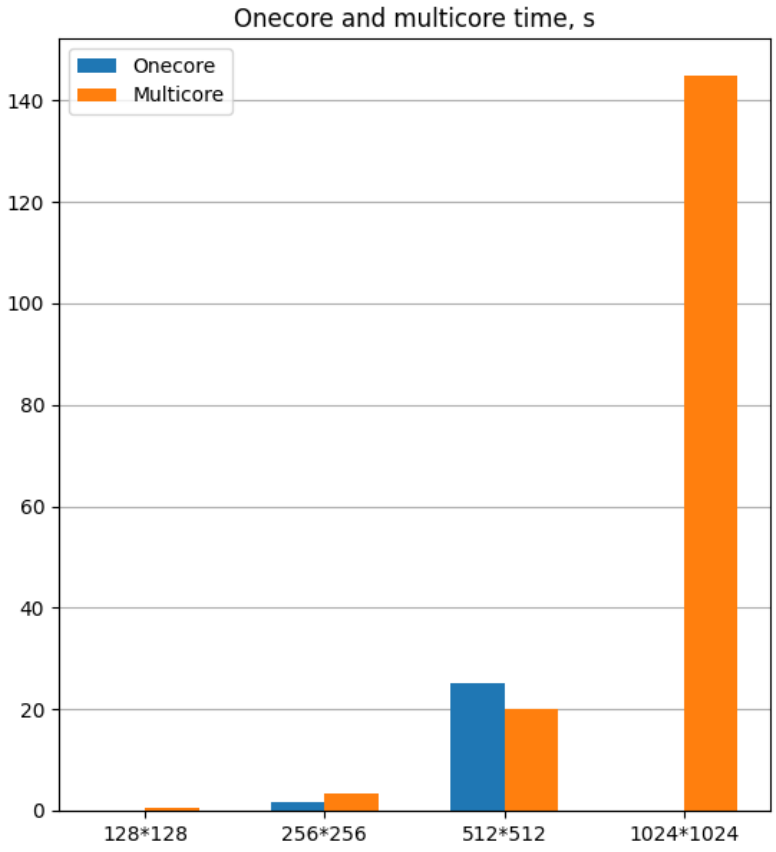
## 5.1 CPU-onecore

| Размер сетки | Время выполнения, с | Точность | Количество операций |
|---|---|---|---|
| 128*128 | 0.1 | 9.5e-07 | 11136 |
| 256*256 | 1.8 | 9.8e-07 | 37376 |
| 512*512 | 25 | 9.8e-07 | 120832 |

## 5.2 CPU-multicore

| Размер сетки | Время выполнения, с | Точность | Количество операций |
|---|---|---|---|
| 128*128 | 0.5 | 9.5e-07 | 11136 |
| 256*256 | 3.5 | 9.8e-07 | 37376 |
| 512*512 | 20 | 9.8e-07 | 120832 |
| 1024*1024 | 145 | 9.89e-07 | 365568 |

