**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Технологийпараллельного программирования

**Отчет по лабораторной работе №6**

**Тема:** «Параллельное вычисление произведения двух матриц на графическом процессоре.»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-357 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Акмурзин М.Э. |  |  |  |
| Принял | Добровольцев А.С. |  |  |  |

**Уфа 2023**

**Цель:** на примере задачи параллельного вычисления произведения двух матриц научиться проводить декомпозицию на двумерные блоки, а также оптимизировать программы для GPU с использованием разделяемой памяти.

**Теоретическая часть**

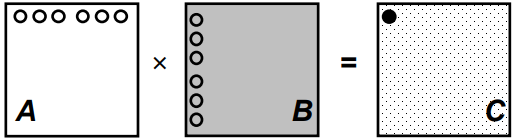
Глобальная память – память, в которую могут записывать и читать все нити GPU. Скорость доступа при это не самая большая.

Разделяемая память – быстрая память, расположенная на мультипроцессоре. Объявляется разделяемая память с помощью спецификатора \_\_shared\_\_.

Вычисление матриц

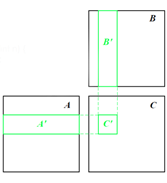
Требуется перемножить квадратные матрицы и вычислить квадрат евклидовой нормы результирующей матрицы:

Матричное умножение выражается формулой



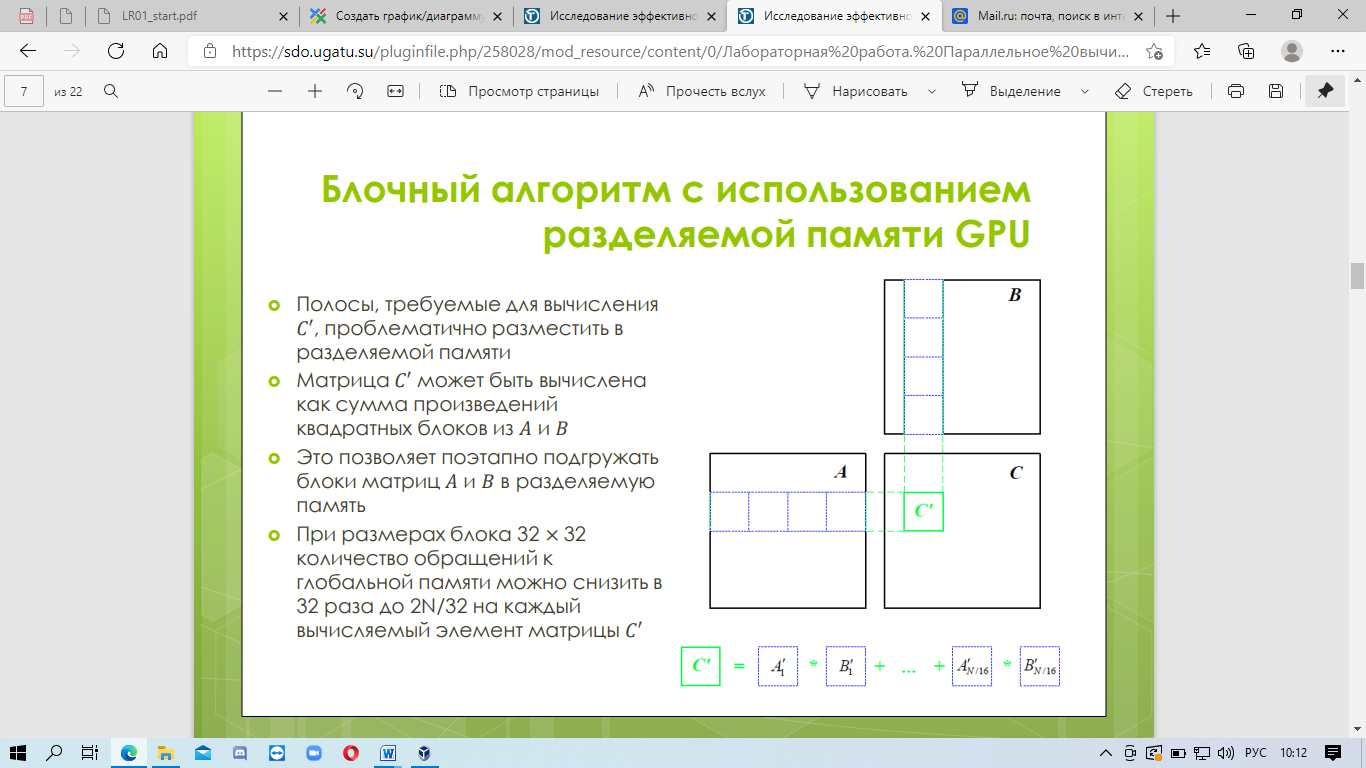
*Рисунок 1.Умножение матриц.*

На GPU это можно реализовать, убрав два внешних цикла из классического алгоритма умножения матриц. Простейшим способом будет использование глобальной памяти.



