



# USING INTEL® ADVISOR WITH THE ROOFLINE MODEL TO BOOST YOUR APPLICATIONS

Profiling and improving your vectorization

# 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>

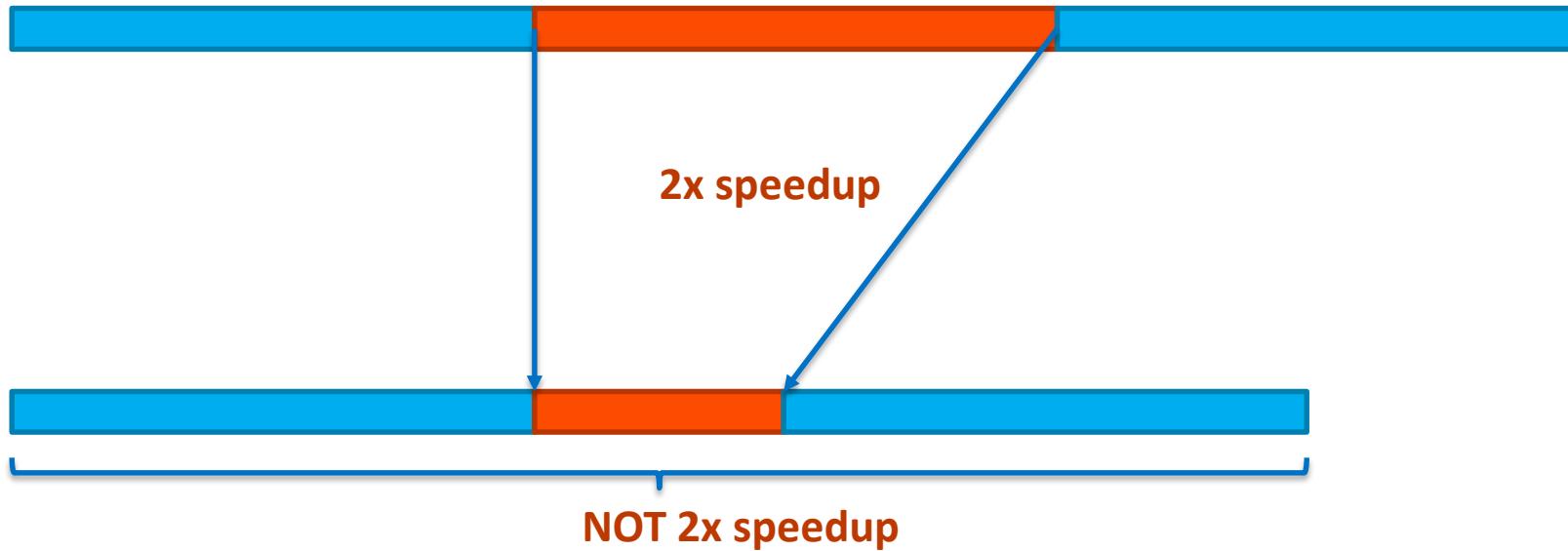
At Intel:

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

# AMDHAL'S LAW

- $S_{total} = \frac{1}{(1-p) + \frac{p}{s}}$
- S = speedup
- P = proportion of execution time that benefit from threading

# AMDHAL'S LAW



# CHANGING HARDWARE IMPACTS SOFTWARE

More cores → More Threads → Wider vectors

								
								
Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge EP	Intel® Xeon® processor code-named Ivy Bridge EP	Intel® Xeon® processor code-named Skylake EP	Intel® Xeon Phi™ coprocessor Knights Corner	Intel® Xeon Phi™ processor & coprocessor Knights Landing <sup>1</sup>
Core(s)	1	2	4	6	8	12	28	61
Threads	2	2	8	12	16	24	56	244
SIMD Width	128	128	128	128	256	256	256	512
*Product specification for launched and shipped products available on ark.intel.com.								

\*Product specification for launched and shipped products available on ark.intel.com.

1. Not launched or in planning.

- High performance software must be both:
  - Parallel (multi-thread, multi-process)
  - Vectorized

# DON'T USE A SINGLE VECTOR LANE!

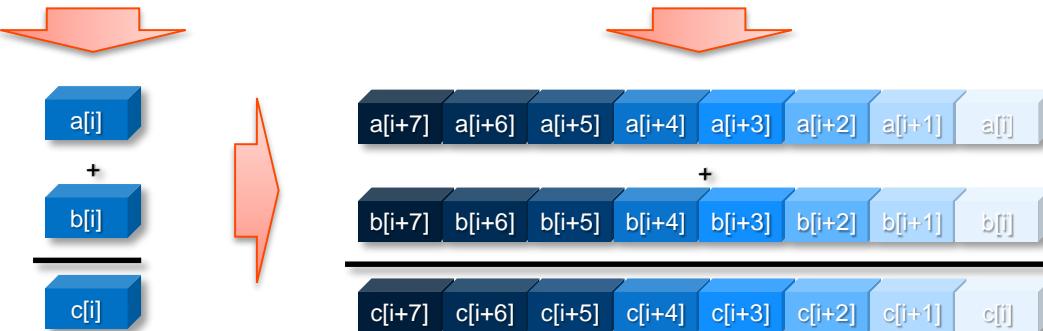
UN-VECTORIZED AND UN-THREADED SOFTWARE WILL UNDER PERFORM



# SINGLE INSTRUCTION MULTIPLE DATA (SIMD)

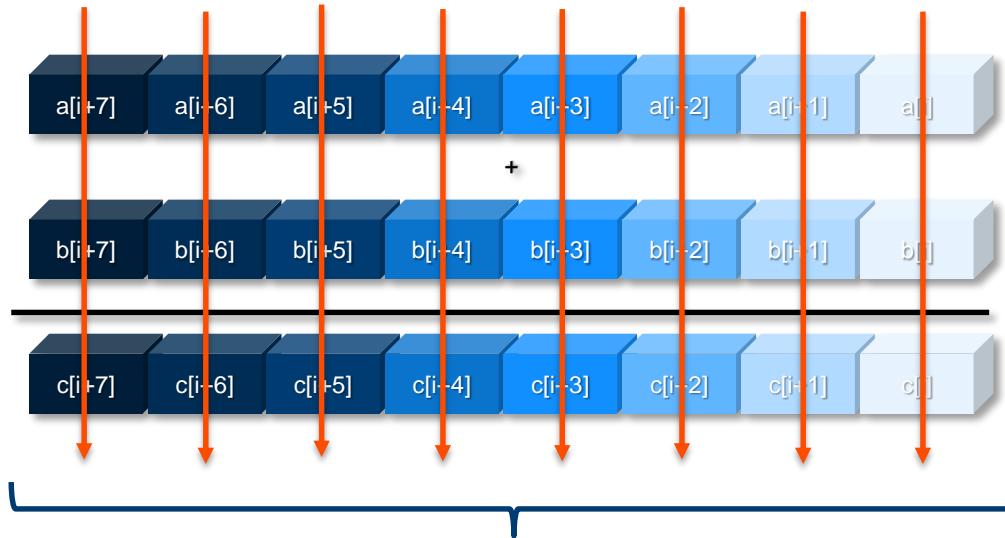
```
for(i = 0; i <= MAX; i++)  
    c[i] = a[i] + b[i];
```

```
for(i = 0; i <= MAX; i+8)  
    c[i:8] = a[i:8] + b[i:8];
```



# VECTORIZATION TERMS

## Vector lanes



Vector length(VL): Elements of the vector

All elements of vector are of the same data types

# VECTORIZATION OF A LOOP

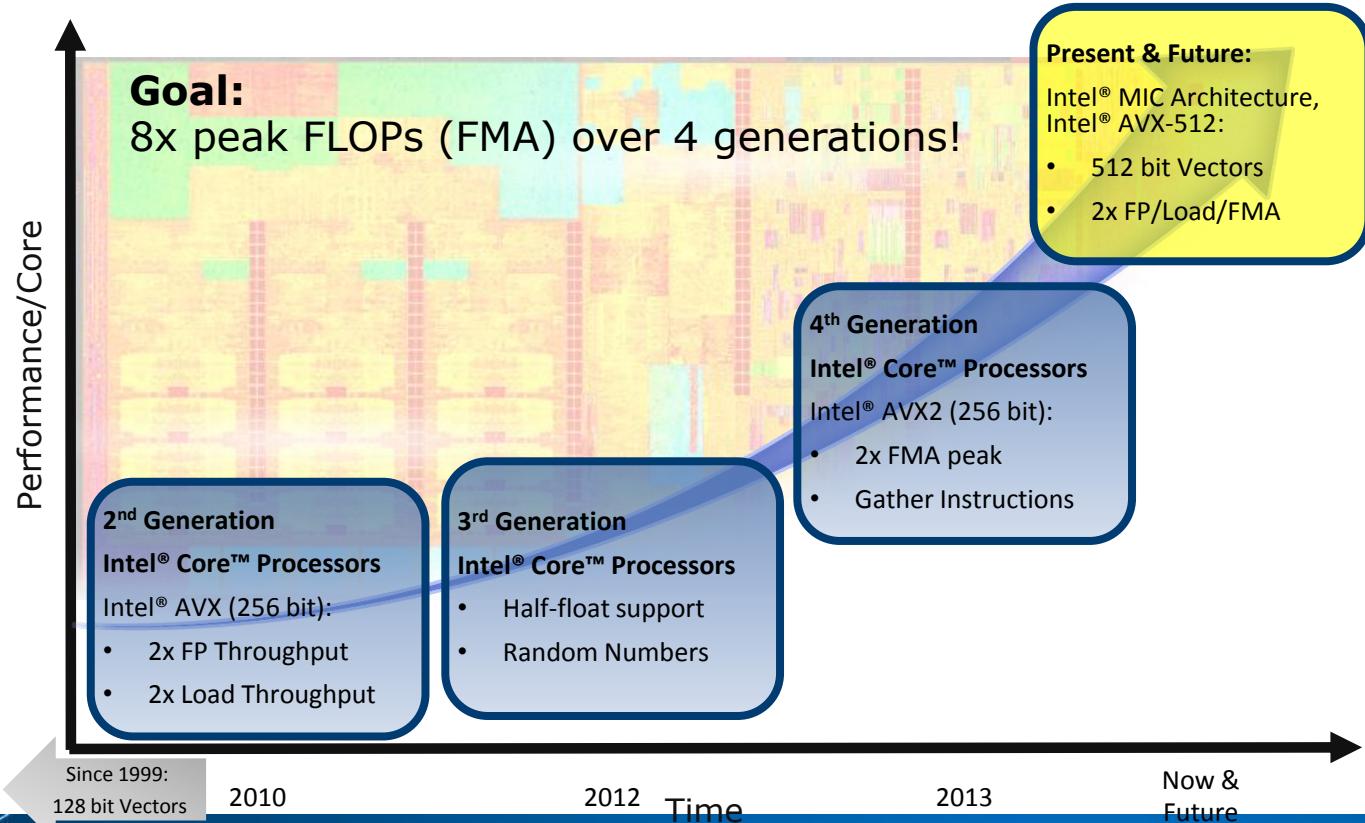
```
for(i = 0; i < MAX; i++)
    A[i] = B[i] + C[i];
```



```
// Vectorized loop body
for(i = 0; i < MAX; i+=VL)
    A[i:VL] = B[i:VL] + C[i:VL];

// loop reminder
for(i = i-VL; i < MAX; i++)
    A[i] = B[i] + C[i];
```

# EVOLUTION OF SIMD FOR INTEL PROCESSORS

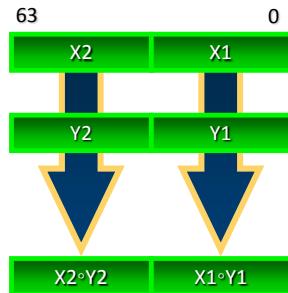


© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

For more complete information about compiler optimizations, see our [Optimization Notice](#).

# SIMD TYPES FOR INTEL® ARCHITECTURE I

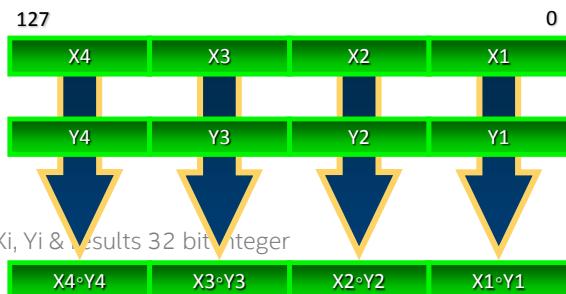


## MMX™

Vector size: **64 bit**

Data types:

- 8, 16 and 32 bit integer
- VL: 2, 4, 8



- Illustrations: Xi, Yi & Results 32 bit integer

## SSE

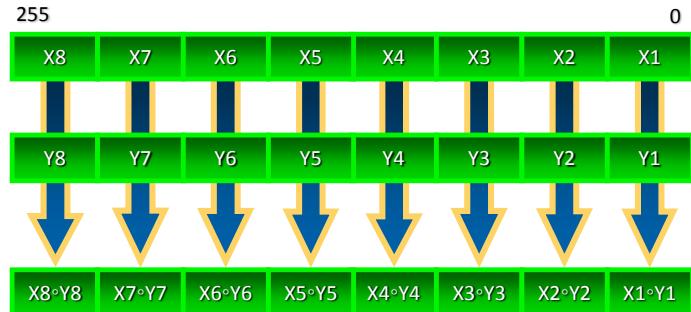
Vector size: **128 bit**

Data types:

- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 2, 4, 8, 16

# SIMD TYPES FOR INTEL® ARCHITECTURE II



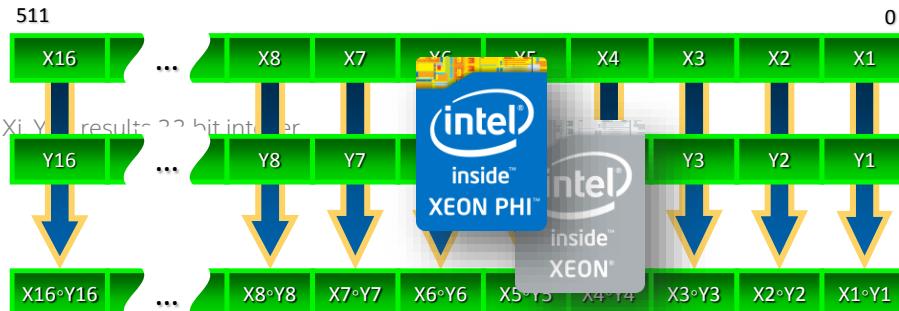
## AVX

Vector size: **256 bit**

Data types:

- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 4, 8, 16, 32



## Intel® AVX-512 & Intel® MIC Architecture

Vector size: **512 bit**

Data types:

- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

VL: 8, 16, 32, 64

# SSE VECTOR TYPES

Intel® SSE



4x single precision FP

Intel® SSE2



2x double precision FP



16x 8 bit integer



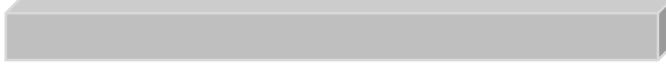
8x 16 bit integer



4x 32 bit integer



2x 64 bit integer



plain 128 bit

# SSE GENERATIONS

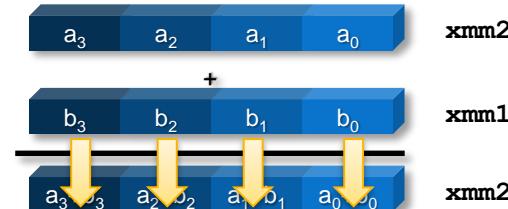
1999	2000	2004	2006	2007	2008
Intel® SSE	Intel® SSE2	Intel® SSE3	Intel SSSE3	Intel® SSE4.1	Intel® SSE4.2
<b>70 new instructions</b> 4 single-precision vector FP scalar FP instructions cacheability instructions control & conversion instructions media extensions	<b>144 new instructions</b> 2 double-precision vector FP 8/16/32/64 vector integer 128-bit integer memory & power management	<b>13 new instructions</b> FP vector calculation x87 integer conversion 128-bit integer unaligned load thread sync.	<b>32 new instructions</b> enhanced packed integer calculation	<b>47 new instructions</b> packed integer calculation & conversion better vectorization by compiler load with streaming hint	<b>7 new instructions</b> string (XML) processing POP-Count CRC32

# SSE PACKED VS. SCALAR

- Packed SSE instructions operate on all elements per vector
- Most of these instructions have **scalar** versions operating only on one element of vector
- Avoid scalar versions and only **use packed instructions** to exploit SIMD capabilities!

