Теория Параллелизма

Отчет
Уравнение теплопроводности
(cublas + openacc)

Выполнил 21932, Бабенко Егор Степанович

18.03.2022

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. Для CPU использовалась программа из прошлого задания.

Используемый компилятор: *pgc++*

Для компиляции версии с cublas использовалось:

- pgc++ t3.cpp -o t3_GPU.pg -fast -acc=gpu -O2 -D OPENACC__ -Mcudalib=cublas

Для дополнительной профилировки: -D NVPROF_

Используемый профилировщики: *nvprof, nsys*

Nsys использовался с OpenACC trace, cuBLAS trace, а также NVTX trace.

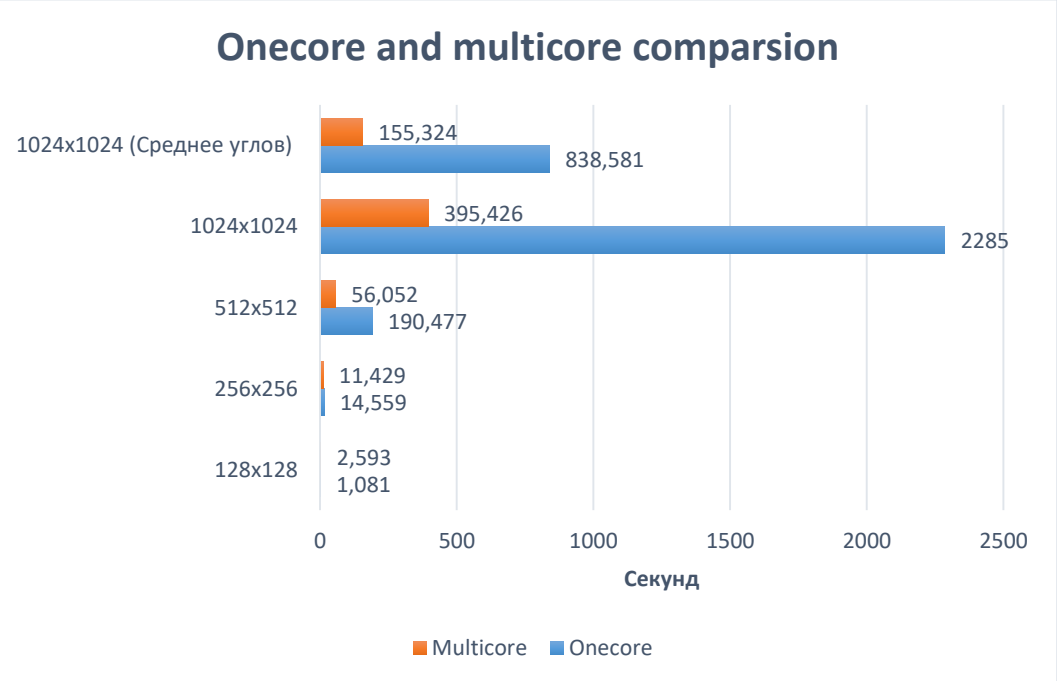
Замер времени работы всей программы производился с помощью команды time. Время высчитывалось как среднее между пятью запусками.

