МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

Институт информационных технологий, математики и механики
Кафедра математического обеспечения и суперкомпьютерных технологий

ЛАБОРАТОРНАЯ РАБОТА
«Линейная фильтрация изображений (горизонтальное разбиение). Фильтр Гаусса (ядро 3x3)»

Выполнил: студент группы 381506-3
Галочкин Борис Владимирович
_____ Подпись

Руководитель:
Сысоев Александр Владимирович
_____ Подпись

Нижний Новгород

2018

Оглавление

Введение

Под фильтрацией изображений понимают операцию, имеющую своим результатом изображение того же размера, полученное из исходного по некоторым правилам. Обычно интенсивность (цвет) каждого пикселя результирующего изображения обусловлена интенсивностями (цветами) пикселей, расположенных в некоторой его окрестности в исходном изображении.

Правила, задающие фильтрацию, могут быть самыми разнообразными. Операция, заключающаяся в последовательном применении двух или более фильтраций, тоже является фильтрацией. Таким образом, можно говорить о составных фильтрах, соответствующих комбинациям простых. Фильтрация изображений является одной из самых фундаментальных операций компьютерного зрения, распознавания образов и обработки изображений. Фактически, с той или иной фильтрации исходных изображений начинается работа подавляющего большинства методов.

Размытие по Гауссу — это характерный фильтр размытия изображения, который использует нормальное распределение (также называемое Гауссовым распределением, отсюда название) для вычисления преобразования, применяемого к каждому пикселю изображения.

В случае двух измерений эта формула задает поверхность, имеющей вид концентрических окружностей с распределением Гаусса от центральной точки. Пиксели, где распределение отлично от нуля используются для построения матрицы свертки, которая применяется к исходному изображению. Значение каждого пикселя становится средне взвешенным для окрестности. Исходное значение пикселя принимает наибольший вес (имеет наивысшее Гауссово значение), и соседние пиксели принимают меньшие веса, в зависимости от расстояния до них.

1. Постановка задачи

В рамках проделанной работы были поставлены следующие задачи:

- Написать программную реализацию фильтра Гаусса
- Проверить корректность работы программы, сравнив результаты с изображениями, полученными применением того же фильтра из библиотеки OpenCV.
- Добавить тесты, для возможности проверки корректности работы после внесения каких-либо изменений.
- Реализовать скрипт (checker), запускающий все тесты и сохраняющий полученные результаты в файл.
- Реализовать распараллеливание алгоритма, используя OpenMP, проверить корректность работы с помощью checker’a, получить ускорения при выполнении на нескольких потоках.
- Реализовать распараллеливание алгоритма, используя библиотеку TBB, проверить корректность работать, получить ускорения при выполнении на нескольких потоках.
- Проанализировать полученные результаты и сделать выводы.

2. Реализация генератора

На вход реализованной программе необходимо передавать имя бинарного файла, из которого будет считано изображение, для хранения был выбран следующий формат, очень похожий на BMP, первые байты ширина и высота, далее пиксели изображения построчно.

Так как работа происходит с изображениями различных форматов, для стресс-тестов нужно было создать удобный конвертер, и возможность генерировать набор тестов из набора изображений. Таким образом получился следующий процесс:

- Специальный скрипт скачивал случайные изображения с сайта imgur и сохранял в указанную папку
- Генератор, для каждого изображения из данной папки:
    - Изменял размер изображения до достаточно большого, чтобы можно было увидеть ускорения при работе с параллельными реализациями, но случайного, чтобы тесты отличались друг от друга.
    - Применял фильтр Гаусса из библиотеки OpenCV с радиусом 1 (ядро 3x3) к увеличенному изображению.
    - Сохранял оба результата в папку с тестовыми данными в требуемом для программы бинарном формате, как N и N.ans, где N номер изображения (теста)

Стоит заметить, что использование библиотеки OpenCV позволяет поддерживать большинство известных форматов изображений и сохранять их в нужном бинарном виде.

3. Реализация cheker’a

Основную работы выполняет генератор, задачами checker’a же являются:

- Запуск программы с реализацией фильтра Гаусса и нужными аргументами (число потоков и путь к файлу с исходными данными).
- Чтение выходного файла программы, после завершения её работы, в котором помимо результирующего изображения будет записано время работы.
- Попиксельное сравнение изображения-эталона, полученного из генератора, и выходного.
- Запись в лог количества не совпавших пикселей, их относительного количества в процентах, и времени работы программы.
- Если процент не совпавших меньше минимально заданного, тест считается пройденным и будет выведен положительный результат

