

Теория параллелизма

Отчет

Уравнение теплопроводности

Выполнил: Пучков Владислав, гр. 22933

Дата 20.05.2024

Цель: Реализовать решение уравнения теплопроводности в двумерной области с использованием разностной схемы (пятиточечный шаблон) на равномерных сетках. Программа должна учитывать линейную интерполяцию на границах и заданные значения в углах, ограничивать точность до 10^{-6} и максимальное число итераций до 10^6 . Реализация должна быть на C++ с использованием OpenACC для переноса на GPU. Необходимо сравнить производительность на CPU и GPU, провести профилирование и оптимизацию кода.

Используемый компилятор: pgc++ 23.11-0

Используемый профилировщик: NVIDIA Nsight Systems

Как производили замер времени работы: При помощи Библиотеки chrono замерил время начала и конца цикла итераций

Выполнение на CPU

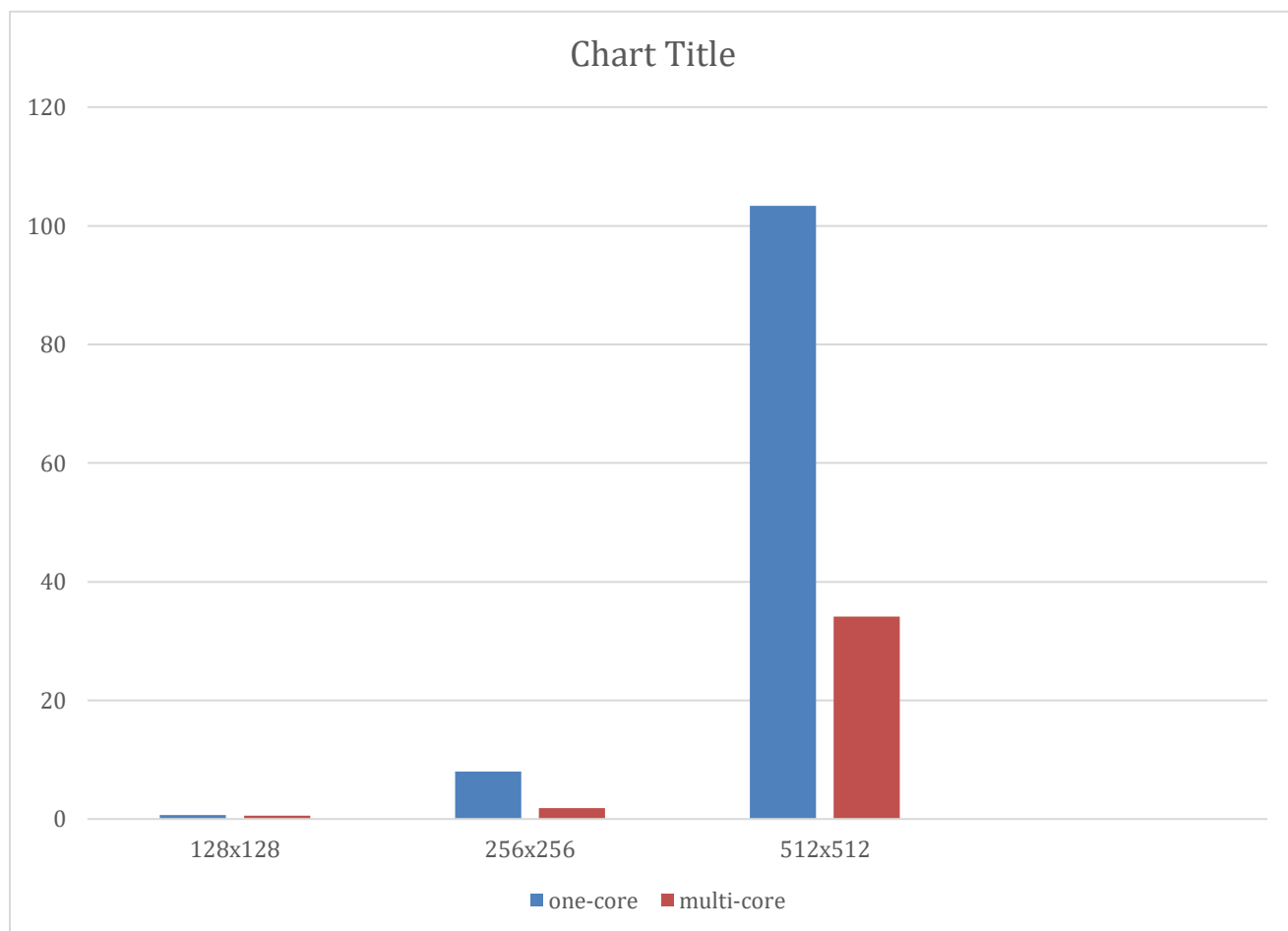
CPU-onescore

Размер сетки	Время выполнения (сек)	Точность	Количество операций
128*128	0,65	0.000001	31000
