

Введение в векторизацию кода

Приобретаемая компетенция

Применение знаний и умений по написанию
кода, *эффективно* задействующему
инструкции *современных* *процессоров*
архитектур IA-32 и AMD64.

Вопросик

Какие современные микроархитектуры сейчас применяются в процессорах?

Вопросик

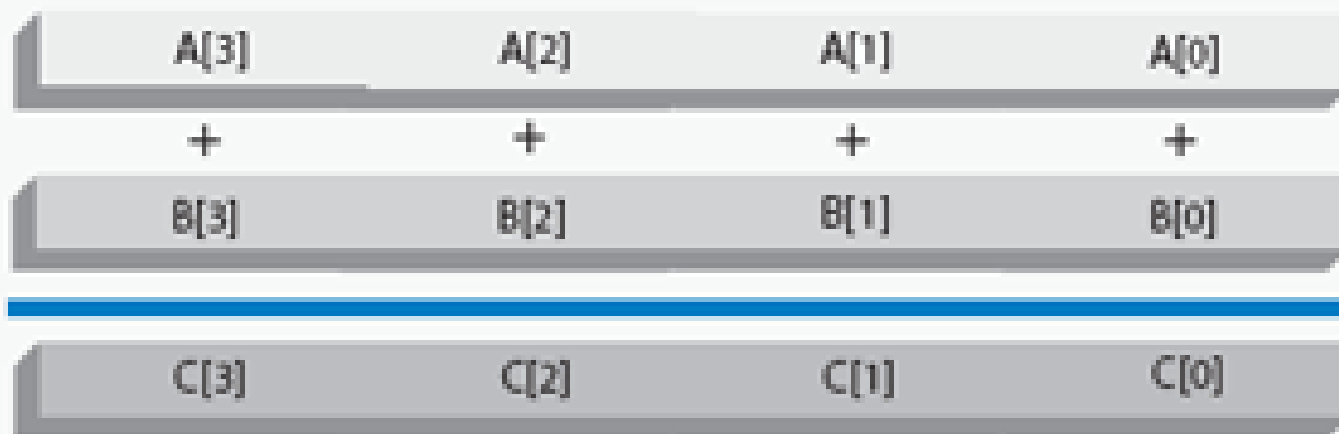
К какому типу архитектуры относятся современные микроархитектуры?

Пример векторизации

Задействуются все 4 арифметико-логические устройства процессора

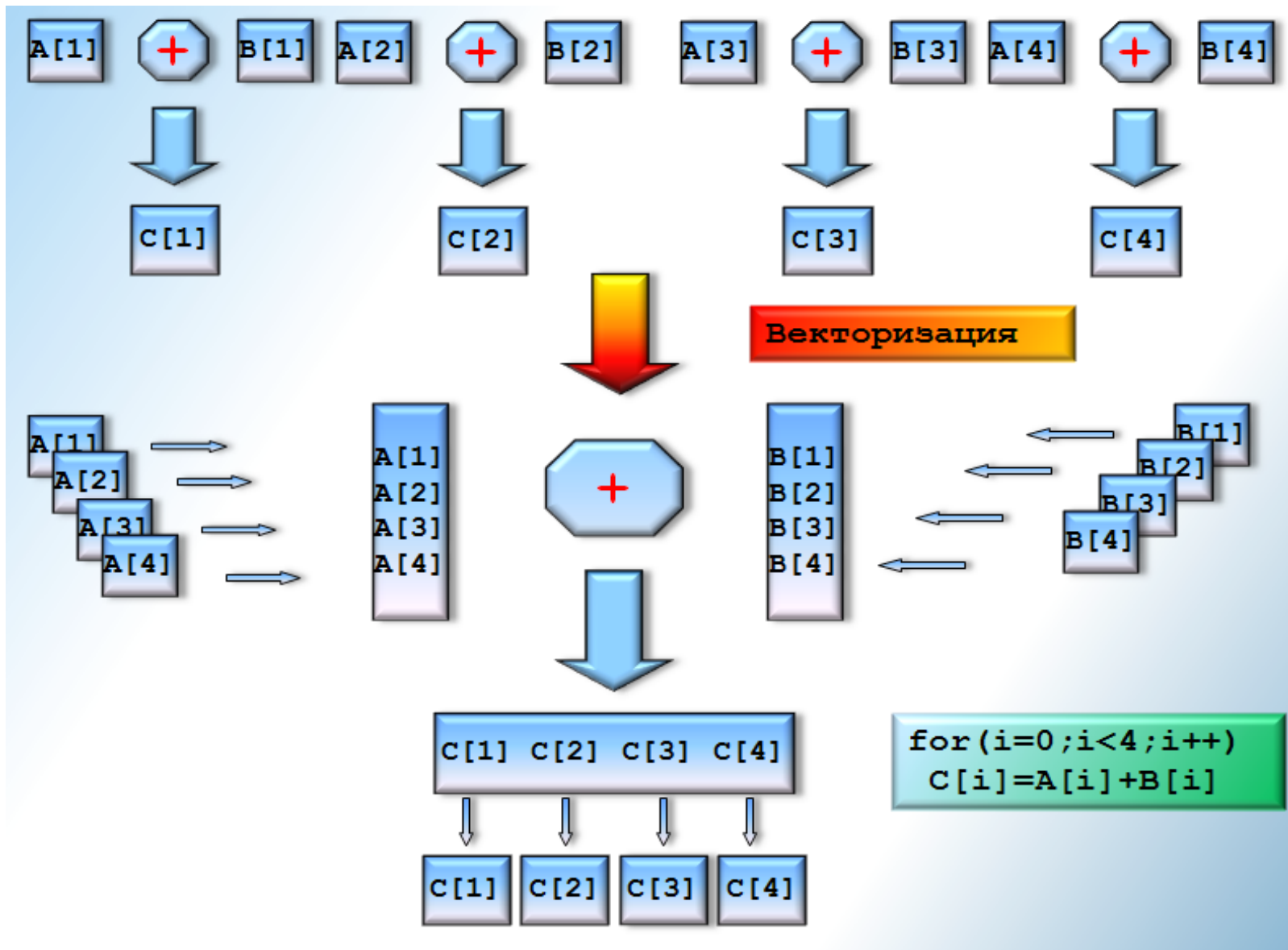
Vectorization Converts Loops

```
for (i=0; i<=MAX; i++)  
  c[i]=a[i]+b[i];
```

