



# **НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО**

## **ЦЕНТР КОМПЕТЕНЦИЙ ONEAPI В ННГУ**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ**



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО  
ЦЕНТР КОМПЕТЕНЦИЙ oneAPI в ННГУ  
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ  
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Пошаговая оптимизация программ  
в практических приложениях:  
реализация фильтра Гаусса**

# Содержание

---

- ❑ Цель работы
- ❑ Постановка задачи
- ❑ Оптимизация фильтра Гаусса
  - Базовая реализация фильтра
  - Оптимизация работы с памятью
  - Алгоритмическая оптимизация
  - Векторизация циклов
  - Распараллеливание программной реализации
  - Использование оптимизированной реализации фильтра на примере Intel Integrated Performance Primitives (Intel IPP)

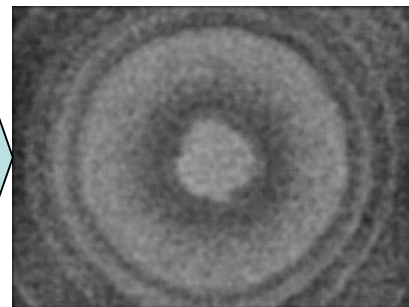
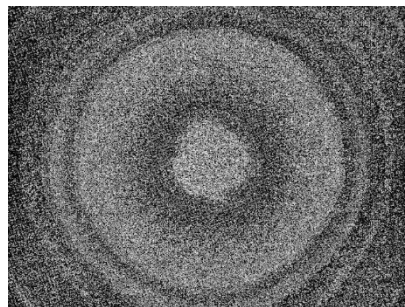
# Цели

---

- ❑ Рассмотреть на примере прикладной задачи (реализация фильтра Гаусса) методов оптимизации приложений
- ❑ Получить навыки использования библиотек OpenCV и Intel IPP

# Постановка задачи...

- Фильтр Гаусса используется:
  - для размытия изображения и уменьшения шумов
  - в качестве подготовительного этапа алгоритмов машинного обучения, связанных с обработкой изображений (сегментация, поиск объектов на изображении, классификация изображений, ...)
- Пример применения фильтра Гаусса:



# Постановка задачи...

## □ Методы хранения изображений

- Изображение может быть представлено набором пикселей
- Каждый пиксель имеет свой цвет. Цвет задается одним или несколькими значениями *интенсивности*
  - В черно-белых изображениях цвет представлен одним значением интенсивности. В цветных – несколькими
- Структура хранения изображения: последовательный набор интенсивностей для каждого пикселя



- Значения интенсивности могут быть заданы как целыми числами типа `unsigned char`, так и вещественными числами типа `float`
  - При наложении фильтра Гаусса будем использовать вещественный тип данных

# Постановка задачи...

□ Фильтр Гаусса задается ядром:

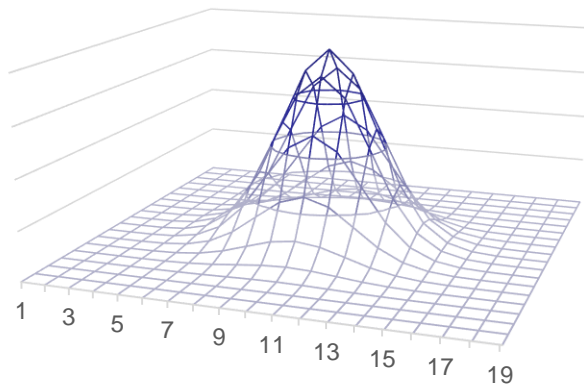
$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- $x, y$  – координаты точки
- $\sigma$  – среднеквадратическое отклонение (от значения зависит сила размытия)

□ Пример матрицы ядра ( $N = 5, \sigma = 2$ )

