The background of the slide is a blue-toned image of a person's face, rendered in a style where the pixels are composed of binary code (0s and 1s). The person's mouth is open as if shouting or speaking. The overall aesthetic is digital and high-tech.

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

Дмитрий Петунин

# Векторные операции (SIMD)

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



# VECTOR REGISTERS IN INTEL ARCHITECTURES

AVX2  
16 vector  
registers



8x floats



4x doubles

AVX512  
32 vector  
registers



16x floats



8x doubles

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

# Без векторизации

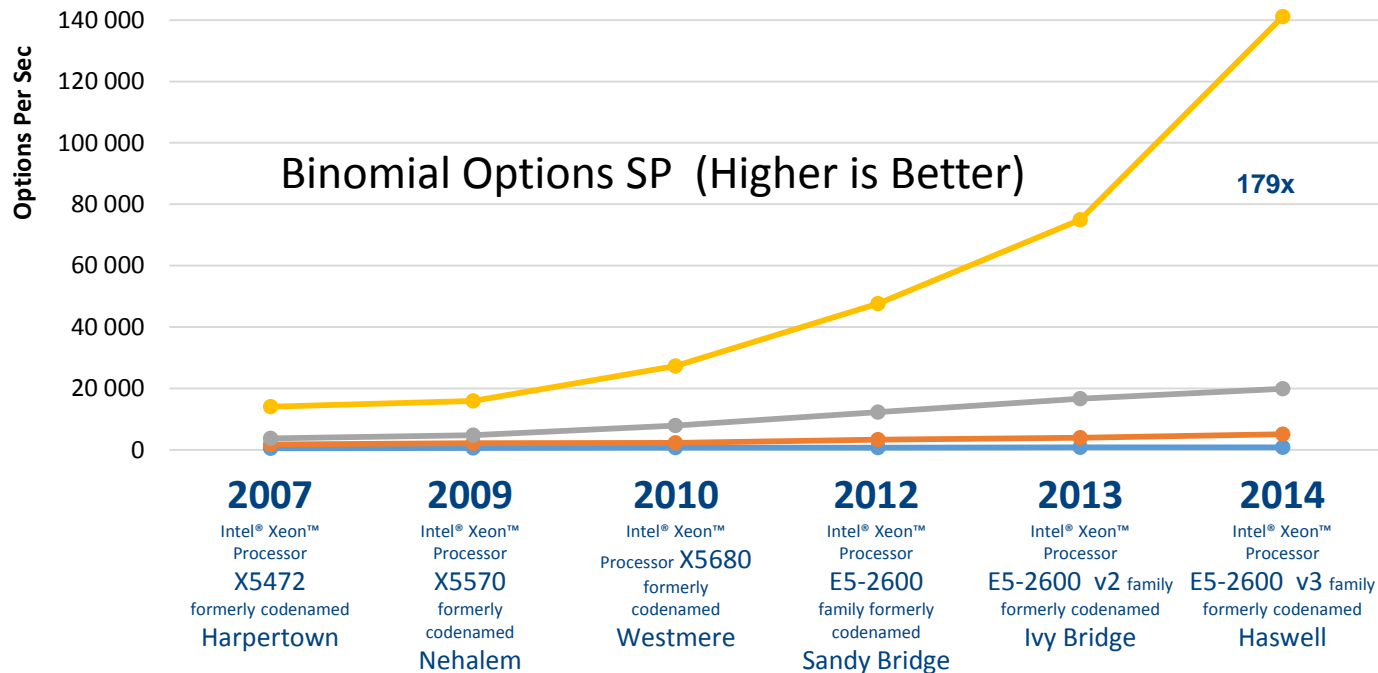


# Используйте весь параллелизм



Intel Advisor XE:	Threading	Vectorization
	✓	✓

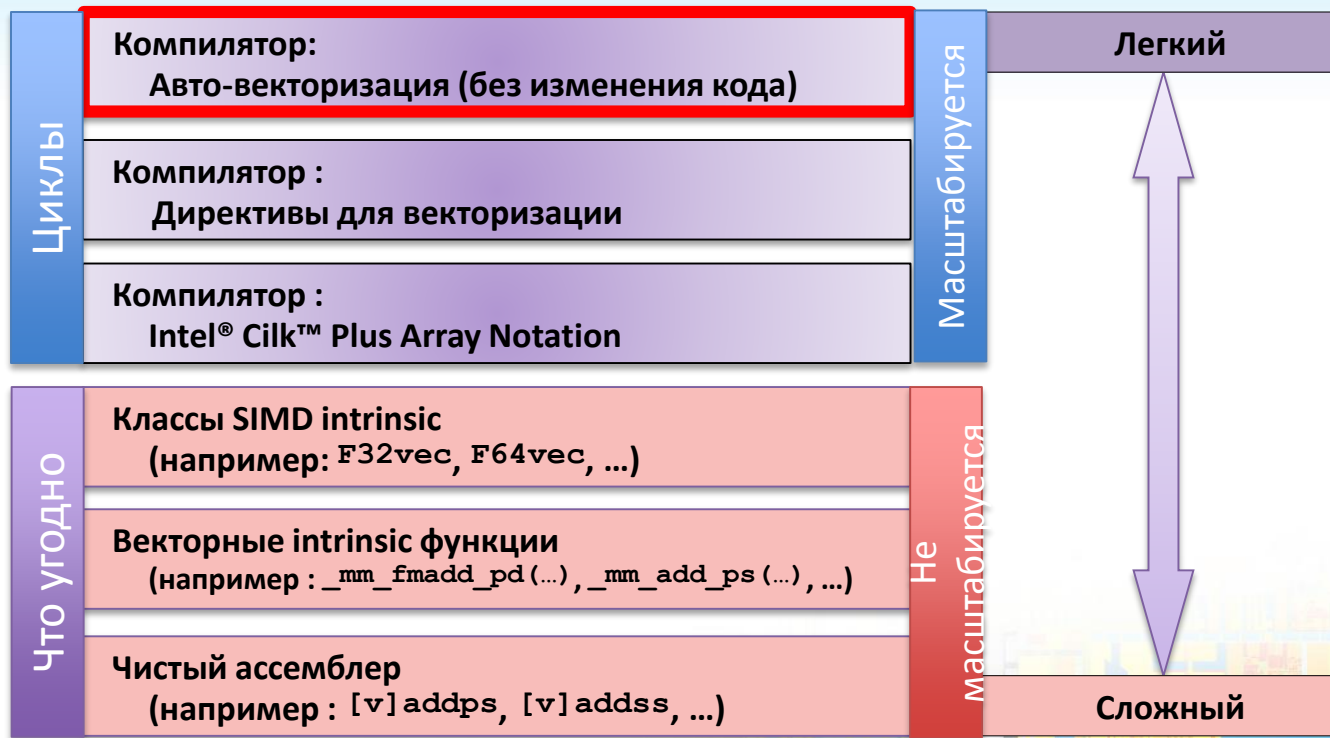
# Каков потенциал?