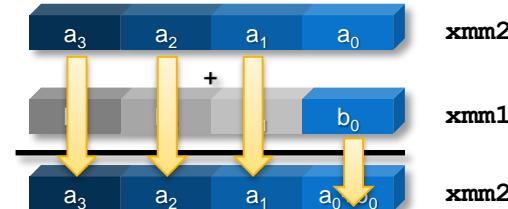
Packed single-precision FP Addition:

```
addps xmm2, xmm1  
single-precision FP data type  
packed execution mode
```



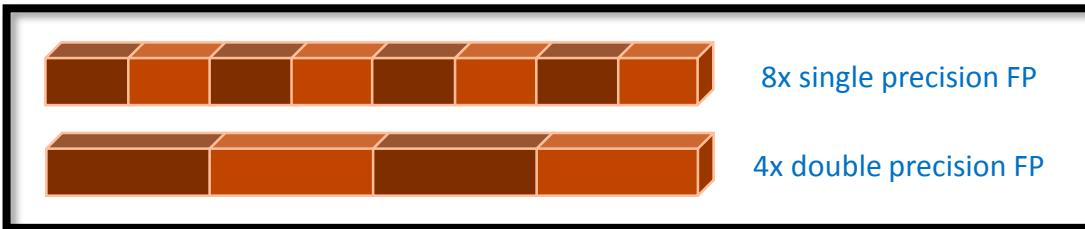
Scalar single-precision FP Addition:

```
addss xmm2, xmm1  
single-precision FP data type  
scalar execution mode
```

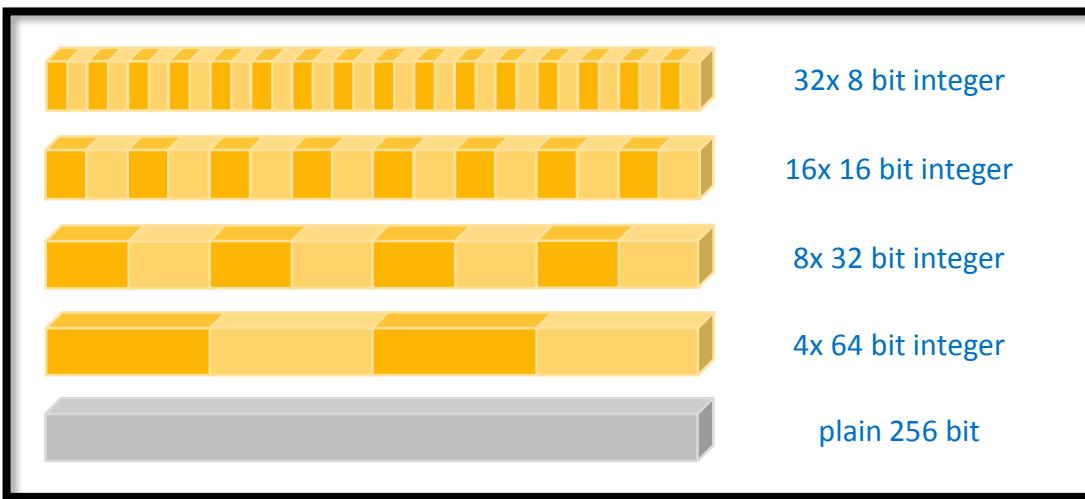


# AVX VECTOR TYPES

Intel® AVX



Intel® AVX2

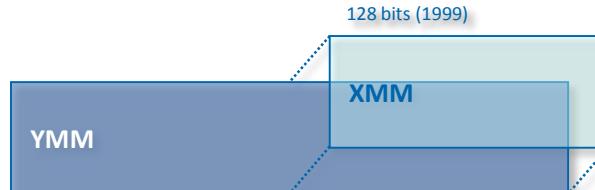


# AVX GENERATIONS

- 2010: Initial version of Intel® AVX in 2<sup>nd</sup> generation Intel® Core™ processors:
  - Double register size of SSE, twice as much vector elements (2x peak FLOP)
  - Support for single- & double-precision FP
  - Load/Store size increased from 128 bit to 256 bit!
- 2012: 3<sup>rd</sup> generation Intel® Core™ processor improvements:
  - Non-deterministic random number generator
  - Half-precision conversion (from/to single-precision)
- 2013: 4<sup>th</sup> generation Intel® Core™ processor improvements:
  - Intel® AVX2 (with integer support)
  - FMA (2x more peak FLOP with same vector length)
  - Gather non adjacent memory locations
- Future: Intel® AVX-512

# AVX IS SSE EXTENSION

- A 256 bit vector extension to SSE:
  - SSE uses dedicated 128 bit registers called XMM (8 for IA-32 & 16 for Intel® 64)
  - Extends all XMM registers to 256 bit called YMM
  - Lower 128 bit of YMM register are mapped/shared with XMM
- AVX works on either
  - The whole 256 bit
  - The lower 128 bit; zeros the higher 128 bit
  - AVX counterparts for almost all existing SSE instructions:  
For initial generation (Intel® AVX) full 256 bit vectors for FP only; integers will follow!



# INTEL® ADVISOR





## Boosting your application by threading and vectorization

- Compiler will not always vectorize
  - Check for Loop Carried Dependencies using [Intel® Advisor](#)
  - All clear? Force vectorization.  
C++ use: pragma simd, Fortran use: SIMD directive
- Not all vectorization is efficient vectorization
  - Stride of 1 is more cache efficient than stride of 2 and greater. Analyze with [Intel® Advisor](#).
  - Consider data layout changes  
[Intel® SIMD Data Layout Templates](#) can help

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the [Intel® SIMD Data Layout Templates](#) (Intel® SDLT) to map data into a more efficient layout for vectorization.

# GET FASTER CODE FASTER! INTEL® ADVISOR

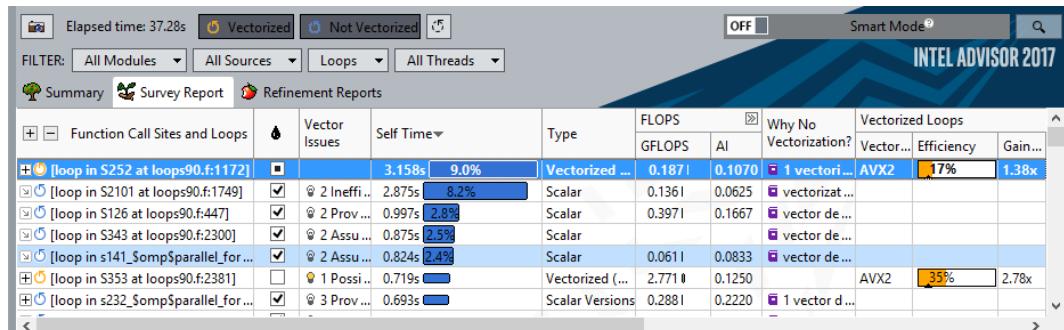
## Vectorization Optimization

Have you:

- Recompiled for AVX2 with little gain
- Wondered where to vectorize?
- Recoded intrinsics for new arch.?
- Struggled with compiler reports?

Data Driven Vectorization:

- What vectorization will pay off most?
- What's blocking vectorization? Why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma SIMD?



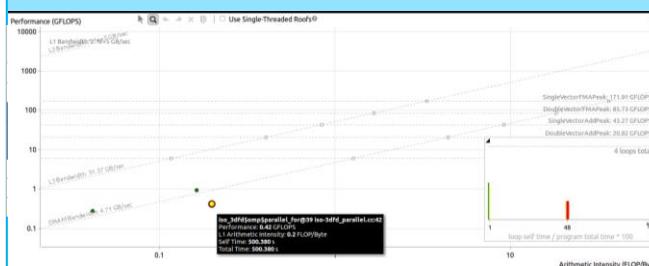
"Intel® Advisor's Vectorization Advisor permitted me to focus my work where it really mattered. When you have only a limited amount of time to spend on optimization, it is invaluable."

Gilles Civario  
Senior Software Architect  
Irish Centre for High-End Computing

# 4 STEPS TO EFFICIENT VECTORIZATION

## Intel® Advisor – Vectorization Advisor

### 1. Compiler diagnostics + Performance Data + Roofline Model + SIMD efficiency information



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

**⚠ 2 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
```

### 3. Loop-Carried Dependency Analysis

#### Problems and Messages

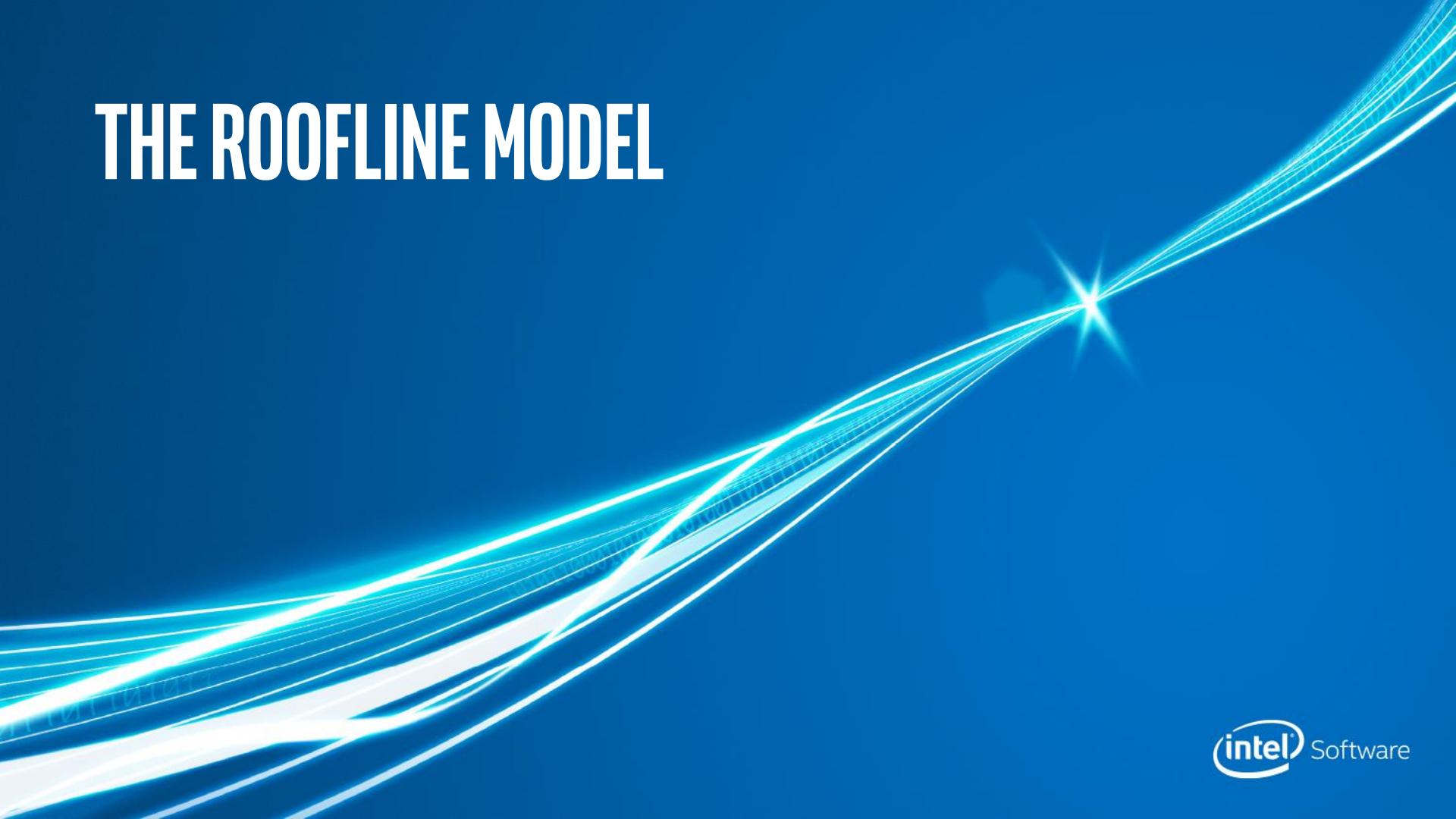
ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5	Write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

### 4. Memory Access Patterns Analysis

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

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runRawLoops.cxx:637	lcalcs.exe	
	635	j2 = ( j2 + 64-1 ) ;			
	636	p[1p][0] += v[12+32];			
	637	p[1p][1] += z[j2+32];			
	638	j2 += e[12+32];			
	639	j2 += f[j2+32];			
P23	0; 0	Unit stride	runRawLoops.cxx:638	lcalcs.exe	
	626	j1 = 64-1;			
	627	j1 = 64-1;			
	628	p[1p][2] += b[j1][11];			

# THE ROOFLINE MODEL



# WHAT IS THE ROOFLINE MODEL ?

Do you know how fast you should run ?

- Comes from Berkeley
- Performance is limited by equations/implementation & code generation/hardware
- 2 hardware limitations
  - PEAK Flops
  - PEAK Bandwidth
- The application performance is bounded by hardware specifications

$$\text{Gflop/s} = \min \begin{cases} \textbf{\textit{Platform PEAK}} \\ \textbf{\textit{Platform BW}} * \textbf{\textit{AI}} \end{cases}$$

Arithmetic Intensity  
(Flops/Bytes)



# PLATFORM PEAK FLOPS

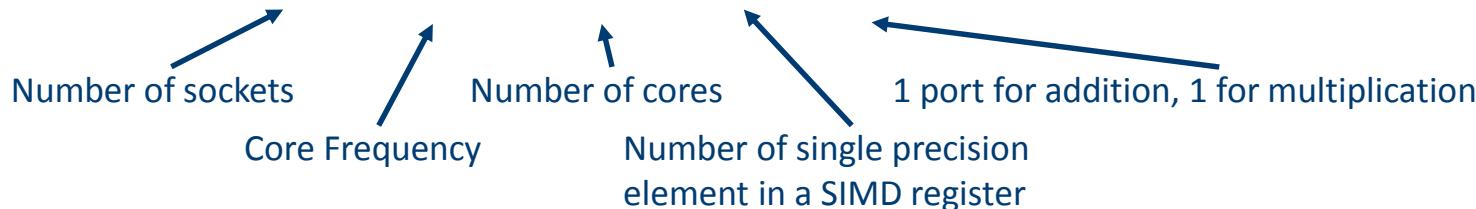
How many floating point operations per second

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \textbf{\textit{Platform PEAK}} \\ \textbf{\textit{Platform BW}} * \textbf{\textit{AI}} \end{array} \right.$$

- Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

$$\text{PEAK FLOP} = 2 \times 2.7 \times 12 \times 8 \times 2 = 1036.8 \text{ Gflop/s}$$



- More realistic value can be obtained by running Linpack  
=~ 930 Gflop/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

# PLATFORM PEAK BANDWIDTH

How many bytes can be transferred per second

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Platform PEAK} \\ \text{Platform BW} * AI \end{array} \right.$$

- Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

$$\text{PEAK BW} = 2 \times 1.866 \times 8 \times 4 = 119 \text{ GB/s}$$

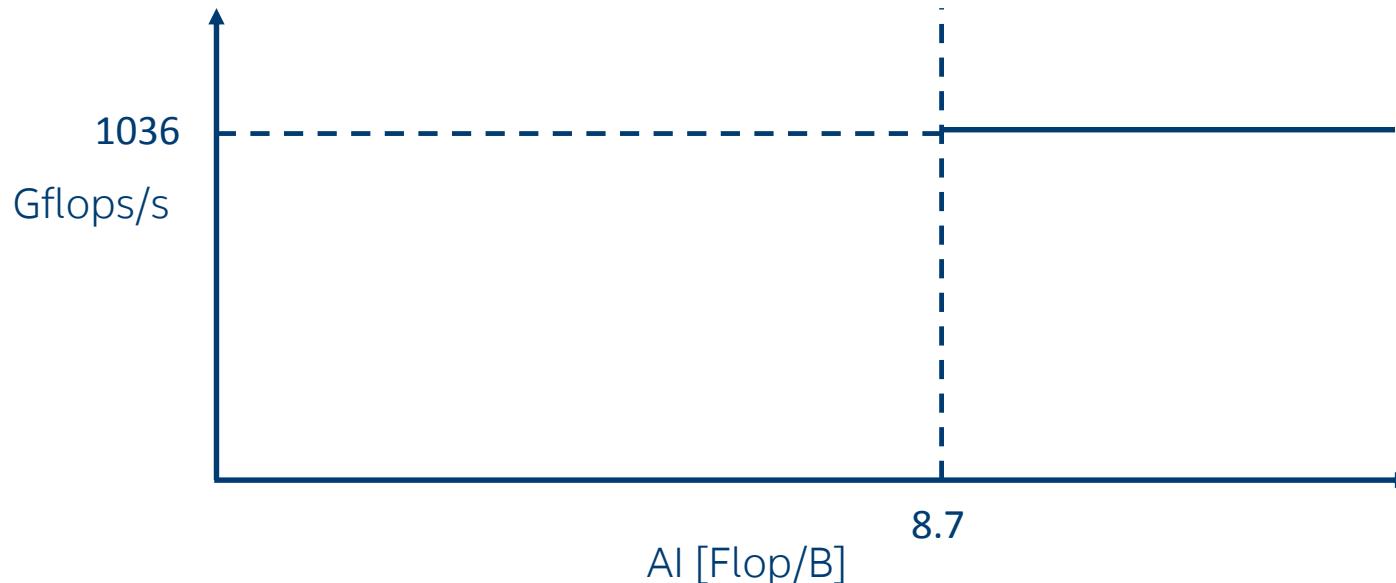
Number of sockets      Memory Frequency      Byte per channel      Number of mem channels

- More realistic value can be obtained by running Stream  
≈ 100 GB/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

# DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \min \begin{cases} \textbf{\textit{Platform PEAK}} \\ \textbf{\textit{Platform BW}} * \text{AI} \end{cases}$$



2 sockets Intel® Xeon® Processor E5-2697 v2

Peak Flop = 1036 Gflop/s

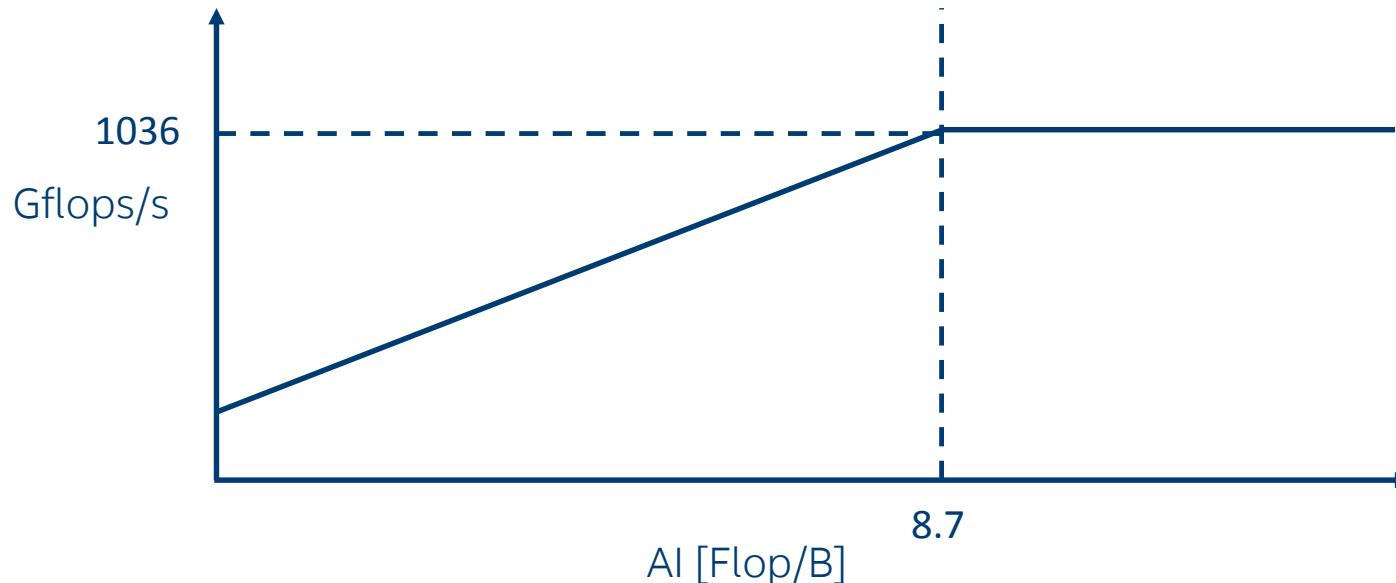
Peak BW = 119 GB/s

# DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \min \begin{cases} \textbf{\textit{Platform PEAK}} \\ \textbf{\textit{Platform BW}} * \text{AI} \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2  
Peak Flop = 1036 Gflop/s  
Peak BW = 119 GB/s



# WHAT IS THE PERFORMANCE BOUNDARY?

Manual way to do it

- Manual counting on matrix/matrix multiplication

