

Зимняя школа ННГУ по оптимизации алгоритмов компьютерного зрения 2022

SIMD и автовекторизация

Екатерина Антакова,
старший инженер в Интел



Содержание

1. Введение. Что такое SIMD
2. Векторные расширения инструкций
 - Типы данных и регистры
 - Обзор типов операций
3. Векторизация, её диагностика и улучшение
 - 3.1. Как писать код для векторизации. Примеры
 - 3.2. Диагностика векторизации: отчёты и инструменты
 - 3.3. Включение векторизации в разных языках

Цели лекции

- Узнать о векторизации, чтобы её использовать
- Научиться проверять наличие векторизации
- Научиться включать автовекторизацию и улучшать её

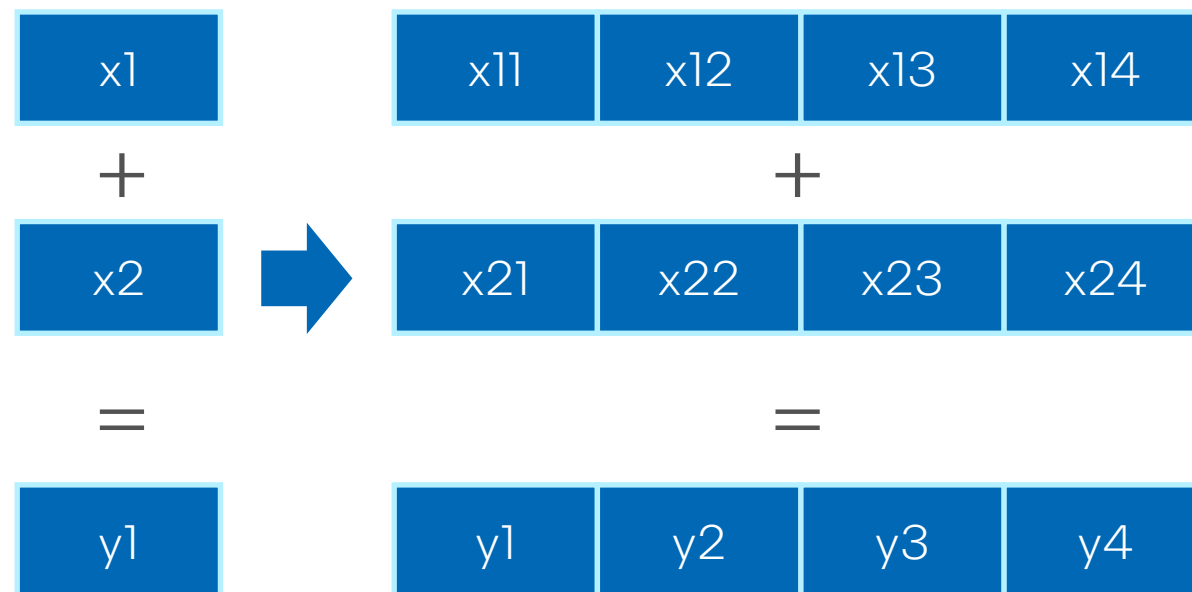
“The ability to perform
Vectorization has become
a key skill”

—Andrew Ng

From “[Neural Networks and Deep Learning](#)” course

Часть 1. Введение / SIMD = Single Instruction Multiple Data

- SIMD – одновременное исполнение одной инструкции на нескольких элементах данных с экономией времени исполнения и мощности
- Примеры инструкций:
 - MMX, SSE, SSE2, SSE3..., AVX, AVX-2, AVX-512
- SIMD реализовано через специальные регистры и векторные инструкции



SIMD и векторизация – синонимы в этой лекции

Часть 1. Введение. Преимущества и недостатки векторизации

- + Ускорение - обработка многих элементов за один процессорный такт
 - Хорошо ускоряется код, ограниченный вычислениями (compute-bound)
- + Векторизация эффективно обрабатывает короткие типы данных
 - Чем меньше тип данных (Int32 -> Int16 -> Int8), тем больше ускорение
- Алгоритмы, ограниченные пропускной способностью диска (I/O) или памяти, мало выиграют от векторизации
- Часто нужны доработка кода и расположения данных, подсказки компилятору для векторизации