### 5.3 Диаграмма сравнения время работы CPU-onecore и CPU-multicore

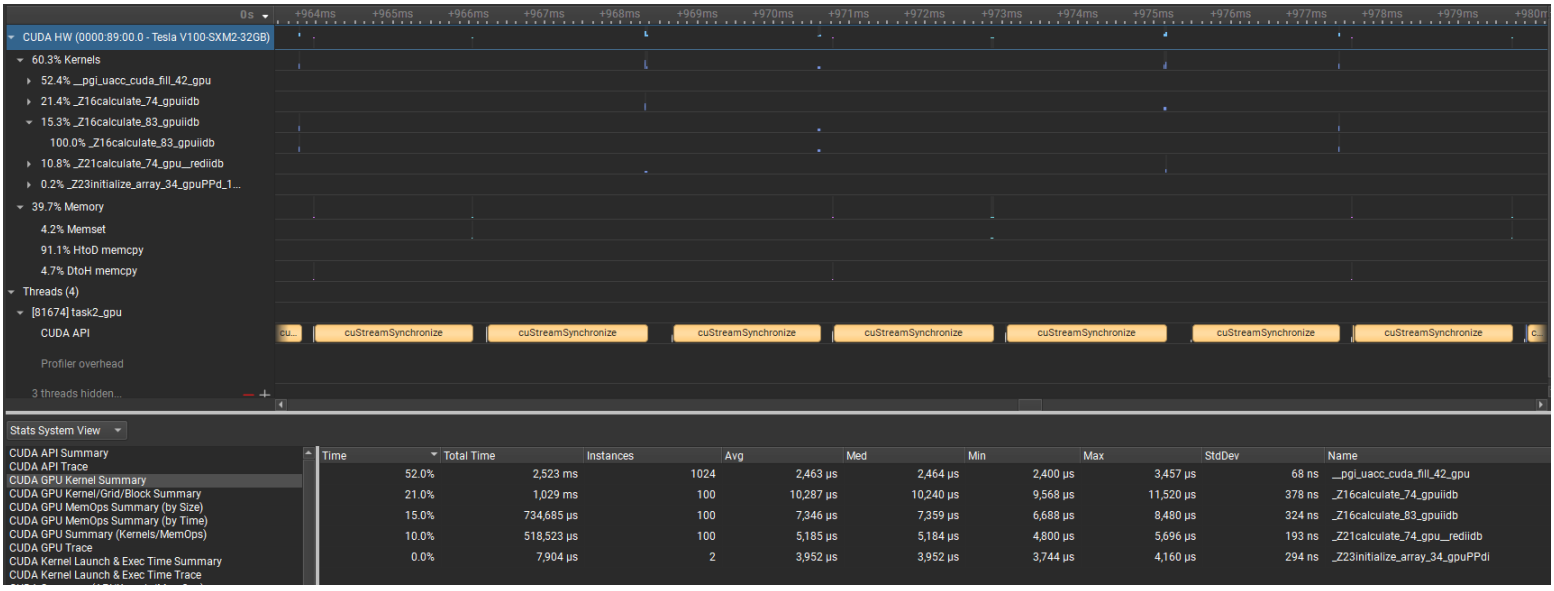


# 6 Выполнение на GPU

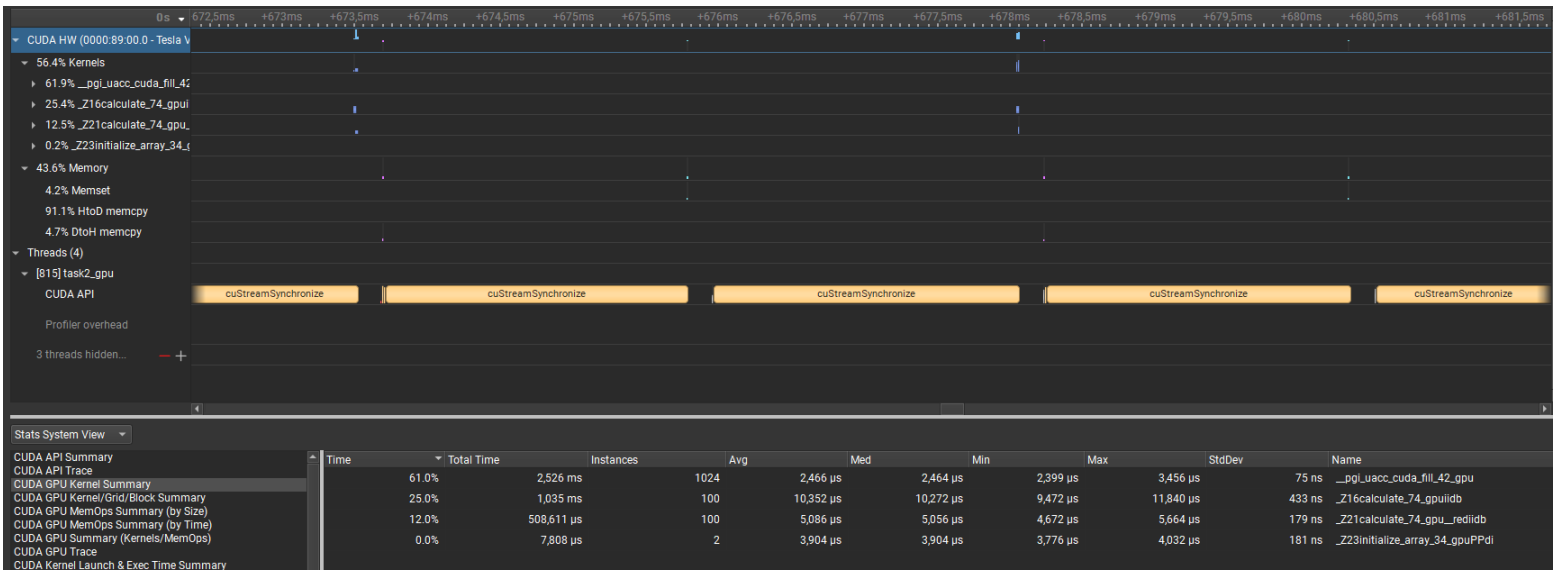
## 6.1 Этапы оптимизации на сетке 512*512 (количество итераций при профилировании 100)

