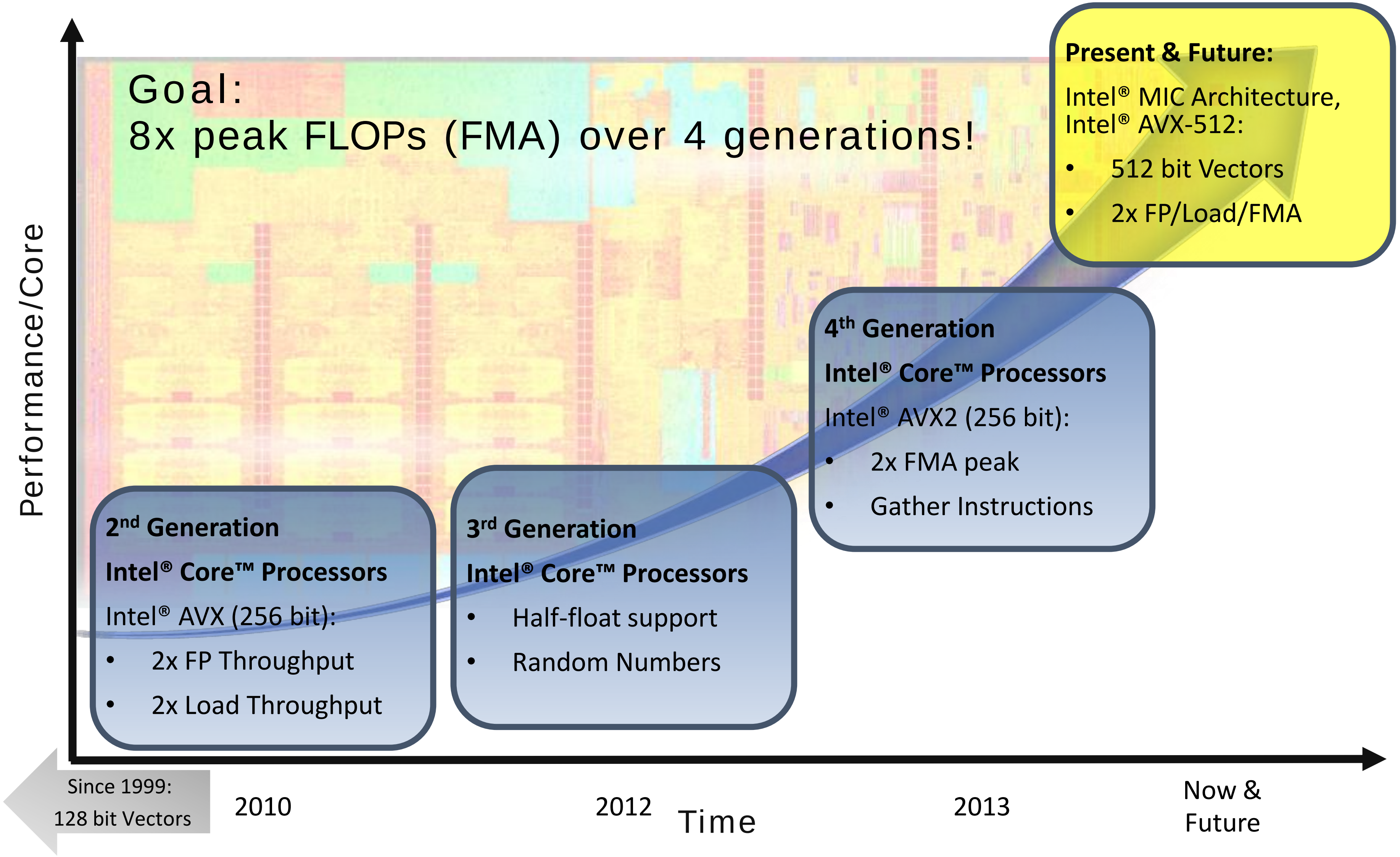
VISUALIZING AND FINDING OPTIMIZATION OPPORTUNITIES WITH INTEL® ADVISOR ROOFLINE FEATURE

Intel Software Developer Conference – London, 2017

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

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

Arithmetic Intensity
(Flops/Bytes)

PLATFORM PEAK FLOPS

How many floating point operations per second

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

- Theoretical value can be computed by specification
  Example with 2 sockets Intel® Xeon® Processor E5-2697 v2
  PEAK FLOP = 2 x 2.7 x 12 x 8 x 2 = **1036.8 Gflop/s**

Number of sockets

Core Frequency

Number of cores

Number of single precision element in a SIMD register

1 port for addition, 1 for multiplication

- 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

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

- 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

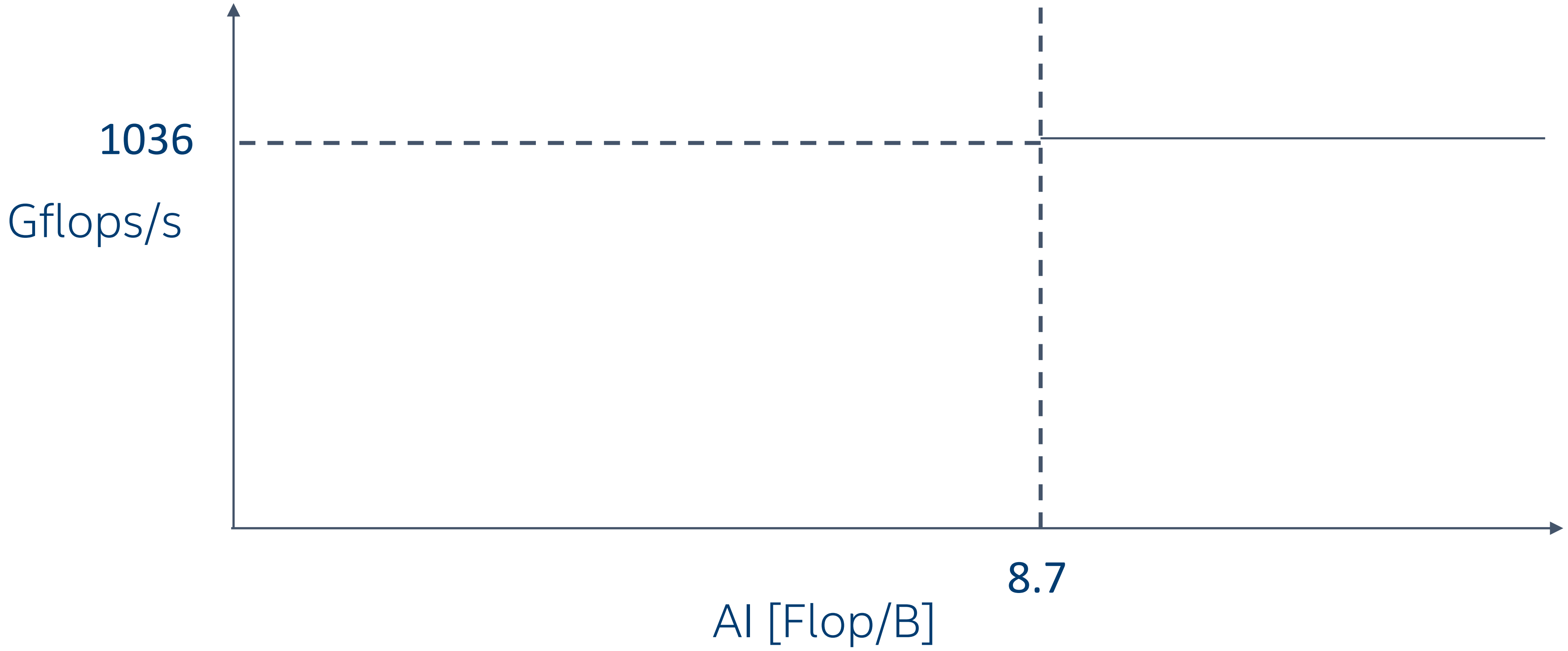
- More realistic value can be obtained by running Stream
  =~ 100 GB/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