	<u>Threaded</u>	<u>Vectorized</u>
	✓	✓
	✓	Scalar
Single Thread		✓
Single Thread		Scalar

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to

# 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];
    }
}
```

# 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];
    }
}
```

# 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 Future



# 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 Future



# 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

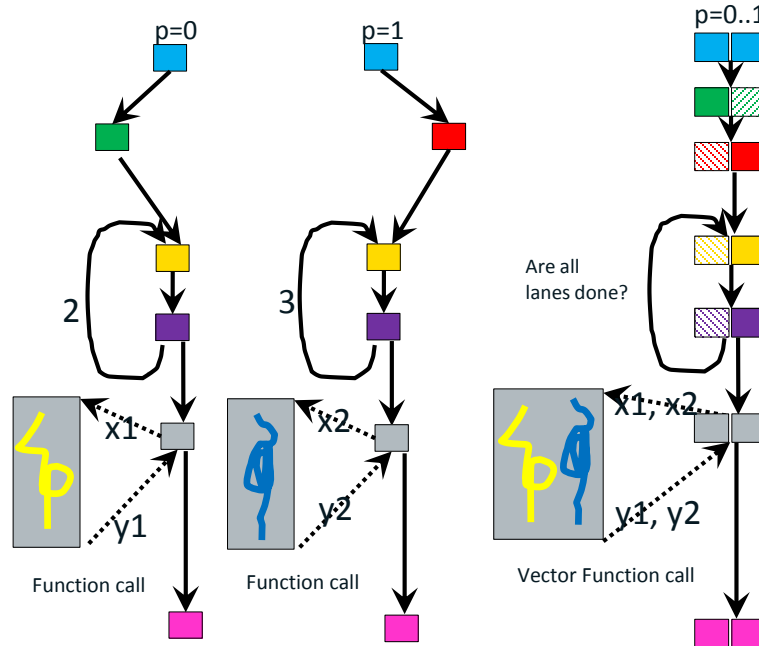
```
for (i=0; i<N; i++)
  A[B[i]] = C[i]*D[i]
```

## 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

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization
			Loop Type Why No Vectorization?
[loop in runCForAllLambdaLoops]	0.094s	0.094s	Scalar vector dependence prevents vector...
[loop in runCForAllLambdaLoops]	0.140s	3.744s	Scalar inner loop was already vectorized
[loop in std::Complex_base<double,struct_C_double_complex>::...]	0.031s	0.031s	Vectorized (Body)

Vectorized SSE: SSE2 loop processing Float32; Float64 data type  
Peeled loop: loop stats were reordered

Function Call Sites and Loops	Self Time	Total Time
[loop in std::basic_string<char,struct_std::char_traits<char>,class_std::allocat...	0.000s	5...
[loop in std::basic_string<char,struct_std::char_traits<char>,class_std::allocat...	0.000s	5...
[loop in std::num_put<char,class_std::ostreambuf_iterator<char,struct_st...	0.000s	5...

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

**Issue: Peeled/Remainder loop(s) present**

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

**Recommendation: Align memory access**  
Projected maximum performance gain: High  
Projection confidence: Medium

...one of the memory accesses in the source loop does not have a byte boundary.

```
SIZE*sizeof(float), 32);
```

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

Total Time	Trip Counts			Iteration Duration	Call Count
	Median	Min	Max		
3.151s	1	1	1	3.1509s	1
0.440s	1	1	1	< 0.0001s	2408000
0.010s	1	1	2	< 0.0001s	207596
0.010s	1	2	1	< 0.0001s	1173619
0.010s	1	3	1	< 0.0001s	1312315

### 4. Loop-Carried Dependency Analysis

**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

### 5. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCrawLoops	runCrawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCrawLoops	runCrawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCrawLoops	runCrawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

**Memory Access Patterns**

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCrawLoops.cxx:637	lcals.exe	
P23	0; 0	Unit stride	runCrawLoops.cxx:638	lcals.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCrawLoops.cxx:628	lcals.exe	

```

635     j2 = ( j2 & 64-1 ) ;
636     p[ip][0] += y[j2+32];
637     p[ip][1] += z[j2+32];
638     i2 += e[j2+32];
639     j2 += f[j2+32];

```

```

626     i1 &= 64-1;
627     j1 &= 64-1;
628     p[ip][2] += b[j1][11];

```

# Диагностика SIMD циклов

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vector...	Efficiency	Vector L...
[loop at stl_algo.h:4740 in std:tr...]		0.170s	0.170s		Scalar	non-vectorizable loop ins ...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...]		0.150s	0.150s		Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s...]		0.020s	0.020s	4	Remainder				
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop at loopstl.cpp:3509 in s2...]	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX	~69%	8
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~98%	8
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at stl_numeric.h:247 in std...]	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve...			

Top Down Source Loop Assembly Assistance Recommendations Compiler Diagnostic Details

File: loopstl.cpp:3509 s273\_

Line	Source	Total Time	%	Loop Time	%
3504	forttime_ (&tl);				
3505	i_1 = *ntimes;				
3506	for (nl = 1; nl <= i_1; ++nl)	0.010s		0.200s	
	[loop at loopstl.cpp:3506 in s273_] Scalar Loop. Not vectorized: inner loop was already vectorized No loop transformations were applied				
3507	{				
3508	i_2 = *n;				
3509	for (i_ = 1; i_ <= i_2; ++i_)	0.010s		0.160s	
	[loop at loopstl.cpp:3509 in s273_] Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Inserts; Extracts; Masked St				
	Selected (Total Time):	0.010s			

# Диагностика SIMD циклов

Intel Advisor XE 2016

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 54.44s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vector...	Efficiency	Vector L...
[loop at stl_algo.h:4740 in std:tr...]		0.170s	0.170s		Scalar	non-vectorizable loop ins ...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...]		0.150s	0.150s		Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s...]		0.020s	0.020s	4	Remainder				
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop at loopstl.cpp:3509 in s2...]	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX	~69%	8
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~98%	8
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at stl_numeric.h:247 in std...]	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve...			

Top Down Source Loop Assembly Assistance Recommendations Compiler Diagnostic Details

File: loopstl.cpp:3509 s273\_

Line	Source	Total Time	%	Loop Time	%
3504	forttime_ (&tl);				
3505	i_1 = *ntimes;				
3506	for (nl = 1; nl <= i_1; ++nl)	0.010s		0.200s	
	[loop at loopstl.cpp:3506 in s273_] Scalar Loop. Not vectorized: inner loop was already vectorized No loop transformations were applied				
3507	{				
3508	i_2 = *n;				
3509	for (i_ = 1; i_ <= i_2; ++i_)	0.010s		0.160s	
	[loop at loopstl.cpp:3509 in s273_] Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Inserts; Extracts; Masked St...				
	Selected (Total Time):	0.010s			

# Диагностика SIMD циклов

ng parallelism? 📷

Intel Advisor XE 2016

on Report 📊 Suitability Report

📁 All Modules 📁 All Sources 🔍

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	<a href="#">Collapse</a>	<a href="#">Collapse</a>	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
<b>0.160s  </b>	<b>0.160s  </b>	<b>12</b>	<b>Expand</b>		<b>AVX</b>	~69%	<b>8</b>
0.150s	0.150s	125; 4	<a href="#">Expand</a>	<a href="#">Expand</a>	AVX	~96%	8
0.150s	0.150s	12	<a href="#">Expand</a>	<a href="#">Expand</a>	AVX	~100%	4
0.150s	0.150s	49	Scalar	❑ vector dependence preve ...			

Скалярный или векторный



# Диагностика SIMD циклов

ng parallelism? 📷 Intel Advisor XE 2016

on Report 📊 Suitability Report

📄 All Modules 📄 All Sources 🔍

Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization:	Efficiency	Vector L...
0.170s	0.170s		Scalar	non-vectorizable loop ins ...		
0.170s	0.170s	12; 4	<a href="#">Collapse</a>	<a href="#">Collapse</a>	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
<b>0.160s</b>	<b>0.160s</b>	<b>12</b>	<b>Expand</b>	<b>Expand</b>	AVX ~69%	<b>8</b>
0.150s	0.150s	125; 4	<a href="#">Expand</a>	<a href="#">Expand</a>	AVX ~96%	8
0.150s	0.150s	12	<a href="#">Expand</a>	<a href="#">Expand</a>	AVX ~100%	4
0.150s	0.150s	49	Scalar	vector dependence preve ...		

Тип инструкций

Скалярный или векторный

# Диагностика SIMD циклов

ng parallelism? 📷 Intel Advisor XE 2016

on Report 📊 Suitability Report

All Modules All Sources

Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization:	Efficiency	Vector Length
0.170s	0.170s		Scalar	non-vectorizable loop ins ...		
0.170s	0.170s	12; 4	<a href="#">Collapse</a>	<a href="#">Collapse</a>	~100%	4
0.150s	0.150s	12	Vectorized (Body)			4
0.020s	0.020s	4	Remainder			
0.170s	0.170s	500	Scalar	but ...		4
<b>0.160s</b>	<b>0.160s</b>	<b>12</b>	<b>Expand</b>		~69%	<b>8</b>
0.150s	0.150s	125; 4	<a href="#">Expand</a>	<a href="#">Expand</a>	~96%	8
0.150s	0.150s	12	<a href="#">Expand</a>	<a href="#">Expand</a>	~100%	4
0.150s	0.150s	49	Scalar	vector dependence preve ...		