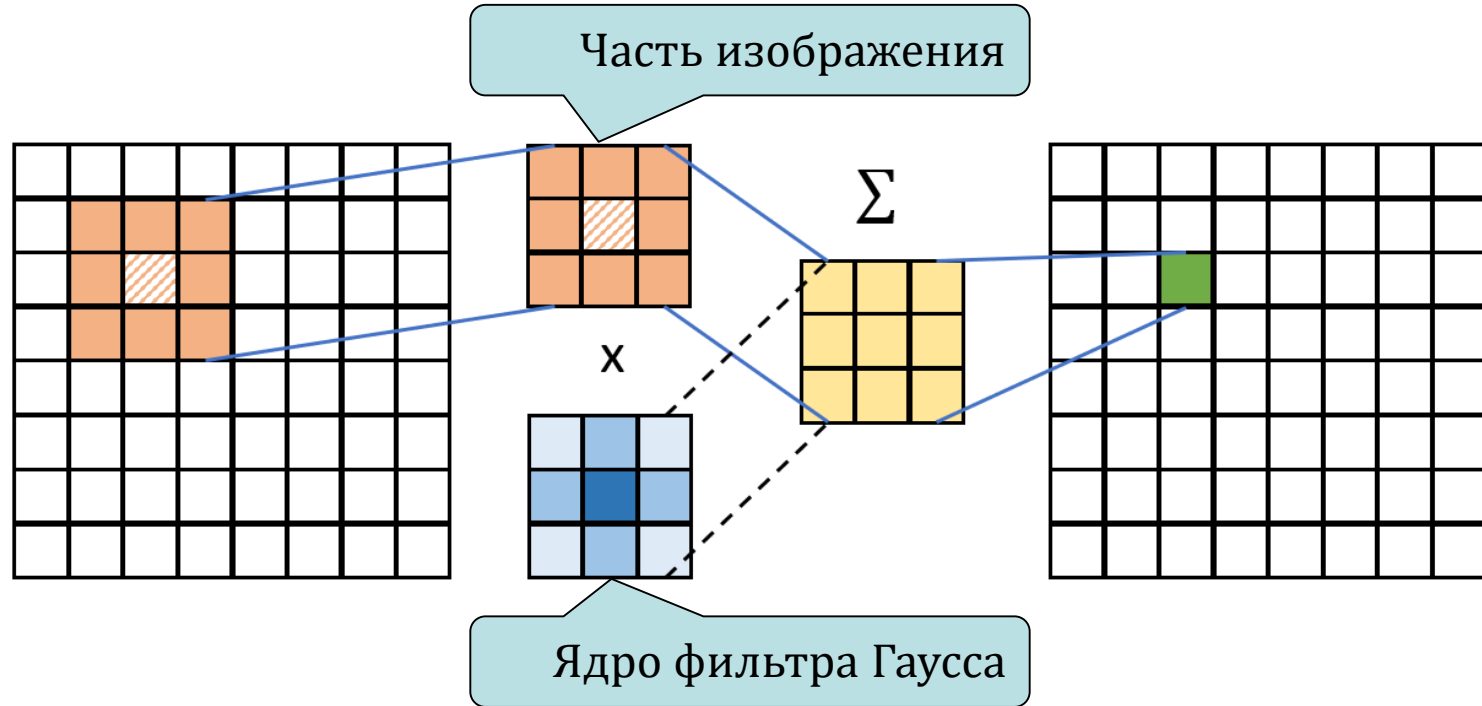
0,015	0,021	<b>0,024</b>	0,021	0,015
0,021	0,031	<b>0,035</b>	0,031	0,021
<b>0,024</b>	<b>0,035</b>	<b>0,040</b>	<b>0,035</b>	<b>0,024</b>
0,021	0,031	<b>0,035</b>	0,031	0,021
0,015	0,021	<b>0,024</b>	0,021	0,015

N=19 sigma=2



# Постановка задачи

- Общая схема применения фильтра для получения значений каждого пикселя:





# 1. БАЗОВАЯ РЕАЛИЗАЦИЯ

# Программная реализация...

## ❑ Проект опишем следующим CMake файлом:

```
cmake_minimum_required(VERSION 3.20)
PROJECT (hpc_lab)

set(CMAKE_CXX_STANDARD 11)

find_package(OpenCV REQUIRED)
find_package(OpenMP REQUIRED)

include_directories( ${OpenCV_INCLUDE_DIRS} )
add_compile_options(-qopt-report-phase=vec,loop -qopt-report=2 )

file(GLOB TARGET_SRC "*.c")
file(GLOB TARGET_HD "*.h")
add_executable( filter ${TARGET_SRC} ${TARGET_HD} )

target_link_libraries(filter ${OpenCV_LIBS} )
target_link_libraries(filter OpenMP::OpenMP_CXX)
```

# Программная реализация...

## □ Загрузка изображения и вызов реализации фильтра:

```
#include ...
using namespace cv;
int main()
{
    double time;
    int sizeFilter = 19;
    double sigma = 2;
    std::string image_path = samples::findFile("starry_night.jpg");
    Mat img = imread(image_path, IMREAD_COLOR);    //загрузка изображения
    Mat imgFloat, imgFloatDist;
    img.convertTo(imgFloat, CV_32FC3, 1 / 255.0); //конвертация типа данных
    imgFloat.copyTo(imgFloatDist);
    time = omp_get_wtime();
    filterBase(imgFloatDist, imgFloat, sigma, sizeFilter); //вызов фильтра
    time = omp_get_wtime() - time;
    std::cout << "Time base: " << time << std::endl;
    return 0;
}
```

# Программная реализация...

## ❑ Заполнение матрицы фильтра:

```
float* create2DFilter(int n, float sigma)
{
    float* filter = new float[n * n];
    float coeff = 1.0f / (2.0f * PI * sigma * sigma);
    int middle = n / 2;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            float x = j - middle;
            float y = i - middle;
            filter[i * n + j] = coeff * expf(-(x * x + y * y) / (2.0f * sigma * sigma));
        }
    }
    return filter;
}
```

# Программная реализация...

## □ Базовая реализация фильтра:

```
void filterBase(cv::Mat& dist, cv::Mat& src, float sigma, int sizeFilter)
{
    float* filter = create2DFilter(sizeFilter, sigma);

    int w = src.size().width;
    int h = src.size().height;
    float* srcData = (float *)src.data;
    float* distData = (float *)dist.data;

    int middle = sizeFilter / 2;
    const int cntChannel = 3;
    ...
}
```

# Программная реализация

## □ Базовая реализация фильтра:

```
for (int i = 0; i < h - sizeFilter; i++) {  
    for (int j = 0; j < w - sizeFilter; j++) {  
        for (int c = 0; c < cntChannel; c++) {  
            float sum = 0.f;  
            for (int i_f = 0; i_f < sizeFilter; i_f++) {  
                for (int j_f = 0; j_f < sizeFilter; j_f++) {  
                    int pos_i = (i + i_f) * (w * cntChannel);  
                    int pos_j = (j + j_f) * (cntChannel) + c;  
                    sum += srcData[pos_i + pos_j] * filter[i_f * sizeFilter + j_f];  
                }  
            }  
            int i_d = i + middle, j_d = j + middle;  
            distData[(i_d * w + j_d) * cntChannel + c] = sum;  
        }  
    }  
}  
delete[] filter;  
}
```

# Вычислительная инфраструктура

Процессоры	Intel(R) Xeon(R) Silver 4310T CPU @ 2.30GHz
Число процессоров	1
Общее число ядер	10
Память	512 Gb
Операционная система	Linux CentOS 7
Компилятор, профилировщик, отладчик	Intel® oneAPI

# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Время работы алгоритма: 3.857 с.