DRAWING THE ROOFLINE

Defining the speed of light

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

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

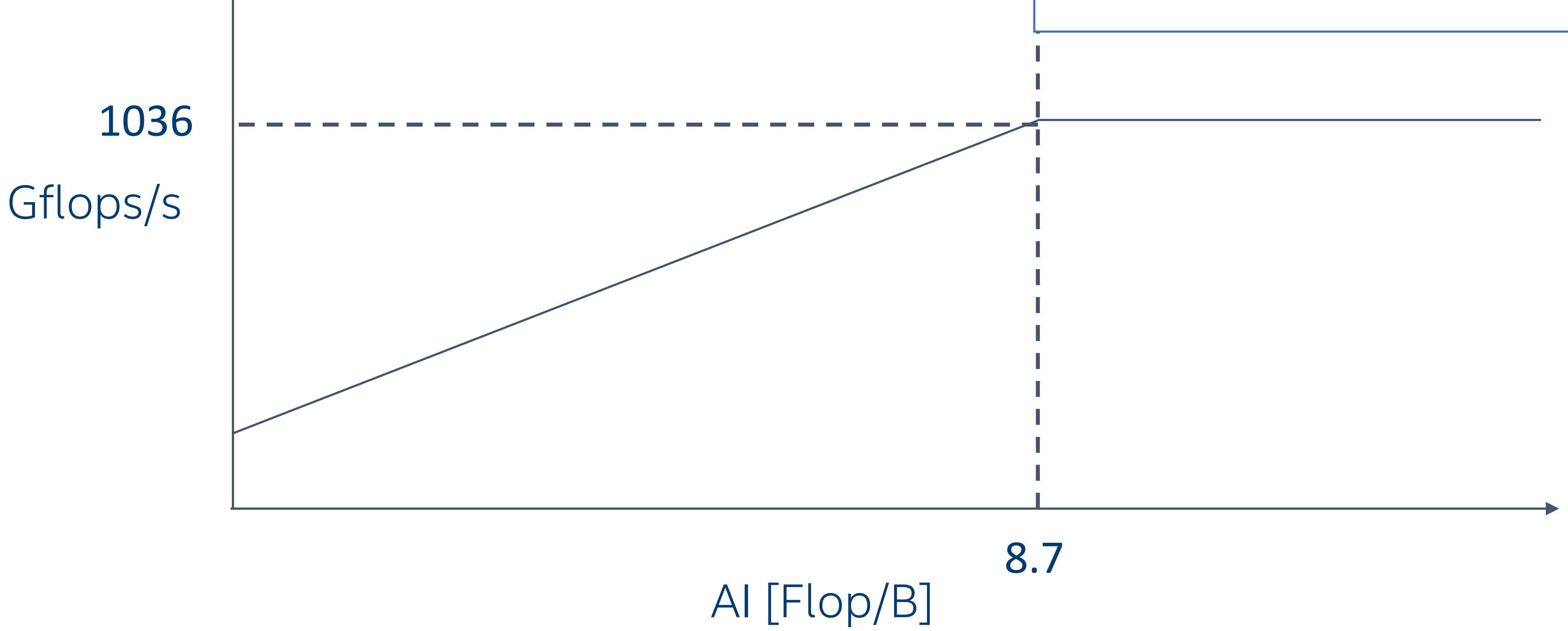


DRAWING THE ROOFLINE

Defining the speed of light

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

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

  # mul = N * N * N

  #Read = 3 * N * N * 4 bytes

  #Write = N * N * 4 bytes

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

