Теория параллелизма

Отчёт

Решение уравнения теплопроводности оптимизированными методами

Выполнила Рыбинцева Светлана 23930
05.05.25

Цель работы: Реализовать оптимизированное решение уравнения теплопроводности (разностная схема – пятиточечный шаблон) на двумерной области на равномерных сетках (128^2, 256^2, 512^2, 1024^2).

Компилятор — pgc++

Профилировщик — “Nsight Systems”

Замер времени проводил используя библиотеку std::chrono

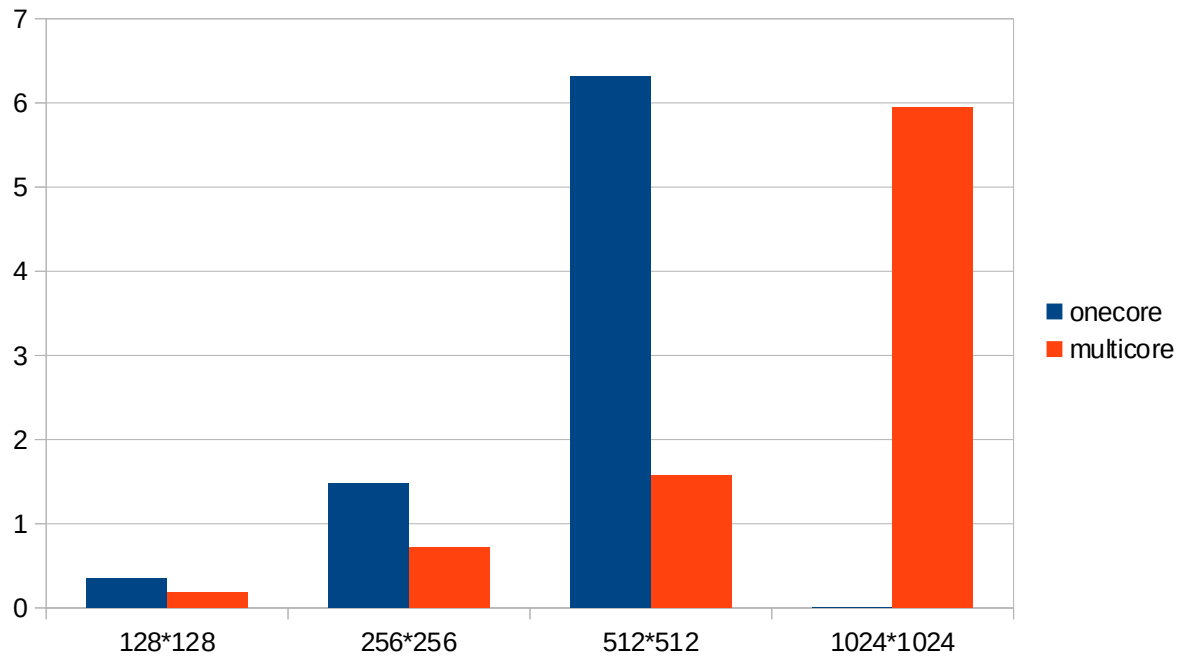
CPU-onecore

Размер сетки	Время выполнения(с)	Точность	Количество итераций
128*128	0.352	0.000001	8167
256*256	1.48	0.000001	8495
512*512	6.32	0.000001	8913

CPU-multicore

Размер сетки	Время выполнения(с)	Точность	Количество итераций
128*128	0.183	0.000001	8167
256*256	0.72	0.000001	8495
512*512	1.572	0.000001	8913
1024*1024	5.941	0.000001	8983