```
for(i=0; i<N; i++)  
    for(j=0; j<N; j++)  
        for(k=0; k<N; k++)  
            c[i][j] = c[i][j] + a[i][k] * b[k][j]
```

- # add =  $N * N * N$                           #Read =  $3 * N * N * 4$  bytes
- # mul =  $N * N * N$                           #Write =  $N * N * 4$  bytes
- $$AI = \frac{2N^3}{16N^2} = \frac{1}{8}N$$

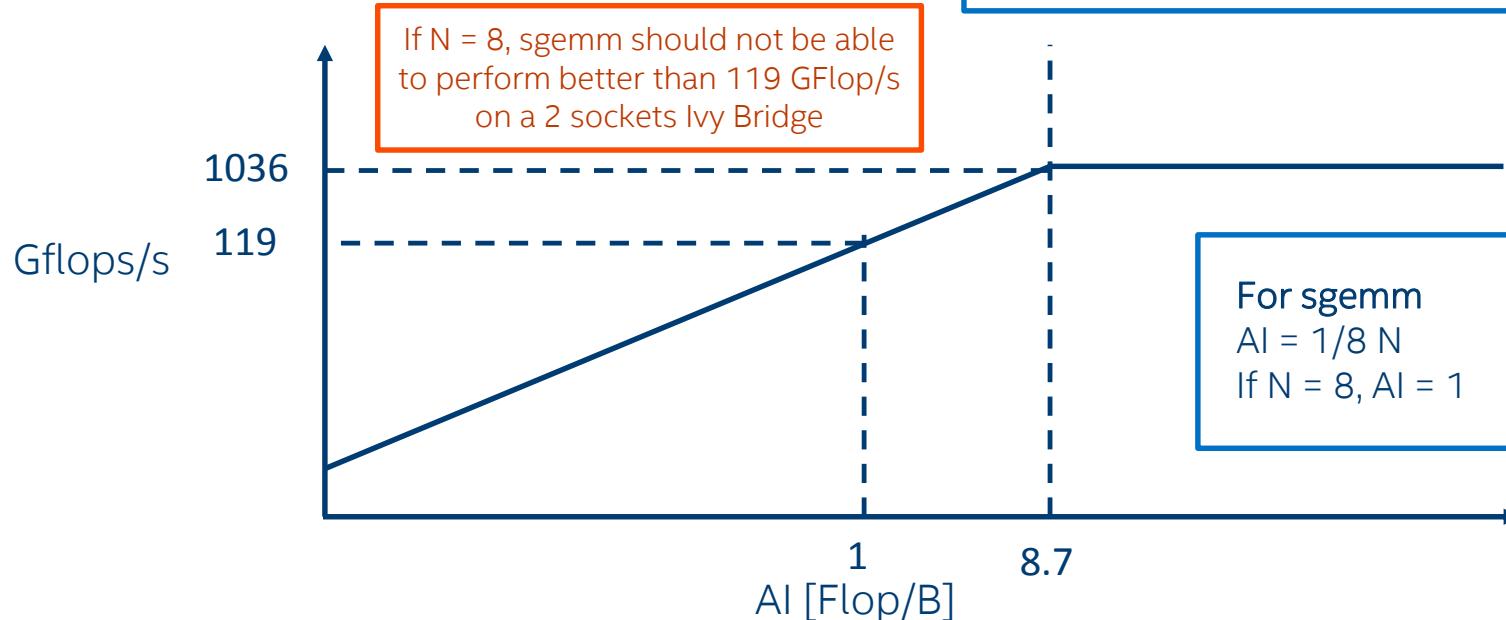
# COMPUTE THE MAXIMUM PERFORMANCE

BW \* Arithmetic Intensity

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \textbf{\textit{Platform PEAK}} \\ \textbf{\textit{Platform BW}} * \text{AI} \end{array} \right.$$

If N = 8, sgemm should not be able  
to perform better than 119 GFlop/s  
on a 2 sockets Ivy Bridge

2 sockets Intel® Xeon® Processor E5-2697 v2  
Peak Flop = 1036 Gflop/s  
Peak BW = 119 GB/s



# HOW CAN I DRAW A POINT?

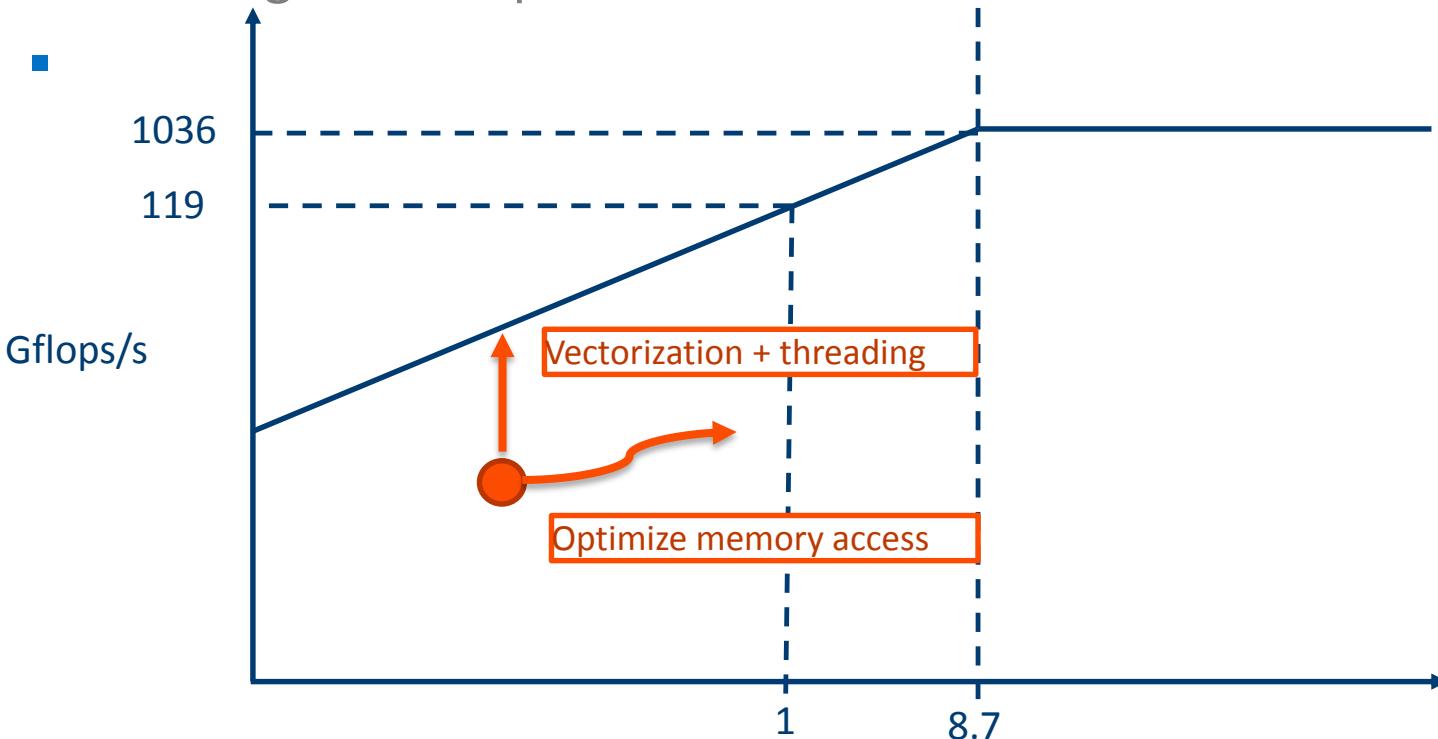
## Retrieving real AI and real Flop/s

In order to know where your application stands on this graph, you need to extract some metrics

- **Effective flops** can be extracted with SDE, or Hardware counters (if available). In practice, effective flops and theoretical flops (manual counting) should be very close. Adding timers in your application allows to also get flop/s
- **Effective AI** requires the amount of bytes transferred from uncore to core + flops. Amount of bytes transferred can be obtained with emon (reading hardware counters).

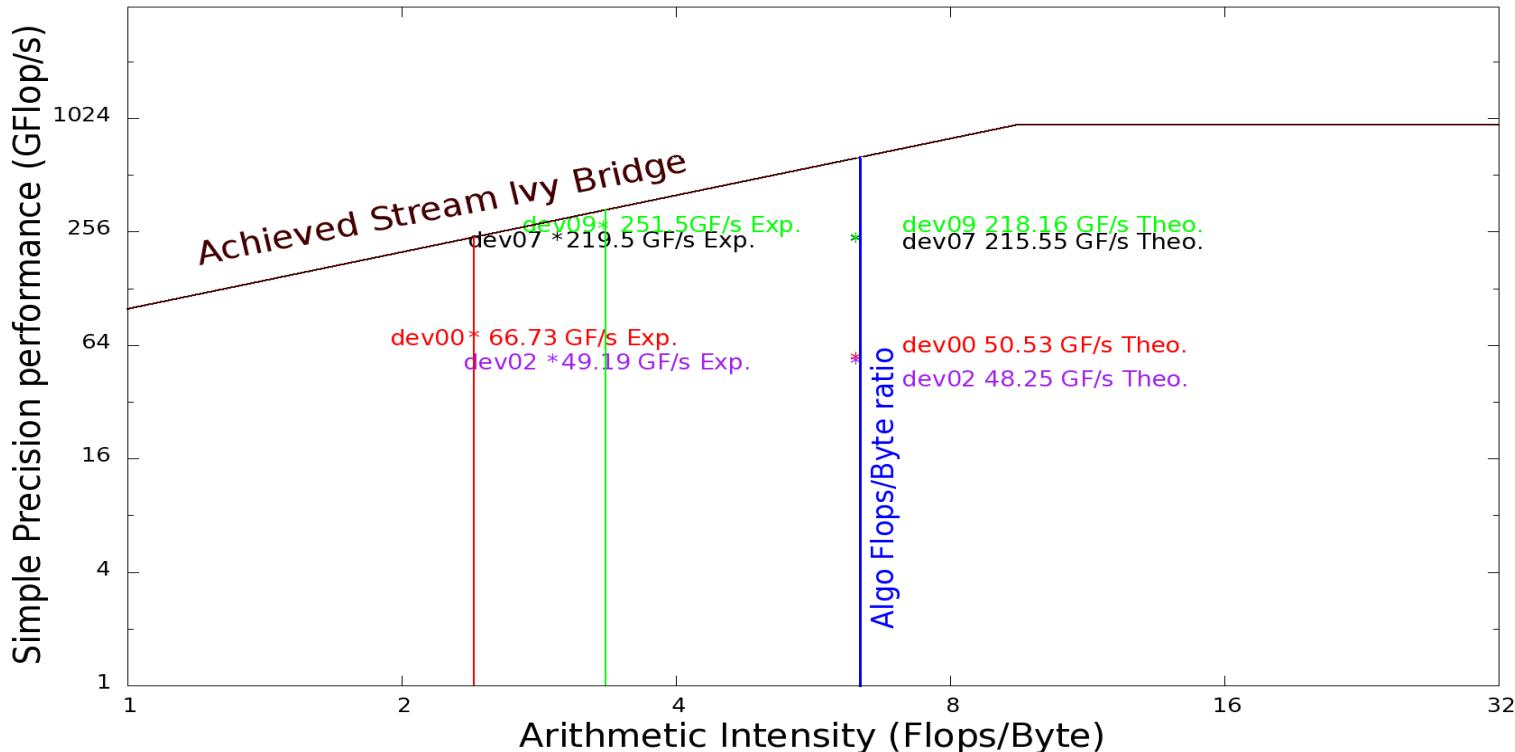
# AND NOW?

How to get better performance?



# UNDERSTANDING THE ROOFLINE

Comparing Arithmetic Intensity on 2S Ivy Bridge for Iso3DFD



# ROOFLINE IN INTEL® ADVISOR

## The cache aware roofline model

Intel® Advisor implements a Cache Aware Roofline Model (CARM)

- “Algorithmic”, “Cumulative (L1+L2+LLC+DRAM)” traffic-based
- Invariant for the given code / platform combination

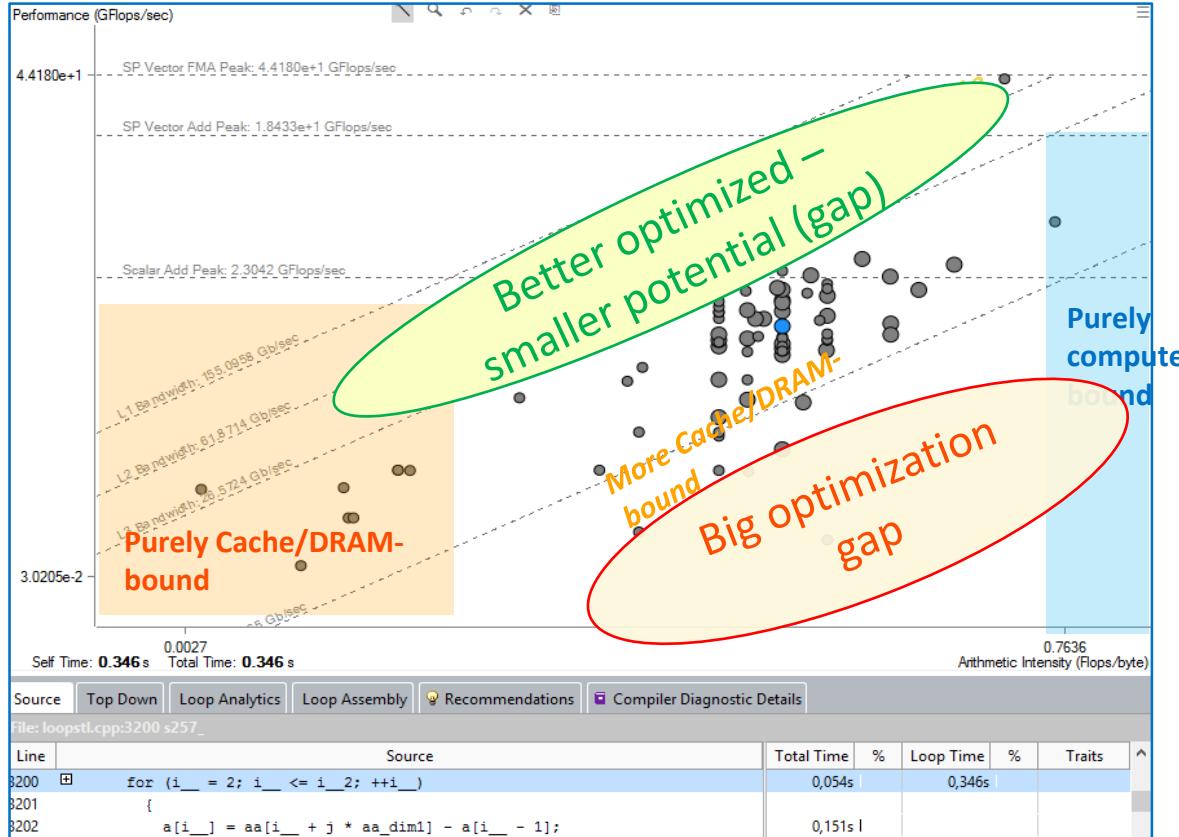
How does it work ?

- Counts every memory movement
- **Bytes and Flops** -> Instrumentation
- **Time** -> Sampling

CARM: Cache aware Roofline Model  
DRAM: DRAM aware Roofline Model  
TRAM: Theoretical Roofline Model

Typically AI\_CARM < AI\_DRAM < AI\_TRAM

# UNDERSTANDING THE ROOFLINE IN INTEL® ADVISOR



© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

For more complete information about compiler optimizations, see our [Optimization Notice](#).

# Data Alignment

# EFFICIENTLY VECTORIZE YOUR CODE

## INTEL ADVISOR - VECTORIZATION ADVISOR

The screenshot shows the Intel Advisor 2017 Vectorization Advisor interface. At the top, there are tabs for 'Elapsed time: 37.28s', 'Vectorized' (selected), 'Not Vectorized', and a search icon. Below are filter dropdowns for 'All Modules', 'All Sources', 'Loops', and 'All Threads'. The main area is titled 'INTEL ADVISOR 2017' and contains a 'Summary' tab with a tree view of function call sites and loops. A table lists various loops with their details:

Loop ID	Function Call Site	Type	FLOPS		Why No Vectorization?	Vectorized Loops		
			GFLOPS	AI		Vector...	Efficiency	Gain...
+ 1	[loop in S252 at loops90.f:1172]	Vectorized ...	0.1871	0.1070	1 vectori...	AVX2	17%	1.38x
2	[loop in S2101 at loops90.f:1749]	Scalar	0.1361	0.0625	vectorizat...			
3	[loop in S126 at loops90.f:447]	Scalar	0.3971	0.1667	vector de...			
4	[loop in S343 at loops90.f:2300]	Scalar	0.8755	0.25	vector de...			
5	[loop in s141_Somp\$parallel_for...]	Scalar	0.8245	0.24%	vector de...			
6	[loop in S353 at loops90.f:2381]	Vectorized (...)	2.7710	0.1250	AVX2	35%	2.78x	
7	[loop in s232_Somp\$parallel_for...]	Scalar Versions	0.2881	0.2220	1 vectori...			

Below the table are tabs for 'Source', 'Top Down', 'Loop Analytics', 'Loop Assembly', 'Recommendations', and 'Why No Vectorization?'. The 'Source' tab displays the Fortran source code for file loops90.f:1172 S252:

