Теория параллелизма

Отчёт
Задача №7

Выполнил 23932, Карпачев Дмитрий Александрович

21.05.2025

Цель: попробовать на практике использрвание openACC на примере решения уравнения теплопроводности разностной схемой, протестировать ускорение на нескольких ядрах cpu и на gpu.

Компилятор: pgc++
Профилировщик: Nsight Systems
Замеры времени: std::chrono::high_resolution_clock (итоговое время в секундах)

Выполнение на CPU

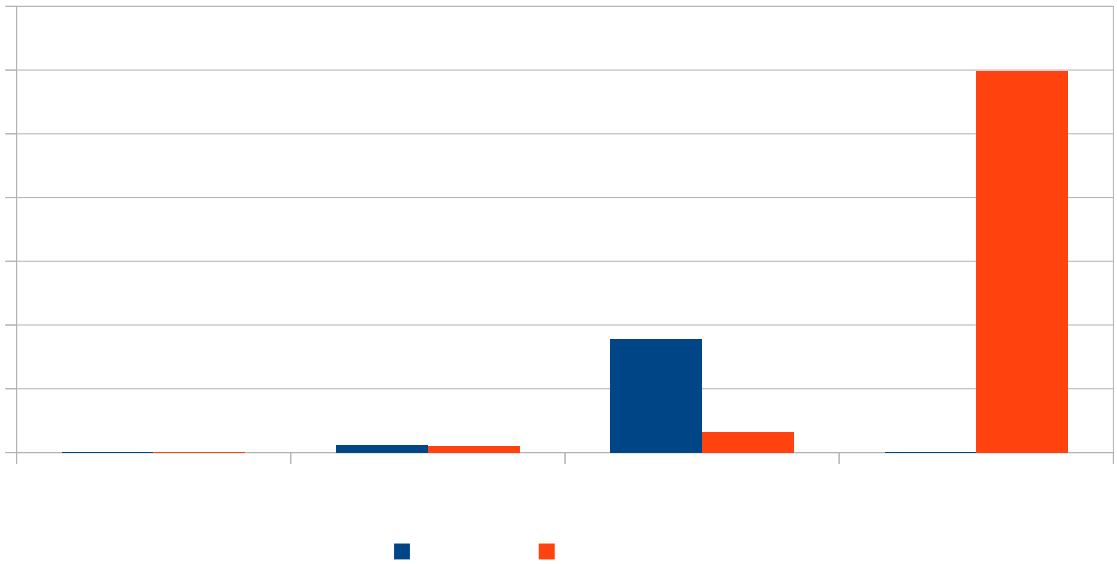
CPU-onecore

Размер сетки	Время выполнения	Точность	Количество итераций
128*128	0.796 сек.	7.53244e-07	31000
256*256	11.502 сек.	9.91241e-07	103000
512*512	178.508 сек.	9.92435e-07	340000
1024*1024	>5 мин.	-	1000000

CPU-multicore

Размер сетки	Время выполнения	Точность	Количество итераций
128*128	1.182 сек.	7.53244e-07	31000
256*256	4.059 сек.	9.91241e-07	103000
512*512	30.562 сек.	9.92435e-07	340000
1024*1024	262.967 сек.	1.36929e-06	1000000