*Рисунок 2. Простейший алгоритм с использованием глобальной памяти GPU.*

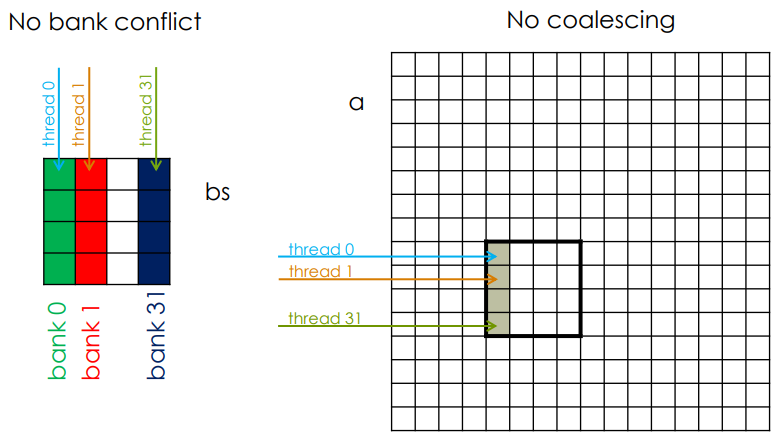
Можно оптимизировать данный алгоритм, снизив количество обращений в долгую глобальную память.



*Рисунок 3. Блочный алгоритм с использованием разделяемой памяти на GPU.*

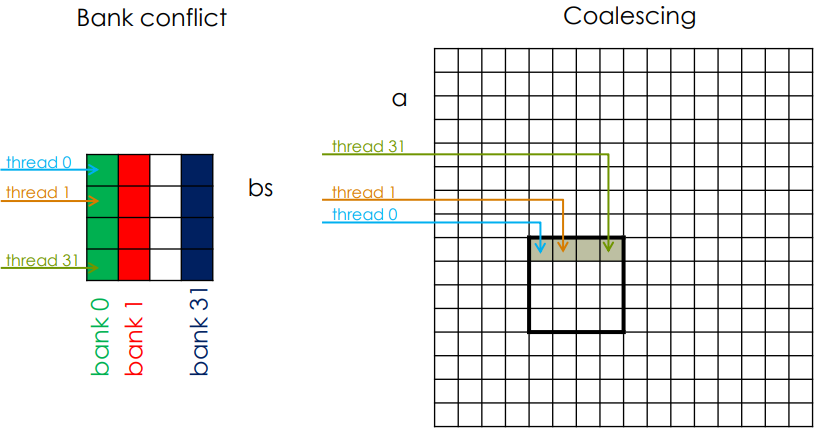
Матриц 𝐶′ может быть вычислена как сумма произведений квадратных блоков из A и B. Это дает нам возможность поэтапно подгружать блоки матриц в разделяемую память. При размерах блока 32 × 32 количество обращений к глобальной памяти можно снизить в 32 раза до 2N/32 на каждый вычисляемый элемент матрицы 𝐶′.

Существуют дальнейшие оптимизации при вычислении произведения матриц, где место B мы используем AT. Т.к. размер матрицы в разделяемой памяти 32, каждый столбец хранится в одном банке. В классической реализации не происходит банк конфликта, но нет coalescing’а.



