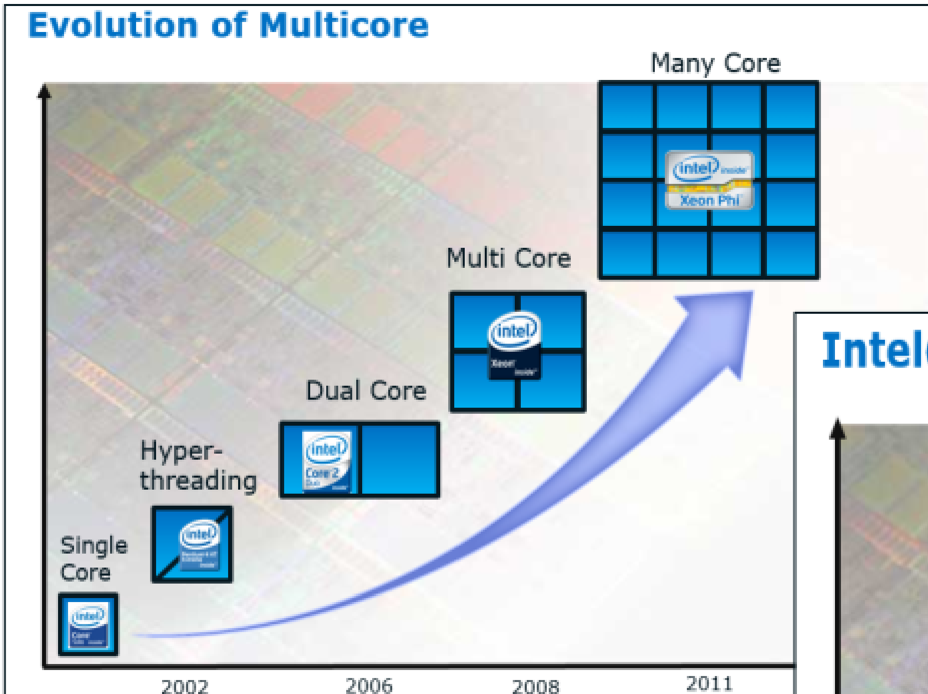
Вспомогательные материалы для преподавателя

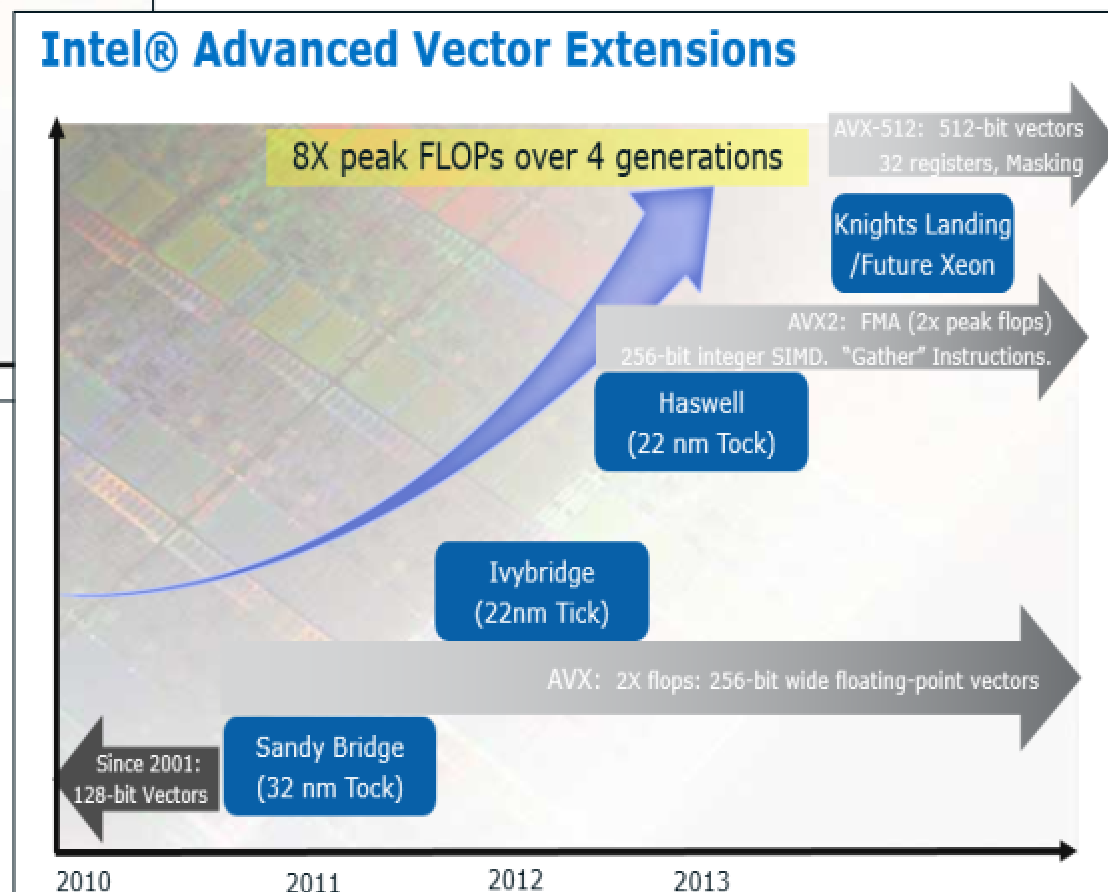
Особенности векторизации кода

Ст. преп. ВШ ПИ Фёдоров С. А.