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

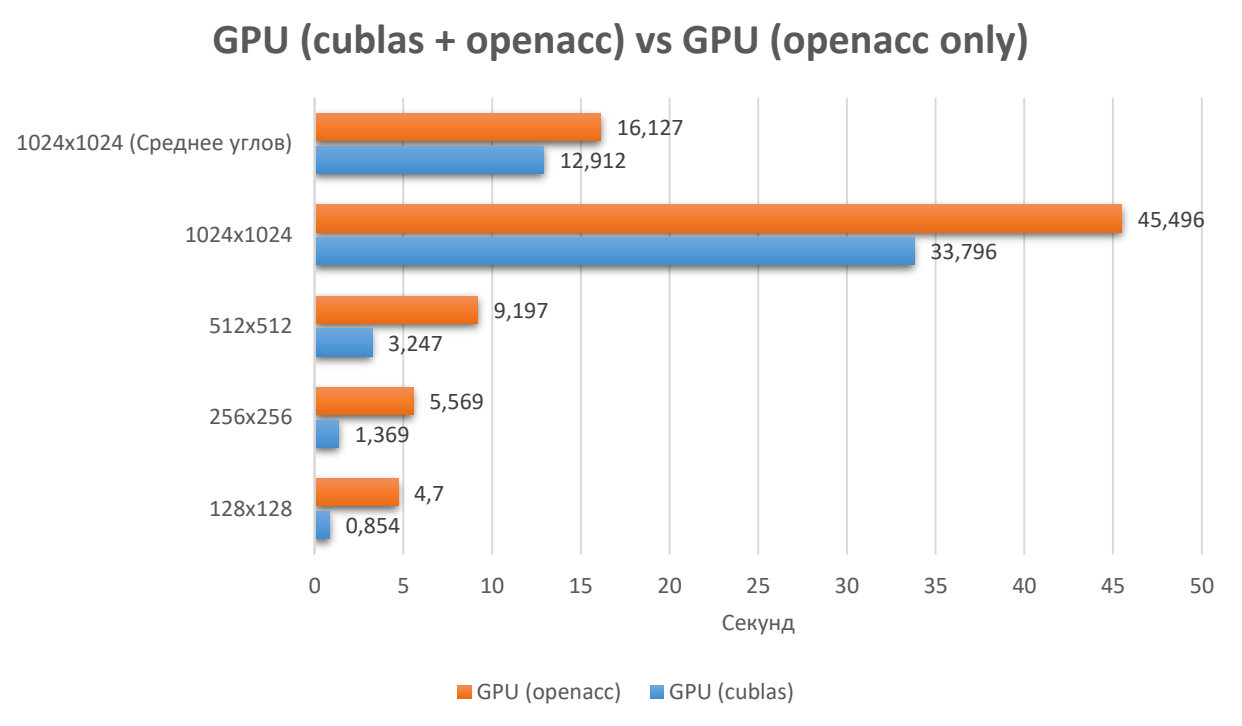


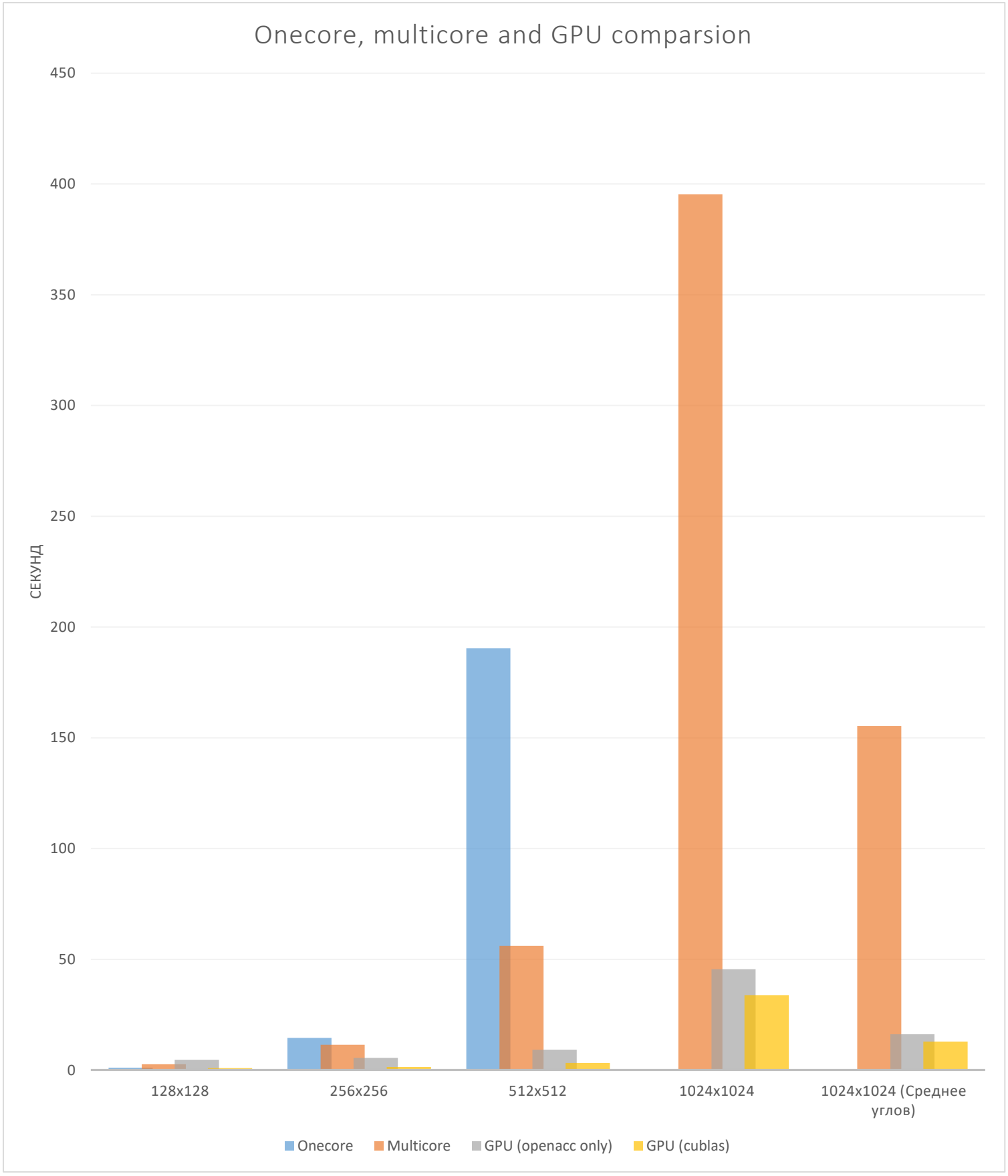
GPU-optimized (openacc only)

