Теория Параллелизма

Отчет

Уравнение теплопроводности (openacc)

Выполнил 21932, Бабенко Егор Степанович

08.03.2022

Цель работы

Реализовать решение уравнение теплопроводности методом Якоба (пятиточечный шаблон) для двумерной сетки. Произвести профилирование программы для GPU и оптимизацию кода. Произвести сравнение времени работы на CPU и GPU.

Используемый компилятор: *pgc++*

Для компиляции различных версий использовались команды:

- pgc++ t2.cpp -o t2_MultiCore.pg -fast -acc=multicore -O2 -Mconcur=allcores
- pgc++ t2.cpp -o t2_OneCore.pg -fast -O2
- pgc++ t2.cpp -o t2_GPU.pg -fast -acc=gpu -O2 -D OPENACC__

Для дополнительной профилировки: -D NVPROF_

Используемый профилировщики: *nvprof, nsys*

Nsys использовался с OpenACC trace, а также NVTX trace.

Замер времени работы всей программы производился с помощью команды time. Время высчитывалось как среднее между пятью запусками.

CPU-onecore

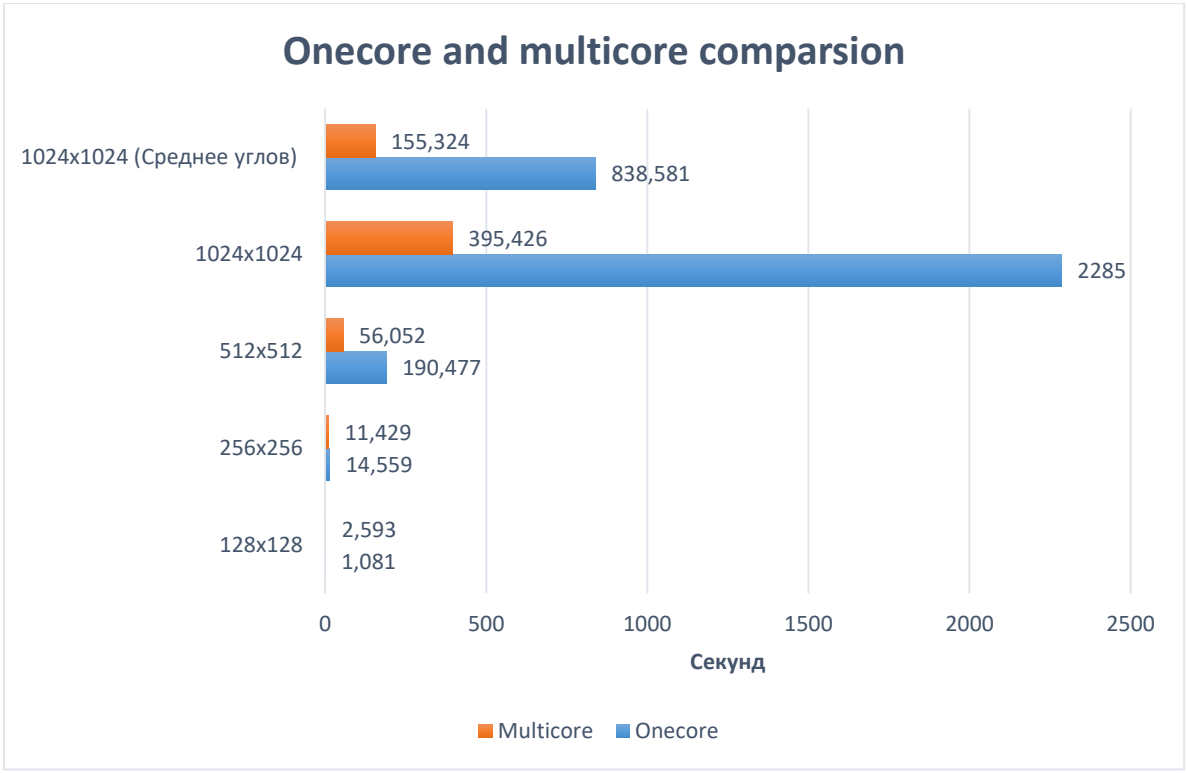
Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	1.081	9.9998e-07	30074
256x256	14.559	9.9993e-07	102885
512x512	190.477	9.99984e-07	339599
1024x1024	2285.354	1.36929e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	838,581	9.99993e-07	364620

