Введение в математические библиотеки Intel

26 ноября 2020

Бакшаев Андрей

инженер по разработке ПО

Integrated Performance Primitives, Intel

Обзор

- Введение
- MKL
- DAAL
- IPP

Введение

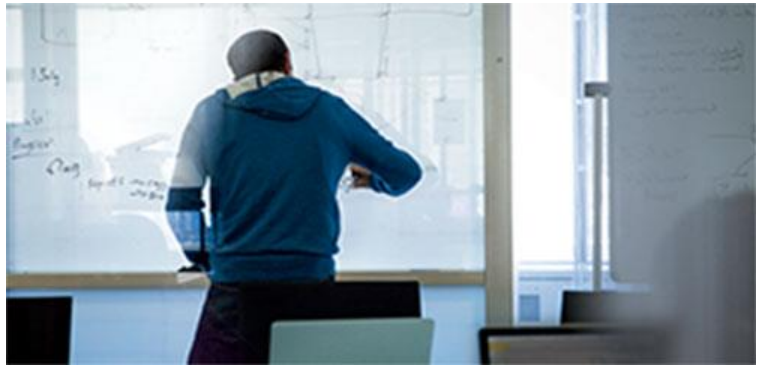
Математический Библиотеки Intel Performance Libraries (IPL) позволяют ускорять приложения и сокращать время на разработку. Все эти библиотеки бесплатные и позволяют добиться впечатляющей производительности на процессорах Intel®.

Intel® Integrated Performance Primitives (IPP)

Оптимизированная библиотека для обработки изображений, сигналов, сжатия данных, криптографии. Работает на многих операционных системах.

Intel® Math Kernel Library (MKL)

Широкоизвестная математическая библиотека для умножения матриц, систем линейных уравнений.



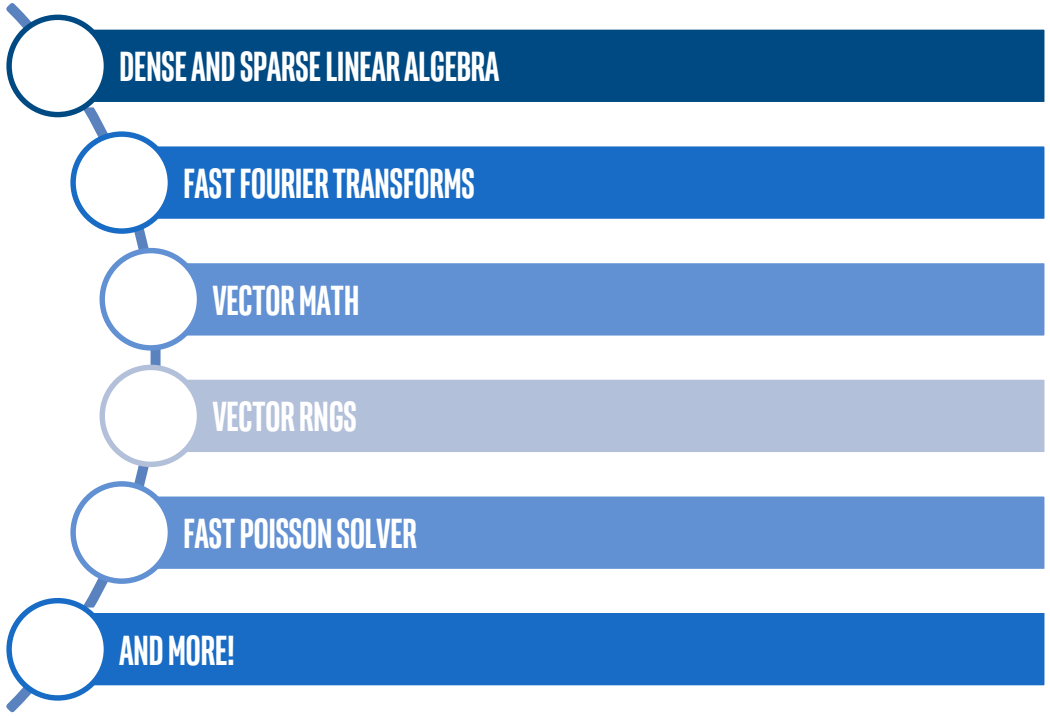
Intel® Data Analytics Acceleration Library (DAAL)

Библиотека для машинного обучения и обработки больших данных.

and more...

Intel® MKL

- Ускоряет научные, инженерные, финансовые и приложения для машинного обучения. Содержит высокооптимизированные, распареллеленные и векторизованные математические функции
- Содержит стандартные алгоритмы для плотной и разреженной линейной алгебры(BLAS, LAPACK, PARDISO), FFTs, векторную математику, статистические функции, глубокое обучение, сплайны и другие функции
- Автоматически определяет процессор и запускает наиболее подходящий код библиотеки
- Использует все возможности векторизиции и рационально использует кэш процессора
- Автоматически задействует все ядра процессора
- Масштабируется от процессора до кластера
- Fortran и C инерфейсы



Operating System: Windows*, Linux*, MacOS[1]*

MKL подробнее

LINEAR ALGEBRA

- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Iterative sparse solvers
- PARDISO*
- Cluster Sparse Solver

FFTS

- Multidimensional
- FFTW interfaces
- Cluster FFT

VECTOR RNGS

- Congruential
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neirderreiter
- Non-deterministic