```
1165      integer ld,n,m,i,j,k
1166      real a(n),b(m),c(n),d(n),e(n),aa(ld,n),bb(ld,n),cc(ld,n)
1167      real t1,t2,chksum,ctime,dtime,csld,s,t
1168      call init(ld,n,a,b,c,d,e,aa,bb,cc,'s252 ')
1169      call forttime(t1)
1170      do nl= 1,ntimes
1171      do nl= 1,ntimes
1172      a(:n)=b(:n)*c(:n)+eoshift(b(:n)*c(:n),1)
1173      call dummy(ld,n,a,b,c,d,e,aa,bb,cc,1.)
1174      enddo
1175      call forttime(t2)
1176      t2= t2-t1-ctime-(dtime*float(ntimes))
1177      !
```

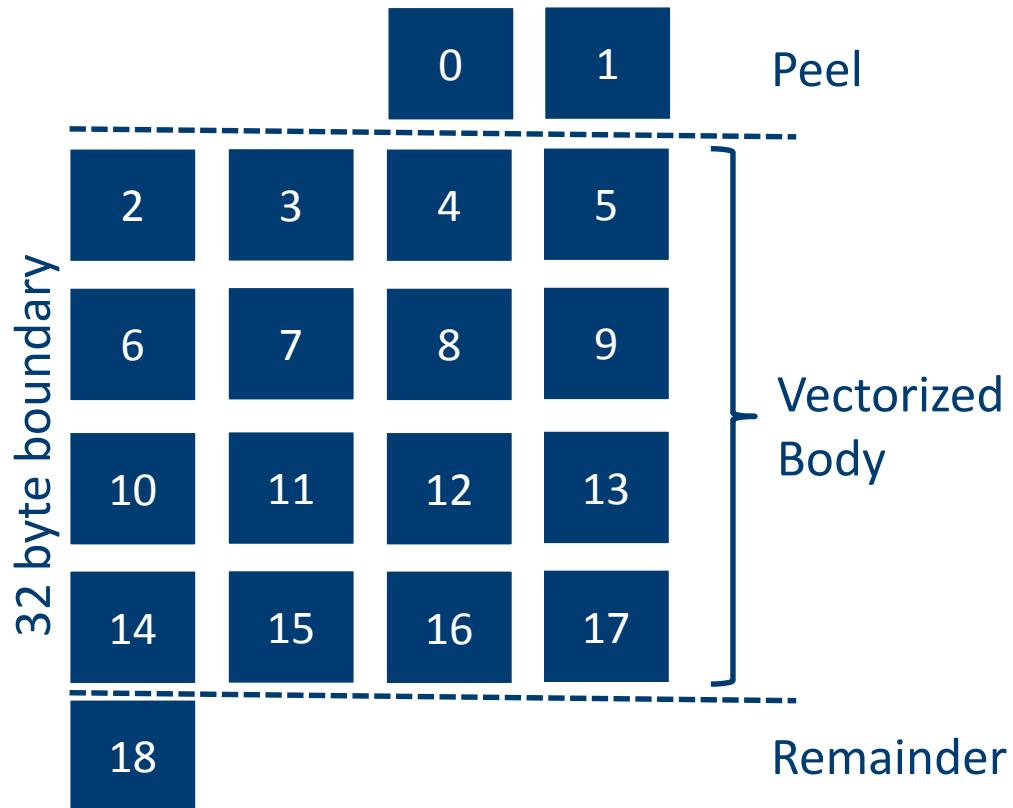
At the bottom right, it says 'Selected (Total Time):' with a progress bar.

# Some Parts of a Vector Are Faster Than Others

- A typical vectorized loop consists of
- Main vector body
  - Fastest among the three!
- Optional peel part
  - Used for the unaligned references in your loop. Uses Scalar or slower vector
- Remainder part
  - Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.
- Larger vector register means more iterations in peel/remainder
  - Make sure you Align your data!
  - Make the number of iterations divisible by the vector length!

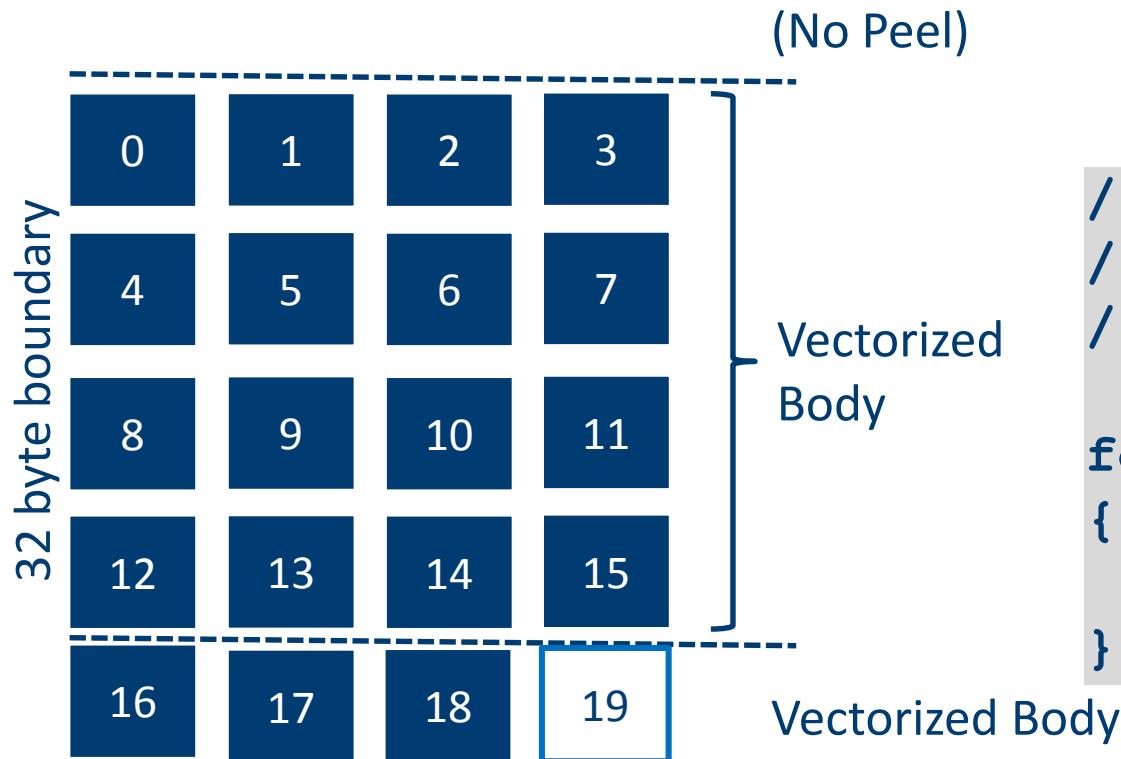
This is where we want our loops  
to be executing!

# WHAT ARE PEELS AND REMAINDERS?



```
// xAVX  
// 256 bits wide regs  
// holds 4 x 64bit vals  
  
void Func(double *pA)  
{  
    for (int i=0; i<19;i++)  
        pA[i] = ...;  
}
```

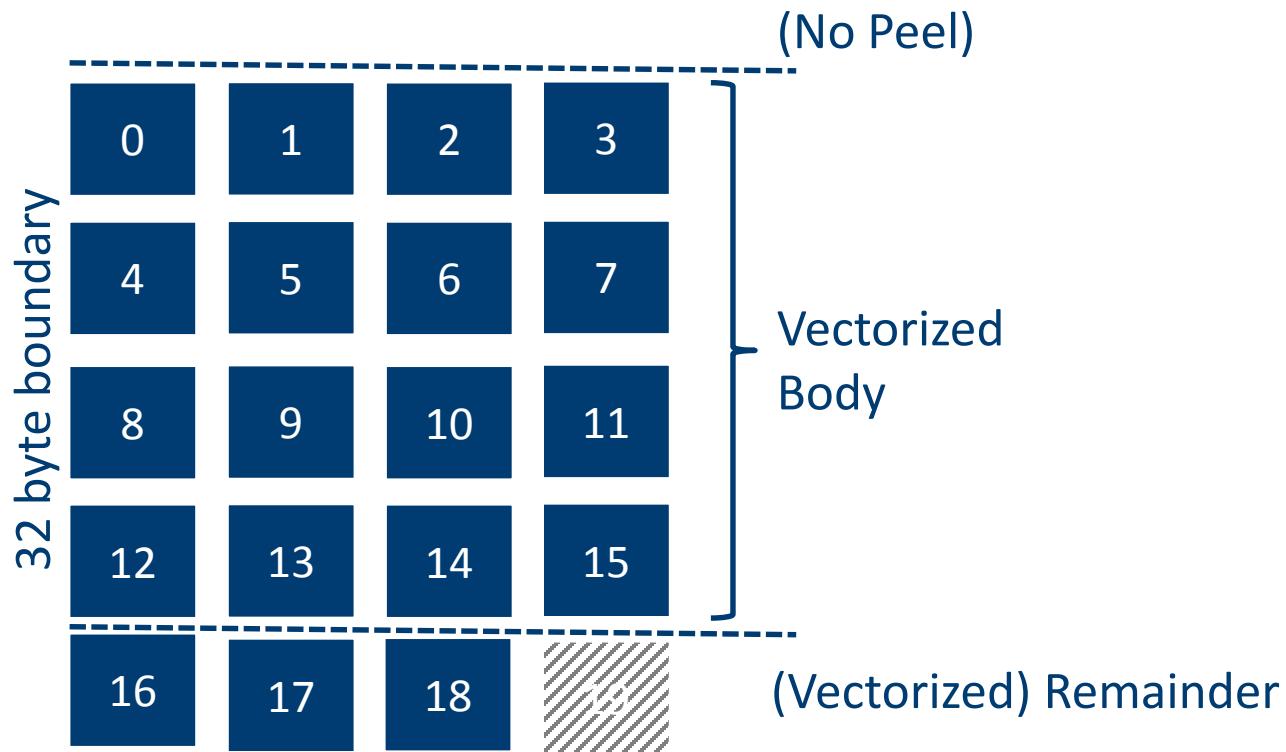
# WORKING ON ALIGNED ARRAYS