| Этап № | Время выполнения, с | Точность | Количество операций | Комментарии (что было сделано) |
|---|---|---|---|---|
| 1 | 0.95 | 0.107 | 100 | Распараллелены циклы, reduction(max:error) |
| 2 | 0.73 | 0.107 | 100 | Замена копирования массива swap'ом через указатели |
| 3 | 0.7 | 0.035 | 100 | Изначальная инициализация массива значениями 20 |
| 4 | 0.5 | 0.036 | 100 | Асинхронный запуск ядер |
| 5 | 0.25 | N/A* | 100 | Подсчет ошибки не каждую итерацию |
| 6 | 0.23 | N/A* | 100 | Работа с матрицами происходит только на GPU. Операции с ошибкой стали периодическими и происходят в основном, на девайсе, ошибка обновляется на хосте только для проверки. |

* Из-за того, что период пересчёта ошибки намного меньше размера сетки, нет возможности определить точность на сотой итерации.




Этап 1




Этап 2

**Почему инициализация значениями 20?** Для начала посмотрим на финальный вывод (для простоты взята сетка 8x8):

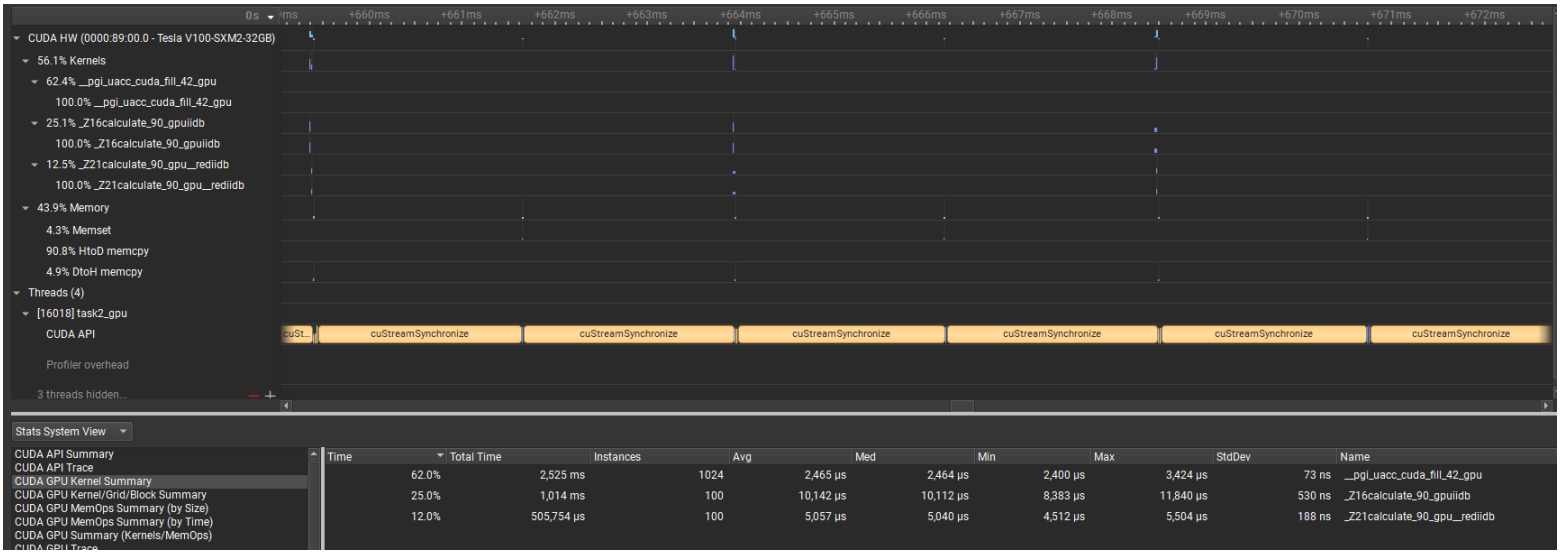
```
10      11.43   12.86   14.29   15.71   17.14   18.57   20
11.43   12.86   14.29   15.71   17.14   18.57   20      21.43
12.86   14.29   15.71   17.14   18.57   20      21.43   22.86
14.29   15.71   17.14   18.57   20      21.43   22.86   24.29
15.71   17.14   18.57   20      21.43   22.86   24.29   25.71
17.14   18.57   20      21.43   22.86   24.29   25.71   27.14
18.57   20      21.43   22.86   24.29   25.71   27.14   28.57
20      21.43   22.86   24.29   25.71   27.14   28.57   30
```

