Теория Параллелизма

Отчет
Уравнение теплопроводности
cuda

Выполнил 21932, Бабенко Егор Степанович

04.04.2022

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: *nvcc*

Для компиляции версии с cuda использовалась:

- nvcc -O2 t4.cu -o bin/t4_GPU

Для дополнительной профилировки: -D NVPROF_

Используемый профилировщики: *nvprof, nsys*

Nsys использовался с CUDA trace, а также NVTX trace.

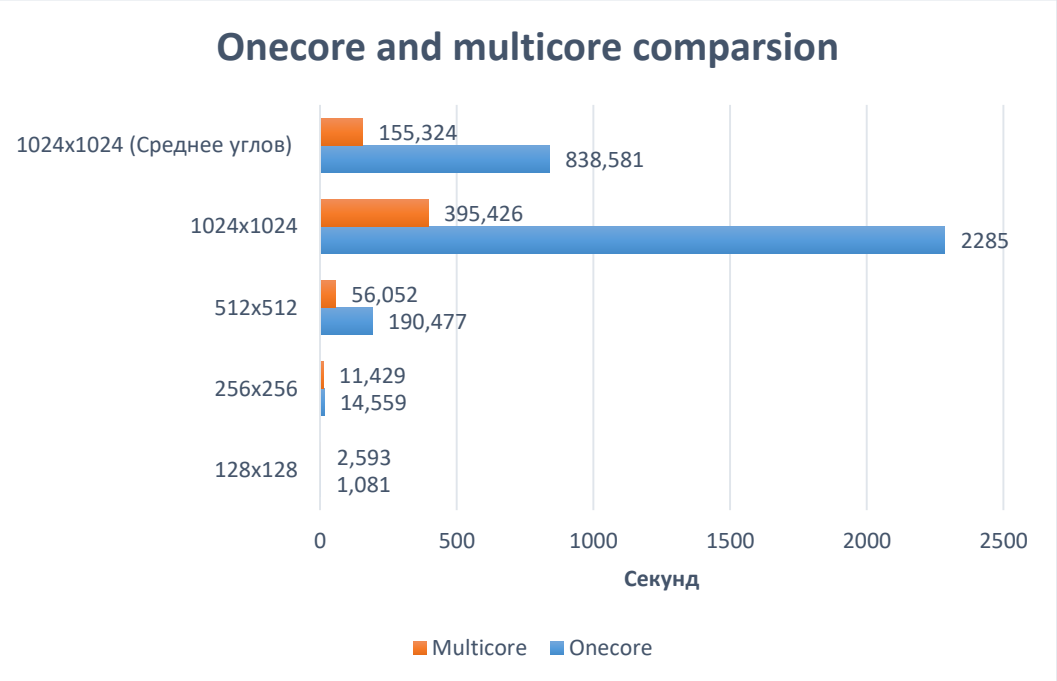
Замер времени работы всей программы производился с помощью команды time. Время высчитывалось как среднее между пятью запусками.

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)

