

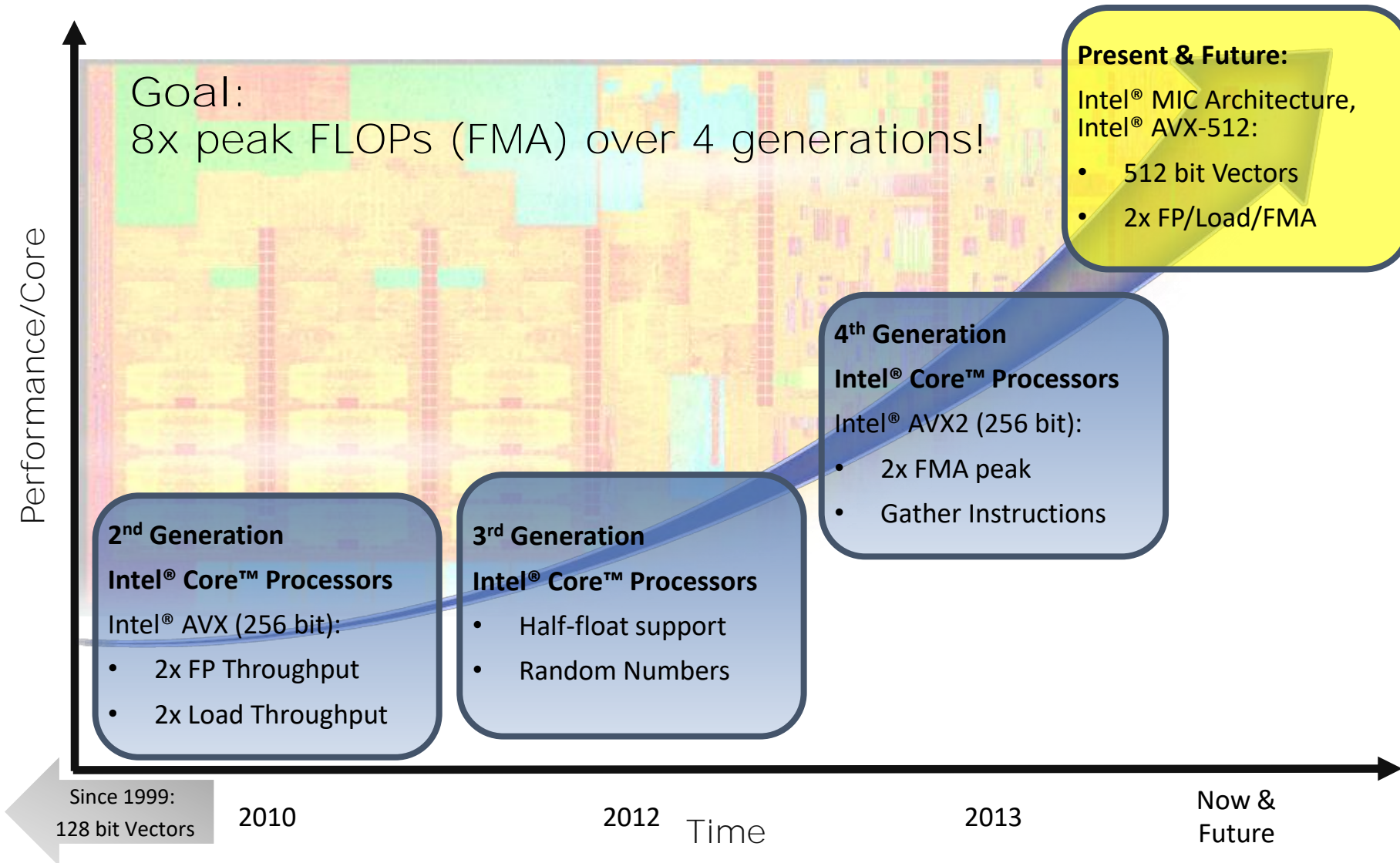


VISUALIZING AND FINDING OPTIMIZATION OPPORTUNITIES WITH INTEL[®] ADVISOR ROOFLINE FEATURE

AGENDA

- **Vectorization is becoming more and more important**
- What is the theoretical roofline model ?
- How is it implemented in Advisor ?
- Some examples

EVOLUTION OF SIMD FOR INTEL PROCESSORS



AGENDA

- Vectorization is becoming more and more important
- **What is the theoretical roofline model ?**
- How is it implemented in Advisor ?
- Some examples

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

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

Arithmetic Intensity (Flops/Bytes) 

PLATFORM PEAK FLOPS

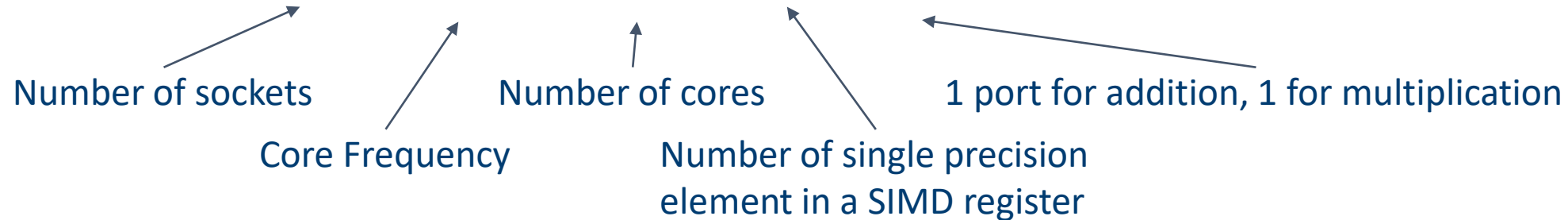
How many floating point operations per second

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

- Theoretical value can be computed by specification

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

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



- 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
PEAK BW = 2 x 1.866 x 8 x 4 = 119 GB/s