Как можно заметить значения лежат в отрезке [10, 30], значит, имеет смысл изначально инициализировать массив значениями из этого диапазона. Также стоит отметить, что модой в матрице является число 20 (а ещё и средним арифметическим границ отрезка). Значит изначальная инициализация массива 20-ю существенно приблизит ответ к правильному, ещё до запуска основного алгоритма.

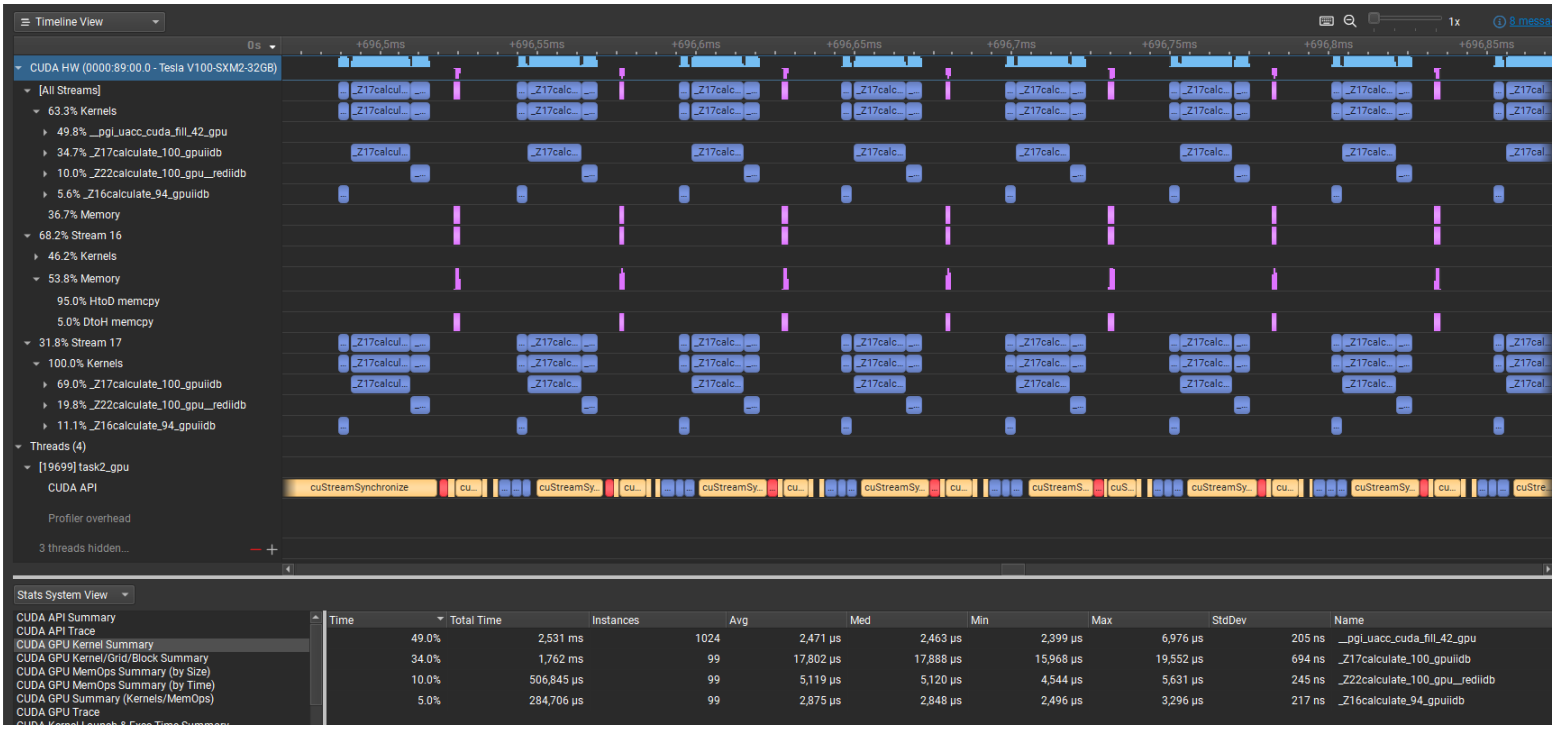
Сравнение поведения программы на сетке 128x128.

| Инициализация | Время выполнения, с | Точность | Количество операций |
|---|---|---|---|
| Нет | 1.1 | 1e-06 | 30074 |
| Да | 0.4 | 9.995e-07 | 11073 |

Как видно в таблице, изначальная инициализация дает хороший прирост производительности.