SUMMARY STATISTICS

- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

VECTOR MATH

- Trigonometric
- Hyperbolic
- Exponential
- Log
- Power
- Root

AND MORE

- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

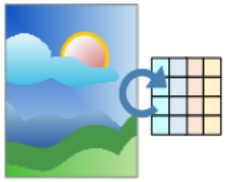
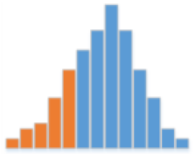
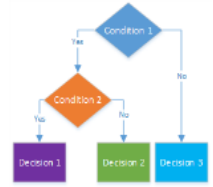
Intel® Data Analytic Acceleration Library (DAAL)

- Оптимизированная библиотека для классических алгоритмов машинного обучения
- Специальные режимы работы с очень большими объемами входных данных.
- Интерфейсы для языков Python*, C++, and Java* и сред машиннго обучения Spark* и Hadoop*

- Распространяется как Open Source продукт
- Через Github: https://github.com/intel/daal
- А также YUM, APT-GET, PIPY and Conda, Maven repositories

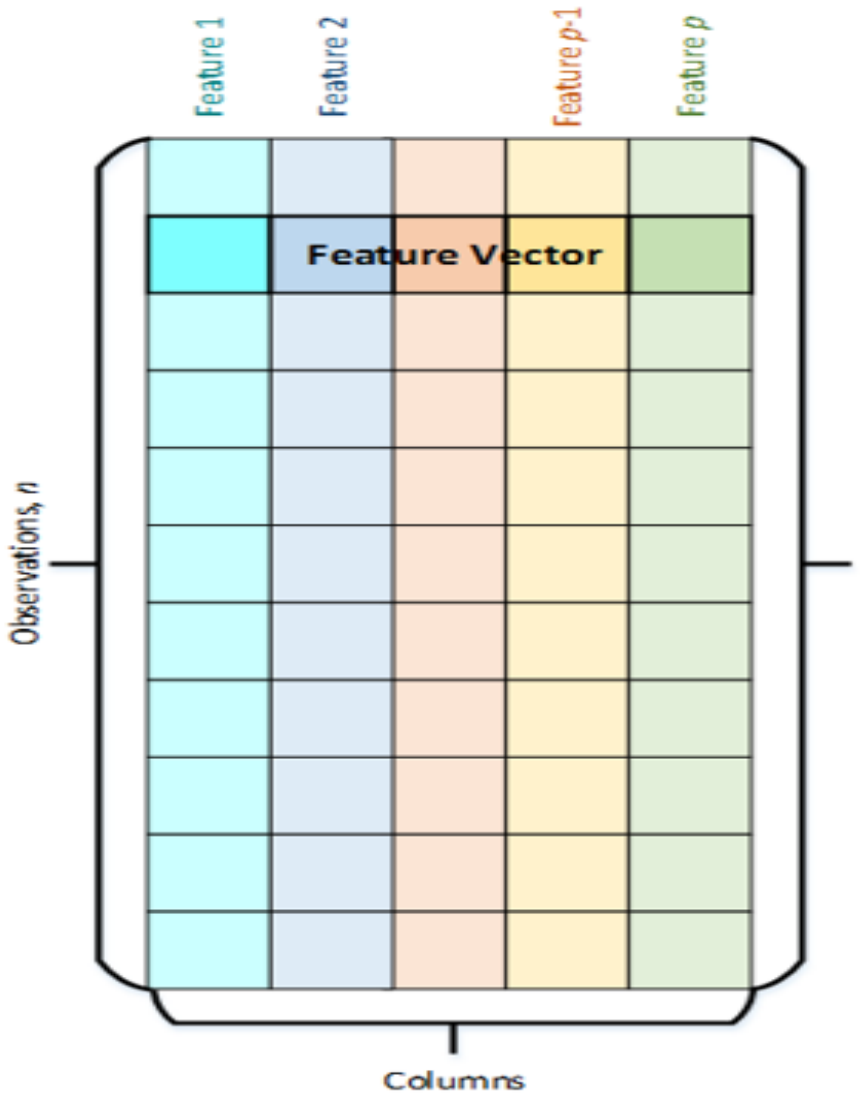
- Доступен в виде статической и динамической библиотек
- Многоплатформенность

Идея DAAL заключается в том, чтобы создать одну библиотеку для работы на всех этапах аналитики данных, исключая подобные многослойные реализации, при этом оптимизировать её под железо. Использование этого решения позволяет получить значительный прирост производительности.

Pre-processing/ Подготовка	Transformation/ 2х мерная таблица	Analysis/ Анализ	Modeling/ Моделирование	Validation/ Проверка	Decision Making/ Принятие решений
					
Decompression,	Aggregation,	Summary Statistics	Machine Learning (Training)	Hypothesis testing	Forecasting
Filtering, Normalization	Dimension Reduction	Clustering, etc.	Parameter Estimation	Model errors	Decision Trees, etc.
			Simulation		

Learn More: software.intel.com/daal

DAAL. Управление данными



Библиотека состоит и трех основных компонент:

Управление данными;

Алгоритмы и

Сервисы.

