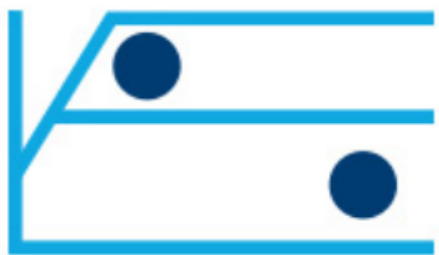


# INTEL® ADVISOR FOR OPENMP ON CPU AND INTEGRATED GPU

Kirill Rogozhin, November 2019

# Intel Advisor

Roofline: Characterize and optimize on **CPU**



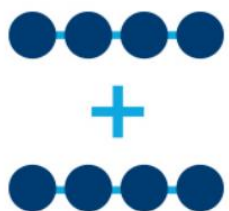
Offload Advisor: identify best offload candidates



Roofline: Characterize and optimize on **GPU**



Vectorization



```
omp simd
omp declare simd
...
```

Memory



Threading



```
omp parallel_for
omp task
...
```

```
omp target
omp teams
omp map...
```

# Advisor Roofline

Characterize and optimize CPU code

# What is the Roofline Model?

Characterization of your application performance in the context of the hardware

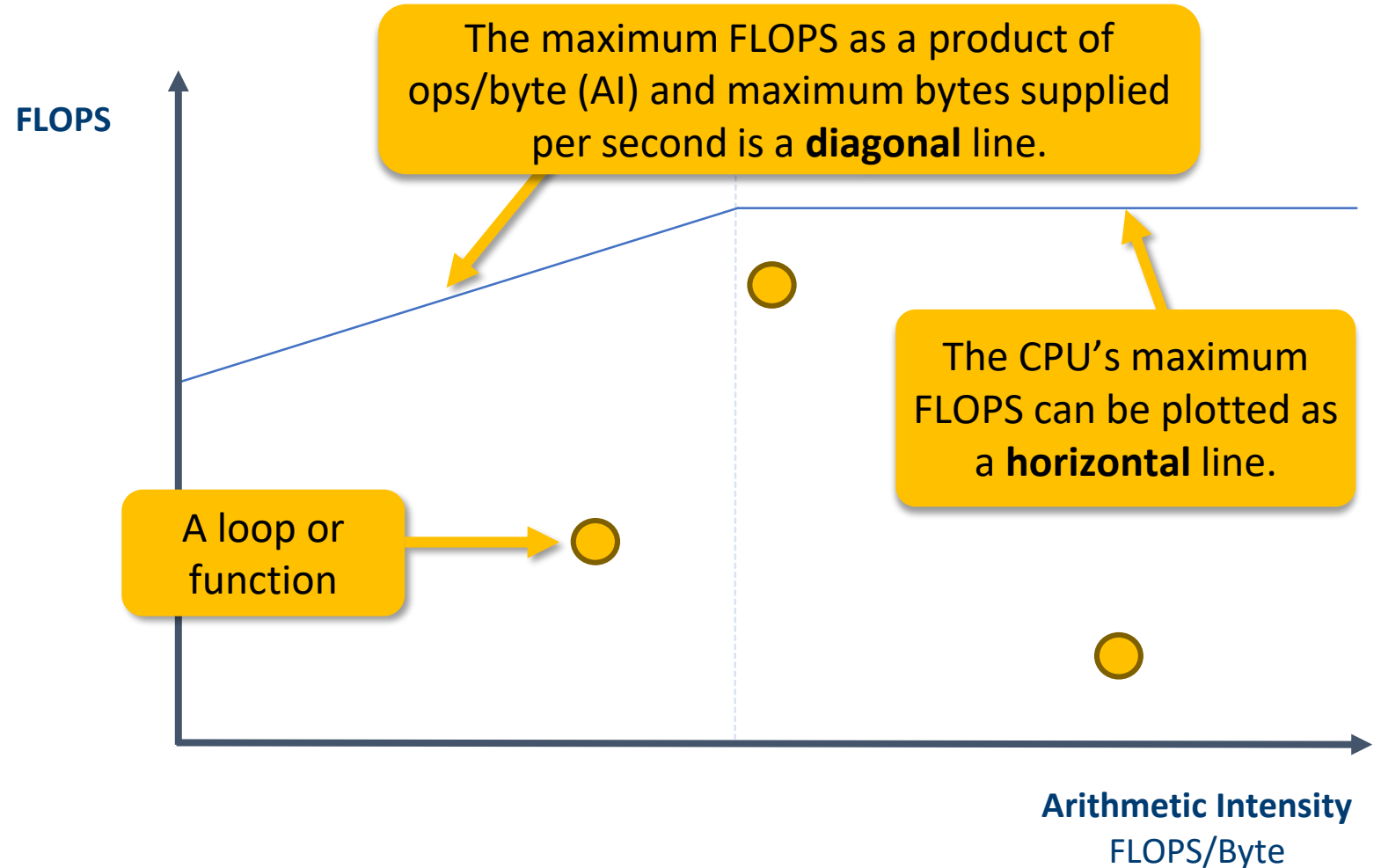
It uses two simple metrics

- Flop count
- Bytes transferred

2 Operations

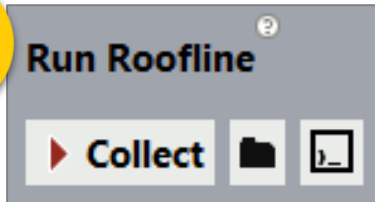
$$a_i = b_i + c_i * d_i$$

1W+3R = 4\*4bytes = 16 bytes



# The Roofline Chart in Intel® Advisor

1

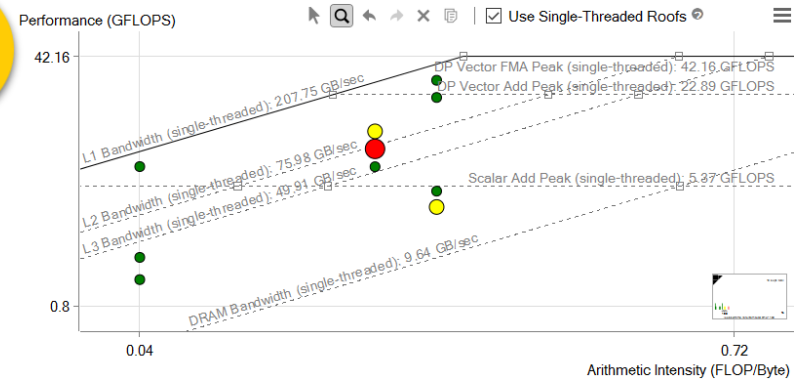


A single button or CLI command runs the Survey and FLOPS analyses to generate the Roofline chart.

2



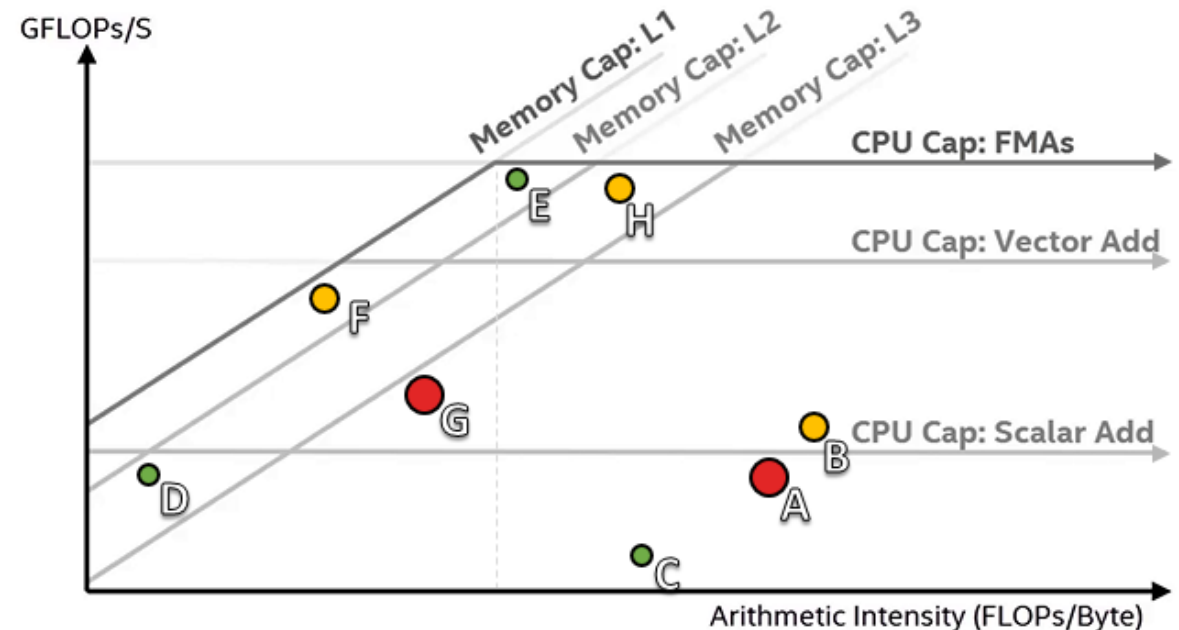
3



Focus on big dots with headroom.

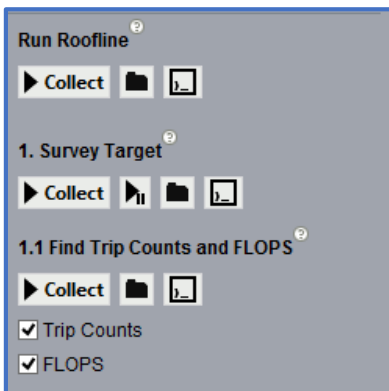
- Intel® Advisor sizes and color-codes dots by relative time taken.
- Space between dots and their uppermost limits is room to improve.