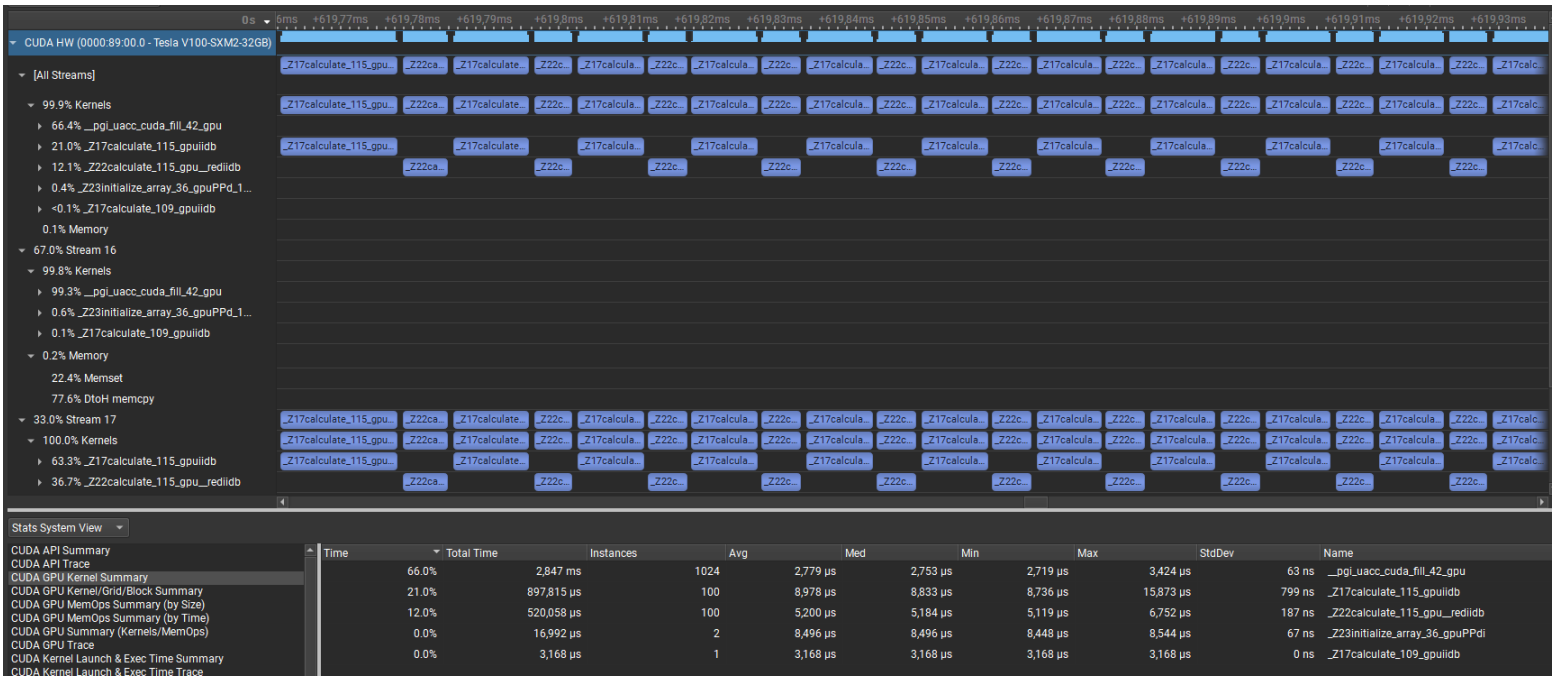

Этап 3




Этап 4

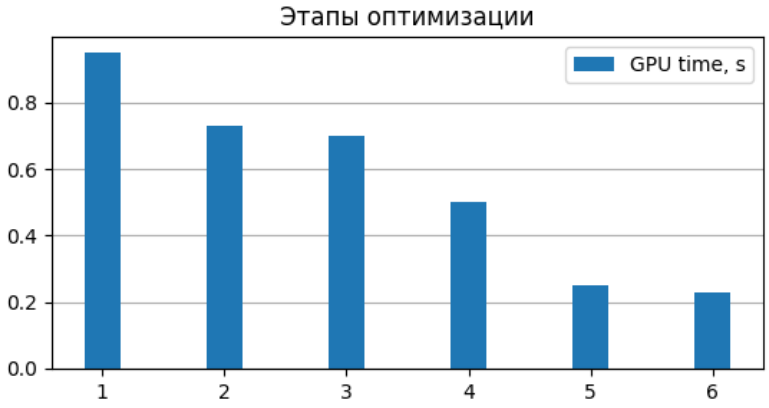

CUDA HW (0000:89:00.0 - Tesla V100-SXM2-32GB)
[All Streams]
58.5% Kernels
67.0% __pgi_uacc_cuda_fill_42_gpu
20.9% _Z17calculate_108_gpuiidb
12.2% _Z22calculate_108_gpu__rediidb
41.5% Memory
80.7% Stream 16
48.6% Kernels
51.4% Memory
>99.9% HtoD memcpy
<0.1% DtoH memcpy
19.3% Stream 17
100.0% Kernels
63.2% _Z17calculate_108_gpuiidb
36.8% _Z22calculate_108_gpu__rediidb
Threads (4)
Stats System View
CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Summary
CUDA GPU MemOps Summary (by Size)
CUDA GPU MemOps Summary (by Time)
CUDA GPU Summary (Kernels/MemOps)
Time
Total Time
Instances
Avg
Med
Min
Max
StdDev
Name
67.0% 2,857 ms 1024 2,790 µs 2,752 µs 2,719 µs 14,016 µs 406 ns __pgi_uacc_cuda_fill_42_gpu
20.0% 889,404 µs 100 8,894 µs 8,832 µs 8,736 µs 14,400 µs 559 ns _Z17calculate_108_gpuiidb
12.0% 518,716 µs 100 5,187 µs 5,184 µs 5,151 µs 6,272 µs 118 ns _Z22calculate_108_gpu__rediidb

Этап 5


CUDA HW (0000:89:00.0 - Tesla V100-SXM2-32GB)
[All Streams]
99.9% Kernels
66.4% __pgi_uacc_cuda_fill_42_gpu
21.0% _Z17calculate_115_gpuiidb