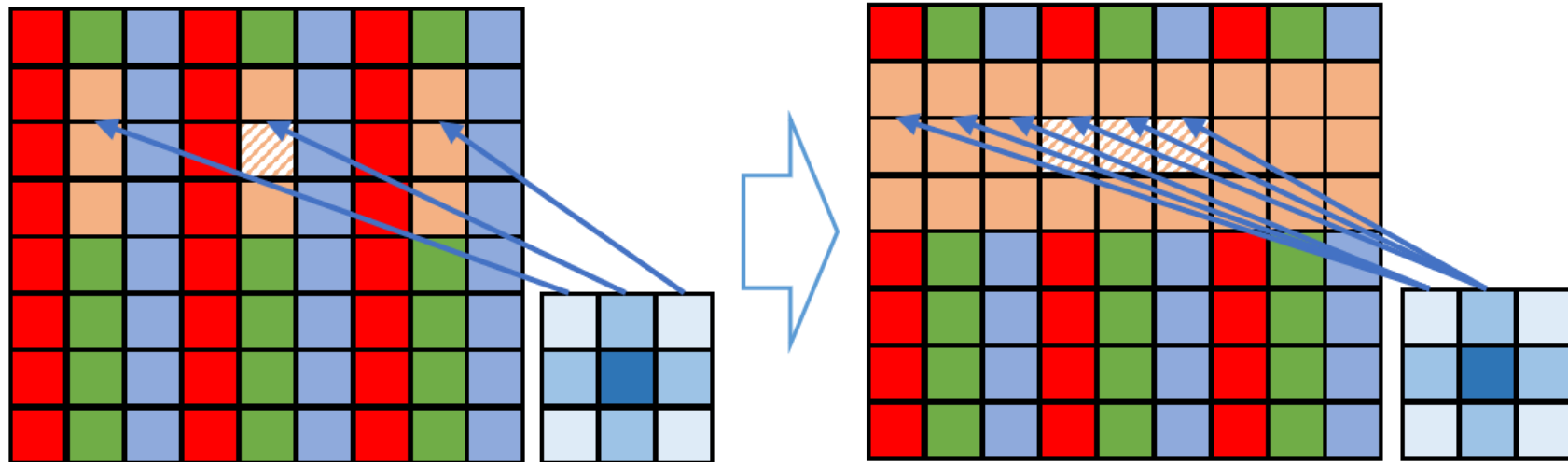




## 2. ОПТИМИЗАЦИЯ РАБОТЫ С ПАМЯТЬЮ

# Оптимизация работы с памятью

- В базовой версии не оптимальный обход по памяти



- Решение – изменение порядка циклов

# Программная реализация...

## □ Инициализация результирующей матрицы:

```
void filterChannel(cv::Mat &dist, cv::Mat& src, float sigma, int sizeFilter)
{
    float* filter = create2DFilter(sizeFilter, sigma);
    int w = src.size().width;
    int h = src.size().height;
    float* srcData = (float *)src.data;
    float* distData = (float*)dist.data;
    int middle = sizeFilter / 2;
    const int cntChannel = 3;
    for (int i = 0; i < h - sizeFilter; i++) {
        for (int j = 0; j < w - sizeFilter; j++) {
            int i_d = i + middle;
            int j_d = j + middle;
            for (int c = 0; c < cntChannel; c++) {
                distData[(i_d * w + j_d) * cntChannel + c] = 0.0f;
            }
        }
    }
}
```