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

GPU-optimized (cublas)

Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	0.854	9.98757e-07	30120
256x256	1.369	9.99703e-07	102910
512x512	3.247	9.99965e-07	339603
1024x1024	33.796	1.36929e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	12.912	9.99993e-07	364703

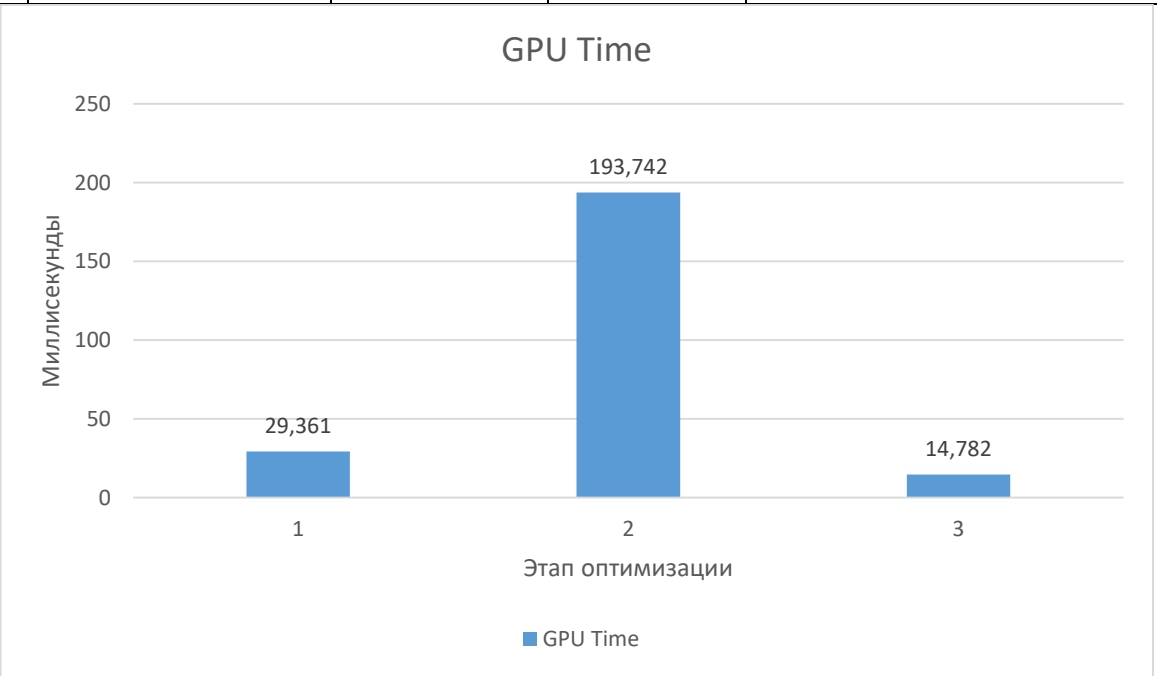



Onecore, multicore and GPU comparsion
450
400
350
300
250
200
150
100
50
0
СЕКУНД
128x128
256x256
512x512
1024x1024
1024x1024 (Среднее углов)
Onecore
Multicore
GPU (openacc only)
GPU (cublas)