На выходе будет получен следующий лог:

```
INFO:root:test0: AC. Results are equal. Time: 0.063s. Bad 0.0% 0/87235
INFO:root:test1: AC. Results are equal. Time: 0.018s. Bad 0.0% 0/23829
INFO:root:test2: AC. Results are equal. Time: 0.087s. Bad 0.0% 0/120780
INFO:root:test3: AC. Results are equal. Time: 0.041s. Bad 0.0% 0/56340
INFO:root:test4: AC. Results are equal. Time: 0.063s. Bad 0.0% 0/86260
INFO:root:test5: AC. Results are equal. Time: 0.072s. Bad 0.0% 0/99360
INFO:root:test6: AC. Results are equal. Time: 0.050s. Bad 0.0% 0/66924
INFO:root:test7: AC. Results are equal. Time: 0.051s. Bad 0.0% 0/69486
INFO:root:test8: AC. Results are equal. Time: 0.026s. Bad 0.0% 0/34992
INFO:root:test9: AC. Results are equal. Time: 0.051s. Bad 0.0% 0/70478
INFO:root:test10: AC. Results are equal. Time: 0.068s. Bad 0.0% 0/92367
INFO:root:test11: AC. Results are equal. Time: 0.059s. Bad 0.0% 0/81107
INFO:root:test12: AC. Results are equal. Time: 0.037s. Bad 0.0% 0/50634
INFO:root:test13: AC. Results are equal. Time: 0.046s. Bad 0.0% 0/60534
INFO:root:test14: AC. Results are equal. Time: 0.045s. Bad 0.0% 0/58912
INFO:root:test15: AC. Results are equal. Time: 0.036s. Bad 0.0% 0/50190
INFO:root:test16: AC. Results are equal. Time: 0.032s. Bad 0.0% 0/42656
INFO:root:test17: AC. Results are equal. Time: 0.054s. Bad 0.0% 0/73023
INFO:root:test18: AC. Results are equal. Time: 0.019s. Bad 0.0% 0/25670
```

Листинг 1. Вывод checker’a

4. Реализация алгоритма фильтрации

4.1. Алгоритм размытия с использованием фильтра Гаусса

Итак, пусть исходное изображение будет задано яркостью x(m,n)(то же самое применимо для каждого канала в отдельности, просто для простоты изложения). Размытие по Гауссу с радиусом r считается по формуле

$$y(m,n) = \frac{1}{2\pi r^2} \sum_{u,v} e^{\frac{-(u^2+v^2)}{2r^2}} x(m+u, n+v)$$

Где u и v это относительные координаты пикселей относительно центра ядра, которые ограниченны радиусом r, для ядра 3 на 3 r = 1.

Данная формула определяет значение веса для каждого элемента ядра, при расчете итогового значения элемента в центре. Таким образом при r = 1 нужно посчитать значения весов в 9 точках. В качестве оптимизации можно задать статическую матрицу с весами и использовать её (так как коэффициенты не зависят от значения текущего элемента, а только от относительного расположения), но главной задачей является демонстрация работы параллельных алгоритмов, так что на каждой итерации цикла вычисления производились заново. Итоговое значение яркости вычислялось как усредненная сумма значений всех элементов ядра, умноженных на веса и деленное на сумму весов.

Так же стоит принять во внимание граничные ряды пикселей изображения, так как ядро будет выходить за границы, есть несколько вариантов решения проблемы, мной был выбран вариант, когда вместо пикселей за “рамкой”, берутся их “зеркально отраженные” внутри.

4.2. Параллельная версия, с использованием OpenMP

Так как алгоритмы обработки изображений почти всегда линейны, каких-либо проблем с разделением задачи между потоками не возникло, в задании указано горизонтальное разбиение, поэтому перед внешним циклом (по строкам) была указана программная директива *#pragma omp parallel for*.

Поскольку объем данных обычно будет достаточно большим, необходимо динамическое их распределение между потоками, параметр *schedule(guided)*, означает, что объемы работ между потоками будут распределяться динамически, но размер порции уменьшается экспоненциально.

```
#pragma omp parallel for schedule(guided) shared(image)
for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
                process_current_area(…);
        }
}
```