# Программная реализация

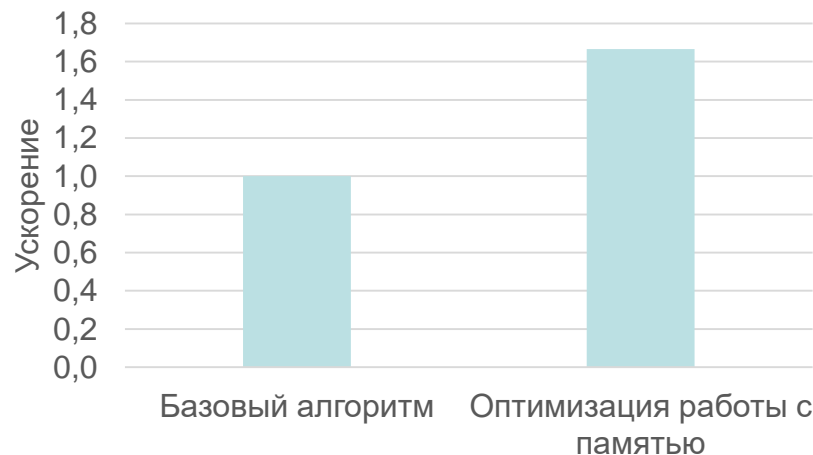
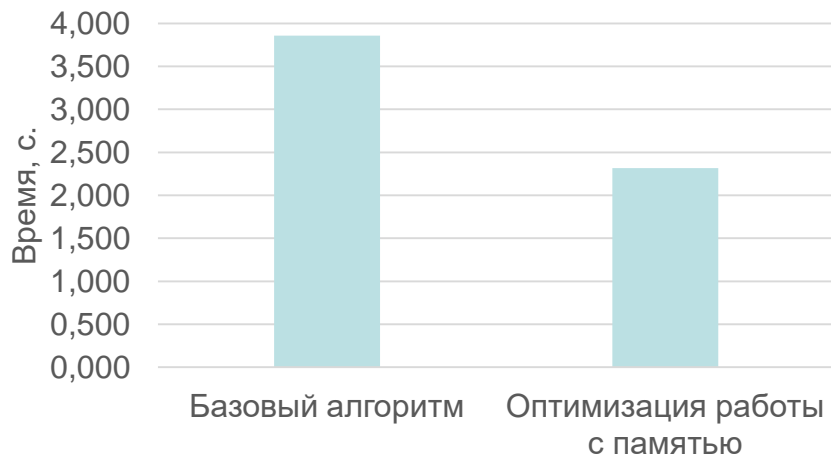
## □ Реализация фильтра с новым порядком циклов:

```
for (int i = 0; i < h - sizeFilter; i++) {
    for (int j = 0; j < w - sizeFilter; j++) {
        int i_d = i + middle;
        int j_d = j + middle;
        for (int i_f = 0; i_f < sizeFilter; i_f++) {
            for (int j_f = 0; j_f < sizeFilter; j_f++) {
                for (int c = 0; c < cntChannel; c++) {
                    int pos_i = (i + i_f) * (w * cntChannel);
                    int pos_j = (j + j_f) * (cntChannel);
                    distData[(i_d * w + j_d) * cntChannel + c] +=
                        srcData[pos_i + pos_j + c] * filter[i_f * sizeFilter + j_f];
                }
            }
        }
    }
}
```

# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Время работы алгоритма: 2.314 с.



# 3. АЛГОРИТМИЧЕСКАЯ ОПТИМИЗАЦИЯ

# Алгоритмическая оптимизация

- Возможно следующее разложение ядра фильтра Гаусса\*:

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

- Пример использования разложения:

2	2	4	2	2	1
5	2	4	4	2	0
5	1	1	0	5	1
0	4	0	0	3	3
5	0	5	1	4	0
1	5	3	2	4	2

1	2	1
2	4	2
1	2	1

*	*	*	*	*	*
*	44	43	44	34	*
*	37	24	29	39	*
*	34	22	23	38	*
*	42	39	36	39	*
*	*	*	*	*	*

— Какова сложность каждой версии реализации фильтра?

2	2	4	2	2	1
5	2	4	4	2	0
5	1	1	0	5	1
0	4	0	0	3	3
5	0	5	1	4	0
1	5	3	2	4	2

1
2
1

*	*	*	*	*	*
17	7	13	10	11	2
15	8	6	4	15	5
10	9	6	1	15	7
11	9	13	4	15	5
*	*	*	*	*	*

1	2	1
---	---	---

*	*	*	*	*	*
*	44	43	44	34	*
*	37	24	29	39	*
*	34	22	23	38	*
*	42	39	36	39	*
*	*	*	*	*	*

\*М. Moradifar and A. Shahbahrani, "Performance Improvement of Gaussian Filter using SIMD Technology," *International Conference on Machine Vision and Image Processing (MVIP)*(2020): 1-6.

# Программная реализация...

## ❑ Заполнение вектора фильтра:

```
float* create1DFilter(int n, float sigma)
{
    float* filter = new float[n];
    float coeff = 1.0f / (sqrtf(2.0f * PI) * sigma);
    int middle = n / 2;

    for (int i = 0; i < n; i++)
    {
        float x = i - middle;
        filter[i] = coeff * expf(-(x * x) / (2.0f * sigma * sigma));
    }

    return filter;
}
```



# Программная реализация...

## □ Инициализация временной матрицы:

```
float* filter1D = create1DFilter(sizeFilter, sigma);
cv::Mat tmp;
src.copyTo(tmp);
int w = src.size().width;
int h = src.size().height;
float* srcData = (float*)src.data;
float* tmpData = (float*)tmp.data;
float* distData = (float*)dist.data;

int middle = sizeFilter / 2;
const int cntChannel = 3;

for (int i = 0; i < h - sizeFilter; i++) {
    for (int j = 0; j < w * cntChannel; j++) {
        int i_d = i + middle;
        tmpData[i_d * w * cntChannel + j] = 0.0f;
    }
}
```

# Программная реализация...

## □ Первая фаза применения фильтра:

```
for (int i = 0; i < h - sizeFilter; i++)
{
    int i_d = i + middle;
    for (int j = 0; j < w * cntChannel; j++)
    {
        for (int i_f = 0; i_f < sizeFilter; i_f++)
        {
            int pos_i = (i + i_f) * (w * cntChannel);
            int pos_j = j;
            tmpData[i_d * w * cntChannel + j] += srcData[pos_i + pos_j] * filter1D[i_f];
        }
    }
}
```

# Программная реализация...

## □ Инициализация результирующей матрицы:

```
for (int i = 0; i < h - sizeFilter; i++)
{
    for (int j = 0; j < w - sizeFilter; j++)
    {
        int i_d = i + middle;
        int j_d = j + middle;
        for (int c = 0; c < cntChannel; c++)
        {
            distData[(i_d * w + j_d) * cntChannel + c] = 0.f;
        }
    }
}
```