```
// xAVX
// 256 bits wide regs
// holds 4 x 64bit vals

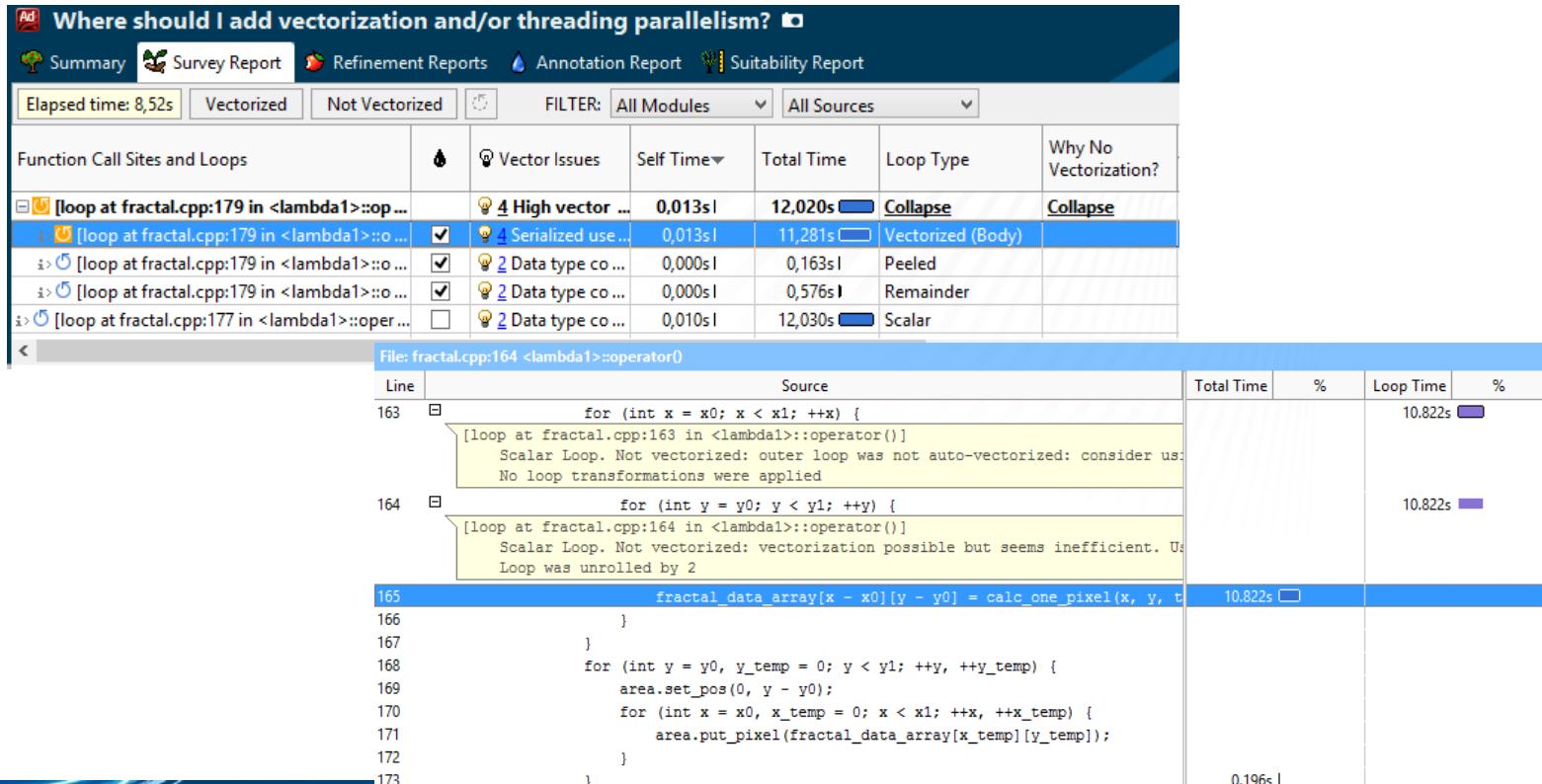
for(i=0; i<lbsy.nq+1; i++)
{
    // . .
}
```

# ALIGNMENT AND REMAINDERS ON AVX512



# IS MOST EXECUTION IN THE FAST PART OF THE VECTOR?

INTEL ADVISOR SHOWS YOU

Where should I add vectorization and/or threading parallelism? 

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
[loop at fractal.cpp:179 in <lambda1>::o ...]	4 Serialized use...	0,013s	11,281s	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	Scalar	

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

Line	Source	Total Time	%	Loop 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 us: No loop transformations were applied	10.822s			
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	10.822s			
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			

© 2017 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

For more complete information about compiler optimizations, see our [Optimization Notice](#).

# THE DEPENDENCY ANALYSIS

# FORCING VECTORIZATION

## Verifying dependency with variables

- Have you seen in compilation reports :  
remark #15344: loop was not vectorized: vector dependence prevents vectorization
- Is it safe to add **#pragma omp simd** ?

### 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)

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M]
```

Input

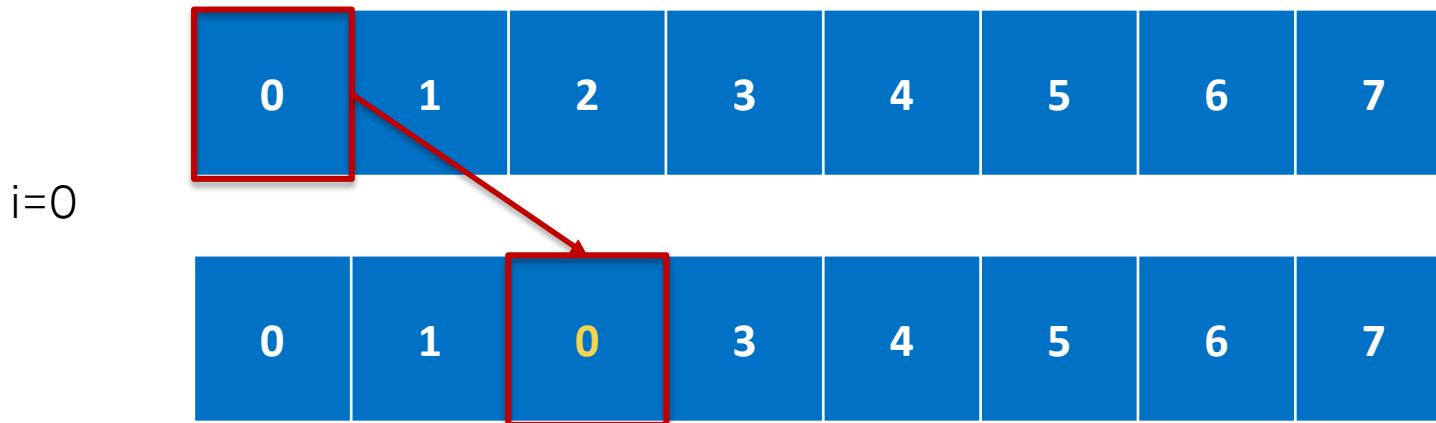
0 1 2 3 4 5 6 7

# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

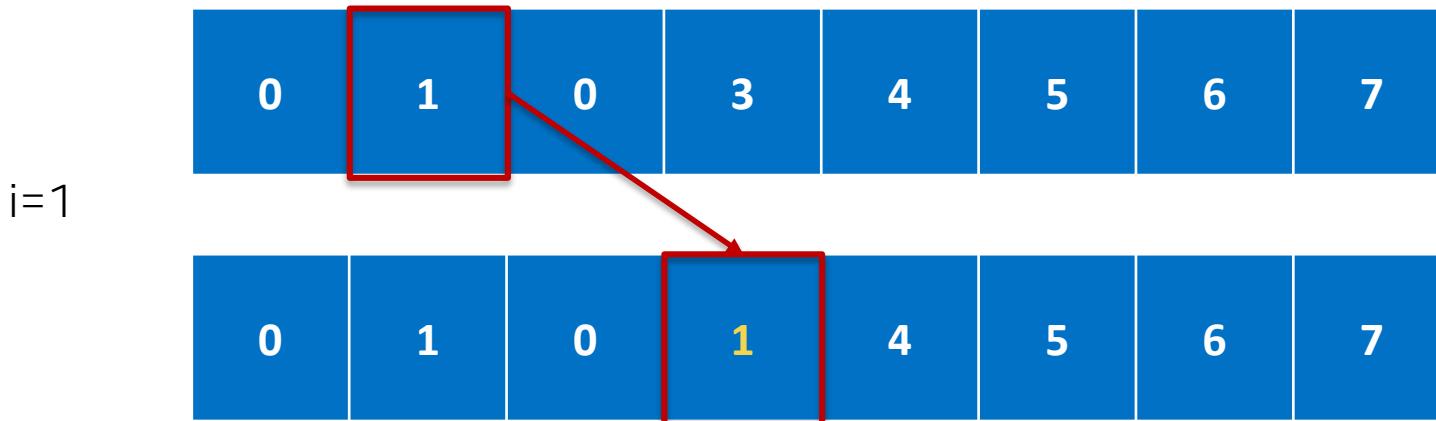


# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

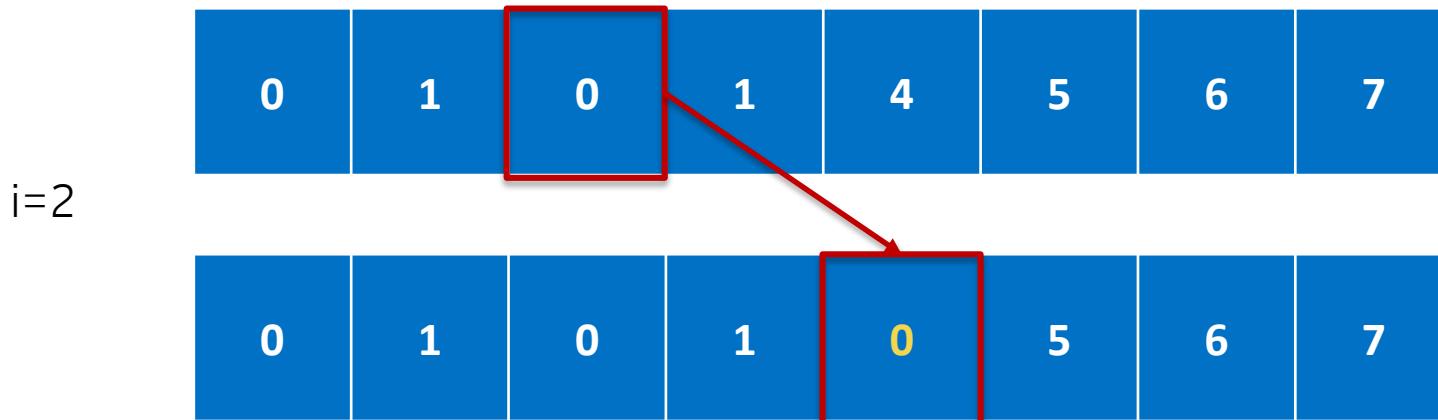


# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

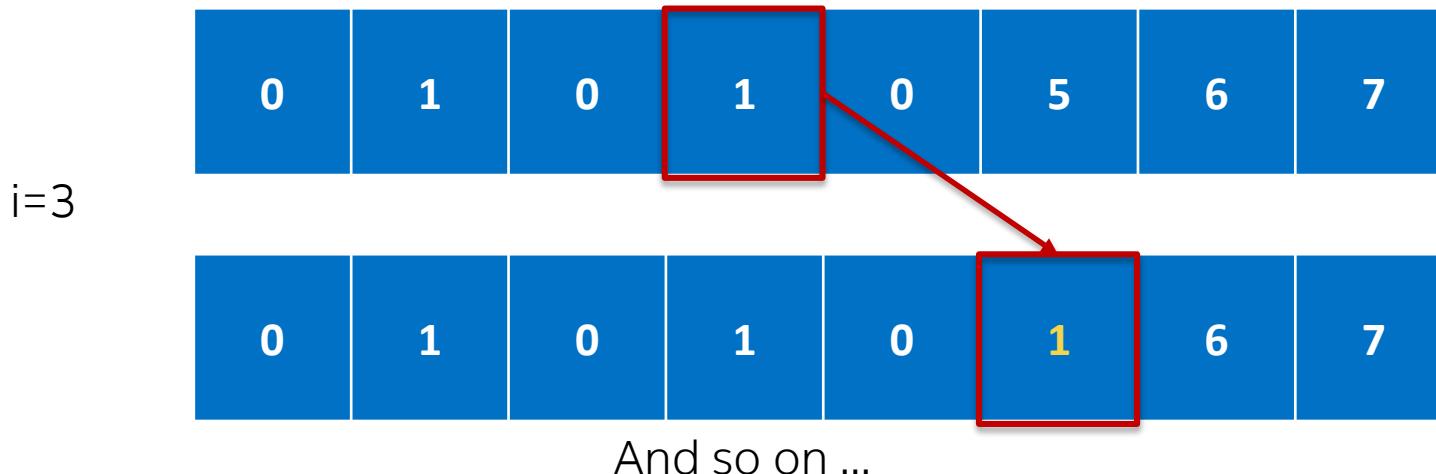


# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```



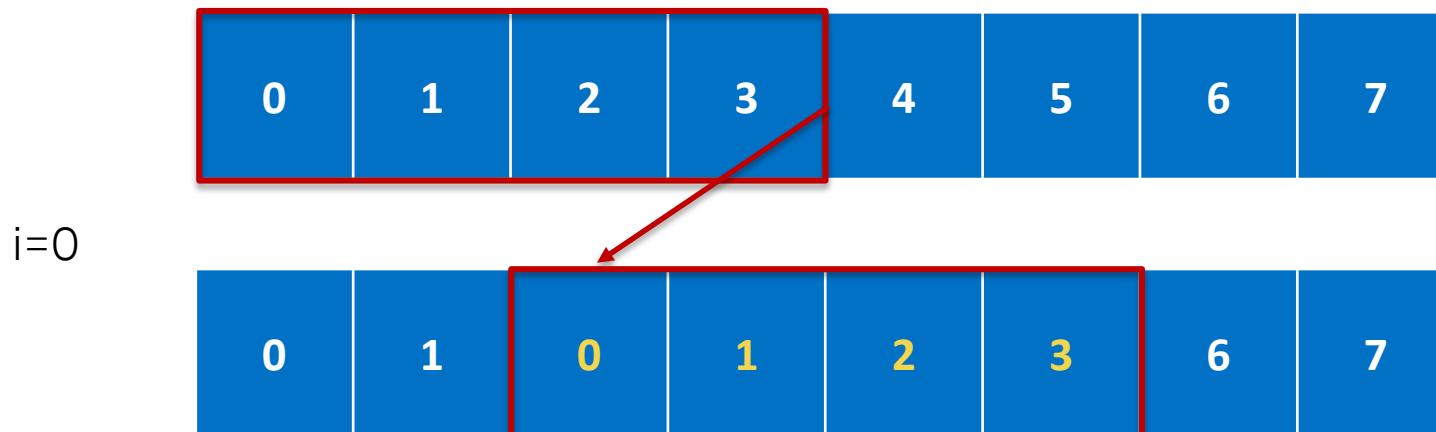
# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

With a vector length of 4



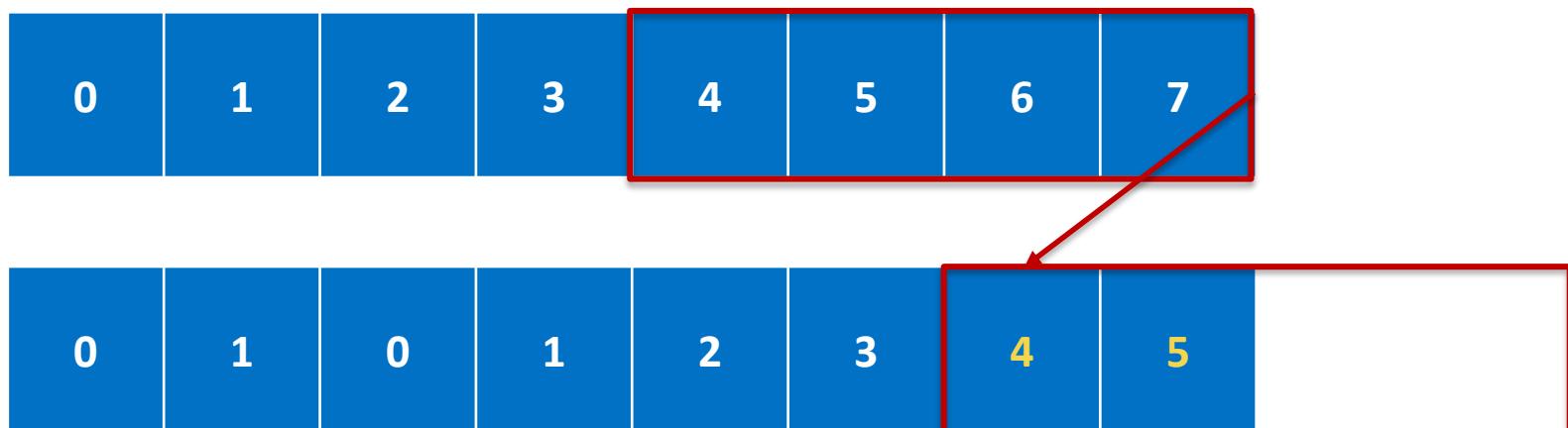
# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

