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

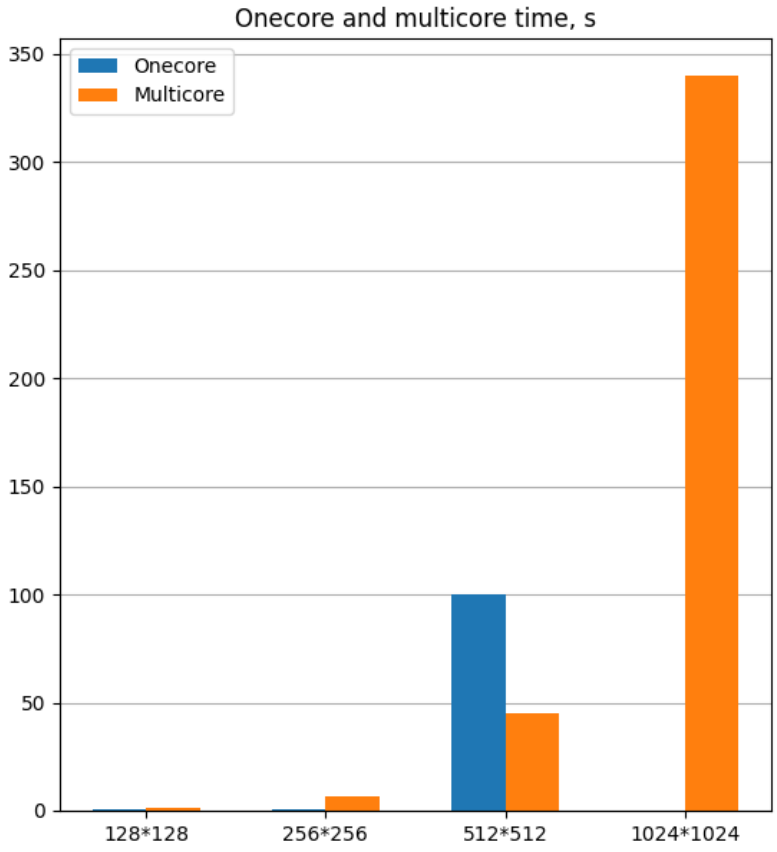
5.1 CPU-onecore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	0.5	9.52e-07	11136
256*256	0.67	9.85e-07	37376
512*512	100	9.77e-07	120832

5.2 CPU-multicore

Размер сетки	Время выполнения, с	Точность	Количество операций
128*128	1.1	9.52e-07	11136
256*256	6.5	9.85e-07	37376
512*512	45	9.78e-07	120832
1024*1024	340	9.89e-07	365568