Feature – это одно из свойств объекта. (Цвет глаз, возраст, рост, температура воды и так далее)

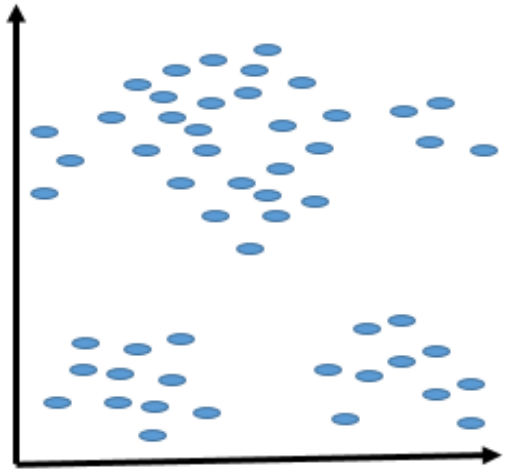
Feature Vector - набор свойств. Размер **P**.

Observations - множество наблюдений(векторов). Количество **N**.

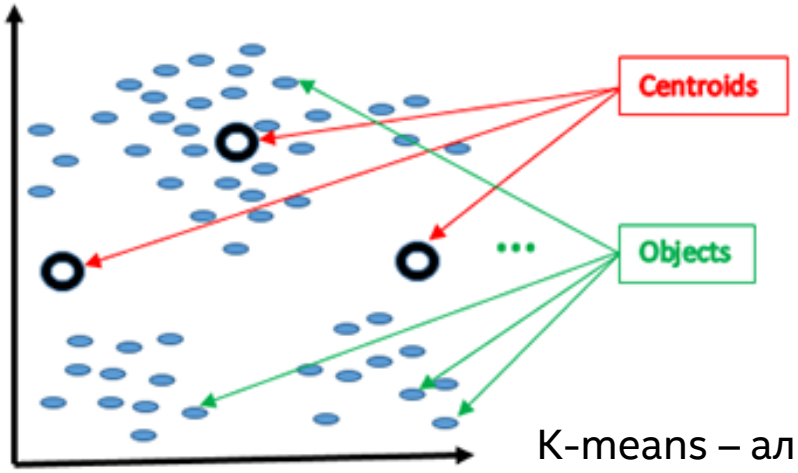
В DAAL данные хранятся в виде таблиц, в которых строками являются наблюдения (**Observations**), а столбцами – свойства (**Features**).

https://m.habr.com/ru/company/intel/blog/265347/

DAAL. Алгоритмы



Алгоритмы состоят из классов, реализующих анализ данных и их моделирование. Они включают декомпозицию, кластеризацию, классификацию и регрессионные алгоритмы, а также ассоциативные правила.



batch processing – все данные доступны сразу и обрабатываются целиком

online processing - данные доступны не полностью и обрабатываются блоками по мере поступления

distributed processing - данные обрабатываются на нескольких вычислительных узлах и объединяются на итоговом узле

K-means – алгоритм кластеризации данных https://en.wikipedia.org/wiki/K-means_clustering

https://software.intel.com/en-us/articles/improve-performance-of-k-means-with-intel-data-analytics-acceleration-library

Intel® IPP

Что такое Intel® IPP?

Intel IPP это около 6000 функций для обрабоки ***изображений, сигналов, сжатия данных и криптографии***

Почему используют Intel® IPP?

- Высокая производительность
- Стандартизованный API
- Регулярные обновления
- Подробная документация
- Техподдержка
- Многоплатформенность

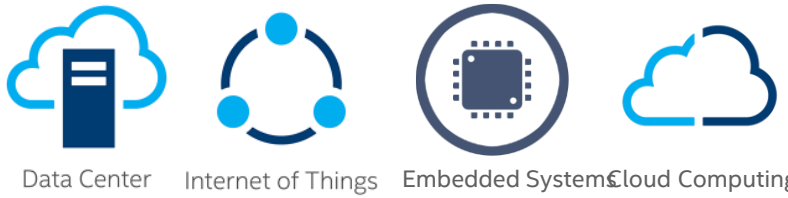
Как поставляется Intel® IPP?

- Intel® Parallel Studio XE
- Intel® System Studio
- Free Tools Program
- IPP Crypto open source
- YUM, APT-GET и Conda packages

Оптимизация для

Supports

Где используется



Обработка изображений

- Медицина
- Компьютерное зрение
- Видеонаблюдение
- Графические редактор
- Поиск объектов на видео

Обработка сигналов

- Обработка аудиосигналов
- Удаление эхо
- Сортировка данных
- Анализ спектров сигнала

Сжатие данных и криптография

- Дата центры
- Идентификация
- Электронная подпись
- Информационная безопасность

Find out more at: ***http://software.intel.com/intel-ipp***
http://software.intel.com/en-us/forums/intel-integrated-performance-primitives

IPP подробнее

Обработка изображений ippIP

Компьютерное зрение ippCV

Цветовые пространства ippCC

Обработка сигналов ippSP

Векторная математика ippVM

Сжатие данных ippDC

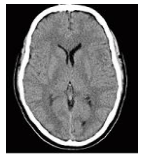
Криптография ippCrypto

Обработка строк ippCH

Intel® IPP Benefits to Applications

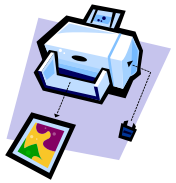
Cloud and Server application
- Web image processing(resize, filtering, etc.)
- Web data compression and transferring, data encryption/decryption

Medical Images
- CT, MRI signal processing
- Medical image processing



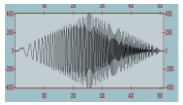
Storage
- Storage data compression
- Storage data encryption/decryption

Print Imaging
- Image enhancement and correction
- Data compression

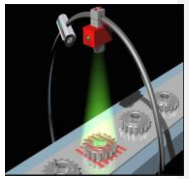


Digital Surveillance
- Computer vision
- Image recognition

Signal Processing
- Seismic data analysis, radar and sonar signal processing.