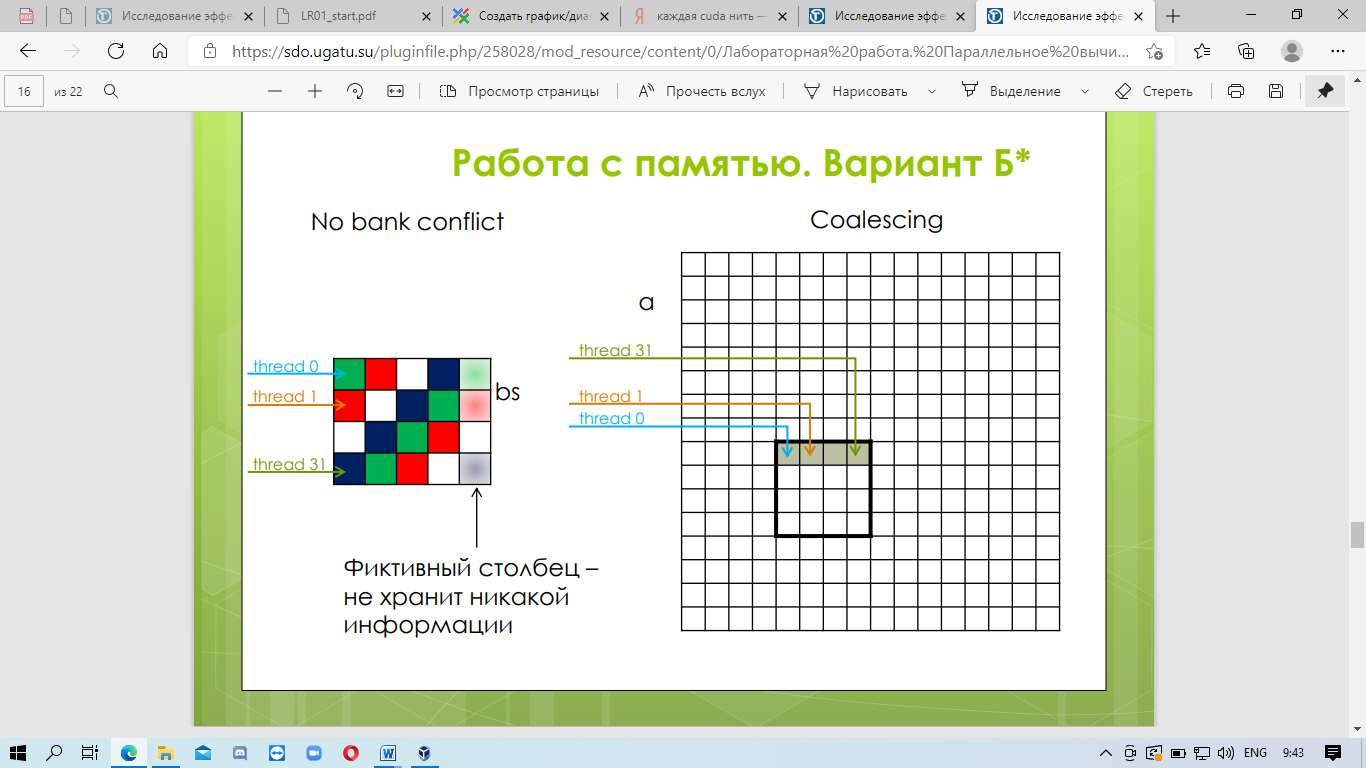
*Рисунок 4. Вариант А.*

Второй вариант – считывать строки из глобальной памяти и записывать в столбцы, тогда происходит coalescing, но при этом существует банк конфликт 32 порядка.



*Рисунок 5. Вариант Б.*

Попробуем оптимизировать данный алгоритм, избавившись от банк-конфликтов. Достаточно добавить фиктивный столбец, который не хранит данных, тогда элементы столбца будут лежать в разных банках, следовательно, не будет конфликтов.



*Рисунок 6. Вариант Б\*.*

**Практическая часть**

Параметры графического процессора, на котором проводились вычисления:

1. Архитектура AMPERE
2. CUDA Capability Major/Minor version number 8.6
3. Total amount of global memory 8GB
4. Пропускная способность памяти 224 GB/s
5. Число СUDA-ядер 2560
6. Пиковая производительность 9.098 TFLOPS

**Задачи**

* Разработка программы, реализующей перемножения двух квадратных матриц на CPU (последовательно и параллельно);
* Реализация простейшего алгоритма вычисления матричного произведения на GPU с использованием глобальной памяти;
* Реализация блочного алгоритма вычисления матричного произведения с использованием разделяемой памяти;
* Реализация различных вариантов вычисления ;
* Тестирование производительности и профилирование разработанных программ.

**https://github.com/Mis-prog/\_tpp\_5sem/tree/lab6**

**Индивидуальное задание №1.**

Задание:

1. Размерности матриц предполагаются кратными 32 (если нет, программа завершается с сообщением об ошибке).
2. Перемножаются матрицы 𝐴 и 𝐵 с элементами вещественного типа одинарной точности, сгенерированные ранее на CPU.
3. Необходимо предусмотреть расчет нормы матрицы 𝐶.
4. Реализовать последовательную и многопоточную (OpenMP) версии для CPU, а также с использованием глобальной и разделяемой памяти для GPU.