Листинг 2. OpenMP

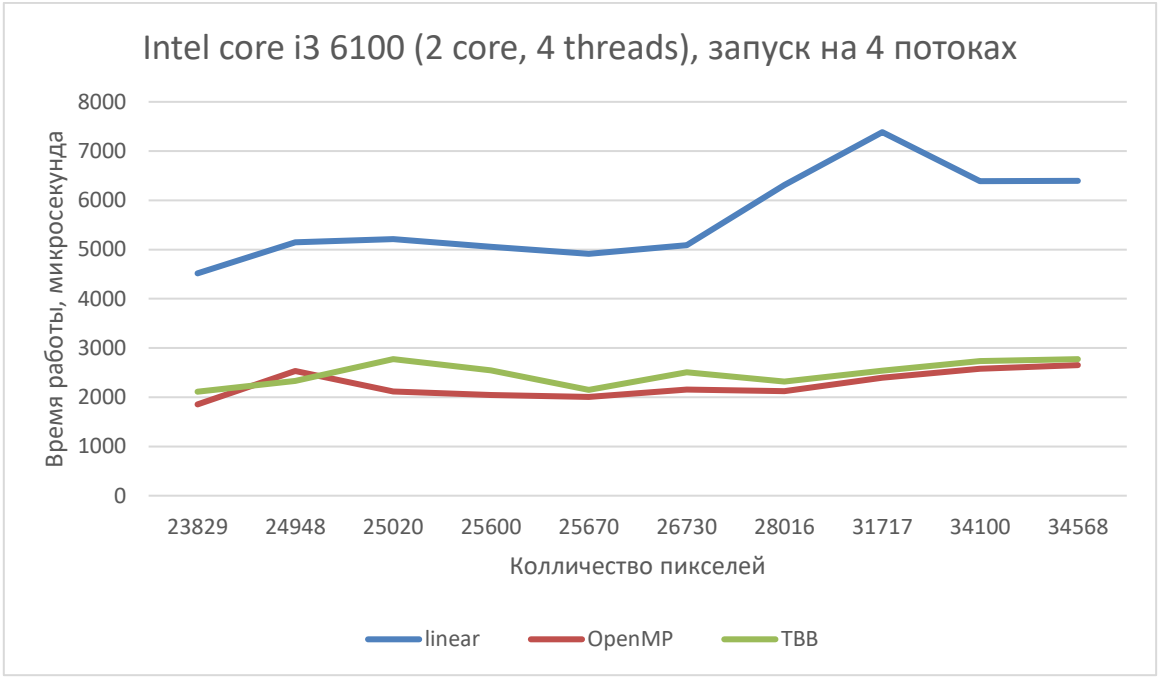
4.3. Параллельная версия, с использованием TBB

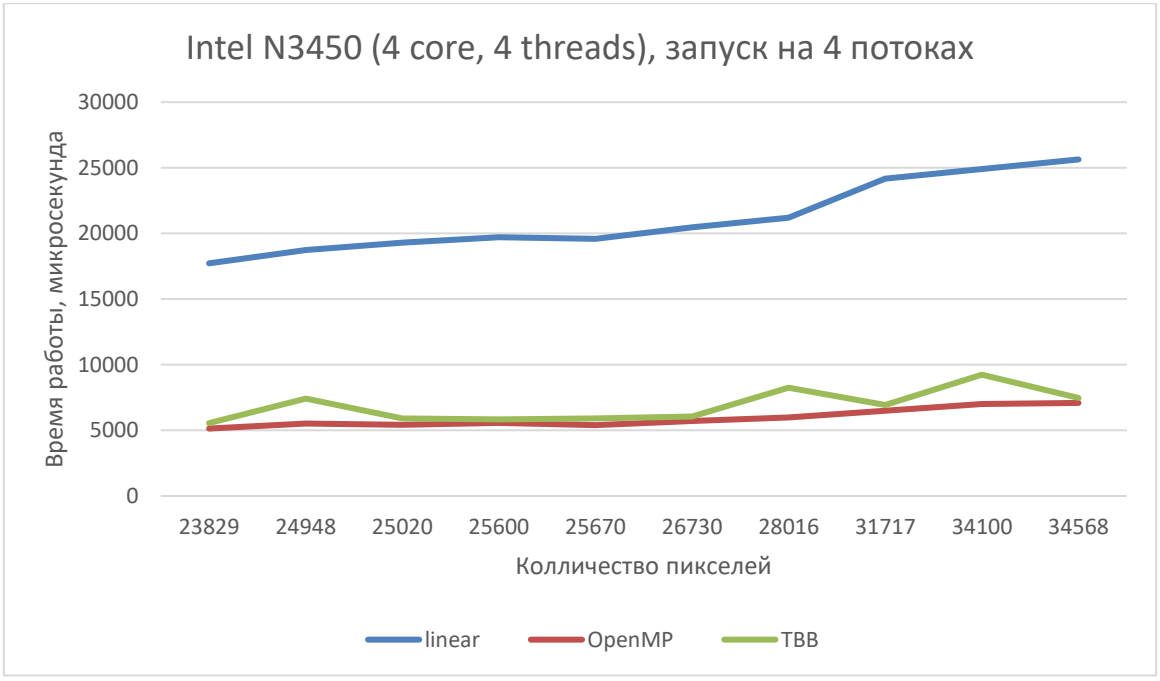
Параллельный код, с использование библиотеки *Intel TBB*, не сильно отличается от предыдущей версии, для разбиения изображения по высоте использовалось итерационное пространство и метод *parallel_for*. Так же можно динамически изменять порцию работы, которую будет выполнять каждый поток, но размеры изображений могут быть различны и подобрать подходящий, не так просто, поэтому было решено оставить этот выбор библиотеке, возможно по данной причине ускорения *TBB* слегка уступают *OpenMP* (хорошо видно на графике, результаты OpenMP более плавные, из-за чего среднее значение ускорения лучше).

