Теория Параллелизма

Отчет

Уравнение теплопроводности

MPI/NCCL

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: *mpic++*

Для компиляции различных версий использовались:

- mpic++ t5_MPI.cu –O2
- mpic++ t5_nccl.cu –O2 -lnccl

Для дополнительной профилировки: -D NVPROF_

Также для всех замеров с MPI и nccl представленных ниже использовался флаг –D THREAD_ANALOG_MODE (Изменяет структуру нитей и блоков для вызова функций видеокарты)

Используемый профилировщики: *nvprof, nsys*

Nsys использовался с CUDA trace, MPI, а также NVTX trace.

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

GPU (openacc)

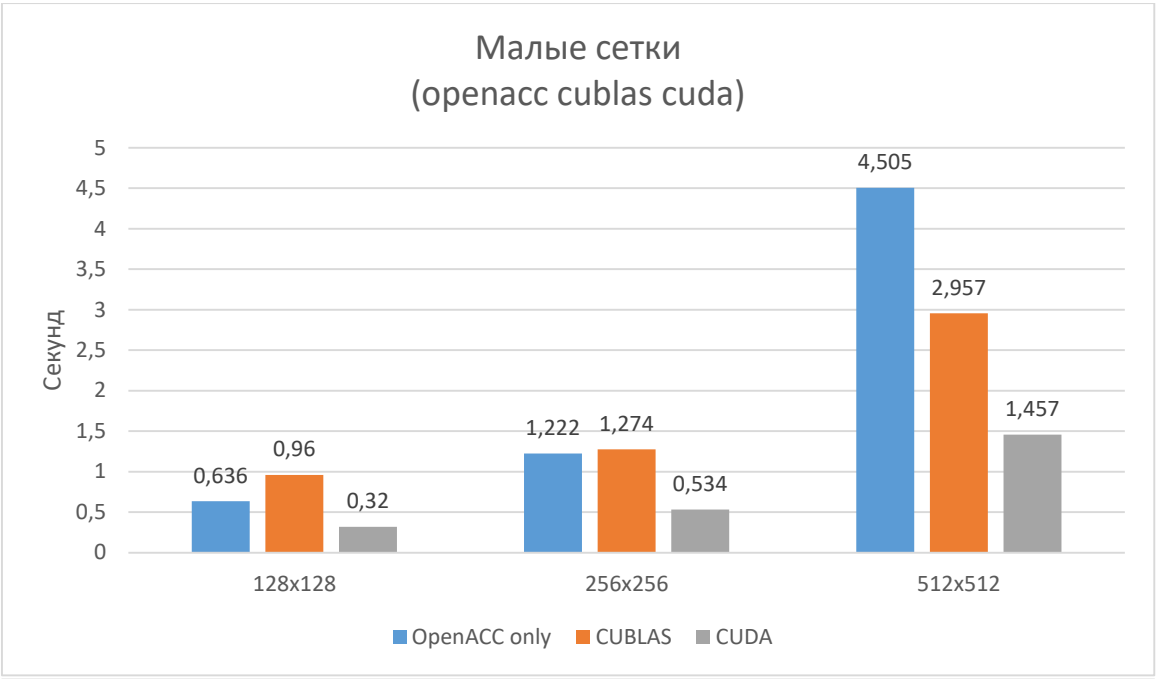
Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	0.636	9.93271e-07	30096
256x256	1.222	9.98489e-07	102904
512x512	4.505	9.99134e-07	339644
1024x1024*(расчет ошибки каждую итерацую*	40.795	1.36929e-06	1000000
1024x1024 *(расчет ошибки каждые 400 итераций)*	29.724	1.36929e-06	1000000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