With a vector length of 4



# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

With a vector length of 4

i=1

Doesn't work here



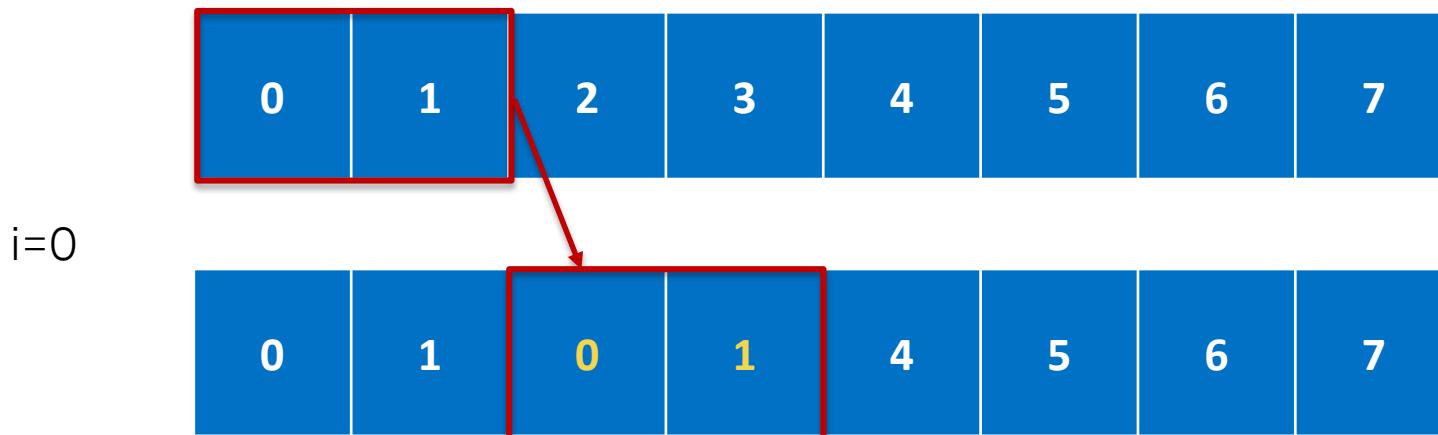
# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

With a vector length of 2



# FORCING VECTORIZATION

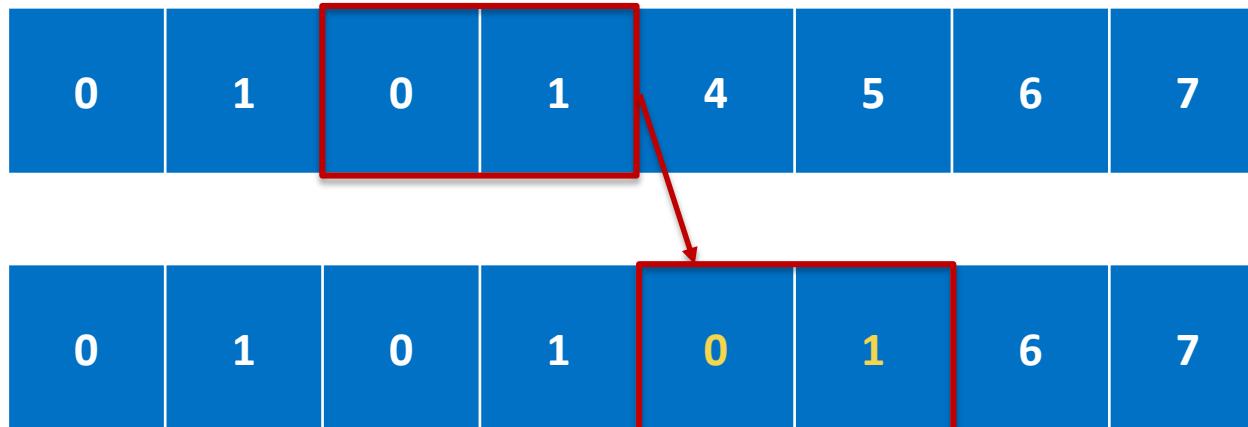
## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?

```
for (i=0; i<N; i++)
    A[i] = A[i-M] // with M=2
```

With a vector length of 2

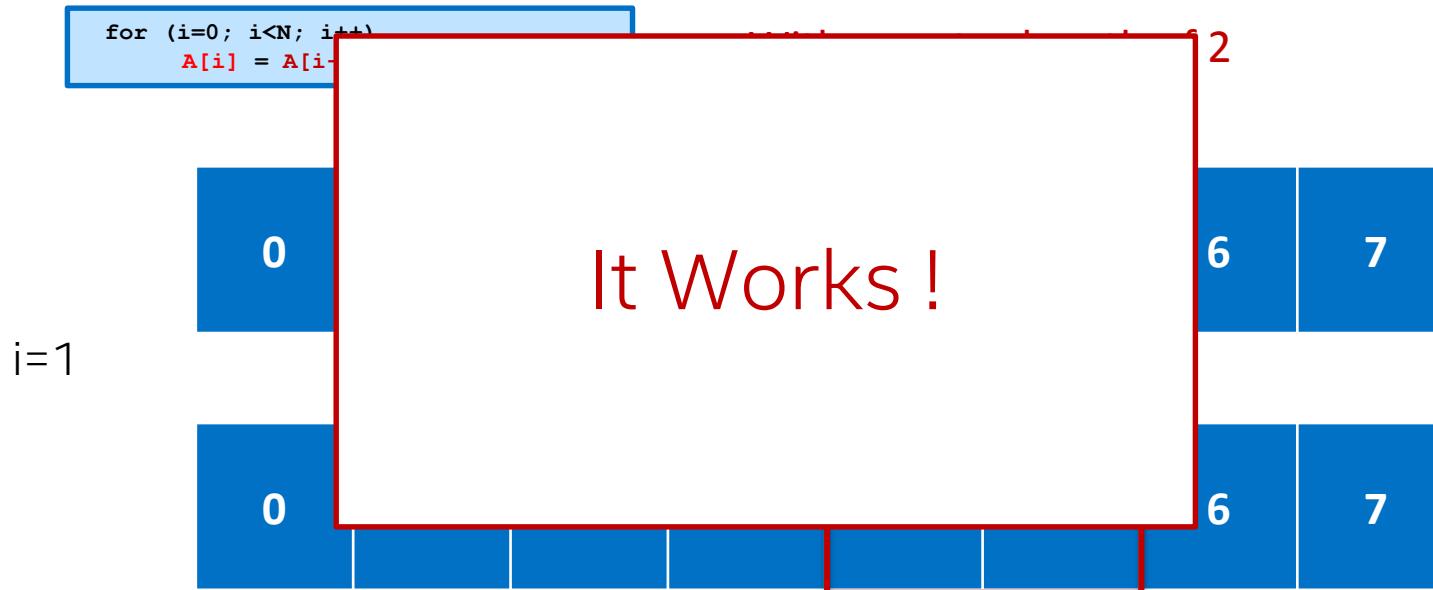
i=1



# FORCING VECTORIZATION

## Verifying dependency with variables

- Is it possible to vectorize loop carried dependencies ?



# Is it safe to vectorize?

The screenshot shows the Intel Advisor XE 2016 interface. The title bar says "Where should I add vectorization and/or threading parallelism?". The main window displays a table of function call sites and loops, categorized by self time and total time. A tooltip on the last row states: "[loop at Driver.c:146 in main] 0.016s | 12.483s | Vectorized | 1 | 1000000 | Scalar | vector dependence prevents vectorization".

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

2.1 Check Correctness

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



[Command Line](#)

Select loop for  
Correct  
Analysis and  
press play!

Vector Dependence  
prevents  
Vectorization!

# HOW DOES INTEL® ADVISOR DISPLAY THE RESULTS?

The screenshot shows the Intel Advisor interface. At the top, a navigation bar includes 'Survey Report', 'Refinement Reports', 'Annotation Report', and 'Suitability Report'. Below this, a summary table provides details about the site: Site Name (loop\_site\_6), Site Function (main), Site Info (main.cpp:13), Loop-Carried Dependencies (RAW:1, WAR:1, WAW:1), Strides Distribution (91% / 0% / 9%), and Access Pattern (Mixed strides). A large blue callout box labeled 'Detected dependencies' points to a table titled 'Problems and Messages' which lists five detected issues: P1 (Parallel site information), P3 (Read after write dependency), P4 (Write after write dependency), and P5 (Write after read dependency). Another blue callout box labeled 'Source lines with Read and Write accesses detected' points to a table titled 'Write after read dependency: Code Locations' which shows two entries: X17 Read and X18 Read, both from main.cpp:22 and main.cpp:23 respectively, with specific line numbers (20-24 and 21-23) highlighted.

- Received recommendations to force vectorization of a loop:
  1. Mark-up loop and check for REAL dependencies
  2. Explore dependencies with code snippets
- In this example 3 dependencies were detected:
  - RAW – Read After Write
  - WAR – Write After Read
  - WAW – Write After Write
- **This is NOT a good candidate to force vectorization!**

# CACHE SIMULATION

# Utilize memory sub-system effectively

Intel® Advisor: Cache simulation in frames of MAP analysis (\*)

- Memory Performance Insights

- Explores efficiency of caching:
  - Cache Line Utilization
  - Misses
  - Evictions

Read for ownership

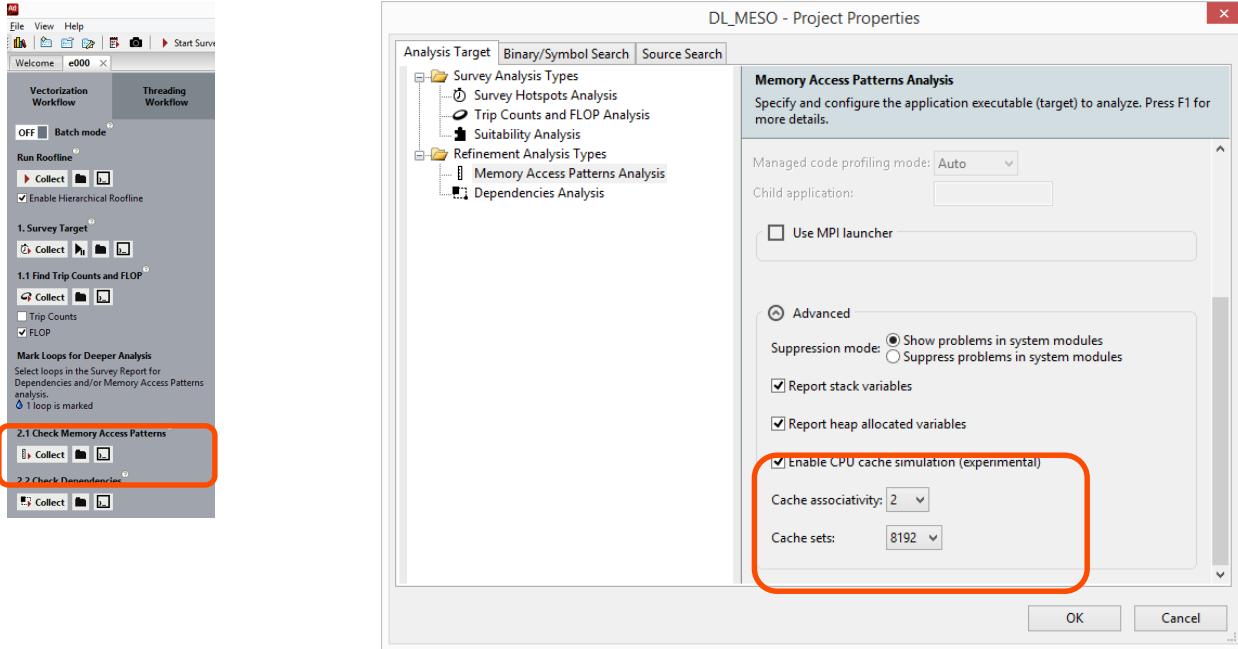


Site Name	Recommendations	Cache Line Utilization	Memory Loads	Memory Stores	Cache Misses	RFO Cache Misses	Dirty Evictions
loop_site_157	2 Potential excessive caching present	32 threads fit into cache	5922	94	433	47	0
loop_site_163		n/a	0	0	0	0	0
loop_site_170		32 threads fit into cache	54	2	14	1	0

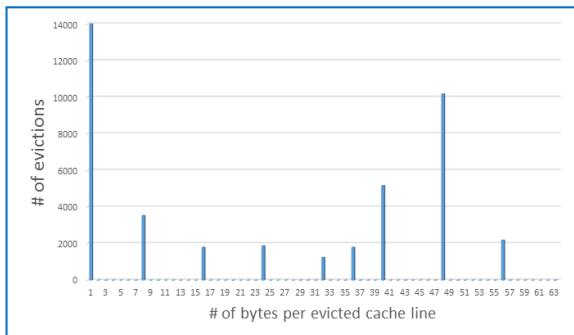
- Suggests next optimization steps
- \* enable by additional option in Project Properties

# MEMORY ACCESS (LATENCY FOCUSED) DEEP DIVE ANALYSIS

\$ export ADVIXE\_EXPERIMENTAL  
=cachesim



# LATENCY/SIMD (AOS/SOA) OPTIMIZATION & ANALYSIS WITH ADVISOR “MEMORY ACCESS PATTERN”



Intel Advisor interface showing memory access analysis results.

**Utilization Report:**

Site Name	Utilization	Recommendations
loop_site_21	99.6%	Cache efficiency for site, 100% - maximum utilization
loop_site_18	50.0%	1 Inefficient memory access patterns present
loop_site_14	50.0%	1 Inefficient memory access patterns present
loop_site_9	6.3%	
loop_site_32	6.3%	
loop_site_7	6.3%	
loop_site_40	n/a	
loop_site_29	n/a	
loop_site_28	n/a	1 Inefficient memory access patterns present
loop_site_27	n/a	1 Inefficient memory access patterns present
loop_site_26	n/a	1 Inefficient memory access patterns present
loop_site_25	n/a	1 Inefficient memory access patterns present
loop_site_23	n/a	1 Proven (real) dependency present
loop_site_15	n/a	

**Memory Access Patterns Report:**

ID	Stride	Type	Source	Nested Function	Variable references	Max. Site Footprint	Modules	Site Name	Access Type
P1	25	Constant stride	lbpBGK.cpp:380		block 0x4ce9d00d allocated at lbpSUB.cpp:78	4MB	gemm-main.exe	loop_site_20	Read
P2	256; 1024	Constant stride	gemm-main.cpp:170			4MB	gemm-main.exe	loop_site_20	Write
P3	1024	Constant stride	gemm-main.cpp:170			4MB	gemm-main.exe	loop_site_20	Write

**Recommendations:**

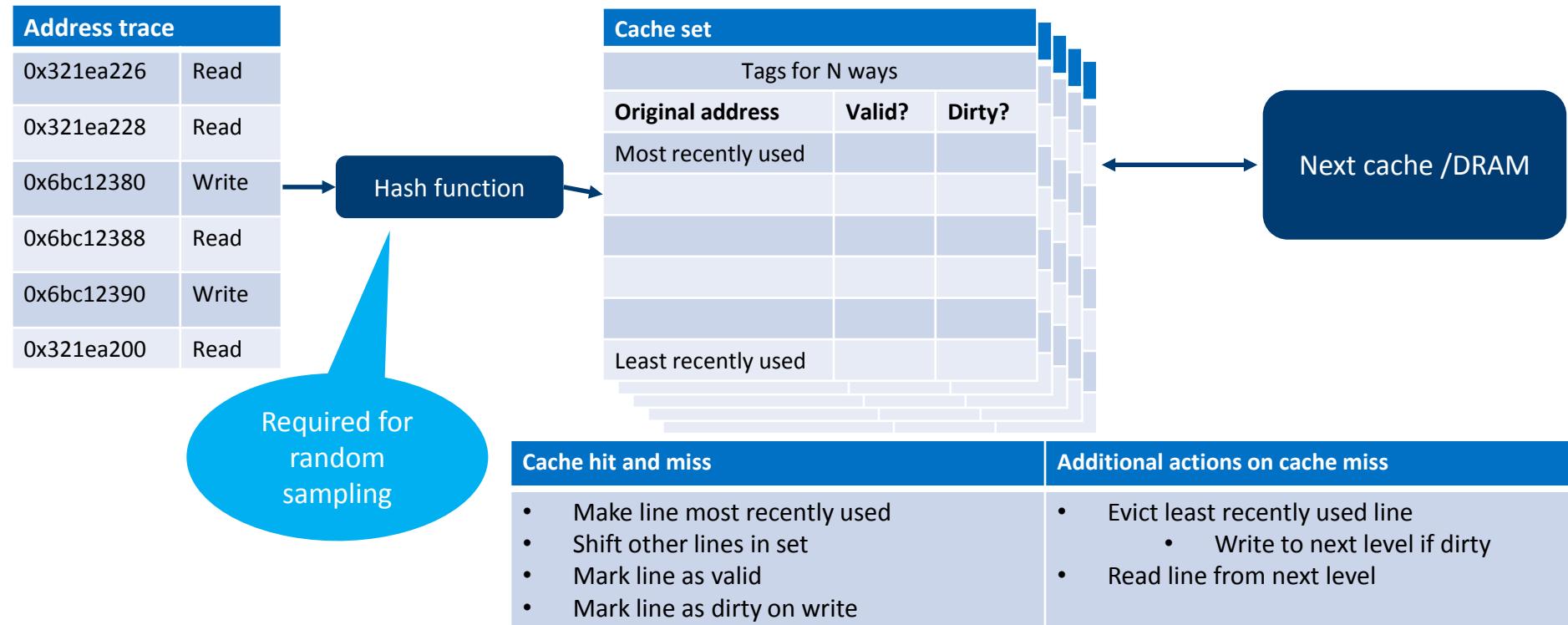
- Fit into cache... 3445340 Cache Line ... 640150 Memory Loads 42893 Cache Misses 10 RFO Cache Misses 22933 Dirty Eviction