Number of sockets
↗

Memory Frequency
↗

Byte per channel
↘

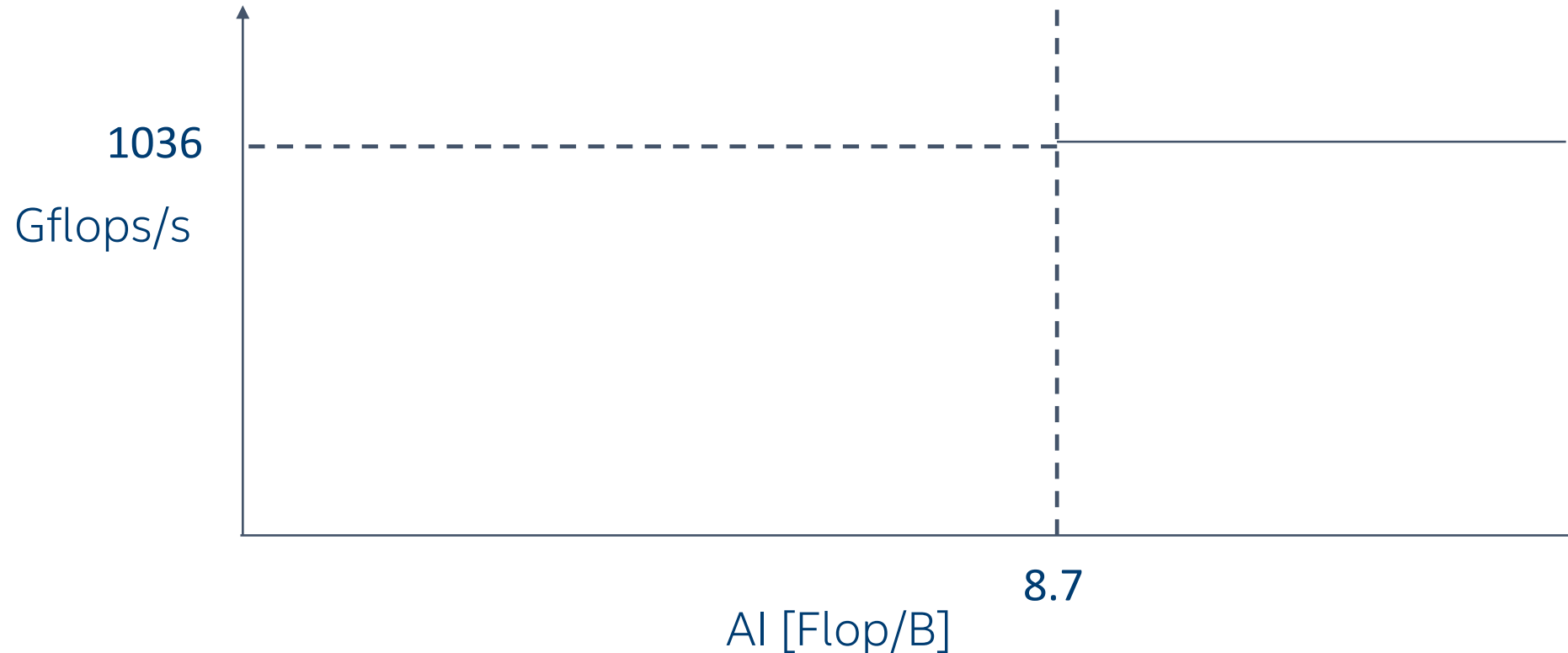
Number of mem channels
↘
- More realistic value can be obtained by running Stream
= ~ 100 GB/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

DRAWING THE ROOFLINE

Defining the speed of light

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

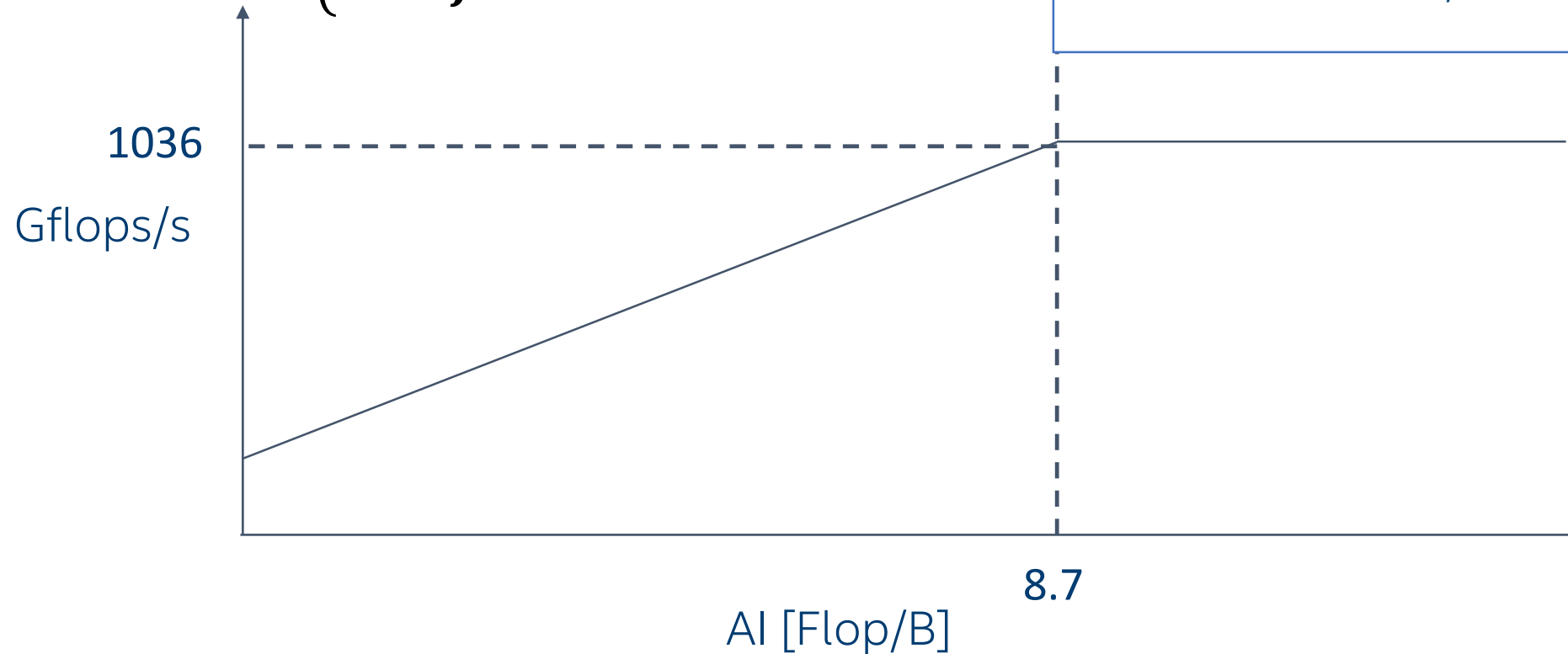
2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s