Диаграмма сравнения время работы CPU-one и CPU-multi

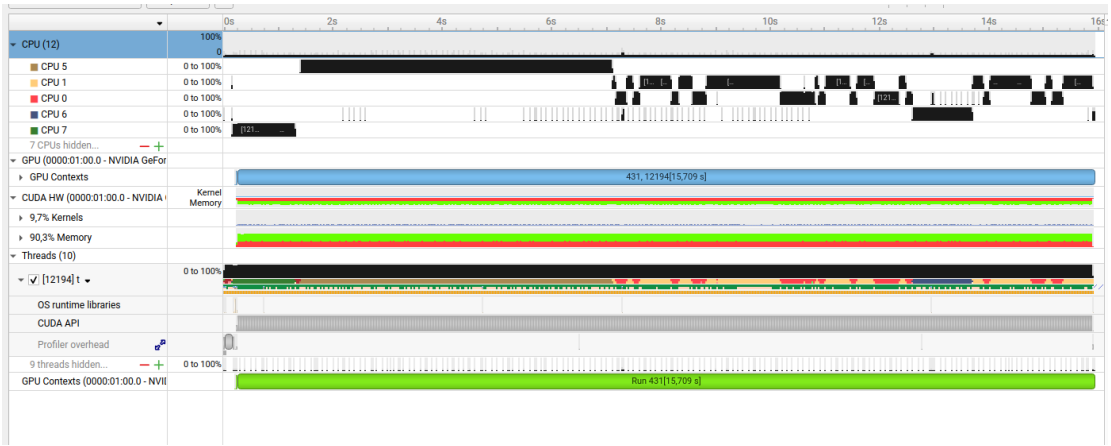


Выполнение на GPU

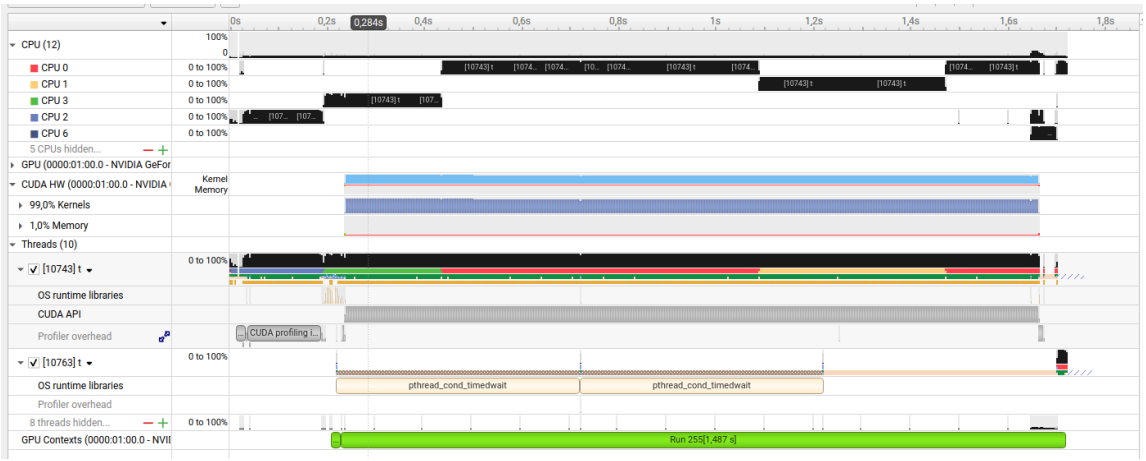
Этапы оптимизации на сетке 512*512

Этап №	Время выполнения	Точность	Максимальное количество итераций	Комментарии
1	15.37	10^-6	1_000_000	Не оптимизированная
2	1.44	10^-6	1_000_000	Убрано лишнее копирование
3	0.47	10^-6	1_000_000	Убрано лишнее копирование между матрицами

1)



2)



3)