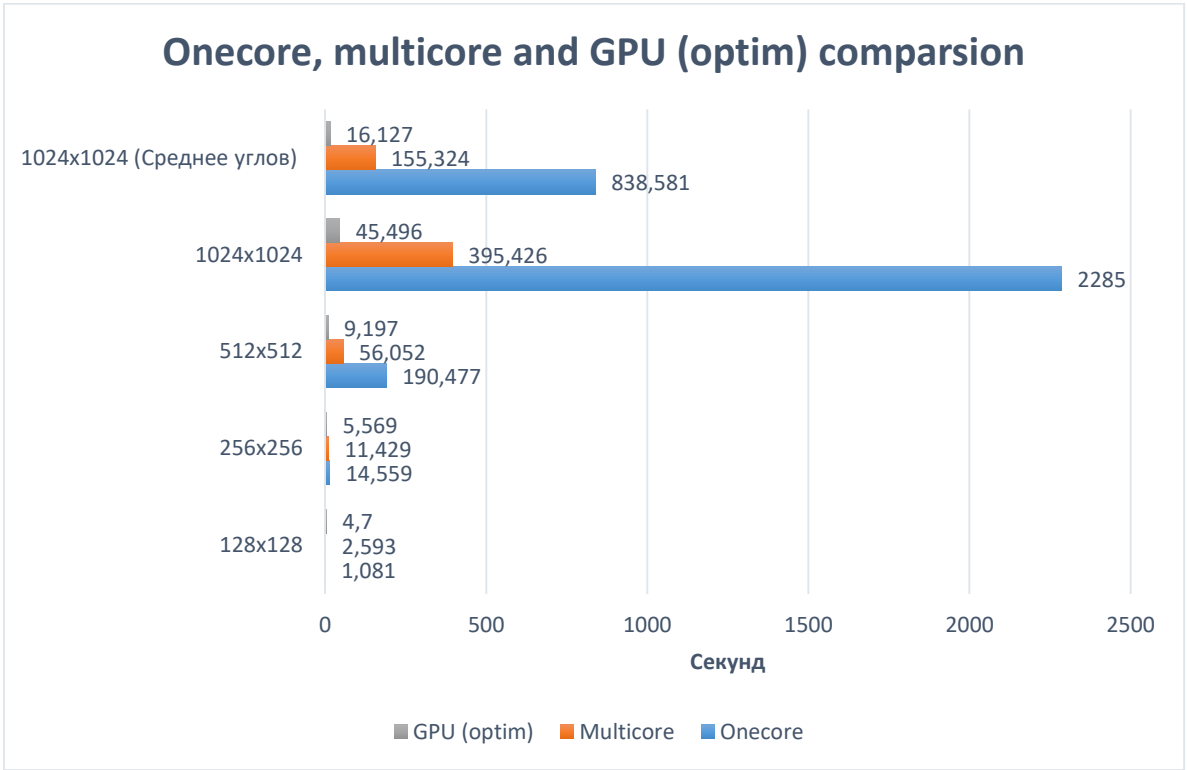
CPU-multicore

Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	2.593	9.9998e-07	30074
256x256	11.429	9.9993e-07	102885
512x512	56.052	9.99984e-07	339599
1024x1024	395.426	1.36929e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	155.324	9.99993e-07	364620

GPU-optimized

Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	4.700	9.98757e-07	30078
256x256	5.569	9.99703e-07	102888
512x512	9.197	9.99965e-07	339600
1024x1024	45.496	1.36929e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	16.127	9.99993e-07	364620