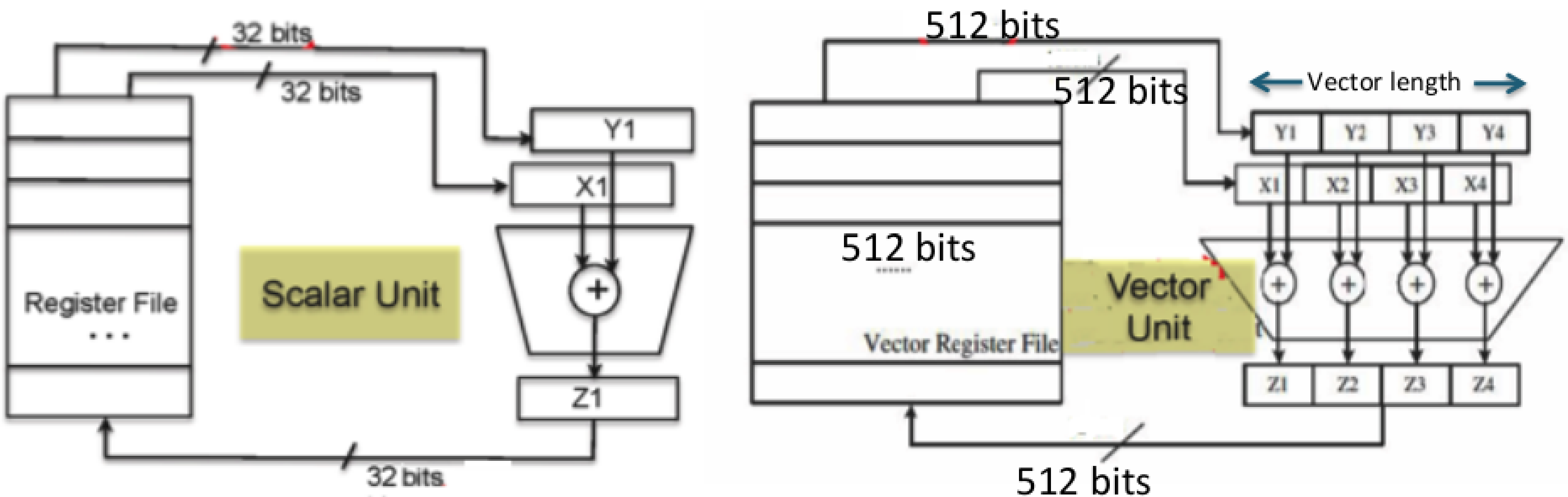
Эволюция архитектур



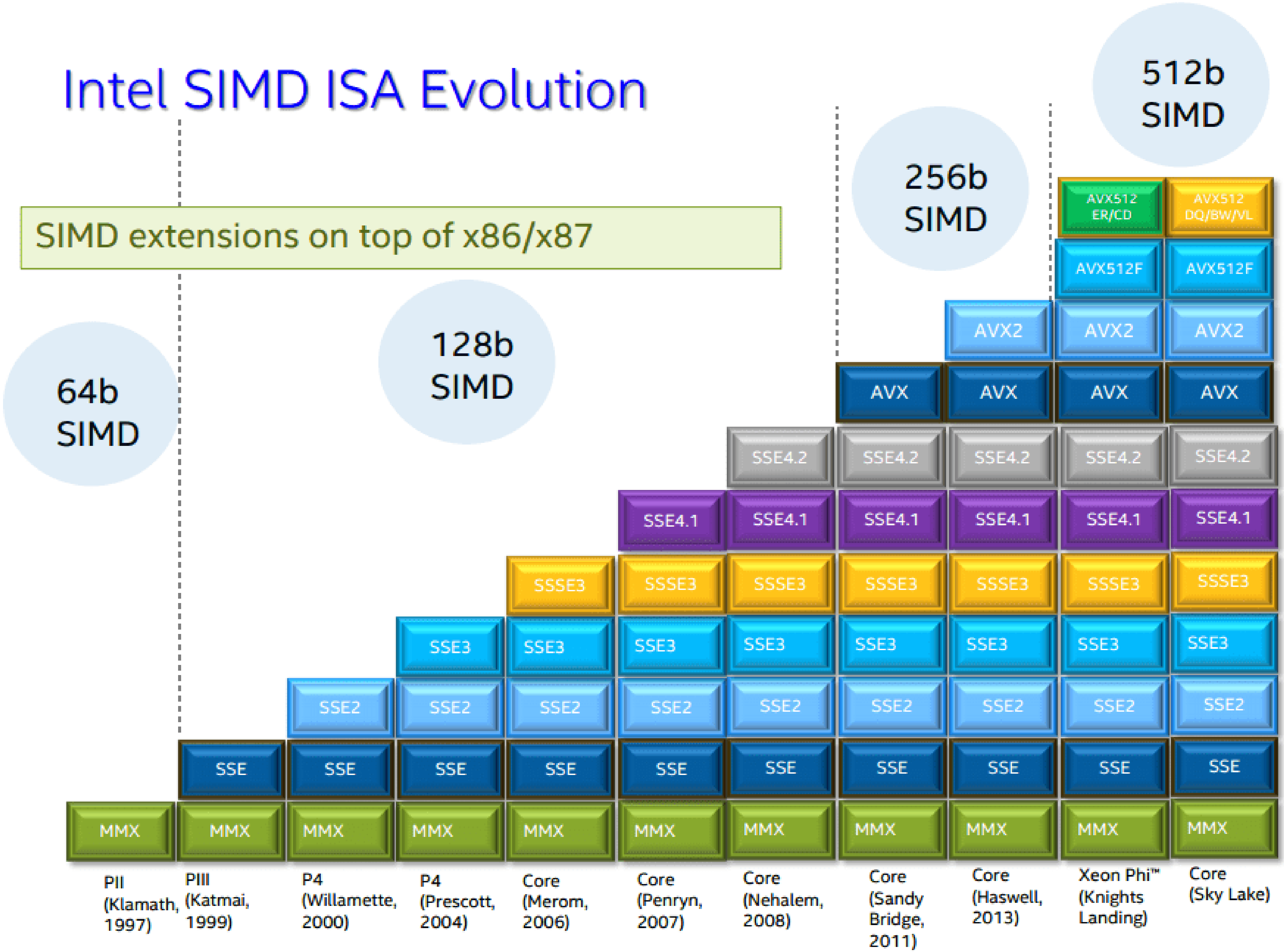
All future processors will have multiple cores with SIMD instructions