Roofs above a dot indicate potential sources of bottlenecks.



# Getting Roofline data in Intel® Advisor

<p>Roofline :</p> <p>Axis X: <b>AI</b> = <b>#FLOP</b> / <b>#Bytes</b></p> <p>Axis Y: <b>FLOP/S</b> = <b>#FLOP</b> (mask aware) / <b>#Seconds</b></p>	Overhead
<p>Step 1: Survey (-collect survey)</p> <ul style="list-style-type: none"><li>- Provide <b>#Seconds</b></li><li>- <i>Root access not needed</i></li><li>- User mode sampling, non-intrusive.</li></ul>	1x
<p>Step 2: FLOPS (-collect tripcounts -flops)</p> <ul style="list-style-type: none"><li>- Provide <b>#FLOP</b>, <b>#Bytes</b>, AVX-512 Mask</li><li>- <i>Root access not needed</i></li><li>- Precise, instrumentation based, count number of instructions</li></ul>	5-10x



# Questions to answer with Roofline

1

Am I doing well? How far am I from the pick?

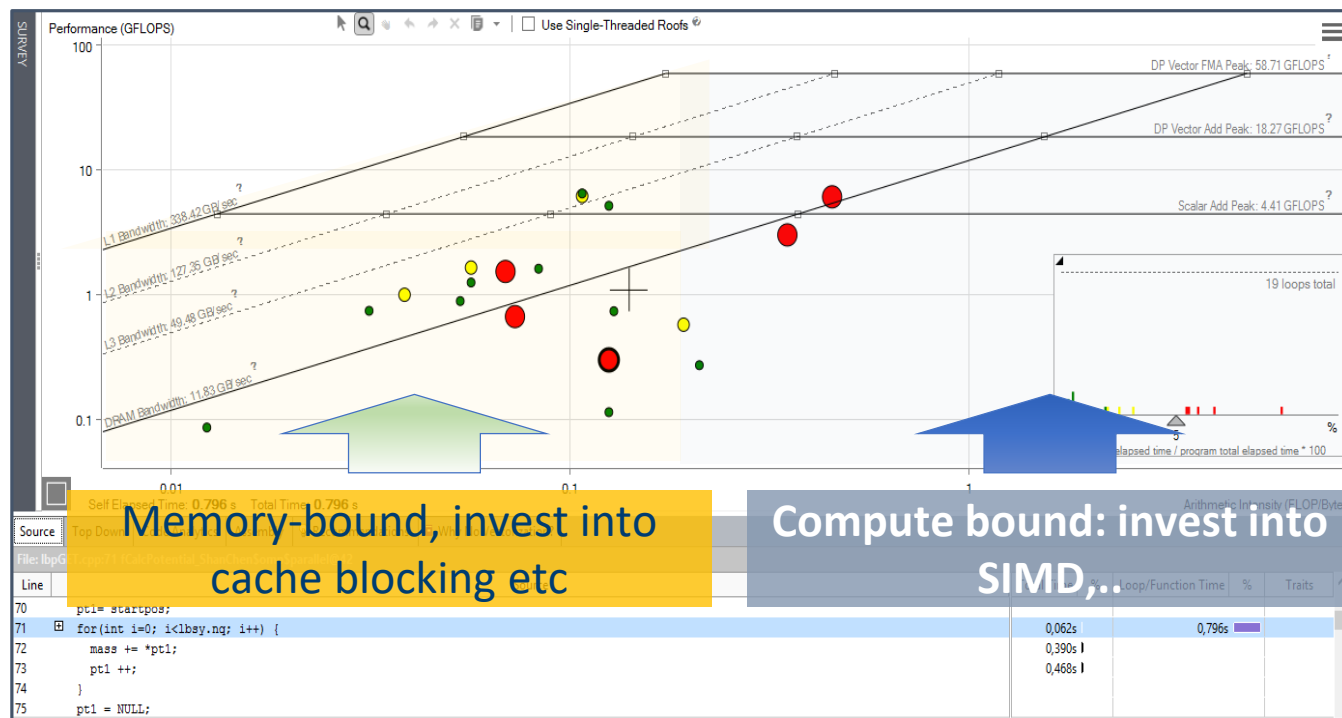
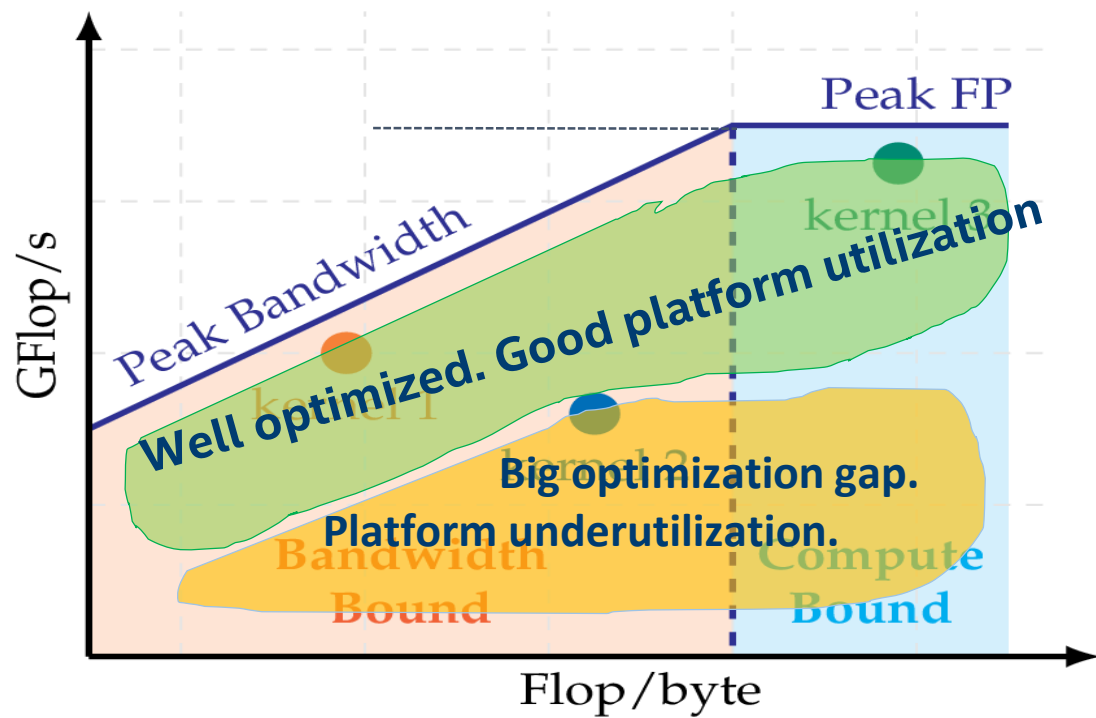
(do I utilize hardware well or not?)