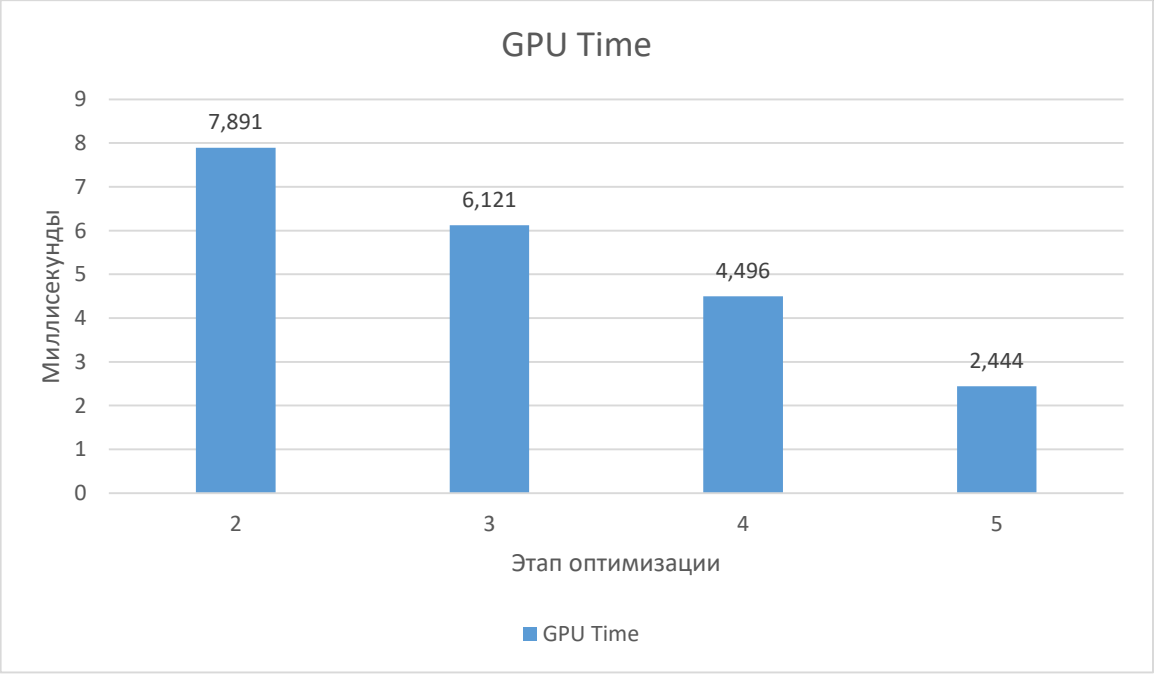

Onecore and multicore comparsion
1024x1024 (Среднее углов)
155,324
838,581
1024x1024
395,426
2285
512x512
56,052
190,477
256x256
11,429
14,559
128x128
2,593
1,081
0
500
1000
1500
2000
2500
Секунд
Multicore
Onecore


Onecore, multicore and GPU (optim) comparsion
1024x1024 (Среднее углов)
16,127
155,324
838,581
1024x1024
45,496
395,426
2285
512x512
9,197
56,052
190,477
256x256
5,569
11,429
14,559
128x128
4,7
2,593
1,081
0
500
1000
1500
2000
2500
Секунд
GPU (optim)
Multicore
Onecore

Этапы оптимизации кода на GPU

Этапы оптимизации на сетке 512x512

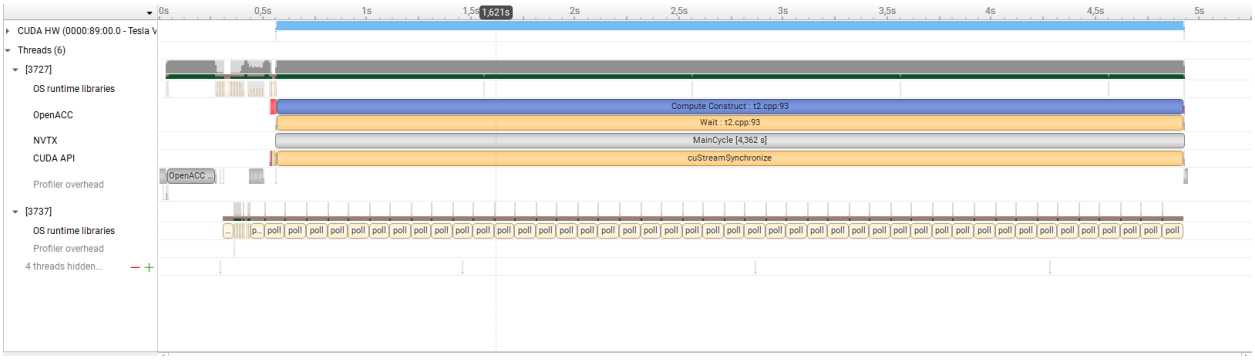
№	Время выполнения основного цикла	Ошибка	Кол-во итераций	Комментарий
1	4.362 с	0.107043	100	Попытка распараллелить весь цикл while (не получилось)
2	7.891 мс	0.107043	100	Распараллеливание только внутреннего цикла
3	6.121 мс	0.107043	100	Reduction на error, ручное управление памятью
4	4.496 мс	0.107043	100	Замена копирование DtoD на swap указателей внутри GPU, а также асинхронный запуск внутреннего цикла
5	2.444 мс	0.107043	100	Обновление error на хосте каждые 5 итераций (т.к. ошибка постоянно уменьшается, нет смысла проверять каждую итерацию)
6	2.543 мс	0.107043	100	Заполнение изначальной сетки среднем значением по углам. Ускорение за счет уменьшения кол-ва итераций для достижения точности