DRAWING THE ROOFLINE

Defining the speed of light

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



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

- $AI = \frac{2N^3}{16N^2} = \frac{1}{8}N$

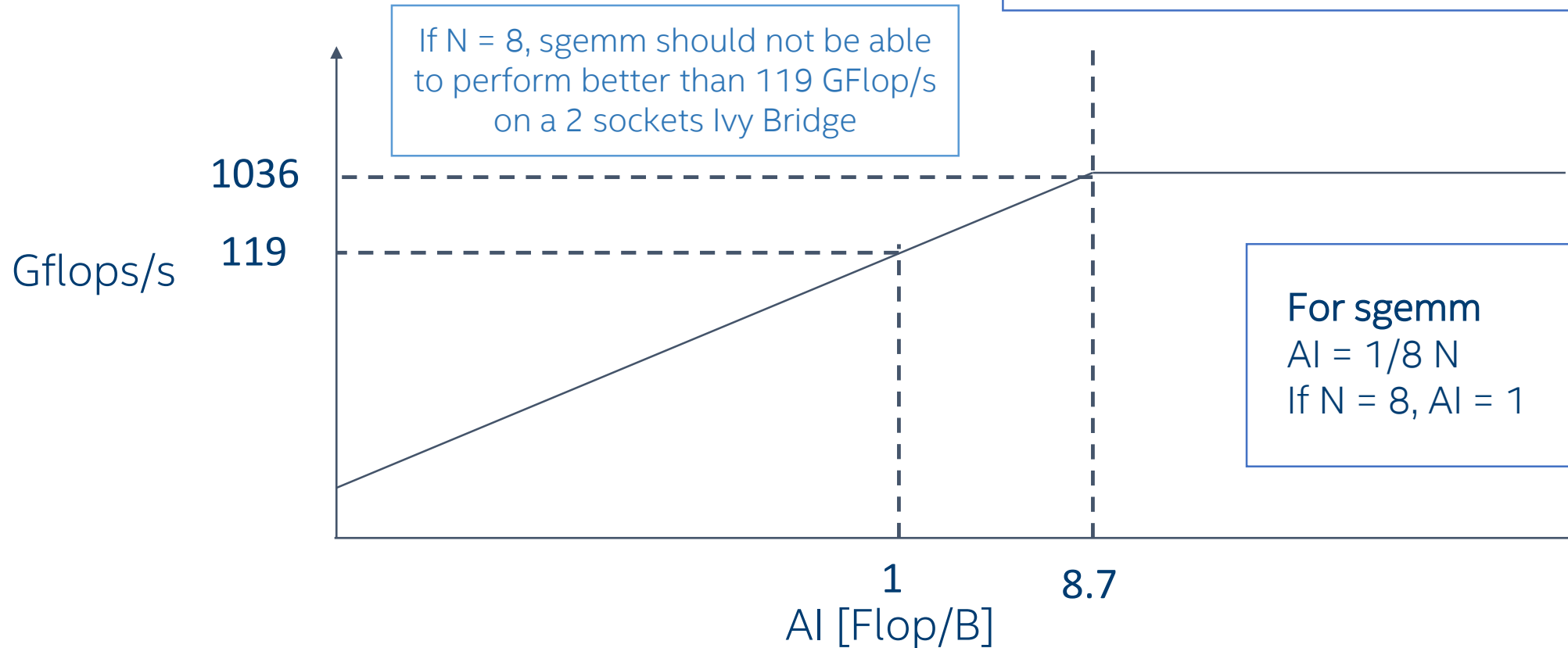
COMPUTE THE MAXIMUM PERFORMANCE

BW * Arithmetic Intensity

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

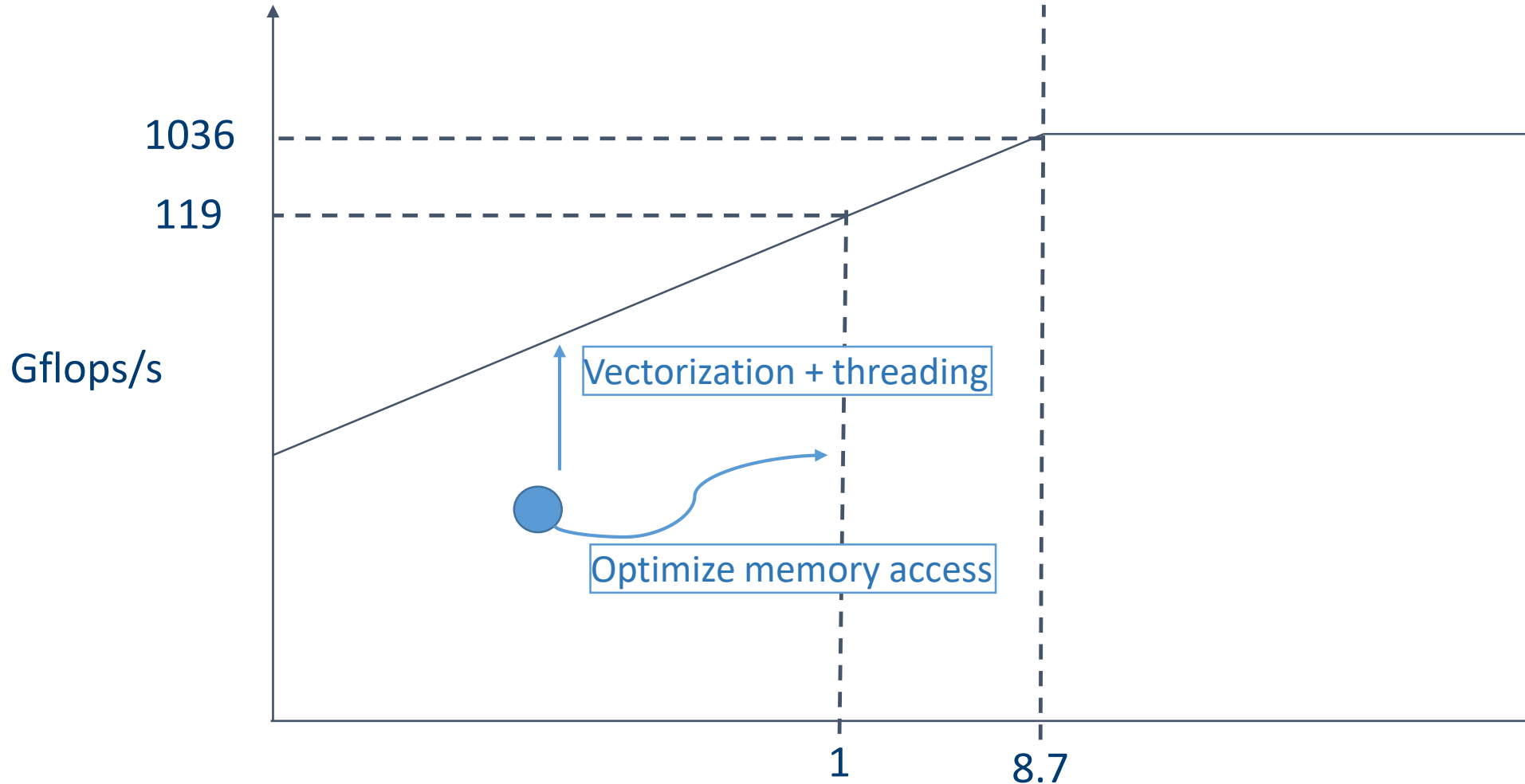
2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