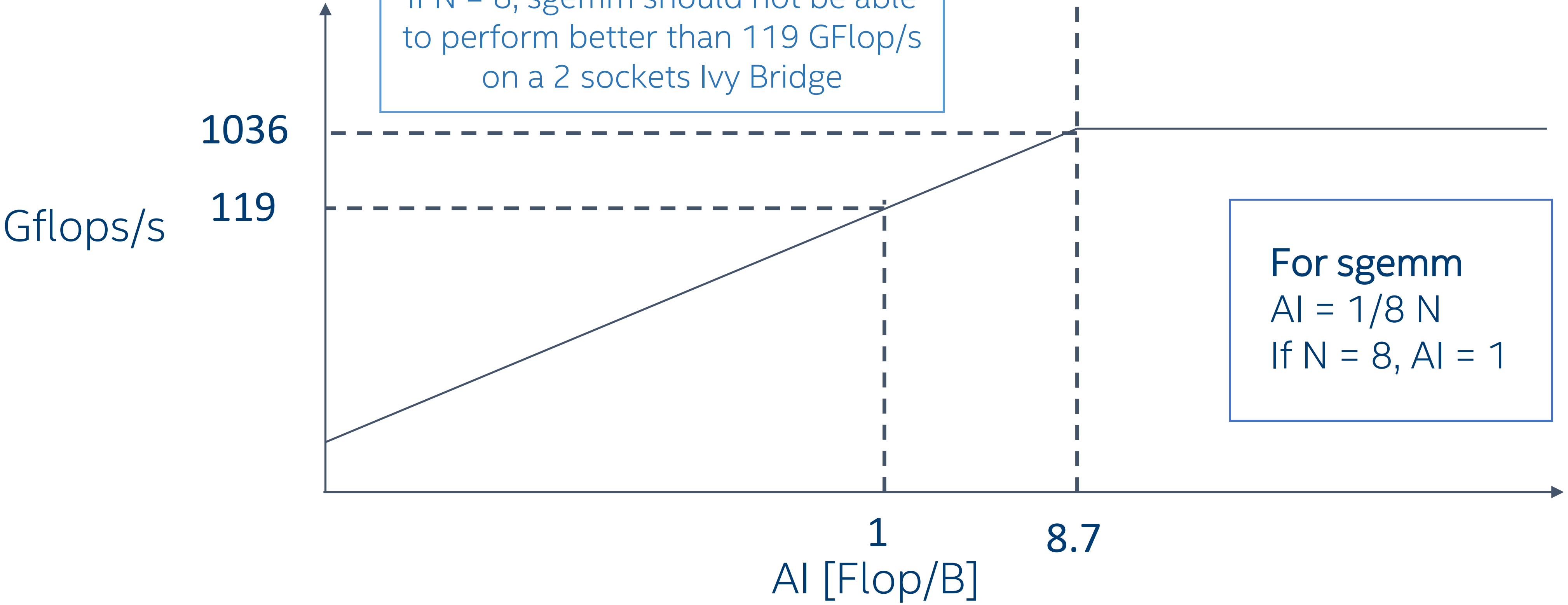
COMPUTE THE MAXIMUM PERFORMANCE

BW * Arithmetic Intensity

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

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

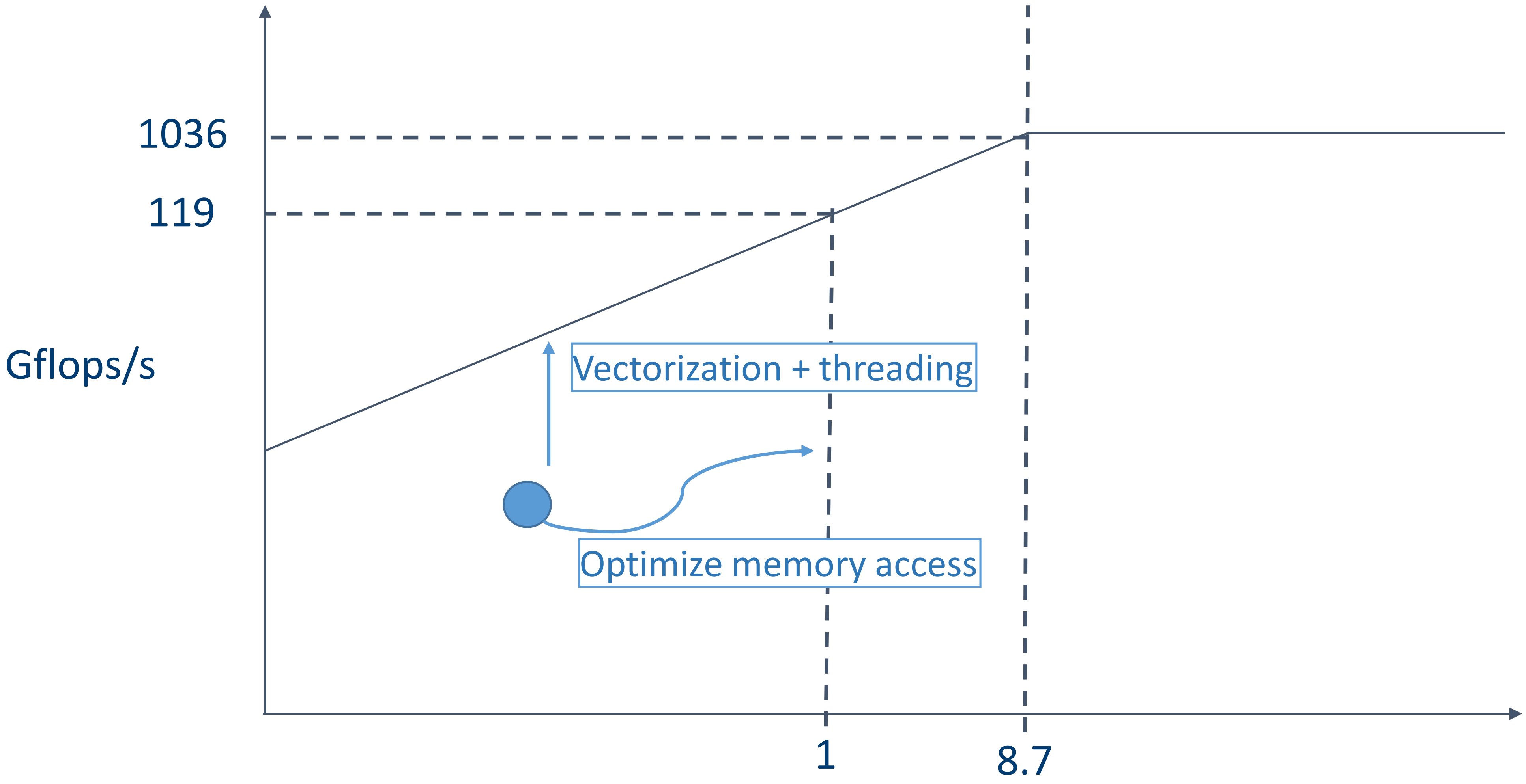
If N = 8, sgemm should not be able to perform better than 119 GFlop/s on a 2 sockets Ivy Bridge



For sgemm
AI = 1/8 N
If N = 8, AI = 1

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

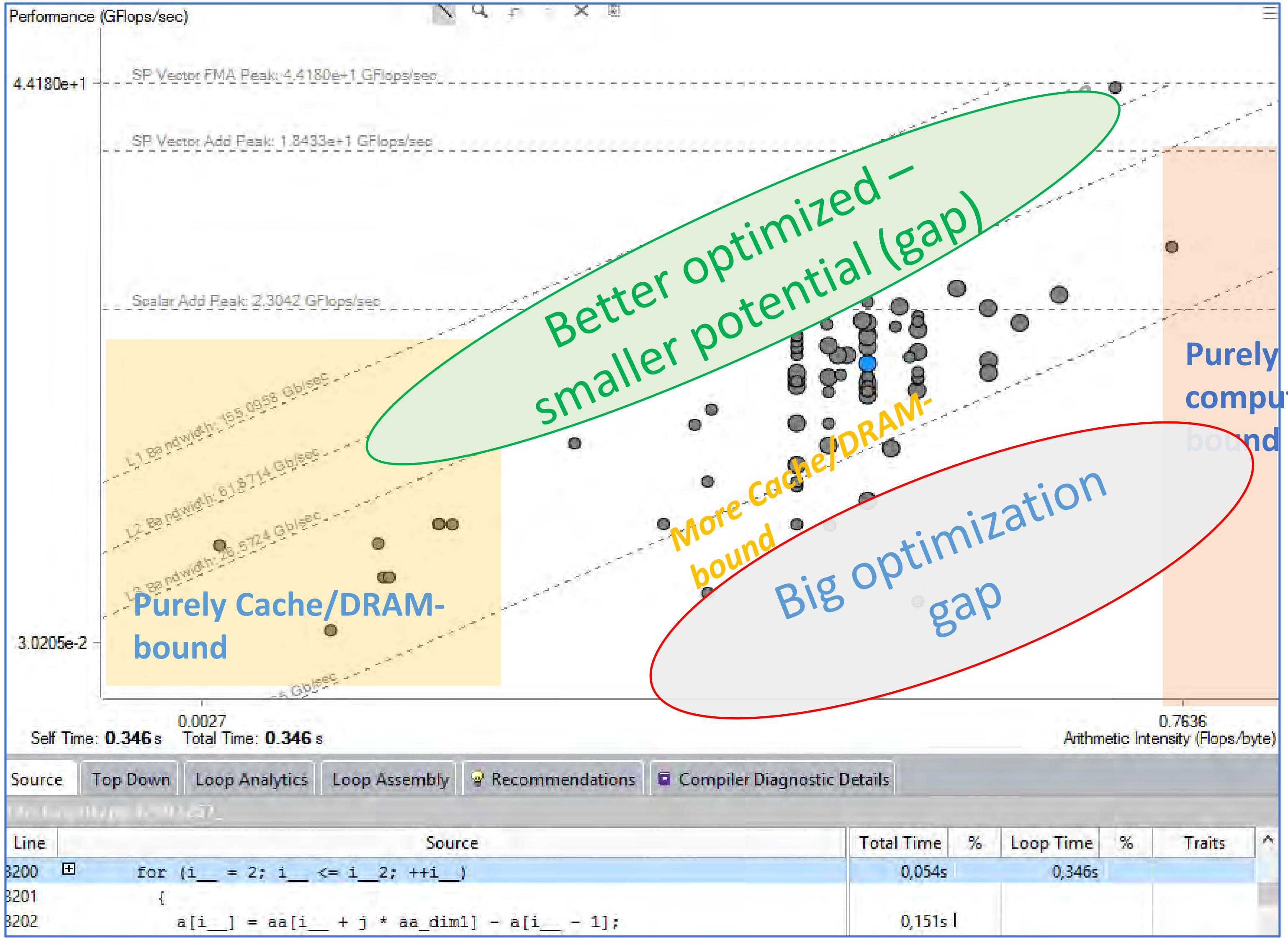
How does it work ?