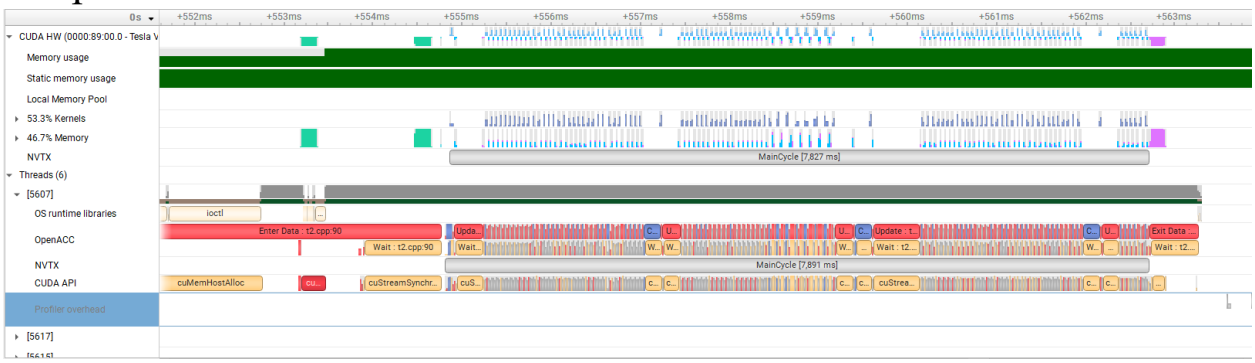
Возможной оптимизацией кода на GPU еще может быть уменьшение количества обменов данных между CPU и GPU. Т.е. делать проверку на

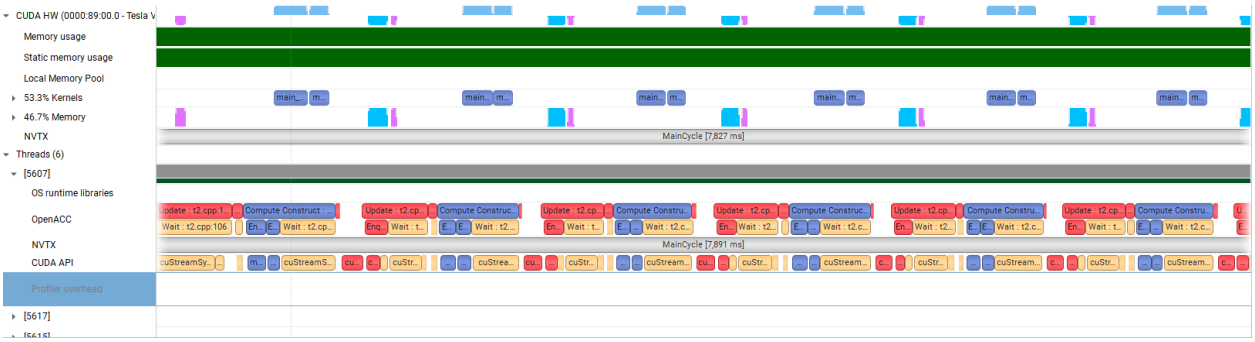
ошибки и количества итераций на GPU, тем самым мы уменьшим количество синхронизаций до минимального количества.