Часть 2. Инструкции / Векторные расширения инструкций

- Векторные расширения присутствуют в наборах команд процессоров различных производителей и архитектур
 - Intel: MMX – SSE – SSE2 – SSE3 – SSE4 – AVX – AVX-512
 - AMD: 3DNow!
 - ARM: NEON
- Далее рассмотрим некоторые векторные инструкции
- Узнаем, как соотносится расчетный код на C/C++ и машинный код, порождаемый компилятором

Часть 2. Инструкции / Основные типы операций

	Скалярная	Векторная
▪ <i>Арифметические операции</i> : сложение, вычитание, умножение, деление, FMA (Fused Multiply-Add) для вычислений с плавающей запятой	ADD	VADDPD
	DIV	VDIVPS
▪ <i>Операции преобразования типов</i> : повышающие и понижающие преобразования	CBW	CVTPS2PD
▪ <i>Логические операции</i> : векторные сравнения, поиск минимума и максимума и т.д.	CMP	PCMPEQW
	XOR	XORPD
▪ <i>Операции доступа к данным</i> (загрузка/выгрузка память/регистр; scatter/gather, предвыборка, streaming stores). Могут применяться маскирование, shuffle	MOV	MOVDQA

Часть 2. Инструкции / Регистры

Векторное расширение	Название регистров	Ширина регистра	Сколько float-чисел обрабатывает одновременно
SSE	xmm0 to xmm15	128 бит	4
AVX2	ymm0 to ymm15	256 бит	8
AVX512	zmm0 to zmm31	512 бит	16

vaddpd

add (mul, sub, div...) – тип операции

's'=scalar, 'p'=packed

's'=single precision, 'd'=double precision

Часть 3. Векторизация, её диагностика и улучшение

3.1. Включение авто-векторизации в C++, Fortran, C#, Python

3.2. Как писать код для векторизации. Примеры

3.3. Инструменты для диагностики векторизации

3.1. Включение векторизации / C++, Fortran

Доступно во всех современных компиляторах

- Ключи компиляции для векторизации на примере Intel® Compiler
 - `-O2`, `-O3` – высокие уровни оптимизации
 - `-fopenmp` – для переносимой векторизации через стандарт OpenMP 4.0 и выше
 - `-qvec-report=5` – для получения отчётов о векторизации
 - `-fp-model`, `-fimf-precision` – для настройки точности вычислений и математических функций
- Директивы для компилятора добавляют в код для ручной векторизации
 - `#pragma vector`
 - `#pragma omp simd`

3.1. Включение векторизации / C#

- Использовать .NET Core 3.0 и выше, где поддерживаются стандарты работы с SIMD
- Включить пакет System.Numerics
- Использовать типы данных Vector, Vector2, Vector3, Matrix3x2, Matrix4x4...
- Пример кода

```
using System.Numerics;

var lanes = Vector<int>.Count;
float[] arr = new float[1_200_000];
Array.Fill<float>( arr, 23.74f );
var v8sum = new Vector<float>();

for( int i = 0; i < arr.Length; i+=lanes )
{
    var v8temp = new Vector<float>( arr, i );
    v8sum += v8temp;
}
```

Материалы и пример кода из статьи
Adrian Jurczak
[“SIMD usage in C++, C# and Rust”](#)

3.1. Включение векторизации / Python

- Использовать оптимизированные библиотеки numpy, scipy, scikit-learn или полностью оптимизированную сборку Python, например Intel® Distribution for Python*
- Применять рекомендованные паттерны программирования:
 - избегать for-циклов в Python
 - использовать numpy arrays, broadcasting
- Для трудоёмких кодов задействовать библиотеки с native-частью:
 - Intel® DAAL, OpenCV
 - В ML-фреймворках проверять подключение оптимизированных backend-реализаций