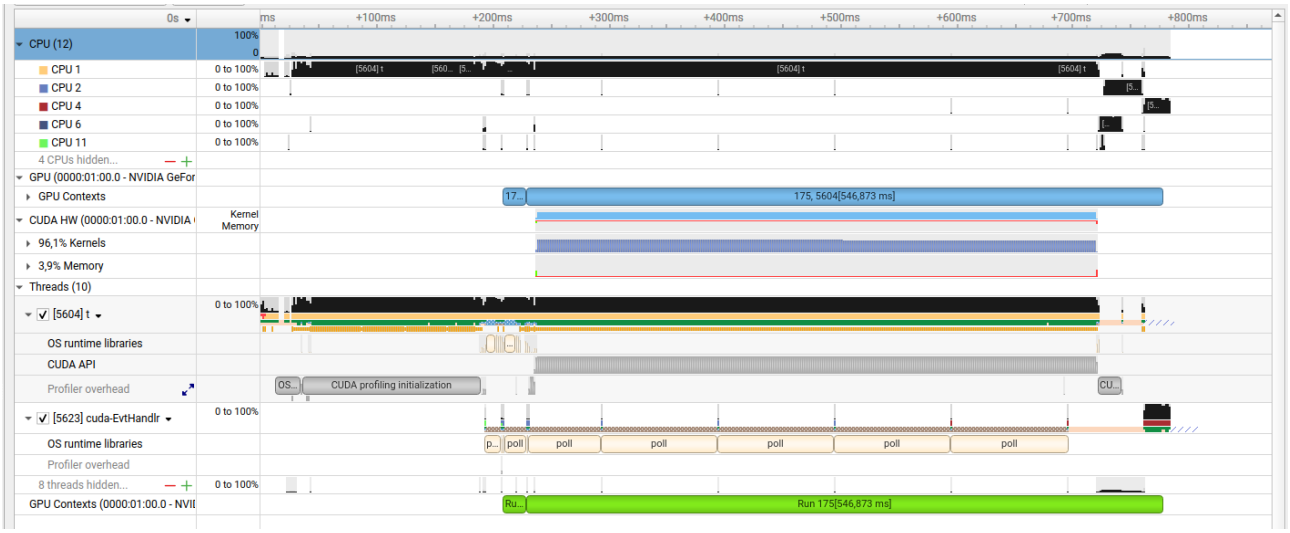
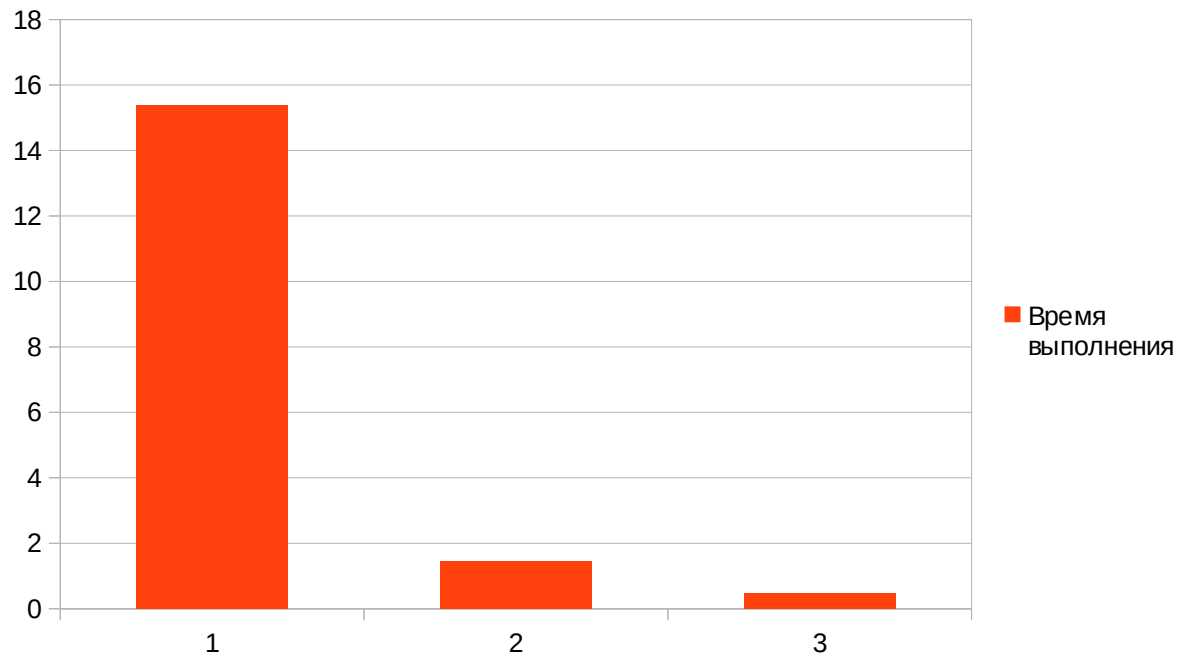

0s
ms
+100ms
+200ms
+300ms
+400ms
+500ms
+600ms
+700ms
+800ms
CPU (12)
100%
0
CPU 1
0 to 100%
CPU 2
0 to 100%
CPU 4
0 to 100%
CPU 6
0 to 100%
CPU 11
0 to 100%
4 CPUs hidden...
GPU (0000:01:00.0 - NVIDIA GeFor
GPU Contexts
175, 5604[546,873 ms]
CUDA HW (0000:01:00.0 - NVIDIA
Kernel
Memory
96,1% Kernels
3,9% Memory
Threads (10)
[5604] t
0 to 100%
OS runtime libraries
CUDA API
Profiler overhead
CUDA profiling initialization
[5623] cuda-EvtHandlr
0 to 100%
OS runtime libraries
poll
Profiler overhead
8 threads hidden...
0 to 100%
GPU Contexts (0000:01:00.0 - NVII
Run 175[546,873 ms]