12.1% _Z22calculate_115_gpu__rediidb
0.4% _Z23initialize_array_36_gpuPPd_1...
<0.1% _Z17calculate_109_gpuiidb
0.1% Memory
67.0% Stream 16
99.8% Kernels
99.3% __pgi_uacc_cuda_fill_42_gpu
0.6% _Z23initialize_array_36_gpuPPd_1...
0.1% _Z17calculate_109_gpuiidb
0.2% Memory
22.4% Memset
77.6% DtoH memcpy
33.0% Stream 17
100.0% Kernels
63.3% _Z17calculate_115_gpuiidb
36.7% _Z22calculate_115_gpu__rediidb
Stats System View
CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Summary
CUDA GPU MemOps Summary (by Size)
CUDA GPU MemOps Summary (by Time)
CUDA GPU Summary (Kernels/MemOps)
CUDA GPU Trace
CUDA Kernel Launch & Exec Time Summary
CUDA Kernel Launch & Exec Time Trace
Time
Total Time
Instances
Avg
Med
Min
Max
StdDev
Name
66.0% 2,847 ms 1024 2,779 µs 2,753 µs 2,719 µs 3,424 µs 63 ns __pgi_uacc_cuda_fill_42_gpu
21.0% 897,815 µs 100 8,978 µs 8,833 µs 8,736 µs 15,873 µs 799 ns _Z17calculate_115_gpuiidb
12.0% 520,058 µs 100 5,200 µs 5,184 µs 5,119 µs 6,752 µs 187 ns _Z22calculate_115_gpu__rediidb
0.0% 16,992 µs 2 8,496 µs 8,496 µs 8,448 µs 8,544 µs 67 ns _Z23initialize_array_36_gpuPPdi
0.0% 3,168 µs 1 3,168 µs 3,168 µs 3,168 µs 3,168 µs 0 ns _Z17calculate_109_gpuiidb