3.2. Способы векторизации. Как писать код

Обсудим в этой лекции

1. Использовать высокопроизводительные библиотеки, эффективно включающие векторные инструкции
2. Написать программу на C/C++ или Fortran и использовать авто-векторизацию современных компиляторов
3. Добавлять специальные ключи и директивы компилятора (подсказки)
4. Применять особые типы данных, дружественные векторизации вроде Intel® SIMD Data Layout Templates
5. Использовать классы интринсиков для SIMD и функции-интринсики
6. Реализовать векторизацию на ассемблере

Больше контроль.
Сложнее реализация и поддержка

3.2. Как писать код. Векторизованные библиотеки

- Для математики в C++, Fortran: Intel® MKL, OpenBLAS, BLIS
- Другие оптимизированные библиотеки по типам операций для физики, медиа-кодов fftw, OpenCV
- Для точечного использования математических функций $\sin()$, $\log()$, $\exp()$ обращать внимание на LibM, SVML, VML
- *Сложности:*
 - Не все нужные алгоритмы реализованы в библиотеке
 - Реализация в библиотеке, не всегда оптимальна для конкретной задачи
 - Сложности интеграции, поддержки, миграции на другие платформы

3.2. Как писать код. Требования к векторизуемым циклам

- Фиксированное количество итераций
 - Количество итераций должно быть известно до начала цикла
 - Не следует выходить по условию (ранний выход из цикла `break`)
- Нежелательно вызывать функции
 - Допустимы короткие встраиваемые/`inline`-функции
 - Или математические функции с векторной реализацией `pow()`, `sqrt()`, `sin()`...
- Желательно иметь один путь прохождения цикла (`single control flow`)
 - Редкие условия `if()` поддерживают векторизацию, нужны проверки или вынос условий за цикл
 - Без `switch()`-конструкций
- Минимальные зависимости по данным
- Необходим однородный доступ к памяти, загрузка данных, лежащих последовательно либо с одинаковым шагом
- Не следует смешивать объекты разных типов данных в выражениях

3.2. Как писать код. Используем C/C++ или Fortran в сочетании с оптимизирующим компилятором

Ключи компиляции для векторизации:

```
icc -g -O3 -fopenmp -qvec-report=5 example.cpp
```

Директивы для ручной векторизации OpenMP

- `#pragma omp simd` – объявление simd-цикла и определение деталей способа векторизации
- `#pragma omp declare simd` – объявление simd-функции для использования внутри simd-циклов

Директивы для ручной векторизации в Intel® Compiler

- `#pragma ivdep` - “игнорировать” зависимости по данным для векторов
- `#pragma vector [always/aligned/nontemporal]` – требует векторизовать цикл в зависимости от поданного следующего аргумента
- `#pragma omp simd` – поддерживается стандарт OpenMP

3.2. Как писать код. Используем C/C++ или Fortran в сочетании с оптимизирующим компилятором

Пример 1. Запрос компилятору векторизовать цикл, игнорируя зависимости, и с выровненными инструкциями

```
#pragma simd
#pragma vector aligned
for (i=0; i<n; i++)
    a[i] = a[i] * c;
```

Пример 2. Сложный поток управления, авто-векторизация невозможна

```
for (int i=0; i<len; i++)
{
    if (x[i] == 42) break;
    y[i] = x[i];
}
```



```
int m=len;
for (int i=0; i<len; i++)
    if (x[i] == 42) {
        m = i;
        break;
    }
// Векторизация второго цикла теперь возможна
for (int i=0; i<m; i++)
    y[i] = x[i];
```

Пример 3:

```
// Сложный паттерн доступа к данным.
// Невыгоден для авто-векторизации
// и обычно требует изменения
// структуры данных для векторизации.

for (int i = 0; i < len; i++)
{
    b[a[i]] = x;
}
```

3.2. Как писать код. Пример векторизации на C++.

Вынос вычисления константы за пределы цикла

Не векторизуемый цикл

```
for (int i = 0; i < len; i++)  
{  
    y[i] = x[i] * exp(sqrt(len/2.));  
}
```

Векторизация возможна

```
double alpha = exp(sqrt(len/2.));  
for (int i = 0; i < len; i++)  
{  
    y[i] = x[i] * alpha;  
}
```

3.2. Как писать код. Пример зависимости по данным (loop-carried dependency)

Векторизация меняет порядок исполнения итераций цикла по сравнению с последовательным исполнением.