*При компиляции для CPU выставить ключ -O2;*

*При компиляции для GPU выставить ключ -arch=sm\_50 или выше.*

*Таблица 1. Время выполнения программ.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Версия, платформа | Время(ms) | N=256 | N = 1024 | N = 4096 |
| Последовательная, CPU | Время расчета на CPU | 25 | 7114.83 | 556762 |
| Параллельная, CPU | Время расчета на CPU | 8.4235 | 4393.98 | 316807 |
| С использованием глобальной памяти, GPU | Время обмена данными с GPU | 0.32 | 2.6305 | 29.668 |
| Время расчета на GPU | 0.18 | 6.579 | 280.751 |
| Полное время работы на GPU | 0.5 | 9,209 | 310.419 |
| С использованием разделяемой памяти, GPU | Время обмена данными с GPU | 0.285 | 2.204 | 33.892 |
| Время расчета на GPU | 0.1467 | 3.516 | 242.453 |
| Полное время работы на GPU | 0,4317 | 5,72 | 276,345 |

*Таблица 2. Ускорение на GPU относительно параллельной версии на CPU.*

|  |  |  |  |
| --- | --- | --- | --- |
| Версия | N = 256 | N = 1024 | N = 4096 |
| С использованием глобальной памяти | 16,847 | 477,03 | 1 020,57 |
| С использованием разделяемой памяти | 19,512 | 768 | 1 146,41 |

*Таблица 3. Производительность разработанных версий.*

|  |  |  |  |
| --- | --- | --- | --- |
| Версия, платформа | N = 256 | N = 1024 | N = 4096 |
| Параллельная, CPU | 3,21 | 3,49 | 0,41 |
| С использованием глобальной памяти, GPU | 90,69 | 112,97 | 145,85 |
| С использованием разделяемой памяти, GPU | 145,89 | 219,58 | 437,9 |

*Уточнение. Используемое в расчёте время было переведено в секунды.*

По полученным данным, видно, что любая реализация на GPU выигрывает по времени версии на CPU. Более того, расчеты с использованием разделяемой памяти GPU являются наиболее производительными.

**Индивидуальное задание №2.**

Задание:

1. Реализовать три версии ядра перемножения матрицы A на AT с использованием разделяемой памяти.

1. Без конфликтов банков и без коалесинга (Вариант А).

2. С конфликтами банков и коалесингом (Вариант Б).

3. Без конфликтов банков и с коалесингом (Вариант Б\*).

2. Провести исследование полученных реализаций.

При компиляции выставить ключ -arch=sm\_50 (не меньше, чем 50).

*Таблица 4. Время выполнения функции ядра.*

|  |  |  |  |
| --- | --- | --- | --- |
| Версия | N=256 | N=1024 | N=4096 |
| Вариант А | 0,243 | 4.46 | 316.29 |
| Вариант Б | 0.42 | 5.664 | 298.65 |
| Вариант Б\* | 0.17 | 3.34 | 218.15 |

Реальной производительностью называется количество команд или операций, которое выполняет вычислительная система в единицу времени для конкретного алгоритма.

*Таблица 5. Производительность функции-ядра (GFlops)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Версия | Размерность | N=256 | N=1024 | N=4096 |
| Вариант А | Производительность | 138.084 | 481.49 | 434.53 |
| % от пиковой | 1.53 | 5.349 | 4.82 |
| Вариант Б | Производительность | 79.891 | 379.14 | 460.20 |
| % от пиковой | 0.88 | 4.212 | 5.11 |
| Вариант Б\* | Производительность | 197.379 | 642.959 | 630.02 |
| % от пиковой | 2.19 | 7.1439 | 7 |