# Программная реализация

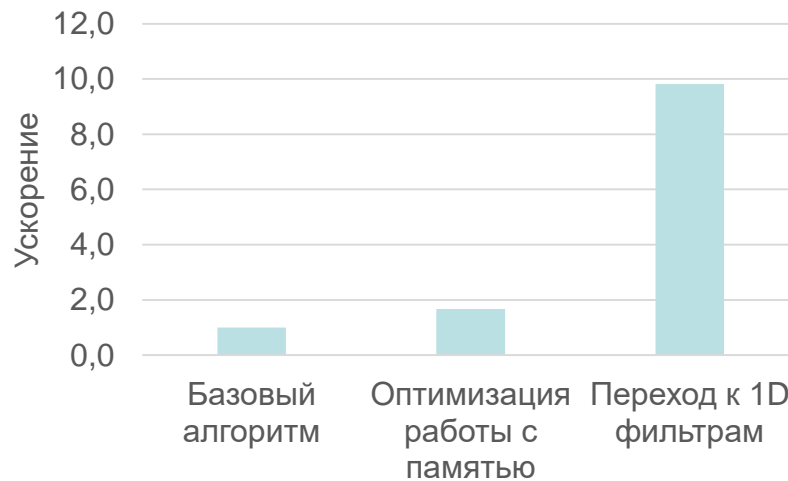
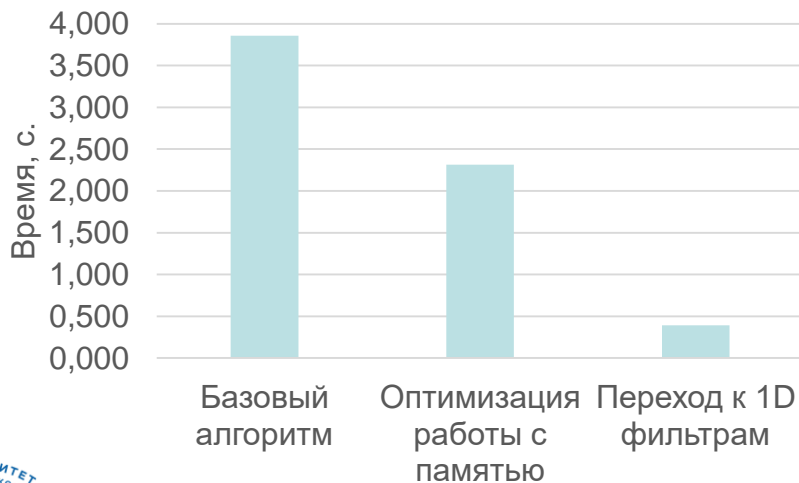
## □ Вторая фаза применения фильтра:

```
for (int i = 0; i < h - sizeFilter; i++)
{
    for (int j = 0; j < w - sizeFilter; j++)
    {
        int i_d = i + middle;
        int j_d = j + middle;
        for (int j_f = 0; j_f < sizeFilter; j_f++)
        {
            for (int c = 0; c < cntChannel; c++)
            {
                int pos_i = (i) * (w * cntChannel);
                int pos_j = (j + j_f) * (cntChannel);
                distData[(i_d * w + j_d) * cntChannel + c] +=
                    tmpData[pos_i + pos_j + c] * filter1D[j_f];
            }
        }
    }
}
```

# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Время работы алгоритма: 0.393 с.



## 4. ВЕКТОРИЗАЦИЯ ЦИКЛОВ

# Векторизация циклов...

## ❑ Отчет векторизации (первая фаза)

```
for (int i = 0; i < h - sizeFilter; i++) {  
    int i_d = i + middle;  
    for (int j = 0; j < w * cntChannel; j++) {  
        for (int i_f = 0; i_f < sizeFilter; i_f++) {  
            int pos_i = (i + i_f) * (w * cntChannel);  
            tmpData[i_d * w * cntChannel + j] += srcData[pos_i + j] * filter1D[i_f];  
        }  
    }  
}
```

Циклы можно поменять местами

Есть зависимость между итерациями

```
...  
LOOP BEGIN at /home/kozinov_e/OpencvImg/filter1D.cpp(40,7)  
    remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence  
is shown below. Use level 5 report for details  
    remark #15346: vector dependence: assumed FLOW dependence between tmpData[(i_d*_width)*3+j] (44:9)  
and tmpData[(i_d*_width)*3+j] (44:9)  
LOOP END  
...
```

# Векторизация циклов...

## □ Реализация первой фазы

```
for (int i = 0; i < h - sizeFilter; i++)
{
    int i_d = i + middle;
    for (int i_f = 0; i_f < sizeFilter; i_f++)
    {
#pragma omp simd
        for (int j = 0; j < w * cntChannel; j++)
        {
            int pos_i = (i + i_f) * (w * cntChannel);
            tmpData[i_d * w * cntChannel + j] += srcData[pos_i + j] * filter1D[i_f];
        }
    }
}
```



# Векторизация циклов...

## □ Отчет векторизации (вторая фаза)

```
for (int i = 0; i < h; i++) {  
    for (int j = 0; j < w - sizeFilter; j++) {  
        int j_d = j + middle;  
        for (int j_f = 0; j_f < sizeFilter; j_f++) {  
            for (int c = 0; c < cntChannel; c++) {  
                int pos_i = (i) * (w * cntChannel);  
                int pos_j = (j + j_f) * (cntChannel) + c;  
                distData[(i * w + j_d) * cntChannel + c] += tmpData[pos_i + pos_j] * filter1D[j_f];  
            }  
        }  
    }  
}
```