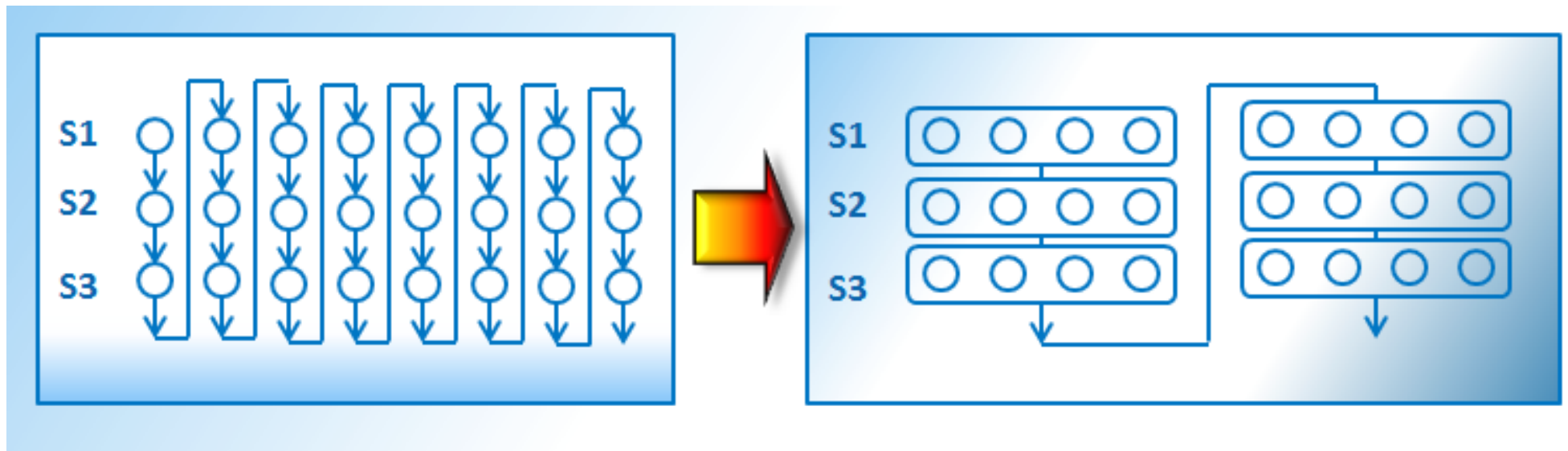


Сравнение векторного кода

Вместо 4-х операций по 2 операнда — одна с 8-мю операндами



Изменение порядка выполнения вычислений при векторизации



Чтение-после-записи

Это потоковая зависимость

$A[0] = 0;$

$\text{for } (j=1; j < \text{MAX}; j++) \quad A[j] = A[j-1] + 1;$

это эквивалентно:

$A[1] = A[0] + 1; \quad A[2] = A[1] + 1;$

$A[3] = A[2] + 1; \quad A[4] = A[3] + 1;$

Результат: $A[j] = 4$

Как можно устранить зависимость между итерациями?

