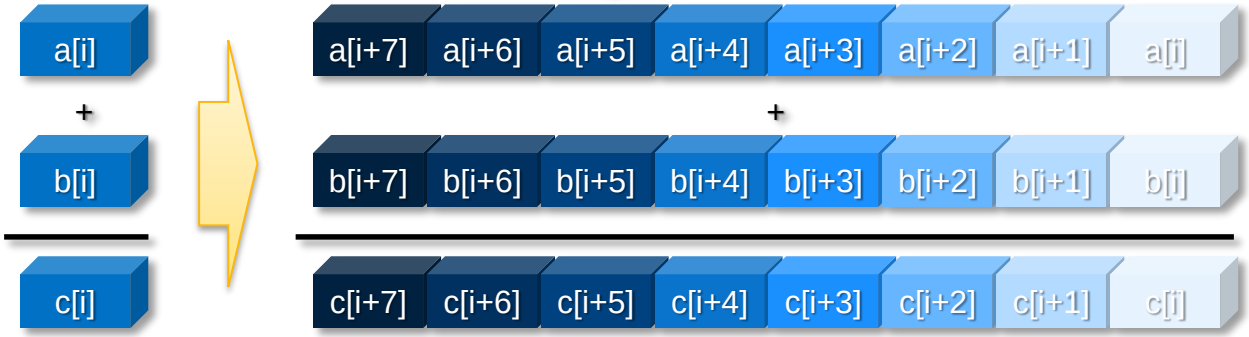
Векторизация и Roofline анализ приложений с помощью Intel® Advisor XE 2017

Дмитрий Петунин

intel Software

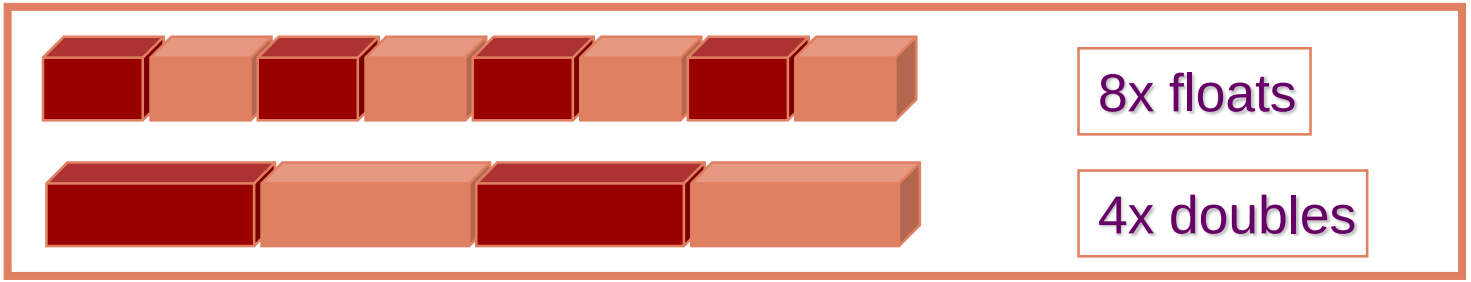
Векторные операции (SIMD)

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

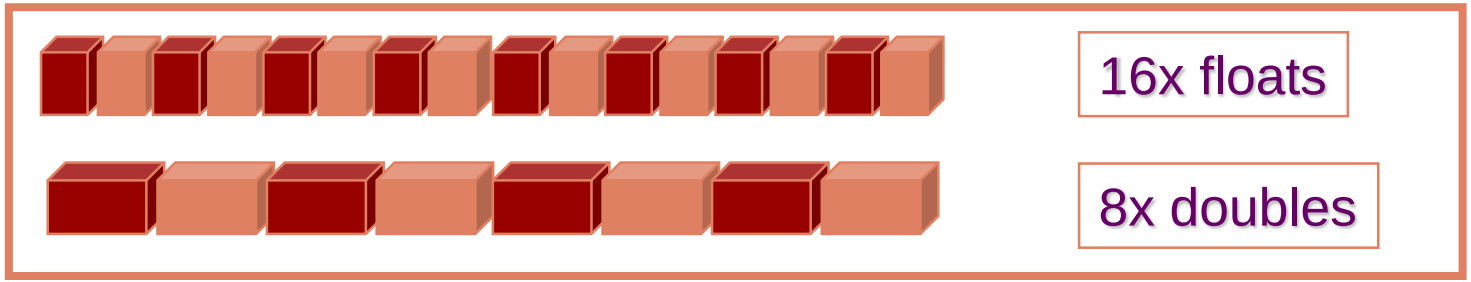


VECTOR REGISTERS IN INTEL ARCHITECTURES

AVX2
16 vector
registers



AVX512
32 vector
registers



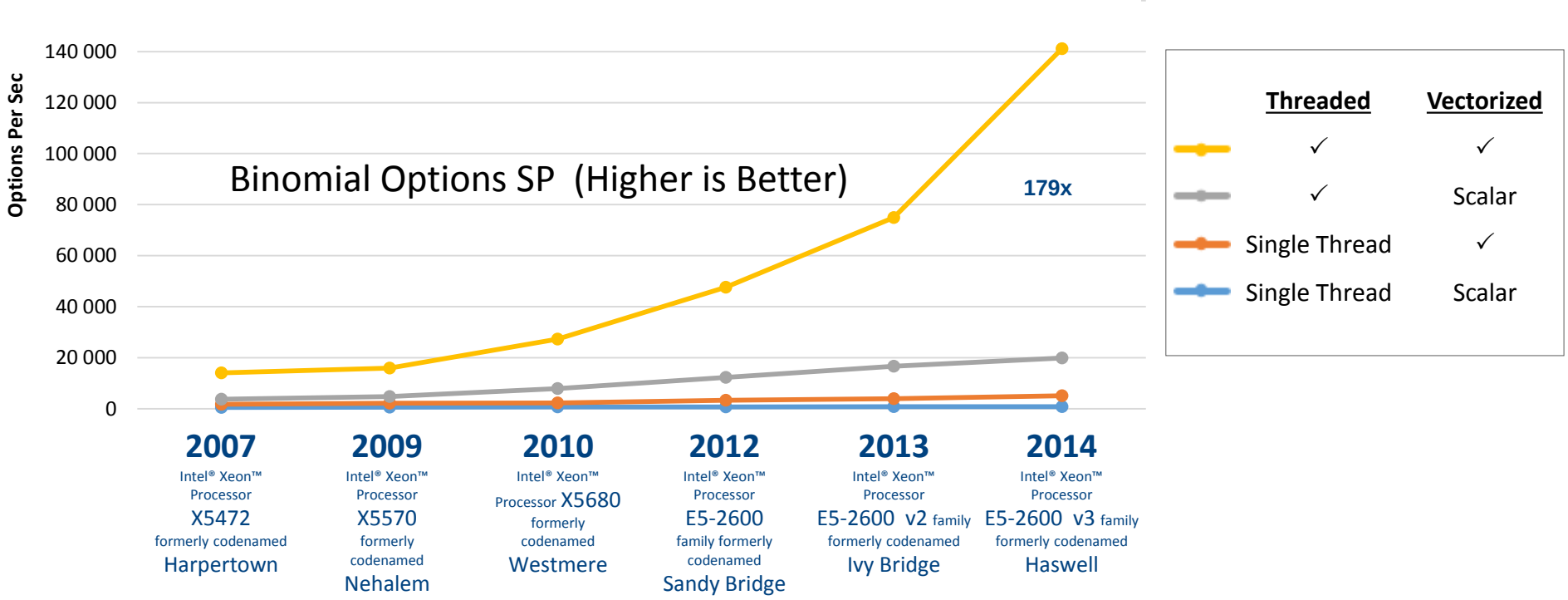
- More and wider vector registers
- Masked vector instructions
- High-accuracy approximate reciprocal instructions

Без векторизации

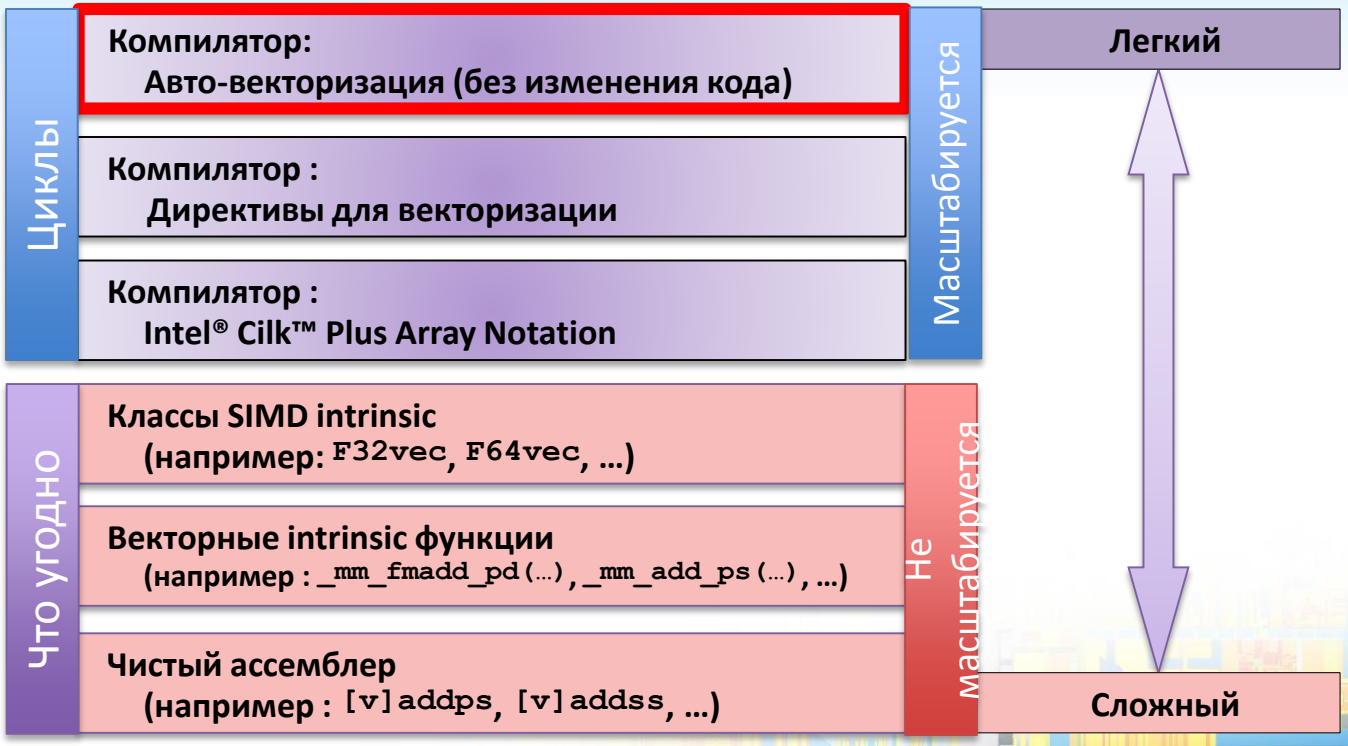
Используйте весь параллелизм

Intel Advisor XE:	Threading	Vectorization
	✓	✓

Каков потенциал?

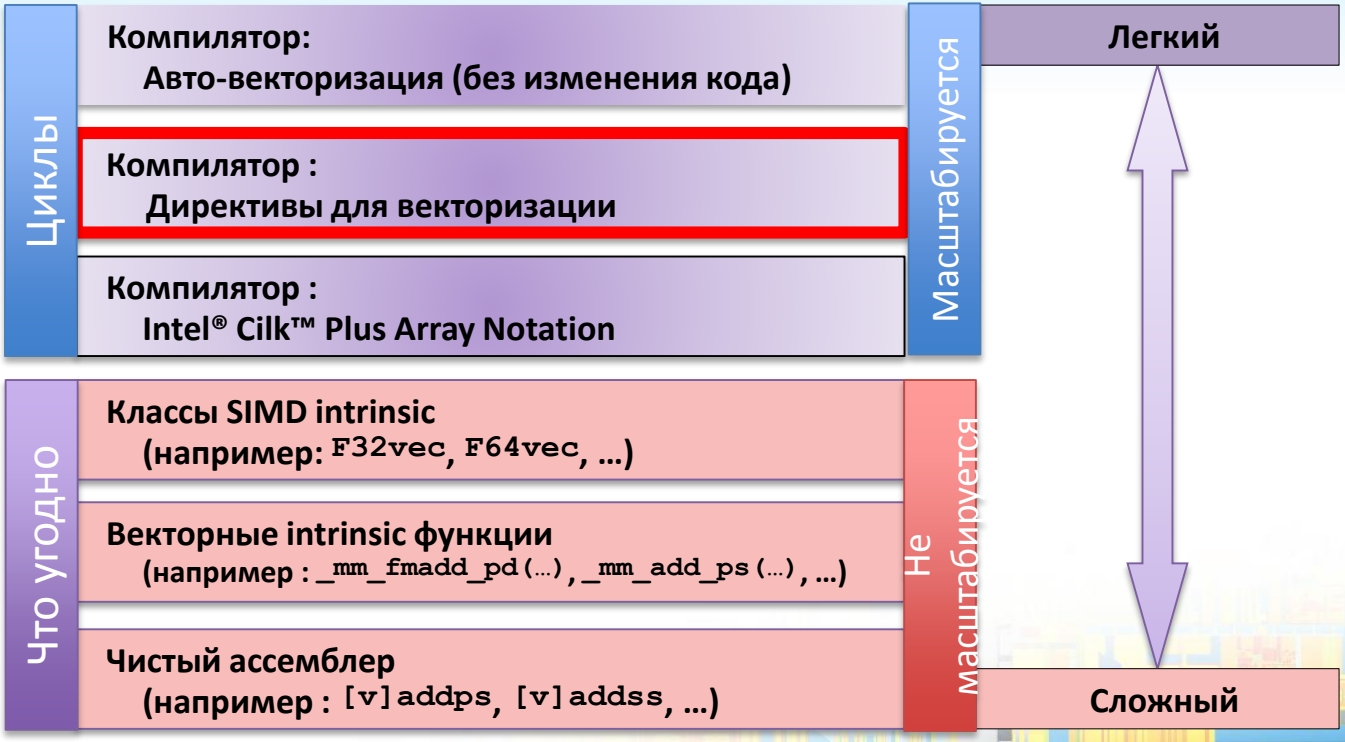


Различные способы векторизации



Code the Future

Различные способы векторизации



Code the Future

Пример vec/v0.cpp

```
void foo(int *A, int N, int n)
{
  int i;

  for (i=N; i<n+N; i++){
    A[i] = A[i] + A[i-N];
  }
}
```

Code the Future

Пример vec/v0.cpp

```
void foo(int *A, int N, int n)
{
  int i;
#pragma omp simd safelen(4)
  for (i=N; i<n+N; i++){
    A[i] = A[i] + A[i-N];
  }
}
```

Code the Future

Пример vec/v01.cpp

```
short sum(float *A, int n)
{
  int i, x = 0, xt = 0, N;



  for (i=0; i<n; i++) {
     xt = x + A[i]*2;
     x = xt + N;
  }
  return x;
}
```

Code the Fut

Пример vec/v01.cpp

```
short sum(float *A, int n)
{
  int i, x = 0, xt = 0, N;

#pragma omp simd reduction(+:x)
  for (i=0; i<n; i++) {
     xt = x + A[i]*2;
     x = xt + N;
  }
  return x;
}
```

Code the Fut

В чём проблема?