Объединенные циклы можно поменять местами

Циклы можно объединить

Есть зависимость между итерациями

LOOP BEGIN at /home/kozinov\_e/OpenCVImg/filter1D.cpp(78,9)

remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence is shown below. Use level 5 report for details

remark #15346: vector dependence: assumed FLOW dependence between distData[(i\*\_width+j\_d)\*3+c] (83:11) and filter1D[j\_f] (83:11)

remark #25436: completely unrolled by 3

LOOP END

# Векторизация циклов

## □ Реализация второй фазы

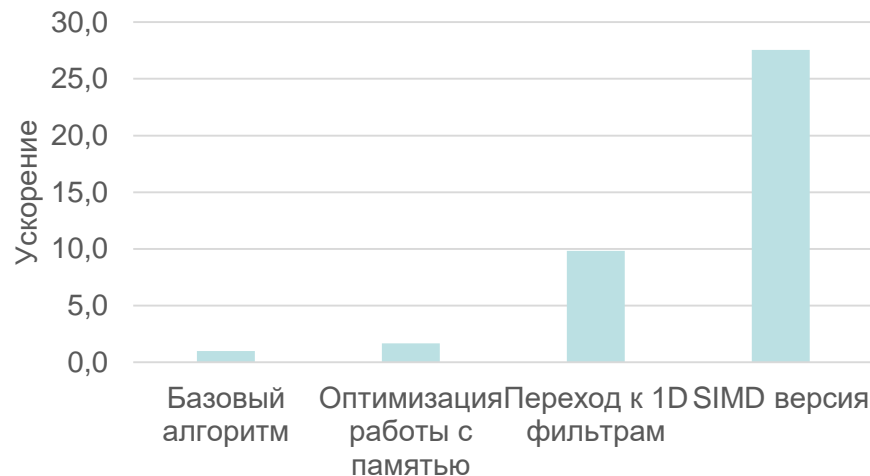
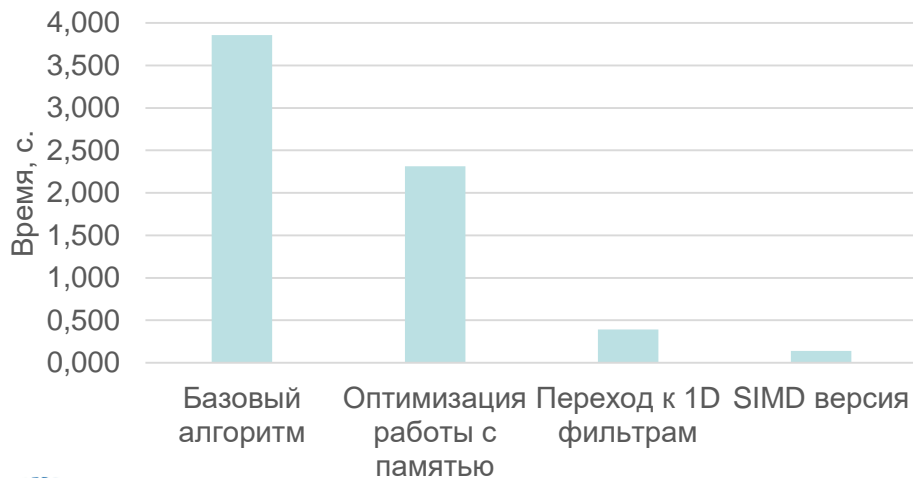
```
for (int i = 0; i < h; i++)
{
    for (int j_f = 0; j_f < sizeFilter; j_f++)
    {
#pragma omp simd
        for (int j = 0; j < (w - sizeFilter) * cntChannel; j++)
        {
            int j_d = j + middle * cntChannel;
            int pos_i = i * w * cntChannel;
            int pos_j = j + j_f * cntChannel;

            distData[i * w * cntChannel + j_d] +=
                tmpData[pos_i + pos_j] * filter1D[j_f];
        }
    }
}
```

# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Время работы алгоритма: 0.140 с. (ускорение 2.8 относительно версии без SIMD)



# 5. РАСПАРАЛЛЕЛИВАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

# Распараллеливание программной реализации

- ❑ Обе фазы применения фильтра Гаусса могут быть распараллелены по внешнему циклу
  - Итерации внешнего цикла независимые, как следствие, достаточно применить