256*256	7,98	0.000001	110000
512*512	103,4	0.000001	340000

CPU-multicore

Размер сетки	Время выполнения (сек)	Точность	Количество операций
128*128	0,52	0.000001	31000
256*256	1,81	0.000001	110000
512*512	34,1	0.000001	340000
1024*1024	321,62	0.000001	1000000

Диаграмма сравнения время работы CPU-one и CPU-multi



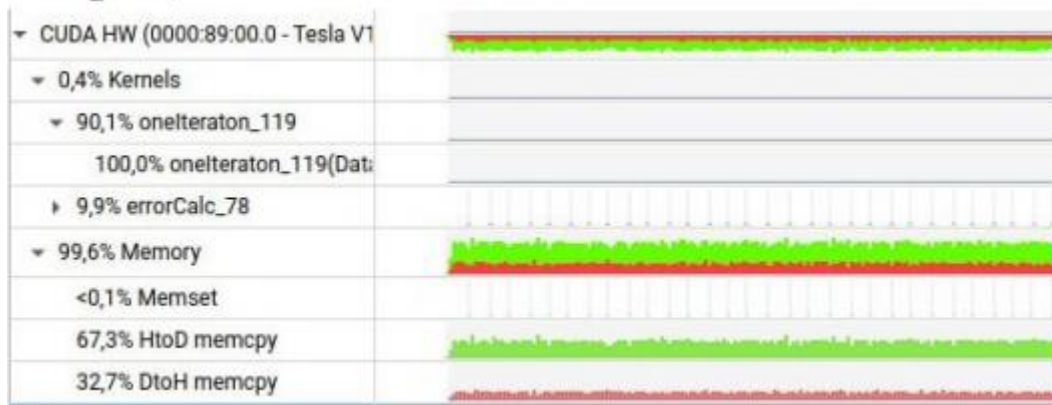
Выполнение на GPU
Этапы оптимизации на сетке 1024*1024

Этап	Время выполнения	Точность	Количество итераций	Комментарии
1	71,83	>0.000001	1000000	Замена swap через temp на swap через указатели
2	44,56	>0.000001	1000000	Возвращение ошибки каждые 1000 операций
3	42,768	>0.000001	1000000	Замена распараллеливания цикла на #pragma acc parallel loop independent collapse(2) vector vector_length(256) gang num_gangs(1024) present(currentMatrix, previousMatrix)
4	34,987	>0.000001	1000000	Выбор самой свободной видеокарты через библиотеку #include <openacc.h> и команду acc_set_device_num(N, acc_device_nvidia);