Компилятор векторизует, если может доказать, что векторизация будет корректной.

Программисту надо следить за независимостью итераций цикла, иначе при форсировании векторизации возможны «падения» программы и некорректные результаты

Прямая зависимость Write After Read (WAR) векторизуется

```
for (int i=0; i<len; i++)  
    a[i] = a[i+1] + b[i];
```

Обратная зависимость Read after write (RAW) **не векторизуется**

```
for (int i=1; i<N; i++)  
    a[i] = a[i-1] + b[i];
```

Полезны автоматические инструменты определения зависимостей, такие как [Intel® Advisor](#)

3.2. Как писать код. Пример оптимизации Memory Access Pattern

Внутренний цикл векторизуется, но шаблон доступа к данным Memory Access Pattern можно оптимизировать

```
for (int x = 0; x < SIZE; x++)
{
    // Print statements on x-axis prevent loop interchange
    cout << "some diagnostic prints" << x << "\n";
    for (int y = 0; y < SIZE; y++)
        results[y][x] = (tableA[y][x]/tableB[y][x]) - ((double)500.0*atan(tableB[y][x]));
}
```

Перестановка циклов и улучшение Memory Access Pattern, но с изменением логики программы

```
for (int y = 0; y < SIZE; y++)
{
    for (int x = 0; x < SIZE; x++)
    {
        results[y][x] = (tableA[y][x] / tableB[y][x]) - ((double) 500.0 * atan(tableB[y][x]));
    }
}
```

Vectorized Loops				Instruction Set Analysis
Vector ...	Efficiency	Gain ...	VL ..	Traits
				Divisions; FMA
AVX512	89%	3.56x	4	Divisions; FMA; Inserts; Scatters

Диагностика неоптимизированного цикла с помощью Intel® Advisor

Vectorized Loops				Instruction Set Analysis
Vector ...	Efficiency	Gain E...	VL (...)	Traits
AVX2	100%	4.85x	4	Divisions; FMA
				Divisions; FMA
AVX2	100%	4.85x	4	Divisions; FMA

Диагностика оптимизированного цикла с помощью Intel® Advisor

3.2. Векторизация. Интринсики и ассемблер

- Большой контроль над логикой исполнения
- Сложность разработки: нужно хорошо знать систему команд и особенности архитектуры, а также иметь опыт низкоуровневого программирования
- Трудности с переносимостью кода на другие программно-аппаратные платформы
- Ресурсы для изучения:
 - Intel® Intrinsics Guide
 - Improving performance with SIMD intrinsics in three use cases
 - Software optimization resources by Agner Fog

```
22  #if defined(USE_AVX512)
23
24      // Favor using AVX512 if available.
25      static float
26      L2SqrSIMD16Ext(const void *pVect1v, const void *pVect2v, const void *qty_ptr) {
27          float *pVect1 = (float *) pVect1v;
28          float *pVect2 = (float *) pVect2v;
29          size_t qty = *((size_t *) qty_ptr);
30          float PORTABLE_ALIGN64 TmpRes[16];
31          size_t qty16 = qty >> 4;
32
33          const float *pEnd1 = pVect1 + (qty16 << 4);
34
35          __m512 diff, v1, v2;
36          __m512 sum = _mm512_set1_ps(0);
37
38          while (pVect1 < pEnd1) {
39              v1 = _mm512_loadu_ps(pVect1);
40              pVect1 += 16;
41              v2 = _mm512_loadu_ps(pVect2);
42              pVect2 += 16;
43              diff = _mm512_sub_ps(v1, v2);
44              // sum = _mm512_fmadd_ps(diff, diff, sum);
45              sum = _mm512_add_ps(sum, _mm512_mul_ps(diff, diff));
46          }
47
48          _mm512_store_ps(TmpRes, sum);
49          float res = TmpRes[0] + TmpRes[1] + TmpRes[2] + TmpRes[3] + TmpRes[4] + TmpRes[5] + TmpRes[6] +
50                    TmpRes[7] + TmpRes[8] + TmpRes[9] + TmpRes[10] + TmpRes[11] + TmpRes[12] +
51                    TmpRes[13] + TmpRes[14] + TmpRes[15];
52
53          return (res);
54      }
```