- 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

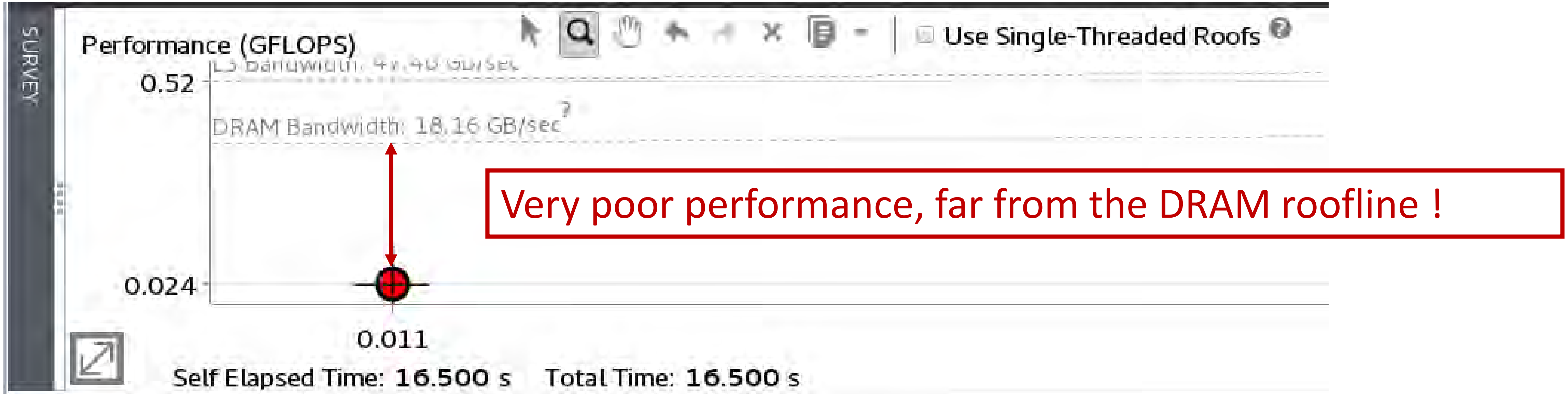
- Matrix/matrix addition

```
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

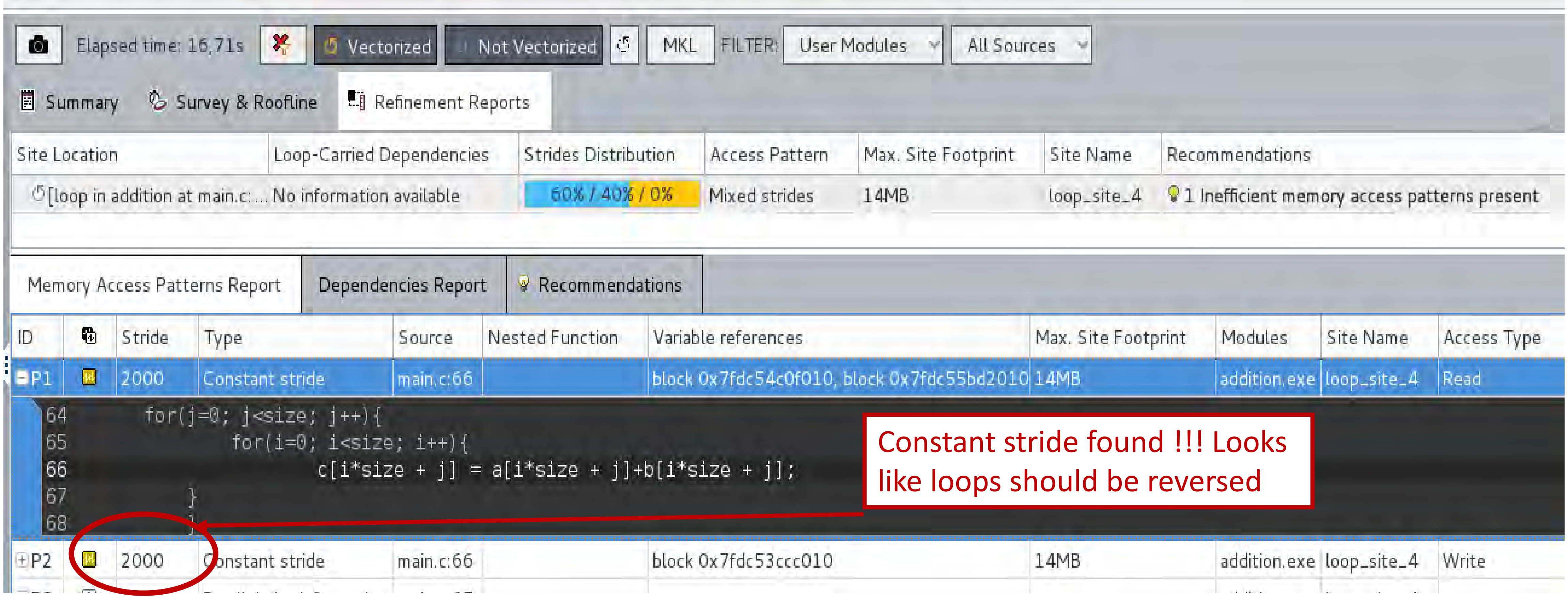
ROOFLINE MODEL AND OPTIMIZATIONS

- Compilation with –O1



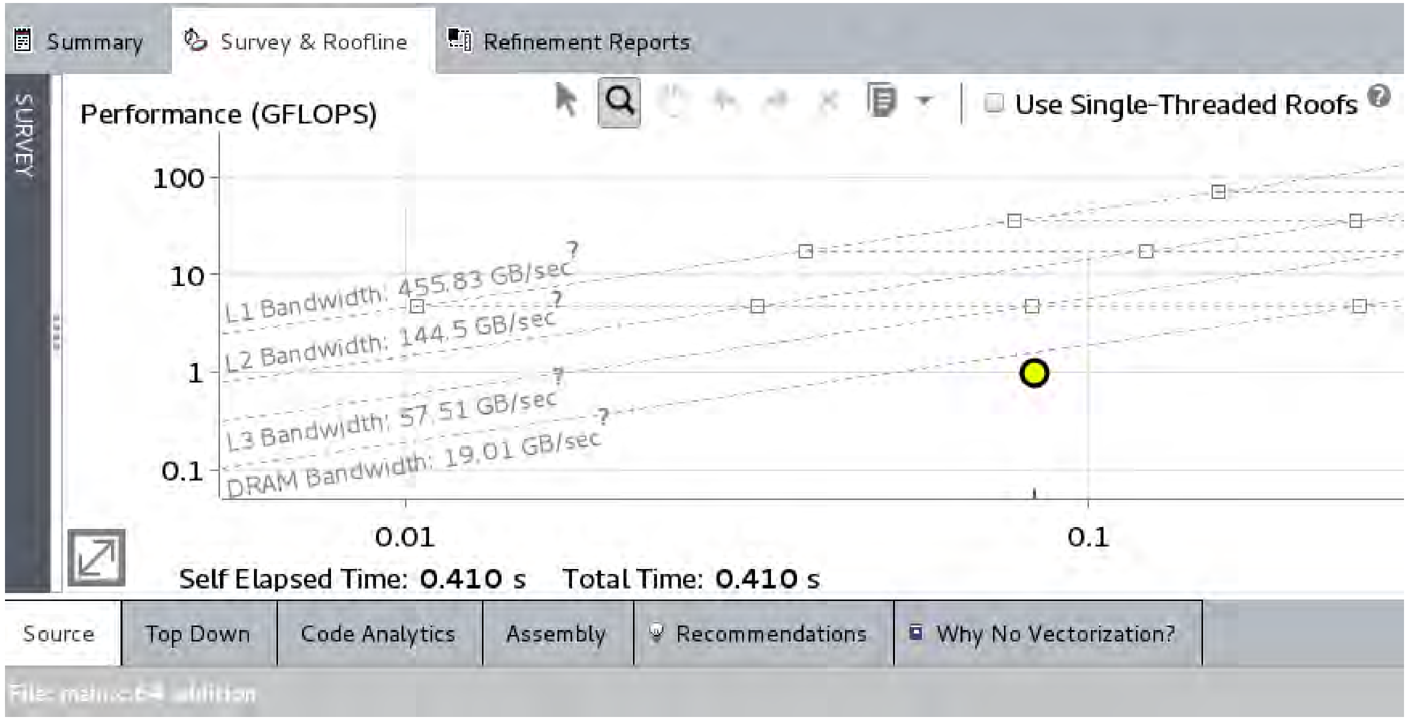
ROOFLINE MODEL AND OPTIMIZATIONS

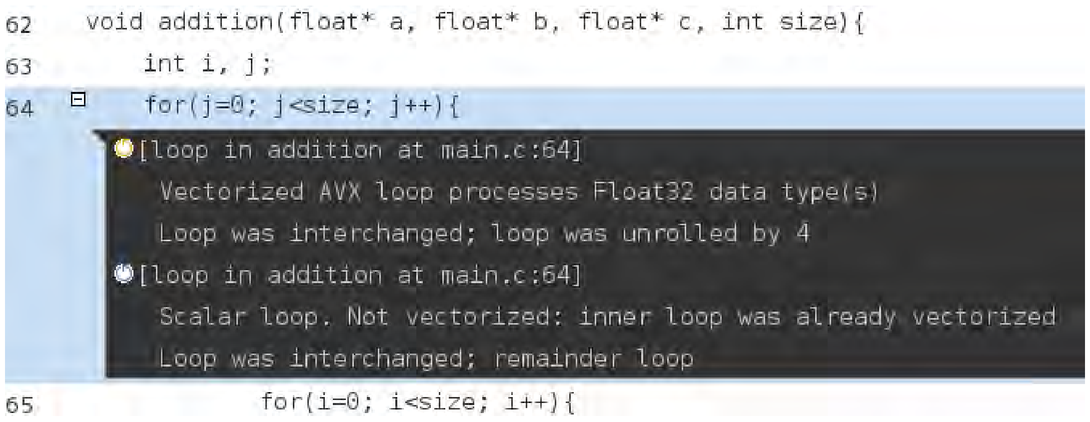
- Lets look at the Memory Access Pattern Analysis