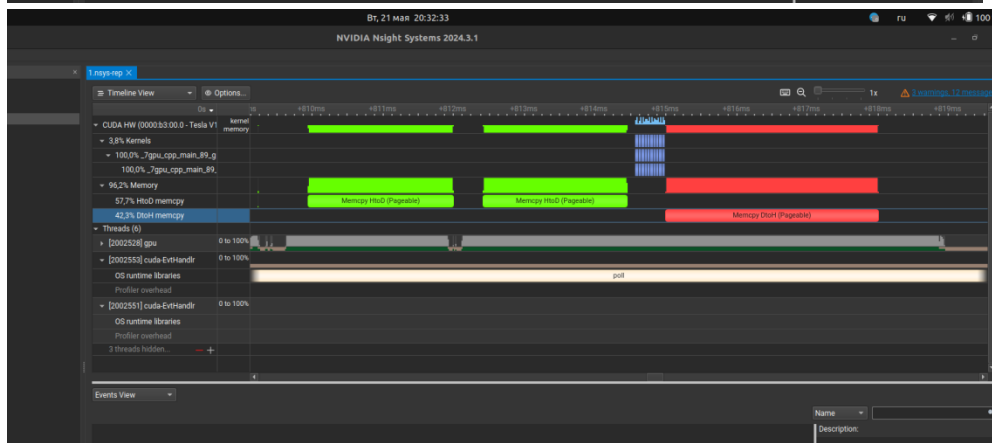
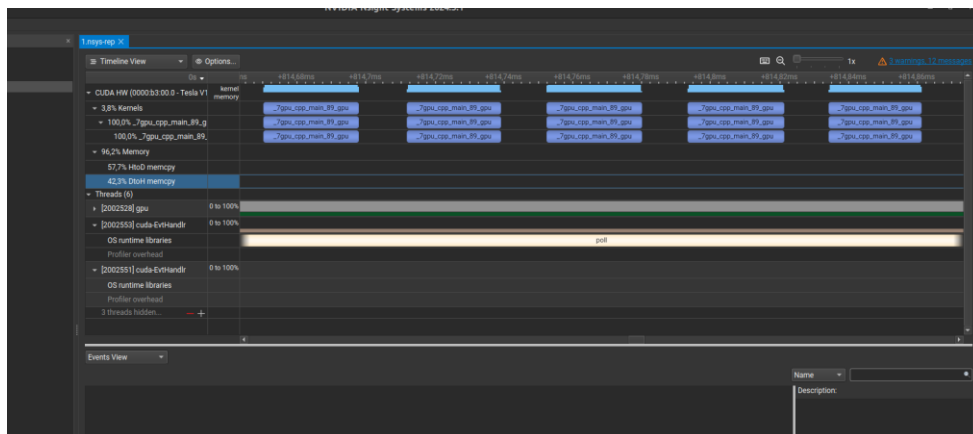
Диаграмма оптимизации



Профилирование:



Большая часть времени GPU отправляет данные на каждой итерации, вместо вычислений