2

Final Bottleneck?

(where will be my limit after I done all optimizations?)

Long-term ROI, optimization strategy



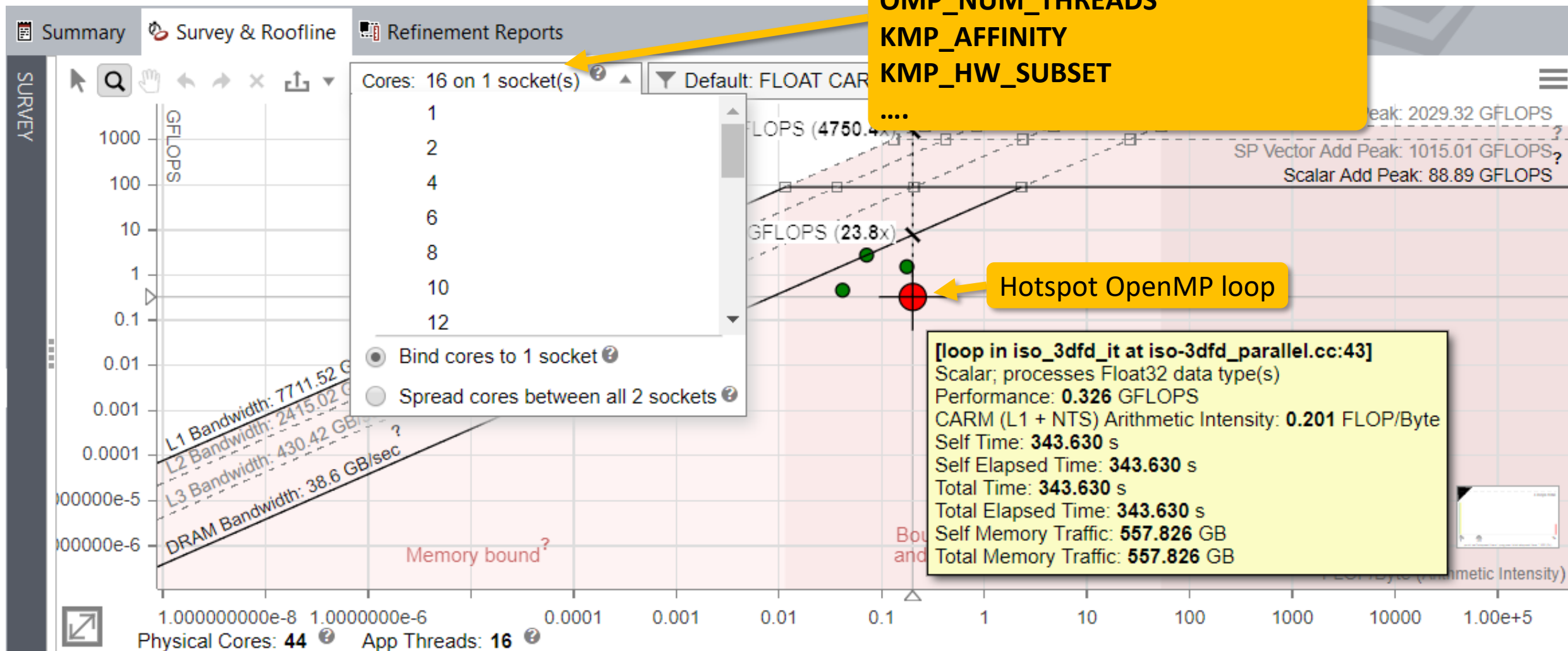
# Advisor Roofline

Characterize and optimize CPU code:

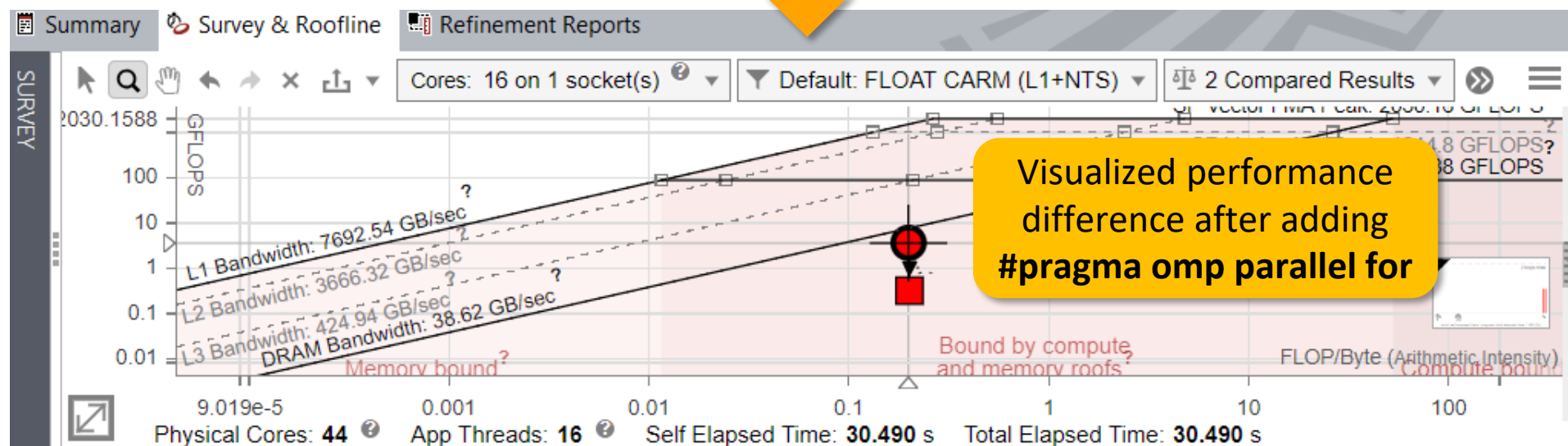
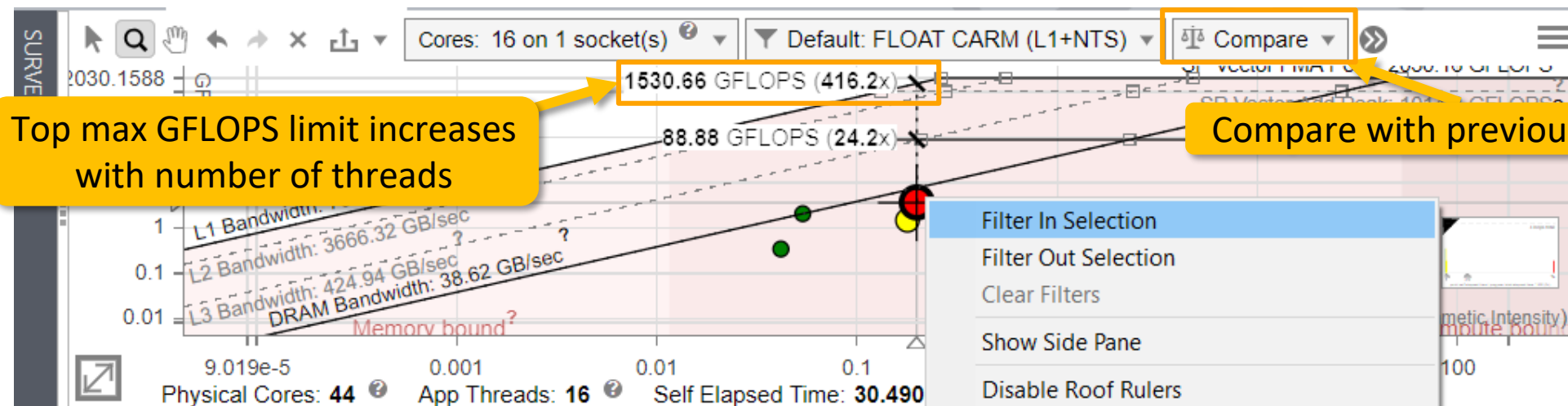
Threading