В чём проблема?

Зависимости по данным

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

В чём проблема?

Зависимости по данным

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

Вызов функции

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

В чём проблема?

Зависимости по данным

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

Вызов функции

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

Возможные зависимости

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
        b[i] = z * a[i];
}
```

В чём проблема?

Зависимости по данным

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

Вызов функции

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

Возможные зависимости

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
        b[i] = z * a[i];
}
```

Переменное число итераций

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

В чём проблема?

Зависимости по данным

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

Вызов функции

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

Возможные зависимости

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
        b[i] = z * a[i];
}
```

Переменное число итераций

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

Неравномерные доступ к памяти

```
 for (i=0; i<N; i++)
     A[B[i]] = C[i]*D[i]
```

В чём проблема?

Зависимости по данным

```
DO I = 1, N
   A(I+1) = A(I) + B(I)
ENDDO
```

Вызов функции

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

Возможные зависимости

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
        b[i] = z * a[i];
}
```

Переменное число итераций

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

Неравномерные доступ к памяти

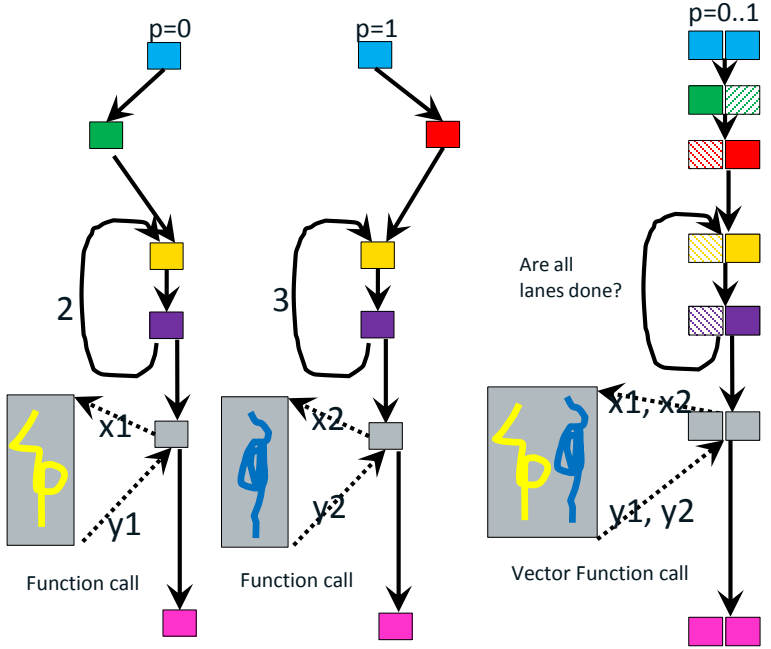
```
 for (i=0; i<N; i++)
     A[B[i]] = C[i]*D[i]
```

Внешний цикл

```
for(i = 0; i <= MAX; i++) {
  for(j = 0; j <= MAX; j++) {
    D[i][j] += 1;
  }
}
```

Vectorization today

```
#pragma omp simd reduction(+:….)
for(p=0; p<N; p++) {
  // Blue work
  if(…) {
     // Green work
  } else {
     // Red work
  }
  while(…) {
     // Gold work
     // Purple work
  }
  y = foo (x);
  Pink work
}
```



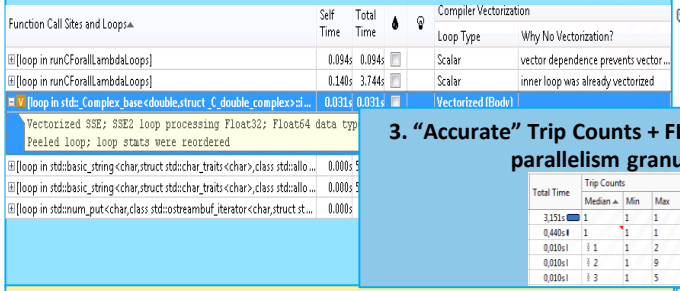
Two fundamental problems
- ✓ Data divergence
- ✓ Control divergence

Increasing need for user guided explicit vectorization
Explicit vectorization maps threaded execution to simd hardware

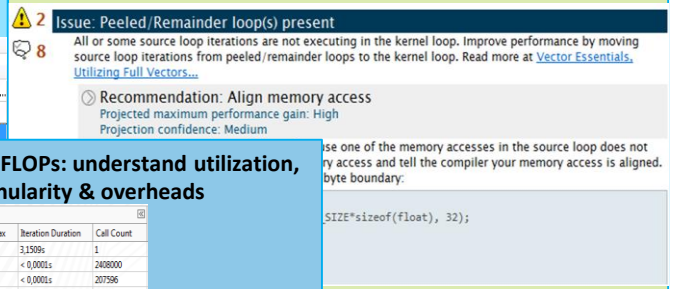
5 Steps to Efficient Vectorization - Vector Advisor

(part of Intel® Advisor, Parallel Studio, Cluster Studio 2016)

