Московский Государственный Университет им. М.В. Ломоносова

Факультет Вычислительной Математики и Кибернетики

Кафедра Суперкомпьютеров и Квантовой Информатики

Изображение выглядит как текст

Автоматически созданное описание

**Практикум на ЭВМ, 7 семестр**

**Отчёт № 2.**

**Optimised image convolution on NVIDIA GPUs using CUDA**

|  |  |
| --- | --- |
|  | Работу выполнил  **Малмыгин Г. А.** |
|  |  |

Москва 2021

**Задача**

1. получает входные параметры командной сроки (типы используемого фильтра и входных данных — про них далее);

2. загружает с диска необходимые изображения;

3. преобразует изображения в линейные массивы (развертка матрицы в линейный массив)

4. копирует эти массивы в память GPU;

5. запускает CUDA-ядра, которые применяют к изображениям необходимый фильтр;

6. выгружает результат в память CPU;

7. выводит 2 времени работы: только CUDA-ядер, а также CUDA-ядер + копирований

данных;

8. сохраняет полученные после фильтрации изображения на диск (также в виде

изображений, которые можно потом посмотреть).

Для выполнения второго задания необходимо реализовать следующие оптимизации разработанной на первом этапе программы:

Оптимизации для обработки больших и малых изображений:  
- Развертка массива, где хранится изображение из массива структур в структуру

массивов для улучшения шаблона доступа к глобальной памяти (Pixel \* -> 3 массива

unsigned char\* для хранения 3 компонент изображения);

* -  Последовательный доступ к памяти от нитей варпа к массиву с изображением;
* -  Использование разделяемой (shared) памяти для применения фильтра (по аналогии со

stencil);

* -  Использование 3x нитей для обработки r/g/b компонент;
* -  Различные походы к передаче фильтра в матрицу (full unroll, константная память);
* -  Развертка циклов, применяющих фильтров внутри каждой нити;
* -  Подбор оптимальных значений размера CUDA блока;
* -  Минимизация числа простаивающих нитей;

Дополнительные оптимизации для обработки набора из маленьких изображений:  
- Выделение памяти (cudaMalloc) под обрабатываемые изображения 1 раз (а не каждый

раз для каждого изображения заново);

* -  Обработка нескольких изображений за раз одним ядром или обработка нескольких

изображений в конкурентном режиме при помощи CUDA-потоков;

* -  Одновременные копирования DtoH, HtoD и запуск ядер;
* -  Параллельная работа с файлами обработка изображений на GPU для групп из N -

изображений: загружаем группу из N изображений с диска, пока их обрабатываем - грузим следующую. Сохранение на диск можно отключить (ifdef \_\_NEED\_TO\_SAVE\_\_).

**Структура работы программы**

В качестве библиотеки для загрузки изображений выбрана stb\_image.

Обработка одного изображения строится следующим образом:

1) В аргументах передается тип фильтра и тип изображения

2) С помощью библиотеки stb производится загрузка изображения в линейный массив типа unsigned char

3) Выделяется память для линейного массива, в котором будет храниться результирующий массив, выделяется память для фильтра, передача данных на устройство, начало замера времени с пересылками.

4)Запуск ядра и замер его времени. Используется линейный грид, каждый поток вычисляет координату пикселя в изображении, для которого будет производиться свертка, свертка пикселя с рассчитанными координатами.

5)Перенос данных результирующего изображения на хост, сохранение изображения, окончание замера времени с пересылками.

**Ядра выбранных фильтров**

Edge detection – выделяет границы объектов на изображении, sharpen – делает изображение более резким, gaussian blur – производит размытие изображения.

**Примеры работы фильтров**

|  |  |  |
| --- | --- | --- |
| Фильтр | Исходное изображение | Обработанное изображение |
| Edge detection 3x3  Image size – 300x300 | **Изображение выглядит как небо, внешний, земля, лошадиные  Автоматически созданное описание** | **Изображение выглядит как карта  Автоматически созданное описание** |
| Sharpen 3x3  Image size – 300x300 | **Изображение выглядит как небо, внешний, земля, лошадиные  Автоматически созданное описание** | **Изображение выглядит как небо, лошадиные, земля, внешний  Автоматически созданное описание** |
| Gaussian blur 5x5  Image size – 300x300 | **Изображение выглядит как небо, внешний, земля, лошадиные  Автоматически созданное описание** | **Изображение выглядит как небо, внешний, лошадиные, земля  Автоматически созданное описание** |
| Edge detection 3x3  Image size – 2000x2000 | **Изображение выглядит как внешний, небо, гора, природа  Автоматически созданное описание** | **Изображение выглядит как природа, пещера, темный  Автоматически созданное описание** |
| Sharpen 3x3  Image size – 2000x2000 | **Изображение выглядит как внешний, небо, гора, природа  Автоматически созданное описание** |  |
| Gaussian blur 5x5  Image size – 2000x2000 | **Изображение выглядит как внешний, небо, гора, природа  Автоматически созданное описание** | **Изображение выглядит как внешний, небо, гора, природа  Автоматически созданное описание** |