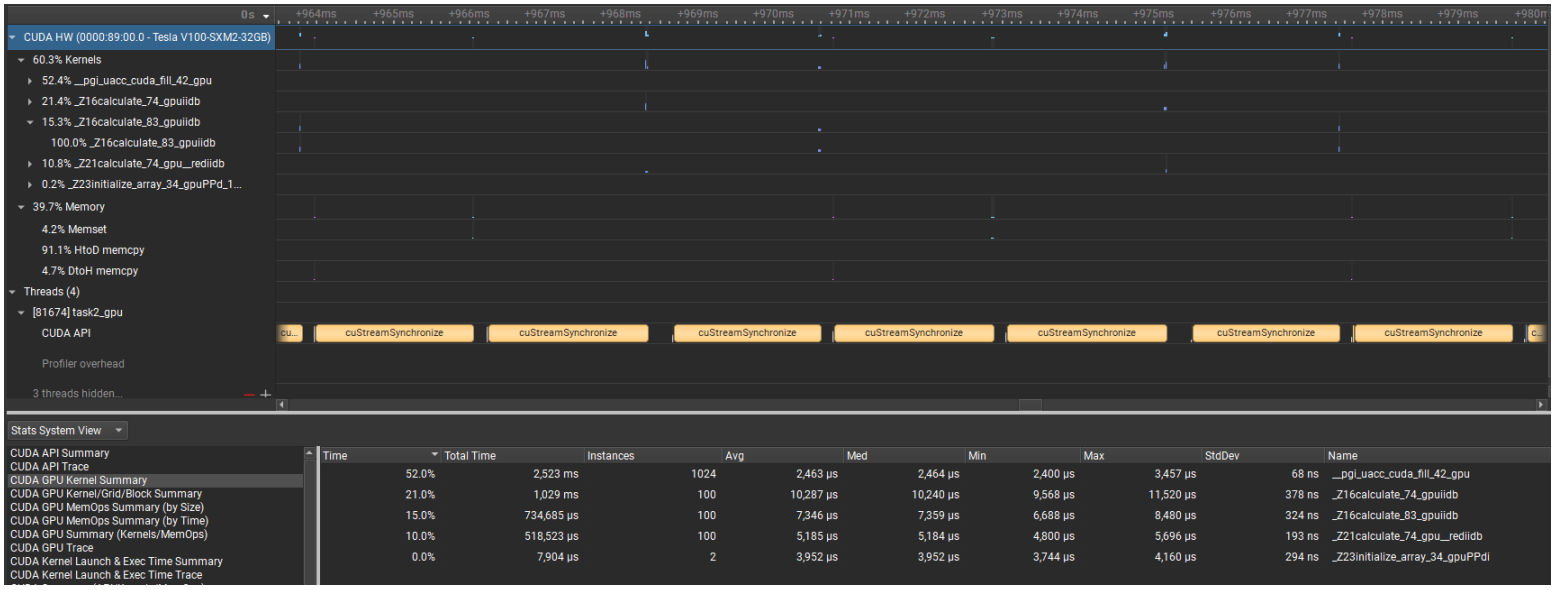
5.3 Диаграмма сравнения время работы CPU-onecore и CPU-multicore