# Roofline: iso3dfd example

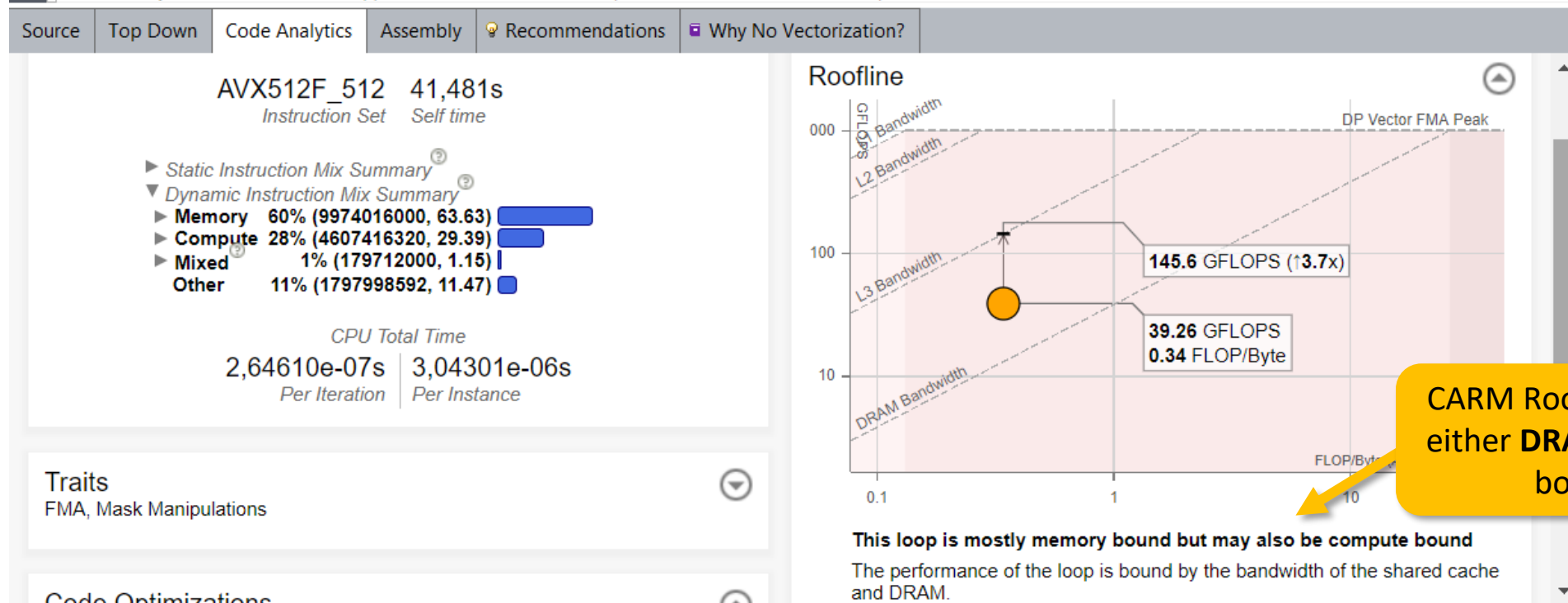
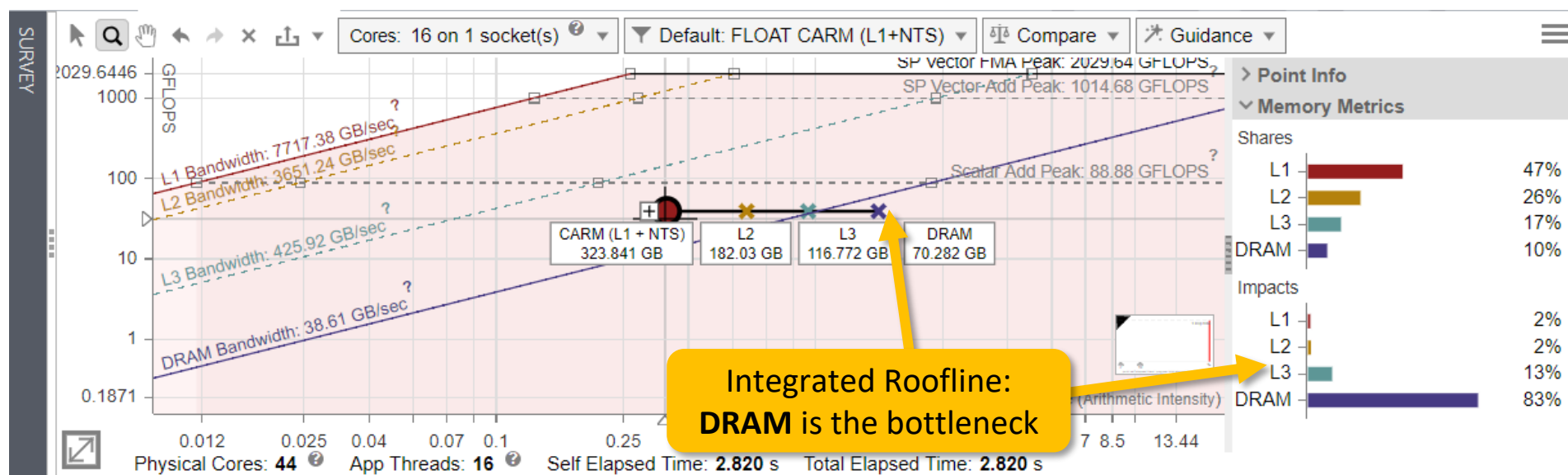


# Roofline: iso3dfd example



# Advisor Roofline

Characterize and optimize CPU code:  
Memory



# Memory access pattern analysis

How should I access data ?