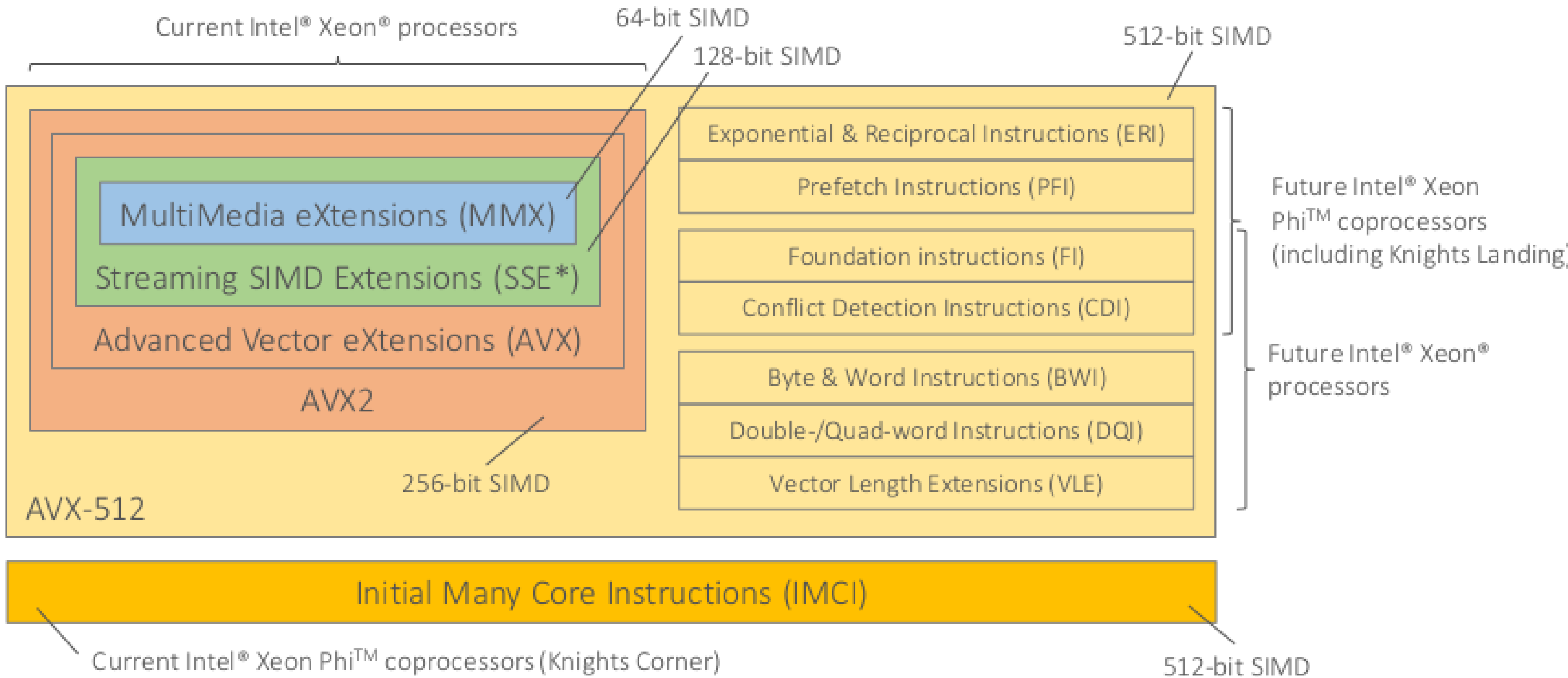
Архитектуры SIMD современных ядер



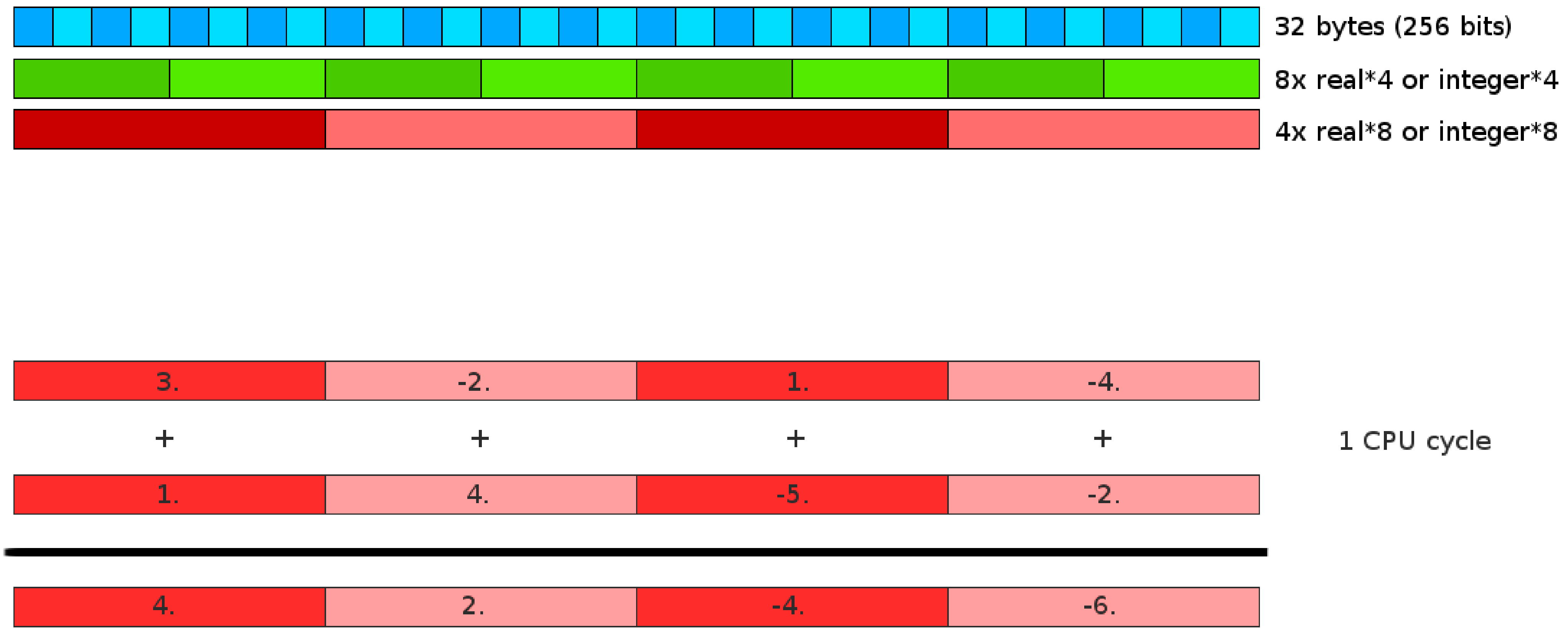
Intel SIMD ISA Evolution



Типы инструкций SIMD у решений Intel



Различия для разновидностей типа



Трансформация кода

```
DO I = 1, N
   Z(I) = X(I) + Y(I)
ENDDO
```

```
DO I = 1, N, 4
  Z(I)   =   X(I) +   Y(I)
  Z(I+1) = X(I+1) + Y(I+1)
  Z(I+2) = X(I+2) + Y(I+2)
  Z(I+3) = X(I+3) + Y(I+3)
ENDDO
```

```
VLOAD  X(I), X(I+1), X(I+2), X(I+3)
VLOAD  Y(I), Y(I+1), Y(I+2), Y(I+3)
VADD   Z(I, ..., I+3)   X+Y(I, ..., I+3)
VSTORE Z(I), Z(I+1), Z(I+2), Z(I+3)
```

Изменение порядка выполнения

```
DO I = 1, 4
  A(I) = B(I) + C(I)
  D(I) = E(I) + F(I)
END DO
```