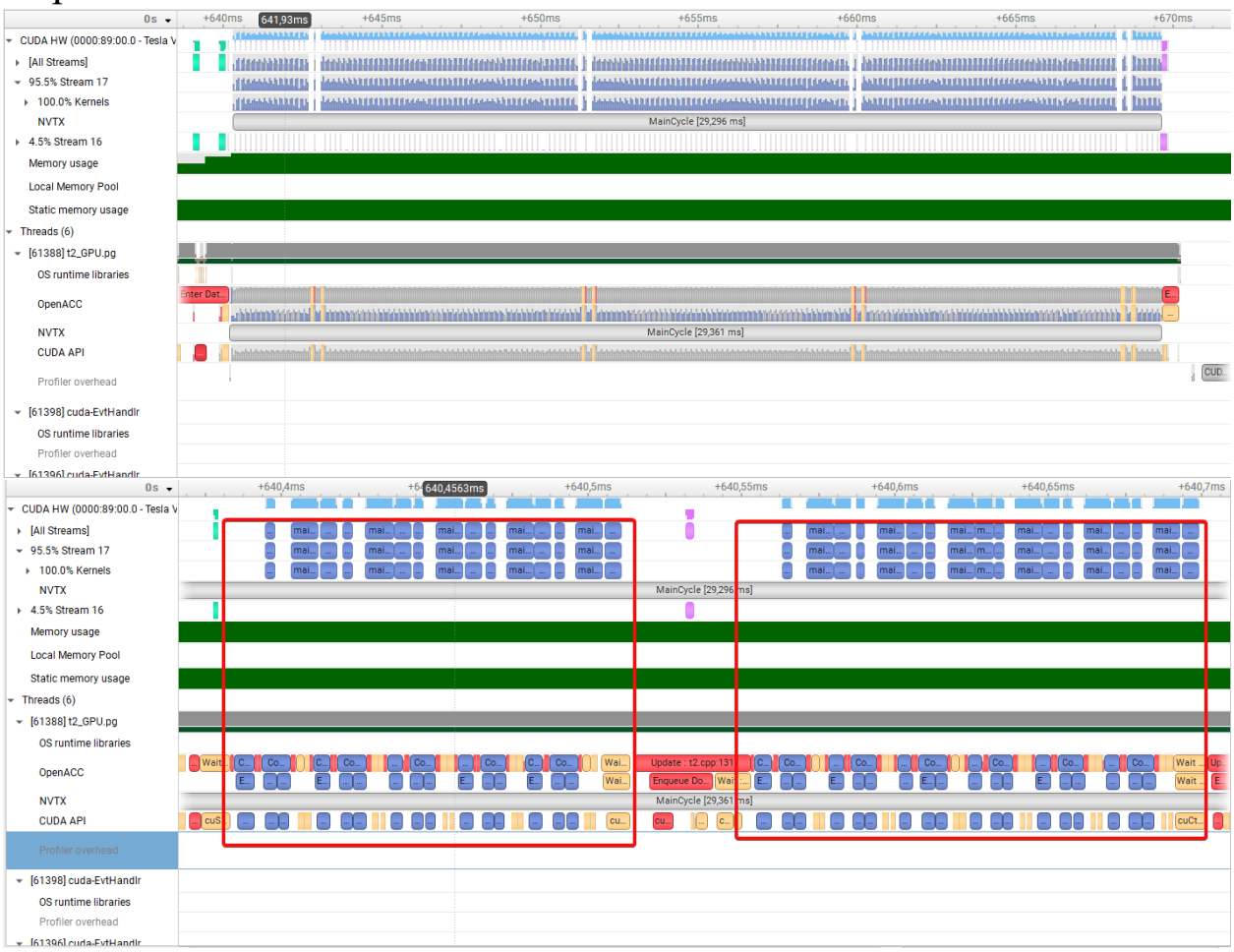
Этапы оптимизации кода на GPU (cublas)

Этапы оптимизации на сетке 512x512

№	Время выполнения основного цикла	Ошибка	Кол-во итераций	Комментарий
1	29,361 мс	0.0105525	1000	Код написанный только с использованием openacc (reduction выполнено через прагмы openacc, результат прошлого задания)
2	193,742 мс	0.0105524	1000	Выполнение расчета ошибки с помощью cublas (перерасчет каждую итерацию)
3	14,782 мс	0.0140183	1000	Перерасчет ошибки каждые 250 итераций. Значение подобрано эмпирически на основе данных профилировщика, как оптимальное для разных размеров сеток и с учетом времени на синхронизацию потоков.