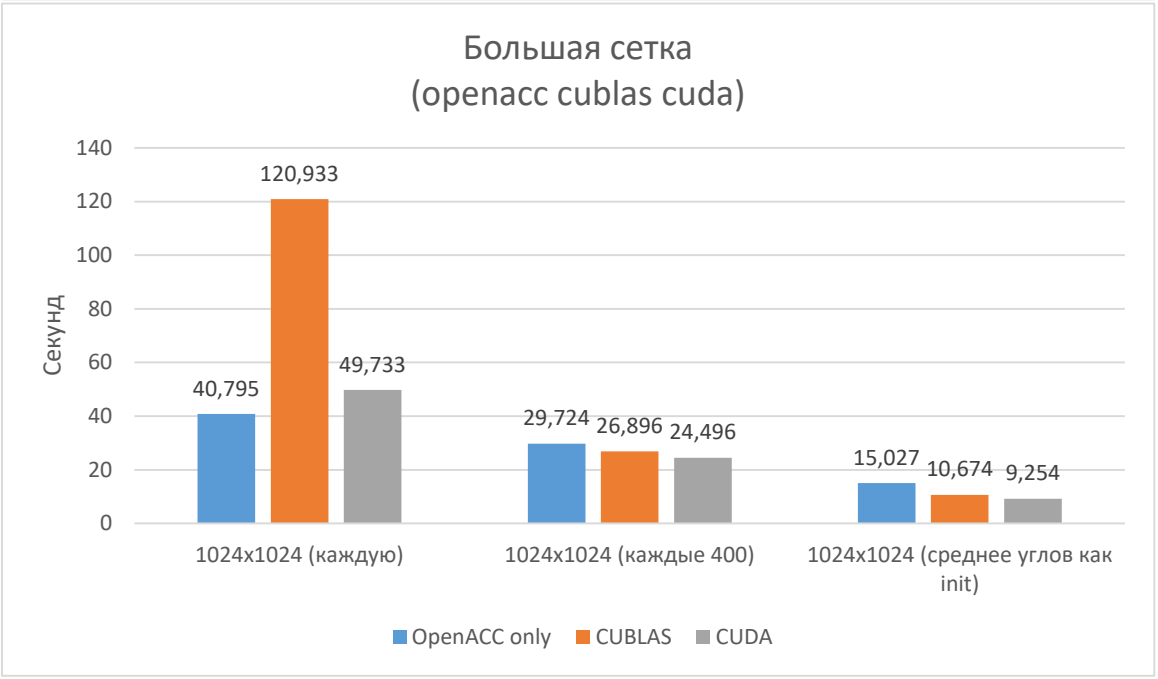
GPU (openacc + cublas)

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 (CUDA)

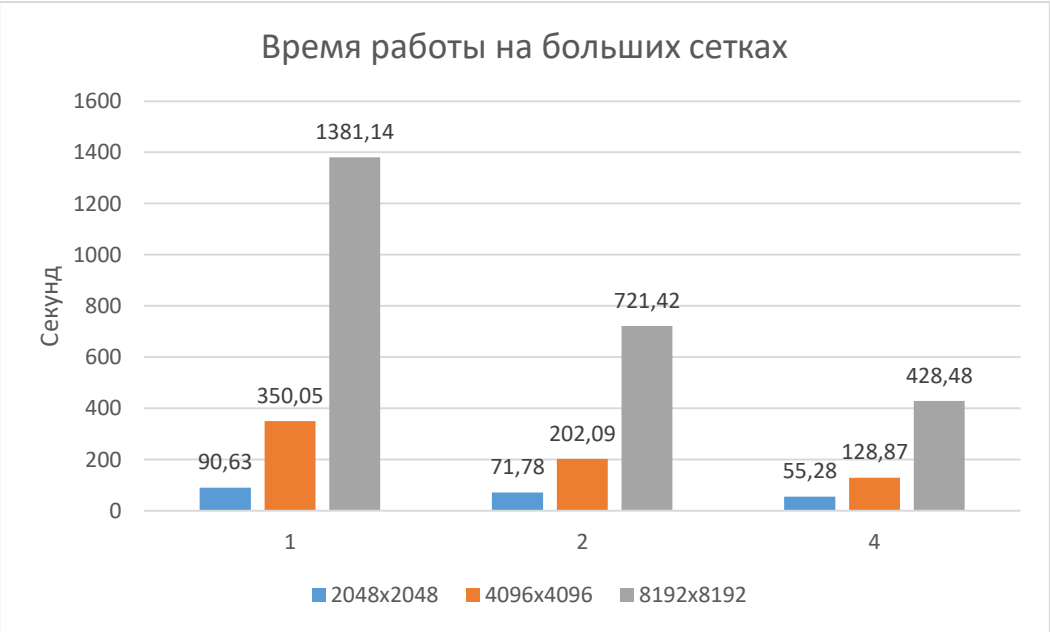
Размер сетки	Время выполнения (сек)	Ошибка	Количество итераций
128x128	0.323	9.05043e-07	30400
256x256	0.534	9.76309e-07	103200
512x512	1.457	9.99965e-07	339600
1024x1024*(расчет ошибки каждую итерацую [без использования cuda graph])*	49.733	1.36929e-06	1000000
1024x1024 *(расчет ошибки каждые 400 итераций)*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