Non-Vector

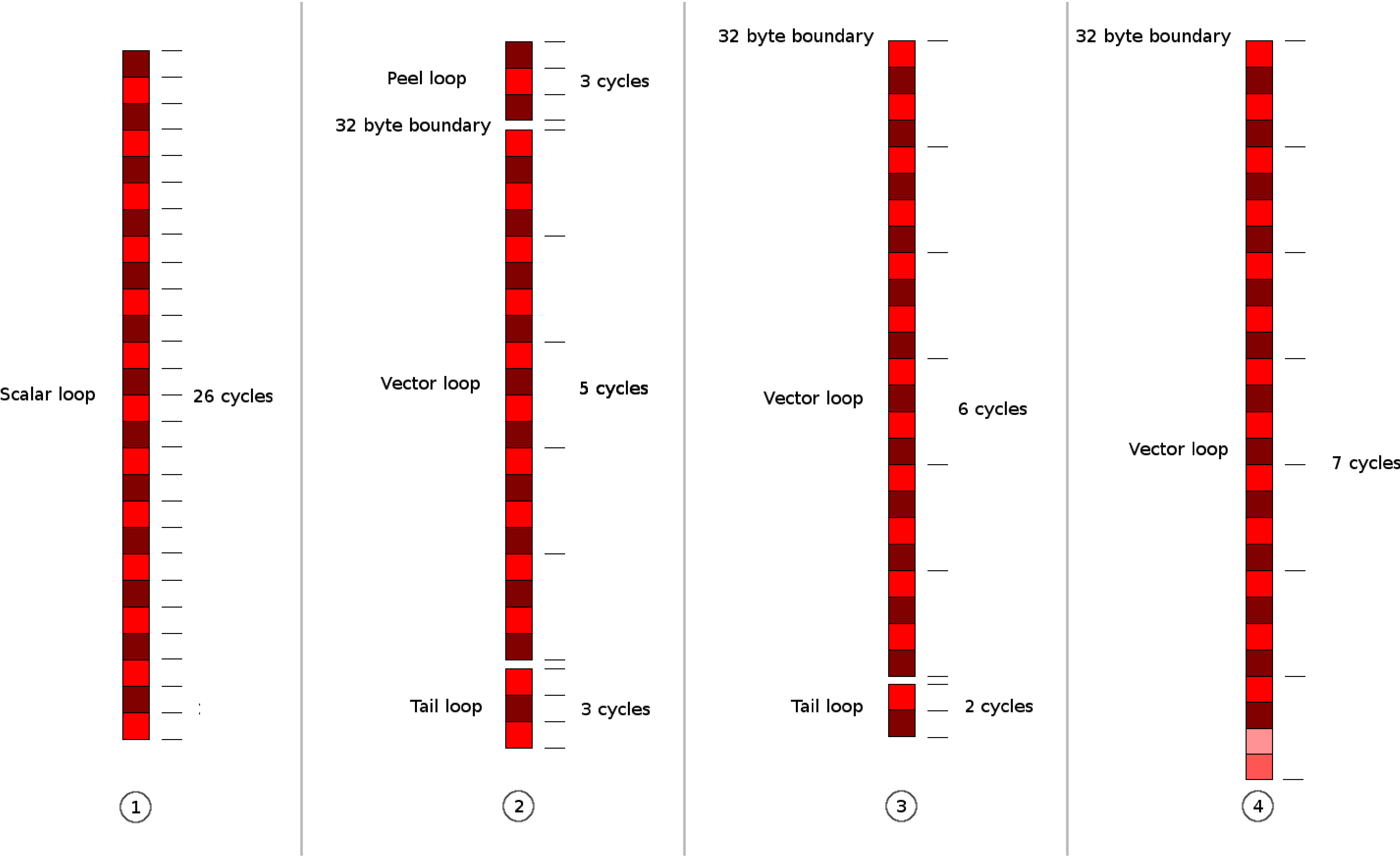
```
A(1) = B(1) + C(1)
 D(1) = E(1) + F(1)
A(2) = B(2) + C(2)
 D(2) = E(2) + F(2)
A(3) = B(3) + C(3)
 D(3) = E(3) + F(3)
A(4) = B(4) + C(4)
 D(4) = E(4) + F(4)
```

Vector

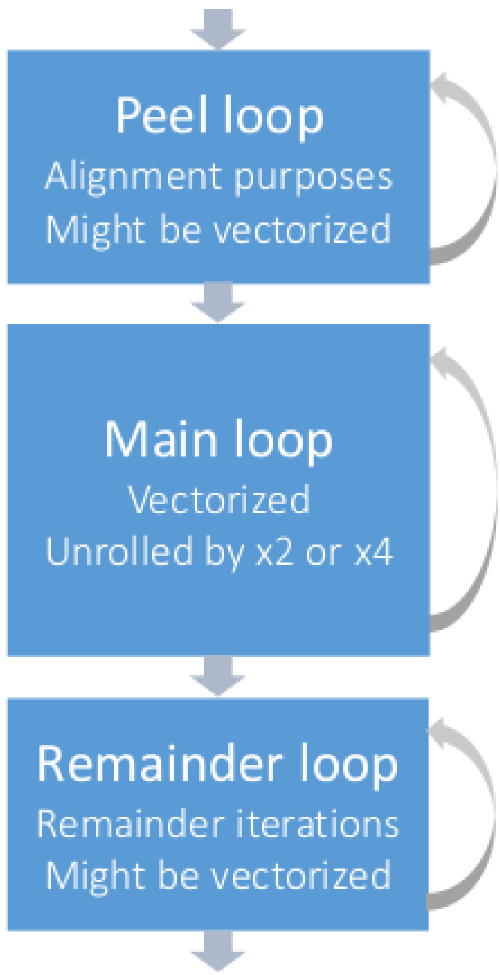
```
A(1) = B(1) + C(1)
A(2) = B(2) + C(2)
A(3) = B(3) + C(3)
A(4) = B(4) + C(4)
 D(1) = E(1) + F(1)
 D(2) = E(2) + F(2)
 D(3) = E(3) + F(3)
 D(4) = E(4) + F(4)
```

Order of execution

Три части порождаемого цикла



Три части порождаемого цикла




```
LOOP BEGIN at gas_dyn2.f90(2330,26)
<Peeled>
   remark #15389: vectorization support: reference AMAC1U has unaligned access
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15301: PEEL LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at gas_dyn2.f90(2330,26)
   remark #25084: Preprocess Loopnests: Moving Out Store
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15399: vectorization support: unroll factor set to 2
   remark #15300: LOOP WAS VECTORIZED
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 8
   remark #15477: vector loop cost: 0.620
   remark #15478: estimated potential speedup: 15.890
   remark #15479: lightweight vector operations: 5
   remark #15488: --- end vector loop cost summary ---
   remark #25018: Total number of lines prefetched=4
   remark #25019: Number of spatial prefetches=4, dist=8
   remark #25021: Number of initial-value prefetches=6
LOOP END

LOOP BEGIN at gas_dyn2.f90(2330,26)
<Remainder>
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```

Условия векторизации

- Данные должны быть сплошными в памяти
- Первый элемент каждого вектора должен быть выровнен
- Не должно быть перекрытий по памяти
- Не должно быть зависимостей итераций в виде чтение-после-записи

На что обращать внимание в коде

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

```
for (int i = 0; i < N; i++)
    a[c[i]] = b[d[i]];
```

```
for (int i = 0; i < N; i++)
    a[i] = foo(b[i]);
```

Для C99 и C++

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

```
void v_add(float *c, float *a, float *b)
{
#pragma ivdep
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

Управляемая векторизация: циклы

```
void v_add(float *c, float *a, float *b)
{
#pragma simd
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

Управляемая векторизация: OpenMP

```
#pragma omp simd reduction(+:sum) aligned(a : 64)
for(i = 0; i < num; i++) {
 a[i] = b[i] * c[i];
 sum = sum + a[i];
}
```

Управляемая векторизация: функции

```
__declspec(vector)
void v_add(float c, float a, float b)
{
     c = a + b;
}
     …
for (int i = 0; i < N; i++)
     v_add(C[i], A[i], B[i]);
```

Управляемая векторизация: функции (Fortran)

```
module fofx
contains
  function f(x)
!dir$ attributes vector :: f
    real, intent(in) :: x
    real f
    f = cos(x * x + 1.) / (x * x + 1.)
  end function
end module

program main
  use fofx
  real a(100), x(100)
  ...
  do i=1,100
    a(i) = f(x(i))
  end do
  ...
end program
```

Line 7

Elemental function in vectorization sense

```
$ ifort –vec-report=3 elemental.f90
...
elemental.f90(67): (col. 11) remark: LOOP WAS
VECTORIZED
...
elemental.f90(7): (col. 18) remark: FUNCTION WAS
VECTORIZED
elemental.f90(7): (col. 18) remark: FUNCTION WAS
VECTORIZED
```

Line 67

Управляемая векторизация: функции (Fortran)