ROOFLINE MODEL AND OPTIMIZATIONS

- Compilation with –O3





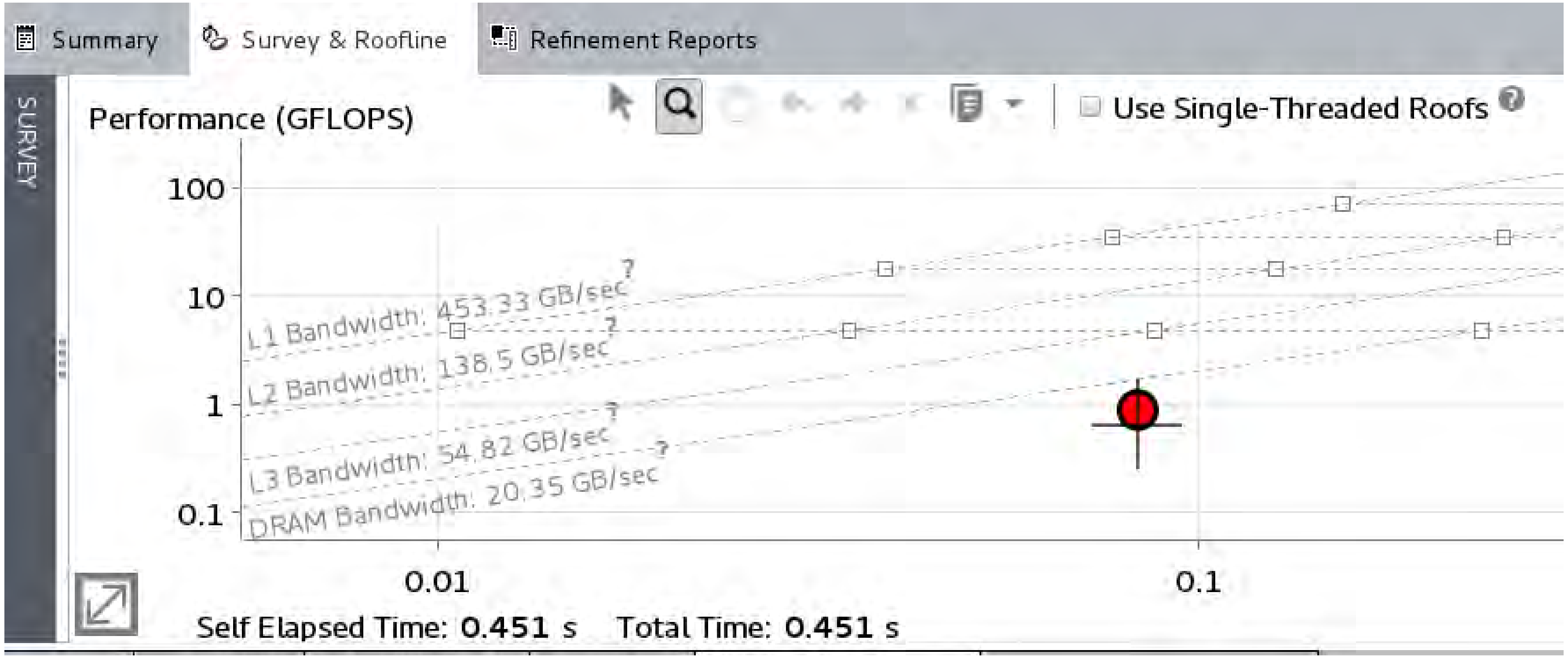
VECTORIZATION OF LOOP CARRIED DEPENDENCY

VECTORIZATION OF LOOP CARRIED DEPENDENCY

- Loop carried dependency

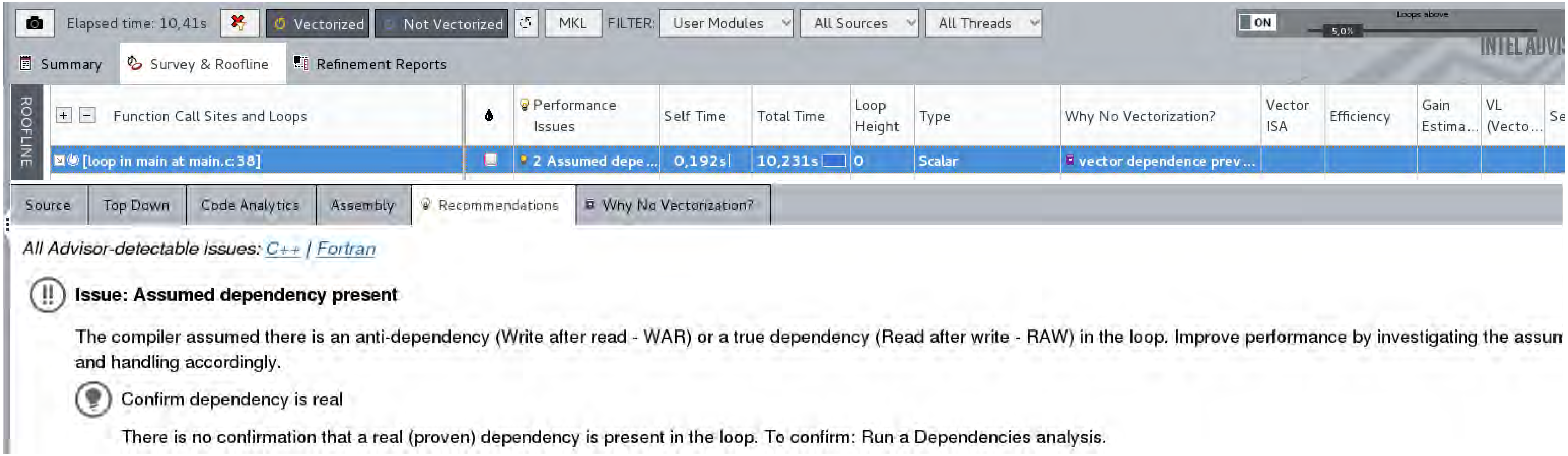
```
void addition(float* a, float* b, float* c, int size){
  int i, j;
  for(i=0; i<size; i++){
    for(j=pad; j<size; j++){
      c[i*size + j] = a[i*size + j]+c[i*size + j-pad];