1. Compiler diagnostics + Performance Data + SIMD efficiency information

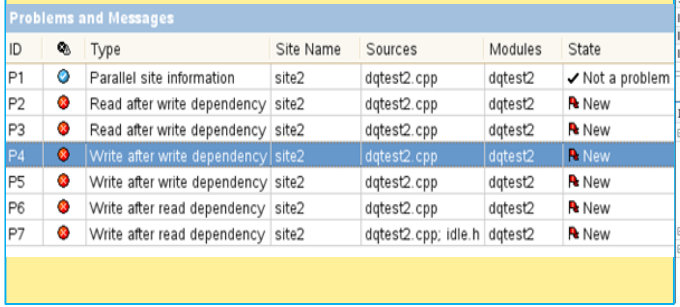


2. Guidance: detect problem and recommend how to fix it

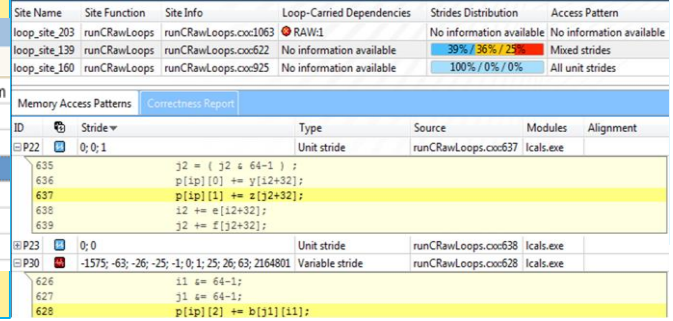


3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

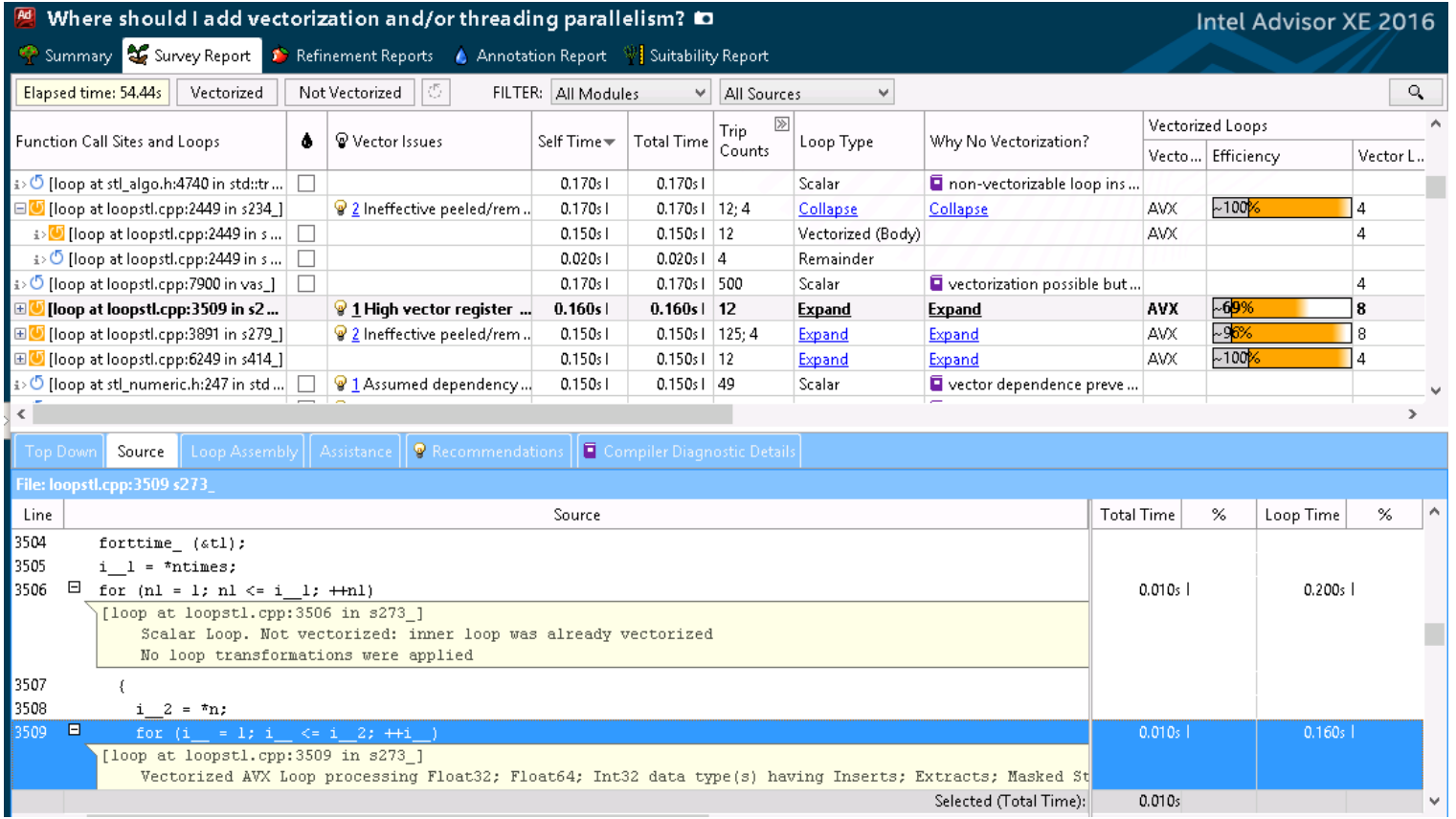
4. Loop-Carried Dependency Analysis



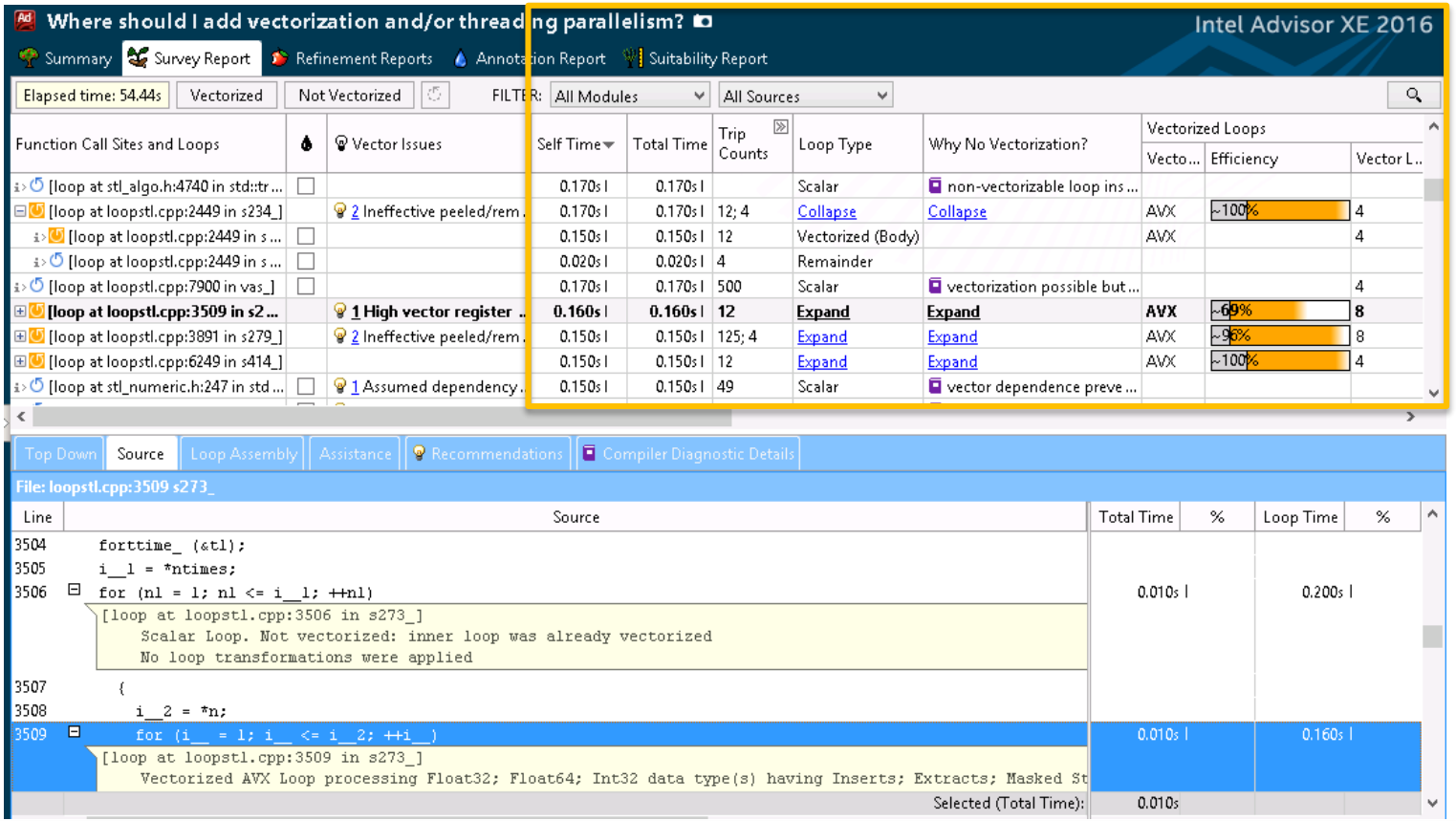
5. Memory Access Patterns Analysis



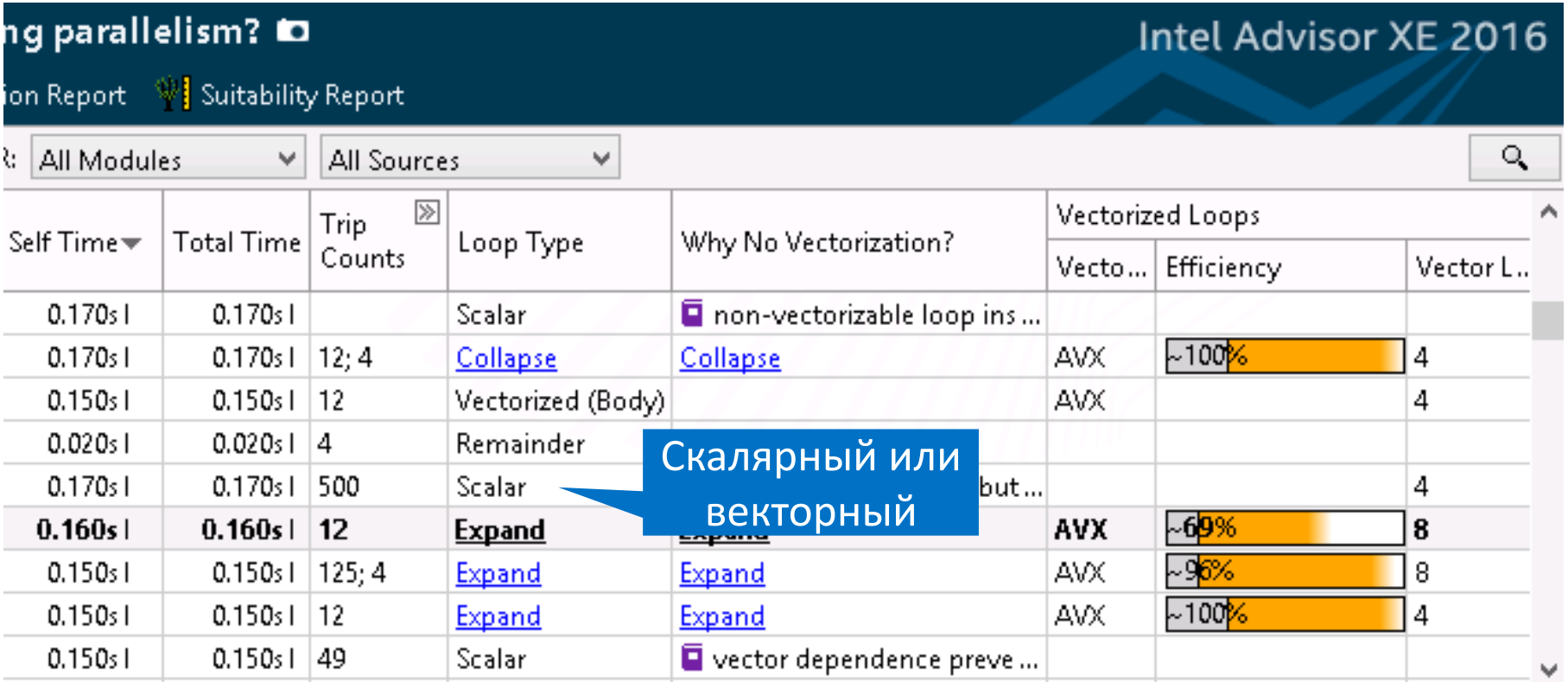
Диагностика SIMD циклов



Диагностика SIMD циклов

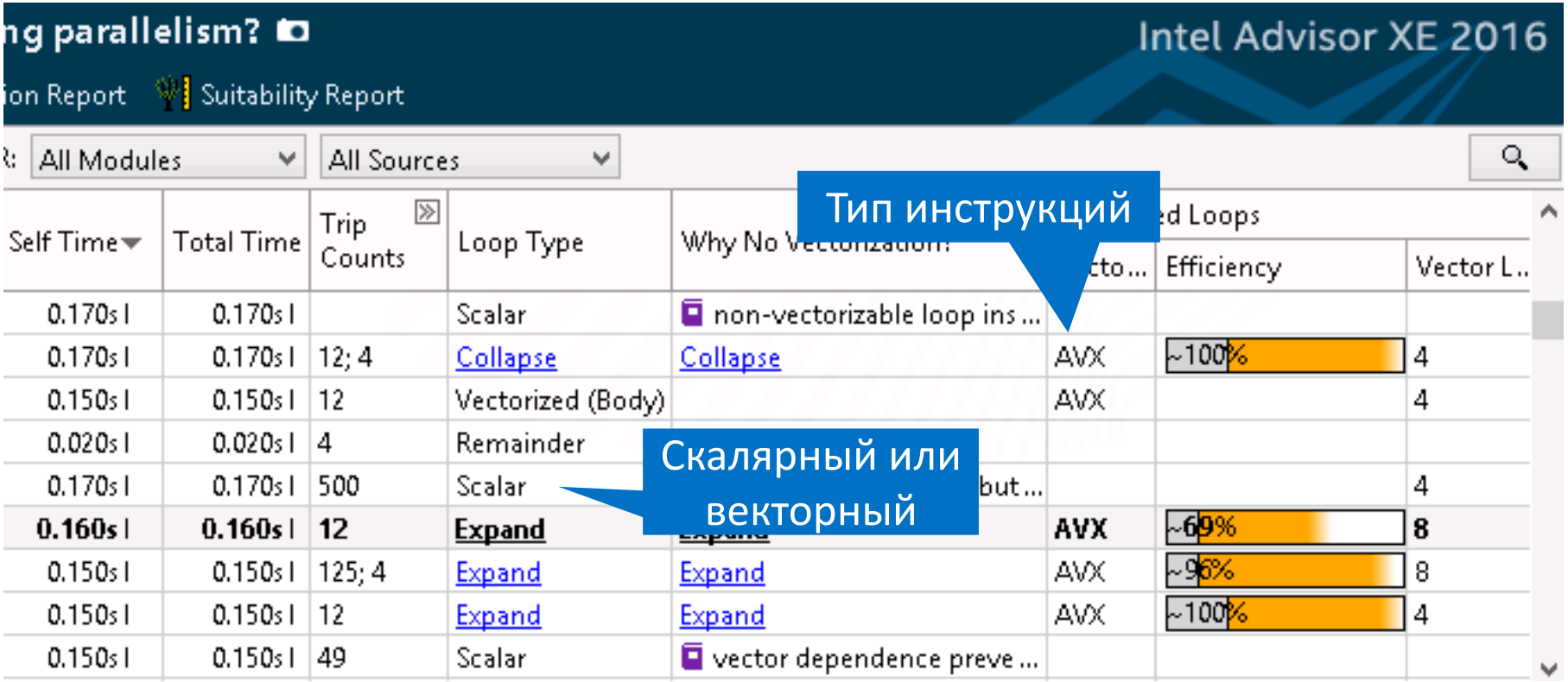


Диагностика SIMD циклов



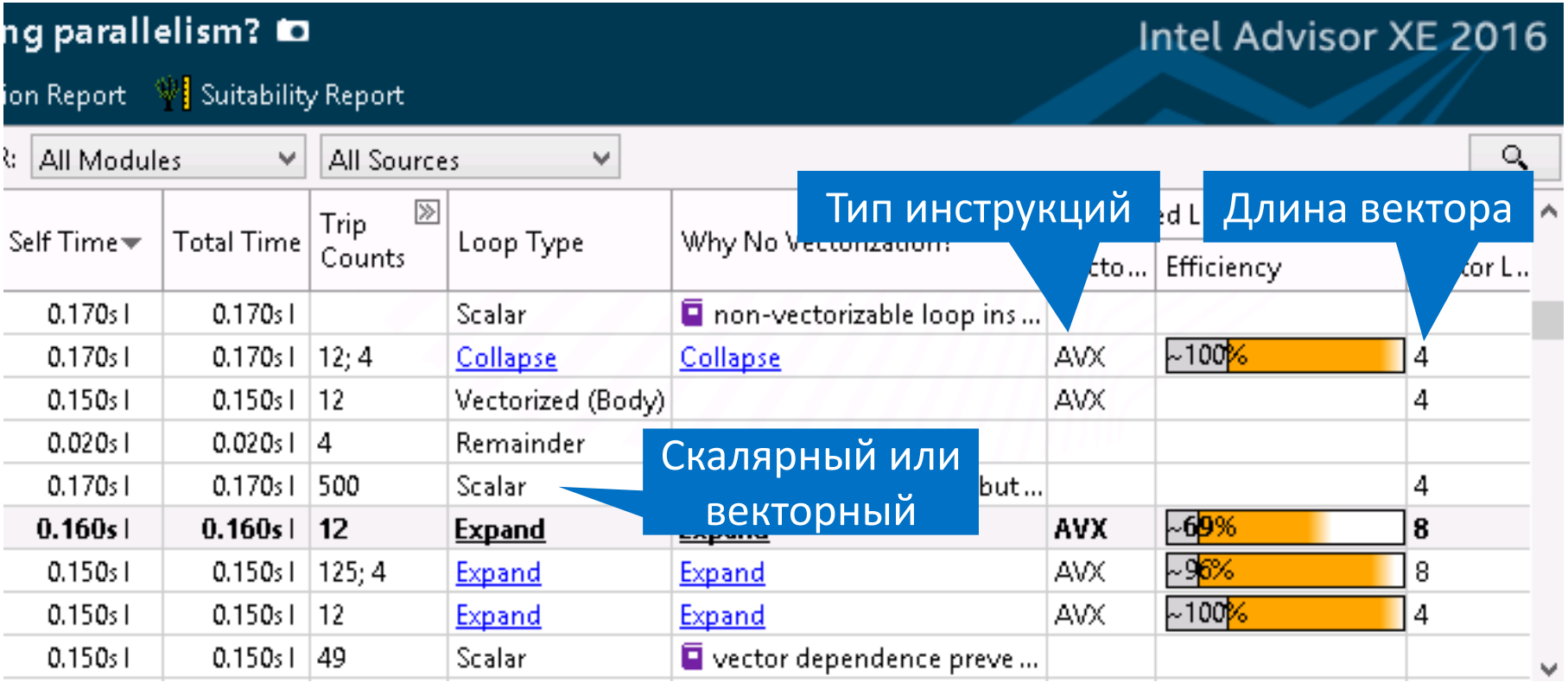
Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
					Vecto...	Efficiency	Vector L...
0.170s	0.170s		Scalar	non-vectorizable loop ins...			
0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
0.150s	0.150s	12	Vectorized (Body)		AVX		4
0.020s	0.020s	4	Remainder				
0.170s	0.170s	500	Scalar	but...			4
0.160s	**0.160s**	**12**	**Expand**		**AVX**	~69%	**8**
0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
0.150s	0.150s	49	Scalar	vector dependence preve...			

Диагностика SIMD циклов



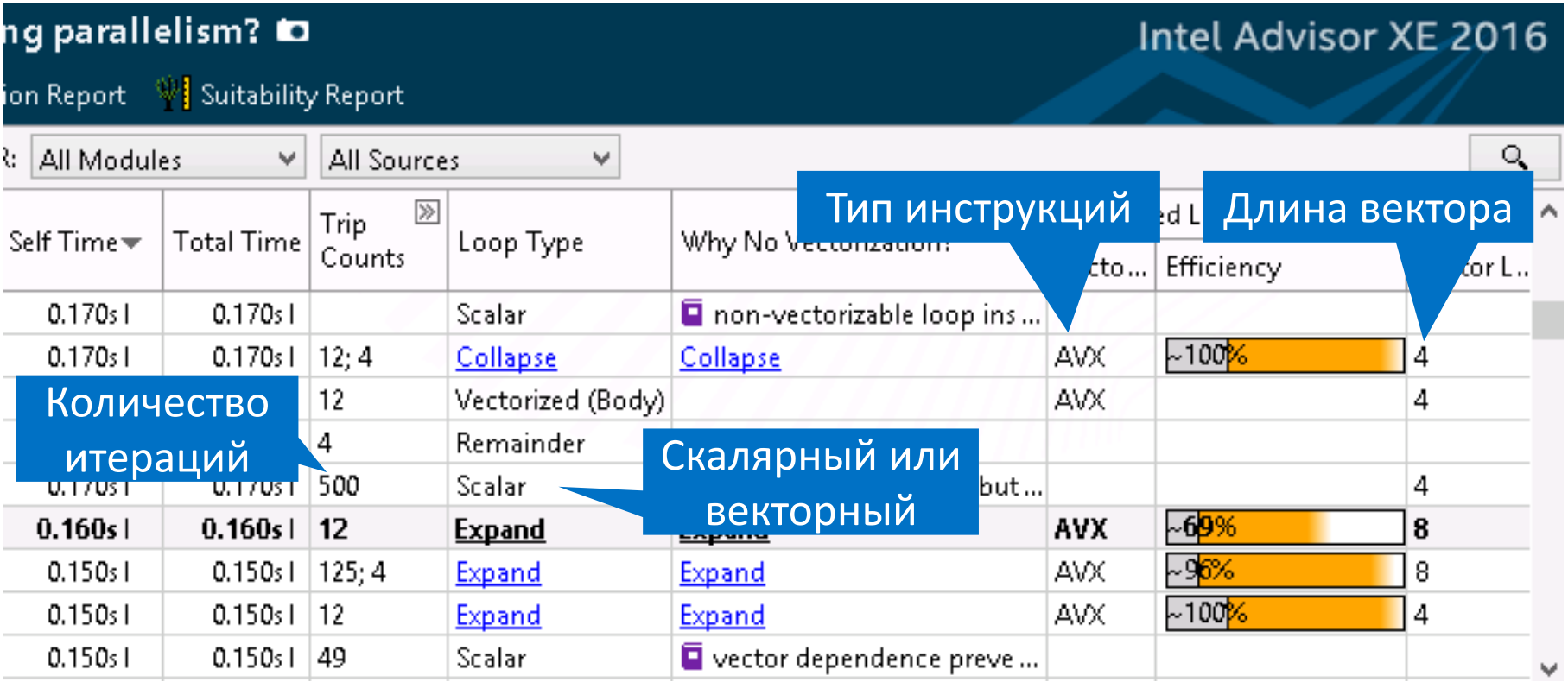
Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	...cto...	...d Loops Efficiency	...d Loops Vector L...
0.170s	0.170s		Scalar	non-vectorizable loop ins...			
0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
0.150s	0.150s	12	Vectorized (Body)		AVX		4
0.020s	0.020s	4	Remainder				
0.170s	0.170s	500	Scalar	...but...			4
0.160s	**0.160s**	**12**	**Expand**		**AVX**	~69%	**8**
0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
0.150s	0.150s	49	Scalar	vector dependence preve...			