Диаграмма сравнения CPU-one и CPU-multi

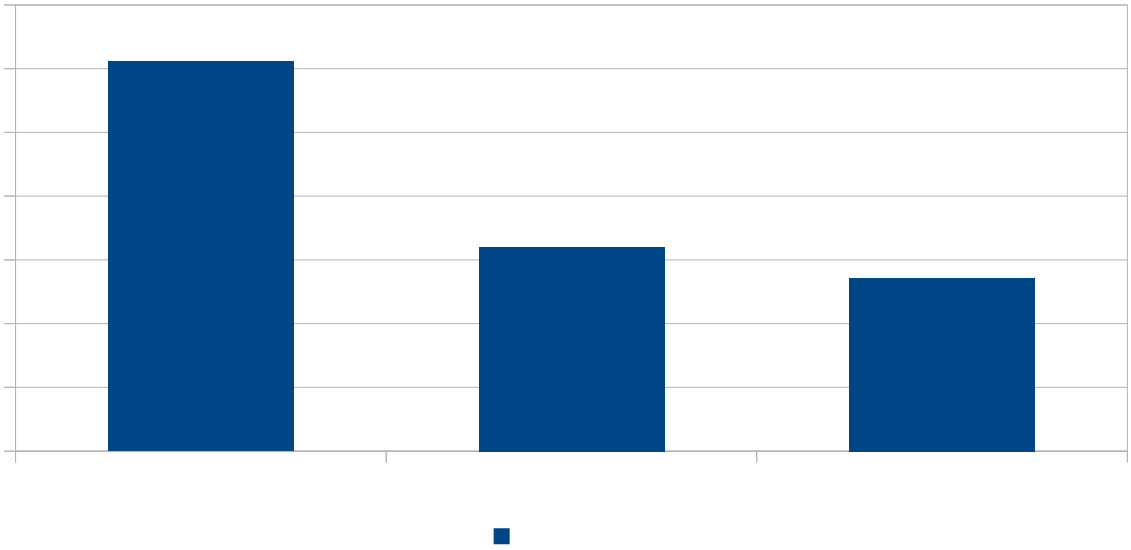


Выполнение на GPU

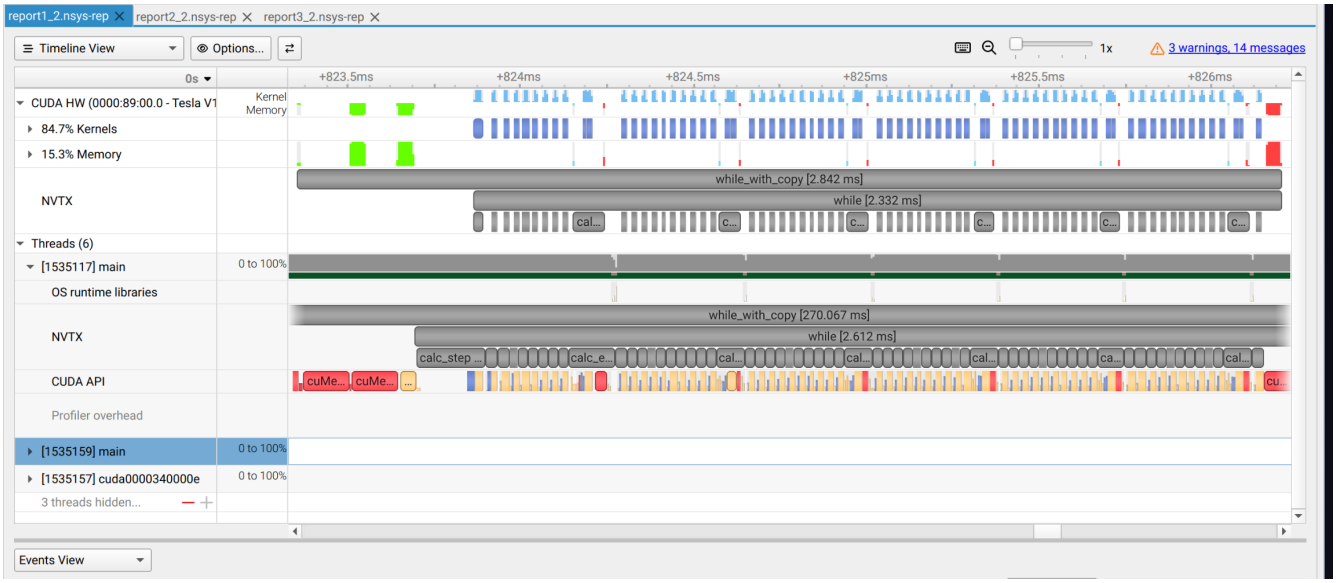
Этапы оптимизации на сетке 512*512

Этап №	Время выполнения	Точность	Максимальное количество итераций	Комментарии
1	30.562 сек.	9.92435e-07	1000000	Базовое применение openACC для ускорения рассчётов
2	15.97 сек.	9.92435e-07	1000000	Замена ленивого swap на циклах на std::swap
3	13.562 сек.	9.92829e-07	1000000	Использование async при использовании openACC