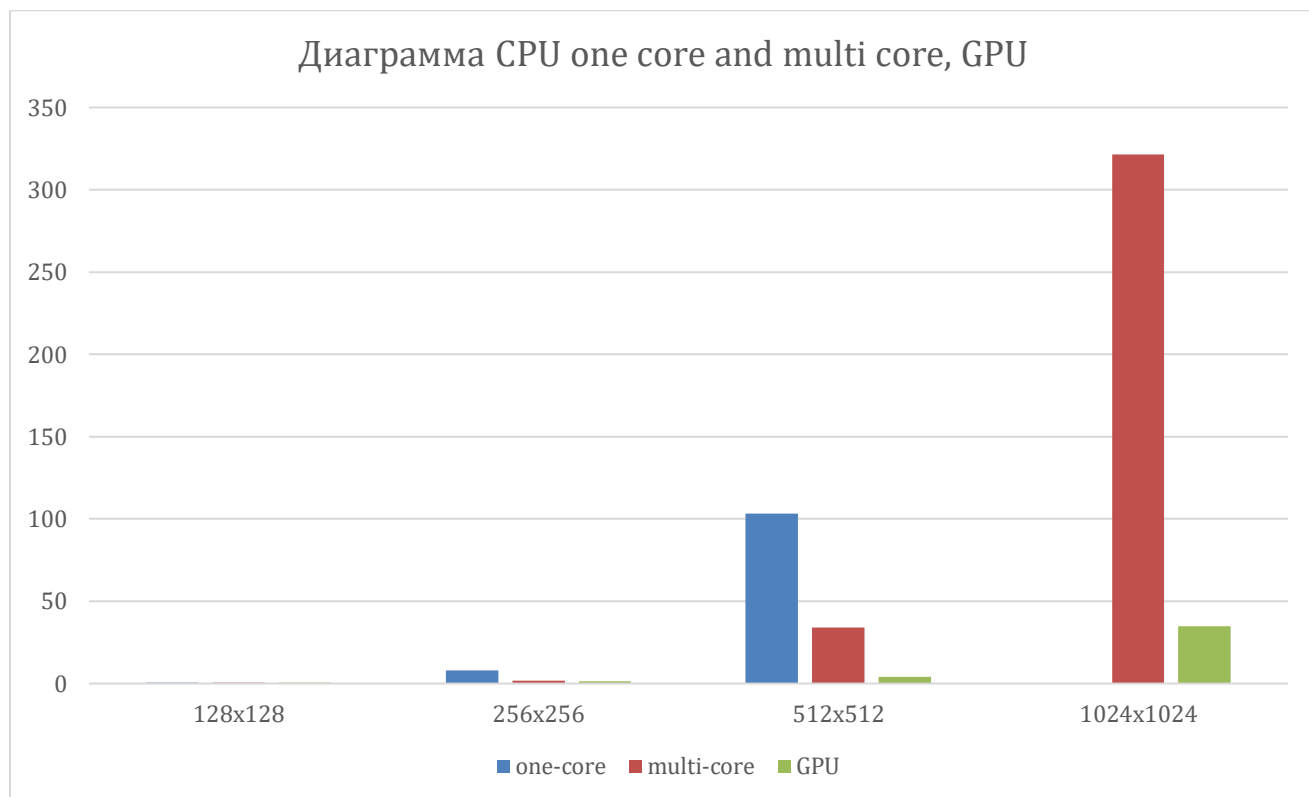


скорость работы была увеличена за счёт того что ошибка вычисляется не на каждой итерации, так как максимизация переменной среди потоков требует времени на синхронизацию между потоками.

GPU - оптимизированный вариант

Размер сетки	Время выполнения(с)	Точность	Количество операций
128*128	0,43	0.000001	40000
256*256	1,27	0.000001	110000
512*512	4,2	0.000001	340000
1024*1024	34,8	>0.000001	1000000

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



GPU - оптимизированный вариант + cuBLAS

Размер сетки	Время выполнения(с)	Точность	Количество операций
128*128	0,5	0.000001	40000
256*256	1,63	0.000001	110000
512*512	5,63	0.000001	340000
1024*1024	33,82	>0.000001	1000000