Диагностика SIMD циклов

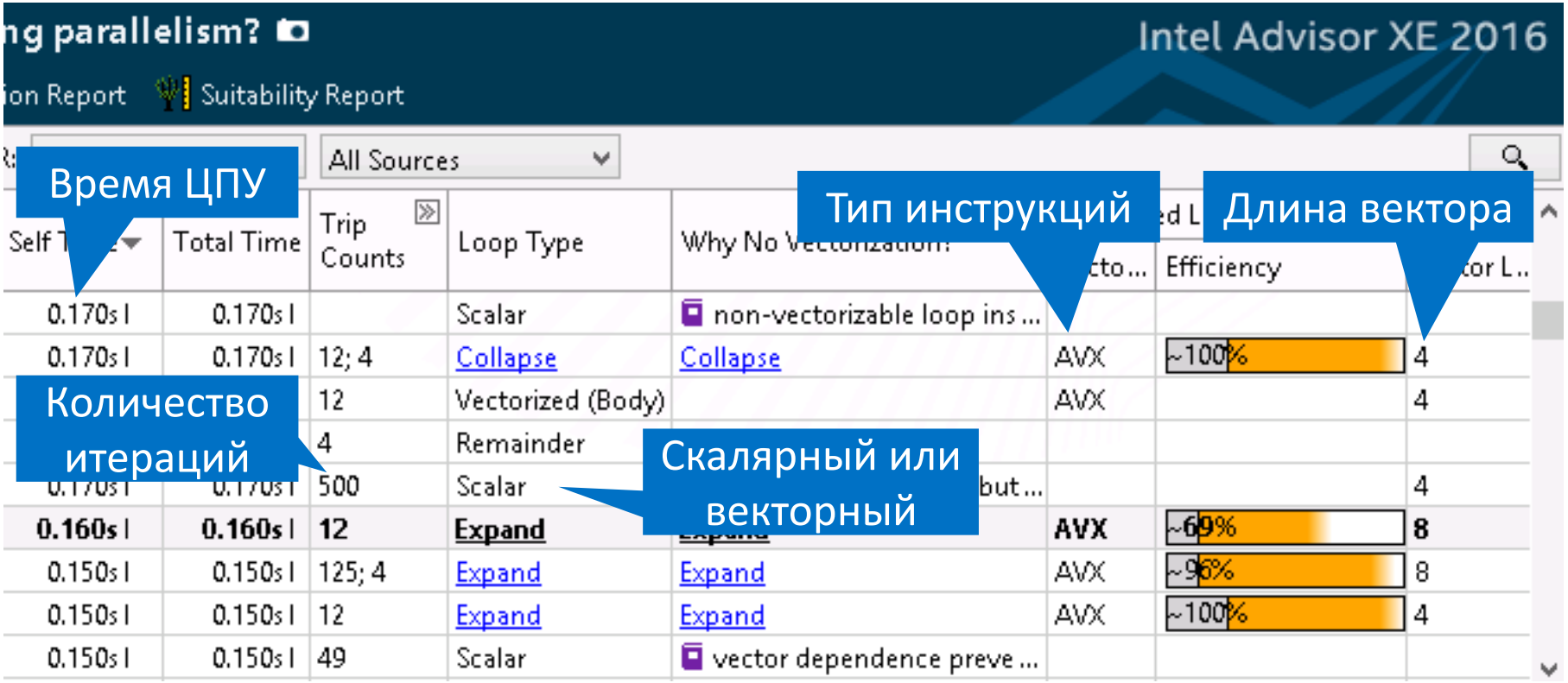


Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	...cto...	Efficiency	...or L...
0.170s	0.170s		Scalar	non-vectorizable loop ins ...			
0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
0.150s	0.150s	12	Vectorized (Body)		AVX		4
0.020s	0.020s	4	Remainder				
0.170s	0.170s	500	Scalar	...but ...			4
0.160s	**0.160s**	**12**	**Expand**		**AVX**	~69%	**8**
0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
0.150s	0.150s	49	Scalar	vector dependence preve ...			

Диагностика SIMD циклов



Диагностика SIMD циклов



Диагностики в исходном коде

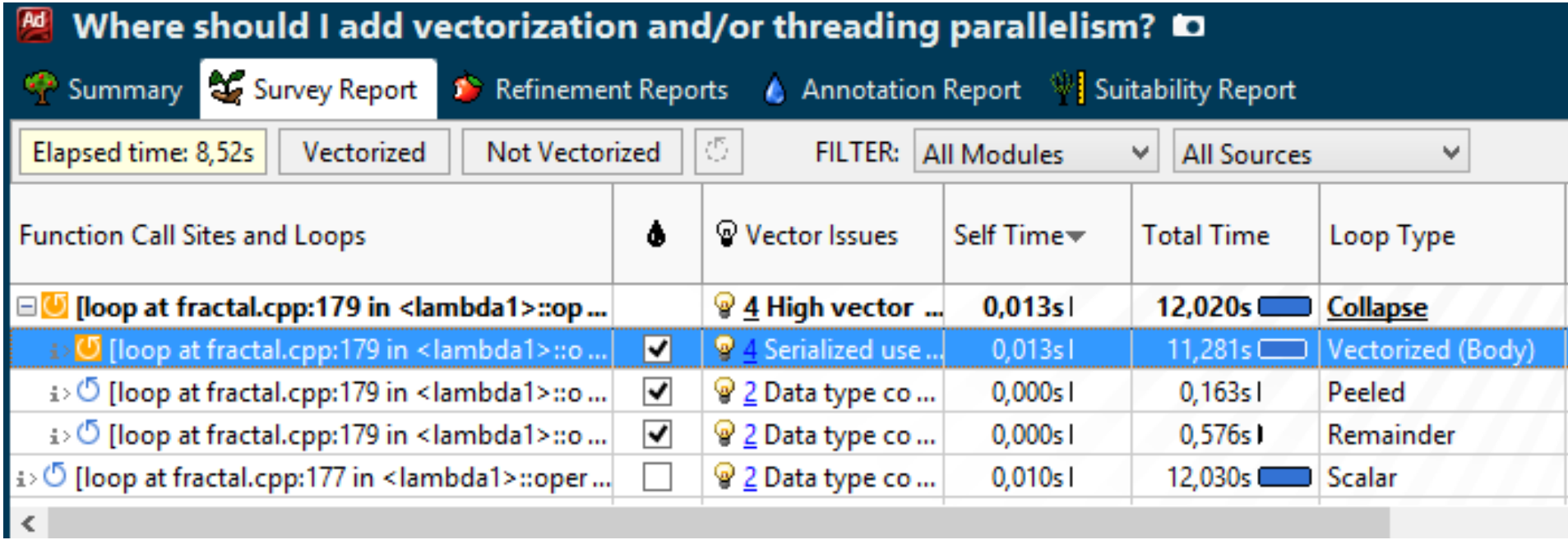
File: fractal.cpp:164 <lambda1>::operator()

Line	Source	Total Time	%
163	for (int x = x0; x < x1; ++x) {		
	[loop at fractal.cpp:163 in <lambda1>::operator()] Scalar Loop. Not vectorized: outer loop was not auto-vectorized: consider usi No loop transformations were applied		
164	for (int y = y0; y < y1; ++y) {		
	[loop at fractal.cpp:164 in <lambda1>::operator()] Scalar Loop. Not vectorized: vectorization possible but seems inefficient. Us Loop was unrolled by 2		
165	fractal_data_array[x - x0][y - y0] = calc_one_pixel(x, y, t	10.822s	
166	}		
167	}		
168	for (int y = y0, y_temp = 0; y < y1; ++y, ++y_temp) {		
169	area.set_pos(0, y - y0);		
170	for (int x = x0, x_temp = 0; x < x1; ++x, ++x_temp) {		
171	area.put_pixel(fractal_data_array[x_temp][y_temp]);		
172	}		
173	}	0.196s	

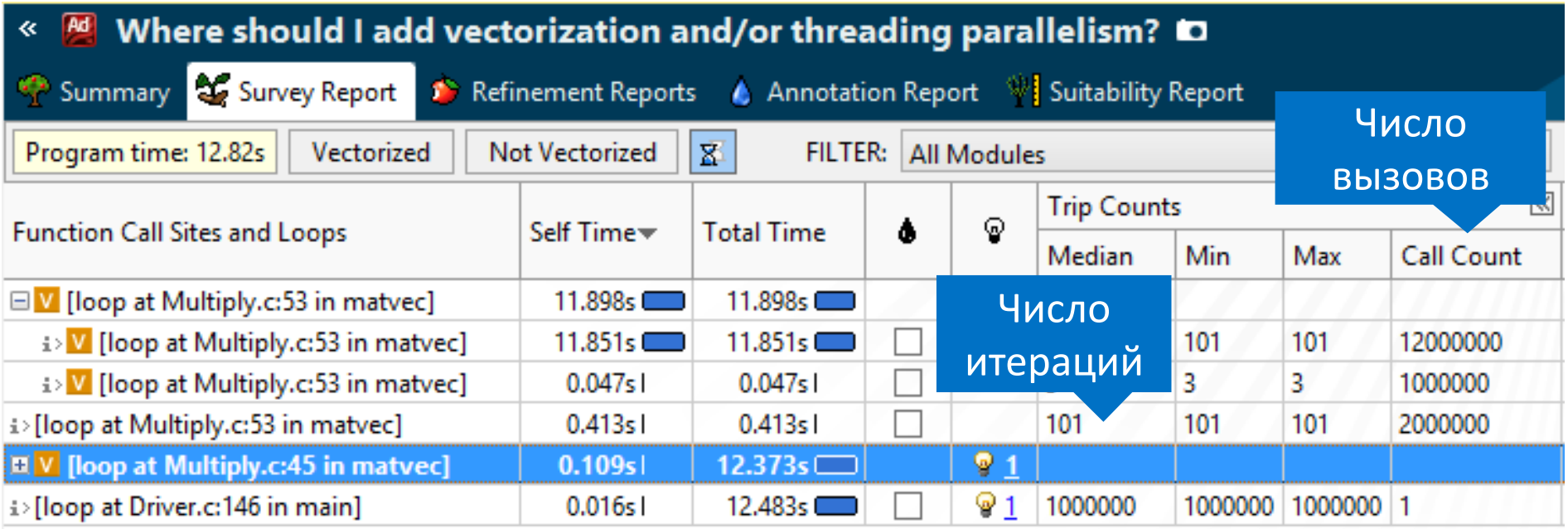
Эффективность векторного цикла

Loops	Vecto...	Efficiency	Estimated Gain	Vect...
[loop at lbpSUB.cpp:1280 in fPropagationS ...	AVX	13%	0,53	4
[loop at lbpGET.cpp:152 in fGetFracSite]	AVX	30%	2,38	8
[loop at lbpGET.cpp:42 in fGetOneMassSite]	AVX	36%	2,86	8
[loop at lbpGET.cpp:78 in fGetTotMassSite]	AVX	36%	2,86	8
[loop at lbpGET.cpp:334 in fGetOneDirecSp ...	AVX	38%	3,05	8
[loop at lbpBGK.cpp:840 in fCollisionBGK]	AVX	100%	2,05	2

