



Практикум на ЭВМ, 7 семестр

Отчёт № 2.

Optimised image convolution on NVIDIA GPUs using CUDA

Работу выполнил
Малмыгин Г. А.

Задача

1. получает входные параметры командной строки (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 3х 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.





$$\textit{Edge detection kernel} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

$$\textit{Sharpen kernel} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

$$\textit{Gaussian Blur kernel} = \frac{1}{256} * \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

Примеры работы фильтров

Фильтр	Исходное изображение	Обработанное изображение
Edge detection 3x3 Image size – 300x300		
Sharpen 3x3 Image size – 300x300		
Gaussian blur 5x5 Image size – 300x300		
Edge detection 3x3 Image size – 2000x2000		

Sharpen 3x3 Image size – 2000x2000		
Gaussian blur 5x5 Image size – 2000x2000		

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

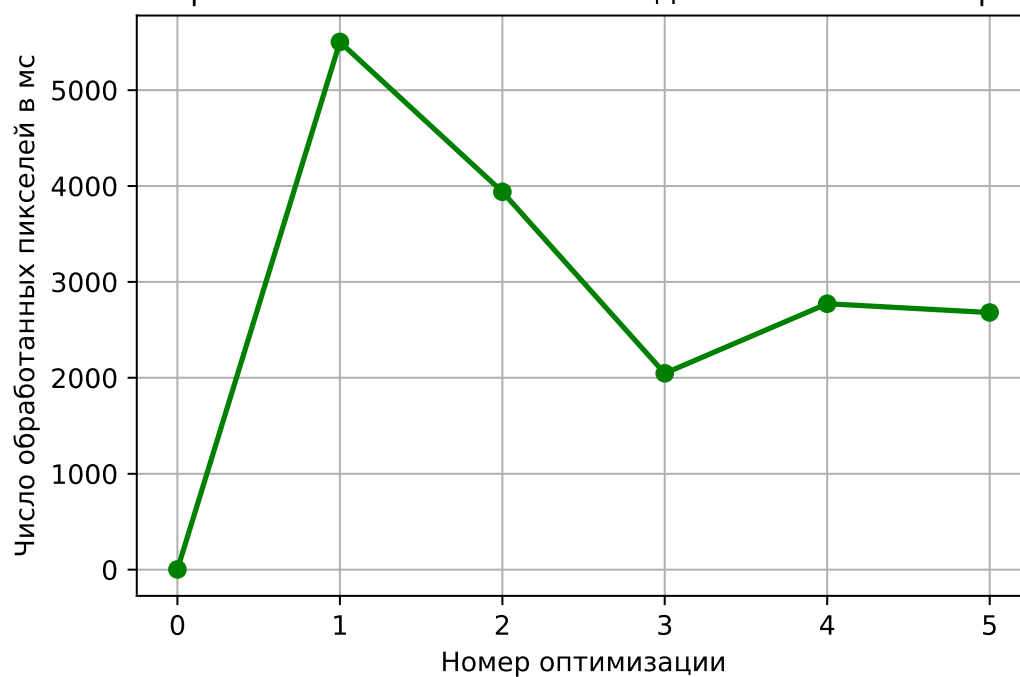
Оптимизации	Время работы программы (только ядер), мс	Время работы программы (ядра и копирования), мс	Ускорение (времени работы ядер и копирования)
Начальная версия (0)	199.439	202.081	1
Начальная версия (добавлены оптимизации для большого изображения)(1)	0.611616	3.31187	61.017189684377705
Выделение 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-поток (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-поток 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.

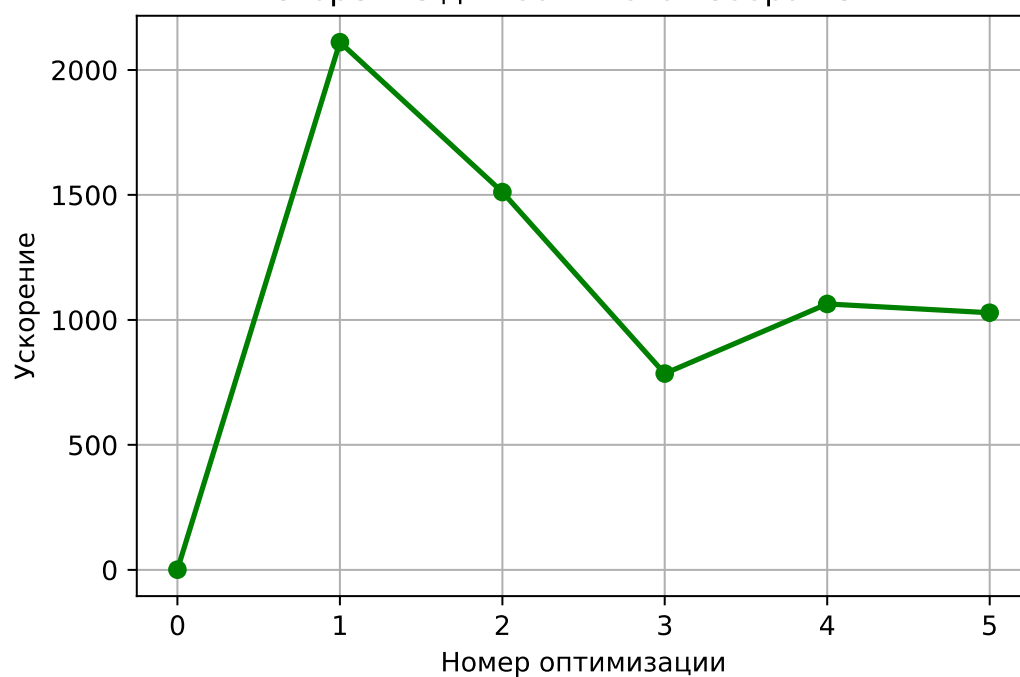
Графики:

Для большого изображения:

Число обработанных пикселей в мс для большого изображения



Ускорение для большого изображения



Для малых изображений:



Профилировка:

Для большого изображения:

Начальная версия:

```
==73515== NVPROF is profiling process 73515, command: ./task1 0 1
1024
3907
KERNEL AND COPY TIME: 1558.43
KERNEL TIME: 1552.76
Image written!
==73515== Profiling application: ./task1 0 1
==73515== Profiling result:
  Start Duration      Grid Size      Block Size      Regs*      SSMem*      DSMem*      Size      Throughput      SrcMemType      DstMemType      Device      Context      Stream      Name
357.41ms 1.6320us              -              -              -              -              -      368      21.037MB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
memcpy HtoD]
357.67ms 2.5776ms              -              -              -              -      -      11.444MB      4.3358GB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
memcpy HtoD]
361.14ms 1.55245s      (3907 1 1)      (1024 1 1)      32      0B      0B      -      -      -      -      Tesla P100-SXM2      1      7      cuda_f
ilter(unsigned char*, unsigned char*, float*, int, int, int, int, int) [222]
1.91370s 2.4553ms              -              -              -              -      -      11.444MB      4.5517GB/s      Device      Pageable      Tesla P100-SXM2      1      7      [CUDA
memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
```

Конечная версия:

```
[edu-cmc-sql20-10@polus-ib task2]$ nvprof --print-gpu-trace ./task2 0 1
Loaded image with
width:2000
height:2000
channels:3

==12288== NVPROF is profiling process 12288, command: ./task2 0 1
blockDim_y : 10
gridDim_y : 3907
Image written!
==12288== Profiling application: ./task2 0 1
==12288== Profiling result:
  Start Duration      Grid Size      Block Size      Regs*      SSMem*      DSMem*      Size      Throughput      SrcMemType      DstMemType      Device      Context      Stream      Name
360.95ms 217.00us              -              -              -              -      -      3.8147MB      17.168GB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
memcpy HtoD]
361.22ms 173.54us              -              -              -              -      -      3.8147MB      21.467GB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
memcpy HtoD]
361.45ms 210.66us              -              -              -              -      -      3.8147MB      17.684GB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
memcpy HtoD]
361.67ms      544ns              -              -              -              -      -      368      63.111MB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
memcpy HtoD]
362.36ms 1.5035ms      (3 3907 1)      (32 32 1)      32      29.297KB      0B      -      -      -      -      Tesla P100-SXM2      1      7      cuda_f
ilter_rgb(unsigned char*, unsigned char*, unsigned char*, unsigned char*, unsigned char*, unsigned char*, int, int, int) [231]
363.90ms 640.68us              -              -              -              -      -      3.8147MB      5.8146GB/s      Device      Pageable      Tesla P100-SXM2      1      7      [CUDA
memcpy DtoH]
364.73ms 338.60us              -              -              -              -      -      3.8147MB      11.002GB/s      Device      Pageable      Tesla P100-SXM2      1      7      [CUDA
memcpy DtoH]
365.28ms 343.21us              -              -              -              -      -      3.8147MB      10.854GB/s      Device      Pageable      Tesla P100-SXM2      1      7      [CUDA
memcpy DtoH]
```

Для маленьких изображений:

Начальная версия:

```
==74284== NVPROF is profiling process 74284, command: ./task1 0 0
1024
88
KERNEL AND COPY TIME: 20.3892
KERNEL TIME: 20.048
Image written!
==74284== Profiling application: ./task1 0 0
==74284== Profiling result:
  Start Duration      Grid Size      Block Size      Regs*      SSMem*      DSMem*      Size      Throughput      SrcMemType      DstMemType      Device      Context      Stream      Name
  memcpy HtoD]      310.75ms  1.5360us      -      -      -      -      -      351.56KB  22.352MB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
  memcpy HtoD]      310.88ms  13.665us      -      -      -      -      -      351.56KB  24.535GB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
  311.82ms  19.789ms      (88 1 1)      (1024 1 1)      32      0B      0B      -      -      -      -      Tesla P100-SXM2      1      7      cuda_f
  iter(unsigned char*, unsigned char*, float*, int, int, int, int, int) [222]
  331.65ms  12.737us      -      -      -      -      -      351.56KB  26.323GB/s      Device      Pageable      Tesla P100-SXM2      1      7      [CUDA
  memcopy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
```

Конечная версия (часть вывода):

```
==13466== Profiling application: ./task2 0 0
==13466== Profiling result:
  Start Duration      Grid Size      Block Size      Regs*      SSMem*      DSMem*      Size      Throughput      SrcMemType      DstMemType      Device      Context      Stream      Name
  memcpy HtoD]      400.28ms  4.3520us      -      -      -      -      -      87.891KB  19.260GB/s      Pageable      Device      Tesla P100-SXM2      1      15      [CUDA
  memcpy HtoD]      400.22ms  4.0960us      -      -      -      -      -      87.891KB  20.464GB/s      Pageable      Device      Tesla P100-SXM2      1      15      [CUDA
  memcpy HtoD]      400.23ms  3.9680us      -      -      -      -      -      87.891KB  21.124GB/s      Pageable      Device      Tesla P100-SXM2      1      15      [CUDA
  memcpy HtoD]      400.25ms  3.9040us      -      -      -      -      -      87.891KB  21.470GB/s      Pageable      Device      Tesla P100-SXM2      1      16      [CUDA
  memcpy HtoD]      400.26ms  3.8400us      -      -      -      -      -      87.891KB  21.828GB/s      Pageable      Device      Tesla P100-SXM2      1      16      [CUDA
  memcpy HtoD]      400.27ms  3.9040us      -      -      -      -      -      87.891KB  21.470GB/s      Pageable      Device      Tesla P100-SXM2      1      16      [CUDA
  memcpy HtoD]      400.28ms  512ns      -      -      -      -      -      368      67.055MB/s      Pageable      Device      Tesla P100-SXM2      1      7      [CUDA
  memcpy HtoD]      400.85ms  38.433us      (3 88 1)      (32 32 1)      32      29.297KB      0B      -      -      -      -      Tesla P100-SXM2      1      15      cuda_f
  iter_rgb(unsigned char*, unsigned char*, unsigned char*, unsigned char*, unsigned char*, unsigned char*, int, int, int) [246]
  400.92ms  37.217us      (3 88 1)      (32 32 1)      32      29.297KB      0B      -      -      -      -      Tesla P100-SXM2      1      16      cuda_f
  iter_rgb(unsigned char*, unsigned char*, unsigned char*, unsigned char*, unsigned char*, unsigned char*, int, int, int) [259]
  400.99ms  3.9680us      -      -      -      -      -      87.891KB  21.124GB/s      Device      Pageable      Tesla P100-SXM2      1      15      [CUDA
  memcopy DtoH]      401.02ms  3.8720us      -      -      -      -      -      87.891KB  21.647GB/s      Device      Pageable      Tesla P100-SXM2      1      15      [CUDA
  memcopy DtoH]      401.05ms  3.8720us      -      -      -      -      -      87.891KB  21.647GB/s      Device      Pageable      Tesla P100-SXM2      1      15      [CUDA
  memcopy DtoH]      401.08ms  3.7760us      -      -      -      -      -      87.891KB  22.198GB/s      Device      Pageable      Tesla P100-SXM2      1      16      [CUDA
  memcopy DtoH]      401.10ms  3.7440us      -      -      -      -      -      87.891KB  22.388GB/s      Device      Pageable      Tesla P100-SXM2      1      16      [CUDA
  memcopy DtoH]      401.12ms  3.8720us      -      -      -      -      -      87.891KB  21.647GB/s      Device      Pageable      Tesla P100-SXM2      1      16      [CUDA
  memcopy DtoH]      890.43ms  3.9040us      -      -      -      -      -      87.891KB  21.470GB/s      Pageable      Device      Tesla P100-SXM2      1      15      [CUDA
  memcopy DtoH]      890.44ms  3.8720us      -      -      -      -      -      87.891KB  21.647GB/s      Pageable      Device      Tesla P100-SXM2      1      15      [CUDA
  memcopy DtoH]      890.45ms  3.9360us      -      -      -      -      -      87.891KB  21.295GB/s      Pageable      Device      Tesla P100-SXM2      1      15      [CUDA
  memcopy DtoH]
```

Выводы: первые оптимизации 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.