Запись-после-чтения

Другой вид зависимости:

$A[0] = 0;$

for ($j=1; j < \text{MAX}; j++$) $A[j-1] = A[j] + 1;$

это эквивалентно:

$A[0] = A[1] + 1; A[1] = A[2] + 1;$

$A[2] = A[3] + 1; A[3] = A[4] + 1;$

Результат: $A[j-1] = 0$

Как можно устранить зависимость между итерациями?

Успешная векторизация

```
void Calculate(float * a, float * b, float * c , int n)
{
    for(int i=0; i<n; i++)
    {
        a[i] = a[i]+b[i]+c[i];
    }
    return;
}
```

```
gcc -c -O3 -march=native -std=c99 -mfpmath=sse -ftree-vectorizer-verbose=2 vector.c
```

Analyzing loop at vector.c:2

Vectorizing loop at vector.c:2

2: created 2 versioning for alias checks.

2: LOOP VECTORIZED.

vector.c:1: note: vectorized 1 loops in function.

Как векторизуется цикл?

Приблизительная схема векторизации цикла

```
for(i=0;i<100;i++)  
  p[i]=b[i]
```

0	p[0]=b[0]
1	p[1]=b[1]
2	p[2]=b[2]
3	p[3]=b[3]
...	
97	p[97]=b[97]
98	p[98]=b[98]
99	p[99]=b[99]



0	p[0]=b[0]
1	p[1]=b[1]
0	p[2:9]=b[2:9]
...	
11	p[90:97]=b[90:97]
0	p[98]=b[98]
1	p[99]=b[99]



Переход к
выравненному адресу



Векторная часть
Векторный регистр -
8 элементов



Хвостовой цикл

Вопросик

Как обеспечить гарантию отсутствия
перекрываний (overlapping) памяти?

Самостоятельная работа студента

Как эффективно проводить умножение матрицы на *современных* архитектурах?

Вопросик

Что требуется соблюдать для эффективной векторизации кода при его написании?

Тенденции

- Опыт векторных процессоров
- Размер расширенных регистров

MMX

- 64 бита
- * Ifpp (Itanium): 64 бита — 2 числа с плавающей запятой
- Поддерживающие процессоры:
 - семейство Intel Pentium 5 (MMX)

3DNow!

- 64 бита — 2 числа с плавающей запятой
- Поддерживающие процессоры:
 - семейства AMD K6-2, K6-3 (3DNow!)
 - National Semiconductor Geode (позже AMD Geode)
 - семейства VIA C3 (Cyrix III) "Samuel", "Samuel 2" "Ezra", "Eden ESP"
 - IDT Winchip 2

SSE

- 64 бита — 2 числа с плавающей запятой
- Поддерживающие процессоры:
 - семейства Pentium 6

SSE2

- 64 бита — 2 числа с плавающей запятой
- Поддерживающие процессоры:
 - Intel NetBurst-based CPUs (Pentium 4, Xeon, Celeron, Pentium D, Celeron D)
 - Intel Pentium M and Celeron M
 - Intel Atom
 - Transmeta Efficeon
 - VIA C7

SSE3

- 64 бита — 2 числа с плавающей запятой
- Поддерживающие процессоры:
 - AMD:
 - Athlon 64, 64 X2, 64 FX, II
 - Opteron, Sempron
 - Phenom, Phenom II
 - Turion 64, 64 X2, X2 Ultra, II X2 Mobile, II X2 Ultra
 - APU
 - FX Series
 - Intel:
 - Celeron D, Celeron
 - Pentium 4, D, Extreme Edition, Dual-Core
 - Core
 - Xeon
 - Atom
 - VIA/Centaur:
 - C7
 - Nano
 - Transmeta Efficeon TM88xx (NOT Model Numbers TM86xx)

SSE4

- 128 бит — 4 числа с плавающей запятой
- Поддерживающие процессоры:
 - Intel
 - Penryn, Nehalem, Silvermont
 - Haswell
 - AMD
 - Barcelona
 - Bulldozer
 - Bobcat
 - Jaguar
 - VIA
 - Nano