**Время работы программы**

|  |  |  |
| --- | --- | --- |
| Тип изображения и тип фильтра | Время выполнения только ядер, мс | Время выполнения ядер и копирований данных, мс |
| Edge detection 3x3  Image size – 300x300 | 19.9053 | 20.2041 |
| Sharpen 3x3  Image size – 300x300 | 19.3012 | 19.5824 |
| Gaussian blur 5x5  Image size – 300x300 | 20.9699 | 21.2468 |
| Edge detection 3x3  Image size – 2000x2000 | 1534.91 | 1539.77 |
| Sharpen 3x3  Image size – 2000x2000 | 1443.6 | 1448.56 |
| Gaussian blur 5x5  Image size – 2000x2000 | 1574.77 | 1579.74 |

Обработка одного изображения размером 2000 на 2000, все замеры производились на фильтре edge detection.

|  |  |  |  |
| --- | --- | --- | --- |
| Оптимизации | Время работы программы (только ядер), мс | Время работы программы (ядра и копирования), мс | Ускорение (времени работы ядра) |
| Начальная программа(0) | 1534.91 | 1539.77 | 1 |
| Развертка массива и двумерный грид, так как три массива нити варпа последовательно обращаются в память(1) | 0.72704 | 3.41424 | 2111.176826584507 |
| Использование разделяемой памяти(2) | 1.01546 | 3.82413 | 1511.5415673684834 |
| Использование трех нитей для rgb компонент(3) | 1.95386 | 4.9088 | 785.5782911774642 |
| Различные подходы к передаче фильтра в матрицу(4) | 1.44282 | 4.15446 | 1063.8263955309742 |
| Развертка циклов, применяющих фильтры внутри каждой нити(5) | 1.49142 | 4.85706 | 1029.1601292727737 |
| Подбор оптимального значения CUDA блока, уменьшение размера CUDA блока не дало результата, поэтому оптимальным остается блок размера 1024(6) | 1.49142 | 4.85706 | 1029.1601292727737 |
| Минимизация числа простаивающих нитей, выполняется за счет изменения размера грида(7) | ~ | ~ | ~ |

Комментарий к таблице: первое улучшение значительно ускорило работу программы, так как использовался в корне неверный подход при передаче массива и подход к его обработке. Использование разделяемой памяти замедлило работу ядра, что можно связать с накладными расходами необходимыми для инициализации разделяемой памяти. Замедление при создании трех компонент также можно связать с появлением дополнительных расходов на инициализацию разделяемой памяти. Использование глобальной памяти для передачи фильтра в матрицу позволило ускорить работу ядра, но развертка цикла не внесла значительного влияния на ускорение программы, что можно связать тем, что компилятор сам развертывает циклы небольшого размера, поэтому ускорения добиться не удалось, используемый с самого начала размер блока равнялся 1024, что является максимальным размером блока на используемом устройстве. Уменьшение размера блока никак не ускоряло работу ядра, поэтому было принято решение о сохранении размера блока размером 1024. Минимизация количества нитей уже было вложено в начальную программу за счет расчета необходимого количества нитей для обработки изображения.

Обработка нескольких изображений (оптимизации для одного изображения из прошлой таблицы уже добавлены), фильтр edge detection, десять изображений

|  |  |  |  |
| --- | --- | --- | --- |
| Оптимизации | Время работы программы (только ядер), мс | Время работы программы (ядра и копирования), мс | Ускорение (времени работы ядер и копирования) |
| Начальная версия (0) | 199.439 | 202.081 | 1 |
| Начальная версия (добавлены оптимизации для большого изображения)(1) | 0.611616 | 3.31187 | 61.017189684377705 |
| Выделение памяти под изображения один раз(2) | 0.5656 | 2.1224 | 95.21343761779119 |
| Обработка нескольких изображений за раз одним ядром или обработка нескольких изображений в конкурентном режиме при помощи CUDA-потоков (в данном случае используются CUDA потоки)(3) | 0.552384 | 4.62874 | 43.657885299239105 |
| Одновременные копирования и запуск ядер (Async + streams)(4) | 0.563552 | 4.38125 | 46.12405135520685 |
| Параллельная работа с файлами обработка изображений на GPU для групп из N изображений (5) | ~ | ~ | ~ |

Комментарий к таблице: ускорение при выделении памяти один раз связано с меньшим обращением к функциям для выделения памяти. Неясно почему использование нескольких CUDA-потоков и совместного использования ядер не дало ускорения. Одновременные копирования позволили немного снизить время копирований, которые теперь выполняются одновременно. Параллельная работа с файлами в цикле по обработке маленьких изображений в целом реализована, проблема заключается в том, что внутри цикла по обработке маленьких изображений находятся функции, синхронизирующие устройство для правильного замера времени, поэтому если расставить функции замера времени вне цикла информация окажется неактуальной и несравнимой с другими результатами, полученными в таблице.