Малые сетки
(openacc cublas cuda)
Секунд
5
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
0,636
0,96
0,32
1,222
1,274
0,534
4,505
2,957
1,457
128x128
256x256
512x512
OpenACC only
CUBLAS
CUDA


Большая сетка
(openacc cublas cuda)
Секунд
140
120
100
80
60
40
20
0
40,795
120,933
49,733
29,724
26,896
24,496
15,027
10,674
9,254
1024x1024 (каждую)
1024x1024 (каждые 400)
1024x1024 (среднее углов как init)
OpenACC only
CUBLAS
CUDA

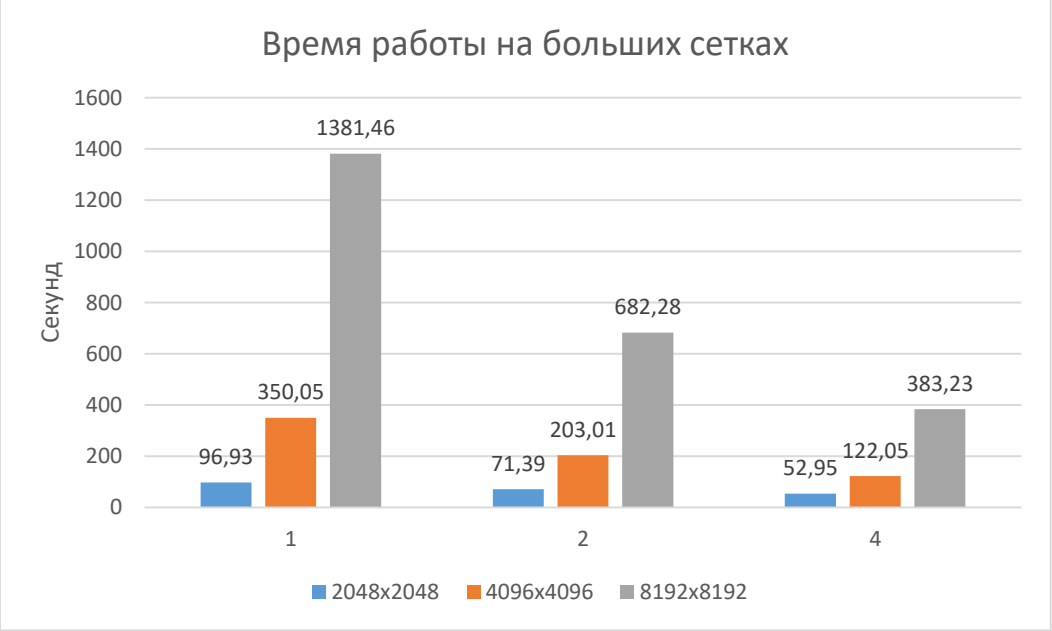
GPU (MPI only)