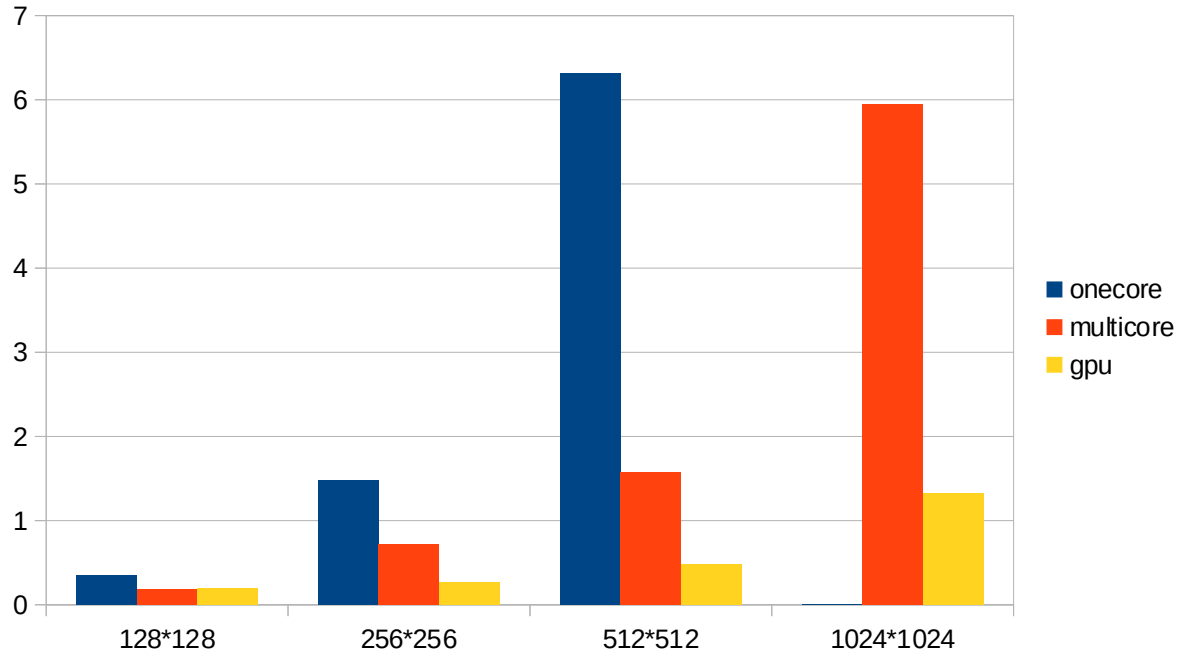
Диаграмма оптимизации



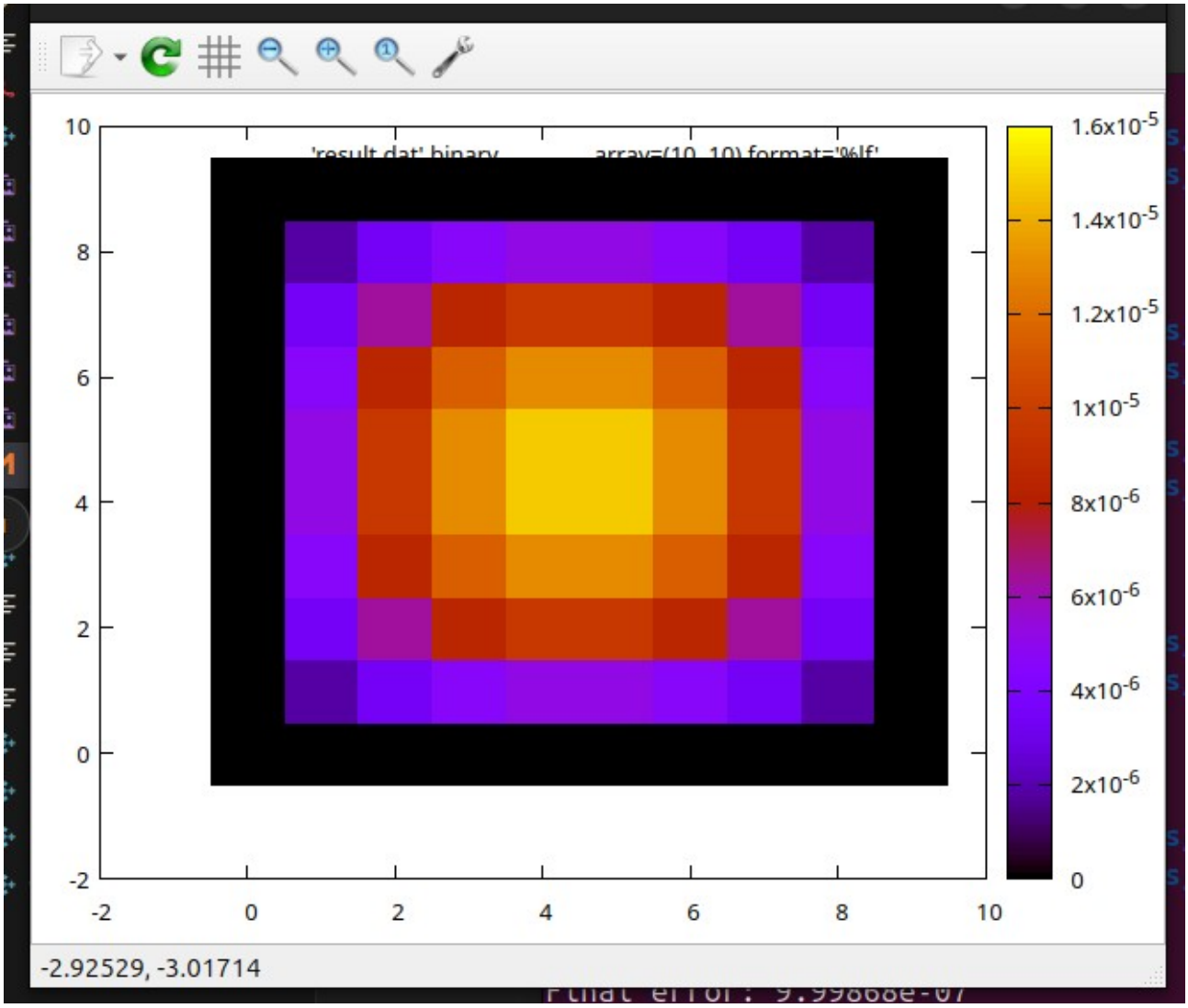
GPU – оптимизированный вариант

Размер сетки	Время выполнения	Точность	Количество итераций
128 * 128	0.20	10^-6	8167
256 * 256	0.27	10^-6	8495
512* 512	0.48	10^-6	8913
1024 * 1024	1.33	10^-6	8983