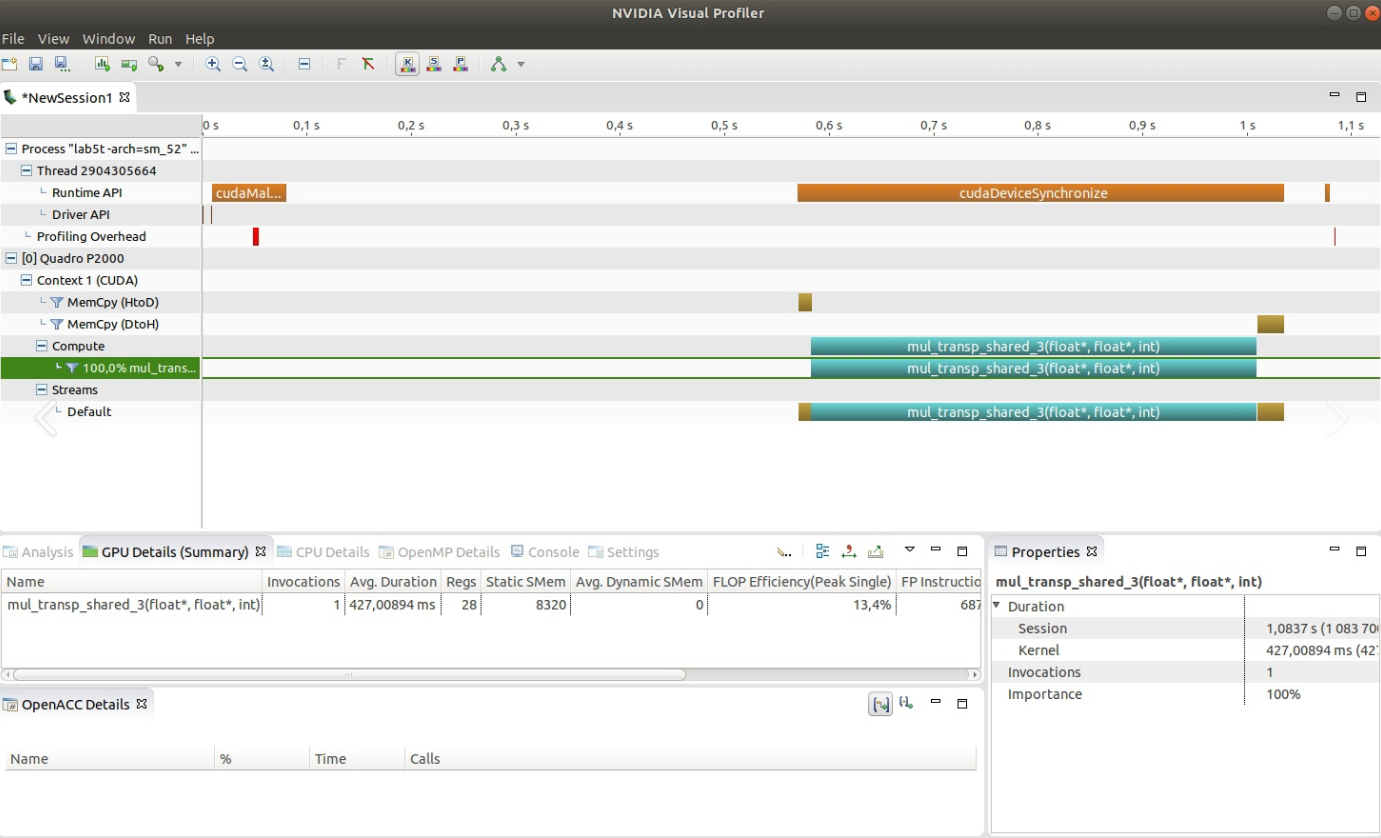
*Рисунок 7. Процент от пиковой производительности ядра.*

https://sun9-39.userapi.com/impg/JZaL-PFA6wFT_IfHkY5JhEvOZ83n60Dp9fFkFA/eqPAgRnxsmg.jpg?size=485x58&quality=96&sign=28a49de3da5f2f8b6f96025d0652cefe&type=album

https://sun9-8.userapi.com/impg/WSPXkr0bGDH4ZUAAIBExEF6Hb5CKoOf2Cthg9g/R2VaJvayXqU.jpg?size=526x53&quality=96&sign=57bb1a0939cebb23a3882526e6b0540c&type=album



*Рисунок 8. Скриншоты банк конфликта для варианта А, Б, Б\* соответственно.*



*Рисунок 9. Скриншот NVIDIA Profiler для Б\* и N=4096.*

**Вывод**

В данной лабораторной работе, на примере произведения двух матриц, была продемонстрирована оптимизация программы для GPU с использованием разделяемой памяти. Были рассмотрены различные способы повышения производительности программы.

**Приложение**

https://github.com/Mis-prog/\_tpp\_5sem/tree/lab6https://github.com/Mis-prog/\_tpp\_5sem/tree/lab6