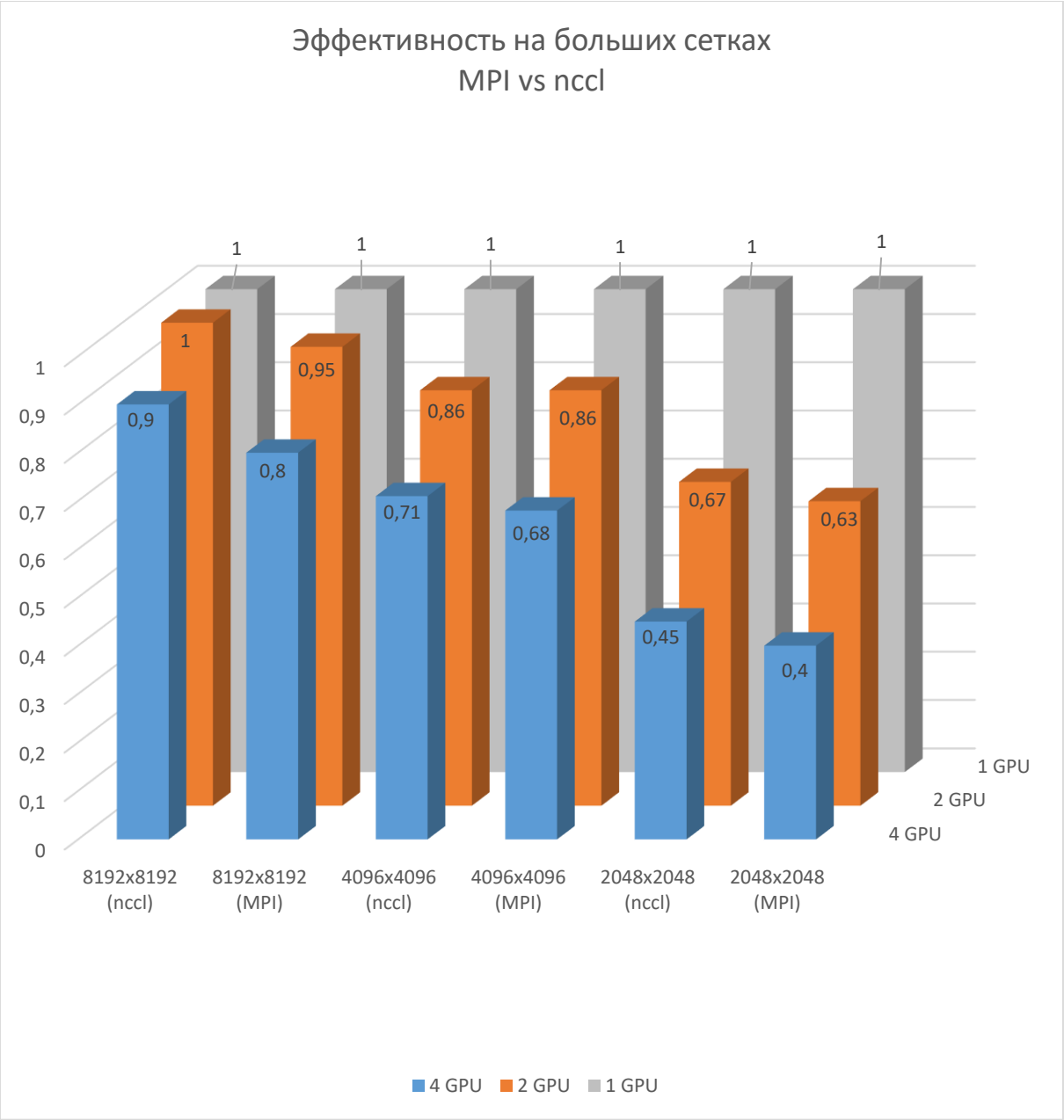
Размер сетки	Кол-во процессов	Время выполнения (сек)	Ошибка	Количество итераций
128x128	1	0,69	9.20191e-07	30400
	2	1,08		
	4	1,96		
256x256	1	0,96	9.92452e-07	103200
	2	1,89		
	4	3,21		
512x512	1	2,47	9.97394e-07	339600
	2	6,45		
	4	7,79		
1024x1024	1	25,54	1.39973e-06	1000000
	2	29,31		
	4	22,43		
2048x2048	1	90,63	1.15583e-05	1000000
	2	71,78		
	4	55,28		
4096x4096	1	350,05	9.82094e-06	1000000
	2	202,09		
	4	128,87		
8192x8192	1	1381,14	1.03114e-05	1000000
	2	721,42		
	4	428,48		



GPU (MPI + nccl)

Размер сетки	Кол-во процессов	Время выполнения (сек)	Ошибка	Количество итераций
128x128	1	0,91	9.05043e-07	30400
	2	1,91		
	4	2,81		
256x256	1	1,22	9.92452e-07	103200
	2	3,48		
	4	4,54		
512x512	1	5,81	9.97394e-07	339600
	2	9,24		
	4	10,28		
1024x1024	1	26,09	1.36929e-06	1000000
	2	35,74		
	4	27,46		
2048x2048	1	96,93	1.15583e-05	1000000
	2	71,39		
	4	52,95		
4096x4096	1	350,07	9.82094e-06	1000000
	2	203,01		
	4	122,05		
8192x8192	1	1381,46	1.03114e-05	1000000
	2	682,28		
	4	383,23		