```
module fofx
contains
  elemental function f(x)
!dir$ attributes vector :: f
    real, intent(in) :: x
    real f
    f = cos(x * x + 1.) / (x * x + 1.)
  end function
end module

program main
  use fofx
  real a(100), x(100)
  ...
  a = f(x)
  ...
end program
```

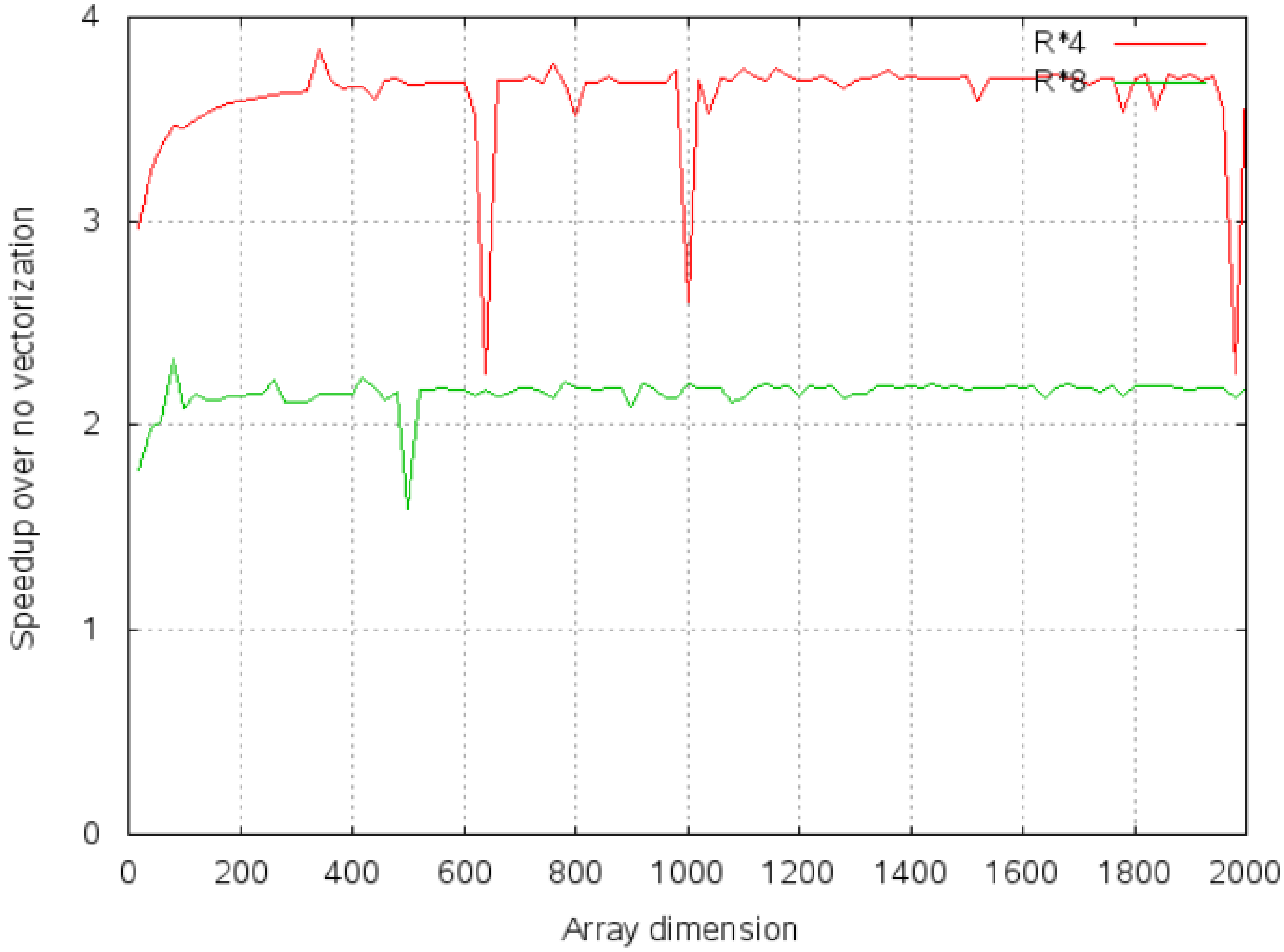
This Fortran 'elemental' clause nothing to do with vect.

Elemental function in vectorization sense

Line 67

```
$ ifort –vec-report=3 elemental.f90
...
elemental.f90(67): (col. 11) remark: LOOP WAS
VECTORIZED
...
elemental.f90(7): (col. 28) remark: FUNCTION WAS
VECTORIZED
elemental.f90(7): (col. 28) remark: FUNCTION WAS
VECTORIZED
```

Прирост производительности для элементных функций



Управляемая векторизация: OpenMP

```
#pragma omp parallel for simd
for(i = 0; i < num; i++) {
 sum = sum + a[i];
}
```

Управляемая векторизация: OpenMP

```
#pragma omp declare simd
float myfunction(float a, float b, float c )
{
 return a * b + c ;
}
```

```
#pragma omp simd
for(i = 0; i < num; i++) {
 OUT[i] = myfunction(arraya[i], arrayb[i], arrayc[i]);
}
```

Явная векторизация

```
a[:]        // All elements
a[2:6]      // Elements 2 to 7
a[:][5]     // Column 5
a[0:3:2]    // Elements 0,2,4
```

```
__declspec(vector)
void v_add(float c, float a, float b)
{
    c = a + b;
}
    …
v_add(C[:], A[:], B[:]);
```

Массивы структур и структуры массивов

```
// Array of Structures (AoS)
struct coordinate {
     float x, y, z;
} crd[N];
     …
for (int i = 0; i < N; i++)
     … = … f(crd[i].x, crd[i],y, crd[i].z);
```

Consecutive elements in memory →

x0	y0	z0	x1	y1	z1	…	x(n-1)	y(n-1)	z(n-1)

```
// Structure of Arrays (SoA)
struct coordinate {
     float x[N], y[N], z[N];
} crd;
     …
for (int i = 0; i < N; i++)
     … = … f(crd.x[i], crd.y[i], crd.z[i]);
```

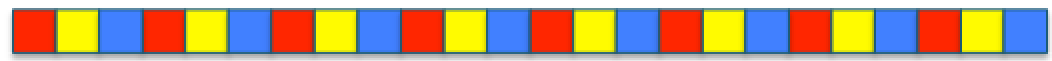
Consecutive elements in memory →

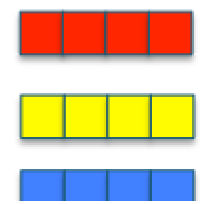
x0	x1	…	x(n-1)	y0	y1	…	y(n-1)	z0	z1	…	z(n-1)

Массив структур