Графики:

Для большого изображения:



Для малых изображений:



Профилировка:

Для большого изображения:

Начальная версия:

Изображение выглядит как текст

Автоматически созданное описание

Конечная версия:

Изображение выглядит как текст

Автоматически созданное описание

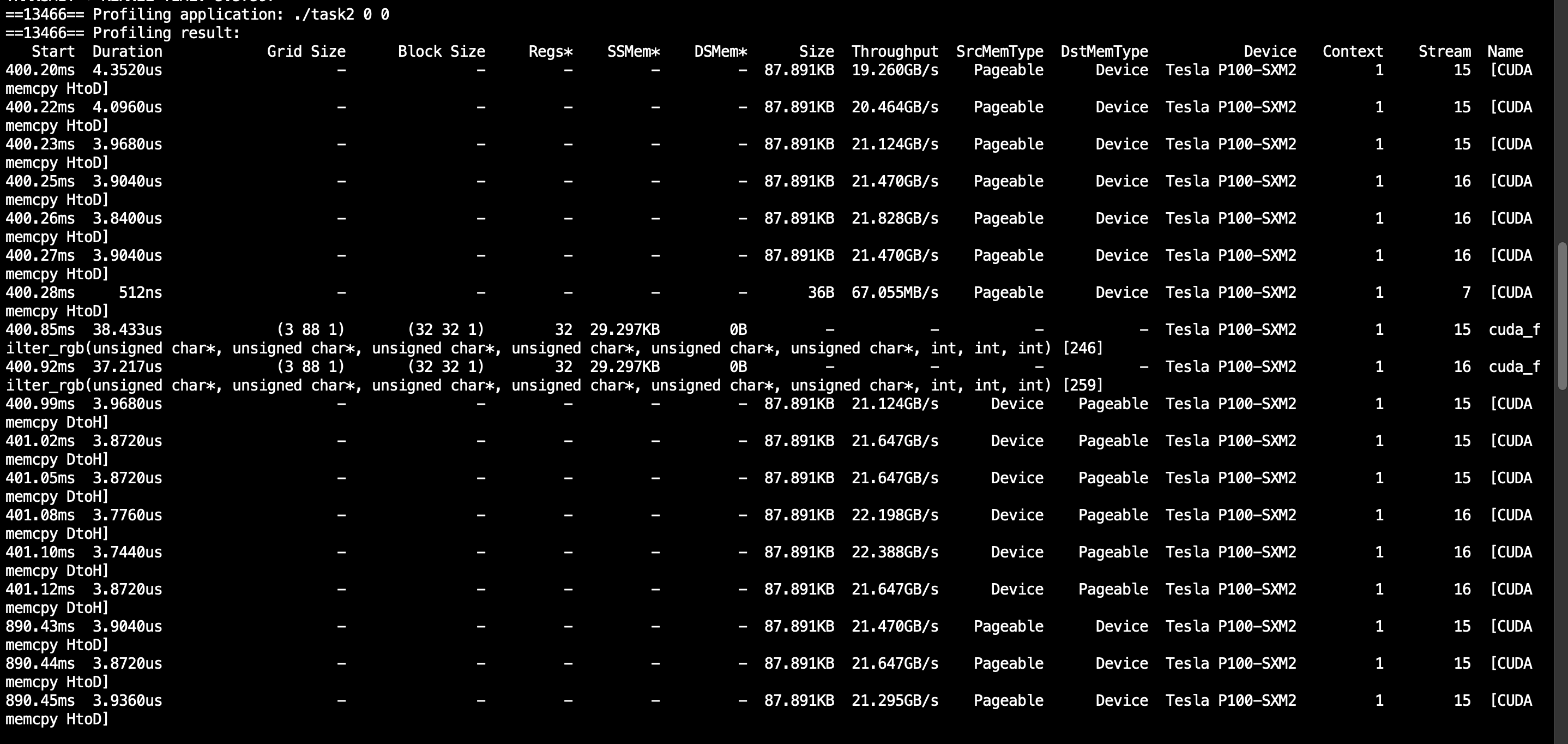
Для маленьких изображений:

Начальная версия:

Изображение выглядит как текст

Автоматически созданное описание

Конечная версия (часть вывода):



Выводы: первые оптимазиции дали значительный прирост в скорости работы программы, что можно связать с неправильным начальным подходом построения CUDA-ядра. Большинство оптимизаций следующих за первой приносили либо небольшой выигрыш в скорости работы программы, либо замедляли время ее работы в связи с накладными расходами.