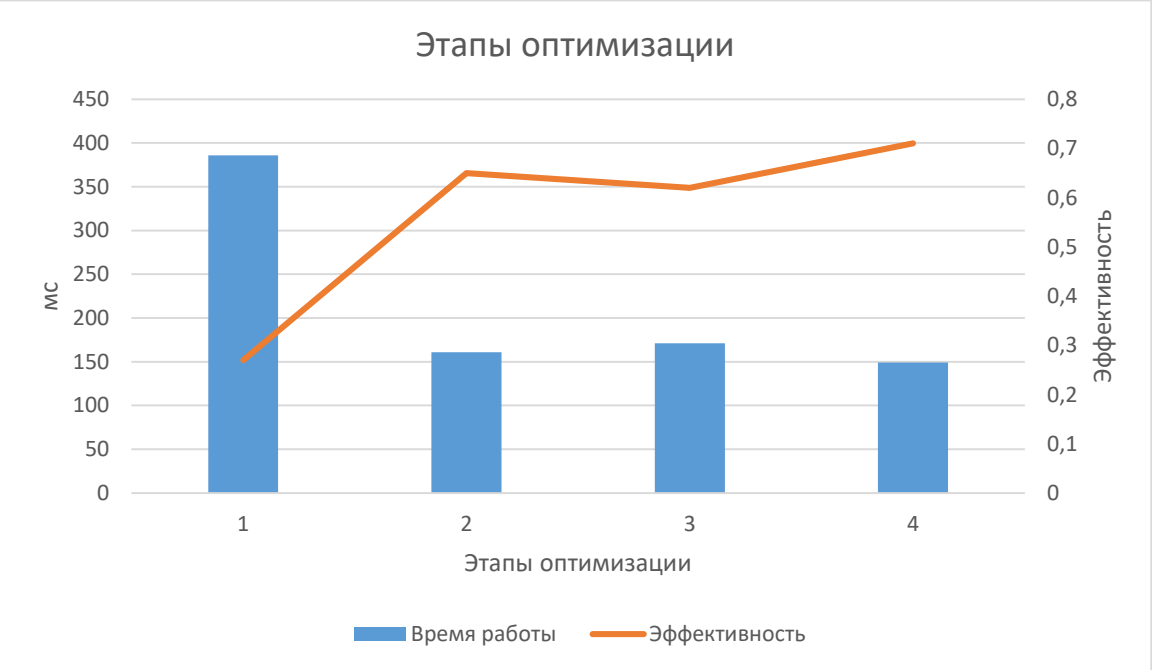


На диаграмме выше мы можем увидеть, что эффективность распараллеливания программы растет вместе с увеличением размера сетки. И на сетке размером 8192x8192 эффективность в четырёх поточном исполнении достигает 90%.

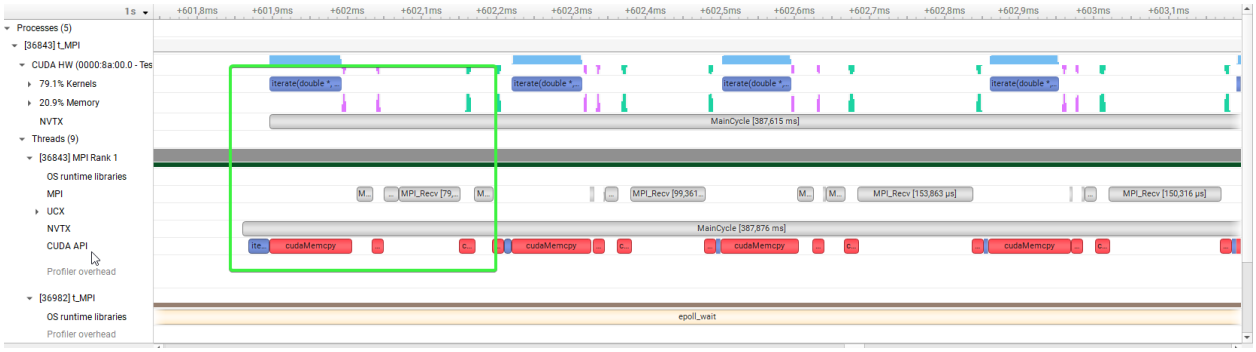
Этапы оптимизации кода на GPU (MPI/nccl)