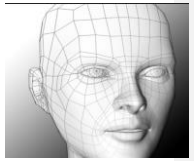


- Machine Vision
  - Image filtering, segmentation
  - Edge detection, pattern recognition



- In-Vehicle Infotainment
  - Image and audio data processing

- Biometric Identification
  - Biometric image and signal processing



- Visual Search
  - Examining image content(color, shape, texture…)

- Communication
  - Wireless communication single processing
  - CRC and MIMO functions for communication.

- And More
  - Digital media, security, mobile……

Gets Good Performance with Intel® IPP

In popular apps like WeChat, QQ*, and QQ Album* the volume of newly generated images reach about 100 petabytes. Some users may try to upload illegal images (e.g., porn). The system has to run a check on each image to try to block them. Imagine trying to search through 100 petabytes of data.*

IPP filter function (ipp_filter2D) took 9ms to perform the operation when compared to 143ms with openCV. ***The IPP filter2D is 15x faster*** *than the OpenCV* plain code.*

Tencent doubled the speed of its image filter System

JD.com business has grown rapidly, from offering approximately 1.5 million SKUs in 2011 to approximately 25.7 million in 2013. Today, JD.com must handle petabytes of data, which takes an efficient, robust, distributed file system.

JD.com ***speeds up its image processing 17x – handling 300,000 images in 162 seconds instead of 2800 seconds.***

JD.com sped image processing with Intel® IPP

More Case Studies

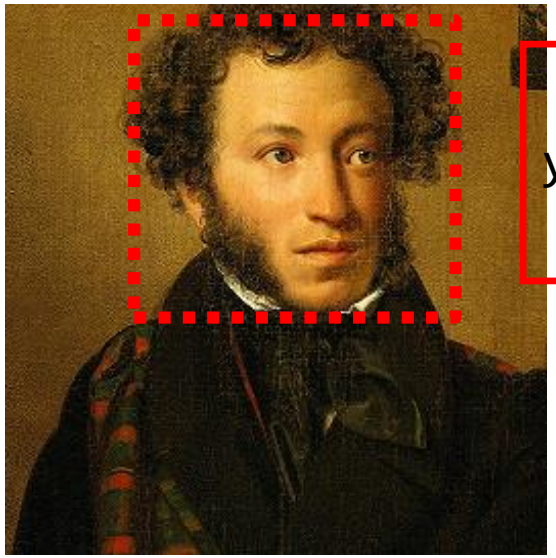

intel

Backup

Часть II. Примеры повышения производительности кода фильтра Blur

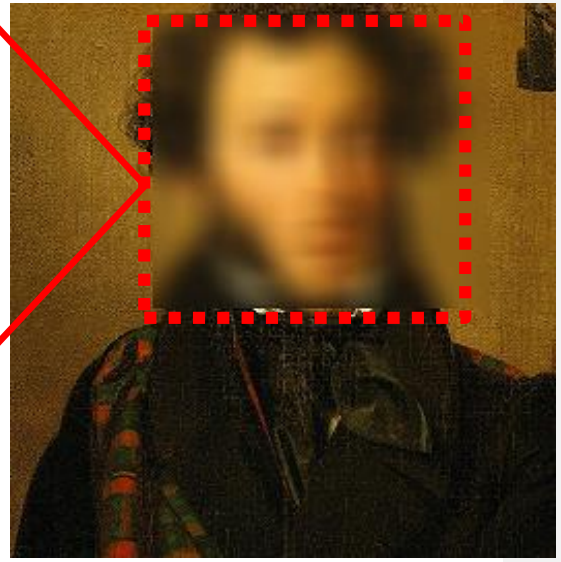
- Краткое описание фильтра размытия Blur
- Первоначальная реализация
- Векторизация и ее применения
- Алгоритмическая оптимизация
- Распараллеливание на несколько потоков с помощью OpenMP

Фильтр размытия(Blur)



```
(x(-1,-1) + x(-1,0) +x(-1,1)+
y(0,0)= x( 0,-1) + x( 0,0) +x( 0,1)+  / 9.0
x( 1,-1) + x( 1,0) +x( 1,1) )
```

- **Фильтр размытия** усредняет изображение по соседним пикселям.



Обладает эффектом удаления мелких деталей на изображении и широко используется в графических редакторах.

Исходная реализация.

```
void blur_c3(uchar* src, uchar* dst, int h, int w) {
 float div = kw*kh;
 for (i = 0; i < h; i++) {
   for (j = 0; j < w; j++) {
     uchar* s = src + 3 * j;
     uchar* d = dst + 3 * j;
     float sum0 = 0, sum1=0, sum2=0;
     for (y = 0; y < kh; y++) {
      for (x = 0; x < kw; x++) {
       sum0 = sum0 + s[3*w*y + 3 * x + 0];
       sum1 = sum1 + s[3*w*y + 3 * x + 1];
       sum2 = sum2 + s[3*w*y + 3 * x + 2];
      }
     }
     d[0] = sum0 / div;  //r channel
     d[1] = sum1 / div;  //g channel
     d[2] = sum2 / div;  //b channel
    }
    src = src + 3*w;
    dst = dst + 3*w;
    } }
```

Простейшая реализация фильтра Blur на языке C для 3 канального RGB изображения.