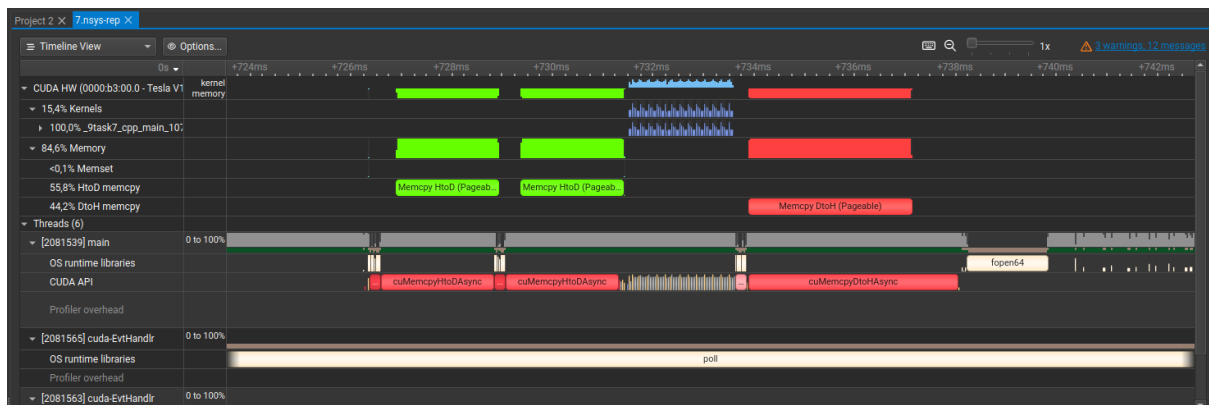
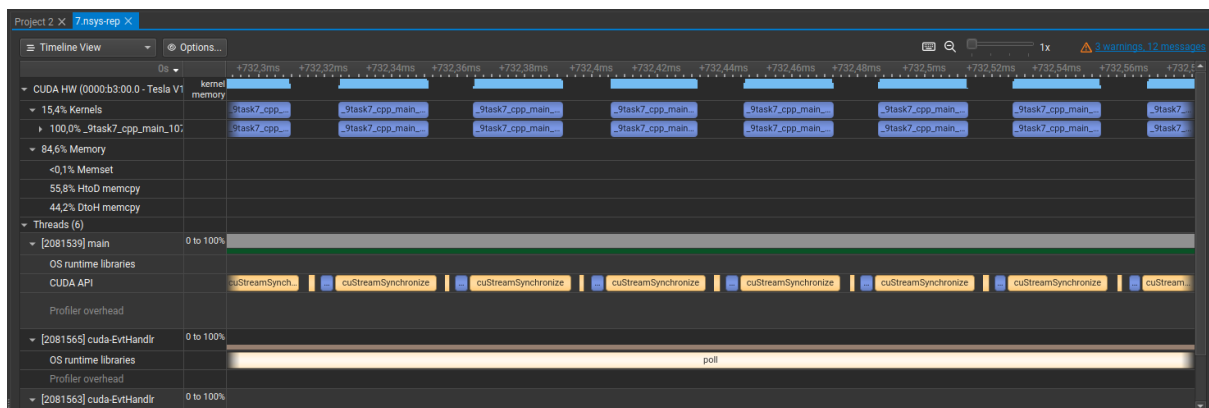