Тип инструкций

Длина вектора

Скалярный или векторный

# Диагностика SIMD циклов

ng parallelism?

on Report Suitability Report

Intel Advisor XE 2016

All Modules All Sources

Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization:	Efficiency	Vector Length
0.170s	0.170s		Scalar	<input checked="" type="checkbox"/> non-vectorizable loop ins ...		
0.170s	0.170s	12; 4	<a href="#">Collapse</a>	<a href="#">Collapse</a>	~100%	4
		12	Vectorized (Body)			4
		4	Remainder			
0.170s	0.170s	500	Scalar	but...		4
<b>0.160s</b>	<b>0.160s</b>	<b>12</b>	<b>Expand</b>	<b>Expand</b>	~69%	<b>8</b>
0.150s	0.150s	125; 4	<a href="#">Expand</a>	<a href="#">Expand</a>	~96%	8
0.150s	0.150s	12	<a href="#">Expand</a>	<a href="#">Expand</a>	~100%	4
0.150s	0.150s	49	Scalar	<input checked="" type="checkbox"/> vector dependence preve ...		

Тип инструкций

Длина вектора

Количество итераций

Скалярный или векторный







# Диагностика SIMD циклов

ng parallelism?

on Report Suitability Report

Intel Advisor XE 2016

All Sources

Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization:	Efficiency	Vector Length
0.170s	0.170s		Scalar	<input checked="" type="checkbox"/> non-vectorizable loop ins ...		
0.170s	0.170s	12; 4	<a href="#">Collapse</a>	<a href="#">Collapse</a>		4
		12	Vectorized (Body)			4
		4	Remainder			
0.170s	0.170s	500	Scalar	but...		4
<b>0.160s</b>	<b>0.160s</b>	<b>12</b>	<b>Expand</b>	<b>Expand</b>		<b>8</b>
0.150s	0.150s	125; 4	<a href="#">Expand</a>	<a href="#">Expand</a>		8
0.150s	0.150s	12	<a href="#">Expand</a>	<a href="#">Expand</a>		4
0.150s	0.150s	49	Scalar	<input checked="" type="checkbox"/> vector dependence preve ...		

Время ЦПУ

Тип инструкций

Длина вектора

Количество  
итераций

Скалярный или  
векторный



# Диагностики в исходном коде

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

Line	Source	Total Time	%
163	<pre>for (int x = x0; x &lt; x1; ++x) {</pre>		
	<p>[loop at fractal.cpp:163 in &lt;lambda1&gt;::operator()] Scalar Loop. Not vectorized: outer loop was not auto-vectorized: consider us No loop transformations were applied</p>		
164	<pre>for (int y = y0; y &lt; y1; ++y) {</pre>		
	<p>[loop at fractal.cpp:164 in &lt;lambda1&gt;::operator()] Scalar Loop. Not vectorized: vectorization possible but seems inefficient. Us Loop was unrolled by 2</p>		
165	<pre>fractal_data_array[x - x0][y - y0] = calc_one_pixel(x, y, t</pre>	10.822s	<input type="checkbox"/>
166	<pre>}</pre>		
167	<pre>}</pre>		
168	<pre>for (int y = y0, y_temp = 0; y &lt; y1; ++y, ++y_temp) {</pre>		
169	<pre>area.set_pos(0, y - y0);</pre>		
170	<pre>for (int x = x0, x_temp = 0; x &lt; x1; ++x, ++x_temp) {</pre>		
171	<pre>area.put_pixel(fractal_data_array[x_temp][y_temp]);</pre>		
172	<pre>}</pre>		
173	<pre>}</pre>	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
i> [loop at lbpBGK.cpp:840 in fCollisionBGK]	AVX	100%	2,05	2



# Части векторного цикла

**Ad** Where should I add vectorization and/or threading parallelism? 📷

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 8,52s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	🔥	💡 Vector Issues	Self Time	Total Time	Loop Type
🔍 [loop at fractal.cpp:179 in <lambda1>::op ...		💡 4 High vector ...	0,013s	12,020s <span style="width: 100px; height: 10px; background-color: #0070C0; display: inline-block;"></span>	<a href="#">Collapse</a>
↳ [loop at fractal.cpp:179 in <lambda1>::o ...	☑	💡 4 Serialized use ...	0,013s	11,281s <span style="width: 100px; height: 10px; background-color: #0070C0; display: inline-block;"></span>	Vectorized (Body)
↳ [loop at fractal.cpp:179 in <lambda1>::o ...	☑	💡 2 Data type co ...	0,000s	0,163s	Peeled
↳ [loop at fractal.cpp:179 in <lambda1>::o ...	☑	💡 2 Data type co ...	0,000s	0,576s	Remainder
↳ [loop at fractal.cpp:177 in <lambda1>::oper ...	☐	💡 2 Data type co ...	0,010s	12,030s <span style="width: 100px; height: 10px; background-color: #0070C0; display: inline-block;"></span>	Scalar

# Число итераций

« Ad Where should I add vectorization and/or threading parallelism? 📷

🌳 Summary 🌱 Survey Report 🍎 Refinement Reports 💧 Annotation Report 🌱 Suitability Report

Program time: 12.82s Vectorized Not Vectorized FILTER: All Modules

Function Call Sites and Loops	Self Time	Total Time	🔥	💡	Trip Counts			
					Median	Min	Max	Call Count
📄 [loop at Multiply.c:53 in matvec]	11.898s	11.898s						
↳ [loop at Multiply.c:53 in matvec]	11.851s	11.851s	<input type="checkbox"/>			101	101	12000000
↳ [loop at Multiply.c:53 in matvec]	0.047s	0.047s	<input type="checkbox"/>			3	3	1000000
↳ [loop at Multiply.c:53 in matvec]	0.413s	0.413s	<input type="checkbox"/>		101	101	101	2000000
📄 [loop at Multiply.c:45 in matvec]	0.109s	12.373s		💡 1				
↳ [loop at Driver.c:146 in main]	0.016s	12.483s	<input type="checkbox"/>	💡 1	1000000	1000000	1000000	1

Число ВЫЗОВОВ

Число итераций



# 1. Диагностика SIMD

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization	
			Loop Type	Why No Vectorization?
[loop in runCforallLambdaLoop]	0.094s	0.094s	Scalar	vector dependence prevents vector...
[loop in runCforallLambdaLoop]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::complex<bool, double, struct_C_double_complex>::...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations				
Peeled loop; loop starts were reconsidered				
[loop in std::basic_string<char, struct_std_char_traits, char, class std::allo...	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char, struct_std_char_traits, char, class std::allo...	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char, class std::ostreambuf_iterator<char, struct.st...	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