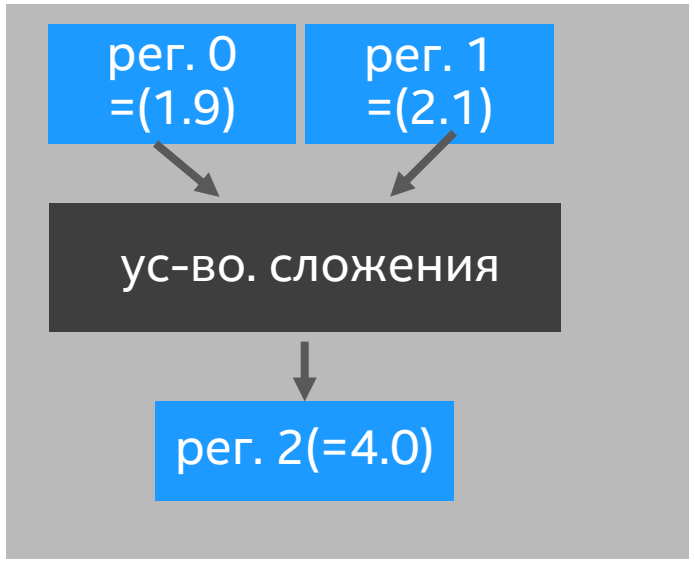
CPU: Intel Xeon Silver 4116 2.1Ghz

Размер изображения: 960 x 540 * 3

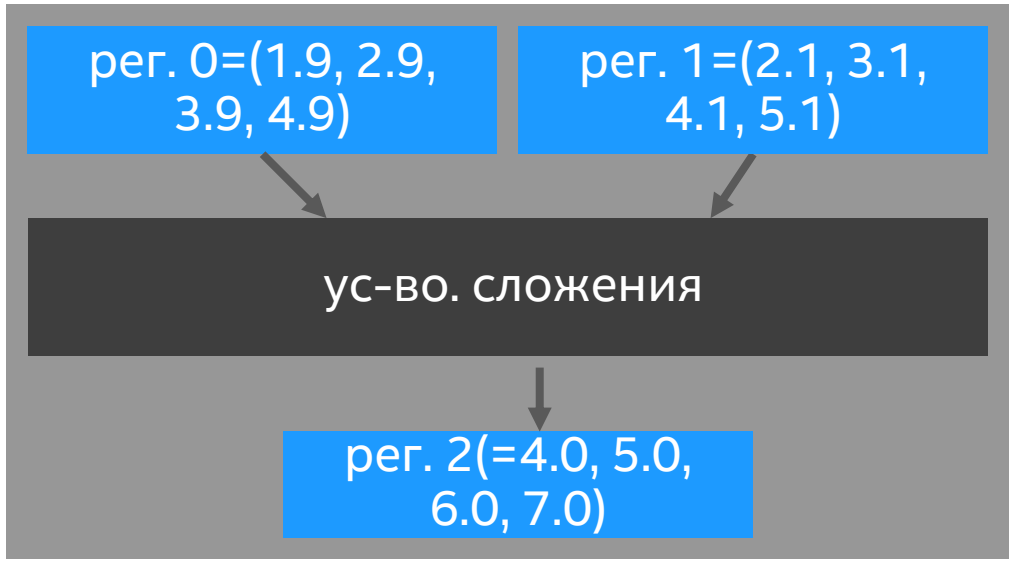
Время: **46.74** млн. тактов

Векторизация кода и интринсики

Обычный процессор



Векторный процессор



- 4/8/../64 элементов в регистре;
- складываются/умножаются одновременно
- SSE 4 floats
- AVX 8 floats
- AVX512 16 floats
- Интринсики для языка C

C код сложения двух массивов:

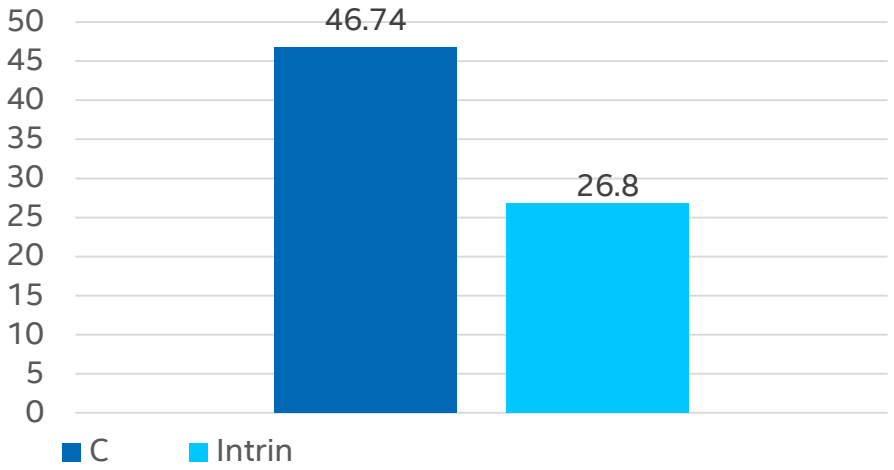
```
float a[N], b[N], c[N];
for (i=0;i<N;i++){
    c[i] = a[i] + b[i];
}
```

```
#include “intrin.h”
float a[N], b[N], c[N];
for (i=0;i<N;i+=4){  // по 4-ре элемента за итерацию
    __m128 x, y, z;
    x = _mm_load_ps(&a[i]);   //загрузить  4 элемента a[i], a[i+1],a[i+2],a[i+3]
    y = _mm_load_ps(&b[i]);   //загрузить 4 из &b[i],...
    z = _mm_add_ps(x, y);     //сложить, (выполняется сразу 4 сложения)
   _mm_store_ps(&c[i],  z);   //сохранить 4 в c[i],c[i+1],c[i+2],c[i+3]
}
```