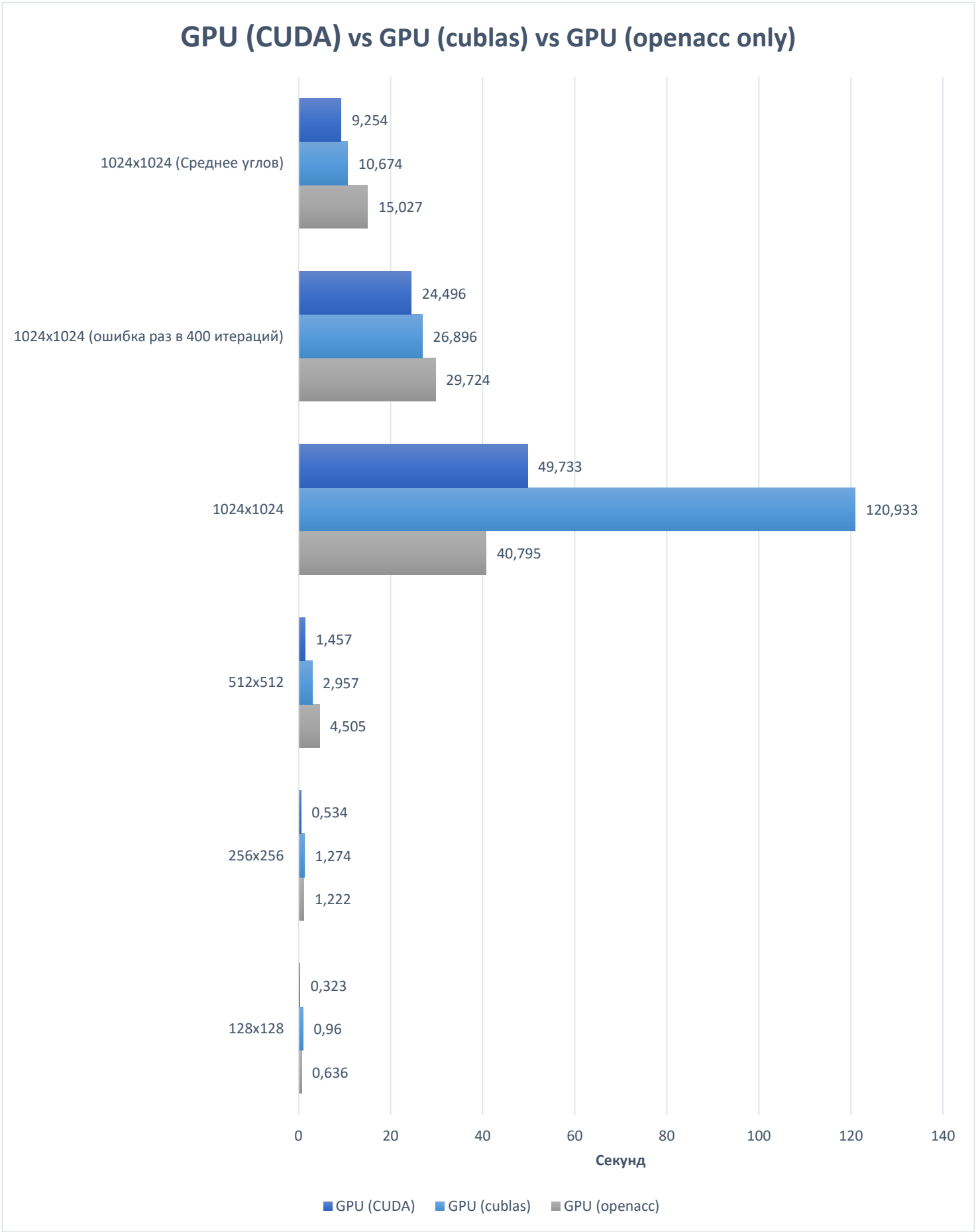
Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	0.636	9.93271e-07	30096
256x256	1.222	9.98489e-07	102904
512x512	4.505	9.99134e-07	339644
1024x1024*(расчет ошибки каждую итерацую*	40.795	1.36929e-06	1000000
1024x1024 *(расчет ошибки каждые 400 итераций)*	29.724	1.36929e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	15.027	9.99663e-07	364648

GPU-optimized (cublas)

Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	0.960	9.93271e-07	30096
256x256	1.274	9.98489e-07	102904
512x512	2.957	9.99134e-07	339644
1024x1024*(расчет ошибки каждую итерацую*	120.933	1.36973e-06	1000000
1024x1024 *(расчет ошибки каждые 400 итераций)*	26.896	1.36973e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	10.674	9.99663e-07	364648

GPU-optimized (CUDA)