Этапы оптимизации на сетке 4096x4096

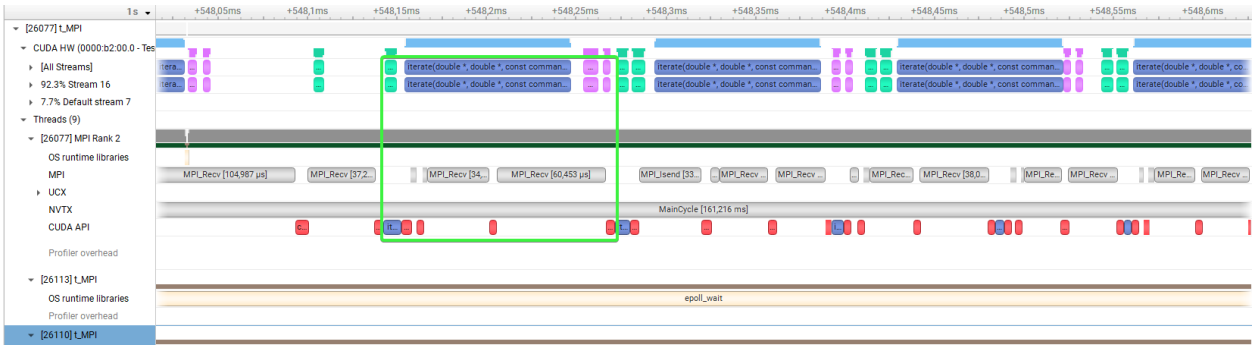
№	Время выполнения основного цикла	Ошибка	Кол-во итераций	Комментарий
1	386 мс	0.0089742	1200	*Версия на MPI. Без использования асинхронности. Время представлено для 4 карт. Эффективность 27%*
2	161 мс	0.0089742	1200	*Версия на MPI. С использованием асинхронности. Время представлено для 4 карт. Эффективность 65%*
3	171 мс	0.0089742	1200	*Версия на nccl. Без асинхронности. Время представлено для 4 карт. Эффективность 62%*
4	149 мс	0.0089742	1200	*Версия на nccl. С асинхронностью. Время представлено для 4 карт. Эффективность 71%*
Также были эксперименты с различной сеткой нитей и блоков для запуска функций на видеокартах. В случаях выше использовалась схема с Thread(1024, 1) [Первое по X, второе по Y]. Для блоков значения подбирались так чтобы максимизировать Y и покрыть весь участок. Были замеры с использованием схемы Thread(32, 32) и равномерными блоками, но в данном случае средняя эффективность одного потока была на 20%-30% ниже.				



Первый этап (выделен один вычислительный вызов. Шаг по сетке +передача граничных значений) Все происходит в последовательном режиме в рамках одного устройства.

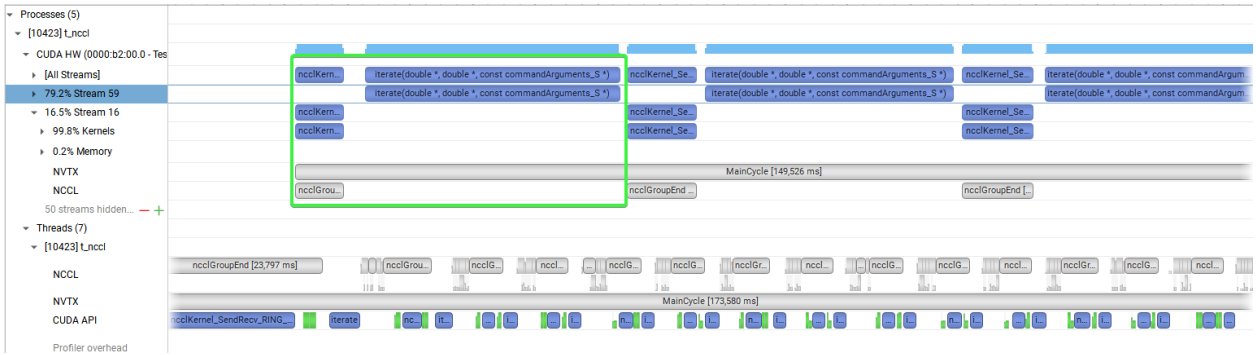


Второй этап (выделен один вычислительный вызов. Шаг по сетке +передача граничных значений) Новый шаг по сетке, а также передача граничных значений происходит в ассинхроном режиме в рамках одного устройства. Проблем с изменением данных по ходу вычисления не происходит т.к. данные передаются в рамках хоста, а перенос их на устройство происходит сразу после окончания шага.



Третий этап

Четвертый этап (выделен один вычислительный вызов. Шаг по сетке +передача граничных значений) Новый шаг по сетке, а также передача граничных значений происходит в ассинхроном режиме в рамках одного устройства. Проблем с изменением данных не происходит из-за того что вызовы nccl происходят в рамках того же потока, что и вызовы основного вычисления.



Вывод:

Используя MPI для запуска сразу нескольких процессов мы можем добиться возможности использовать одновременно несколько GPU в параллельном режиме, а за счет обмена данными между процессами через NCCL можем приблизить на больших стеках эффективность к 1. Для маленьких сеток нет смысла распараллеливать программу на несколько устройств т.к. накладные расходы будут слишком велики и больше времени уйдет на общее и синхронизацию.

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