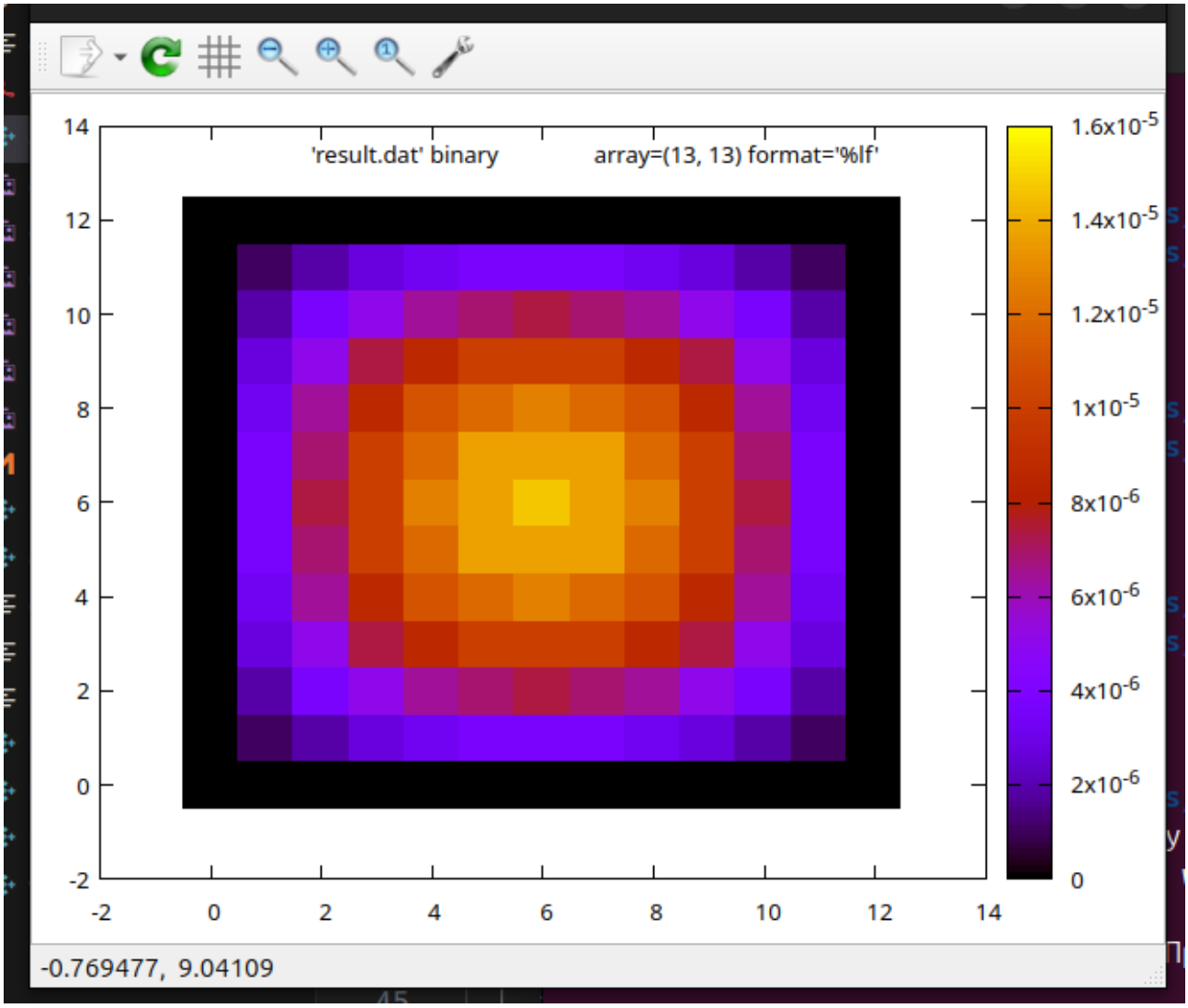
Onecore, multicore and gpu сравнение



10x10


'result.dat' binary array=(10,10) format='%lf'
10
8
6
4
2
0
-2
-2
0
2
4
6
8
10
1.6x10-5
1.4x10-5
1.2x10-5
1x10-5
8x10-6
6x10-6
4x10-6
2x10-6
0
-2.92529, -3.01714