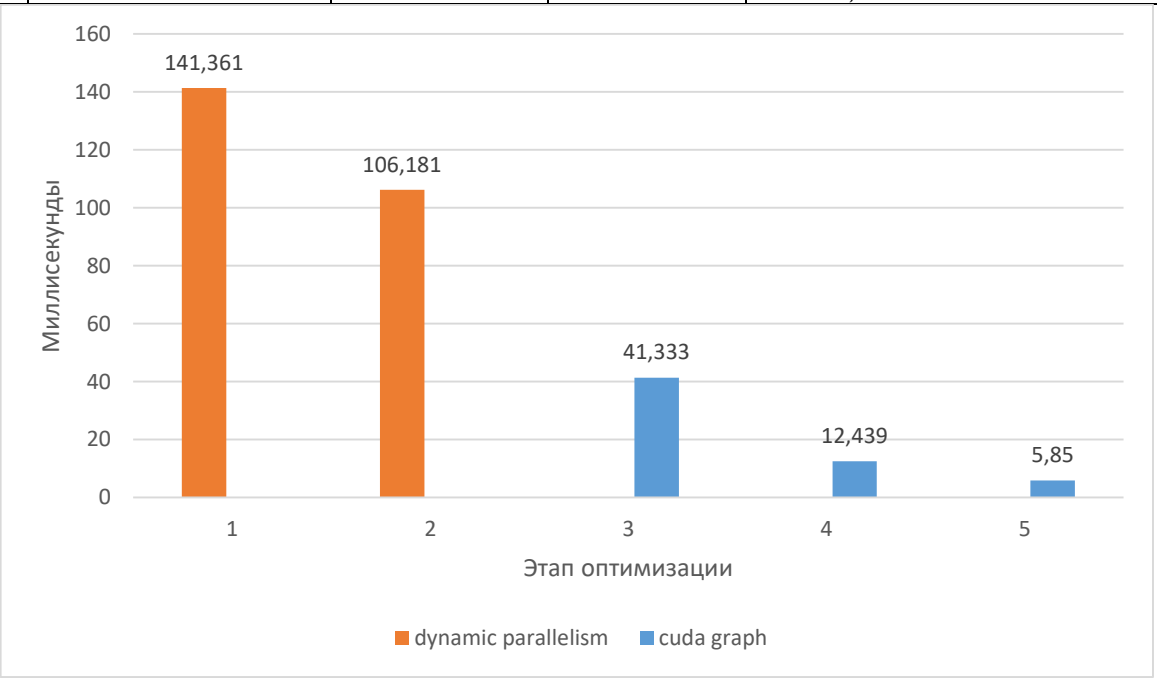
Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	0.323	9.05043e-07	30400
256x256	0.534	9.76309e-07	103200
512x512	1.457	9.99965e-07	339600
1024x1024*(расчет ошибки каждую итерацую [без использования cuda graph])*	49.733	1.36929e-06	1000000
1024x1024 *(расчет ошибки каждые 400 итераций)*	24.496	1.36929e-06	1000000
1024x1024 *(среднее значение углов сетки как стартовое значение)*	9.254	9.97873e-07	364800


GPU (CUDA) vs GPU (cublas) vs GPU (openacc only)
1024x1024 (Среднее углов)
9,254
10,674
15,027
1024x1024 (ошибка раз в 400 итераций)
24,496
26,896
29,724
1024x1024
49,733
120,933
40,795
512x512
1,457
2,957
4,505
256x256
0,534
1,274
1,222
128x128
0,323
0,96
0,636
0
20
40
60
80
100
120
140
Секунд
GPU (CUDA)
GPU (cublas)
GPU (openacc)

Этапы оптимизации кода на GPU (CUDA)