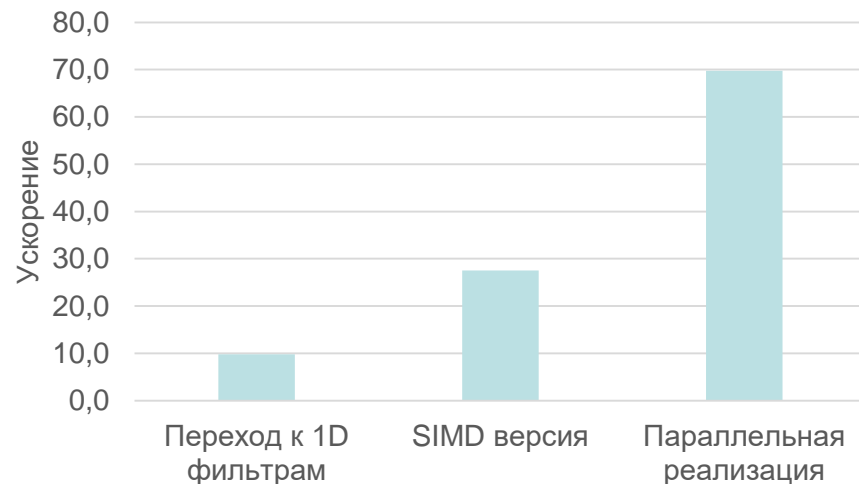
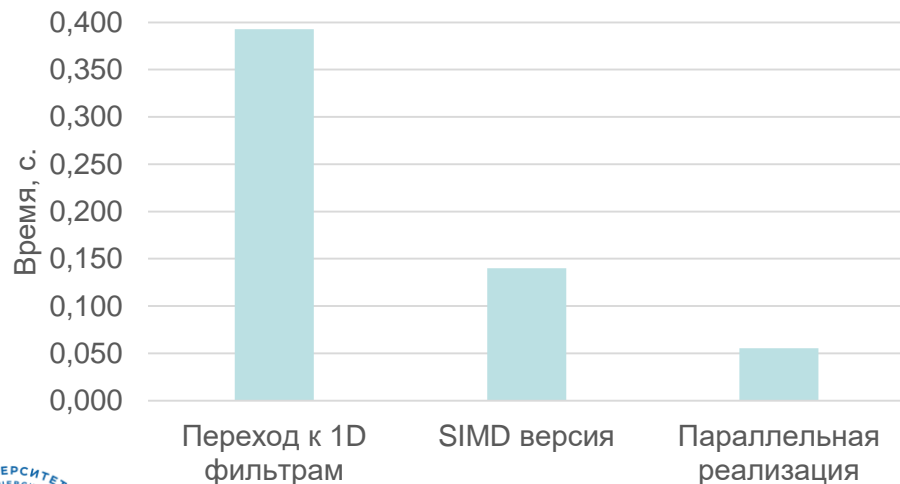
```
#pragma omp parallel for
```

- Реализацию предлагается выполнить самостоятельно

# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Время работы алгоритма: 0.055 с. (дополнительное ускорение 2.5 раза)



# **6. ИСПОЛЬЗОВАНИЕ ОПТИМИЗИРОВАННОЙ РЕАЛИЗАЦИИ ФИЛЬТРА НА ПРИМЕРЕ INTEL INTEGRATED PERFORMANCE PRIMITIVES (INTEL IPP)**

# Программная реализация...

## ❑ Изменение в CMake для использования IPP:

```
cmake_minimum_required(VERSION 3.20)
PROJECT (hpc_lab)

set(CMAKE_CXX_STANDARD 11)

find_package(OpenCV REQUIRED)
find_package(OpenMP REQUIRED)
find_package(IPP REQUIRED)

include_directories( ${OpenCV_INCLUDE_DIRS} )
add_compile_options(-qopt-report-phase=vec,loop -qopt-report=2 )

file(GLOB TARGET_SRC "*.c*")
file(GLOB TARGET_HD "*.h*")
add_executable( filter ${TARGET_SRC} ${TARGET_HD} )

target_link_libraries(filter ${OpenCV_LIBS} )
target_link_libraries(filter OpenMP::OpenMP_CXX)
target_link_libraries(filter IPP::ippcore IPP::ippcv IPP::ipps IPP::ippi)
```



# Программная реализация...

## □ Объявление структур данных:

```
void filterIPP(cv::Mat& dist, cv::Mat& src, float sigma, int sizeFilter) {  
    int w = src.size().width;  
    int h = src.size().height;  
    float* srcData = (float *)src.data;  
    float* distData = (float*)dist.data;  
    int numChannels = 3;  
  
    IppStatus status = ippStsNoErr; // Статус выполнения операции  
    int srcStep = w * numChannels * sizeof(float);  
    int dstStep = w * numChannels * sizeof(float);  
    IppiSize roiSize = {w, h}; // Размер изображения в пикселях  
    Ipp32u kernelSize = sizeFilter; // Размер фильтра  
    Ipp8u* pBuffer = NULL; // Указатели на временную память  
    IppFilterGaussianSpec* pSpec = NULL;  
    int iTmpBufSize = 0, iSpecSize = 0; // Размеры временных буферов  
    IppiBorderType borderType = ippBorderConst; // Способ обработки пикселей на границе  
                                                (не менять значения)  
  
    Ipp32f borderValue[3]{ 0 };
```