Unit stride access are faster

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



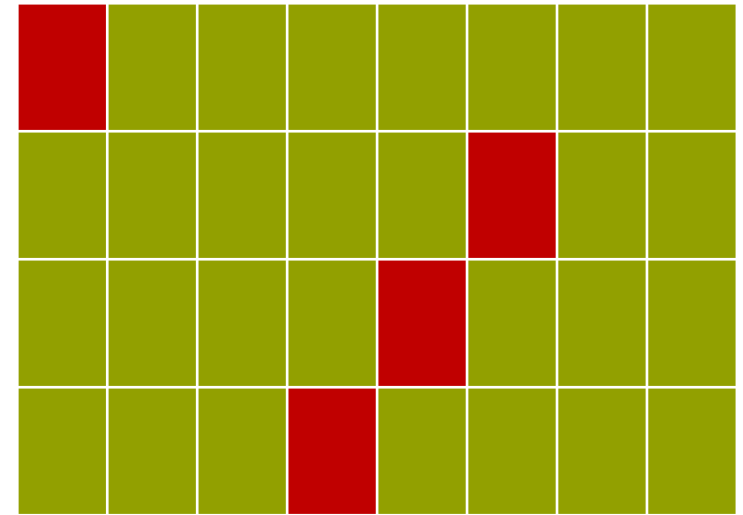
Constant stride are usually worse

```
for (i=0; i<N; i+=2)  
    A[i] = B[i]*d
```



Non predictable access are usually bad

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



# Memory Access Patterns

Summary Survey & Roofline Refinement Reports MAP Source: iso-3dfd_main.cc INTEL ADVISOR				
Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate
[loop in iso_3dfd at iso-3dfd_parallel.cc:4...		50% / 50% / 0%	Mixed Strides	Max. Per-Instruction Addr. Range 55MB
< >				
Memory Access Patterns Report		Dependencies Report	Recommendations	
ID	Stride	Type	Nested Function	Variable references
P1	65536	Constant stride	iso-3dfd_parallel.cc:47	block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7
P2	65536	Constant stride	iso-3dfd_parallel.cc:49	block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7
<pre>47         value += ptr_prev[offset]*coeff[0]; 48         for(int ir=1; ir&lt;=HALF_LENGTH; ir++) { 49             value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]); // horizontal 50             value += coeff[ir] * (ptr_prev[offset + ir*n1] + ptr_prev[offset - ir*n1]); // vertic 51             value += coeff[ir] * (ptr_prev[offset + ir*dimn1n2] + ptr_prev[offset - ir*dimn1n2]);</pre>				
P3	65536	Constant stride	iso-3dfd_parallel.cc:50	block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7
P4	65536	Constant stride	iso-3dfd_parallel.cc:51	block 0x7fd89ffe010 allocated at iso-3dfd_main.cc:184, block 0x7

Strided access in OpenMP region

Memory object

ptr\_prev[offset - ir]

# Advisor Roofline

Characterize and optimize CPU code:

Vectorization



# Advisor Vectorization analysis

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?

Elapsed time: 1462.35s

Vectorized

Not Vectorized

FILTER: All Modules All Sources

Loops

All Threads

OFF

Smart Mode

INTEL ADVISOR 2018

Summary

Survey & Roofline

Refinement Reports

Function Call Sites and Loops

Performance Issues

Self Time

Trip Counts

Why No Vectorization?

Vectorized Loops

			Average	Call Count		Vect...	Efficiency	Gain...	VL (...)	Com..
[loop in runOMPRawLoops\$omp\$...]	2 Assumed dependency present	15.484s	446	101976000	vector dependence prevents vectoriz...					
[loop in runCRawLoops at runCRa...]	2 Assumed dependency present	11.766s	12511	75120000	vector dependence prevents vectorization					
[loop in runCForallLambdalops a...]	2 Assumed dependency present	11.766s	12511	75120000	vector dependence prevents vectorization					
[loop in runCRawLoops at runCRa...]	2 Assumed dependency present	5.156s	19387	3075000	vector dependence prevents vectorization					
[loop in runCForallLambdalops a...]	2 Assumed dependency present	5.125s	19387	3075000	vector dependence prevents vectorization					
[loop in runOMPRawLoops\$omp\$...]	1 Ineffective peeled/remainder loop(s) ..	4.190s	2; 110; 2	112590000 ...		AVX	100%	5.10x	4	5.28x
[loop in runOMPRawLoops\$omp\$...]		3.768s	2	1125900000						
[loop in runOMPRawLoops\$omp\$...]		0.406s	110	12320000		AVX			4	5.28x
[loop in runOMPRawLoops\$omp\$...]		0.016s	2	880000						

Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?