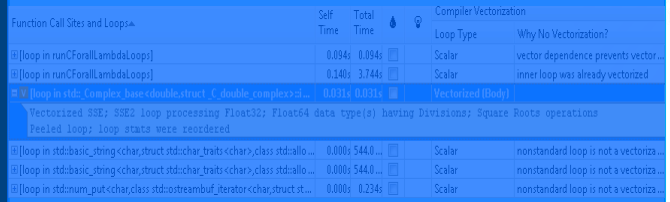
Части векторного цикла



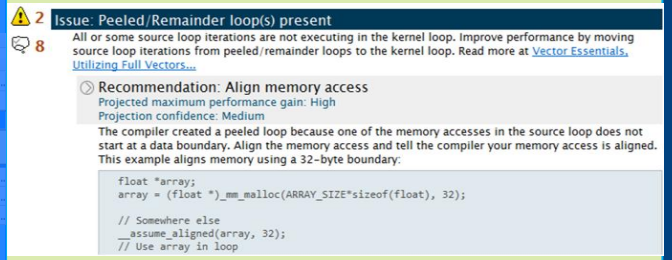
Число итераций



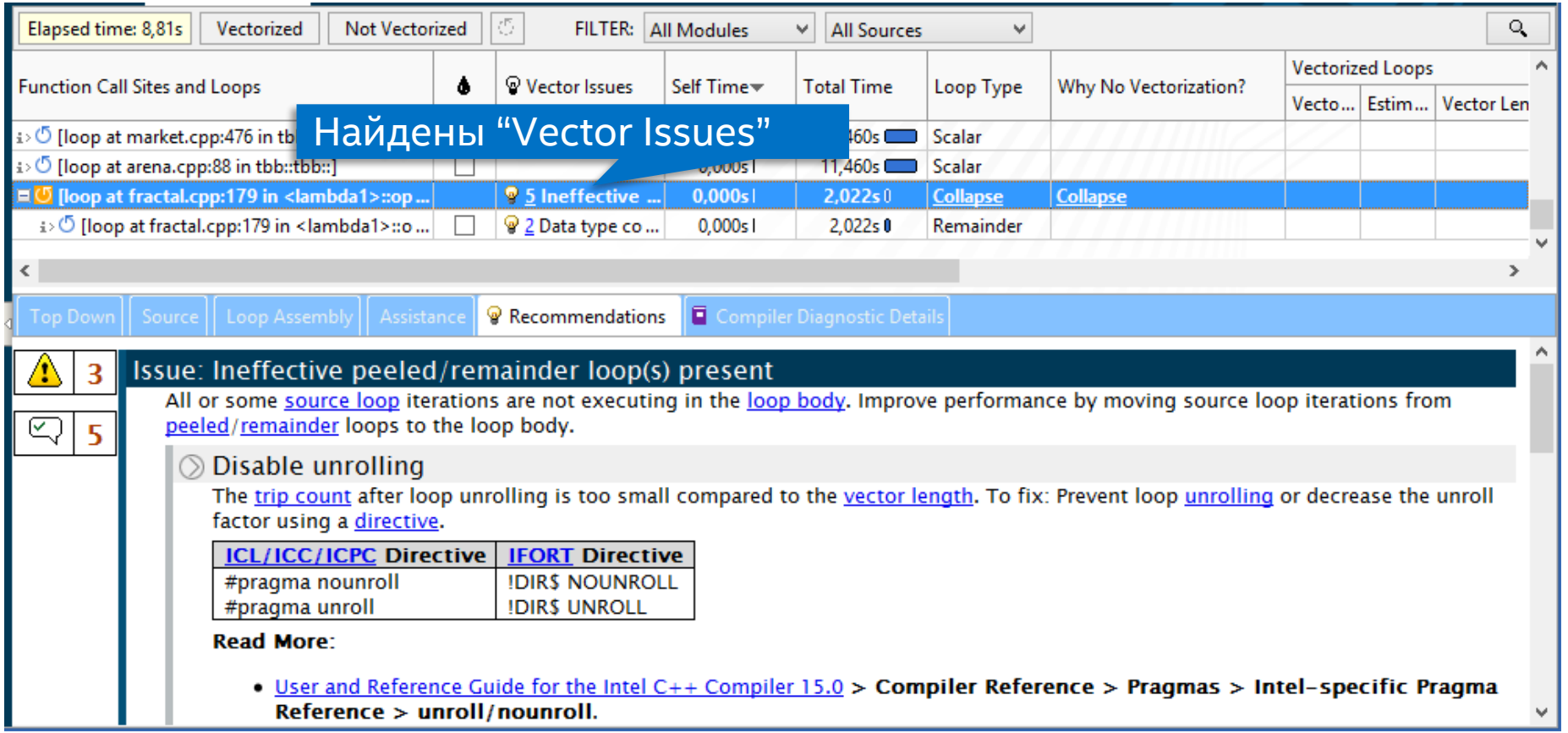
1. Диагностика SIMD



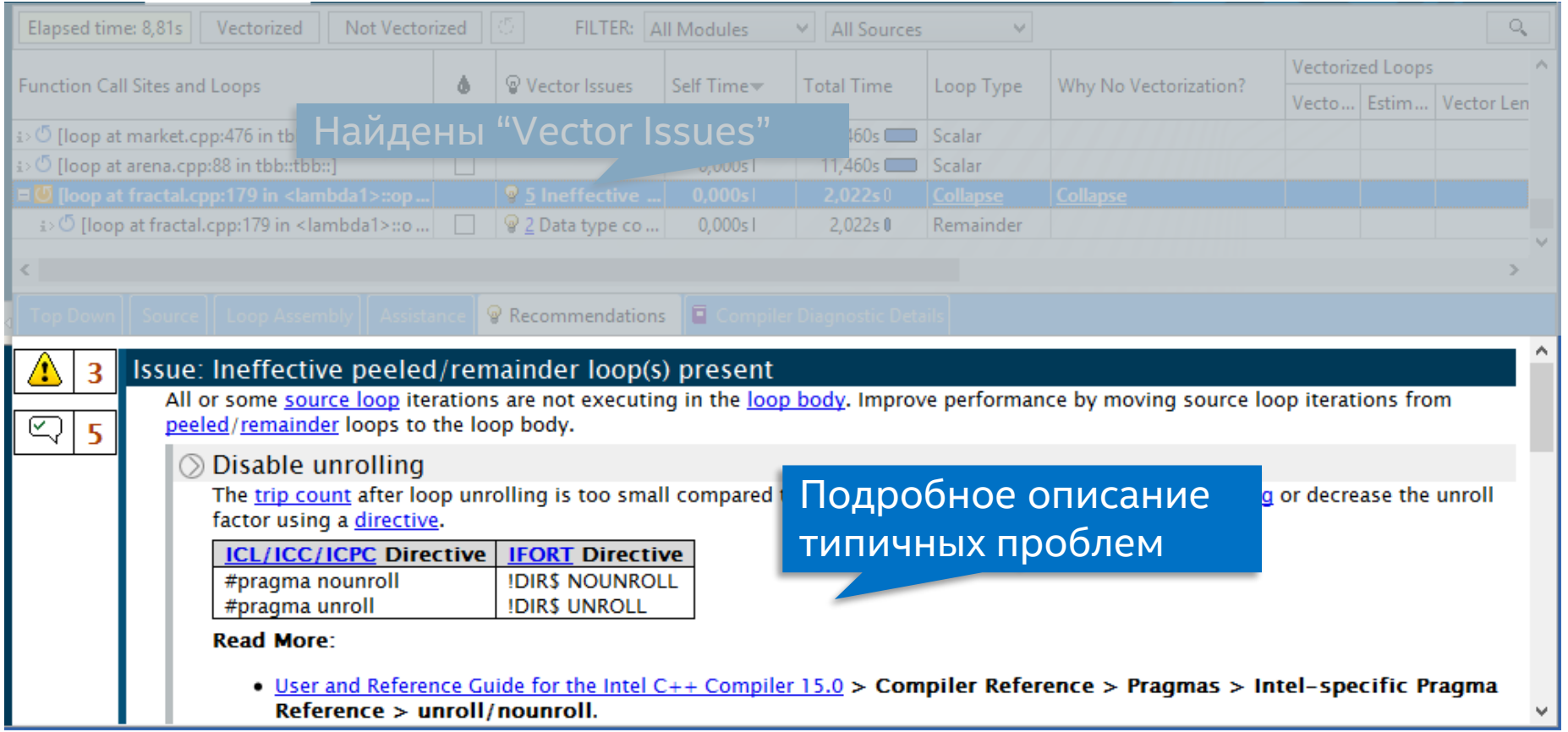
2. Рекомендации



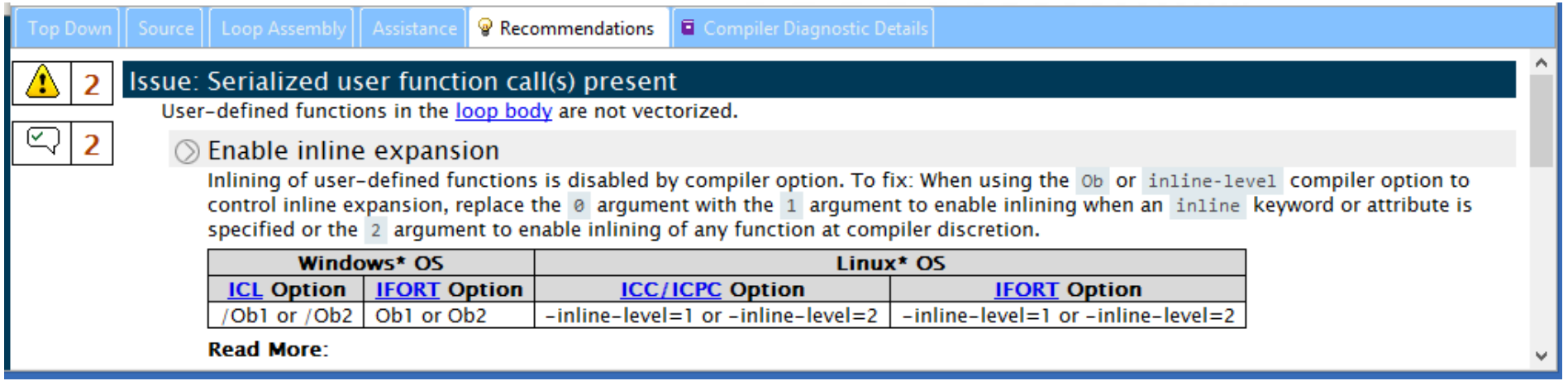
Рекомендации



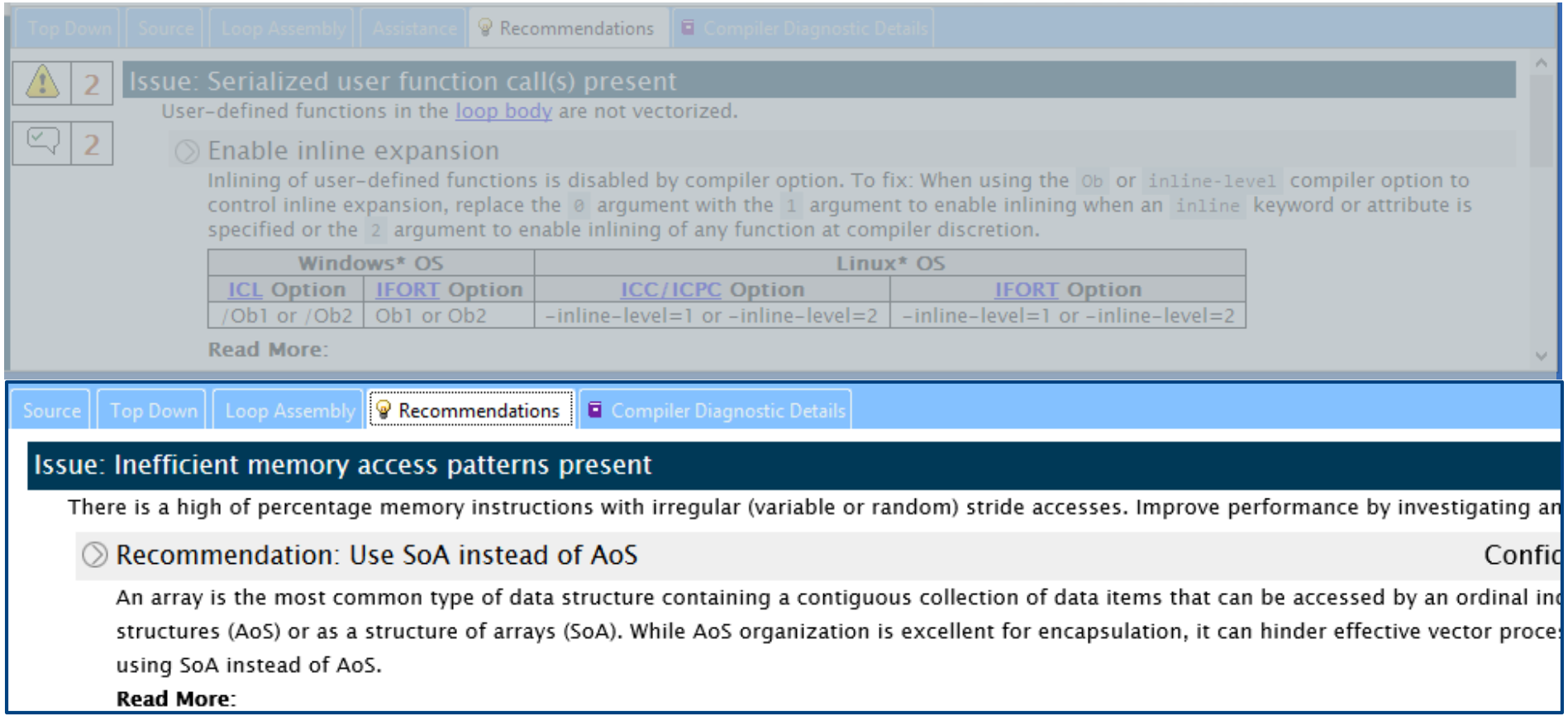
Рекомендации



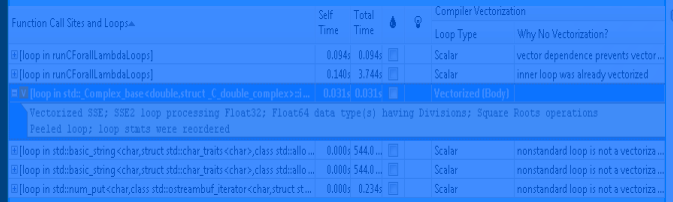
Рекомендации



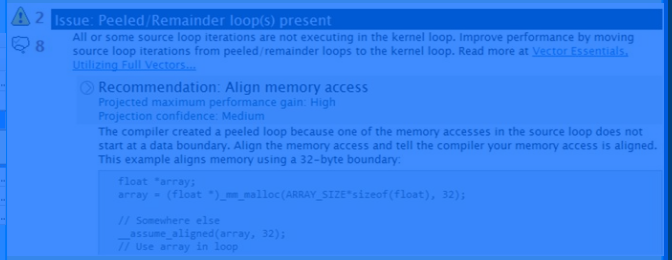
Рекомендации



1. Диагностика SIMD циклов



2. Рекомендации



3. Анализ зависимостей

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	site2	dqtest2.cpp	dqtest2	✔ Not a problem
P2		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6		Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7		Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

Зависимости по данным

```
DO I = 1, N
   A(I) = A(I-1) * B(I)
ENDDO
```

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
        b[i] = z * a[i];
}
```

Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Enable vectorization

Potential performance gain: Information not available until Beta Update release

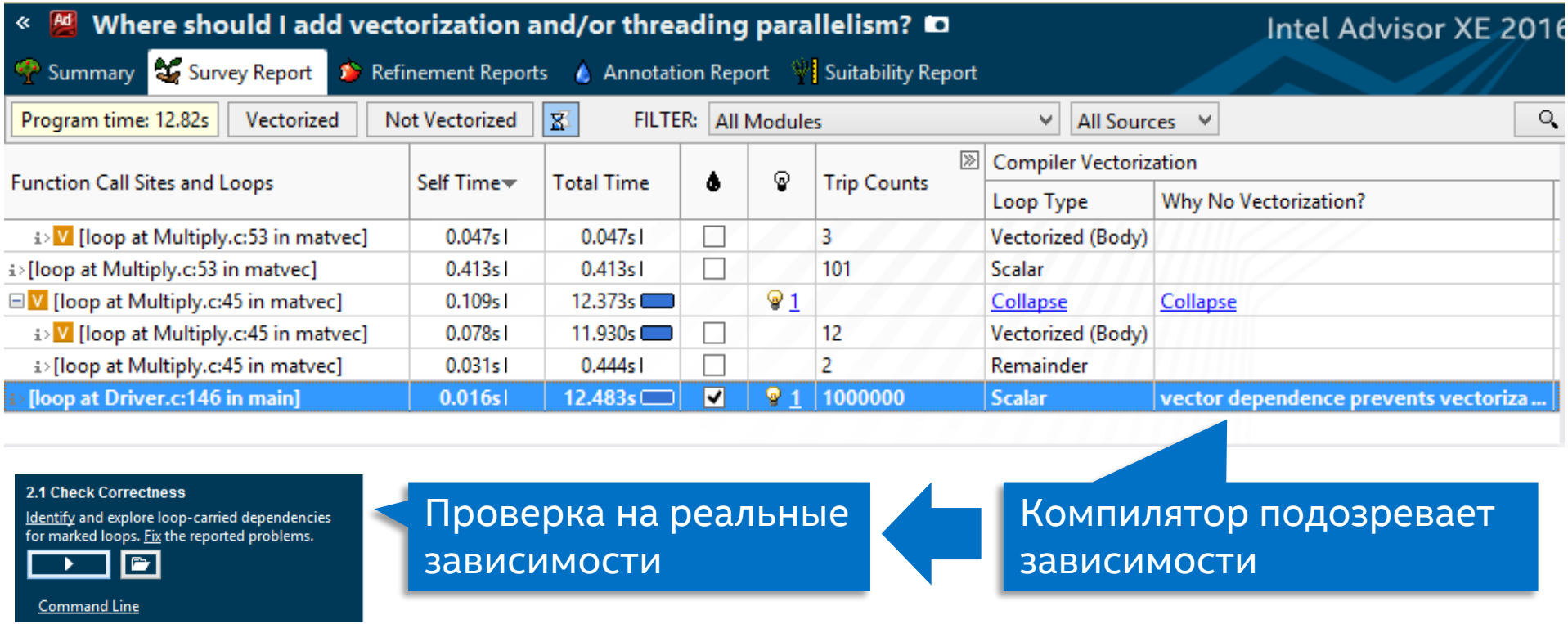
Confidence this recommendation applies to your code: Information not available until Beta Update release

The Correctness analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a directive.

ICL/ICC/ICPC Directive	IFORT Directive	Outcome
#pragma simd or #pragma omp simd	!DIR$ SIMD or !$OMP SIMD	Ignores all dependencies in the loop
#pragma ivdep	!DIR$ IVDEP	Ignores only vector dependencies (which is safest)

Read More:

Анализ зависимостей



Проверка на реальные зависимости

Компилятор подозревает зависимости

Реально существующие зависимости

Memory Access Patterns Report | Correctness Report

Problems and Messages

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	loop_site_6	main.cpp	test_1.exe	✔ Not a problem
P3	Read after write dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	New
P4	Write after write dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	New
P5	Write after read dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	New

Write after read dependency: Code Locations

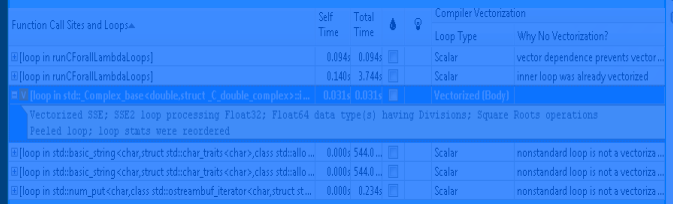
ID	Description	Source	Function	Module	State
X17	Read	main.cpp:22	main	test_1.exe	New

```
20          k += a[9];
21          k *= a[8];
22          k -= a[7];
23          k += a[6];
24          k *= a[5];
```

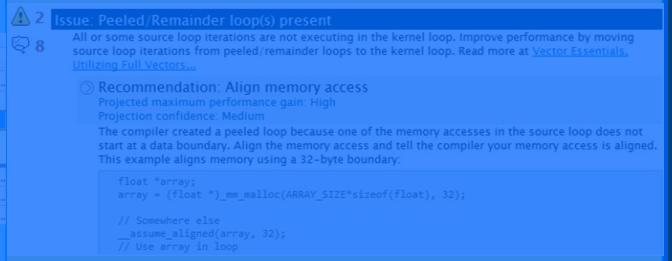
X18	Read	main.cpp:23	main	test_1.exe	New

```
21          k *= a[8];
22          k -= a[7];
23          k += a[6];
```

1. Диагностика SIMD циклов

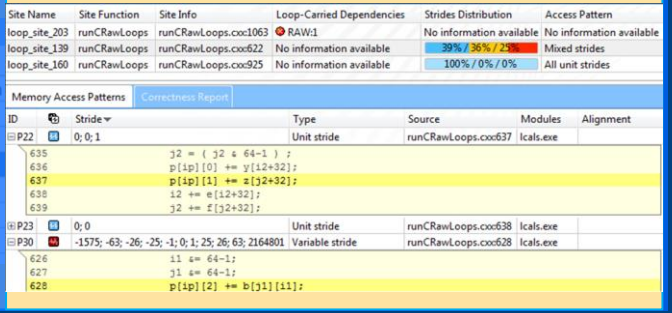


2. Рекомендации



3. Анализ зависимостей

4. Анализ доступов к памяти



Шаблоны доступа к памяти

Unit-Stride access