# 2. Рекомендации



## Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

### Recommendation: Align memory access

Projected maximum performance gain: High  
Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
_assume_aligned(array, 32);
// Use array in loop
```

# Рекомендации

Elapsed time: 8,81s   Vectorized   Not Vectorized   FILTER: All Modules   All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
						Vecto...	Estim...	Vector Len
i> [loop at market.cpp:476 in tb...			1,460s	Scalar				
i> [loop at arena.cpp:88 in tbb::tbb::]		0,000s	11,460s	Scalar				
<b>[5] [loop at fractal.cpp:179 in &lt;lambda1&gt;::op ...</b>	<b>5 Ineffective ...</b>	0,000s	2,022s	<b>Collapse</b>	<b>Collapse</b>			
i> [loop at fractal.cpp:179 in <lambda1>::o ...	2 Data type co ...	0,000s	2,022s	Remainder				

Top Down   Source   Loop Assembly   Assistance   Recommendations   Compiler Diagnostic Details

**Issue: 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 to the loop body.

Disable unrolling  
The [trip count](#) after loop unrolling is too small compared to the [vector length](#). To fix: Prevent loop [unrolling](#) or decrease the unroll factor using a [directive](#).

ICL/ICC/ICPC Directive	IFORT Directive
#pragma nounroll	!DIR\$ NOUNROLL
#pragma unroll	!DIR\$ UNROLL

Read More:

- [User and Reference Guide for the Intel C++ Compiler 15.0 > Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll.](#)

# Рекомендации

Elapsed time: 8,81s   Vectorized   Not Vectorized   FILTER: All Modules   All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
						Vecto...	Estim...	Vector Len
[loop at market.cpp:476 in tb...			1,460s	Scalar				
[loop at arena.cpp:88 in tbb::tbb::]		0,000s	11,460s	Scalar				
[loop at fractal.cpp:179 in <lambda1>::op ...]	5 Ineffective ...	0,000s	2,022s	Collapse	Collapse			
[loop at fractal.cpp:179 in <lambda>::o ...]	2 Data type co ...	0,000s	2,022s	Remainder				

Найдены "Vector Issues"

Top Down   Source   Loop Assembly   Assistance   Recommendations   Compiler Diagnostic Details

**Issue: 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 to the loop body.

Disable unrolling  
The [trip count](#) after loop unrolling is too small compared to the [unroll factor](#) or decrease the unroll factor using a [directive](#).

ICL/ICC/ICPC Directive	IFORT Directive
#pragma nounroll	!DIR\$ NOUNROLL
#pragma unroll	!DIR\$ UNROLL

Read More:

- [User and Reference Guide for the Intel C++ Compiler 15.0 > Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll.](#)

Подробное описание типичных проблем

# Рекомендации

Top Down Source Loop Assembly Assistance Recommendations Compiler Diagnostic Details

## 2 Issue: Serialized user function call(s) present

User-defined functions in the [loop body](#) are not vectorized.

### 2 Enable inline expansion

Inlining of user-defined functions is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

Windows* OS		Linux* OS	
<a href="#">ICL Option</a>	<a href="#">IFORT Option</a>	<a href="#">ICC/ICPC Option</a>	<a href="#">IFORT Option</a>
<code>/Ob1</code> or <code>/Ob2</code>	<code>Ob1</code> or <code>Ob2</code>	<code>-inline-level=1</code> or <code>-inline-level=2</code>	<code>-inline-level=1</code> or <code>-inline-level=2</code>

**Read More:**

# Рекомендации

Top Down Source Loop Assembly Assistance Recommendations Compiler Diagnostic Details

**Issue: Serialized user function call(s) present**  
User-defined functions in the [loop body](#) are not vectorized.

2

2

Enable inline expansion

Inlining of user-defined functions is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

Windows* OS		Linux* OS	
ICL Option	IFORT Option	ICC/ICPC Option	IFORT Option
<code>/Ob1</code> or <code>/Ob2</code>	<code>Ob1</code> or <code>Ob2</code>	<code>-inline-level=1</code> or <code>-inline-level=2</code>	<code>-inline-level=1</code> or <code>-inline-level=2</code>

Read More:

Source Top Down Loop Assembly Recommendations Compiler Diagnostic Details

**Issue: Inefficient memory access patterns present**

There is a high of percentage memory instructions with irregular (variable or random) stride accesses. Improve performance by investigating an

Recommendation: Use SoA instead of AoS

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. Structures (AoS) or as a structure of arrays (SoA). While AoS organization is excellent for encapsulation, it can hinder effective vector processing using SoA instead of AoS.

Read More:

## 1. Диагностика SIMD циклов

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization	
			Loop Type	Why No Vectorization?
[loop in runCForsAllambdaLoop]	0.094s	0.094s	Scalar	vector dependence prevents vector...
[loop in runCForsAllambdaLoop]	0.140s	3.744s	Scalar	inner loop was already vectorized...
[loop in std::complex<bool, double, struct C_double_complex>::...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop starts were reinserted				
[loop in std::basic_string<char, struct std::char_traits<char>, class std::allo...	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char, struct std::char_traits<char>, class std::allo...	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::min<char, class std::ostreambuf_iterator<char, struct st...	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

## 2. Рекомендации

**Issue: Peeled/Remainder loops present**

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

**Recommendation: Align memory access**

Projected maximum performance gain: High  
Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary.

```
Float *array;  
array = (Float *)_mm_malloc(ARRAY_SIZE*sizeof(Float), 32);  
  
// Somewhere else  
__assume_aligned(array, 32);  
// Use array in loop
```

## 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
<code>#pragma simd</code> or <code>#pragma omp simd</code>	<code>!DIR\$ SIMD</code> or <code>!\$OMP SIMD</code>	Ignores all dependencies in the loop
<code>#pragma ivdep</code>	<code>!DIR\$ IVDEP</code>	Ignores only vector dependencies (which is safest)

**Read More:**

# Анализ зависимостей

Intel Advisor XE 2016

Where should I add vectorization and/or threading parallelism?

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Program time: 12.82s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Self Time	Total Time	🔥	💡	Trip Counts	Compiler Vectorization	
						Loop Type	Why No Vectorization?
↳ [loop at Multiply.c:53 in matvec]	0.047s	0.047s	<input type="checkbox"/>		3	Vectorized (Body)	
↳ [loop at Multiply.c:53 in matvec]	0.413s	0.413s	<input type="checkbox"/>		101	Scalar	
↳ [loop at Multiply.c:45 in matvec]	0.109s	12.373s	<input type="checkbox"/>	💡 1		<a href="#">Collapse</a>	<a href="#">Collapse</a>
↳ [loop at Multiply.c:45 in matvec]	0.078s	11.930s	<input type="checkbox"/>		12	Vectorized (Body)	
↳ [loop at Multiply.c:45 in matvec]	0.031s	0.444s	<input type="checkbox"/>		2	Remainder	
↳ [loop at Driver.c:146 in main]	0.016s	12.483s	<input checked="" type="checkbox"/>	💡 1	1000000	Scalar	vector dependence prevents vectoriza ...

## 2.1 Check Correctness

Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.



Command Line

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
<pre>20 k += a[9]; 21 k *= a[8]; 22 k -= a[7]; 23 k += a[6]; 24 k *= a[5];</pre>					
X18	Read	main.cpp:23	main	test_1.exe	🚫 New
<pre>21 k *= a[8]; 22 k -= a[7]; 23 k += a[6];</pre>					

## 1. Диагностика SIMD циклов

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization	
			Loop Type	Why No Vectorization?
[[loop in runCForAllLambdaLoop]	0.094s	0.094s	Scalar	vector dependence prevents vector...
[[loop in runCForAllLambdaLoop]	0.140s	3.744s	Scalar	inner loop was already vectorized...
[[loop in std::complex<bool, double, struct...>.Cdouble.complex.c2...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations				
Peeled loop; loop starts were recompiled				
[[loop in std::basic_string<char, struct std::char_traits<char>, class std::allo...	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[[loop in std::basic_string<char, struct std::char_traits<char>, class std::allo...	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[[loop in std::num_put<char, class std::ostreambuf_iterator<char, struct st...	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

## 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, idl.h	dqtest2	🔴 New

## 2. Рекомендации

### Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

#### Recommendation: Align memory access

Projected maximum performance gain: High  
Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary.