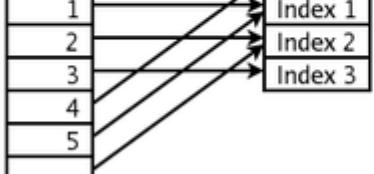
**Code Analysis:**

```

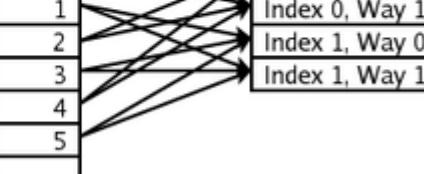
Memory Access Patterns Report Dependencies Report Recommendations
ID Stride Type Source Nested Function Variable references Max. Site Footprint Modules Site Name Access Type
P1 25 Constant stride lbpBGK.cpp:380 block 0x4ce9d00d allocated at lbpSUB.cpp:78 15KB libomp5md.dll; sbe
P2 256; 1024 Constant stride gemm-main.cpp:170
P3 300 Constant stride lbpBGK.cpp:382 block 0x4fe3d56040 allocated at lbpSUB.cpp:247 366KB libomp5md.dll; sbe
380     if(lbphi[11] != 11) {
381         for(l1=0; l1<3*lbsy_nf; l1++)
382             interforce [l1] = 1binterforce[l1*3*lbsy_nf+l1] + postequil*1bbdforce[l1];
    
```

# How cache simulator works?

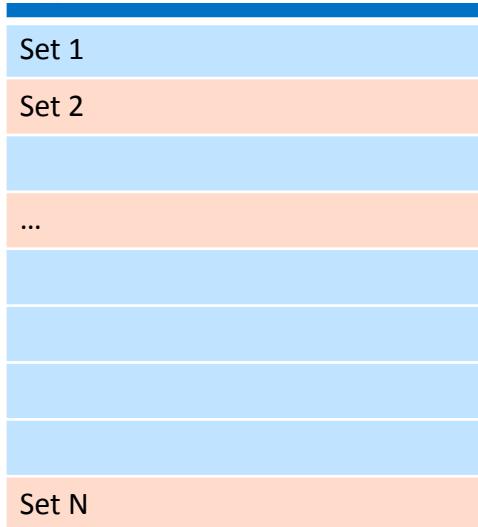




...  
Each location in main memory can be  
cached by just one cache location.



...  
Each location in main memory can be  
cached by one of two cache locations.



# er?

Select some  
sets randomly  
and simulate

Ignore others!

**Scale data to get final result**

Total misses = Simulated misses \* Set count /  
Sampled set count

Random sampling – cache sets are similar, no need to model the whole cache

# Recommendation: enable NT-stores

Intel® Advisor: Cache simulation in frames of MAP analysis (\*)

- Criteria
  - Write-only evictions from cache
  - No streaming stores generated
- Example - stream benchmark:
  - Kernels do not reuse the cache effectively
  - Intel Compiler (since 18.0) generate non-temporal stores for all of the kernels
- \* enable by additional option in Project Properties

The screenshot shows a 'Recommendations' tab in the Intel Advisor interface. A red exclamation mark icon indicates a potential issue: 'ISSUE: POTENTIAL EXCESSIVE CACHING PRESENT'. The confidence level is marked as 'low'. The recommendation is 'Enable non-temporal store'. Below this, there's an example code snippet in Fortran:

```
!DIR$ vector nontemporal
do i=1,N
    arr1(i) = 0
end do
```

Under 'Read More', there are two links: 'nontemporal' and 'Vectorization Resources for Intel® Advisor Users'.

# How to get cache informations out of your laptop ?

- This is possible to extract those information from your computer
  - `cat /sys/devices/system/cpu/cpu[X]/cache/index[Y]/*`
  - You can simulate the behavior of your cache
- You can also retrieve those information from specification
  - Simulate cache behavior for hardware not yet release
  - Optimize your cache blocking for next generation hardware

# FEW COMMENTS



# HOW CAN INTEL® ADVISOR HELP?

What can prevent vectorization or reduce efficiency ?

## 1. Loop-carried dependencies

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

## 4. Outer vs inner loops

**Intel® Advisor MAP analysis**

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

## 5. Indirect memory access or non unit-strides

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

## 2. Pointer aliasing (compiler specific)

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

## 3. Loop structure, boundary condition

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

**Intel® Advisor dependency analysis**

## 6. Small trip counts not multiple of VL

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

**Intel® Advisor Trip count**



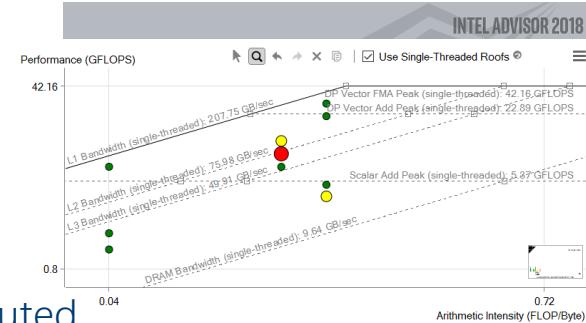
INTEL® ADVISOR 2018

WHAT'S NEW?

# New! Roofline, Faster Analysis & More...

## Intel® Advisor – Vectorization Optimization

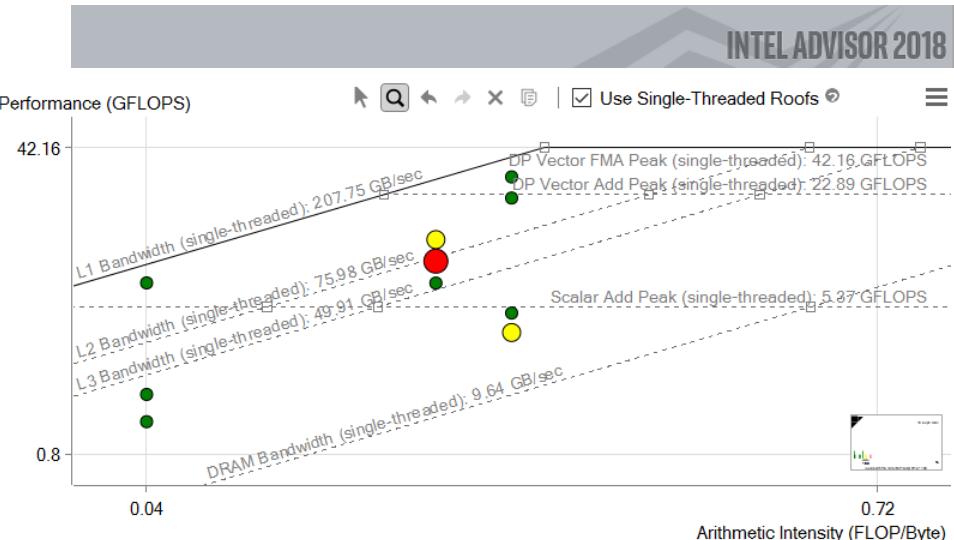
- Roofline analysis helps you optimize effectively
  - Find high impact, but under optimized loops
  - Does it need cache or vectorization optimization?
  - Is a more numerically intensive algorithm a better choice?
- Faster data collection
  - Filter by module - Calculate only what is needed.
  - Track refinement analysis – Stop when every site has executed
- Make better decisions with more data, more recommendations
  - Intel MKL friendly – Is the code optimized? Is the best variant used?
  - Function call counts in addition to trip counts
  - Top 5 recommendations added to summary
  - Dynamic instruction mix – Expert feature shows exact count of each instruction
- Easier MPI launching
  - MPI support in the command line dialog



# Find Effective Optimization Strategies

Intel Advisor: Cache-aware roofline analysis

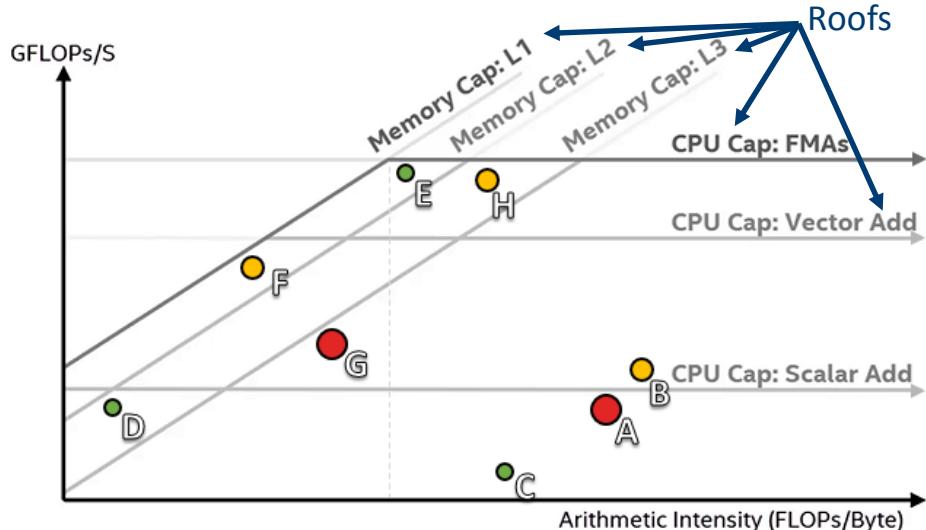
- Roofline Performance Insights
  - Highlights poor performing loops
  - Shows performance “headroom” for each loop
    - Which can be improved
    - Which are worth improving
  - Shows likely causes of bottlenecks
  - Suggests next optimization steps



# Find Effective Optimization Strategies

Intel Advisor: Cache-aware roofline analysis

- Roofs Show Platform Limits
  - Memory, cache & compute limits
- Dots Are Loops
  - Bigger, red dots take more time so optimization has a bigger impact
  - Dots farther from a roof have more room for improvement
- Higher Dot = Higher GFLOPs/sec
  - Optimization moves dots up
  - Algorithmic changes move dots horizontally



Which loops should we optimize?

- A and G are the best candidates
- B has room to improve, but will have less impact
- E, C, D, and H are poor candidates

[Roofline tutorial video](#)



INTEL® ADVISOR 2017

WHAT WAS NEW?

New!

# NEW FOR 2017! AVX-512, FLOPS, & MORE...

## Intel® Advisor – Vectorization Optimization



- Next Gen Intel® Xeon Phi™ Support
- Tune for AVX-512 with or without AVX-512 hardware
- Precise FLOPS calculation

- Enhanced Memory Access Analysis
- Easier Selection of High Impact Loops
- Batch Mode Workflow Saves Time
- Fast Answers with Loop Analytics

Elapsed time: 39.44s    Vectorized    Not Vectorized    FILTER: All Modules ▾ All Sources ▾ Loops ▾ All Threads ▾    OFF Smart Mode ▾    Search

Summary Survey Report Refinement Reports

		Vector Issues	Self Time ▾	Total Time	Type	FLOPS ▾		Why No Vectorization?	Vectorized Loops ▾				Trip Counts	Instr ▾			
						GFLOPS	AI		Vect...	Efficiency	Gain...	VL (...)					
<span style="color: orange;">[loop in S252 at loops90.f:1172]</span>	<span style="color: orange;">[loop in S252 at loops90.f:1172]</span>	<span style="color: orange;">✓</span>	<span style="color: orange;">1 Possib...</span>	<span style="color: orange;">3.080s</span>	<span style="color: orange;">8.5%</span>	<span style="color: orange;">3.080s</span>	<span style="color: orange;">Vectorized Ve...</span>	<span style="color: orange;">0.191</span>	<span style="color: orange;">0.10...</span>	<span style="color: orange;">1 vectori...</span>	<span style="color: orange;">AVX2</span>	<span style="color: orange;">17%</span>	<span style="color: orange;">1.38x</span>	<span style="color: orange;">8</span>	<span style="color: orange;">999; 62; 1</span>	<span style="color: orange;">Divis...</span>	
<span style="color: green;">[loop in S252 at loops90.f:1172]</span>	<span style="color: green;">[loop in S252 at loops90.f:1172]</span>	<span style="color: green;">✓</span>	<span style="color: green;">1 Possib...</span>	<span style="color: green;">2.970s</span>	<span style="color: green;">8.2%</span>	<span style="color: green;">2.970s</span>	<span style="color: green;">Scalar</span>	<span style="color: green;">0.0671</span>	<span style="color: green;">0.0833</span>	<span style="color: green;">vectorizat...</span>					<span style="color: green;">999</span>	<span style="color: green;">Divis...</span>	
<span style="color: green;">[loop in S252 at loops90.f:1172]</span>	<span style="color: green;">[loop in S252 at loops90.f:1172]</span>	<span style="color: green;">✓</span>	<span style="color: green;">0.090s</span>	<span style="color: green;">1</span>	<span style="color: green;">0.090s</span>	<span style="color: green;">1</span>	<span style="color: green;">Vectorized (Bo...</span>	<span style="color: green;">4.333</span>	<span style="color: green;">0</span>	<span style="color: green;">0.1250</span>	<span style="color: green;">AVX2</span>				<span style="color: green;">8</span>	<span style="color: green;">62</span>	<span style="color: green;">FMA</span>
<span style="color: green;">[loop in S252 at loops90.f:1172]</span>	<span style="color: green;">[loop in S252 at loops90.f:1172]</span>	<span style="color: green;">✓</span>	<span style="color: green;">1 Possib...</span>	<span style="color: green;">0.020s</span>	<span style="color: green;">1</span>	<span style="color: green;">0.020s</span>	<span style="color: green;">Scalar</span>			<span style="color: green;">vectorizat...</span>					<span style="color: green;">1</span>	<span style="color: green;">Divis...</span>	
<span style="color: green;">[loop in S2101 at loops90.f:1749]</span>	<span style="color: green;">[loop in S2101 at loops90.f:1749]</span>	<span style="color: green;">✓</span>	<span style="color: green;">2 Possib...</span>	<span style="color: green;">2.580s</span>	<span style="color: green;">7.1%</span>	<span style="color: green;">2.580s</span>	<span style="color: green;">Scalar</span>	<span style="color: green;">0.152</span>	<span style="color: green;">0.0625</span>	<span style="color: green;">vectorizat...</span>							<span style="color: green;">FMA</span>
<span style="color: green;">[loop in S126 at loops90.f:447]</span>	<span style="color: green;">[loop in S126 at loops90.f:447]</span>	<span style="color: green;">✓</span>	<span style="color: green;">1 Assu...</span>	<span style="color: green;">1.068s</span>	<span style="color: green;">1</span>	<span style="color: green;">1.068s</span>	<span style="color: green;">Scalar</span>	<span style="color: green;">0.370</span>	<span style="color: green;">0.1667</span>	<span style="color: green;">vector de...</span>							<span style="color: green;">FMA</span>
<span style="color: green;">[loop in S343 at loops90.f:2300]</span>	<span style="color: green;">[loop in S343 at loops90.f:2300]</span>	<span style="color: green;">✓</span>	<span style="color: green;">1 Assu...</span>	<span style="color: green;">1.020s</span>	<span style="color: green;">1</span>	<span style="color: green;">1.020s</span>	<span style="color: green;">Scalar</span>			<span style="color: green;">vector de...</span>							<span style="color: green;">FMA</span>
<span style="color: orange;">[loop in S353 at loops90.f:2381]</span>	<span style="color: orange;">[loop in S353 at loops90.f:2381]</span>	<span style="color: orange;">✓</span>	<span style="color: orange;">1 Possib...</span>	<span style="color: orange;">0.880s</span>	<span style="color: orange;">1</span>	<span style="color: orange;">0.880s</span>	<span style="color: orange;">Vectorized (Bo...</span>	<span style="color: orange;">2.274</span>	<span style="color: orange;">0</span>	<span style="color: orange;">0.1250</span>	<span style="color: orange;">AVX2</span>	<span style="color: orange;">35%</span>	<span style="color: orange;">2.78x</span>	<span style="color: orange;">8</span>	<span style="color: orange;">62; 4; 1</span>	<span style="color: orange;">FMA</span>	