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



AND NOW?

How to get better performance?



AGENDA

- Vectorization is becoming more and more important
- What is the theoretical roofline model ?
- **How is it implemented in Advisor ?**
- Some examples

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

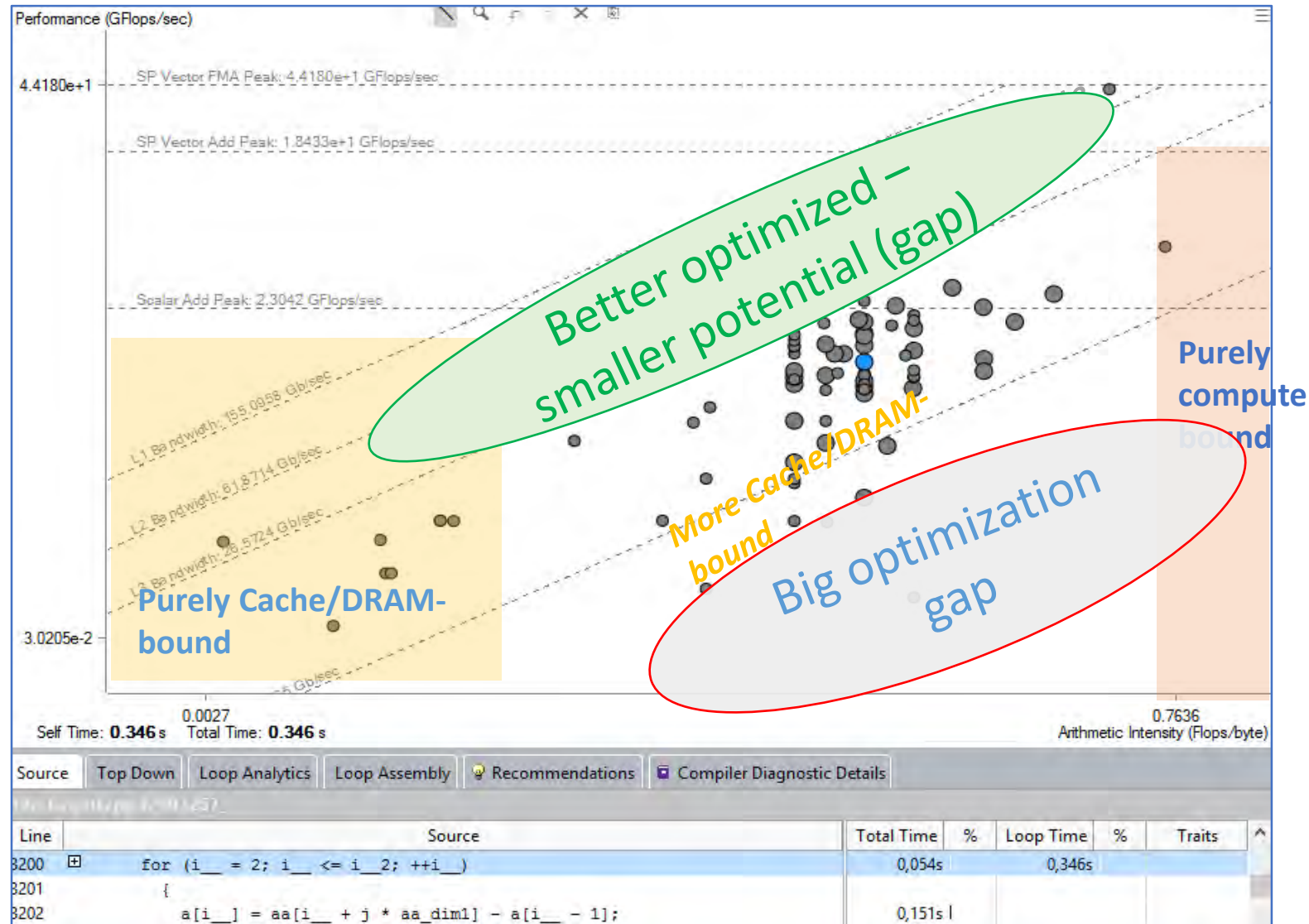
How does it work ?

- 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



AGENDA

- Vectorization is becoming more and more important
- What is the theoretical roofline model ?
- How is it implemented in Advisor ?
- **Some examples**