AVX

- 256 бит — 8 чисел с плавающей запятой
- Поддерживающие процессоры:
 - Intel
 - Sandy Bridge, E
 - Ivy Bridge, E
 - Haswell, E
 - Broadwell, E
 - AMD:
 - Bulldozer
 - Piledriver
 - Steamroller
 - Excavator
 - Jaguar
 - Puma

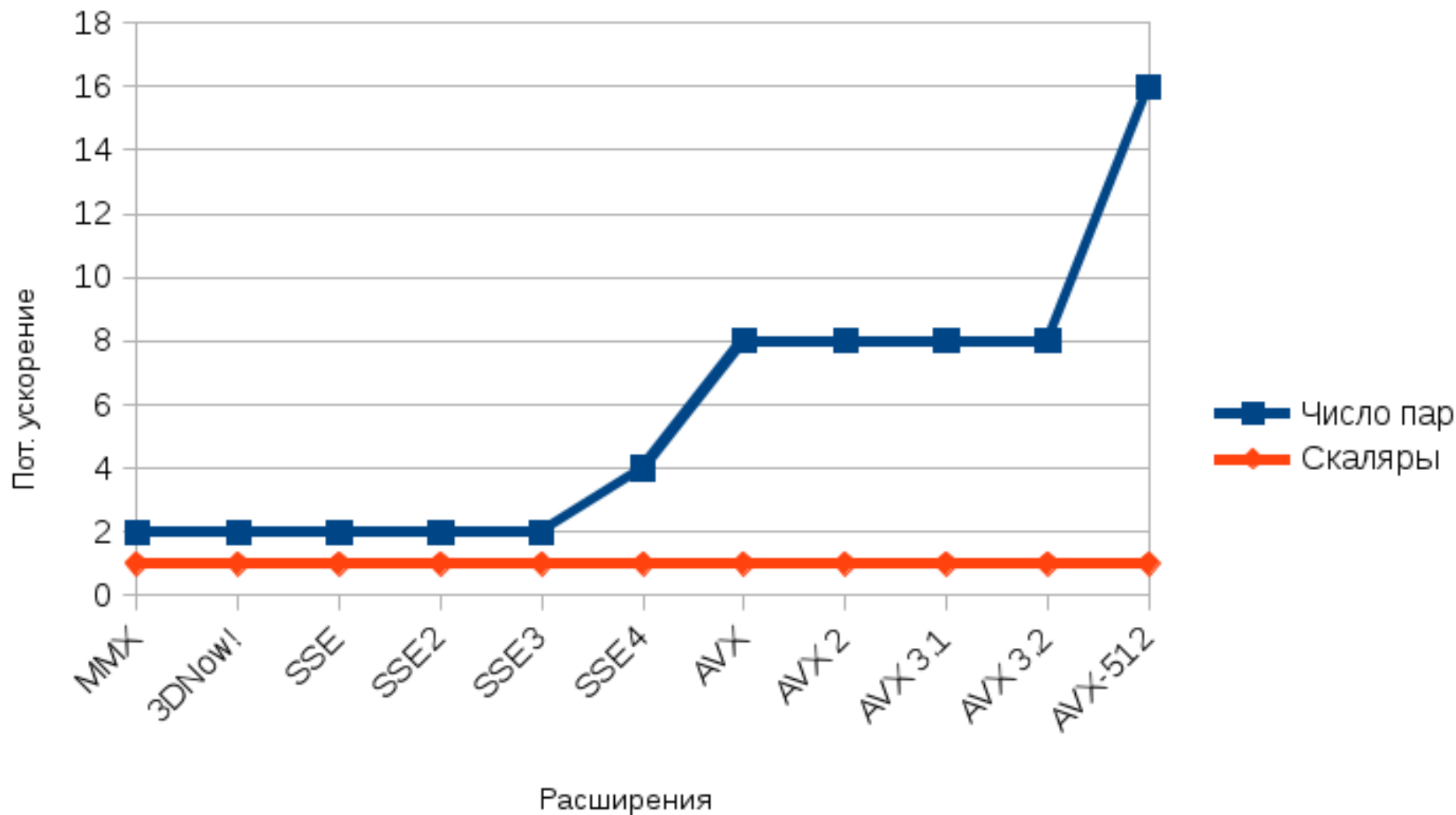
AVX2

- 256 бит — 8 чисел с плавающей запятой
- Поддерживающие процессоры:
 - Intel
 - Haswell (Q2 2013), E (Q3 2014)
 - Broadwell (Q4 2014), Broadwell E (2015)
 - Skylake (2015)
 - Cannonlake (2017)
 - AMD
 - Excavator (2015)