```
Float *array;  
array = (Float *)_mm_malloc(ARRAY_SIZE*sizeof(Float), 32);  
  
// Somewhere else  
__assume_aligned(array, 32);  
// Use array in loop
```

## 4. Анализ доступов к памяти

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cox1063	RAW1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cox622	No information available	39% / 96% / 23%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cox925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns		Correctness Report			
ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.cox637	lcal.exe	
<pre>635 j2 = ( j2 + 64 - 1 ) ; 636 p[4p][0] += y[12+32]; 637 p[4p][1] += z[12+32]; 638 12 += e[12+32]; 639 j2 += f[32+32];</pre>					
P23	0; 0	Unit stride	runCRawLoops.cox638	lcal.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cox628	lcal.exe	
<pre>626 11 += 64 - 1; 627 11 += 64 - 1; 628 p[4p][2] += b[j1][11];</pre>					

# 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]
```



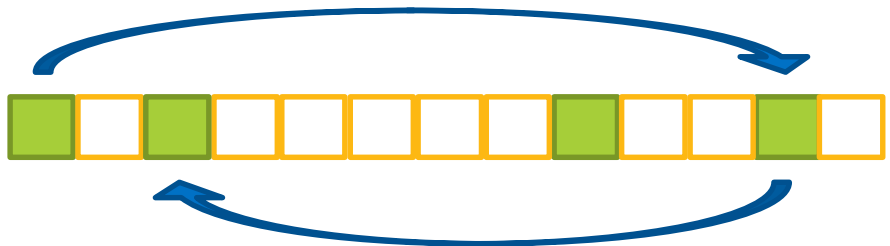
## Constant stride access

```
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

**Where should I add vectorization and/or threading parallelism?**

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 8,52s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...	4 High vector ...	0,013s	12,020s	Collapse	Collapse
i> [loop at fractal.cpp:179 in <lambda1>::o ...	4 Serialized use ...	0,013s	11,281s	Vectorized (Body)	
i> [loop at fractal.cpp:179 in <lambda1>::o ...					
i> [loop at fractal.cpp:179 in <lambda1>::o ...					
i> [loop at fractal.cpp:177 in <lambda1>::oper ...					

Выделяем интересующие нас циклы

**2.2 Check Memory Access Patterns**  
Identify and explore complex memory accesses for marked loops. Fix the reported problems.

Command Line

Запускаем анализ шаблонов доступа

# Шаблоны доступа

Unit/Constant/Variable

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_133	grid_intersect	grid.cpp:559	No information available	23% / 1% / 76%	Mixed strides
loop_site_131	grid_intersect	grid.cpp:581	No information available	21% / 4% / 75%	Mixed strides
loop_site_145	grid_intersect	grid.cpp:562	No information available	21% / 4% / 75%	Mixed strides
loop_site_135	initialize_2D_buffer	find_hotspots.cpp:92	No information available	42% / 0% / 58%	Mixed strides

Memory Access Patterns Report    Correctness Report

ID	Icon	Stride	Type	Source	Modules	Alignment
P17	📄	8	Constant stride	intersect.cpp:141	find_hotspots.exe	
P19	📄	8	Constant stride	intersect.cpp:141	find_hotspots.exe	