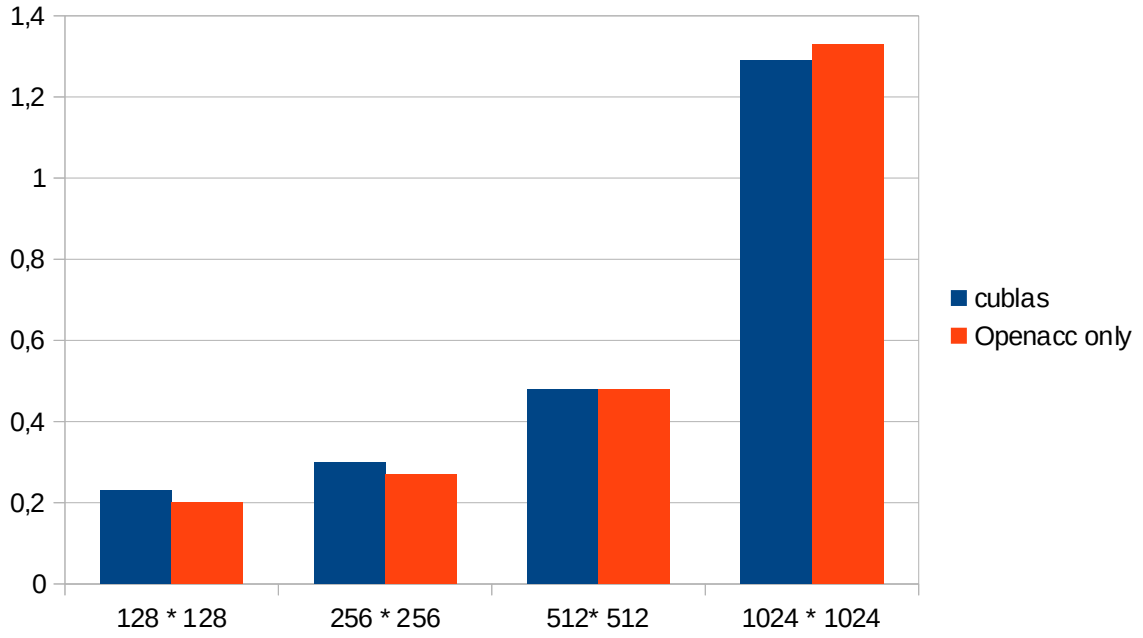
13x13

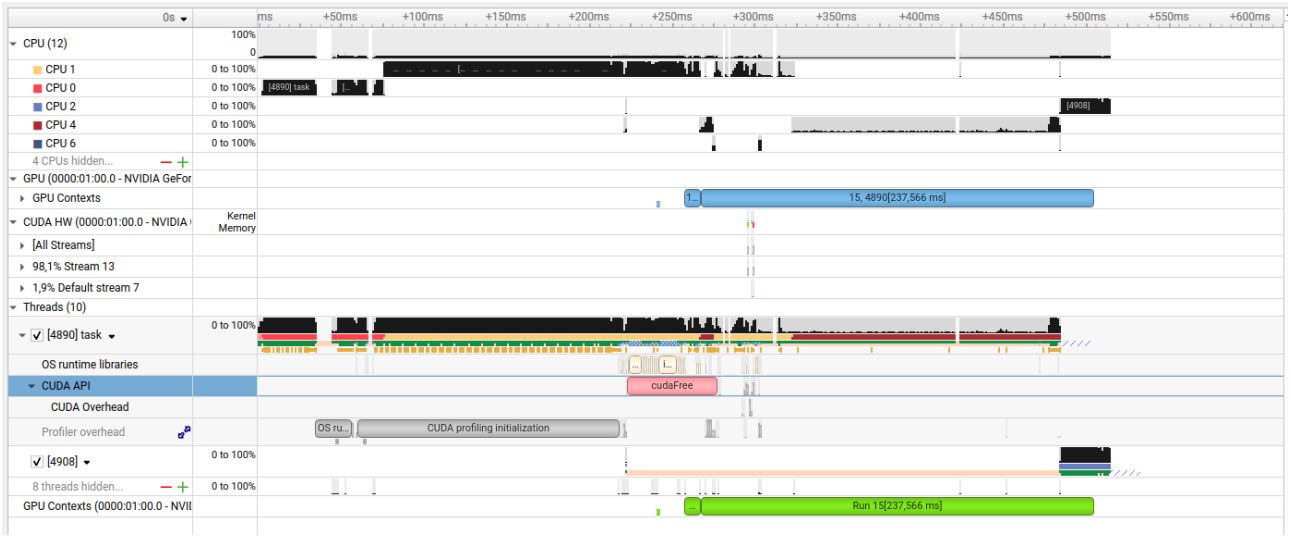

'result.dat' binary
array=(13, 13) format='%lf'
14
12
10
8
6
4
2
0
-2
-2
0
2
4
6
8
10
12
14
1.6x10-5
1.4x10-5
1.2x10-5
1x10-5
8x10-6
6x10-6
4x10-6
2x10-6
0
-0.769477, 9.04109

Оптимизированная версия программы показывает лучшую производительность при сохранении той же точности результатов. Основные улучшения достигнуты за счет эффективного использования параллелизма на GPU.

Cublas:

Размер сетки	Время выполнения	Точность	Количество итераций
128 * 128	0.23	10^-6	8167
256 * 256	0.3	10^-6	8495
512* 512	0.48	10^-6	8913
1024 * 1024	1.29	10^-6	8983