https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Оптимизированный код

```
void blur_c3_simd(uchar* src, uchar* dst, int h, int w) {
  __m128 vdiv = _mm_set1_ps(1.0/(kw*kh));
  for (i = 0; i < h; i++) {
    for (j=0; j < w; j++) {
      uchar* s = src + 3 * j;
      uchar* d = dst + 3 * j;
      __m128i v0, v1;
      __m128  vf, vs;
      vs = _mm_setzero_ps();
      for (y = 0; y < kh; y++) {
         for (x = 0; x < kw; x++) {
           v0  = _mm_loadu_si128(s + 3*w*y + 3 * x + 0);
           v1  = _mm_cvtepu8_epi32(v0);
           vf  = _mm_cvtepi32_ps(v1);
           vs  = _mm_add_ps(vs, vf);
         } }
      vf   = _mm_mul_ps(vs, vdiv);
      v0   = _mm_cvtps_epi32(vf);
      d[0] = _mm_extract_epi8(v0, 0);
      d[1] = _mm_extract_epi8(v0, 4);
      d[2] = _mm_extract_epi8(v0, 8);
    }
    src = src + 3*w;
    dst = dst + 3*w;
    } }
```



Время: **26.8** млн. тактов

Алгоритмическая оптимизация.

Зачастую модификация вычислений исходного алгоритма может дать гораздо больший прирост производительности, чем низкоуровневая оптимизация на интринзиках.

Blur это среднее между соседними пикселями в квадратной области

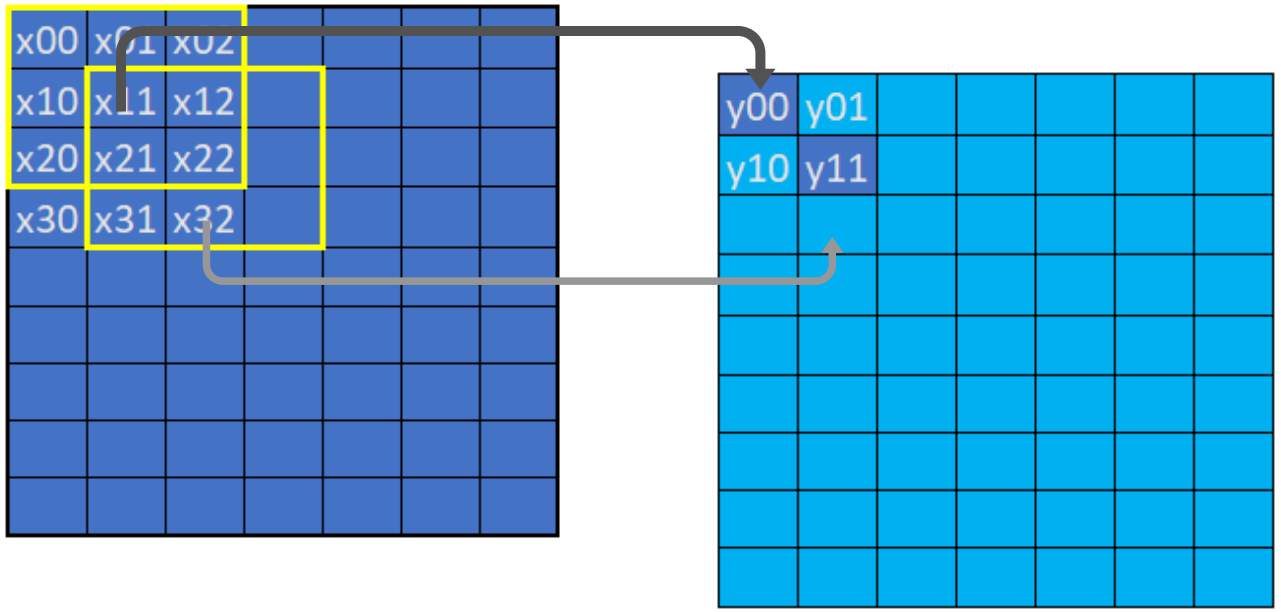
$$1/9 * \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Но он обладает свойством сепарабельности, т.е. изображение можно сначала отфильтровать горизонтальным, а затем вертикальным фильтрами:

$$1/3 * \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \quad \text{и} \quad 1/3 * \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

При этом сокращается количество вычислений на больших размерах фильтра 5x5,7x7,...