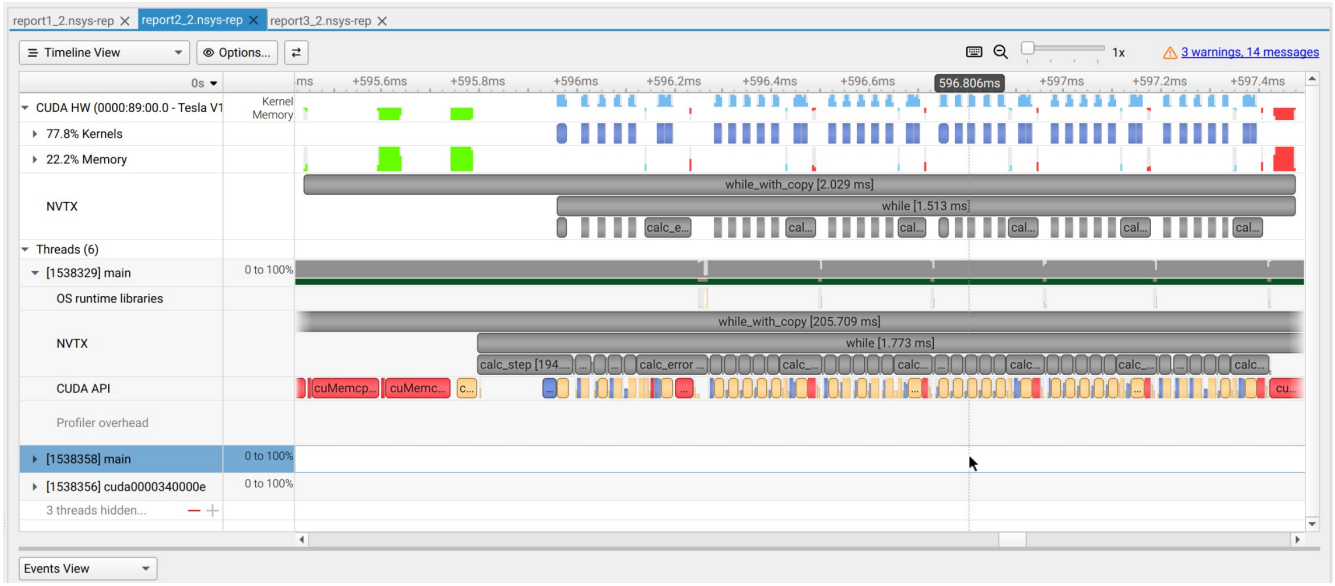
Диаграмма оптимизации



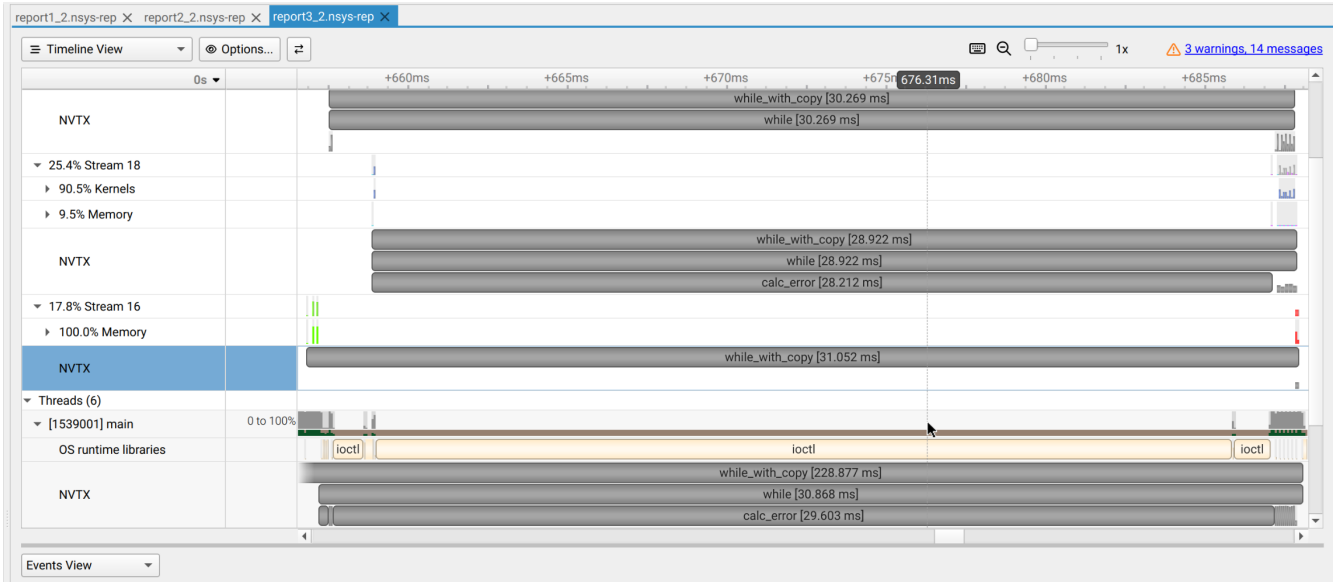
Профилирование этап 1

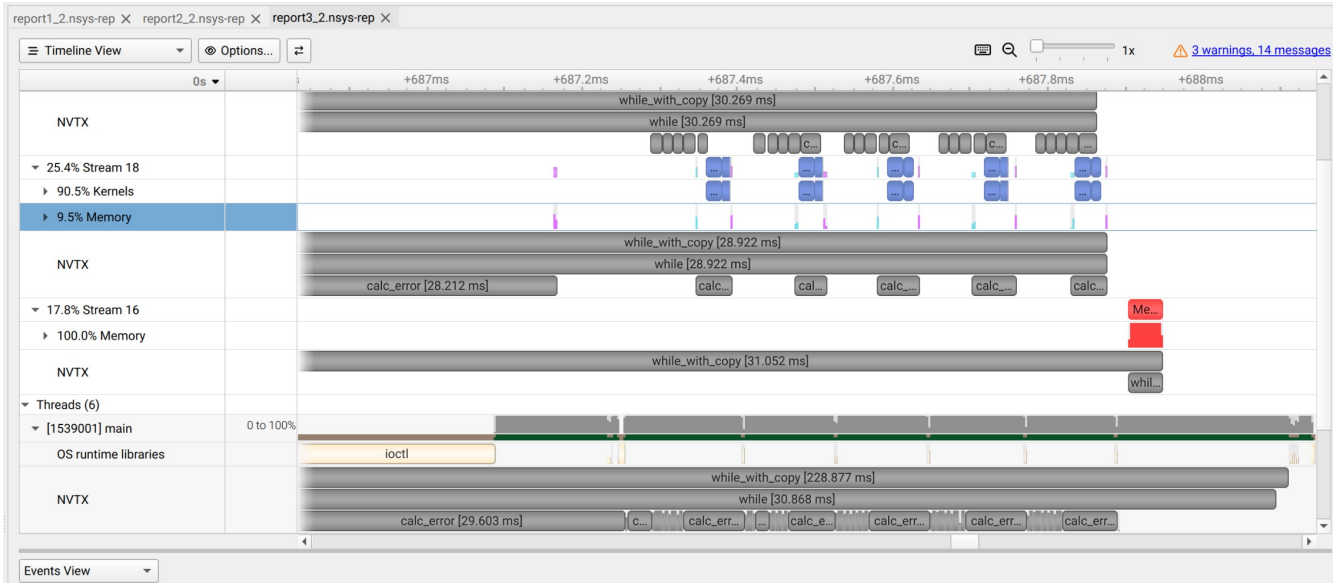


Профилирование этап 2



Профилирование этап 3





GPU - оптимизированный вариант

Размер сетки	Время выполнения	Точность	Количество итераций
128*128	0.485 сек.	7.581e-07	32000
256*256	1.463 сек.	9.92822e-07	104000
512*512	13.562 сек.	9.92829e-07	341000
1024*1024	128.804 сек.	1.37589e-06	1000000