# Программная реализация

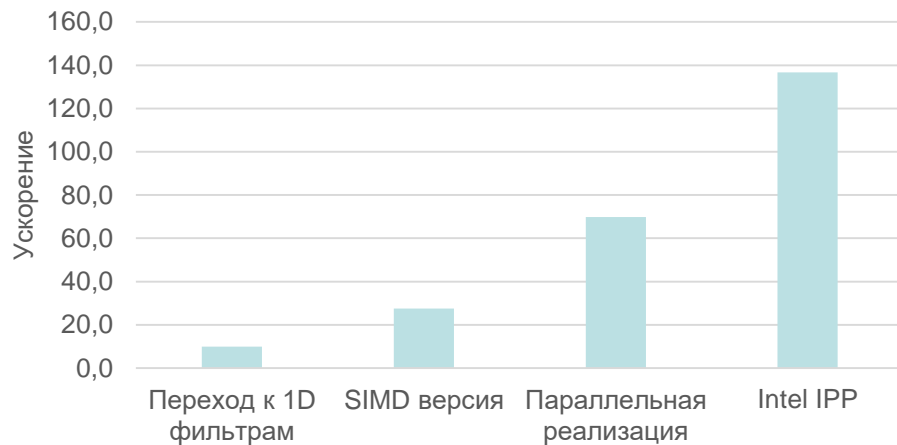
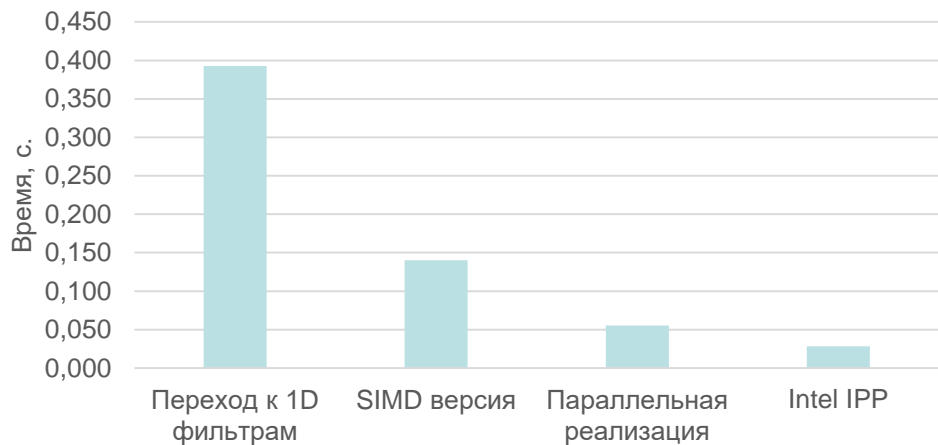
## □ Вызов реализации фильтра:

```
// Определение размеров временных буферов
status = ippiFilterGaussianGetBufferSize(roiSize, kernelSize, ipp32f,
    numChannels, &iSpecSize, &iTmpBufSize);
// Выделение памяти
pSpec = (IppFilterGaussianSpec*)ippsMalloc_32f(iSpecSize);
pBuffer = ippsMalloc_8u(iTmpBufSize);
// Инициализация фильтра
status = ippiFilterGaussianInit(roiSize, kernelSize, sigma,
    borderType, ipp32f, numChannels, pSpec, pBuffer);
// Применение фильтра к изображению
status = ippiFilterGaussianBorder_32f_C3R(srcData, srcStep, dstData, dstStep,
    roiSize, borderValue, pSpec, pBuffer);
// Освобождение выделенной памяти
ippsFree(pBuffer);
ippsFree(pSpec);
}
```

# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Время работы алгоритма: 0.028 с. (ускорение 2 раза)



# Результаты вычислительных экспериментов

## □ Решаемая задача:

- Размер тестового изображения: 2560x2027
- Размер окна фильтра: 19x19
- Изображение цветное: 3 канала
- Сводная таблица результатов вычислительных экспериментов

	Время, с.	Ускорение
Базовый алгоритм	3.857	1.0
Оптимизация работы с памятью	2.315	1.7
Переход к 1D фильтрам	0.393	9.8
SIMD версия	0.140	27.5
Параллельная реализация	0.055	69.8
Intel IPP	0.028	136.7

# Заключение

- ❑ В рамках работы решалась задача реализации фильтра Гаусса
- ❑ В процессе реализации были выполнены основные шаги оптимизации приложений:
  - Оптимизация работы с памятью
  - Алгоритмическая оптимизация
  - Векторизация циклов
  - Распараллеливание программной реализации
- ❑ На последнем этапе произведено сравнение полученной реализации с реализацией из библиотеки Intel IPP
  - Результаты вычислительных экспериментов показали сравнимые результаты производительности итоговой версии реализации фильтра и фильтра из Intel IPP

# Литература

1. M. Moradifar and A. Shahbahrami, "Performance Improvement of Gaussian Filter using SIMD Technology," *International Conference on Machine Vision and Image Processing (MVIP)*(2020): 1-6.
2. Amiri, Hossein, and Asadollah Shahbahrami. "SIMD programming using Intel vector extensions." *Journal of Parallel and Distributed Computing* 135 (2020): 83-100.
3. Intel, R. "Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4." (2018).
4. OpenCV tutorial: [https://docs.opencv.org/4.x/d9/df8/tutorial\\_root.html](https://docs.opencv.org/4.x/d9/df8/tutorial_root.html)
5. OpenMP tutorial: <https://www.openmp.org/resources/tutorials-articles/>
6. Intel IPP tutorial:  
<https://www.intel.com/content/www/us/en/develop/documentation/get-started-with-ipp-for-oneapi-windows/top.html>

# Авторский коллектив

---

- ❑ Мееров Иосиф Борисович, к.т.н., доцент, зам. зав. каф. МОСТ
- ❑ Сысоев Александр Владимирович, к.т.н., доцент каф. МОСТ
- ❑ Линев Алексей Владимирович, зав. лаб. интернета вещей, каф. ПРИН
- ❑ Волокитин Валентин Дмитриевич, программист лаборатории СТиВВ, каф. МОСТ
- ❑ Козинов Евгений Александрович, к.т.н., преподаватель каф. МОСТ
- ❑ Панова Елена Анатольевна, инженер лаборатории СТиВВ, каф. МОСТ

# Контакты

---

Нижегородский государственный университет

<http://www.unn.ru>

Центр компетенций oneAPI в ННГУ

<http://hpc-education.unn.ru/ru/центр-компетенций-oneapi-в-ннгу>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>