```
139
140  intstruct->num++;
141  intstruct->list[intstruct->num].obj = obj;
142  intstruct->list[intstruct->num].t = t;
143 }
```

Constant stride, "Array of Structures"

P1.	📄	0	Unit stride	intersect.cpp:141	find_hotspots.exe	
P1.	📄	0	Unit stride	intersect.cpp:141	find_hotspots.exe	
P2.	📄	-8; -2; 0; 1; 2; ...	Variable stride	intersect.cpp:141	find_hotspots.exe	
P2.	📄	-8; -2; 0; 1; 2; ...	Variable stride	intersect.cpp:141	find_hotspots.exe	
P21	📄	4	Constant stride	intersect.cpp:142	find_hotspots.exe	

Unit stride access

Variable or random access

# 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>

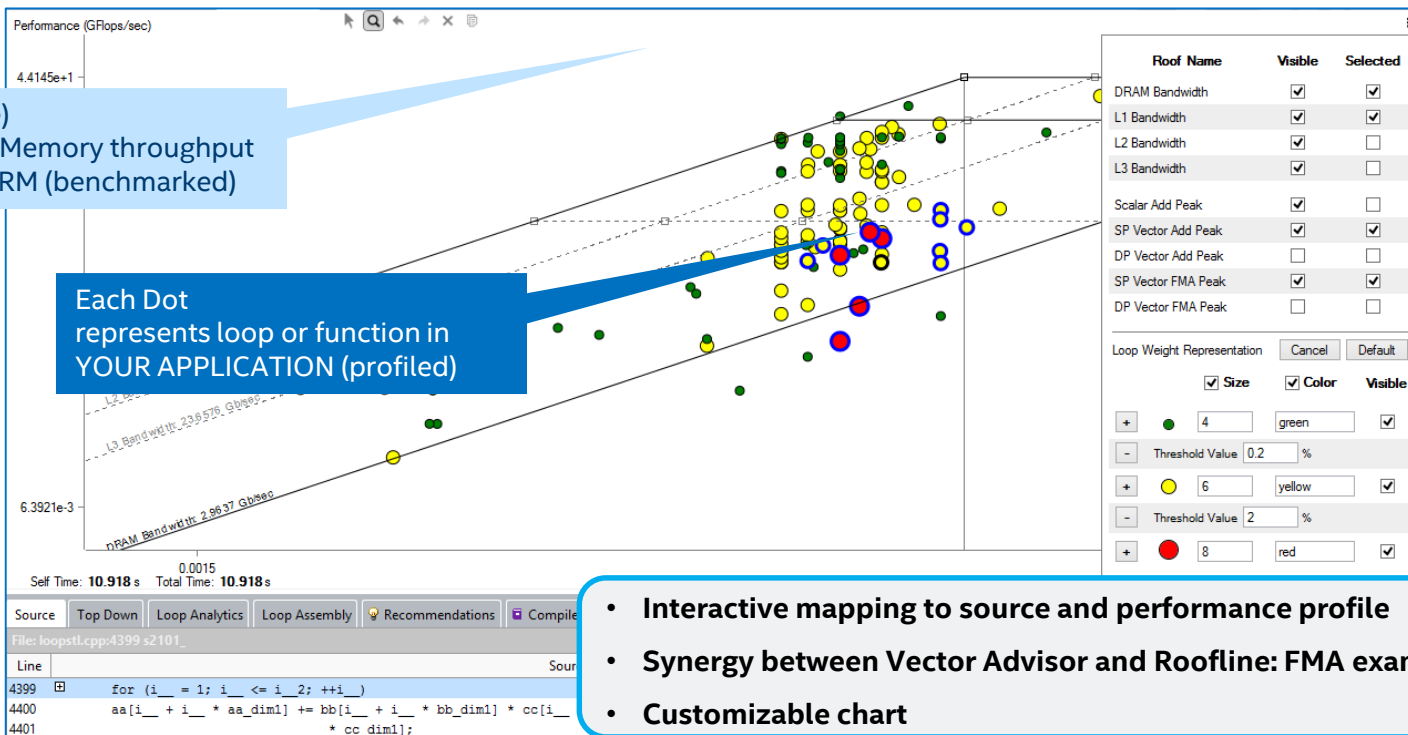
At Intel:

**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)

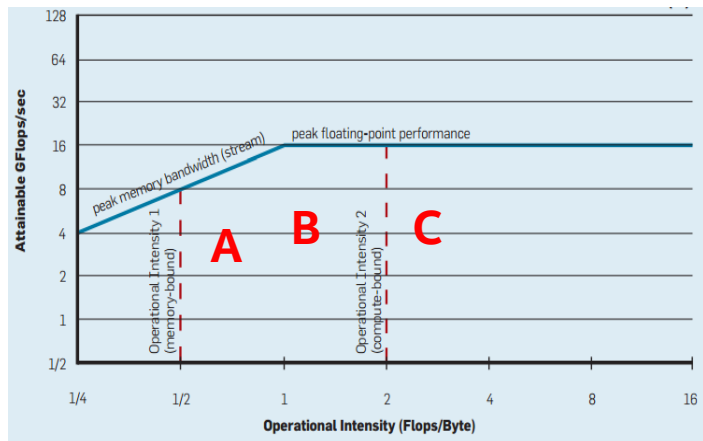


- Interactive mapping to source and performance profile
- Synergy between Vector Advisor and Roofline: FMA example
- Customizable chart

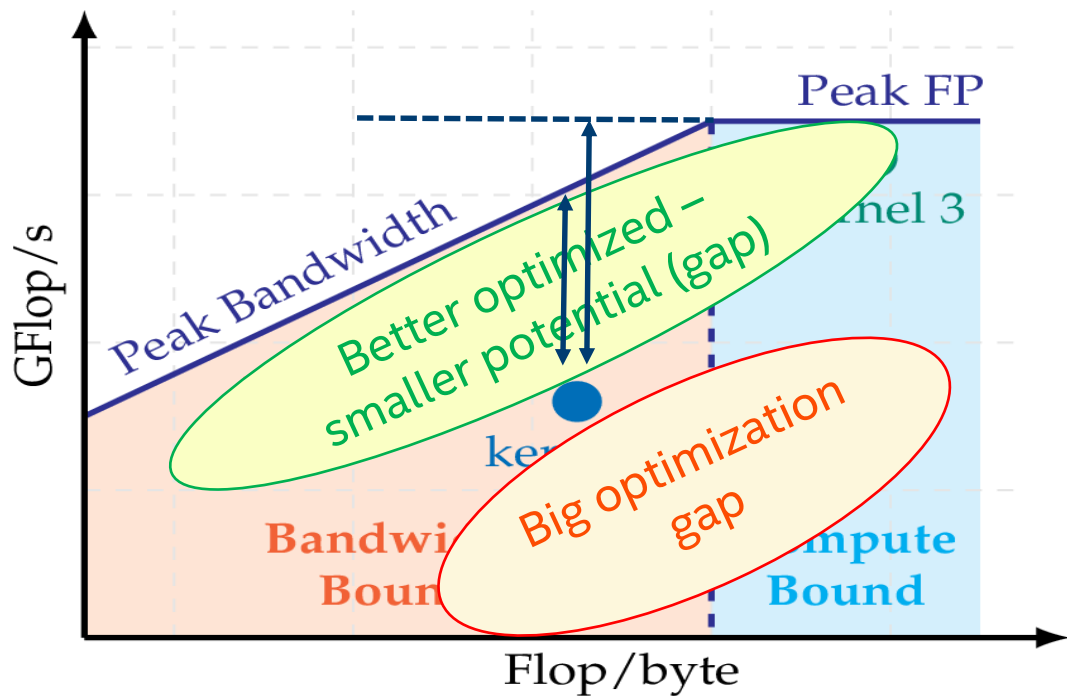
## Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.  
\*Other names and brands may be claimed as the property of others.

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

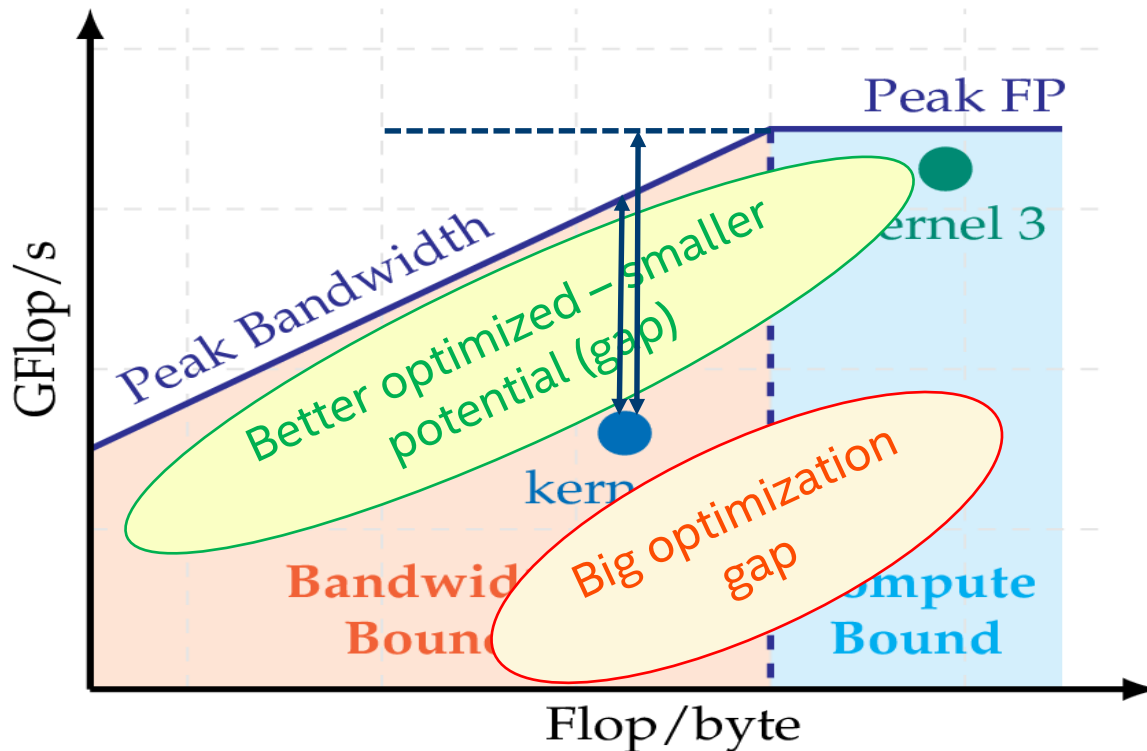


What makes loops  
**A, B, C** different?



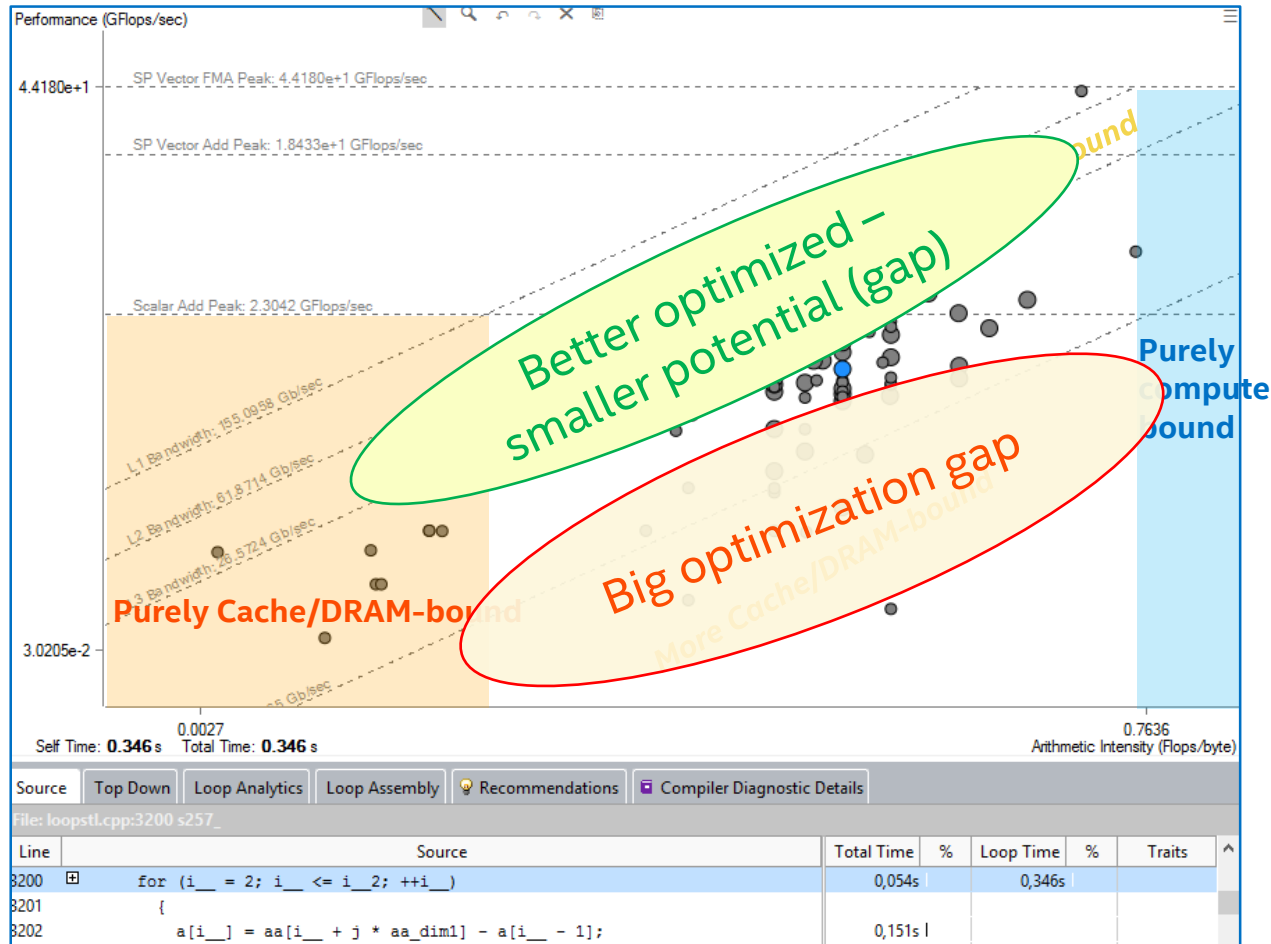
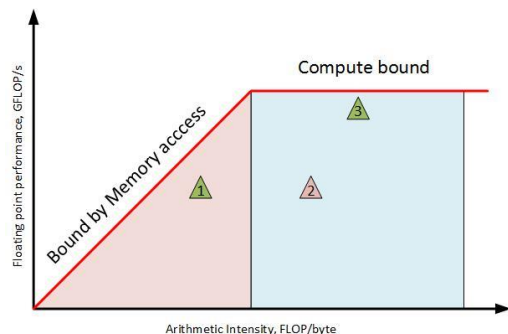
# Am I bound by VPU/CPU or by Memory?

## ROOFLINE ANALYSIS



# Advisor “Cache-aware” Roofline

$$AI = \#FLOP/\#Byte$$



## Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.  
\*Other names and brands may be claimed as the property of others.



# Cache-Aware vs. Classic Roofline

$$AI = \# \text{ FLOP} / \# \text{ BYTE}$$

AI\_DRAM =

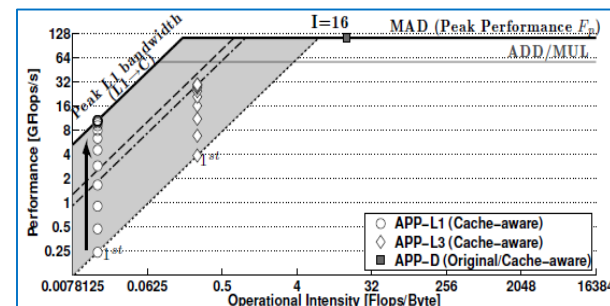
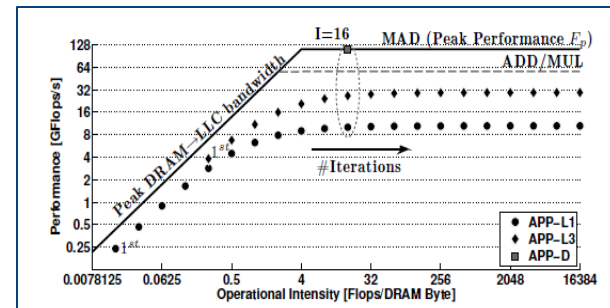
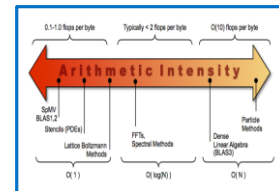