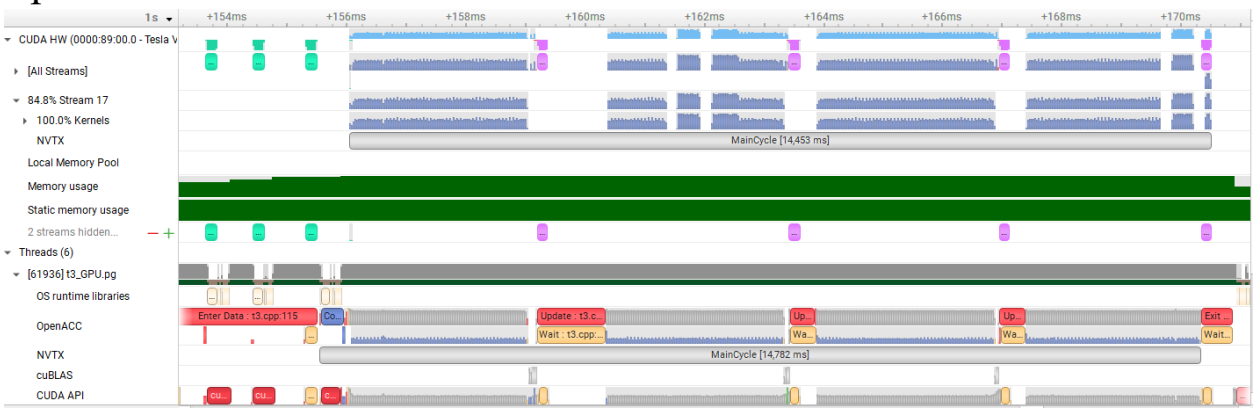


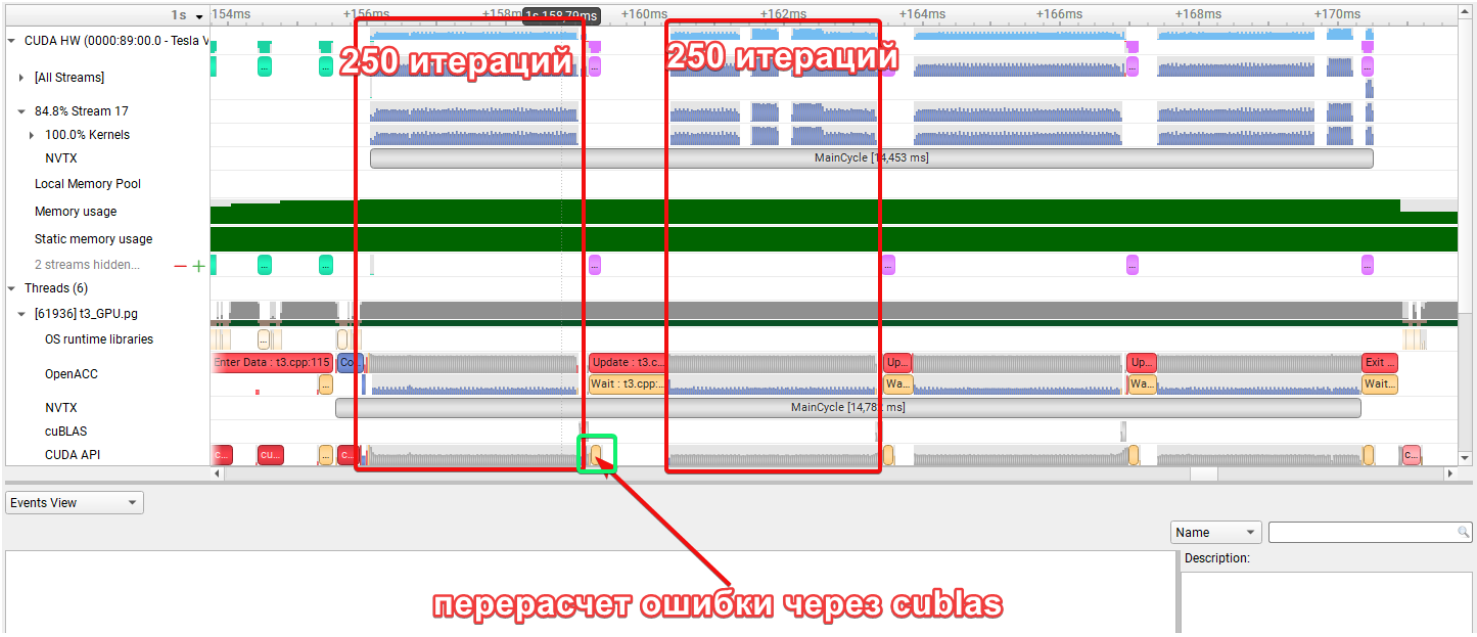
Первый этап:



Второй этап

Третий этап


1s
+154ms
+156ms
+158ms
+160ms
+162ms
+164ms
+166ms
+168ms
+170ms
CUDA HW (0000:89:00.0 - Tesla V
[All Streams]
84.8% Stream 17
100.0% Kernels
NVTX
MainCycle [14,453 ms]
Local Memory Pool
Memory usage
Static memory usage
2 streams hidden...
Threads (6)
[61936] t3_GPU.pg
OS runtime libraries
OpenACC
Enter Data : t3.cpp:115
Update : t3.c...
Wait : t3.cpp:...
MainCycle [14,782 ms]
cuBLAS
CUDA API


250 итераций
250 итераций
CUDA HW (0000:89:00.0 - Tesla V
[All Streams]
84.8% Stream 17
100.0% Kernels
NVTX
MainCycle [14,453 ms]
Local Memory Pool
Memory usage
Static memory usage
2 streams hidden...
Threads (6)
[61936] t3_GPU.pg
OS runtime libraries
OpenACC
Enter Data : t3.cpp:115
Update : t3.c...
Wait : t3.cpp:...
cuBLAS
CUDA API
Events View
Name
Description:
перерасчет ошибки через cublas

Вывод:

Используя cublas мы можем добиться меньшего времени за счет того что производим синхронизацию вычислительных потоков как можно реже, а само вычисление ошибки производим достаточно быстро. В итоге получаем время вычисление уравнения на маленьких сетках меньше, чем на CPU, за счет грамотного использования ресурсов видеокарты и минимизации синхронизаций данных между CPU/GPU и вычислительными потоками.

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

#include <cublas_v2.h>

#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, bool initUsingMean){
    std::memset(mainArr, 0, sizeof(double)*(n)*(m));

    for(int i = 0; i < n*n && initUsingMean; 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,0,1) = (at(mainArr,0,m-1)-at(mainArr,0,0))/(m-1)*1+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 = 250;
constexpr double negOne = -1;

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

    bool showResultArr = false;
    bool initUsingMean = 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;
        }
        else if(std::strcmp(argv[arg], "-O") == 0){
            initUsingMean = 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];

    double* inter = new double[n*m];

    initArrays(F, Fnew, n, m, initUsingMean);

    double error = 1;
    int iteration = 0;

    int itersBetweenUpdate = 0;

    cublasHandle_t handle;

    cublasCreate(&handle);

    cublasStatus_t status;
    int max_idx = 0;

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

#ifdef NVPROF_
    nvtxRangePush("MainCycle");
#endif
    do {
        #pragma acc parallel loop collapse(2) present(Fnew[:n*m], F[:n*m]) vector_length(128) async
        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));
            }
        }

        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 data present(inter[:n*m], Fnew[:n*m], F[:n*m]) wait
            {
                #pragma acc host_data use_device(Fnew, F, inter)
                {

                    status = cublasDcopy(handle, n*m, F, 1, inter, 1);
                    if(status != CUBLAS_STATUS_SUCCESS) std::cout << "copy error" << std::endl, exit(30);

                    status = cublasDaxpy(handle, n*m, &negOne, Fnew, 1, inter, 1);
                    if(status != CUBLAS_STATUS_SUCCESS) std::cout << "sum error" << std::endl, exit(40);

                    status = cublasIdamax(handle, n*m, inter, 1, &max_idx);
                    if(status != CUBLAS_STATUS_SUCCESS) std::cout << "abs max error" << std::endl, exit(41);
                }
            }

            #pragma acc update self(inter[:n*m])
            error = fabs(inter[max_idx]);

            itersBetweenUpdate = -1;
        }
        iteration++;
        itersBetweenUpdate++;
    } while(iteration < iterations && error > eps);

#ifdef NVPROF_
    nvtxRangePop();
#endif

    #pragma acc exit data delete(Fnew[:n*m]) copyout(F[:n*m])
    cublasDestroy(handle);

    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;
}
```