Этап 6

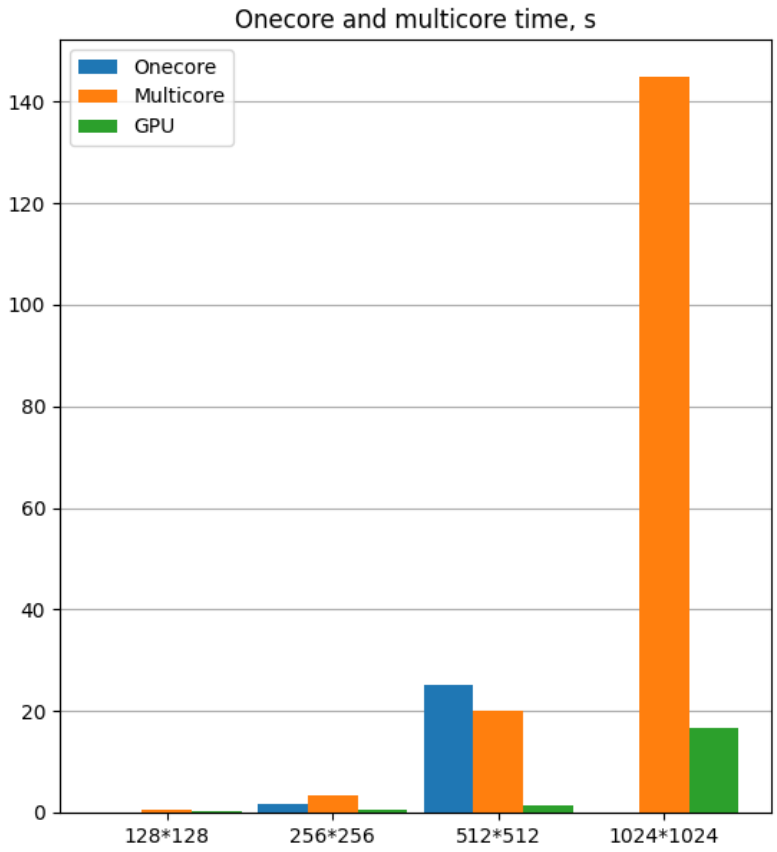
### 6.2 Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)




### 6.3 GPU – оптимизированный вариант

| Размер сетки | Время выполнения, с | Точность | Количество опреаций |
|---|---|---|---|
| 128*128 | 0.3 | 9.5e-07 | 11136 |
| 256*256 | 0.5 | 9.8e-07 | 37376 |
| 512*512 | 1.5 | 9.8e-07 | 120832 |
| 1024*1024 | 16.7 | 9.9e-07 | 365568 |

# 7 Диаграмма сравнения времени работы CPU-one, CPU-multi, GPU (оптимизированный вариант) для разных размеров сеток




# 8 Вывод

При грамотном использовании ресурсов GPU можно достичь ускорения работы программы в десятки раз. GPU все еще уступает CPU в маленьких сетках (<= 128), однако отставание некритично.

# 9 Приложение

## 9.1 Ссылка на GitHub

https://github.com/busyhedg03/ParallelismTheory/tree/master/task_2

```cpp
#include <chrono>
#include <cmath>
#include <iostream>

#ifdef _FLOAT
    #define T float
    #define MAX std::fmaxf
    #define STOD std::stof
#else
    #define T double
    #define MAX std::fmax
    #define STOD std::stod
#endif

// Вывести значения двумерного массива
void print_array(T **A, int size)
{
    for (int i = 0; i < size; ++i)
    {
        for (int j = 0; j < size; ++j)
        {
            // Значение с GPU
            #pragma acc kernels present(A)
            printf("%.2f\t", A[i][j]);
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