```
type coords
  real :: x, y, z
end type
type (coords) :: p(100)
real dsquared(100)

do i=1,100
  dsquared(i) = p(i)%x**2 + p(i)%y**2 + p(i)%z**2
end do
```

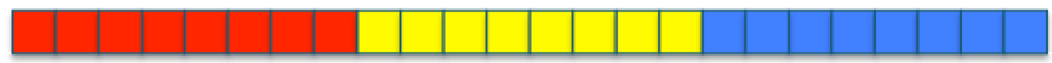


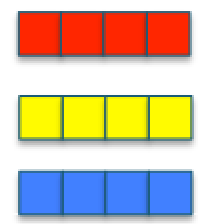


Структура массивов

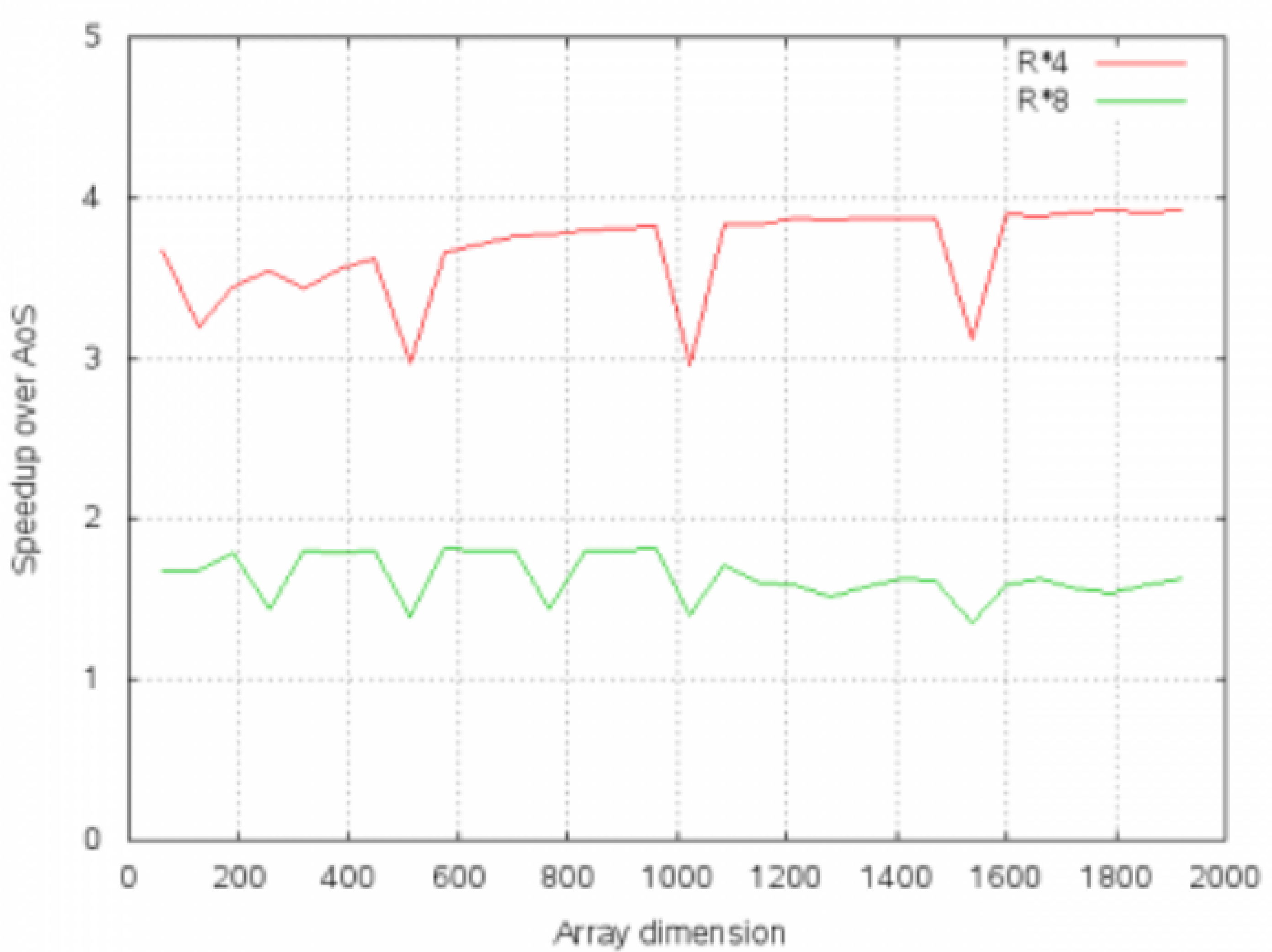
```
type coords
  real :: x(100), y(100), z(100)
end type
type (coords) :: p
real dsquared(100)

do i=1,100
  dsquared(i) = p%x(i)**2 + p%y(i)**2 + p%z(i)**2
end do
```





Прирост производительности для структуры массивов



Используйте средства компиляторов

```
edison11 h/hjw> cc -vec-report=6 -c mm.c
mm.c(7): (col. 2) remark: loop was not vectorized: loop was transformed to memset or memcpy
mm.c(10): (col. 2) remark: vectorization support: reference c has aligned access
mm.c(10): (col. 2) remark: vectorization support: reference c has aligned access
mm.c(10): (col. 2) remark: vectorization support: reference a has aligned access
mm.c(6): (col. 2) remark: vectorization support: unroll factor set to 4
mm.c(6): (col. 2) remark: PERMUTED LOOP WAS VECTORIZED
mm.c(9): (col. 2) remark: loop was not vectorized: not inner loop
mm.c(7): (col. 2) remark: loop was not vectorized: not inner loop
```

Используйте средства компиляторов: Cray

```
57.  + 1-----------<        do it=1,itmax
58.  + 1 br4--------<          do j=1,n
59.  + 1 br4 b------<             do k=1,n
60.    1 br4 b Vr2--<                do i=1,nr
61.    1 br4 b Vr2                      c(i,j) = c(i,j) + a(i,k) * b(k,j)
62.    1 br4 b Vr2-->                end do
63.    1 br4 b------>             end do
64.    1 br4-------->          end do
65.    1------------>       end do
```

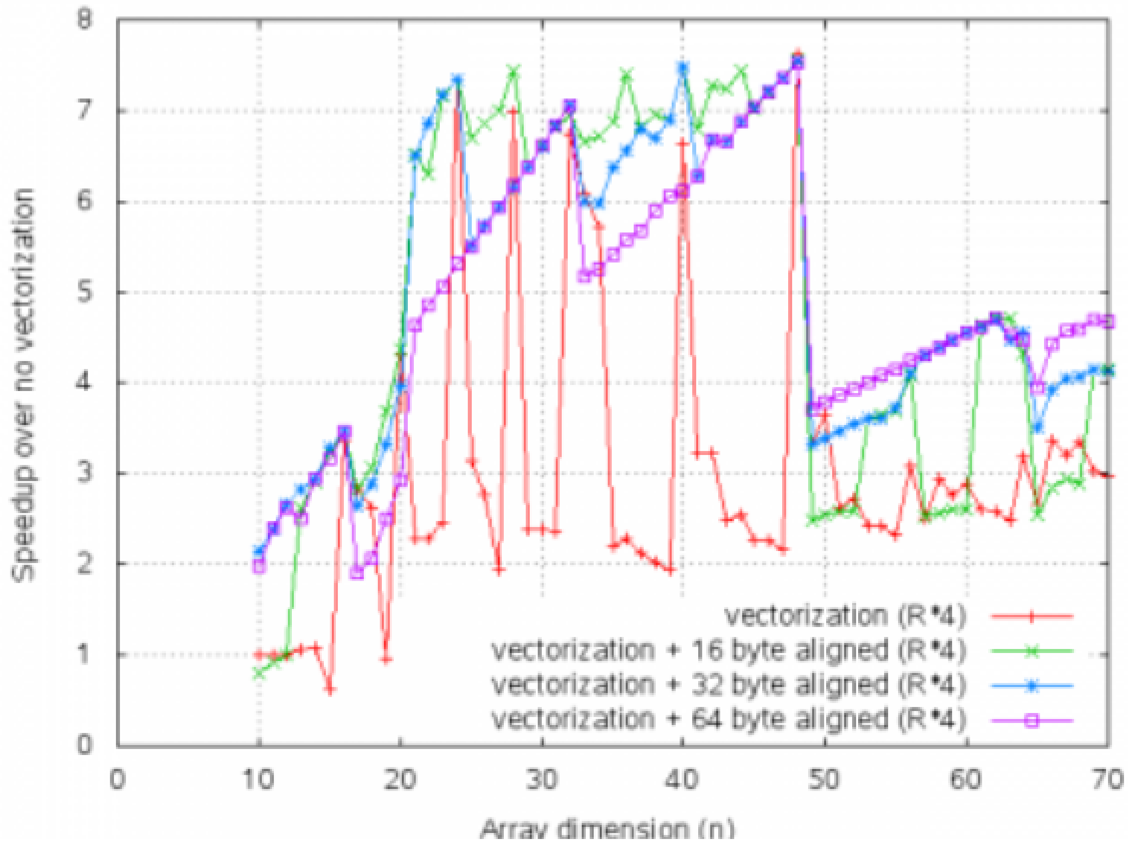
```
b - blocked
r - unrolled
V - Vectorized
```