```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

Шаблоны доступа к памяти

Unit-Stride access

```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

Constant stride access

```
for (i=0; i<N; i++)
    point[i].x = x[i]
```

Шаблоны доступа к памяти

Unit-Stride access

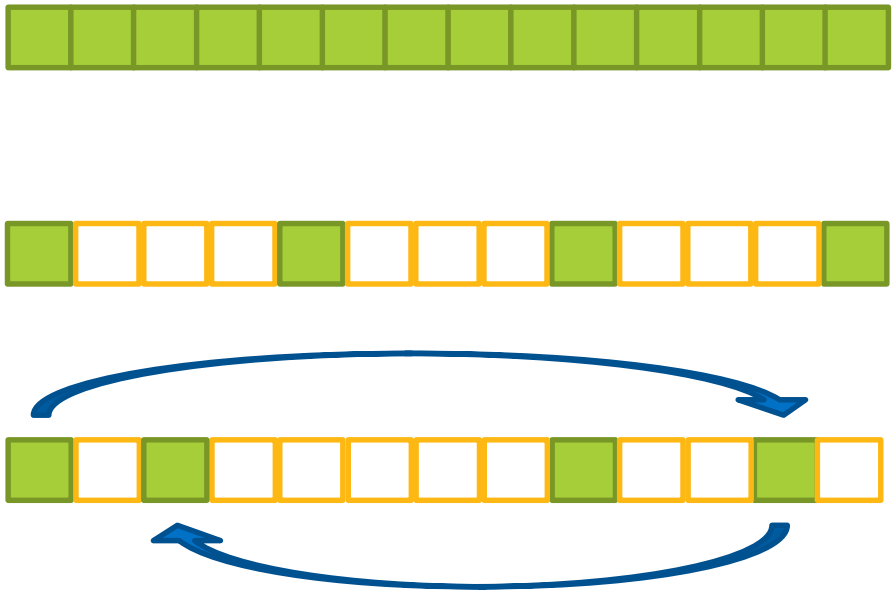
```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

Constant stride access

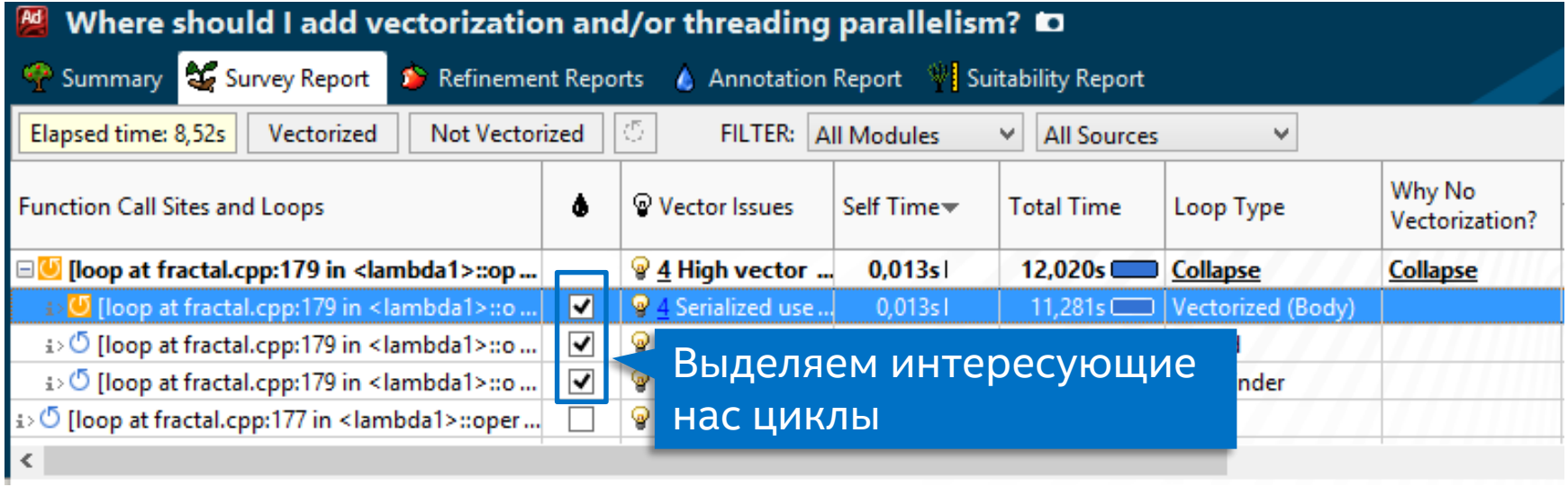
```
for (i=0; i<N; i++)
    point[i].x = x[i]
```

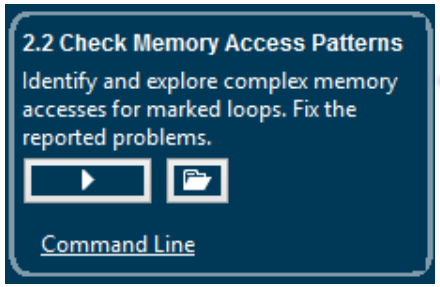
Variable stride access

```
for (i=0; i<N; i++)
    A[B[i]] = C[i]*D[i]
```



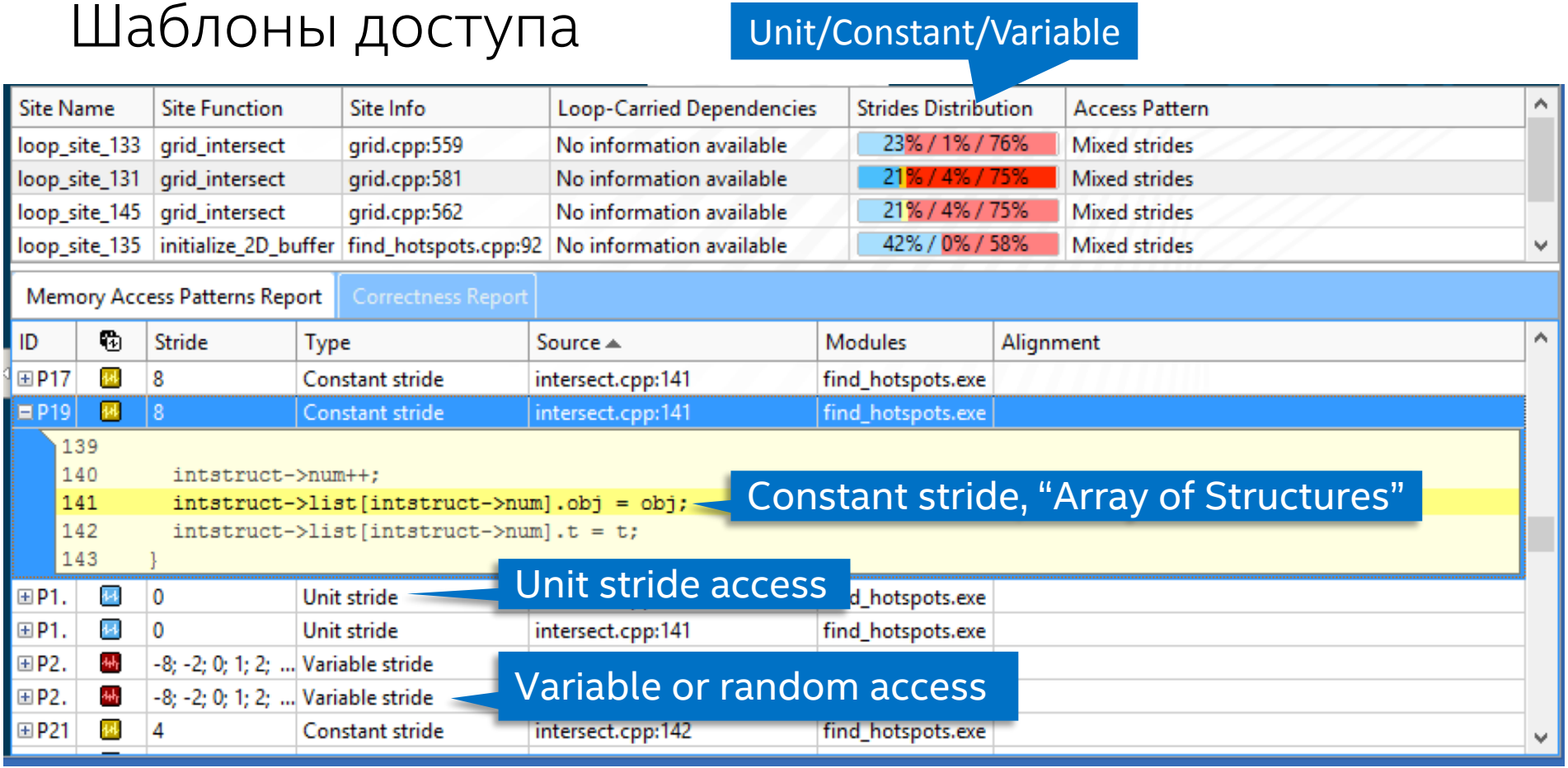
Анализ шаблонов доступа





Запускаем анализ шаблонов доступа

Шаблоны доступа



Improving vectorization: data layout

Reminder from "Best practices for Vectorization" talk

Vectorization more efficient with unit strides

- Non-unit strides will generate gather/scatter
- Unit strides also better for data locality
- Compiler might refuse to vectorize

AoS vs SoA

- Layout your data as Structure of Arrays (SoA)

Traverse matrices in the right direction

- C/C++: `a[i][:]`, Fortran: `a(:,i)`
- Loop interchange might help
  - Usually the compiler is smart enough to apply it
  - Check compiler optimization report