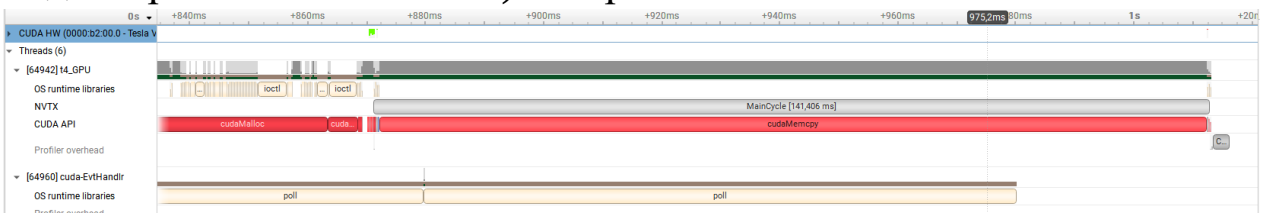
Этапы оптимизации на сетке 512x512

№	Время выполнения основного цикла	Ошибка	Кол-во итераций	Комментарий
1	141,361 мс	0.0105525	1000	*Программа написанная с использованием динамического параллелизма (reduction через DeviceReduce)*
2	106,181 мс	0.0105525	1000	*Программа написанная с использованием динамического параллелизма (reduction через BlockReduce и редукция между блоками через DeviceReduce)*
3	41,333 мс	0.0105524	1000	*Реализация программы через вызов с хоста основной функции и редукции через DeviceReduce*
4	12,439 мс	0.008775	1200*	*Реализация через CUDA graph (создание графа на последовательное исполнение 400 обновлений сетки), расчет ошибки каждые 400 итераций через DeviceReduce*
5	5,850 мс	0.008775	1200*	*Использование reduction с помощью редукции по блокам BlockReduce и последующей редукции самих блоков с помощью DeviceReduce.*