    }
  }
}
```

VECTORIZATION OF LOOP CARRIED DEPENDENCY



VECTORIZATION OF LOOP CARRIED DEPENDENCY

- Loop carried dependency



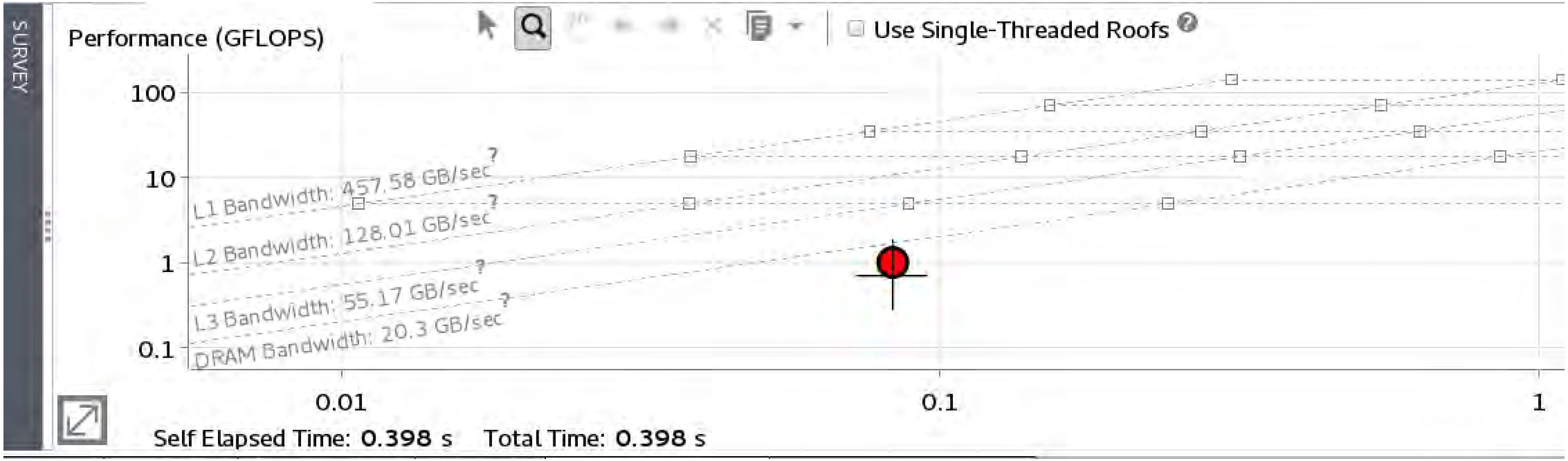
VECTORIZATION OF LOOP CARRIED DEPENDENCY

- Loop carried dependency

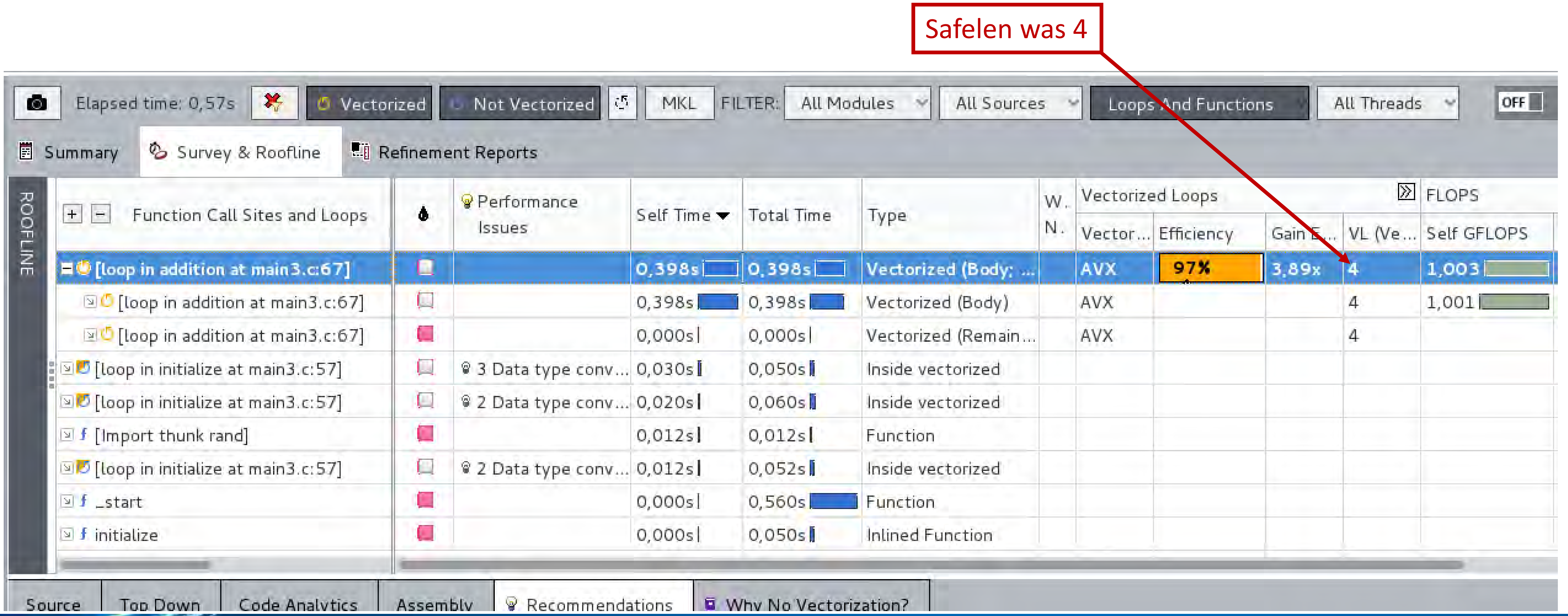
```
void addition(float* a, float* b, float* c, int size){
  int i, j;
  for(i=0; i<size; i++){
    #pragma omp simd safelen(4)