# NEXT GEN INTEL® XEON PHI™ PROCESSOR SUPPORT

Vectorization Advisor runs on and optimizes for Intel® Xeon Phi Processors

Loops	Vector Issues	Self Time▼	Loop Type	Vectorized Loops			Instruction Set Analysis		
				Vector ISA	Efficiency	Gain Esti...	VL (V...)	Traits	Data Types
= <b>1</b> Loop	3 Possible i...	35.226s <b>5.4X</b>	Vectorized+Threaded (Body; Peeled; Re...	AVX512	-28%	2.21x	8	Divisions; FMA; Gathers	Float32; ...
= <b>1</b> Loop	2 Possible i...	26.025s <b>4.0%</b>	Vectorized (Body)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ... 256/512 AVX2; AVX512ER_512; AVX512...
= <b>1</b> Loop	1 High vecto...	5.876s <b>1</b>	Vectorized (Peeled)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ... 256/512 AVX2; AVX512ER_512; AVX512... Masked Lc
= <b>1</b> Loop	1 High vecto...	3.324s <b>1</b>	Vectorized (Remainder)+Threaded (Open...	AVX512			8	Divisions; Gathers; FMA	Float32; ... 256/512 AVX2; AVX512ER_512; AVX512... Masked Lc
= <b>1</b> Loop		34.599s <b>5.3%</b>	Vectorized (Body; Remainder)	AVX512	-70%	5.64x	8	Divisions; FMA; Square Roots	Float32; ... 256/512 AVX2; AVX512ER_512; AVX512... Masked Lc
= <b>1</b> Loop	1 Possible in...	33.849s <b>5.2%</b>	Vectorized (Body; Peeled; Remainder)	AVX512	-28%	2.24x	8	Divisions; FMA; Gathers	Float32; ... 256/512 AVX2; AVX512ER_512; AVX512... Masked Lc
= <b>1</b> Loop		19.839s <b>3.1%</b>	Vectorized (Body; Remainder)	AVX512	72%	11.48x	16; 8		Float32; ... 256/512 AVX2; AVX512F_512 Masked Lc

Source | Top Down | Loop Assembly | Recommendations | Compiler Diagnostic Details

Issue: Possible inefficient memory access patterns present

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

Recommendation: Confirm inefficient memory access patterns

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

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.

Recommendation: Collect trip counts data

The Survey Report lacks [trip counts](#) data that might generate more precise recommendations. To fix: Run a [Trip Counts analysis](#).

Recommendation: Align data

Recommendation: Add data padding

The [trip count](#) is not a multiple of [vector length](#). To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

Windows® OS	Linux® OS
/Opt-assume-safe-padding	-fno-optimize-sse

## AVX-512 ERI – specific to Intel® Xeon Phi

Efficiency (72%), Speed-up (11.5x), Vector Length (16)

Confidence: Need More Data

Performance optimization problem and advice how to fix it

### Program metrics

Elapsed Time: 142.79s

Vector Instruction Set: AVX, AVX2, AVX512, SSE, SSE2

Number of CPU Threads: 4

### Loop metrics

Total CPU time 454.08s



Time in 88 vectorized loops 41.86s



# START TUNING FOR AVX-512 WITHOUT AVX-512 HARDWARE

## Intel® Advisor - Vectorization Advisor

- Use `-axCOMMON-AVX512 -xAVX` compiler flags to generate both code-paths
  - AVX(2) code path (executed on Haswell and earlier processors)
  - AVX-512 code path for newer hardware
- Compare AVX and AVX-512 code with Intel Advisor

Loops	Self Time	Loop Type	Vectorized Loops					Instruction Set Analysis				Advanced	
			Vect...	Efficiency	Gain...	VL ...	Compiler Es...	Traits	Data T...	Vector W...	Instruction Sets	Vectorization De...	
[-] [loop in s352_at loopstl.cpp:5939]	0,641s I	Vectorized (Body)	AVX2	-54%	2,15x	4	2,15x	FMA; Inserts	Float32	128	AVX; FMA		
[+] [loop in s352_at loopstl.cpp:5939]	n/a	Remainder [Not Executed]				4		FMA					
[+] [loop in s352_at loopstl.cpp:5939]	0,641s I	Vectorized (Body)	AVX2			4	2,15x	Inserts; FMA					
[+] [loop in s352_at loopstl.cpp:5939]	n/a	Vectorized (Body) [Not Executed]	AVX512			16	3,20x	Gathers; FMA					
[+] [loop in s352_at loopstl.cpp:5939]	n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	2,70x	Gathers; FMA					
[-] [loop in s125_A\$omp\$parallel_for@ ...]	0,496s I	Vectorized Versions	AVX2	-100%	13,54x	8	<13,54x	FMA; NT-stores					
[+] [loop in s125_A\$omp\$parallel_for...]	n/a	Peeled [Not Executed]				8		FMA					
[+] [loop in s125_A\$omp\$parallel_for...]	n/a	Remainder [Not Executed]				8		FMA					
[+] [loop in s125_A\$omp\$parallel_for...]	0,465s I	Vectorized (Body)	AVX2		8	13,54x							
[+] [loop in s125_Z\$omp\$parallel_for...]	n/a	Vectorized (Peeled) [Not Executed]	AVX512		16	6,77x		FMA					
[+] [loop in s125_Z\$omp\$parallel_for...]	n/a	Vectorized (Body) [Not Executed]	AVX512		32	30,61x		NT-store					
[+] [loop in s125_Z\$omp\$parallel_for...]	n/a	Vectorized (Remainder) [Not Executed]	AVX512		16	9,78x		FMA					

Inserts (AVX2) vs.  
Gathers (AVX-512)

Speed-up estimate:  
13.5x (AVX2) vs.  
30.6x (AVX-512)

# PRECISE REPEATABLE FLOPS METRICS

## Intel® Advisor – Vectorization Optimization

- FLOPS by loop and function
- All recent Intel processors  
(not co-processors)
- Instrumentation (count FLOPs) plus sampling (time with low overhead)
- Adjusted for masking with AVX-512 processors

INTEL ADVISOR 2017

Function Call Sites and Loops	FLOPS							
	GFLOPS	AI	L1 GB/s	GFLOP	FLOP Per Iteration	L1 GB	L1 Bytes Per Iteration	
[loop in matvec at Multiply.c:69]	0.8260	0.1633	5.0586	3.0720	32	18.8160	196	
[loop in matvec at Multiply.c:60]	0.9120	0.1633	5.5853	3.0720	32	18.8160	196	
[loop in matvec at Multiply.c:69]	1.2480	0.2500	4.9920	1.3440	4	5.3760	16	
[loop in matvec at Multiply.c:60]	1.5920	0.2500	6.3699	1.3440	4	5.3760	16	
[loop in matvec at Multiply.c:69]	3.0550	0.2500	12.2205	0.0960	16	0.3840	64	
[loop in matvec at Multiply.c:60]	6.2820	0.2500	25.1279	0.0960	16	0.3840	64	

# ENHANCED MEMORY ACCESS ANALYSIS

Are you bandwidth or compute limited?

- Measure Footprint
  - Compare to cache size  
Does it fit in L2 cache?
- Variable References
  - Map data to variable names  
for easier analysis
- Gather/Scatter
  - Detect unneeded  
gather/scatters that reduce  
performance

Site Location	Loop-Carried Dependencies	Strides Distribution ▲	Access Pattern	Max. Site Footprint
[loop in s4117_at loopstl.cpp:76..]	No information available	50% / 50% / 0%	Mixed strides	192B
[loop in s442_at loopstl.cpp:6815]	No information available	56% / 0% / 44%	Mixed strides	256B
[loop in s272_at loopstl.cpp:3447]	No information available	60% / 0% / 40%	Mixed strides	320B

Memory Access Patterns Report		Dependencies Report		Recommendations			
ID	Stride	Type	Source	Nested Function	Variable references	Access Footprint	Module
P2		Gather stride	loopstl.cpp:3450		a, c, d	320B	lcd...
		3448 if (e[i_] >= *t)					
		3449 {					
		3450 a[i_] += c_[i_] * d_[i_];					
		3451 b[i_] += c_[i_] * c_[i_];					
		3452 }					

Module: lcd_cxx!0x432340						
Address	Line	Assembly	Physical Stride	Op	Mask	Notes
0x43265a	3450	vgatherdpsz (\$r8,\$zmm8,4), \$k1, \$zmm2		bit*		
0x432661	3403	leaq (\$r13,\$rsi,1), \$r8		bit*		
0x432666	3450	vgatherdpsz (\$r9,\$zmm8,4), \$k3, \$zmm1		bit*		
0x43266d	3450	vgatherdpsz (\$r8,\$zmm8,4), \$k2, \$zmm4		bit*		
0x432674	3450	vfmadd213ps \$zmm2, \$zmm1, \$zmm4				
0x43267a	3403	leaq (\$rcx,\$rsi,1), \$r9		bit*		
0x43267e	3450	vmovupsz \$zmm4, (\$rsi,\$rdx,1){\$k6}		bit*		

**Gather/scatter details**

**Pattern:** "Unit"

Instruction accesses values in contiguous memory throughout the loop:

- unit stride within instruction
- stride between iterations = vector length

Horizontal stride (bytes): 4

Vertical stride (bytes): 64

Mask is constant

Mask: [1111111111111111]

Active elements in the mask: 100,0%

# EASIER SELECTION OF HIGH IMPACT LOOPS

Smart mode helps you prioritize quickly

- Smart Mode On
  - Simplified column layout – heuristics prioritize high impact loops
  - Slider adjusts filter threshold to hide low impact loops
- Smart Mode Off
  - Full data available for expert

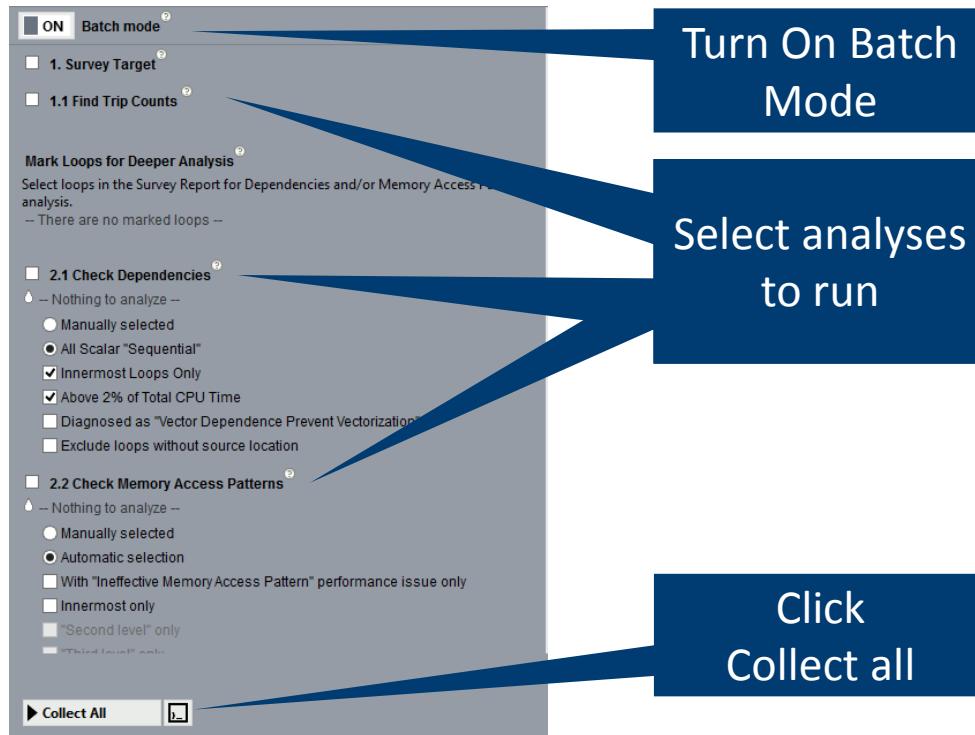
The screenshot shows the Intel Advisor 2017 interface with the following details:

- Toolbar:** Includes "Elapsed time: 39.44s", "Vectorized", "Not Vectorized", a slider labeled "Loops above 2.0%", and a search icon.
- Filter Options:** "FILTER: User Modules, All Sources, Loops And Functions, All Threads".
- Summary Tab:** Active tab.
- Survey Report Tab:** Available tab.
- Refinement Reports Tab:** Available tab.
- Data Table:** Shows analysis results for three loops. The first loop is highlighted in blue and has a checkmark in the "Vector Issues" column. The second and third loops also have checkmarks in the same column.

	Vector Issues	Self Time	Total Time	Loop Heig...	Type	Vect... ISA	GFLOPS	
[loop in S252 at loops90.f:1172]	<input checked="" type="checkbox"/>	1 Possi...	2.970s	3.080s	0	Vectorized Ve...	AVX2 0.0671	
[loop in S252 at loops90.f:1172]	<input checked="" type="checkbox"/>	1 Possib...	2.970s	2.970s	7.6%	0	Scalar	0.0671
[loop in S2101 at loops90.f:1749]	<input type="checkbox"/>	2 Possib...	2.580s	2.580s	6.6%	0	Scalar	0.1521

# BATCH MODE WORKFLOW SAVES TIME

## Intel® Advisor - Vectorization Advisor



Turn On Batch Mode

Select analyses to run

Click Collect all

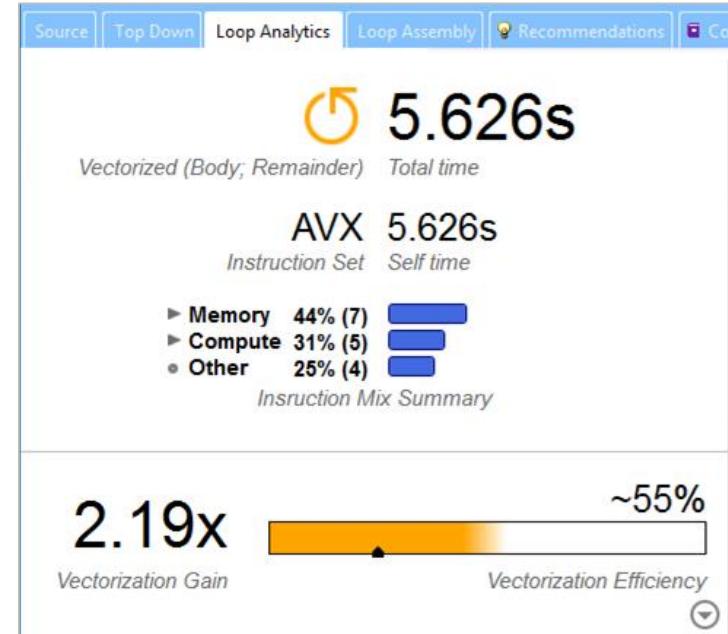
- Start several analyses with a single click
- Contains pre-selected criteria for advanced analyses

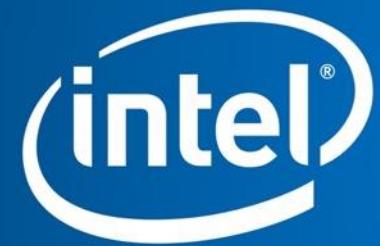
# LOOP ANALYTICS

Get the data you need to plan optimization

- Fast Answers

- How much performance can be gained?
- Is it worth optimizing?
- Static analysis estimate:  
Is it memory bound or compute bound?





Software