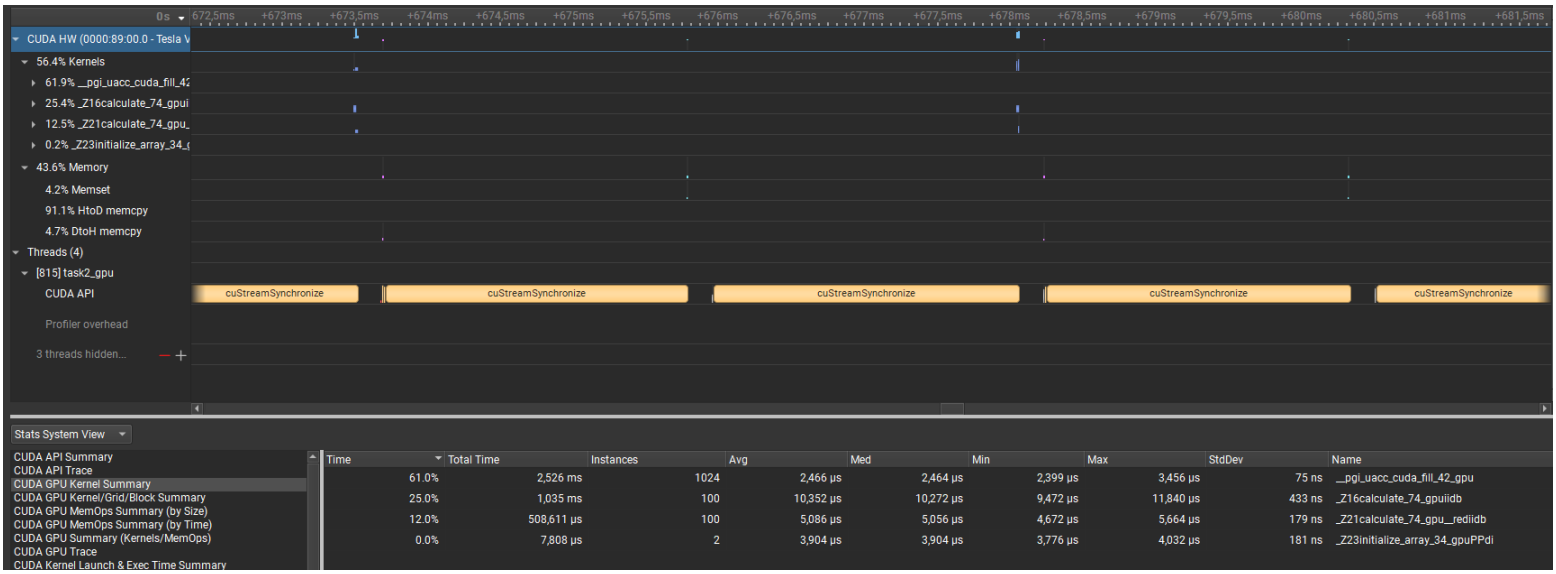
6 Выполнение на GPU

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



Этап 1



Этап 2

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

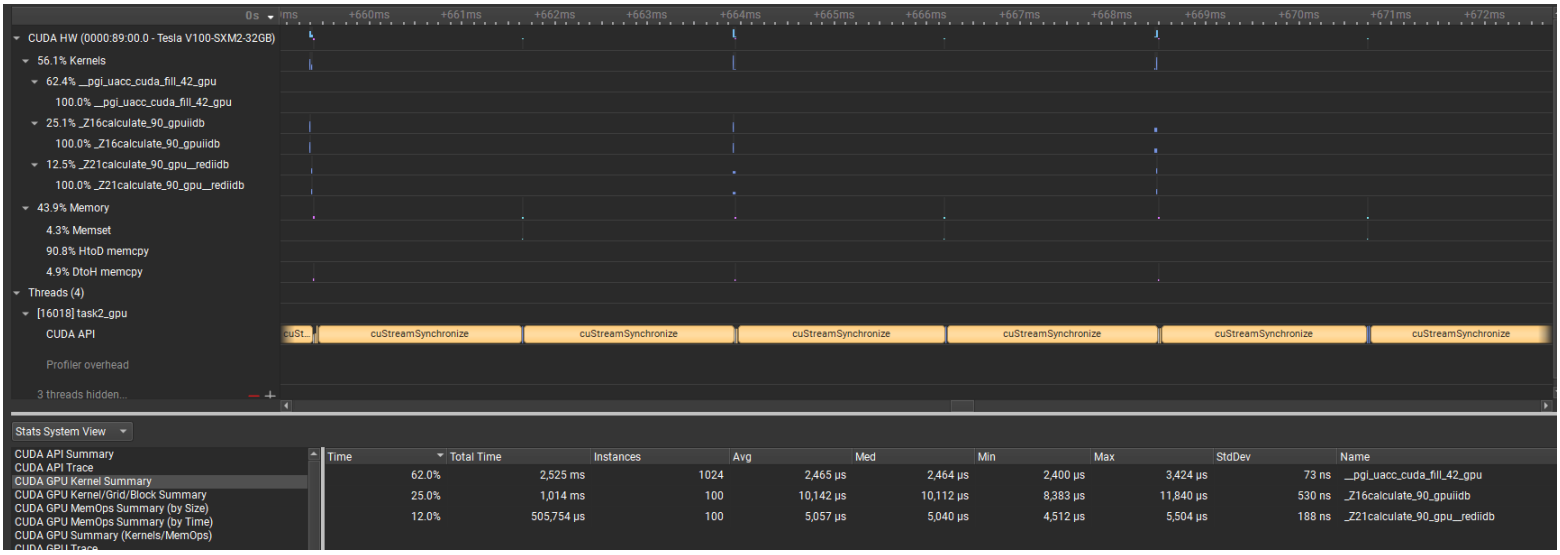
```
10      11.43   12.86   14.29   15.71   17.14   18.57   20
11.43   12.86   14.29   15.71   17.14   18.57   20      21.43
12.86   14.29   15.71   17.14   18.57   20      21.43   22.86
14.29   15.71   17.14   18.57   20      21.43   22.86   24.29
15.71   17.14   18.57   20      21.43   22.86   24.29   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.

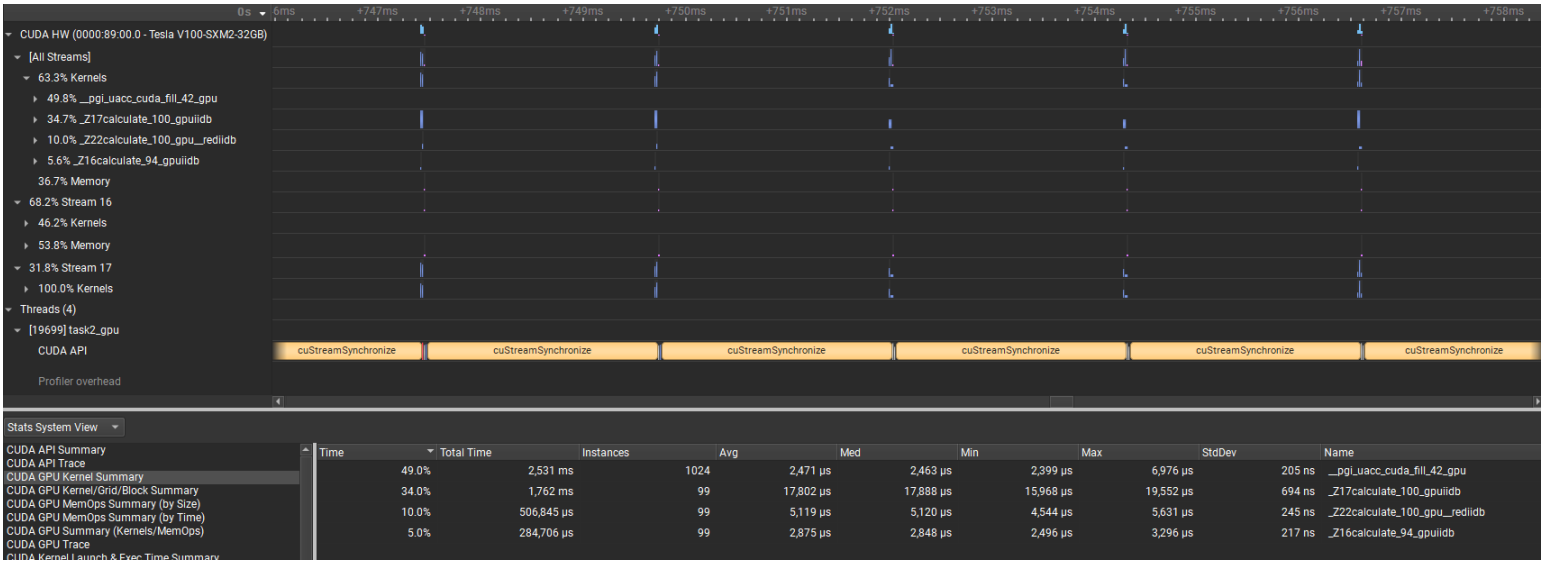
Сравнение поведения программы на сетке 128x128.

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

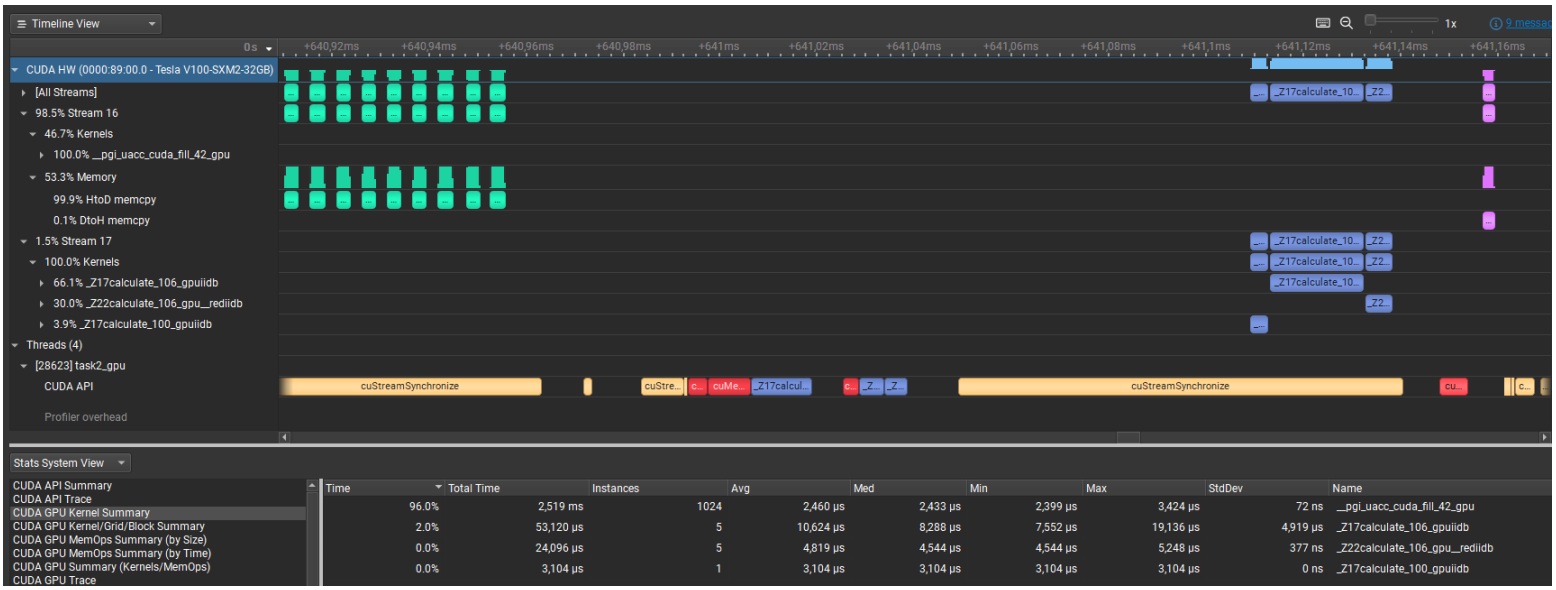
Как видно в таблице, изначальная инициализация дает хороший прирост производительности.



Этап 3

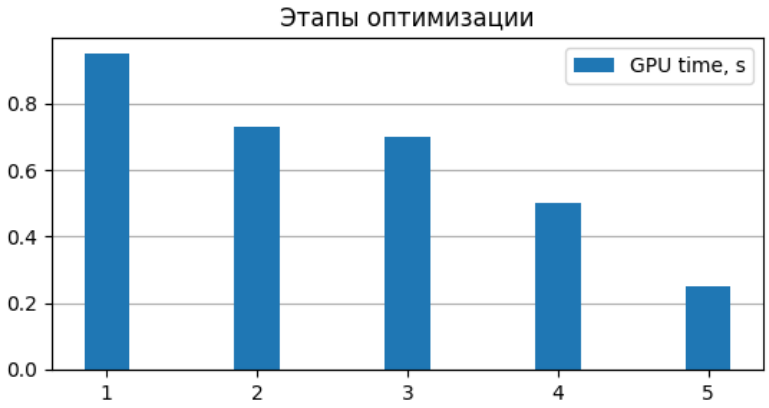


Этап 4



Этап 5

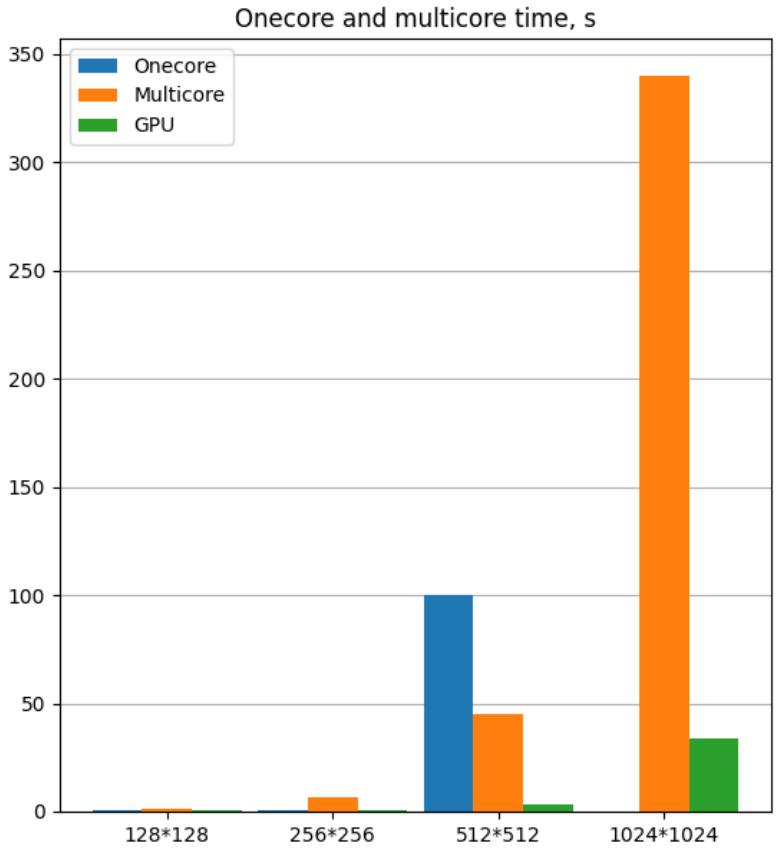
6.2 Диаграмма оптимизации (по горизонтали номер этапа; по вертикали время работы)



6.3 GPU – оптимизированный вариант

Размер сетки	Время выполнения, с	Точность	Количество опреаций