```
ftn-6254 ftn: VECTOR File = matmat.F, Line = 57
  A loop starting at line 57 was not vectorized because a recurrence was found on "c" at line 61
ftn-6294 ftn: VECTOR File = matmat.F, Line = 58
  A loop starting at line 58 was not vectorized because a better candidate was found at line 60.
ftn-6049 ftn: SCALAR File = matmat.F, Line = 58
  A loop starting at line 58 was blocked with block size 8.
ftn-6005 ftn: SCALAR File = matmat.F, Line = 58
  A loop starting at line 58 was unrolled 4 times.
```

«Наивное» выравнивание данных

```
real, allocatable :: a(:,:), b(:,:), c(:,:)
!dir$ attributes align : 32 :: a,b,c
...
allocate (a(npadded,n))
allocate (b(npadded,n))
allocate (c(npadded,n))
...
do j=1,n
   do k=1,n
!dir$ vector aligned
      do i=1,npadded
         c(i,j) = c(i,j) &
            + a(i,k) * b(k,j)
      end do
   end do
end do

!... Ignore c(n+1:npadded,:)
```



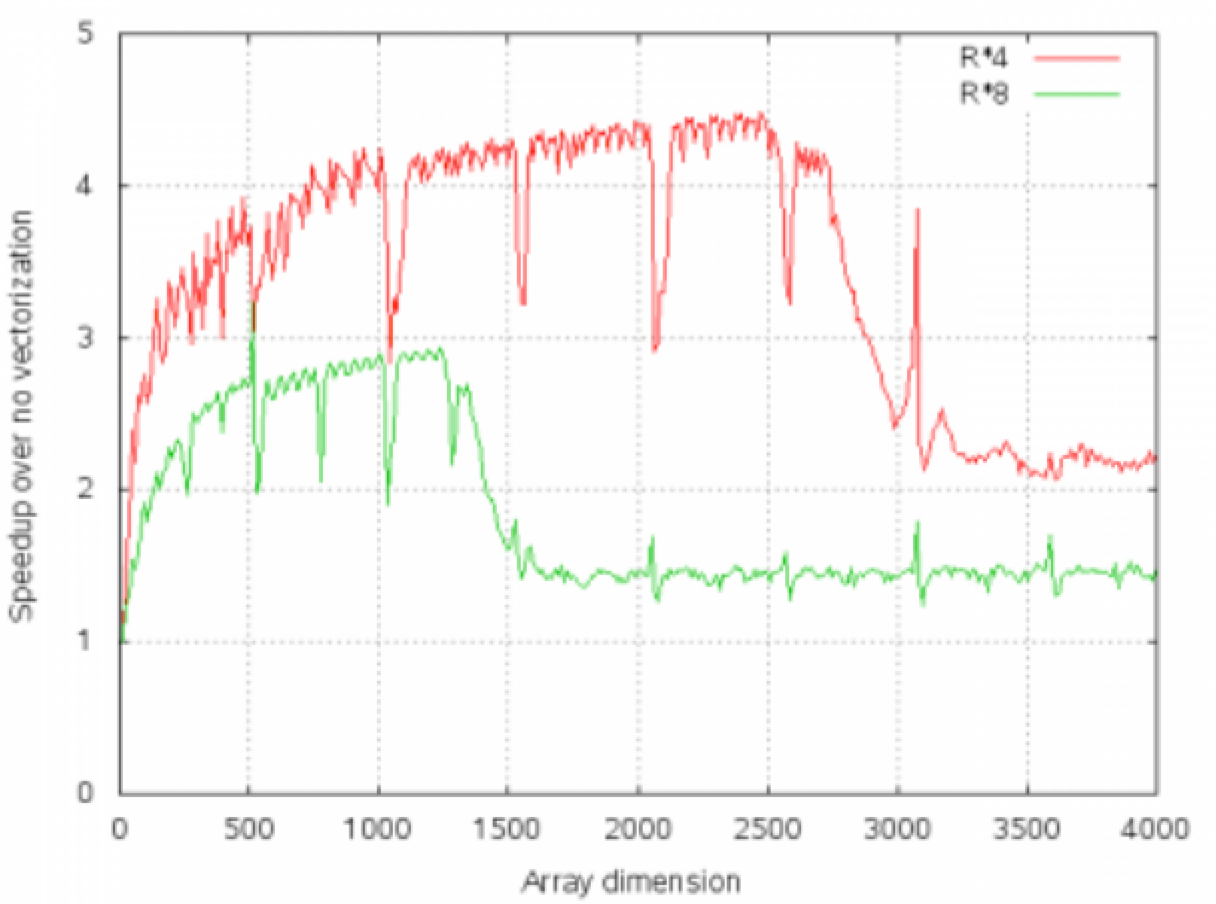
Выравнивание каждой строки

```
% cat Driver.c
...
#define COLBUF 1
...
#define COLWIDTH COL+COLBUF
...
        FTYPE a[ROW][COLWIDTH]  __attribute__((aligned(16)));
        FTYPE b[ROW]                    __attribute__((aligned(16)));
        FTYPE x[COLWIDTH]               __attribute__((aligned(16)));
...