ROOFLINE MODEL AND COMPILER OPTIMIZATIONS

ROOFLINE MODEL AND OPTIMIZATIONS

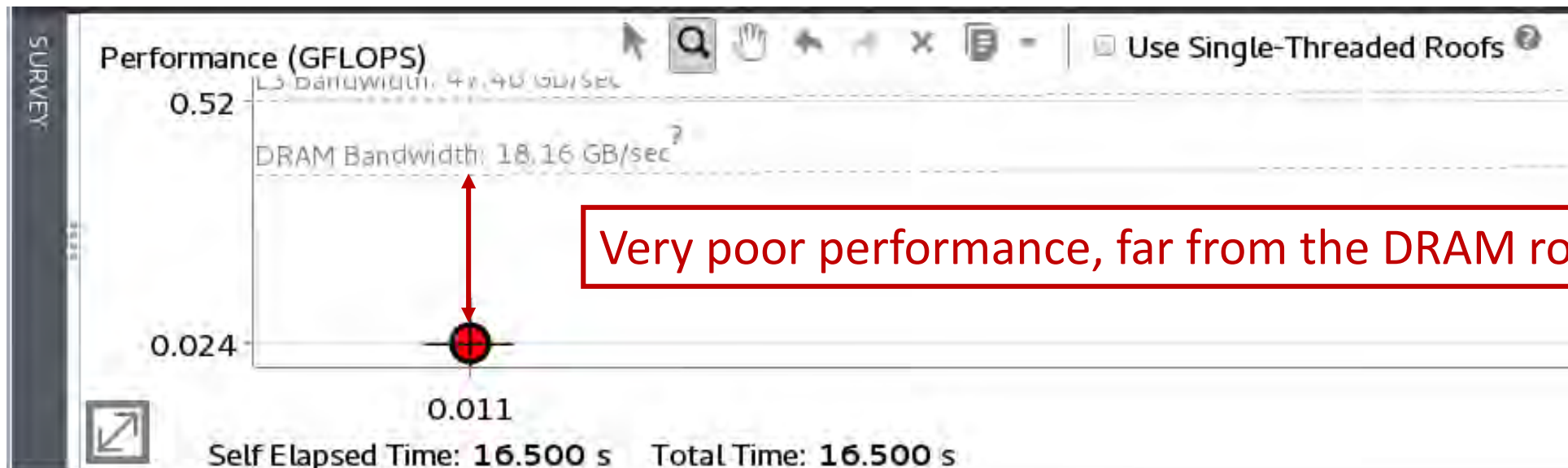
- Matrix/matrix addition

```
void addition(float* a, float* b, float* c, int size){  
    int i, j;  
    for(j=0; j<size; j++){  
        for(i=0; i<size; i++){  
            c[i*size + j] = a[i*size + j]+b[i*size + j];  
        }  
    }  
}
```

- Let's have a look at the roofline model

ROOFLINE MODEL AND OPTIMIZATIONS

- Compilation with -O1



ROOFLINE MODEL AND OPTIMIZATIONS

- Lets look at the Memory Access Pattern Analysis

The screenshot shows the Intel VTune Memory Access Pattern Analysis tool interface. At the top, there are filters for 'Elapsed time: 16,71s', 'Vectorized', 'Not Vectorized', 'MKL', and 'FILTER: User Modules'. Below these are tabs for 'Summary', 'Survey & Roofline', and 'Refinement Reports'. The main table displays memory access patterns with columns: Site Location, Loop-Carried Dependencies, Strides Distribution, Access Pattern, Max. Site Footprint, Site Name, and Recommendations. A row shows 'loop in addition at main.c:...' with a '60% / 40% / 0%' stride distribution and a recommendation of '1 Inefficient memory access patterns present'.

Below the main table is a 'Memory Access Patterns Report' section with a table of detailed access patterns. The first entry, P1, is highlighted with a red box and a red annotation. The code snippet for P1 is as follows:

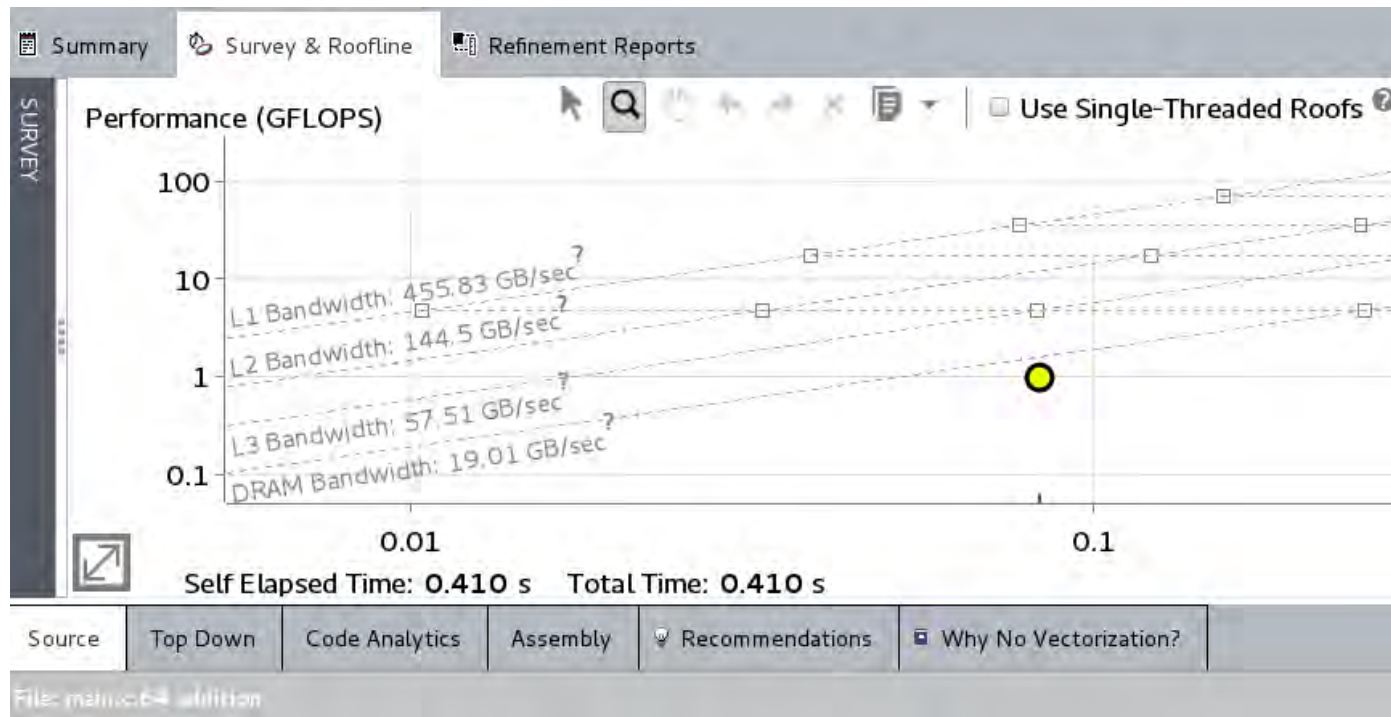
```
64     for(j=0; j<size; j++){
65         for(i=0; i<size; i++){
66             c[i*size + j] = a[i*size + j]+b[i*size + j];
67         }
68     }
```

The annotation 'Constant stride found !!! Looks like loops should be reversed' is written in red text next to the code. The table entry for P1 shows a 'Constant stride' of 2000, source 'main.c:66', variable references 'block 0x7fdc54c0f010, block 0x7fdc55bd2010', a 'Max. Site Footprint' of 14MB, and an 'Access Type' of 'Read'.

