Министерство Образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Нижегородский Государственный Университет им. Н.И. Лобачевского» «Институт Информационных Технологий Математики и Механики»

Отчет по лабораторной работе
Фильтрация изображений с использованием ядра Гаусса

Выполнил:
студент группы 0836-2
Катаев Р.Д.

Проверил:
ассистент каф. МОСТ, ИИТММ
Козинов Е.А.

г. Нижний Новгород
2018 г.

Постановка задачи

При цифровой съемке практически неизбежно возникновение различных видов шумов, вызванных несовершенством сенсора/АЦП, наводками и другими факторами. Для удаления таких шумов применяются различные алгоритмы удаления шума. Один из основных подходов к цифровой обработке изображений – использование алгоритмов размытия, наиболее широко известным из которых является размытие по Гауссу.

рис. 1: Зашумленное изображение до и после применения размытия по Гауссу

С математической точки зрения, размытие по Гауссу – это операция свертки по изображению с использованием функции Гаусса:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad (1)$$

Значения, полученные с помощью данного распределения, используются для построения сверточной матрицы (ядра), которая затем применяется к каждому пикселю изображения. При этом значение обрабатываемого пикселя – взвешенная сумма всех пикселей окружения. Исходный пиксель входит в сумму с наибольшим весом, для остальных же пикселей вес уменьшается по мере удаления от центра ядра.

Цель данной работы: реализовать программу, фильтрующую изображения с помощью размытия по Гауссу, затем произвести параллелизацию данной программы с использованием технологий OpenMP и Intel TBB с проверкой корректности параллельных версий.

Шаги выполнения:

1. Реализовать последовательную версию программы
2. Реализовать программу-генератор тестов для проверки корректности параллелизации
3. Реализовать программу для проверки корректности, принимающую на вход файл тестового задания и выводящее метрику схожести между изображением, обработанным с помощью собственного алгоритма и результатом работы алгоритма, принимаемого за заведомо корректный (в качестве проверочной была выбрана функция библиотеки OpenCV)
4. Оценить ускорение от использования параллелизма

Метод решения

Для решения задачи будет применяться ядро размера 3x3, коэффициенты которого будут вычисляться по формуле (1). Пример такого ядра для $\sigma = 1.5$:

$$\begin{pmatrix} 0.411112 & 0.64118 & 0.411112 \\ 0.64118 & 1 & 0.64118 \\ 0.411112 & 0.64118 & 0.411112 \end{pmatrix} \quad (2)$$

Однако, если применить данное ядро к изображению, можно заметить, что изображение стало значительно ярче. Это происходит потому, что конечная взвешенная сумма больше изначального значения пикселя. Чтобы избежать данного эффекта, необходимо нормализовать ядро, разделив каждый его элемент на сумму всех элементов.

$$normalized_{i,j} = \frac{current_{i,j}}{\sum_{i,j} current_{i,j}}$$

Результат такого преобразования для ядра (2):

$$\begin{pmatrix} 0.0789209 & 0.123087 & 0.0789209 \\ 0.123087 & 0.191969 & 0.123087 \\ 0.0789209 & 0.123087 & 0.0789209 \end{pmatrix} \quad (3)$$

Ядро (3) готово к использованию.

Далее ядро поочерёдно накладывается на каждый пиксель изображения, и вычисляется взвешенная сумма по каждому каналу (R, G, B).