```

```cpp
// Инициализация матрицы, чтобы подготовить ее к основному алгоритму
void initialize_array(T **A, int size)
{
    // Инициализируется матрица на GPU
    #pragma acc parallel present(A)
    {
        // Заполнение углов матрицы значениями
        A[0][0] = 10.0;
        A[0][size - 1] = 20.0;
        A[size - 1][size - 1] = 30.0;
        A[size - 1][0] = 20.0;

        // Заполнение периметра матрицы
        T step = 10.0 / (size - 1);
        #pragma acc loop independent
        for (int i = 1; i < size - 1; ++i)
        {
            T addend = step * i;
            A[0][i] = A[0][0] + addend;               // horizontal
            A[size - 1][i] = A[size - 1][0] + addend; // horizontal
            A[i][0] = A[0][0] + addend;               // vertical
            A[i][size - 1] = A[0][size - 1] + addend; // vertical
        }

        // Заполнение 20-ю, чтобы сократить количество операций в основном алгоритме
        #pragma acc loop independent collapse(2)
        for (int i = 1; i < size - 1; ++i)
            for (int j = 1; j < size - 1; ++j)
                A[i][j] = 20.0;
    }
}

// Очистка динамической памяти, выделенной под двумерую матрицу
void delete_2d_array(T **A, int size)
{
    for (int i = 0; i < size; i++)
        delete[] A[i];
    delete[] A;
}

// Основной алгоритм
void calculate(int net_size = 128, int iter_max = 1e6, T accuracy = 1e-6, bool res = false)
{
    // Создание 2-х двумерных матриц, одна будет считаться на основе другой
    T **Anew = new T *[net_size],
      **A = new T *[net_size];
    for (int i = 0; i < net_size; i++)
    {
        A[i] = new T[net_size];
        Anew[i] = new T[net_size];
    }

    #pragma acc enter data create(A[:net_size][:net_size], Anew[:net_size][:net_size])

    // Инициализация матриц
    initialize_array(A, net_size);
    initialize_array(Anew, net_size);

    // Текущая ошибка
    T error = 0;
    // Счетчик итераций
    int iter;
    // Указатель для swap
    T **temp;
    // Флаг обновления ошибки на хосте для обработки условием цикла
    bool update_flag = true;

```

```
    #pragma acc data copy(error)
    {
        for (iter = 0; iter < iter_max; ++iter)
        {
            // Сокращение количества обращений к CPU. Больше сетка - реже стоит сверять значения.
            update_flag = !(iter % net_size);

            if (update_flag)
            {
                // зануление ошибки на GPU
                #pragma acc kernels
                error = 0;
            }

            // Распараллелить циклы с редукцией и запустить асинхронные ядра
            #pragma acc kernels loop independent collapse(2) reduction(max : error) present(A, Anew) async(1)
            for (int i = 1; i < net_size - 1; i++)
                for (int j = 1; j < net_size - 1; j++)
                {
                    Anew[i][j] = (A[i + 1][j] + A[i - 1][j] + A[i][j - 1] + A[i][j + 1]) * 0.25;
                    // Пересчитать ошибку
                    if (update_flag)
                        error = std::max(error, std::abs(Anew[i][j] - A[i][j]));
                }

            // swap(A, Anew)
            temp = A, A = Anew, Anew = temp;
            // Проверить ошибку
            if (update_flag)
            {
                // Синхронизация и обновление ошибки на хосте
                #pragma acc update host(error) wait(1)
                // Если ошибка не превышает точность, прекратить выполнение цикла
                if (error <= accuracy)
                    break;
            }
        }
        // Синхронизация
        #pragma acc wait(1)
    }
    std::cout.precision(2);
    if (res)
        print_array(A, net_size);
    std::cout << "iter=" << iter << ",\terror=" << error << std::endl;

    #pragma acc exit data delete (A[:net_size][:net_size], Anew[:net_size][:net_size])
    delete_2d_array(A, net_size);
    delete_2d_array(Anew, net_size);
}
```

```cpp

int main(int argc, char *argv[])
{
    auto begin_main = std::chrono::steady_clock::now();
    int net_size = 128, iter_max = (int)1e6;
    T accuracy = 1e-6;
    bool res = false;
    for (int arg = 1; arg < argc; arg++)
    {
        std::string str = argv[arg];
        if (!str.compare("-res"))
            res = true;
        else
        {
            arg++;
            if (!str.compare("-a"))
                accuracy = STOD(argv[arg]);
            else if (!str.compare("-i"))
                iter_max = (int)std::stod(argv[arg]);
            else if (!str.compare("-s"))
                net_size = std::stoi(argv[arg]);
            else
            {
                std::cout << "Wrong args!";
                return -1;
            }
        }
    }
    calculate(net_size, iter_max, accuracy, res);
    auto end_main = std::chrono::steady_clock::now();
    int time_spent_main = std::chrono::duration_cast<std::chrono::milliseconds>(end_main - begin_main).count();
    std::cout << "The elapsed time is:\nmain\t\t\t" << time_spent_main << " ms\n";
    return 0;
}
```