    for(j=pad; j<size; j++){
      c[i*size + j] = a[i*size + j]+c[i*size + j-pad];
    }
  }
}
```

In this case, we assume that pad >=4

VECTORIZATION OF LOOP CARRIED DEPENDENCY



VECTORIZATION OF LOOP CARRIED DEPENDENCY



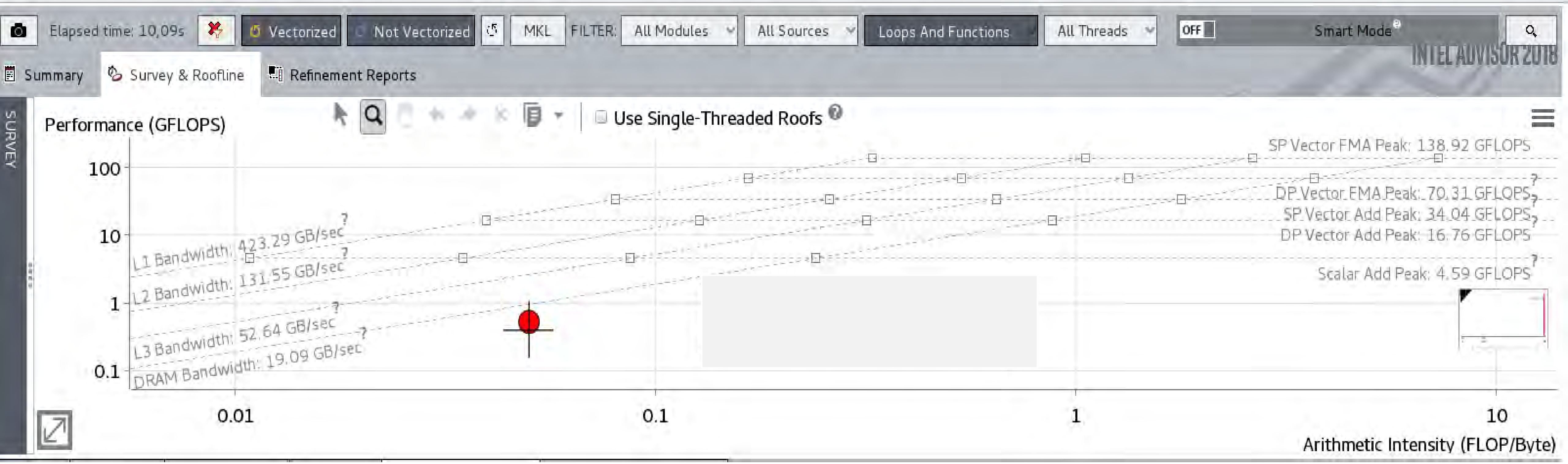
VECTORIZATION OF FUNCTION CALL

VECTORIZATION OF A FUNCTION CALL WITH OMP

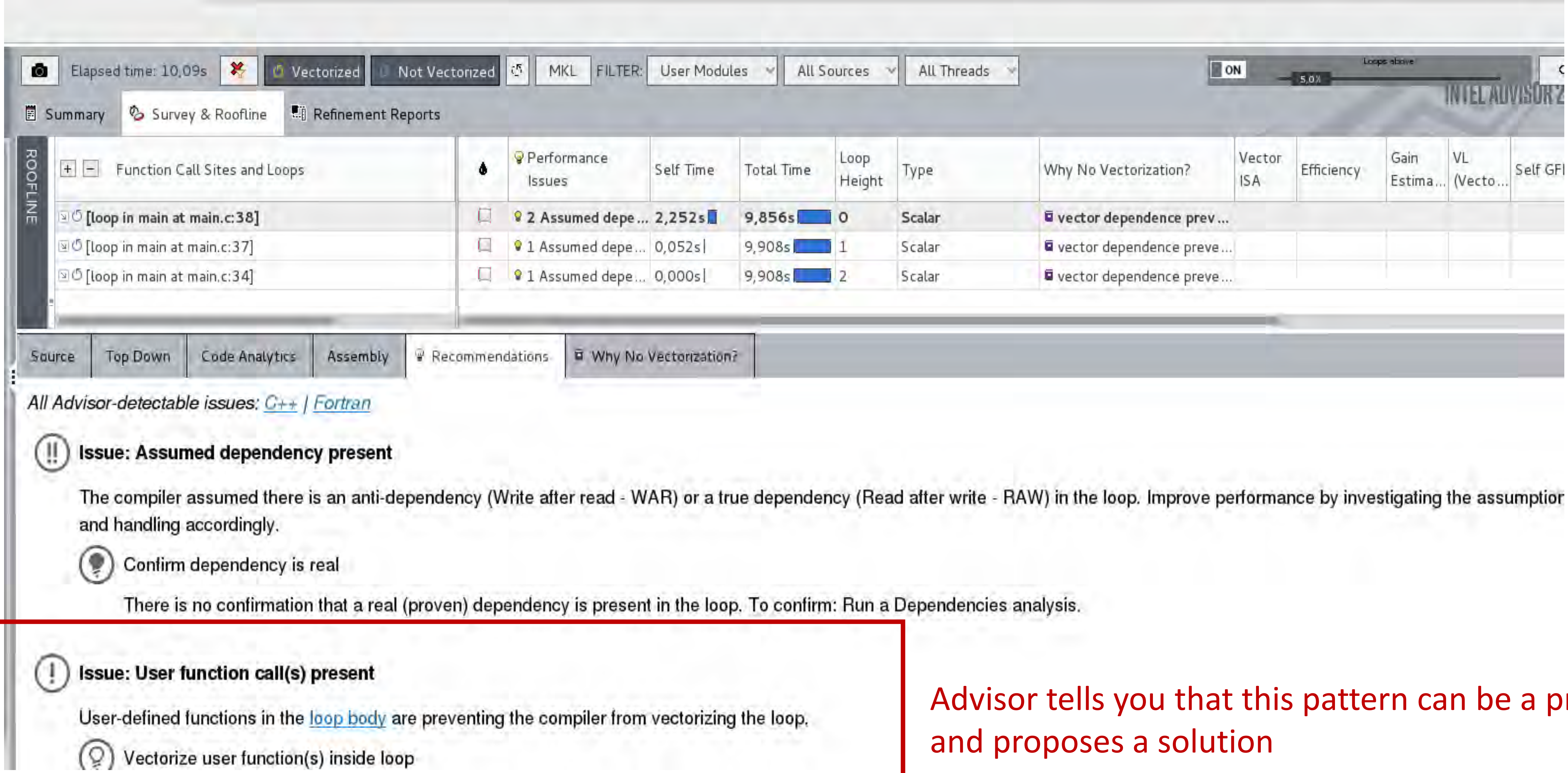
- Function call inside of a loop can kill the vectorization