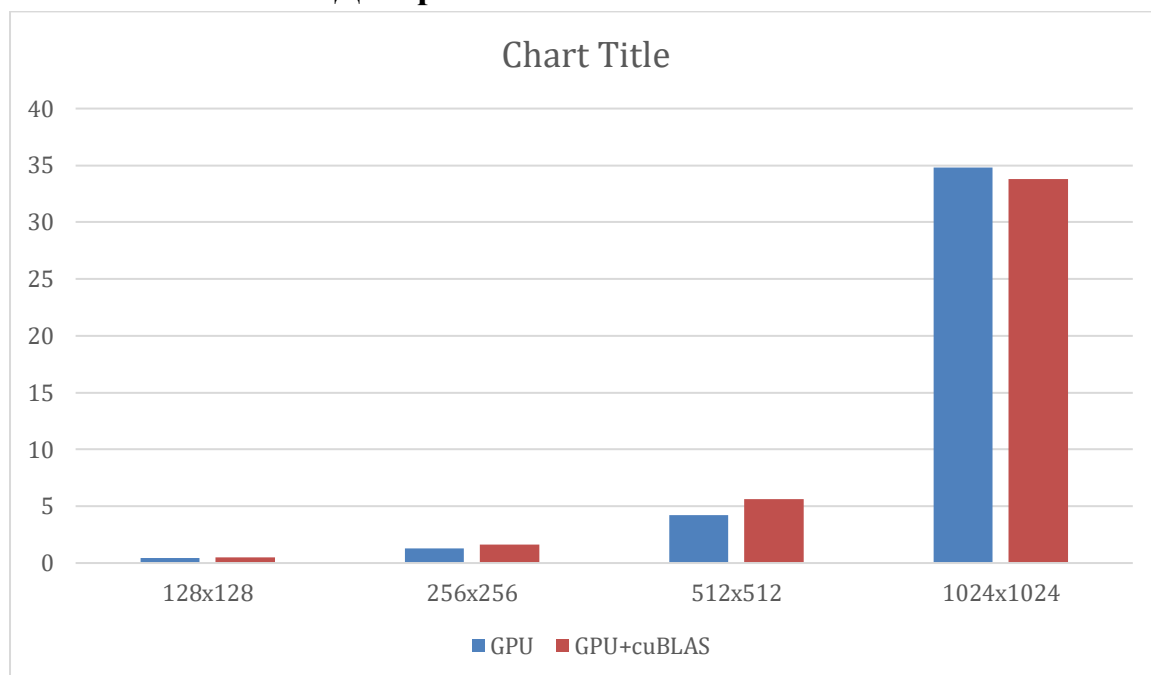
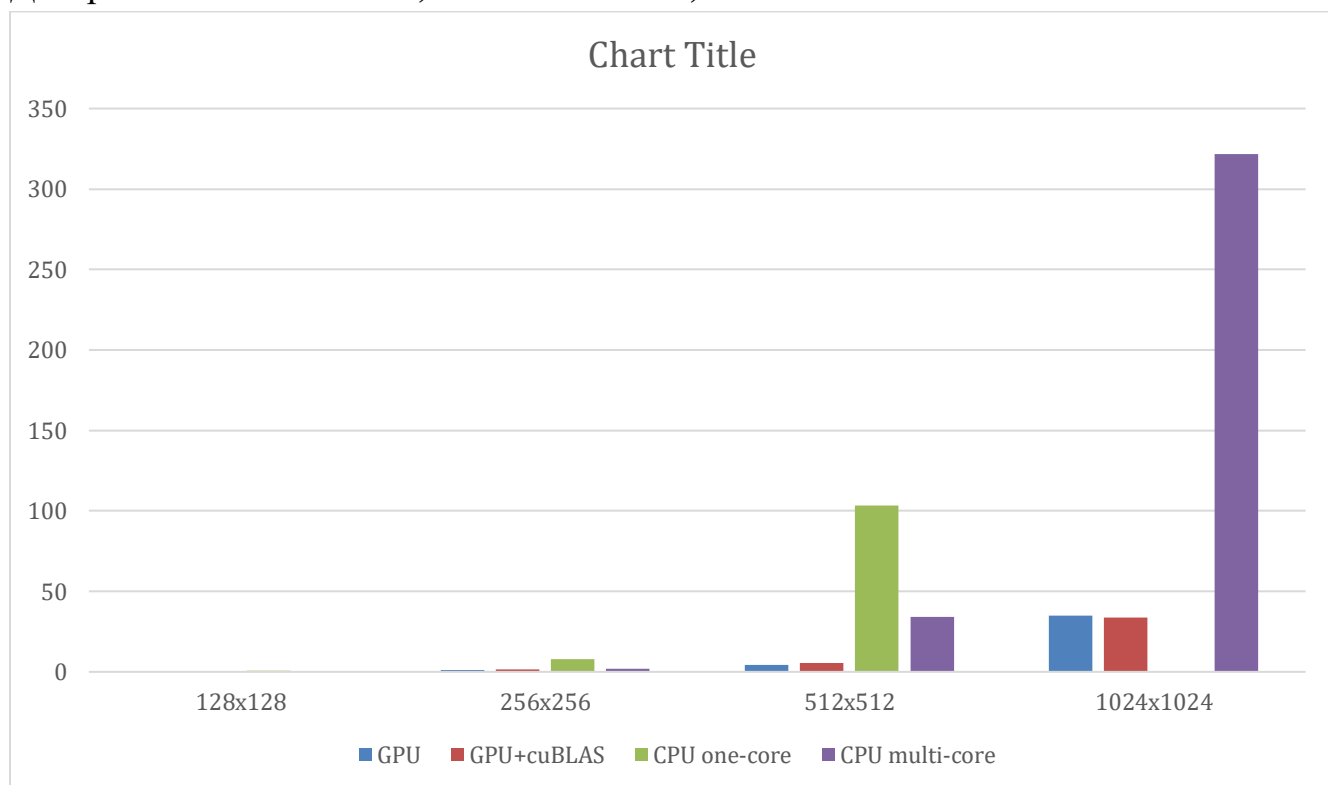