Фильтр Blur. Переиспользование данных.



y00=(x00+x01+x02+**x10+x11+x12+x20+x21+x22**) / 9

y01=(**x10+x11+x12+x20+x21+x22**+x30+x31+x32) / 9

Часть суммы можно переиспользовать

Обработка одной строки

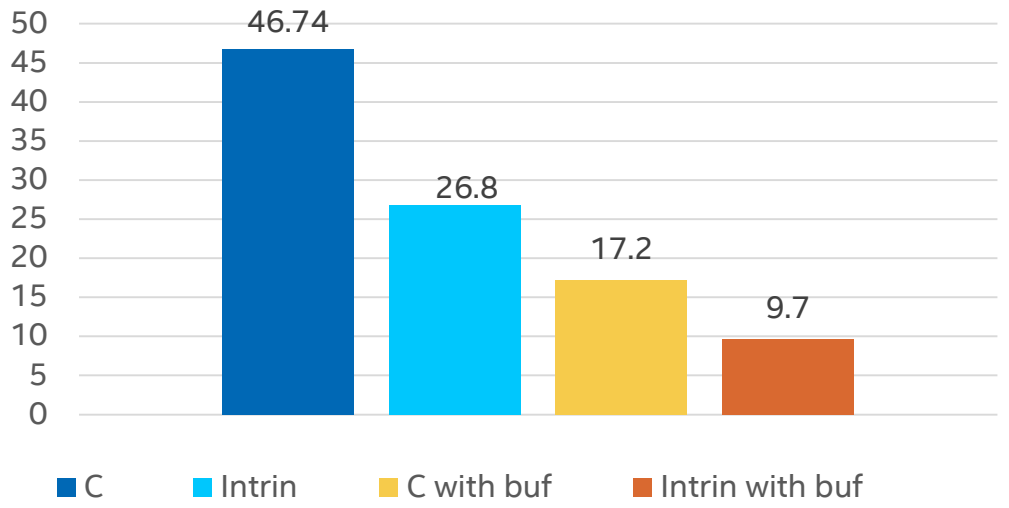
```
void row_c3(uchar* src, int kw,
            int w, float* buf)
{
 int j;
 for (j = 0; j < w; j++) {
  uchar* s = src + 3 * j;
  int x;
  float sum0 = 0, sum1=0, sum2=0;
  for (x = 0; x < kw; x++) {
   sum0 = sum0 + s[3 * x + 0];
   sum1 = sum1 + s[3 * x + 1];
   sum2 = sum2 + s[3 * x + 2];
  }
  buf[3*j + 0] = sum0;
  buf[3*j + 1] = sum1;
  buf[3*j + 2] = sum2;
 }


}
```

Кольцевой буфер.

```
void blur_c3_buf(uchar* src, uchar* dst,
 int h, int w, float* buf[])
{
    for (y = 0; y < kh-1; y++) {
        row_c3(src, kw, w, buf[y]);
        src = src + st;
    }
    int curr_b = 4;
    for (i = 0; i < h; i++) {
        row_c3(src, kw, w, buf[curr_b]);
        for (j = 0; j < w; j++) {
            uchar* d = dst + 3 * j;
            float sum0 = 0, sum1 = 0, sum2 = 0;
            for (y = 0; y < kh; y++) {
                sum0 = sum0 + buf[y][3 * j + 0];
                sum1 = sum1 + buf[y][3 * j + 1];
                sum2 = sum2 + buf[y][3 * j + 2];
            }
            d[0] = (uchar)(sum0 / div);
            d[1] = (uchar)(sum1 / div);
            d[2] = (uchar)(sum2 / div);
        }
        src = src + st;
        dst = dst + st;
        if (curr_b == kh-1) curr_b = 0;
        else curr_b++;
    }}
```

Перед основным циклом складываем горизонтально kh-1 строк. В основном цикле складываем “горизонтально” новую строку, а после “вертикально”.