Вывод матрицы 10*10

```
10.000 11.111 12.222 13.333 14.444 15.556 16.667 17.778 18.889 20.000
11.111 12.222 13.333 14.444 15.556 16.667 17.778 18.889 20.000 21.111
12.222 13.333 14.444 15.556 16.667 17.778 18.889 20.000 21.111 22.222
13.333 14.444 15.556 16.667 17.778 18.889 20.000 21.111 22.222 23.333
14.444 15.556 16.667 17.778 18.889 20.000 21.111 22.222 23.333 24.444
15.556 16.667 17.778 18.889 20.000 21.111 22.222 23.333 24.444 25.556
16.667 17.778 18.889 20.000 21.111 22.222 23.333 24.444 25.556 26.667
17.778 18.889 20.000 21.111 22.222 23.333 24.444 25.556 26.667 27.778
18.889 20.000 21.111 22.222 23.333 24.444 25.556 26.667 27.778 28.889
20.000 21.111 22.222 23.333 24.444 25.556 26.667 27.778 28.889 30.000
```

Визуализация матрицы 128*128 после рассчётов

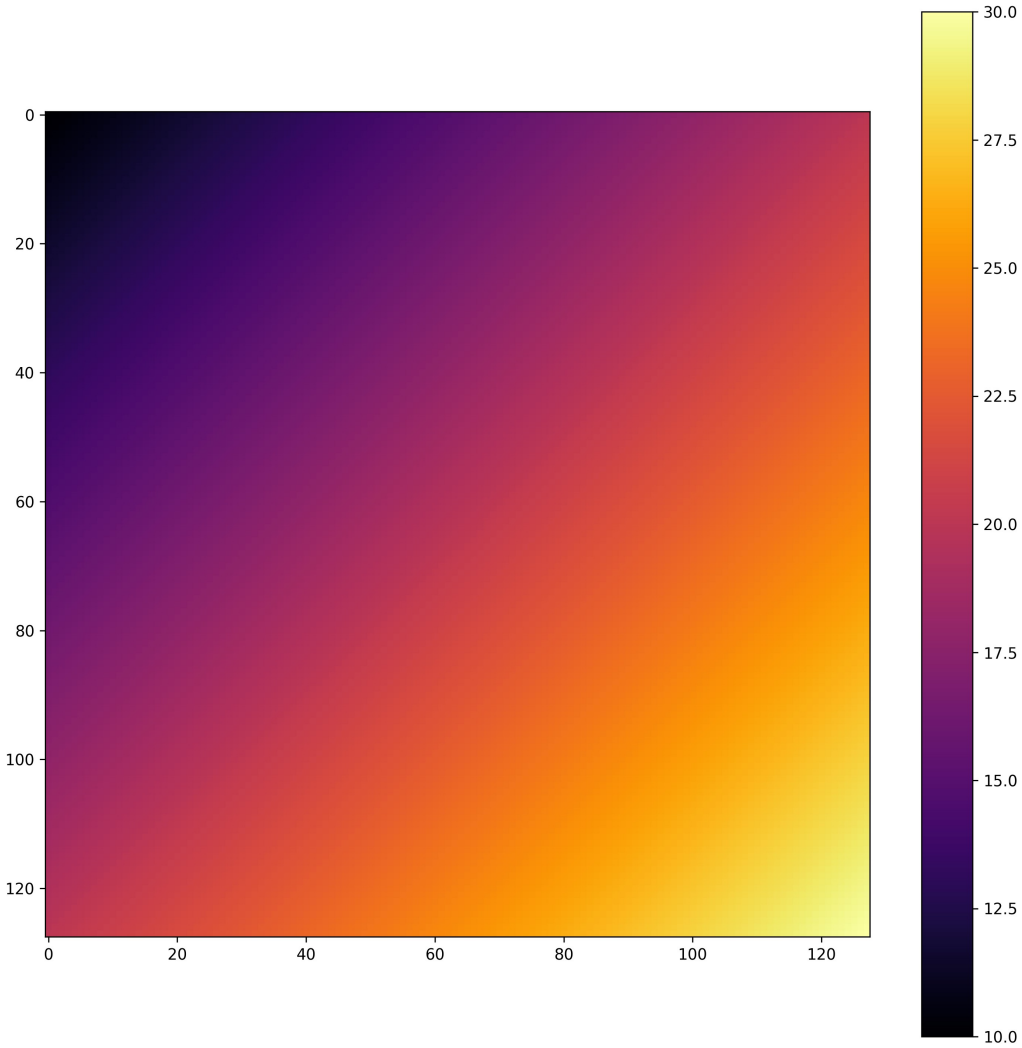
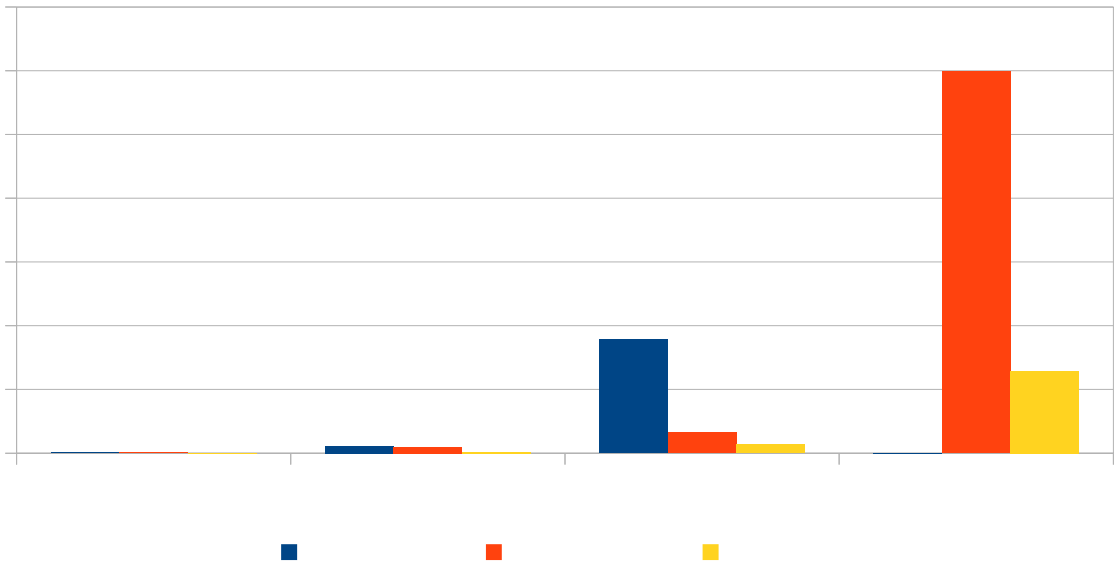
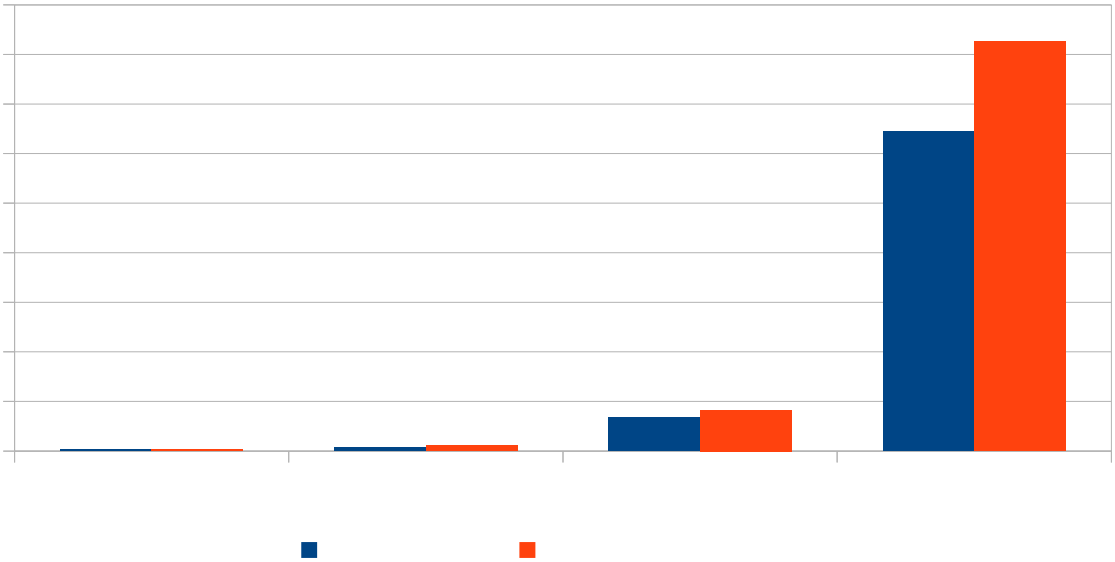