Array of Structures vs Structure of Arrays

```
// 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)

ROOFLINE PERFORMANCE MODEL

- TECHNOLOGY LANDSCAPE
- ROOFING A HOUSE

Available since Intel® Advisor 2017 Update 2

Accessible since Intel® Advisor 2017 Update 1 (experimental feature)

Acknowledgments/References

Roofline model proposed by Williams, Waterman, Patterson:
http://www.eecs.berkeley.edu/~waterman/papers/roofline.pdf

"Cache-aware Roofline model: Upgrading the loft" (Ilic, Pratas, Sousa, INESC-ID/IST, Thec Uni of Lisbon) http://www.inesc-id.pt/ficheiros/publicacoes/9068.pdf

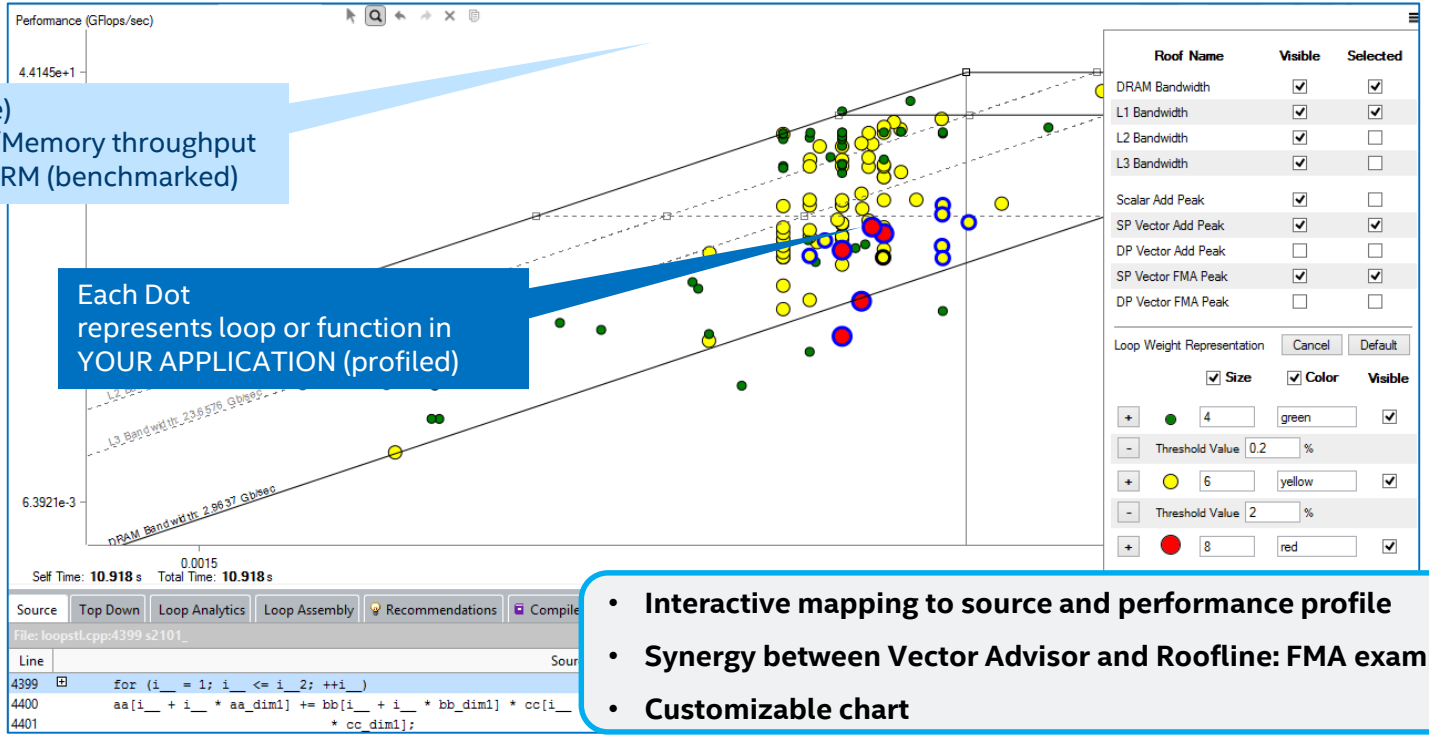
At Intel:

Roman Belenov, **Zakhar Matveev**, Julia Fedorova
SSG product teams, Hugh Caffey,
in collaboration with **Philippe Thierry**

Roofline Automation in Intel Advisor 2017 Update 1-2

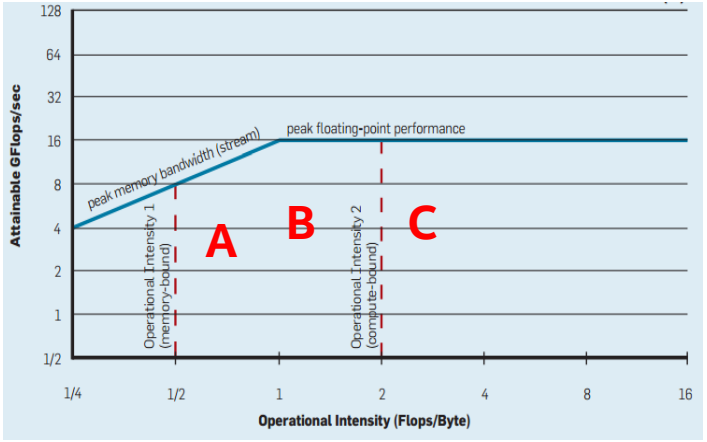
Each Roof (slope)
Gives peak CPU/Memory throughput of your PLATFORM (benchmarked)

Each Dot
represents loop or function in YOUR APPLICATION (profiled)

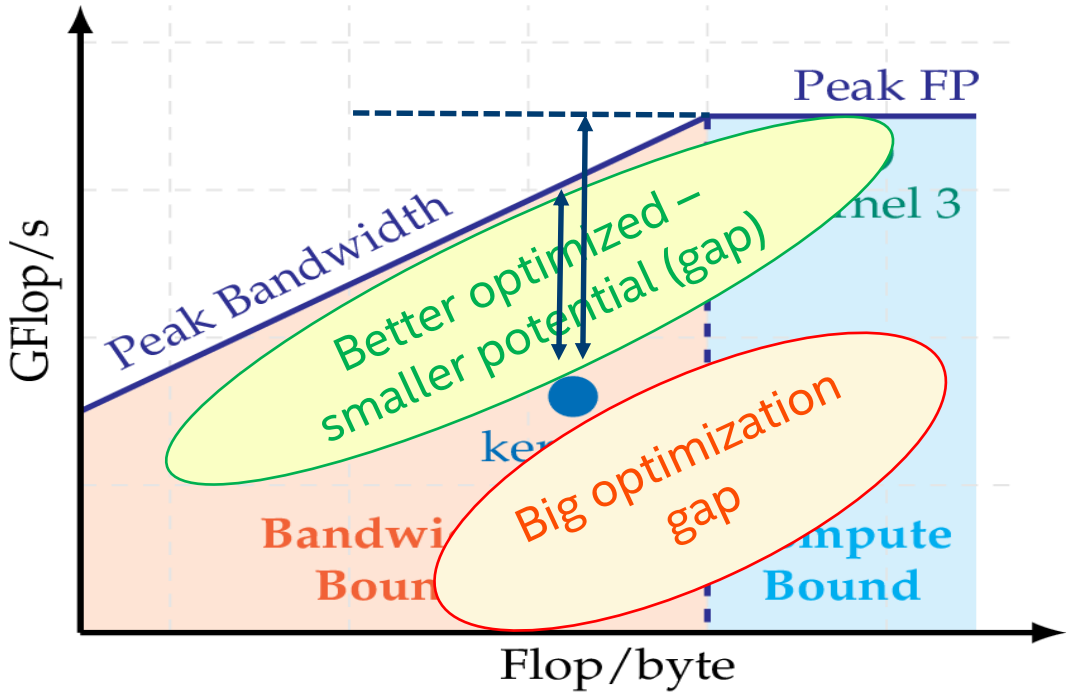


- **Interactive mapping to source and performance profile**
- **Synergy between Vector Advisor and Roofline: FMA example**
- **Customizable chart**

Roofline model: Am I bound by VPU/CPU or by Memory?

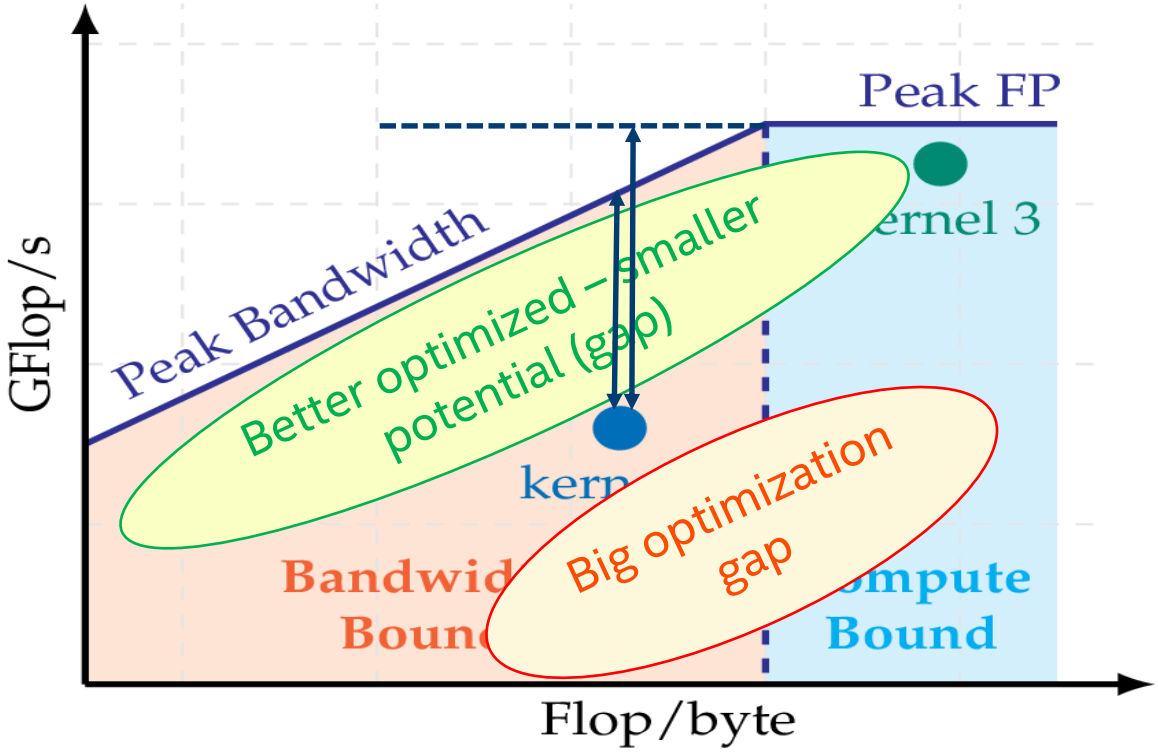


What makes loops **A, B, C** different?



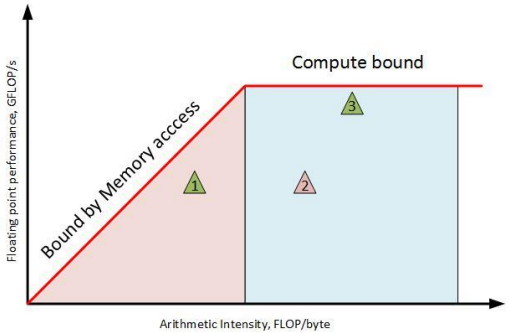
Am I bound by VPU/CPU or by Memory?

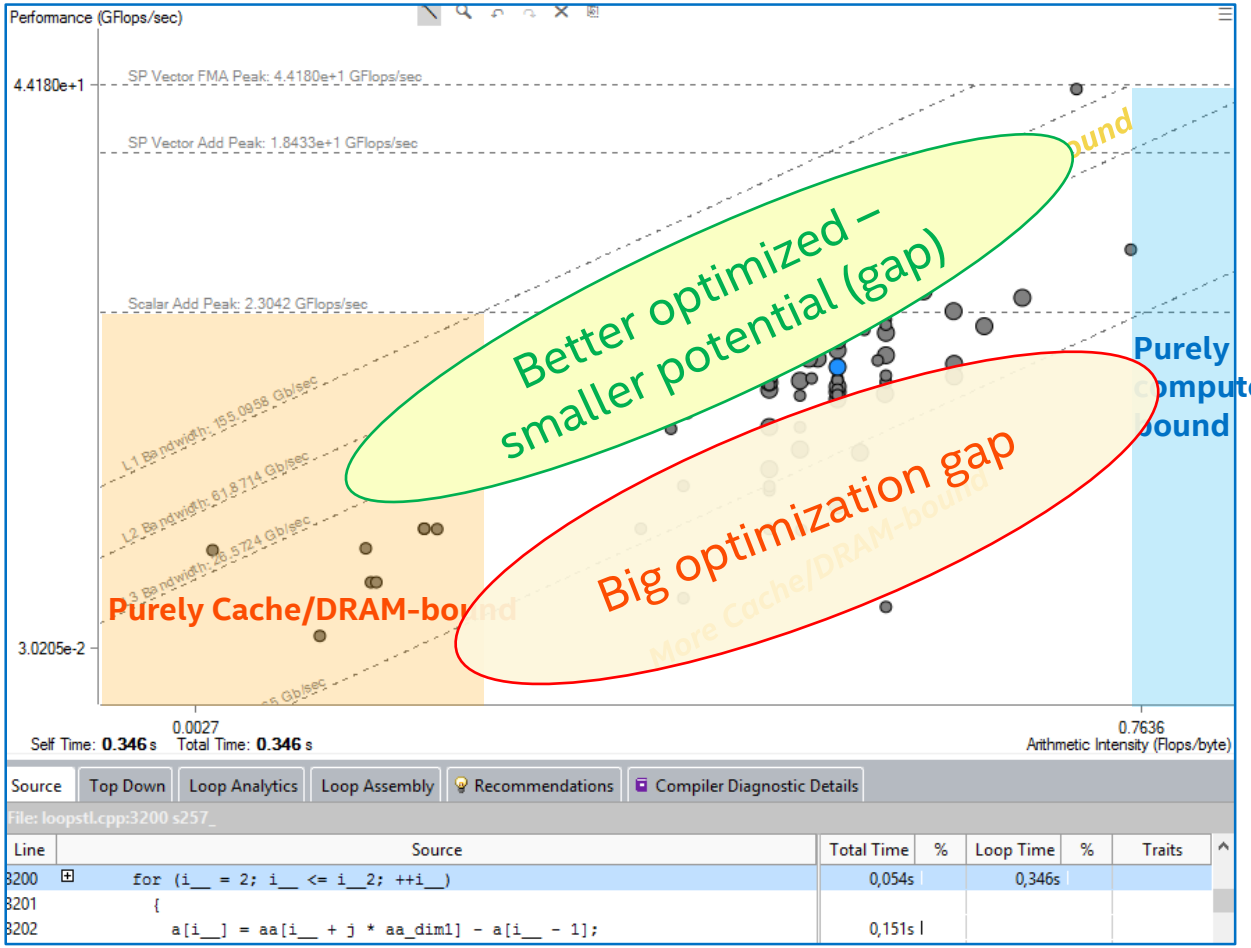
ROOFLINE ANALYSIS



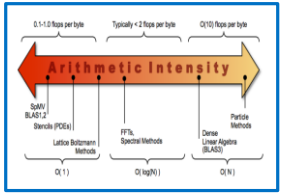
Advisor “Cache-aware” Roofline

AI = #FLOP/#Byte





Cache-Aware vs. Classic Roofline

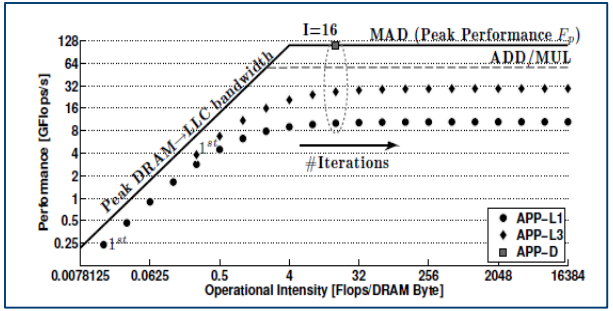


AI = # FLOP / # BYTE

AI_DRAM =

 # FLOP/ # BYTES (CPU & Cache ⇔ **DRAM)**

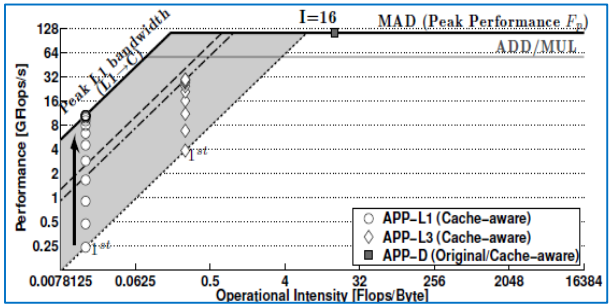
- "DRAM traffic" (or MCDRAM-traffic-based)
- Variable for the same code/platform (varies with dataset size/trip count)
- Can be measured relative to different memory hierarchy levels – cache level, HBM, DRAM



AI_CARM =

 # FLOP / # BYTES (CPU ⇔ Memory Sub-system**)**

- "Algorithmic", "Cumulative (L1+L2+LLC+DRAM)" traffic-based
- Invariant for the given code / platform combination
- Typically AI_CARM < AI_DRAM



Bytes and FLOP/S in Intel (a ka “vector”) Advisor

Intel Advisor: “Cache Aware” Roofline automation

- #**FLOP**, **seconds**, **Bytes** and **IP/RVA mapping** put altogether
- **Break-down** by application phases, loops and functions
- Give FLOP/s from NHM to KNL even if counters do not exist
- Measure L1 <-> Register traffic: what CPU demands from memory sub-system to make a computation
    - Cumulative traffic through L1/L2/LLC/DRAM/MCDRAM

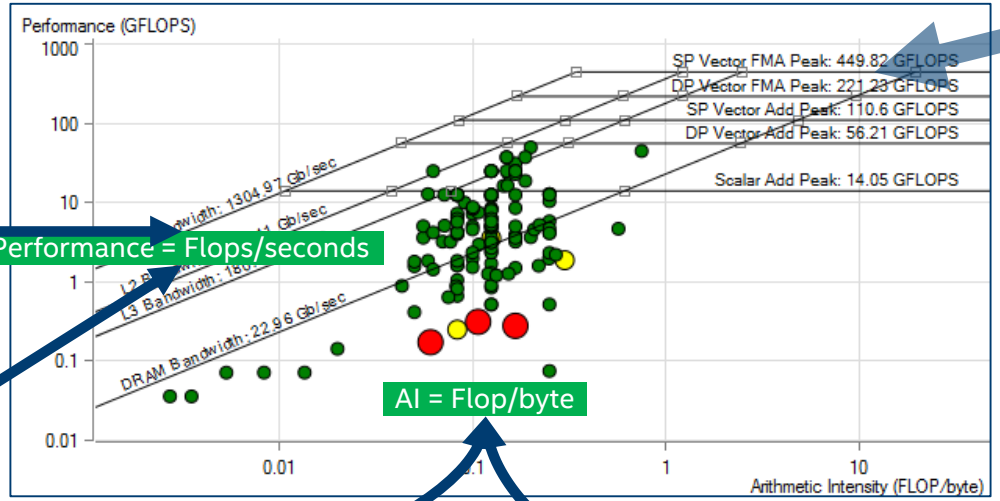
Advisor Roofline: under the hood:

FLOP/S = #FLOP (AVX-512 mask aware) / #Seconds.
AI = #FLOP / #Bytes

Seconds

User-mode **sampling**

Root access not needed



Roofs

Microbenchmarks
Actual peak for the current configuration

FLOPs

Binary **Instrumentation**
Does not rely on CPU counters

Bytes

Binary Instrumentation
Counts operands size (not cachelines)

Mask Utilization and FLOPS profiler

- Long-waiting in HPC: accurate HW independent FLOPs measurement tool
- Not just count FLOPs. Has following additions:
  - (AVX-512 only) Mask-aware. Masked-Memory/Unmasked-Compute pattern aware
  - Unique capability to correlate FLOPs with performance data (obtained without instrumentation). Gives FLOPs/s.
- Lightweight instrumentation, PIN-based, benefits from “threadchecker tools” and more generally Advisor framework integration.

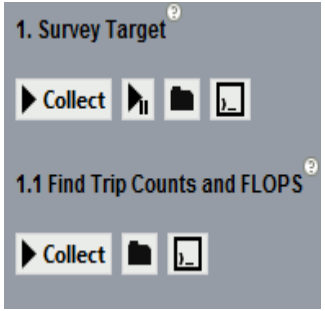
AVX-512 FLOPS and Mask profiler

Low mask population -> low performance (in spite of “high SIMD efficiency”)

Function Call Sites and Loops	Vector Issues	Self Time	Type	FLOPS			Vect…			Instruction Set Analysis	
				GFLOPS	AI	Mask Utilizati…	ector ISA	Efficiency	Gain Esti…	VL	Traits