128*128	0.5	9.52e-07	11136
256*256	0.8	9.85e-07	37376
512*512	3.1	9.78e-07	120832
1024*1024	34	9.89e-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.

9 Приложение

9.1 Ссылка на GitHub

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

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

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

// Вывести значения двумерного массива
void print_array(T **A, int size)
{
#pragma acc update host(A[:size][:size])
    std::cout.precision(4);
    for (int i = 0; i < size; i += 1)
    {
        for (int j = 0; j < size; j += 1)
            std::cout << A[i][j] << "\t";
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

// Инициализация матрицы, чтобы подготовить ее к основному алгоритму
void initialize_array(T **A, int size)
{
    // 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);
    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
    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];
    }

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

    // Больше 30-и ошибки быть не должно
    T error = 30;
    // Счетчик итераций
    int iter = 0;
    // Указатель для swap
    T **temp;
    // Флаг обновления ошибки на хосте для обработки условием цикла
    bool update_flag = true;
#pragma acc data copyin(A[:net_size][:net_size], Anew[:net_size][:net_size]) copy(error)
    {
        while (error > accuracy && iter++ < iter_max)
        {
            // Сокращение количества обращений к CPU. Чем больше сетка, тем реже стоит сверять значения.
            update_flag = !(iter % net_size);
// напомнить о массивах
#pragma acc data present(A, Anew)
// асинхронно
#pragma acc kernels async(1)
            {
                // вернуть ошибку на девайс и занулить
                if (update_flag)
                {
#pragma acc update device(error)
                    error = 0;
                }
#pragma acc loop independent collapse(2) reduction(max \
                                                   : error)
                for (int j = 1; j < net_size - 1; j++)
                {
                    for (int i = 1; i < net_size - 1; i++)
                    {
                        Anew[i][j] = (A[i + 1][j] + A[i - 1][j] + A[i][j - 1] + A[i][j + 1]) * 0.25;
                        if (update_flag)
                            error = std::max(error, std::abs(Anew[i][j] - A[i][j]));
                    }
                }
            }

            // swap(A, Anew)
            temp = A, A = Anew, Anew = temp;

            // синхронизировать и отправить хосту для проверки условия
            if (update_flag)
            {
#pragma acc wait(1)
#pragma acc update self(error)
            }
        }
        if (res)
            print_array(A, net_size);
        std::cout << "iter=" << iter << ",\terror=" << error << std::endl;
    }

    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("-a"))
        {
#ifdef _FLOAT
            accuracy = std::stof(argv[arg + 1]);
#else
            accuracy = std::stod(argv[arg + 1]);
#endif
            arg++;
        }
        else if (!str.compare("-i"))
        {
            iter_max = (int)std::stod(argv[arg + 1]); // 1e6
            arg++;
        }
        else if (!str.compare("-s"))
        {
            net_size = std::stoi(argv[arg + 1]);
            arg++;
        }
        else if (!str.compare("-res"))
        {
            res = true;
        }
    }
    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;
}

```