# FLOP / # BYTES (CPU & Cache  $\leftrightarrow$  DRAM)

- “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  $\leftrightarrow$  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:

$FLOP/S = \#FLOP \text{ (AVX-512 mask aware)} / \#Seconds.$

$AI = \#FLOP / \#Bytes$

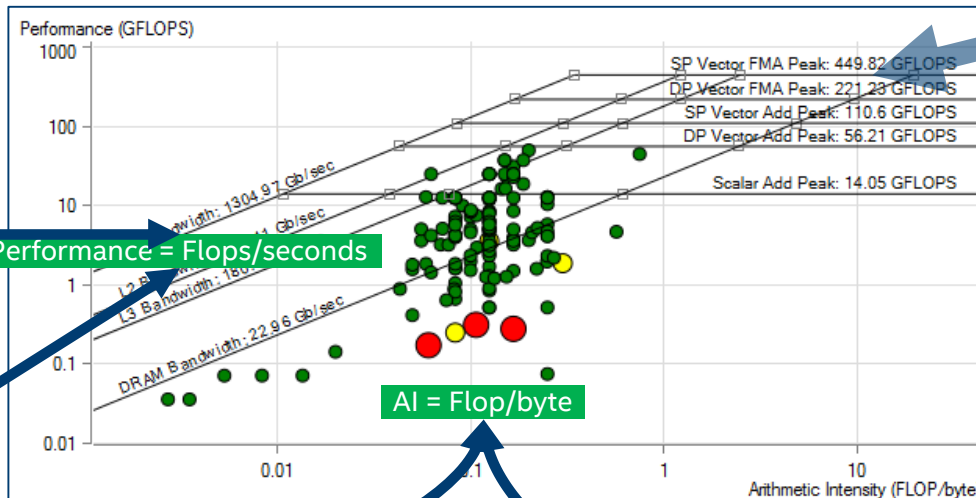
## Seconds

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)

### Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.  
\*Other names and brands may be claimed as the property of others.

# 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					Instruction Set Analysis		
				GFLOPS	AI	Mask Utilization	Vector ISA	Efficiency	Gain Esti...	VL	Traits
[loop in s2711 at loops90.f:16...]		0,010s	Vectorized (Remainder)	0,800	0,1000	100%	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; I
[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 ...]				0,840	0,0833	78%	AVX512	~19%	8.97x	16	2-Source Permutes; Gather

Lower Arithmetic Intensity -> Lower FLOPS and Efficiencies

## Optimization Notice



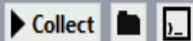
# Getting FLOP/S in Advisor

<b>FLOP/S</b> = #FLOP/Seconds	<b>Seconds</b>	<b>#FLOP</b> - Mask Utilization - #Bytes
<b>Step 1: Survey</b> <ul style="list-style-type: none"><li>- Non intrusive. <i>Representative</i></li><li>- Output: Seconds (+much more)</li></ul>		
<b>Step 2: Trip counts+FLOPS</b> <ul style="list-style-type: none"><li>- Precise, instrumentation based</li><li>- Physically count Num-Instructions</li><li>- Output: #FLOP, #Bytes</li></ul>		

## 1. Survey Target <sup>?</sup>



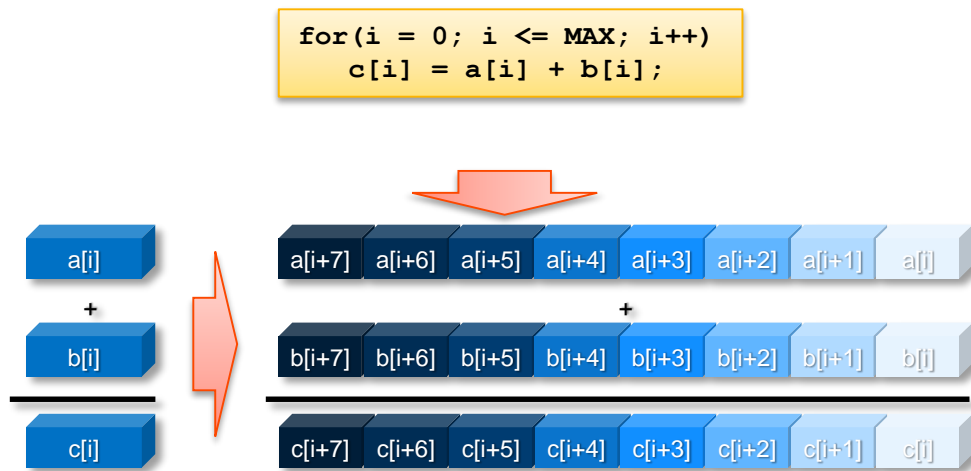
## 1.1 Find Trip Counts and FLOPS <sup>?</sup>



# Why Mask Utilization is Important?

100%

#FLOP (MAX = 8): 8



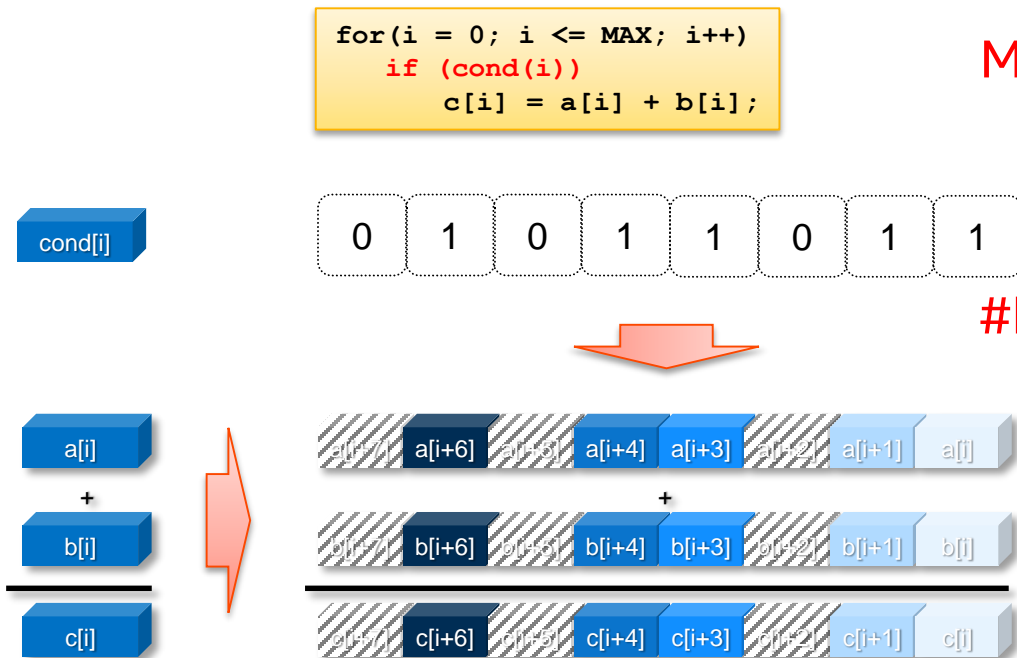
# Why Mask Utilization Important?

3 elements suppressed

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]
```