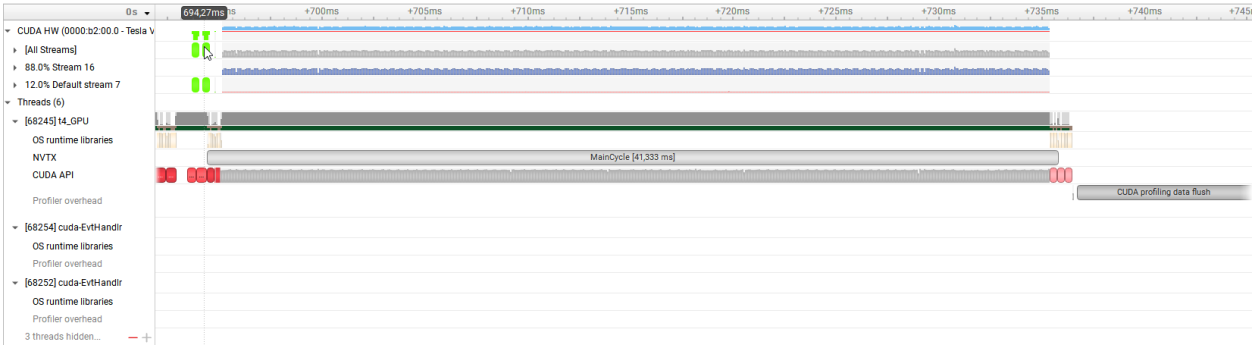


(*) – т.к. граф составлялся для последовательного выполнения 400 операций

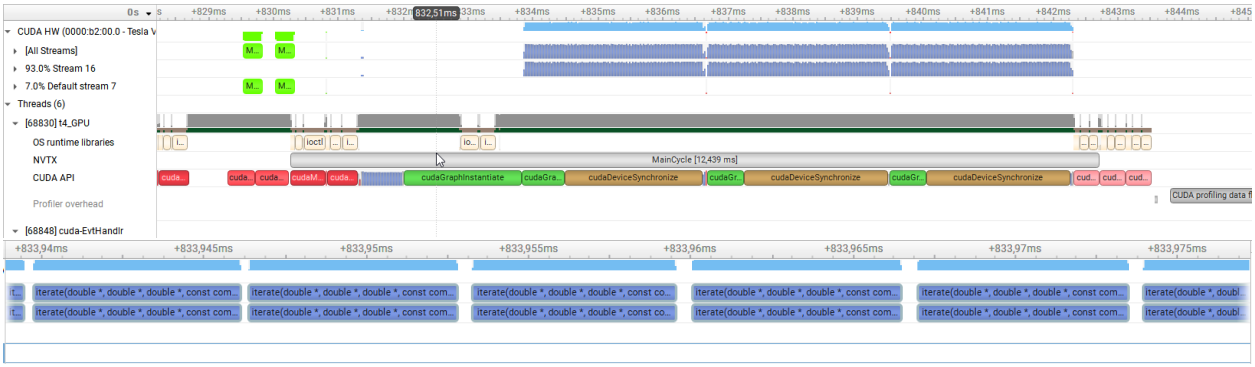
Первый этап (нет информации по использованию, т.к. nsys не поддерживает профилирование с динамическим параллелизмом на видеокартах с CC выше 6.1). Второй аналогично



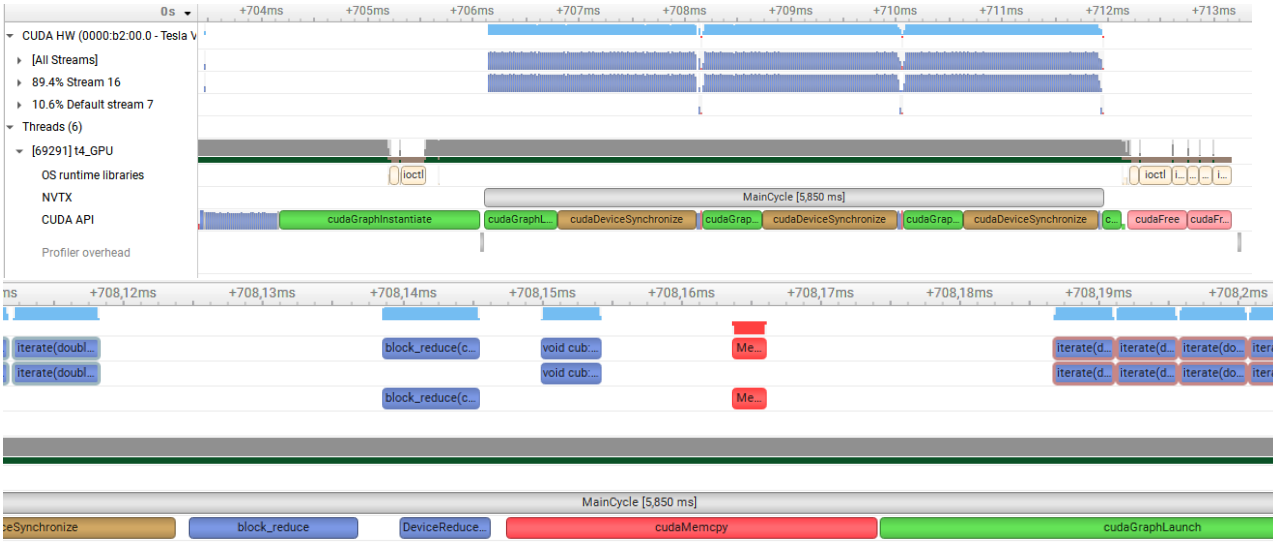
Третий этап



Четвертый этап



Пятый этап



Вывод:

Использование cuda_runtime помогает более грамотно использовать ресурсы GPU. Что в итоге позволяет добиться большей производительности, но и требует большего времени на разработку. Поэтому если необходимо быстро разработать MVP, то стоит использовать openacc/cublas. Тогда как за счет ручного управления можно достичь куда большего, чем при использовании оптимизации компилятором через прагмы openacc. А также использование CUDA graph очень благотворно влияет на производительность программы, за счет того, что уменьшаются задержки между запусками ядер.

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