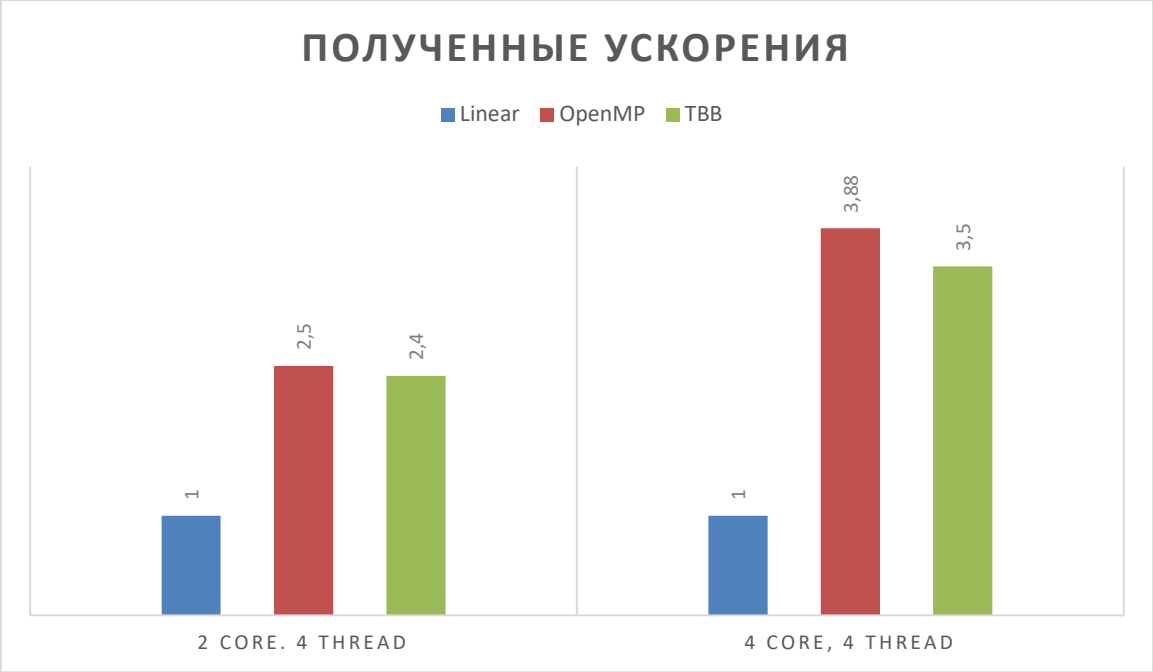
```
tbb::parallel_for(tbb::blocked_range<int>(0, h),
                        [&image, &result, &h, &w, &r]
                        (const tbb::blocked_range<int>& heigth) {
      for (int i = heigth.begin(); i != heigth.end(); ++i) {
          for (int j = 0; j < w; j++) {
                  process_current_area(…);
          }
}
```

Листинг 3. TBB

5. Результаты численных экспериментов






ПОЛУЧЕННЫЕ УСКОРЕНИЯ
Linear
OpenMP
TBB
1
2,5
2,4
1
3,88
3,5
2 CORE. 4 THREAD
4 CORE, 4 THREAD

Заключение

Исходя из результатов экспериментов, представленных выше, можно сделать следующий вывод. Чем больше размер изображения, тем наиболее эффективно распараллеливание, т.к. расходы на раздачу и сбор данных нивелируются временем вычислений, а использование высокоуровневых библиотек значительно упрощает задачу параллелизма и позволяет без особых временных затрат получать солидный прирост в производительности. Поскольку на данный момент очень тяжело найти одноядерную систему, и использовать эти возможности просто необходимо. Но не стоит забывать о том, что параллелизм, при не большом объеме работ, может привести к замедлению алгоритма.

Литература

1. Матричные фильтры обработки изображений https://habr.com/post/142818
2. Лекция по матричным фильтрам (Компьютерная графика) http://www.apmath.spbu.ru/ru/staff/pogozhev/lection07.pdf
3. Введение в OpenMP https://software.intel.com/ru-ru/blogs/2011/11/21/openmp-c
4. TBB *parallel for* https://software.intel.com/en-us/node/506153

дата обращения 21.05.2018