Перывый этап:

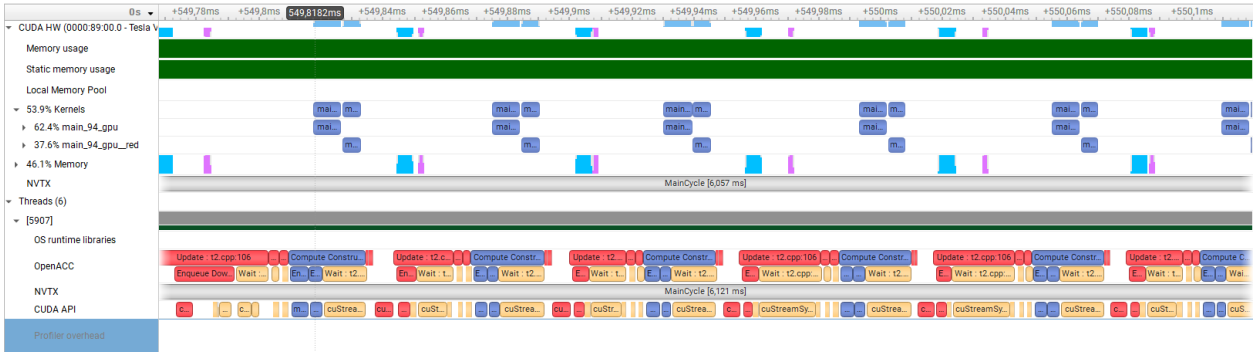


Второй этап:

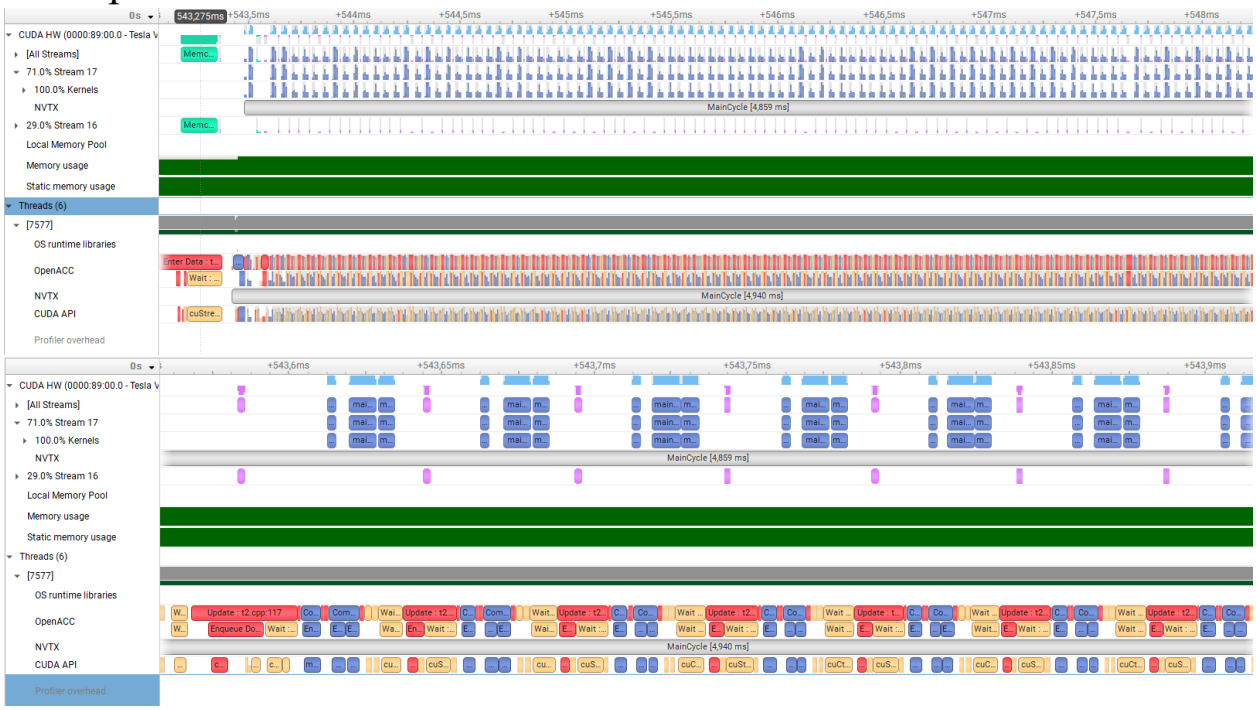




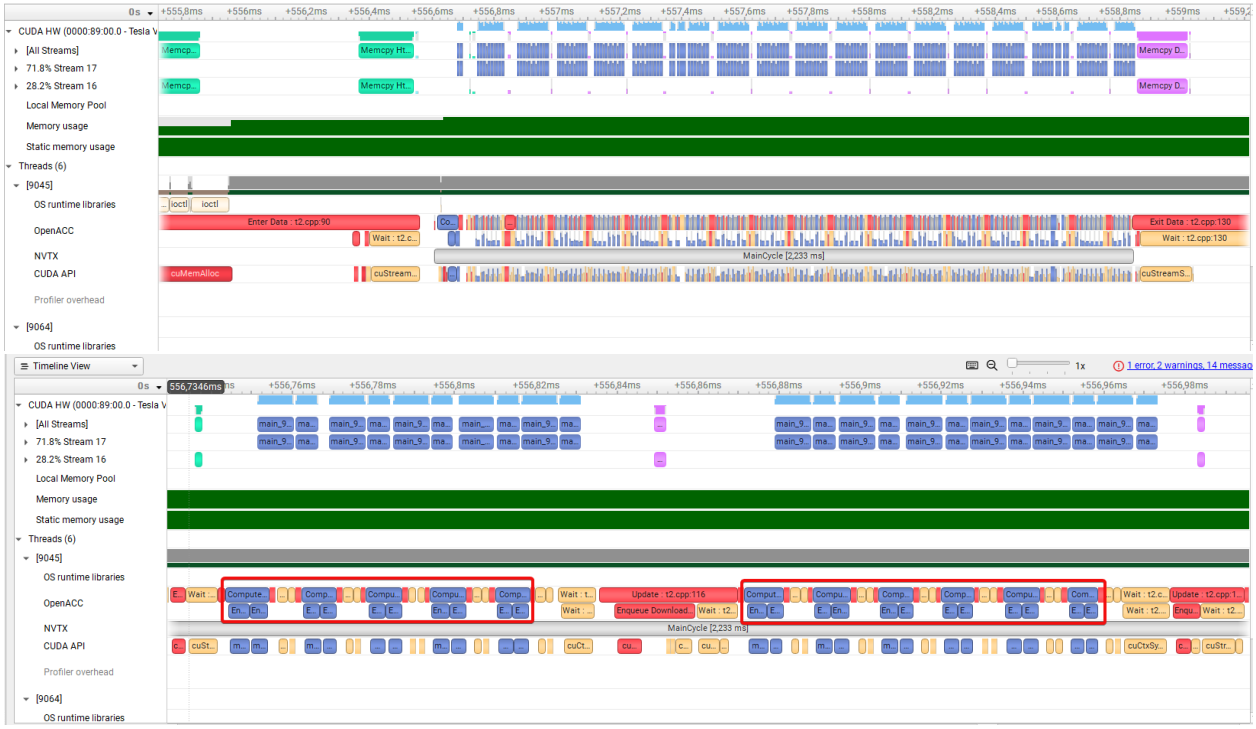
Третий этап:



Четвертый этап



Пятый этап:



Вывод:

Для обработки сетки небольшого размера будет целесообразнее использовать однопоточный вариант, т.к. издержки на инициализацию видеокарты занимают время. Но при увеличении размера сетки видно, что распараллеливание задачи более эффективно, чем выполнение в одном потоке. Т.к. среднее время выполнения одного цикла на GPU, сильно меньше, чем на CPU.