ID	Stride	Type	Source	Nested Function	Variable references	Max. Site Footprint	Modules	Site Name	Access Type
P1	2000	Constant stride	main.c:66		block 0x7fdc54c0f010, block 0x7fdc55bd2010	14MB	addition.exe	loop_site_4	Read
P2	2000	Constant stride	main.c:66		block 0x7fdc53ccc010	14MB	addition.exe	loop_site_4	Write

ROOFLINE MODEL AND OPTIMIZATIONS

- Compilation with -O3



```
62 void addition(float* a, float* b, float* c, int size){
63     int i, j;
64     for(j=0; j<size; j++){
65         for(i=0; i<size; i++){
```

[loop in addition at main.c:64]
Vectorized AVX loop processes Float32 data type(s)
Loop was interchanged; loop was unrolled by 4

[loop in addition at main.c:64]
Scalar loop. Not vectorized: inner loop was already vectorized
Loop was interchanged; remainder loop

VECTORIZATION OF LOOP CARRIED DEPENDENCY

VECTORIZATION OF LOOP CARRIED DEPENDENCY

- Loop carried dependency

```
void addition(float* a, float* b, float* c, int size){  
    int i, j;  
    for(i=0; i<size; i++){  
        for(j=pad; j<size; j++){  
            c[i*size + j] = a[i*size + j]+c[i*size + j-pad];  
        }  
    }  
}
```

VECTORIZATION OF LOOP CARRIED DEPENDENCY



VECTORIZATION OF LOOP CARRIED DEPENDENCY

- Loop carried dependency

The screenshot shows the Intel Advisor interface. At the top, there are filters for 'Vectorized' (checked), 'Not Vectorized', and 'MKL'. Below the filters, there are tabs for 'Summary', 'Survey & Roofline', and 'Refinement Reports'. The 'Survey & Roofline' tab is active. The main table displays a list of function call sites and loops. The first row is highlighted, showing a loop in 'main.c:38' with a self time of 0.192s and a total time of 10.231s. The 'Why No Vectorization?' column for this loop indicates 'vector dependence prev ...'. Below the table, there are tabs for 'Source', 'Top Down', 'Code Analytics', 'Assembly', 'Recommendations', and 'Why No Vectorization?'. The 'Recommendations' tab is active, showing an issue titled 'Issue: Assumed dependency present'. The issue description states: 'The compiler assumed there is an anti-dependency (Write after read - WAR) or a true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.' A sub-point 'Confirm dependency is real' is listed, with the text: 'There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a Dependencies analysis.'

Elapsed time: 10,41s

Vectorized Not Vectorized MKL FILTER: User Modules All Sources All Threads

Summary Survey & Roofline Refinement Reports

Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Loop Height	Type	Why No Vectorization?	Vector ISA	Efficiency	Gain Estima...	VL (Vecto...	Se
[loop in main at main.c:38]	2 Assumed depe...	0,192s	10,231s	0	Scalar	vector dependence prev ...					

Source Top Down Code Analytics Assembly Recommendations Why No Vectorization?

All Advisor-detectable issues: [C++](#) / [Fortran](#)

Issue: Assumed dependency present

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

Confirm dependency is real

There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a Dependencies analysis.

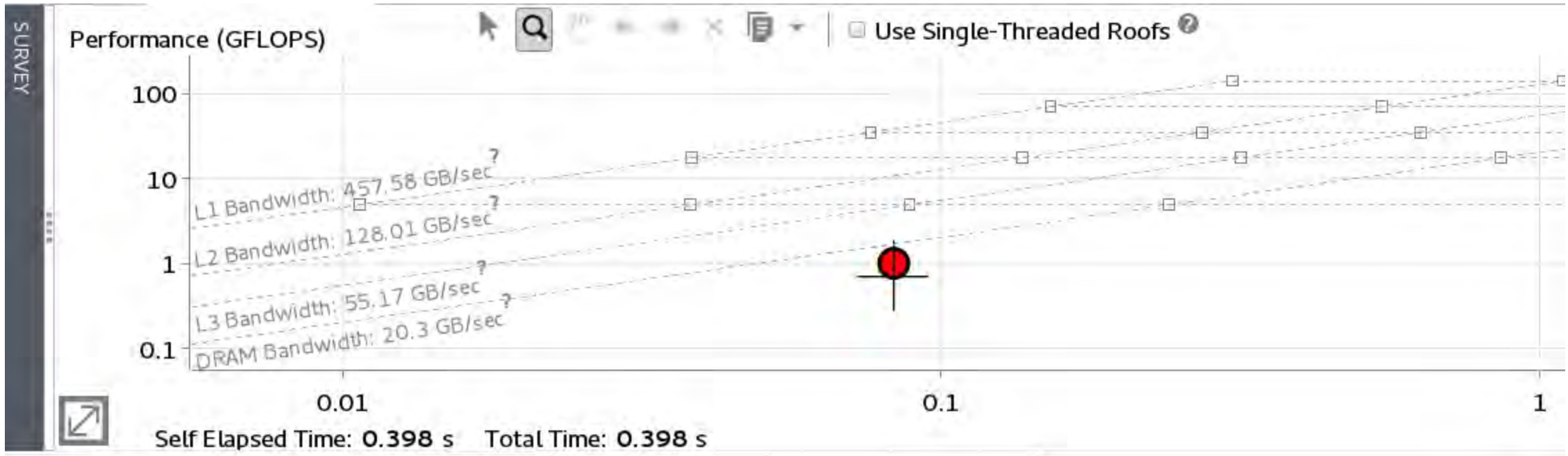
VECTORIZATION OF LOOP CARRIED DEPENDENCY

- Loop carried dependency

```
void addition(float* a, float* b, float* c, int size){  
    int i, j;  
    for(i=0; i<size; i++){  
        #pragma omp simd safelen(4)  
        for(j=pad; j<size; j++){  
            c[i*size + j] = a[i*size + j]+c[i*size + j-pad];  
        }  
    }  
}
```

In this case, we assume that $\text{pad} \geq 4$

VECTORIZATION OF LOOP CARRIED DEPENDENCY



VECTORIZATION OF LOOP CARRIED DEPENDENCY

Safelen was 4

Elapsed time: 0,57s

Vectorized

Not Vectorized

MKL

FILTER:

All Modules

All Sources

Loops And Functions

All Threads

OFF

Summary

Survey & Roofline

Refinement Reports

ROOFLINE

Function Call Sites and Loops

Performance Issues

Self Time

Total Time

Type

W. N.

Vectorized Loops

Vector ...