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in fPropagationSwap at lbpSUB.cpp:1247]	No information available	33% / 5% / 62%	Mixed strides	loop_site_60

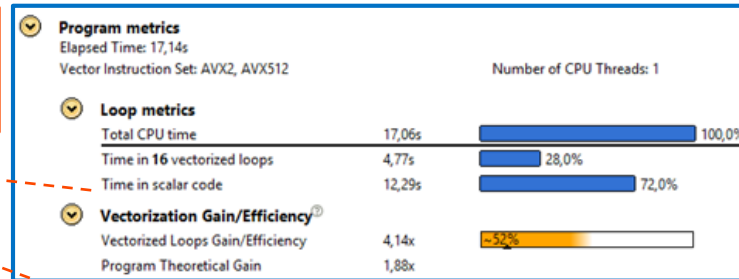
blue color: fraction of unit stride accesses  
 yellow: "fixed" stride accesses ratio  
 red color: fraction of irregular (variable stride) accesses

ID	Stride	Type	Source	Site Name	Variable
P1	3	16% / 84% / 0%	Mixed strides		
<pre> 1246 #endif 1247     for (int m=1; m&lt;=half; m++) { 1248         nextx = fCppMod(i + lbv[3*m] 1249         nexty = fCppMod(j + lbv[3*m+ 1250         nextz = fCppMod(k + lbv[3*m+           </pre>					
P11	0; 1				
P12	-289559; -274359; -14477; -13717; -13679; 723; 302519;				
<pre> 1251         ilnext = (nextx * Ymax + nex 1252 #ifndef SWAP_OVERLAP 1253         f\$swapPair (lbf[i]*lbsitelength + 1*lbsy.           </pre>					

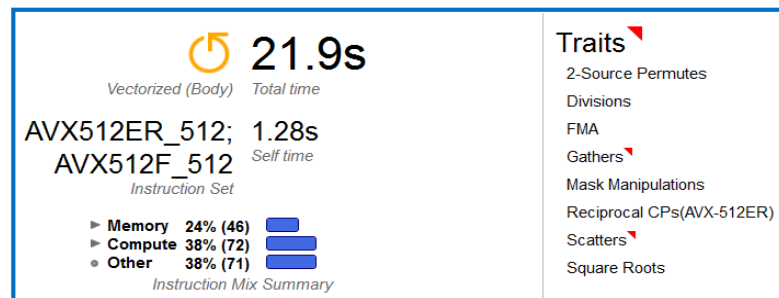
- 16%: percentage of memory instructions with unit stride or stride 0 accesses  
Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration  
Stride 0 = Instruction accesses the same memory from iteration to iteration
- 84%: percentage of memory instructions with fixed or constant non-unit stride accesses  
Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration  
Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4\*sizeof(double)) with each iteration
- 0%: percentage of memory instructions with irregular (variable or random) stride accesses  
Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration  
Typically observed for indirect indexed array accesses, for example, a[index[i]]
- gather (irregular) accesses, detected for v(p)gather\* instructions on AVX2 Instruction Set Architecture

# 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!



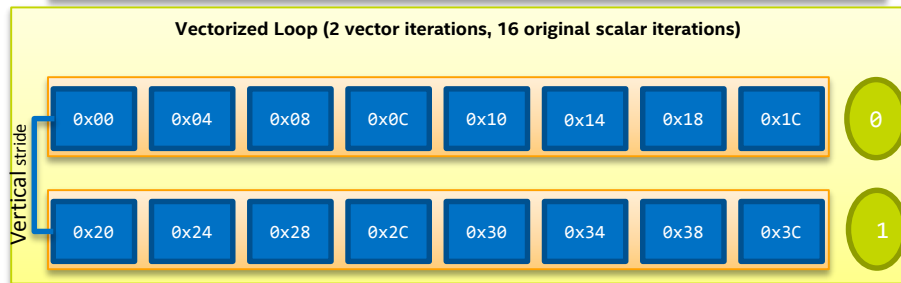
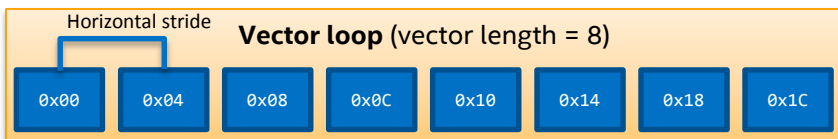
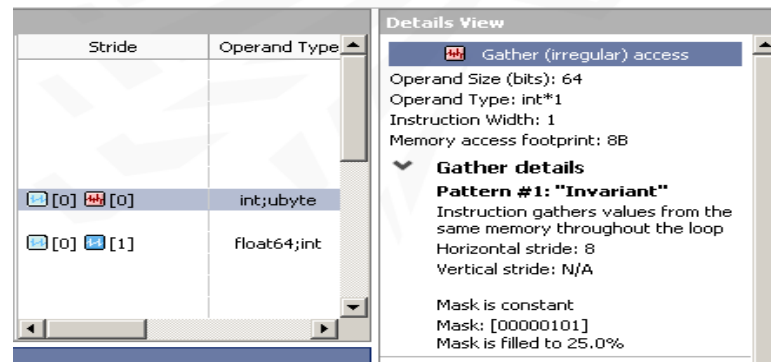
FLOPS And AVX-512 Mask Usage		Vectorized Loops			Instruction Set Analysis		
GFLOPS	AI	Mask Utilization	Vector...	Efficiency	Gain Estim...	VL (...)	Traits
2,080	0,1243	100,0%	AVX512	~100%	17,50x	16; 8	FMA; Mask Manipulations
0,856	0,0809	91,7%	AVX512	~100%	17,69x	16; 8	FMA; Mask Manipulations
0,455	0,1398	89,6%	AVX512	~100%	14,41x	16; 8	FMA; Mask Manipulations
0,234	0,1472	100,0%					Appr. Reciprocals(AVX-512ER); Expon...
0,148	0,1429						FMA
0,095	0,0722	40,1%					FMA; Square Roots; Type Conversions
0,091	0,0208						FMA
0,074	0,1429						FMA





# Gather/Scatter Analysis

## Advisor MAP detects gather “offset patterns”.



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

### Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.  
\*Other names and brands may be claimed as the property of others.