Github: https://github.com/JooudDoo/Parallelism-Tasks

```cpp
#include <iostream>
#include <cstring>
#include <sstream>
#include <math.h>
#include <cmath>

#ifdef OPENACC__
#include <openacc.h>
#endif
#ifdef NVPROF_
#include </opt/nvidia/hpc_sdk/Linux_x86_64/22.11/cuda/11.8/targets/x86_64-linux/include/nvtx3/nvToolsExt.h>
#endif

#define at(arr, x, y) (arr[(x)*n+(y)])

constexpr int LEFT_UP = 10;
constexpr int LEFT_DOWN = 20;
constexpr int RIGHT_UP = 20;
constexpr int RIGHT_DOWN = 30;


void initArrays(double* mainArr, double* subArr, int& n, int& m){
    std::memset(mainArr, 0, sizeof(double)*(n)*(m));

    for(int i = 0; i < n*n; i++){
        mainArr[i] = (LEFT_UP+LEFT_DOWN+RIGHT_UP+RIGHT_DOWN)/4;
    }

    at(mainArr, 0, 0) = LEFT_UP;
    at(mainArr, 0, m-1) = RIGHT_UP;
    at(mainArr, n-1, 0) = LEFT_DOWN;
    at(mainArr, n-1, m-1) = RIGHT_DOWN;
    for(int i = 1; i < n-1; i++){
        at(mainArr,0,i) = (at(mainArr,0,m-1)-at(mainArr,0,0))/(m-1)*i+at(mainArr,0,0);
        at(mainArr,i,0) = (at(mainArr,n-1,0)-at(mainArr,0,0))/(n-1)*i+at(mainArr,0,0);

        at(mainArr,n-1,i) = (at(mainArr,n-1,m-1)-at(mainArr,n-1,0))/(m-1)*i+at(mainArr,n-1,0);
        at(mainArr,i,m-1) = (at(mainArr,n-1,m-1)-at(mainArr,0,m-1))/(m-1)*i+at(mainArr,0,m-1);
    }
    std::memcpy(subArr, mainArr, sizeof(double)*(n)*(m));
}

template<typename T>
T extractNumber(char* arr){
    std::stringstream stream;
    stream << arr;
    T result;
    if (!(stream >> result)){
        throw std::invalid_argument("Wrong argument type");
    }
    return result;
}

constexpr int ITERS_BETWEEN_UPDATE = 5;

int main(int argc, char *argv[]){

    bool showResultArr = false;
    double eps = 1E-6;
    int iterations = 1E6;
    int n = 10;
    int m = n;

    for(int arg = 0; arg < argc; arg++){
        if(std::strcmp(argv[arg], "-eps") == 0){
            eps = extractNumber<double>(argv[arg+1]);
            arg++;
        }
        else if(std::strcmp(argv[arg], "-i") == 0){
            iterations = extractNumber<int>(argv[arg+1]);
            arg++;
        }
        else if(std::strcmp(argv[arg], "-s") == 0){
            n = extractNumber<int>(argv[arg+1]);
            m = n;
            arg++;
        }
        else if(std::strcmp(argv[arg], "-show") == 0){
            showResultArr = true;
        }
    }

    std::cout << "Current settings:" << std::endl;
    std::cout << "\tEPS: " << eps << std::endl;
    std::cout << "\tMax iteration: " << iterations << std::endl;
    std::cout << "\tSize: " << n << 'x' << m << std::endl << std::endl;

    double* F = new double[n*m];
    double* Fnew = new double[n*m];

    initArrays(F, Fnew, n, m);

    double error = 0;
    int iteration = 0;

    int itersBetweenUpdate = 0;

    #pragma acc enter data copyin(Fnew[:n*m], F[:n*m], error)

#ifdef NVPROF_
    nvtxRangePush("MainCycle");
#endif
    do {
        #pragma acc parallel present(error) async(0)
        {
            error = 0;
        }

        #pragma acc parallel loop collapse(2) present(Fnew[:n*m], F[:n*m], error) reduction(max:error) vector_length(128) async(0)
        for(int x = 1; x < n-1; x++){
            for(int y= 1; y < m-1; y++){
                at(Fnew,x,y) = 0.25 * (at(F, x+1,y) + at(F,x-1,y) + at(F,x,y-1) + at(F,x,y+1));
                error = fmax(error, fabs(at(Fnew,x,y) - at(F,x,y)));
            }
        }

        double* swap = F;
        F = Fnew;
        Fnew = swap;

#ifdef OPENACC__
        acc_attach((void**)F);
        acc_attach((void**)Fnew);
#endif
        if(itersBetweenUpdate >= ITERS_BETWEEN_UPDATE && iteration < iterations){
            #pragma acc update self(error) wait
            itersBetweenUpdate = -1;
        }
        else{
            error = 1;
        }
        iteration++;
        itersBetweenUpdate++;
    } while(iteration < iterations && error > eps);
#ifdef NVPROF_
    nvtxRangePop();
#endif

    #pragma acc exit data delete(Fnew[:n*m]) copyout(F[:n*m], error)

    std::cout << "Iterations: " << iteration << std::endl;
    std::cout << "Error: " << error << std::endl;
    for(int x = 0; x < n && showResultArr; x++){
        for(int y = 0; y < m; y++){
            std::cout << at(F,x, y) << ' ';
        }
        std::cout << std::endl;
    }

    delete[] F;
    delete[] Fnew;

    return 0;
}
```