Пример intrinsics: функция поиска расстояния между векторами из Hnswlib - fast approximate nearest neighbor search
<https://github.com/nmslib/hnswlib>

3.2. Векторизация. Пример векторизованного цикла и его ассемблерного кода

```
// Исходный цикл с вычислениями и без зависимостей
for (i = 1; i <= N; ++i__)
{
    result[++k-1] = a[i+j*t]+b[i+j*s]*c[i+j*p];
}
```

```
// Сгенерированный ассемблер с векторными инструкциями AVX2
0x418435          Block 1: 12500000
0x418435      943      vmovups ymm3, ymmword ptr [rdi+0x4]
0x41843a      942      add rdi, 0x20
0x41843e      944      vmovups ymm2, ymmword ptr [rbx+0x4]
0x418443      944      vfmadd213ps ymm3, ymm2, ymmword ptr [r12+0x4]
0x41844a      942      add r12, 0x20
0x41844e      943      vmovntps ymmword ptr [r14+r13*4+0x763c00], ymm3
0x418458      942      add r13, 0x8
0x41845c      942      add rbx, 0x20
0x418460      942      cmp r13, rsi
0x418463      942      jnb 0x418435 <Block 1>
```

3.3. Диагностика векторизации / Отчёты компилятора

- Получить отчёт компилятора о векторизации, добавив ключи компиляции
 - ICC ключи `-qvec-report=5` (число – уровень детализации отчёта)
 - GCC Ключи `-fopt-info-vec`
 - Microsoft* /Qvec-report
- Пример отчёта:

```
LOOP BEGIN at D:\work_eantakov\apps\some_example.f
  remark #15389: vectorization support: reference AA has unaligned access
  remark #15389: vectorization support: reference A has unaligned access
  remark #15381: vectorization support: unaligned access used inside loop
body
  remark #15399: vectorization support: unroll factor set to 2
remark #15300: LOOP WAS VECTORIZED
  remark #15450: unmasked unaligned unit stride loads: 2
```


3.3. Диагностика векторизации / Отчёты компилятора

Пример отчёта о не векторизованных циклах

```
some_example.optprt:
```

```
LOOP BEGIN at D:\work_eantakov\apps\LCD\loops90.f(146,7)
```

```
  remark #15382: vectorization support:
```

```
    call to function calculate() cannot be vectorized
```

```
  remark #15344:
```

```
    loop was not vectorized: vector dependence prevents vectorization
```

```
  remark #15346:
```

```
    vector dependence: assumed OUTPUT dependence between A line 146 and A  
line 147
```

```
  remark #15521:
```

```
    loop was not vectorized: explicitly compute the iteration count before  
executing the loop or try using canonical loop form
```

3.3. Диагностика векторизации / Инструмент анализа и рекомендаций Intel® Advisor

Фокусировка на «тяжёлых» циклах

The screenshot displays the Intel Advisor Roofline tool interface. At the top, it shows 'Elapsed time: 26.96s' and filters for 'Vectorized' and 'Not Vectorized' code. The main table lists various performance issues, including 'Vector register spilling possible' and 'Assumed dependency present'. A specific row is highlighted, showing a 'loop in s4' with a CPU time of 0.170s. Below the table, a detailed explanation is provided for the issue 'Loop with early exits cannot be vectorized unless it meets search loop idiom criteria'. The cause is explained as the compiler not recognizing a search idiom in a loop that may exit early. A C++ example is provided to illustrate an early exit scenario.