Efficiency

Gain E ...

VL (Ve ...

FLOPS

Self GFLOPS

[loop in addition at main3.c:67]

0,398s

0,398s

Vectorized (Body: ...

AVX

97%

3,89x

4

1,003

[loop in addition at main3.c:67]

0,398s

0,398s

Vectorized (Body)

AVX

4

1,001

[loop in addition at main3.c:67]

0,000s

0,000s

Vectorized (Remain...

AVX

4

[loop in initialize at main3.c:57]

3 Data type conv...

0,030s

0,050s

Inside vectorized

[loop in initialize at main3.c:57]

2 Data type conv...

0,020s

0,060s

Inside vectorized

[Import thunk rand]

0,012s

0,012s

Function

[loop in initialize at main3.c:57]

2 Data type conv...

0,012s

0,052s

Inside vectorized

_start

0,000s

0,560s

Function

initialize

0,000s

0,050s

Inlined Function

Source

Top Down

Code Analytics

Assembly

Recommendations

Why No Vectorization?

VECTORIZATION OF FUNCTION CALL

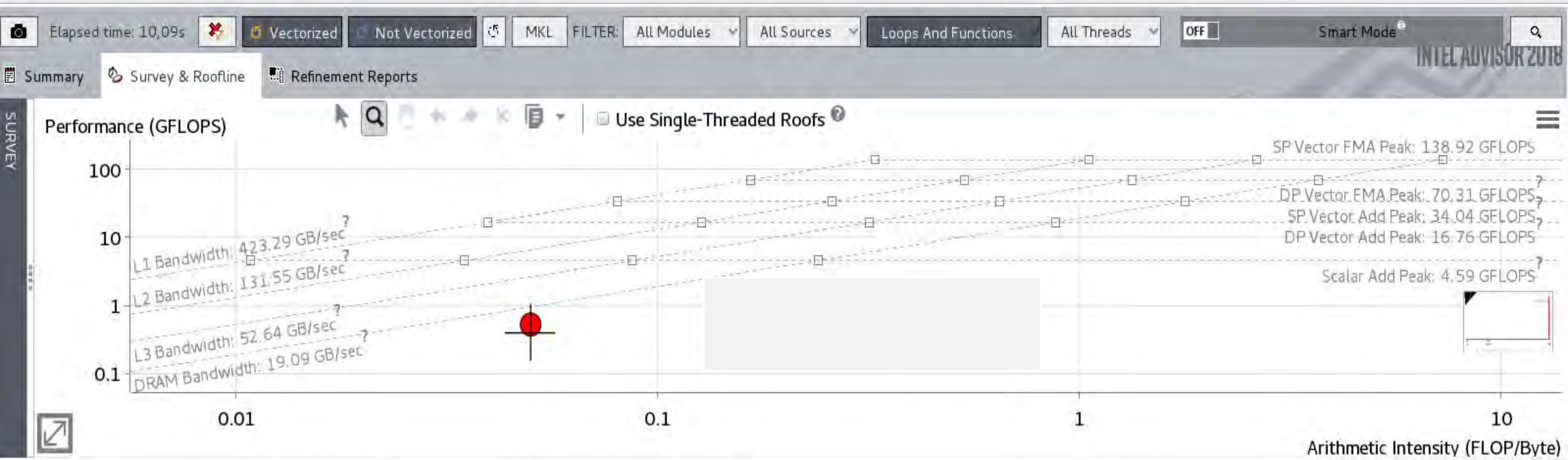
VECTORIZATION OF A FUNCTION CALL WITH OMP

- Function call inside of a loop can kill the vectorization

```
for(int i=0; i<SIZE; i++){  
    for(int j=0; j<SIZE; j++){  
        single_line_addition(a, c, i*SIZE + j);  
    }  
}
```

```
//function is defined in another compilation unit  
void single_line_addition(float* a, float* c, int ind){  
    c[ind] = a[ind]+c[ind];  
}
```

VECTORIZATION OF A FUNCTION CALL WITH OMP



VECTORIZATION OF A FUNCTION CALL WITH OMP

Elapsed time: 10,09s

Vectorized

Not Vectorized

MKL

FILTER:

User Modules

All Sources

All Threads

ON

5,0%

Loops slave

Summary

Survey & Roofline

Refinement Reports

ROOFLINE

+ -

Function Call Sites and Loops

	<div><div></div><div>Performance Issues</div></div>	Self Time	Total Time	Loop Height	Type	Why No Vectorization?	Vector ISA	Efficiency	Gain Estima...	VL (Vecto...	Self GFI
<div><div></div><div></div><div>[loop in main at main.c:38]</div></div>	<div><div></div><div>2 Assumed depe ...</div></div>	2,252s	9,856s	0	Scalar	<div><div></div><div>vector dependence prev ...</div></div>					
<div><div></div><div></div><div>[loop in main at main.c:37]</div></div>	<div><div></div><div>1 Assumed depe ...</div></div>	0,052s	9,908s	1	Scalar	<div><div></div><div>vector dependence preve ...</div></div>					
<div><div></div><div></div><div>[loop in main at main.c:34]</div></div>	<div><div></div><div>1 Assumed depe ...</div></div>	0,000s	9,908s	2	Scalar	<div><div></div><div>vector dependence preve ...</div></div>					

Source

Top Down

Code Analytics

Assembly

Recommendations

Why No Vectorization?

All Advisor-detectable issues: [C++](#) | [Fortran](#)

Issue: Assumed dependency present

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



Confirm dependency is real

There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a Dependencies analysis.

Issue: User function call(s) present

User-defined functions in the [loop body](#) are preventing the compiler from vectorizing the loop.