[loop in s2711 at loops90.f:16 …		0,010s	Vectorized (Remainder)	0,800	0,1000		AVX512	~56%	8.98x	16	FMA
[loop in s252 at loops90.f:1172]	2 Ineffective peeled/r…	0,171s	Vectorized Versions	1,684	0,0968	100%	AVX512	~41%	13.05x	16;	Blends; Divisions; Extracts;
[loop in s116 at loops90.f:257]	1 Ineffective peeled/r…	0,100s	Vectorized (Body; Remainder)	4,951	0,0833	88%	AVX512	~79%	12.68x	16	Unpacks
[loop in s174 at loops90.f:765]	1 Ineffective peeled/r…	0,080s	Vectorized (Body; Remainder)	1,251	0,0833	78%	AVX512	~41%	13.05x	16;	Unpacks
[loop in s173 at loops90.f:7 …	**1 Ineffective peeled …**	**0,090s**	**Vectorized (Body; Remainder)**	**1,111**	**0,0833**	**78%**	**AVX512**	**~41%**	**13.05x**	**16;**	**Unpacks**
[loop in s152 at loops90.f:624]	1 Ineffective peeled/r…	0,010s	Vectorized (Body; Remainder)	10,001	0,0833	89%	AVX512	~37%	11.70x	16;	FMA
[loop in s121 at loops90.f:324]	1 Ineffective peeled/r…	0,020s	Vectorized (Body; Remainder)	4,950	0,0833	88%	AVX512	~36%	11.47x	16;	Unpacks
[loop in s151s at loops90.f:609]	1 Ineffective peeled/r…	0,020s	Vectorized (Body; Remainder)	4,950	0,0833	88%	AVX512	~36%	11.47x	16;	Unpacks
[loop in s131 at loops90.f:521]		0,020s	Vectorized (Body)	4,808	0,0833	100%	AVX512	~36%	11.47x	32	
[loop in s119 at loops90.f:302]	1 Ineffective peeled/r…	0,040s	Vectorized (Body; Remainder)	2,450	0,0833	88%	AVX512	~35%	11.25x	16;	Unpacks
[loop				1,649	0,0833	89%	AVX512	~35%	11.25x	16;	Unpacks
[loop				1,400	0,0833	88%	AVX512	~35%	11.25x	16;	Unpacks
[loop				0,797	0,0833		AVX512	~22%	8.55x	16	FMA
[loop					0,0833	78%	AVX512	~19%		16	2-Source Permutes; Gather

Lower Arithmetic Intensity -> Lower FLOPS and Efficiencies

Getting FLOP/S in Advisor



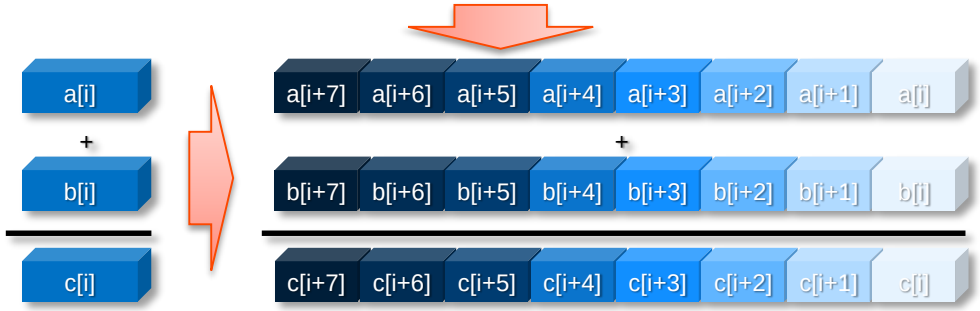
FLOP/S **= #FLOP/Seconds**	**Seconds**	**#FLOP** **- Mask Utilization** **- #Bytes**
Step 1: Survey - Non intrusive. *Representative* - Output: Seconds (+much more)	✓	
Step 2: Trip counts+FLOPS - Precise, instrumentation based - Physically count Num-Instructions - Output: #FLOP, #Bytes		✓

Why Mask Utilization is Important?

100%

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

#FLOP (MAX = 8): **8**



Why Mask Utilization Important?

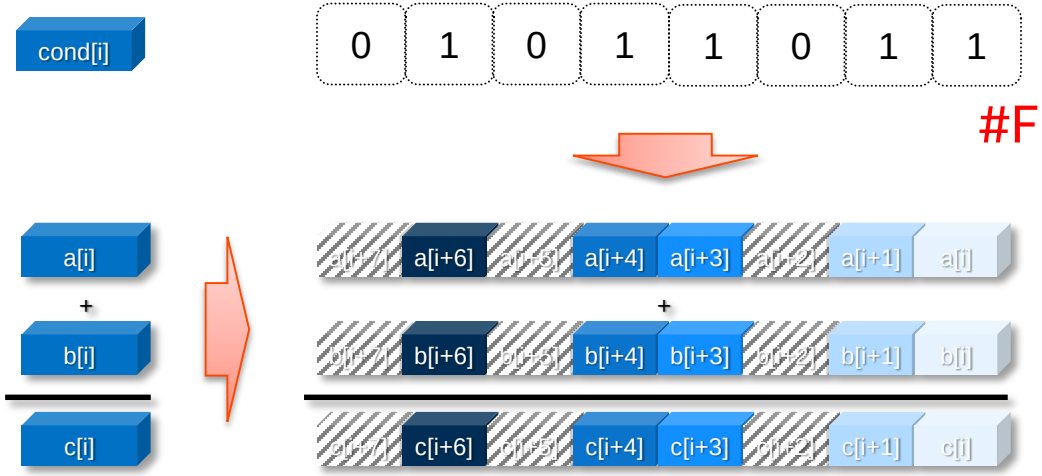
3 elements suppressed

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

Mask Utilization = 5/8

62.5%

#FLOP (MAX = 8): **5**



Advisor Memory Access Pattern (MAP):
know your access pattern

Unit-Stride access

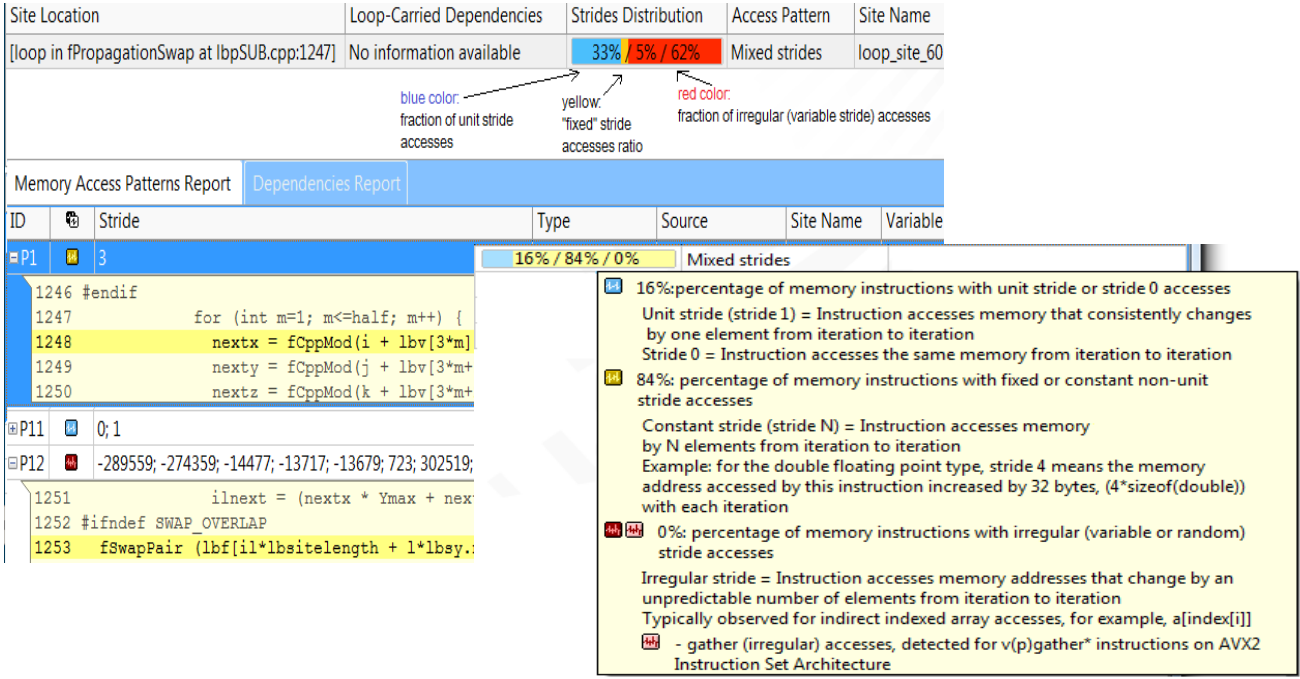
```
for (i=0; i<N; i++)
A[i] = C[i]*D[i]
```

Constant stride access

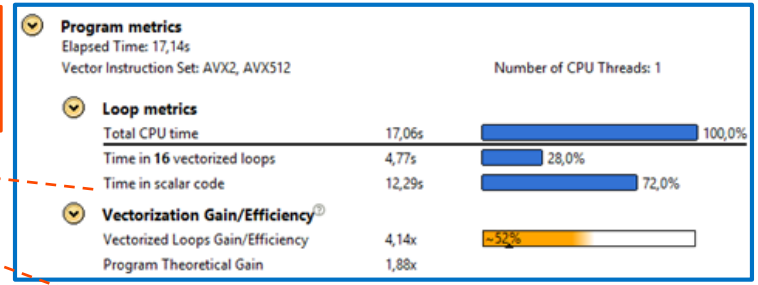
```
for (i=0; i<N; i++)
point[i].x = x[i]
```

Variable stride access

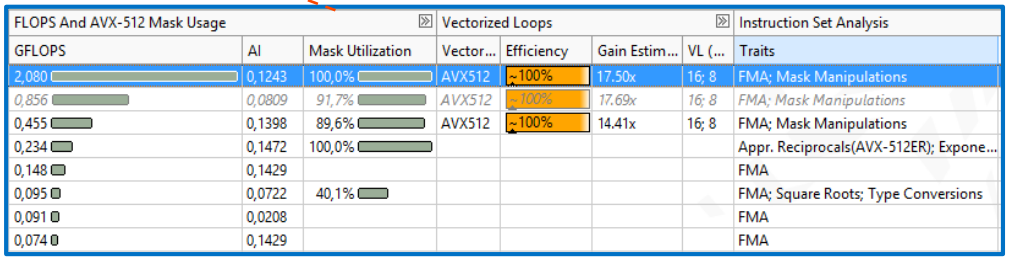
```
for (i=0; i<N; i++)
A[B[i]] = C[i]*D[i]
```

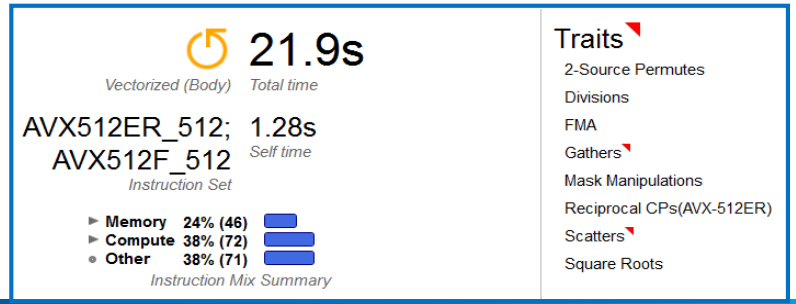


Advisor 2017: **AVX-512** specific performance insights



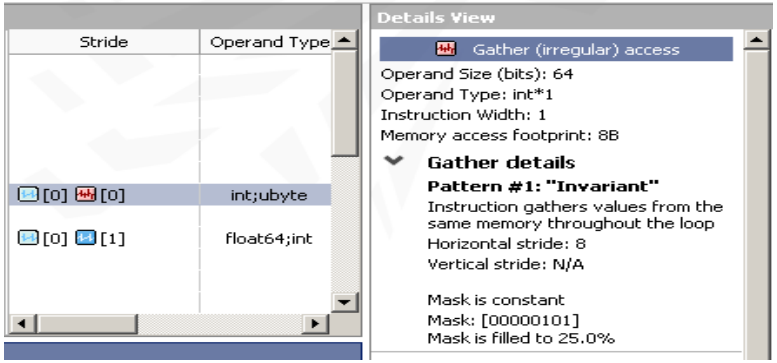
- **Native AVX-512 profiling** on KNL
- Precise **FLOPs and Mask Utilization** profiler
- AVX-512 **Advices and "Traits"**
- And more..
  - Performance **Summary for** AVX-512 codes
  - **AVX-512 Gather/Scatter Profiler**
- No access to AVX-512 Hardware yet?
  - Explore AVX-512 code with –axcode flags and new Advisor Survey capability!

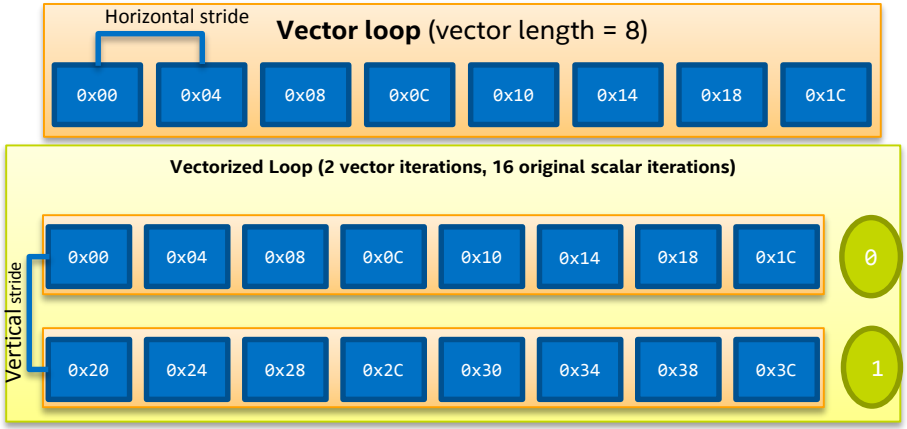




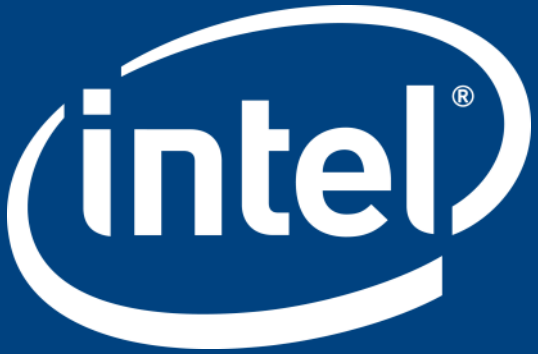
Gather/Scatter Analysis

Advisor MAP detects gather “offset patterns”.





Pattern #	Pattern Name	Horizontal Stride Value	Vertical Stride Value	Example of Corresponding *Fix(es)*
1	Invariant	0	0	OpenMP uniform clause, simd pragma/directive, refactoring
2	Uniform (horizontal invariant)	0	Arbitrary	OpenMP uniform clause, simd pragma/directive
3	Vertical Invariant	Constant	0	OpenMP private clause, simd pragma/directive
4	Unit	1 or -1	|Vertical Stride| = Vector Length	OpenMP linear clause, simd pragma/directive
5	Constant	Constant = X	Constant = X*VectorLength	Subject for AoS -> SoA transformation


intel®