Function Call Site...	Performance Issues	CPU Time	Type	Why No Vectorization?	Vectorized Loops
		Total Time	Self Time		Vector ... Efficiency Gain E...
[loop in s24...	3 Vector register spilling possible	0.220s	0.220s	Vectori...	1 vector dependence prevents vecto... AVX 14% 1.09x
[loop in s21...	3 Assumed dependency present	0.220s	0.220s	Scalar ...	1 vector dependence prevents vecto...
[loop in s32...	2 Assumed dependency present	0.220s	0.220s	Scalar ...	1 vector dependence prevents vecto...
[loop in s35...	2 Possible inefficient memory acces...	0.210s	0.210s	Vectori...	1 vectorization possible but seems in... AVX512 14% 1.09x
[loop in s12...	2 Assumed dependency present	0.200s	0.200s	Scalar	vector dependence prevents vectori...
[loop in s34...	1 Possible inefficient memory acces...	0.180s	0.180s	Scalar	vectorization possible but seems ine...
[loop in s4...		0.170s	0.170s	Scalar	loop with multiple exits cannot b...
[loop in std::...	1 Assumed dependency present	0.170s	0.170s	Scalar	vector dependence prevents vectori...
s2111_omp...		1.399s	0.160s	Function	
[loop in s25...	2 Assumed dependency present	0.160s	0.160s	Scalar ...	1 vector dependence prevents vecto...

Loop with early exits cannot be vectorized unless it meets search loop idiom criteria

Cause: The compiler did not recognize a search idiom in a loop that may exit early. For example: the loop body contains:

- A conditional exit or GOTO statement followed by calculations
- A potential exception - the compiler considers an exception another possible exit (C++ only)

C++ Example: Early exit

```
void c15520(float a[], float b[], float c[], int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        if(a[i] < 0.) break;
        c[i] = sqrt(a[i]) * b[i];
    }
}
```

Детальные пояснения причин не векторизации

Рекомендации по улучшению производительности

3.3. Диагностика векторизации / Инструмент анализа и рекомендаций Intel® Advisor

Диагностика причин неэффективной векторизации

Подробности о типах данных и операциях

Vectorized

Not Vectorized

Filter: All Modules All Sources Loops And Functions All Threads

Customize View

Offline

Refinement Reports

Function	Vectorized	ISA	Efficiency	Gain...	VL...	Trip Counts	Average	Call Count	Instruction Set Analysis	Data Types	Number of ...	Type
[loop in s21...						90	1089900			Float32	3	Scalar
[loop in s12...	AVX2		100%	8.32x	8	125	100000		FMA; NT-stores	Float32	2	Vectorized (Body)
[loop in s35...						36	5500000		FMA	Float32	6	Scalar
[loop in s1...	AVX512		17%	1.32x	8	31; 7; 1	196800; 186200; ...		Inserts; Unpacks	Float32	2; 24; 4	Vectorized Versions
[loop in s23...						250	199600		FMA	Float32	3	Scalar
[loop in std::	AVX2		52%	4.15x	8	30; 4; ...	200000; 174800; 1...		FMA	Float32	2; 4	Vectorized (Body; Peeled; Remai
[loop in s44...	AVX512		21%	1.69x	8	22; 4	1100000; 1000000		FMA; Mask Manipulations	Float32; I...	13; 2	Vectorized (Body; Remainder)
[loop in s22...						44; 3	1100000; 1100000		FMA	Float32	15; 5	Scalar Versions

Source

Top Down

Code Analytics

Assembly

Recommendations

Why No Vectorization?

All Advisor-detectable issues: C++ | Fortran

! Possible inefficient memory access patterns present

Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

💡 Confirm inefficient memory access patterns

There is no confirmation inefficient memory access patterns are present. To fix: Run a [Memory Access Patterns analysis](#).

! Ineffective peeled/remainder loop(s) present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops the loop body.

💡 Add data padding

The trip count 505 is not a multiple of 8*2 (vector length * unroll factor). To fix: Do one of the following:

Ineffective peeled/remainder loop(s) present

Add data padding

Конкретные советы по улучшению данного типа кода

Динамический анализ паттернов доступа к данным и зависимостей

Зимняя Школа ННГУ по оптимизации алгоритмов компьютерного зрения - 2022

intel

27