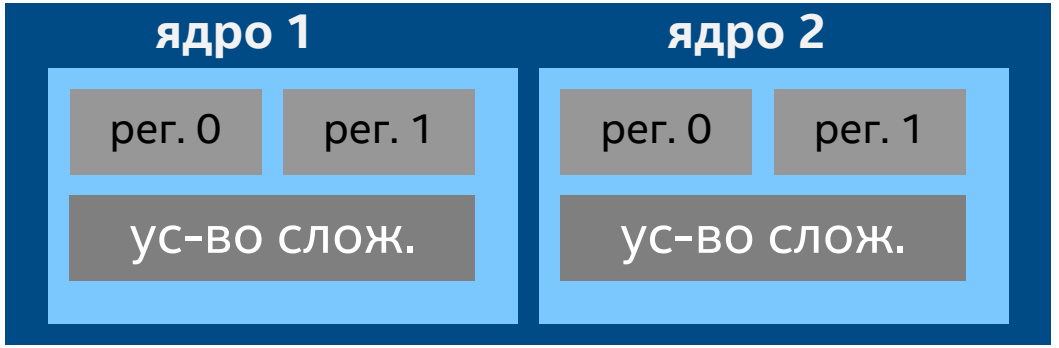


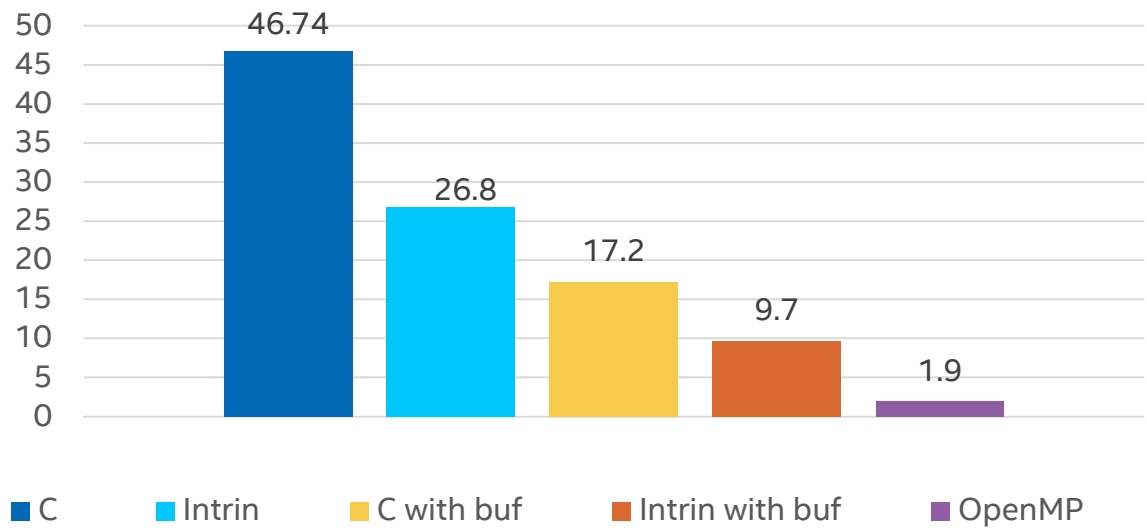
Распараллеливание

```
void blur_c3_simd(uchar* src, uchar* dst, int h, int w)
{
#pragma omp parallel for num_threads(16)
for (i = 0; i < h; i++) {
    for (j=0; j < w; j++) {
     vs = _mm_setzero_ps();
      for (y = 0; y < kh; y++) {
         for (x = 0; x < kw; x++) {
           v0  = _mm_loadu_ps(s+3*w*y+3*x);
           vs  = _mm_add_ps(vs, v0);
          } }
      vf   = _mm_mul_ps(vs, vdiv);
      v0   = _mm_cvtps_epi32(vf);
      d[0] = _mm_extract_epi8(v0, 0);
     }
     src = src + 3*w;
     dst = dst + 3*w;
     } }
```

OpenMP – это расширение языка C, для создания параллельного кода.

Современный многоядерный процессор





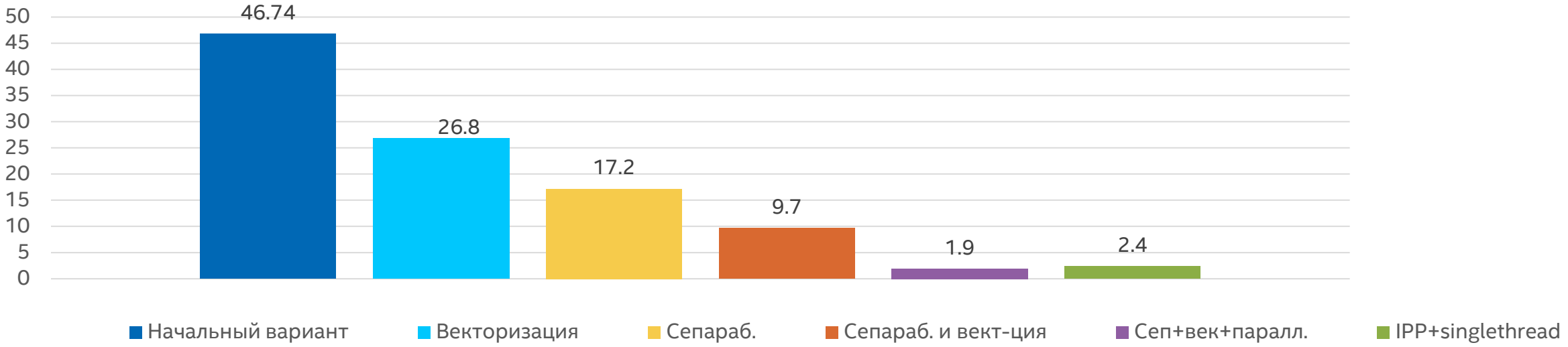
Использование IPP

IppStatus **ippiFilterBoxBorderGetBufferSize**(
IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize)

IppStatus **ippiFilterBoxBorder_8u_C3R**(
Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize,
IppiBorderType border, Ipp8u borderValue[3], Ipp8u* pBuffer)

pSrc - указатель на входное изображение
IppiSize roiSize -“region of interes” размер обрабатываемой области
IppiSize maskSize - размер маски фильтра
int srcStep(dstStep) - шаг между соседними линиями
IppiBorderType border - способ достраивания недостающих пикселей с краев (Repl, Const, InMem)
borderValue[3] - константы для случая бордюра Const
pDst - указатель на обработанное изображение
pBuffer - рабочий буфер
pBufferSize - размер рабочего буфера(зависит от размеров изображения) и возвращается функцией ippiFilterBoxBorderGetBufferSize

Conclusion



- Итоговое ускорение на 16 потоках составляет 24х.
- Эти и многие другие способы оптимизиции кода, а также использование таких инструментов как: Intel Composer, Amplifier, Advisor, TBB позволяет IPL библиотекам MKL, DAAL, IPP, ADL
- добиться их уникальной производительности и быть широко востребованными многими разработчиками ПО.