Диаграмма GPU и GPU+cuBLAS



Вывод: GPU+ cuBLAS работает чуть быстрее чем без cuBLAS'а.

Диаграмма **CPU one-core, CPU multi-core, GPU и GPU+cuBLAS**



10x10

```
● v.puchkov@d2e6a4e2eddd:~/task6_k/gpu$ ./gpu
Итерация: 10000 ошибка: 0
Время: 121 мс, Ошибка: 0, Итерации: 10000
10 11.1111 12.2222 13.3333 14.4444 15.5556 16.6667 17.7778 18.8889 20
11.1111 12.2222 13.3333 14.4444 15.5556 16.6667 17.7778 18.8889 20 21.1111
12.2222 13.3333 14.4444 15.5556 16.6667 17.7778 18.8889 20 21.1111 22.2222
13.3333 14.4444 15.5556 16.6667 17.7778 18.8889 20 21.1111 22.2222 23.3333
14.4444 15.5556 16.6667 17.7778 18.8889 20 21.1111 22.2222 23.3333 24.4444
15.5556 16.6667 17.7778 18.8889 20 21.1111 22.2222 23.3333 24.4444 25.5556
16.6667 17.7778 18.8889 20 21.1111 22.2222 23.3333 24.4444 25.5556 26.6667
17.7778 18.8889 20 21.1111 22.2222 23.3333 24.4444 25.5556 26.6667 27.7778
18.8889 20 21.1111 22.2222 23.3333 24.4444 25.5556 26.6667 27.7778 28.8889
20 21.1111 22.2222 23.3333 24.4444 25.5556 26.6667 27.7778 28.8889 30
```

13x13

```
Итерация: 10000 ошибка: 0
Время: 112 мс, Ошибка: 0, Итерации: 10000
10 10.8333 11.6667 12.5 13.3333 14.1667 15 15.8333 16.6667 17.5 18.3333 19.1667 20
10.8333 11.6667 12.5 13.3333 14.1667 15 15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333
11.6667 12.5 13.3333 14.1667 15 15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667
12.5 13.3333 14.1667 15 15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667 22.5
13.3333 14.1667 15 15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333
14.1667 15 15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667
15 15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667 25
15.8333 16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667 25 25.8333
16.6667 17.5 18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667 25 25.8333 26.6667
17.5 18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667 25 25.8333 26.6667 27.5
18.3333 19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667 25 25.8333 26.6667 27.5 28.3333
19.1667 20 20.8333 21.6667 22.5 23.3333 24.1667 25 25.8333 26.6667 27.5 28.3333 29.1667
20 20.8333 21.6667 22.5 23.3333 24.1667 25 25.8333 26.6667 27.5 28.3333 29.1667 30
```

Вывод: Лучше всего GPU себя показывает на больших матрицах, на небольших размерах GPU, CPU-onecore, CPU-multicore – очень близки по времени. Возврат с GPU на CPU ошибки при большом количестве операций, необязательно проверять ошибку каждый раз, так как копирование занимает также время

Вывод **cuBLAS**: если выполнять операцию редукции (вычисление максимального значения ошибки) на графическом процессоре реализовать через вызовы функций из библиотеки cuBLAS, то это дает небольшое уменьшение время выполнения программы.