Vectorize user function(s) inside loop

Advisor tells you that this pattern can be a problem and proposes a solution

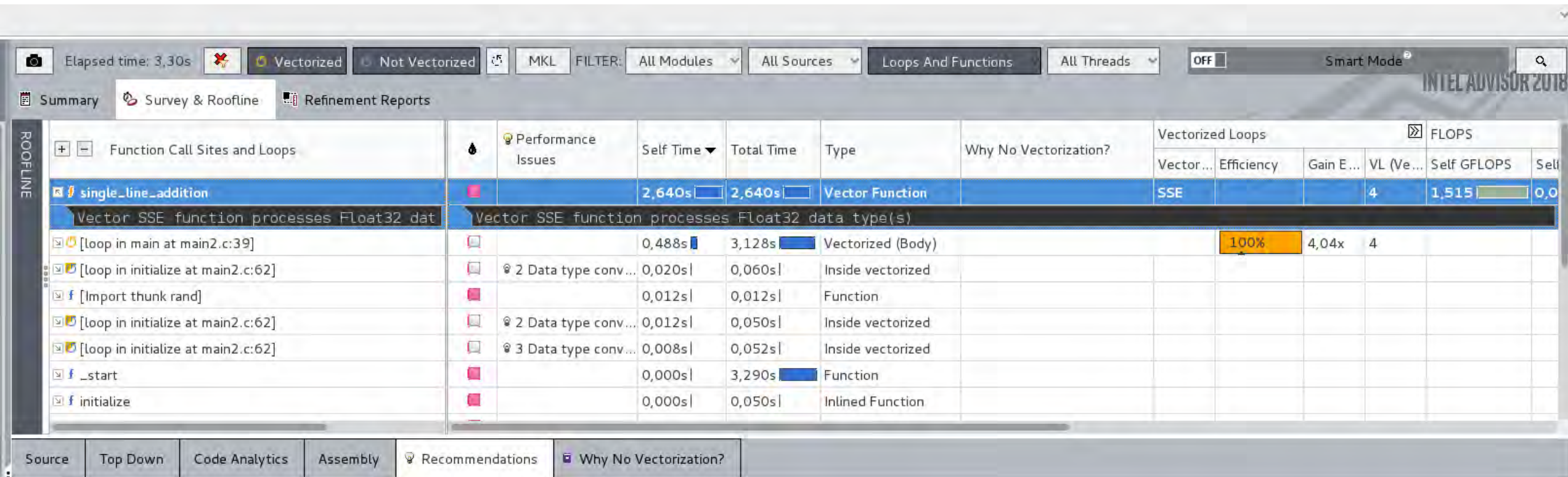
VECTORIZATION OF A FUNCTION CALL WITH OMP

- Omp declare simd

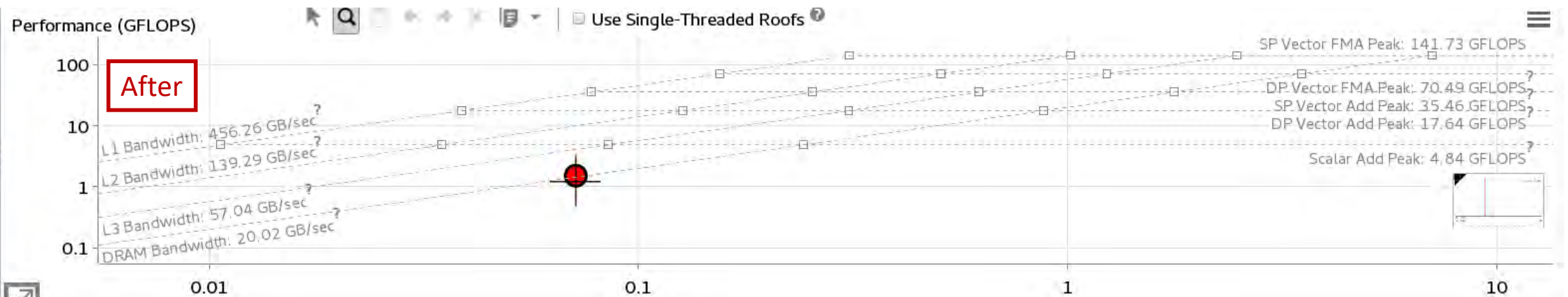
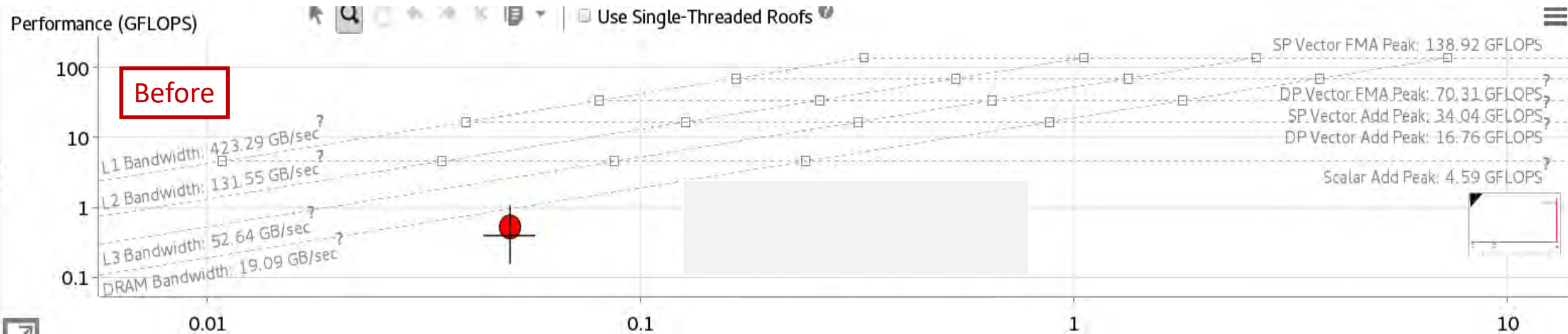
```
for(int i=0; i<SIZE; i++){  
    #pragma omp simd  
    for(int j=0; j<SIZE; j++){  
        single_line_addition(a, c, i*SIZE + j);  
    }  
}
```

```
#pragma omp declare simd uniform(a, c) linear(ind)  
void single_line_addition(float* a, float* c, int ind);
```


VECTORIZATION OF A FUNCTION CALL WITH OMP



VECTORIZATION OF A FUNCTION CALL WITH OMP



Legal Disclaimer & Optimization Notice

- INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
- 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.
- Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804



Software