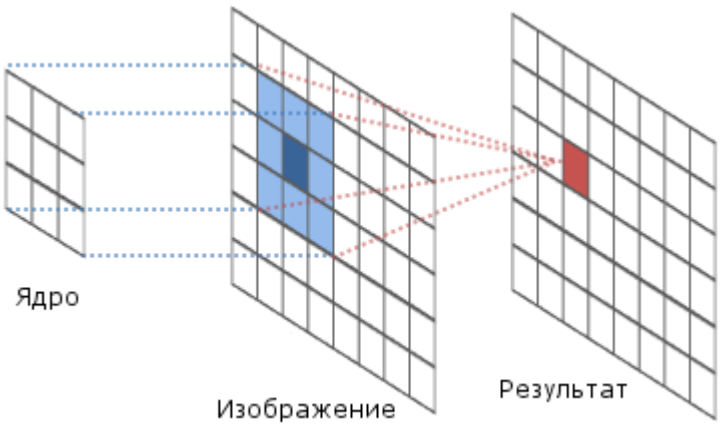


рис. 2: визуализация операции свертки над изображением

Последовательная реализация

1. Чтение файла задания с информацией об изображении и коэффициентом σ
2. Загрузка изображения в память
3. Создание ядра, расчет значений
4. Нормировка ядра
5. Для каждого пикселя произвести операцию свертки, вычислить новые значения для всех 3-х каналов
6. Присвоить пикселю выходного изображения полученное значение, убедившись, что оно находится в отрезке $[0, 255]$
7. Записать полученное изображение на диск

OpenMP – версия

Так как подавляющее большинство вычислительных операций происходит внутри основного цикла, программа хорошо поддаётся параллелизации с помощью директивы `#pragma omp parallel for`. В данной реализации изображение делится статически (по горизонтали). Замер времени выполнения производится с помощью функций `omp_get_wtime()`.

TBB – версия

Для данной версии был создан функтор `Gausser`, содержащий конструктор и метод `operator()`, в котором решение задачи выполняется в двумерном итерационном пространстве `blocked_range2d`. Параллелизация осуществляется с помощью функции `tbb::parallel_for`.

Основной цикл

```
for (int i = 0; i < height; i++)
{
	for (int j = 0; j < width; j++)
	{
		double blue = 0, green = 0, red = 0;
		for (int t = -radius; t <= radius; t++)
		{
			for (int k = -radius; k <= radius; k++)
			{
			int currX = control(i + t, 0, height - 1);
			int currY = control(j + k, 0, width - 1);
			Vec3b neighborCol = in.at<Vec3b>(currX, currY);
			blue += neighborCol.val[0] * gaussCore[t + radius][k + radius];
			green += neighborCol.val[1] * gaussCore[t + radius][k + radius];
			red += neighborCol.val[2] * gaussCore[t + radius][k + radius];
			}
		}
		Vec3b col(control(blue, 0, 255),
			control(green, 0, 255),
			control(red, 0, 255));
		out.at<Vec3b>(i, j) = col;
	}
}
```

Тесты и проверка корректности

В качестве тестовых данных были выбраны изображения формата .jpeg различных размеров, хранящихся в отельной директории.

Генератор создаёт файл, в котором первая строка указывает на название изображения, которое необходимо обработать, а вторая строка – случайное значение параметра σ в диапазоне $[1, 1.9]$.

Программа проверки считывает файлы заданий, вызывает функцию библиотеки OpenCV `GaussianBlur(…)` для исходного изображения со значением σ, указанном в файле теста, а затем подсчитывает метрику SSIM (structural similarity) между изображением, полученным собственным алгоритмом и результатом работы функции OpenCV. Формула метрики SSIM:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y+C)(2\sigma_{xy}+C)}{(\mu_x^2+\mu_y^2+C)(\sigma_x^2+\sigma_y^2+C)}$$ где:

x, y – изображения

μ_x, μ_y – средние значения пикселей для изображений x и y соответственно

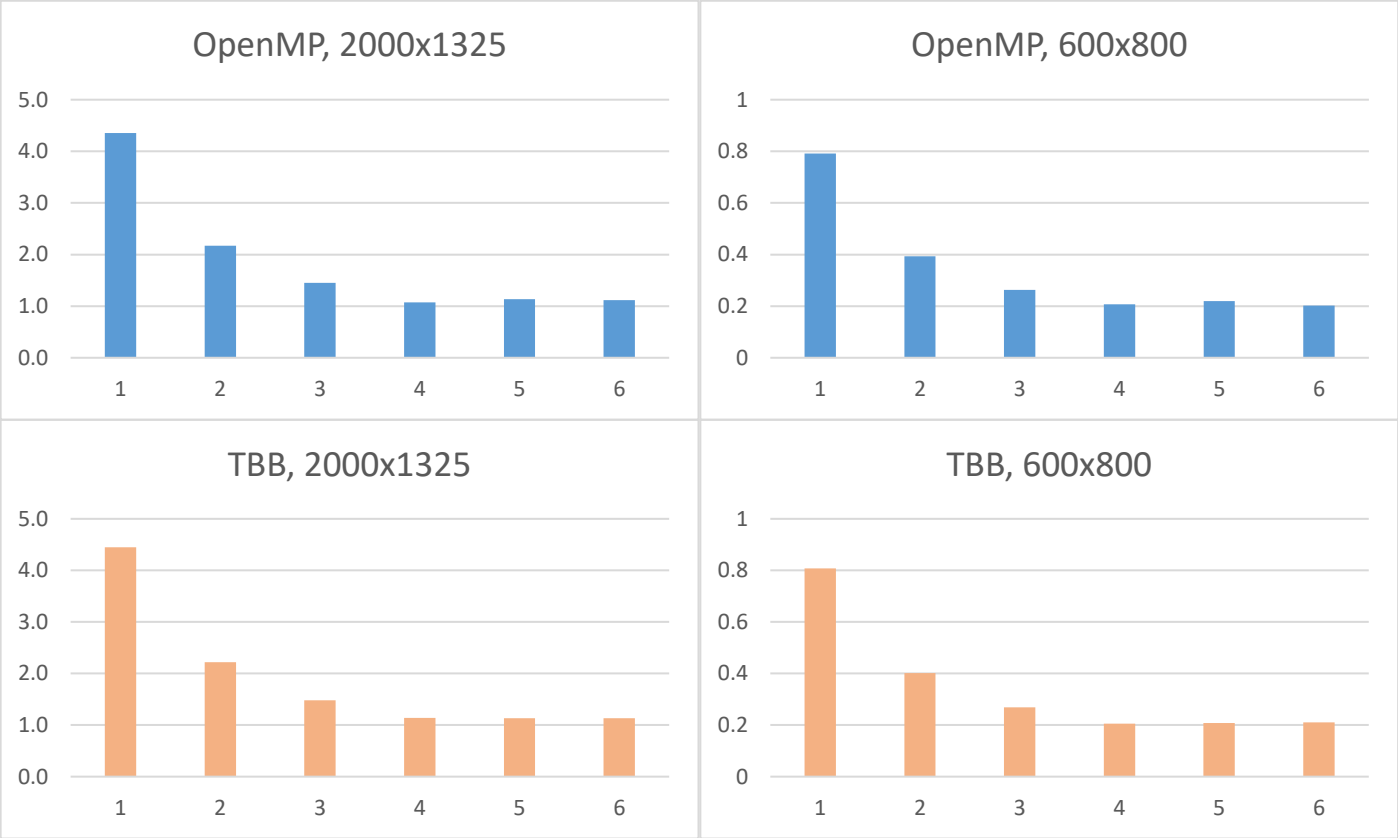
σ_x^2, σ_y^2 – дисперсия

σ_{xy} – ковариация

C – константа, равная 0.01

Производительность

Замеры производительности выполнялись на ПК на базе процессора Intel Core i5 с 4 ядрами.



Результаты замеров показывают стабильное, близкое к линейному ускорение при увеличении числа потоков, однако, ускорение отсутствует при задании количества потоков большего, чем число ядер процессора.

Заключение

В результате выполнения работы была реализована программа, производящая фильтрацию изображений с использованием размытия по Гауссу.

По результатам тестов можно сделать вывод, что данная задача хорошо поддаётся параллелизации, а применение технологий OpenMP и TBB позволит получить значительное ускорение при минимальных изменениях исходного кода.