```
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



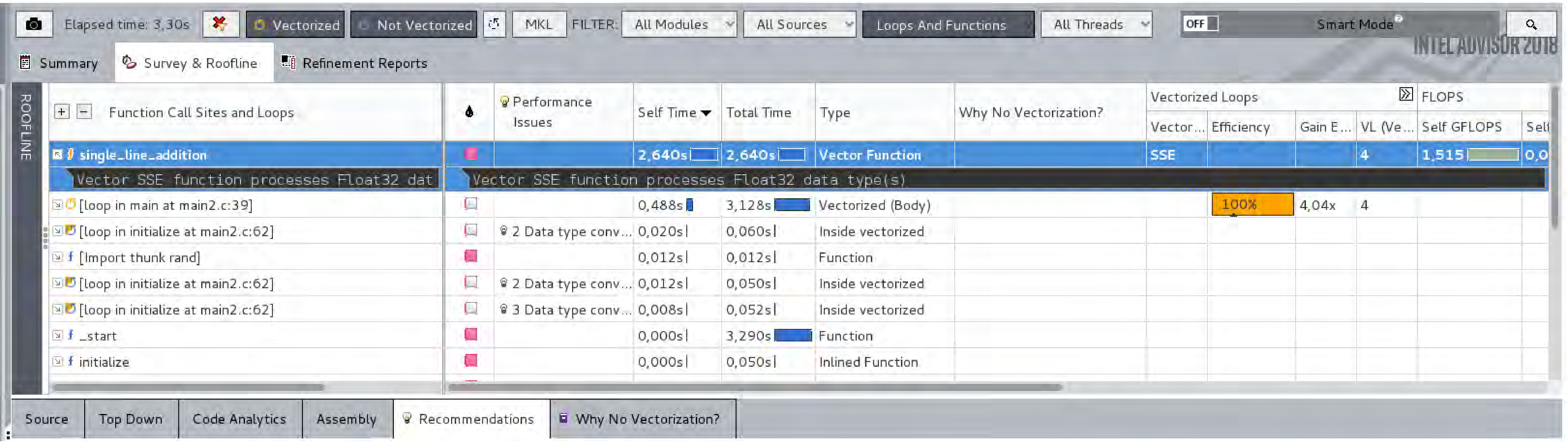
Advisor tells you that this pattern can be a problem and proposes a solution

VECTORIZATION OF A FUNCTION CALL WITH OMP

- Omp declare simd

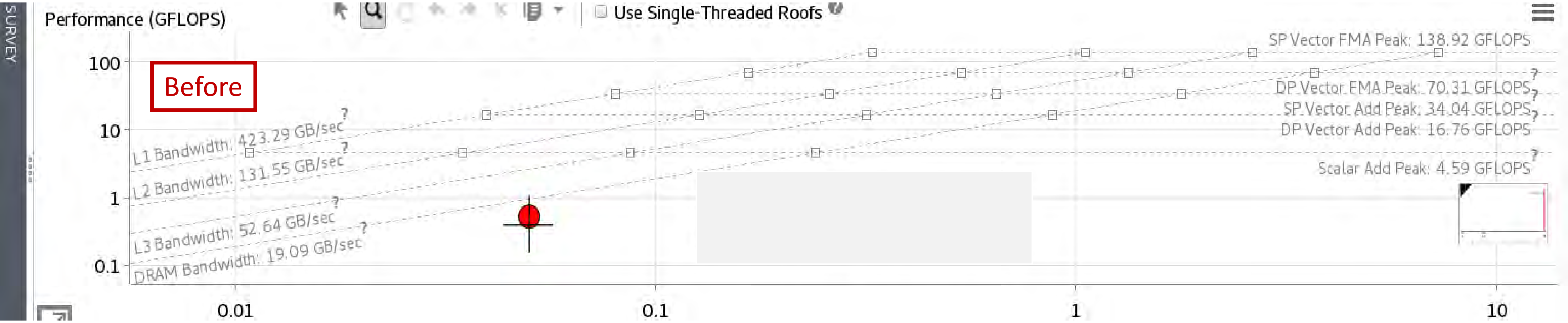
```
for(int i=0; i<SIZE; i++){
  #pragma omp simd
  for(int j=0; j<SIZE; j++){
    single_line_addition(a, c, i*SIZE + j);
  }
}

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

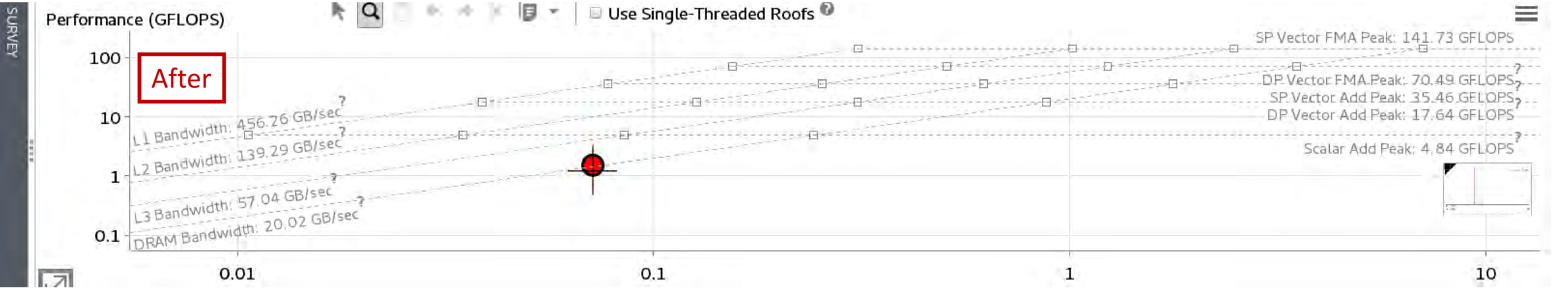
VECTORIZATION OF A FUNCTION CALL WITH OMP



Function Call Sites and Loops	Performance Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vector...	Efficiency	Gain E...	VL (Ve...	FLOPS: Self GFLOPS	Self
single_line_addition		2,640s	2,640s	Vector Function		SSE			4	1,515	0,0
Vector SSE function processes Float32 data type(s)											
[loop in main at main2.c:39]		0,488s	3,128s	Vectorized (Body)			100%	4,04x	4		
[loop in initialize at main2.c:62]	2 Data type conv...	0,020s	0,060s	Inside vectorized							
[Import thunk rand]		0,012s	0,012s	Function							
[loop in initialize at main2.c:62]	2 Data type conv...	0,012s	0,050s	Inside vectorized							
[loop in initialize at main2.c:62]	3 Data type conv...	0,008s	0,052s	Inside vectorized							
_start		0,000s	3,290s	Function							
initialize		0,000s	0,050s	Inlined Function							

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.
-

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