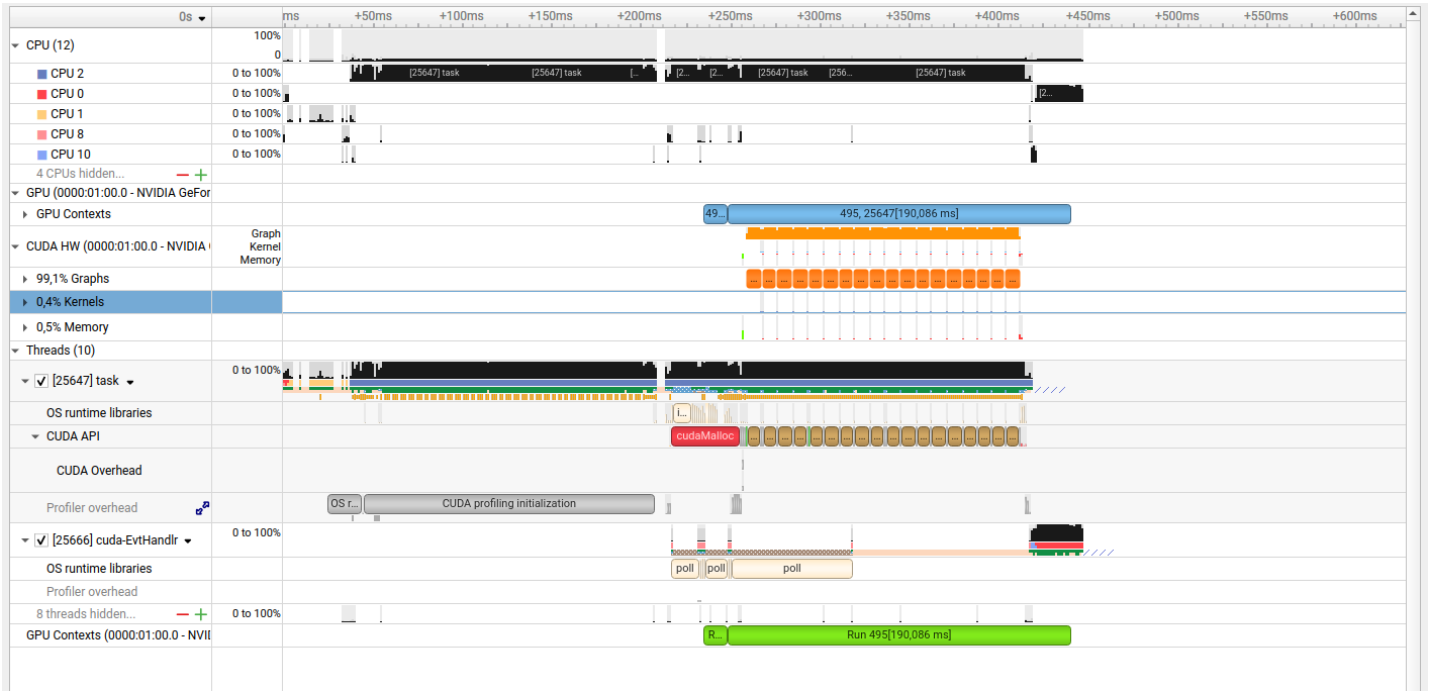


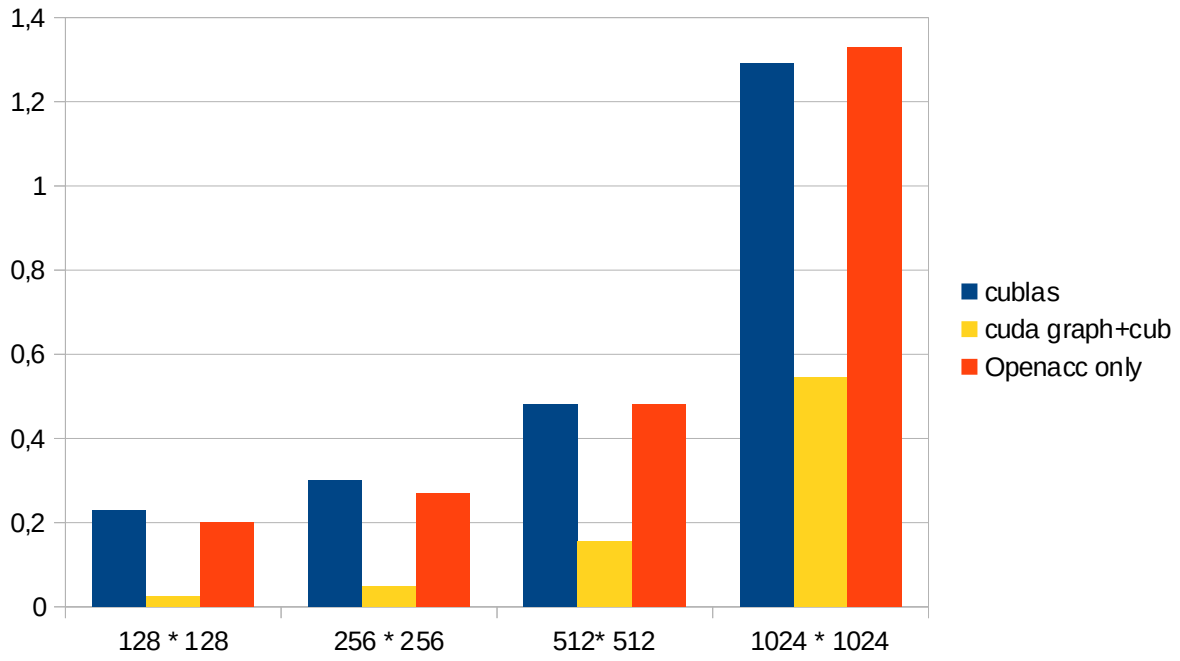


Реализация через cuBLAS работает примерно также как и на openacc, с увеличением размера матрицы начинает работать чуть быстрее

Оптимизированная с помощью cuda graph и cub

Размер сетки	Время выполнения	Точность	Количество итераций
128 * 128	0.024	10^-6	8500
256 * 256	0.049	10^-6	9000
512* 512	0.156	10^-6	9000
1024 * 1024	0.546	10^-6	9000





Программа с cuda graph и cub оказалась самой быстрой.