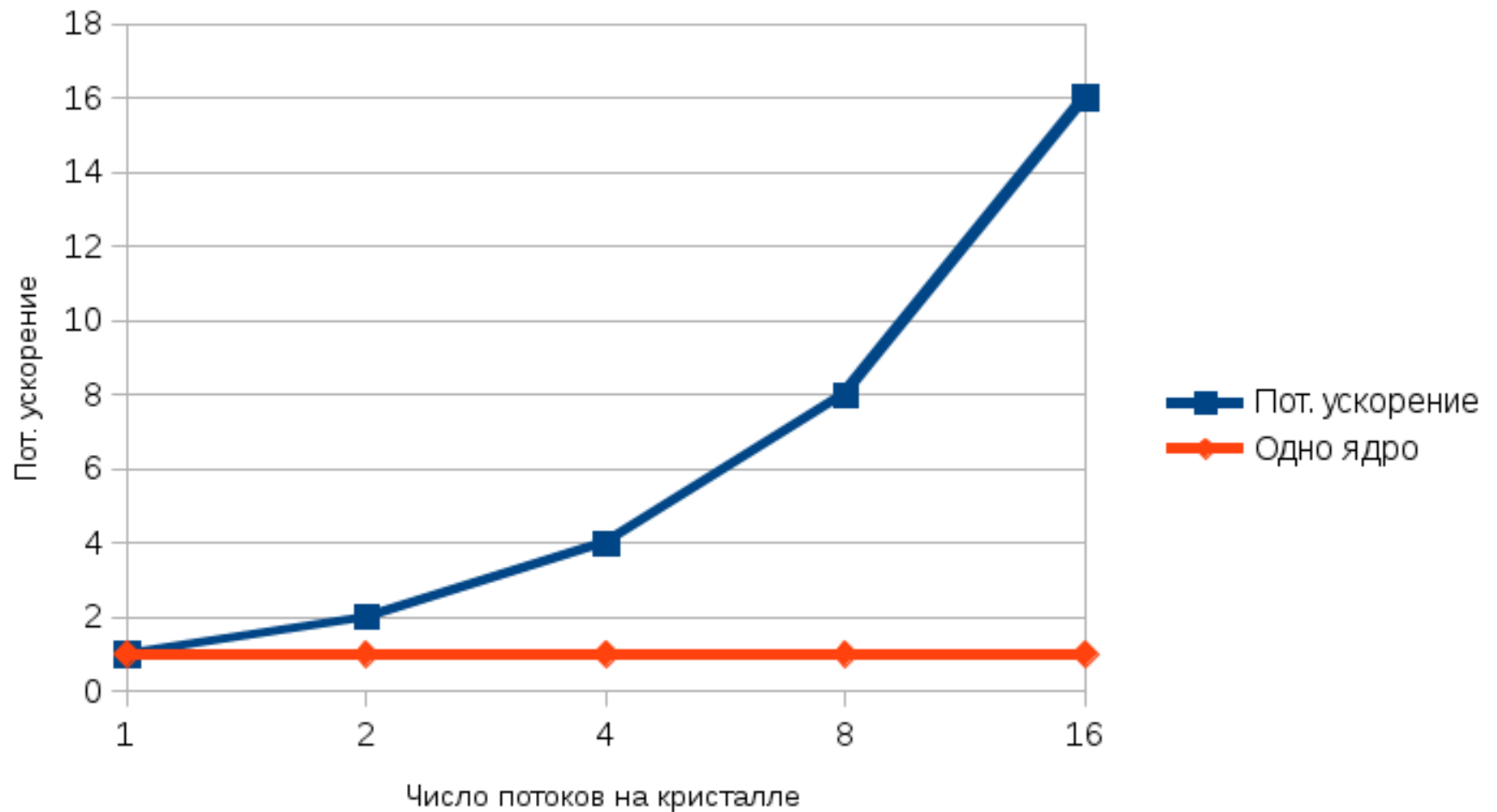
AVX-512

- 512 бит — 16 чисел с плавающей запятой
- Поддерживающие процессоры:
 - Intel
 - Knights Landing Xeon Phi (2015)
 - Skylake Xeon (2016)
 - Cannonlake Xeon (2017)

Ускорение за счёт векторизации



Ускорение за счёт многоядерности и многопоточности



Ускорение за счёт векторизации и многопоточности