Диаграмма сравнения времени работы CPU-one, CPU-multi, GPU(оптимизированный вариант) для разных размеров сеток



Замена редукции на функции библиотеки cuBLAS

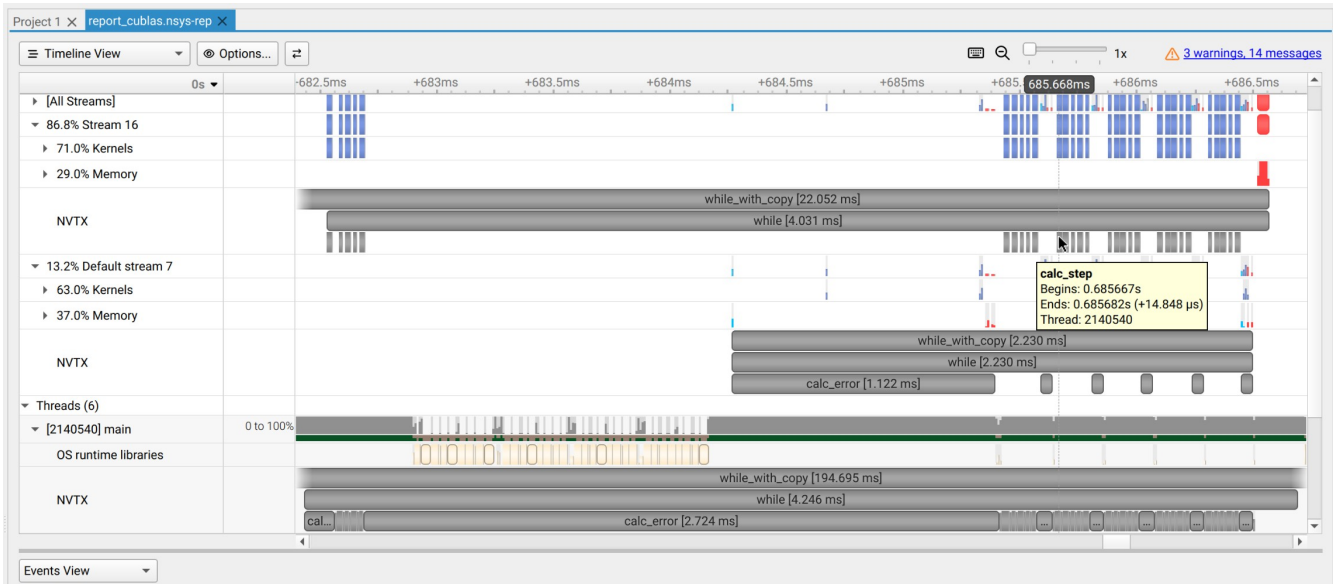
Сравнение программ с cuBLAS и без



cuBLAS реализация

Размер сетки	Время выполнения	Точность	Количество итераций
128*128	0.76 сек.	7.53244e-07	31000
256*256	2.267 сек.	9.91241e-07	103000
512*512	16.576 сек.	9.92435e-07	340000
1024*1024	165.194 сек.	1.36929e-06	1000000

Профилировка



Вывод:

1) Ожидаемо, многоедарный запуск лучше одноядерного, а запуск на gpu значительно обгоняет многоядерный запуск на cpu. Часть возможной производительности на gpu тратится при передаче данных с device на host, поэтому данные операции следует минимизировать.
Также операция редукции по ошибке (max:error) занимает довольно много времени.

2) Использование функций cuBLAS может ускорить выполнения программы на больших наборах данных, и при полной замене оптимизации на cuBLAS, т.к. в данный момент необходимость копировать данные внутри графической памяти замедляет рассчёт максимальной ошибки.