% cat Multiply.c
...
#pragma vector aligned
...
                for (j = 0;j < size2; j++) {
                        b[i] += a[i][j] * x[j];
                }
```

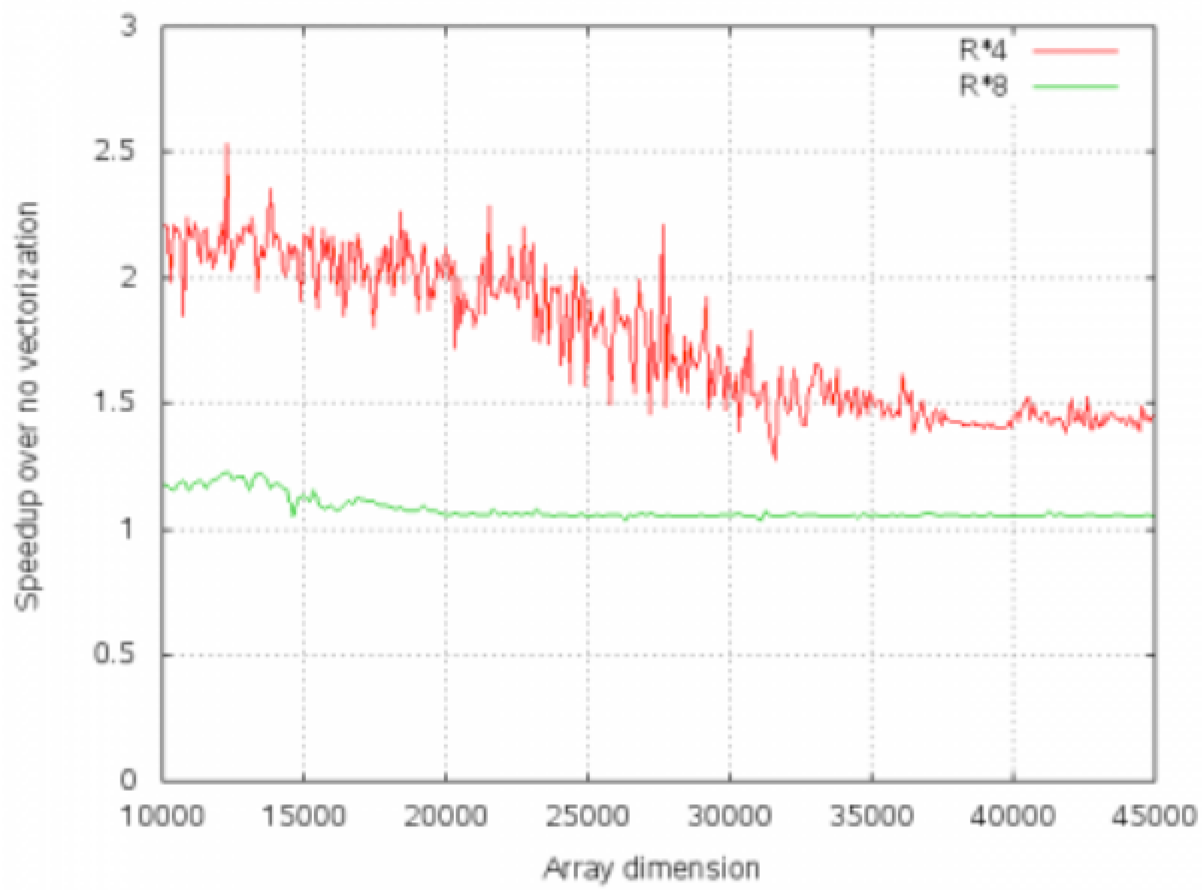
Что ожидать от производительности?

```
do i=1,n
    c(i) = a(i) + b(i)
end do
```



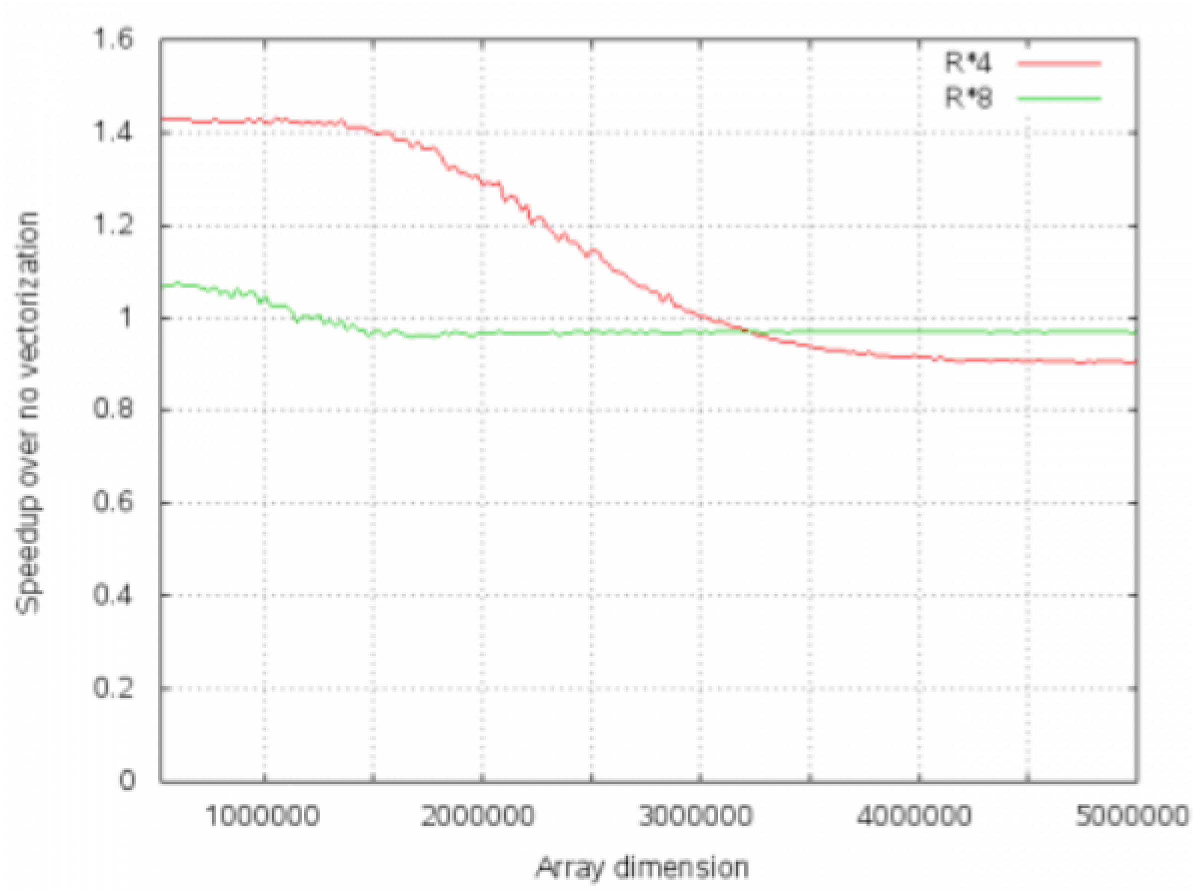
А дальше?

```
do i=1,n
    c(i) = a(i) + b(i)
end do
```



?

```
do i=1,n
    c(i) = a(i) + b(i)
end do
```



Блочная работа с СОЗУ

```
do i = 1, n
    do j = 1, m
        c += a(i) * b(j)
    enddo
enddo
```

Loads From DRAM:
n*m + n

```
do jout = 1, m, block
    do i = 1, n
        do j = jout, jout+block
            c += a(i) * b(j)
        enddo
    enddo
enddo
```

Loads From DRAM:
m/block * (n+block)
= **m*n/block** + m