# Enabling Vectorization

1

Vector Issues	Self Time▼	Total Time	Type	W
• 2 Assumed dependency present	20.030s	20.030s	Scalar Versions	▢
	13.508s	13.508s	Scalar	▢
	6.895s	27.750s	Scalar	▢

2

Check dependencies

3

Use **#pragma omp simd**

Report	Refinement Reports
	Loop-Carried Dependencies
h_4_v253.cpp:183]	✓ No dependencies found
_3_1_5_kernel_max.cpp:80]	✓ No dependencies found

Vector Issues	Self Time▼	Total Time	Type	Vectorized Loops			
				Vector ISA	Efficiency	Gain ...	VL (V...
	10.507s	22.989s	Scalar				
• 2 Possible inef...	1.762s	3.190s	Vectorized Ver...	AVX512	73%	5.84x	8

# Offload Advisor

Define regions to be offloaded

# Offload Advisor Summary

Intel® Advisor Beta, build 604296



Intel® Advisor Beta

## OFFLOAD ADVISOR

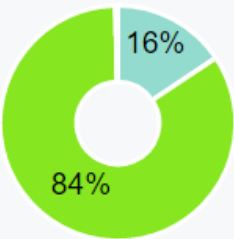
[Summary](#) | [Offloaded Regions](#) | [Non Offloaded Regions](#) | [Call Tree](#) | [Configuration](#) | [Logs](#)

Speed Up for Accelerated Code ? 4.2x | Number of Offloads ? 2 | Fraction of Accelerated Code ? 95%

### Program metrics ?

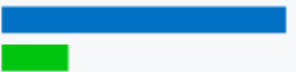


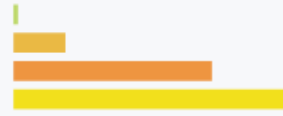


Target Platform	Gen11 GT2	Time on Host ?	0.30s
Number of Offloads ?	2	Time on Accelerator ?	1.61s
Speed Up for Accelerated Code ?	4.2x	Data Transfer Tax ?	0s
Amdahl's Law Speed Up ?	3.6x	Invocation Tax ?	<0.01s
Fraction of Accelerated Code ?	95%	Code Transfer Tax ?	<0.01s



# Explore offload candidates

## Top offloaded ?

Location ?	Speed Up ?	Bounded By ?	Data Transfer ?
[loop in iso_3dfd_it at iso3dfd.cpp:233]	4.27x 	 DRAM_BW	465.83MB
[loop in initializeFT\$omp\$parallel@51 at iso3dfd.cpp:59]	1.10x 	 DRAM_BW	357.97MB

Offload candidates

Performance bounding factors

- Total Execution Time by Compute: <0.01s
- Total Execution Time by L3 BW (s): 0.010
- Total Execution Time by LLC BW (s): 0.039
- Total Execution Time by Memory BW Time (s): 0.055
- Data Transfer Tax(s): 0
- Invocation Tax(s): <0.00001
- Kernel Code Transfer Tax(s): <0.00001

# Offloaded regions details and source view

Hierarchy	Loop/Function >		Offload Information		Column configurator Custom filter	Source Name: [loop in iso_3dfd_it at iso3dfd.cpp:233]
	Elapsed Time (s)	Total Time ↓	Dependency Type	Estimated Speed Up		
> [loop in iso_3dfd_it at iso3dfd.cpp:233]	6.68s	130.878s (94.53%)	Parallel: Assumed	4.27x		<pre>225 { 226     izEnd = MIN(bz+n3_Tblock, n3End); 227     iyEnd = MIN(by+n2_Tblock, n2End); 228     ixEnd = MIN(n1_Tblock, n1End-bx); 229     //ixEnd = MIN(bx+n1_Tblock, n1End); 230     //ixEnd = MIN(bx+n1_Tblock, n1End); 231 232     for(size_t iz=bz; iz&lt;izEnd; iz++) { 233         for(size_t iy=by; iy&lt;iyEnd; iy++) { 234             ptr_next = &amp;ptr_next_base[iz*dimn1n2 + iy 235             ptr_prev = &amp;ptr_prev_base[iz*dimn1n2 + iy 236             ptr_vel = &amp;ptr_vel_base[iz*dimn1n2 + iy*n 237 238 &lt; &gt;</pre>
> [loop in initializeFT\$omp\$parallel@51 a	0.06s	0.719s (0.52%)	Parallel: Explicit	1.10x		

The region can be **4.27x** faster on Gen11

Add “**#pragma omp target**” here

# Data transfer and memory mapping

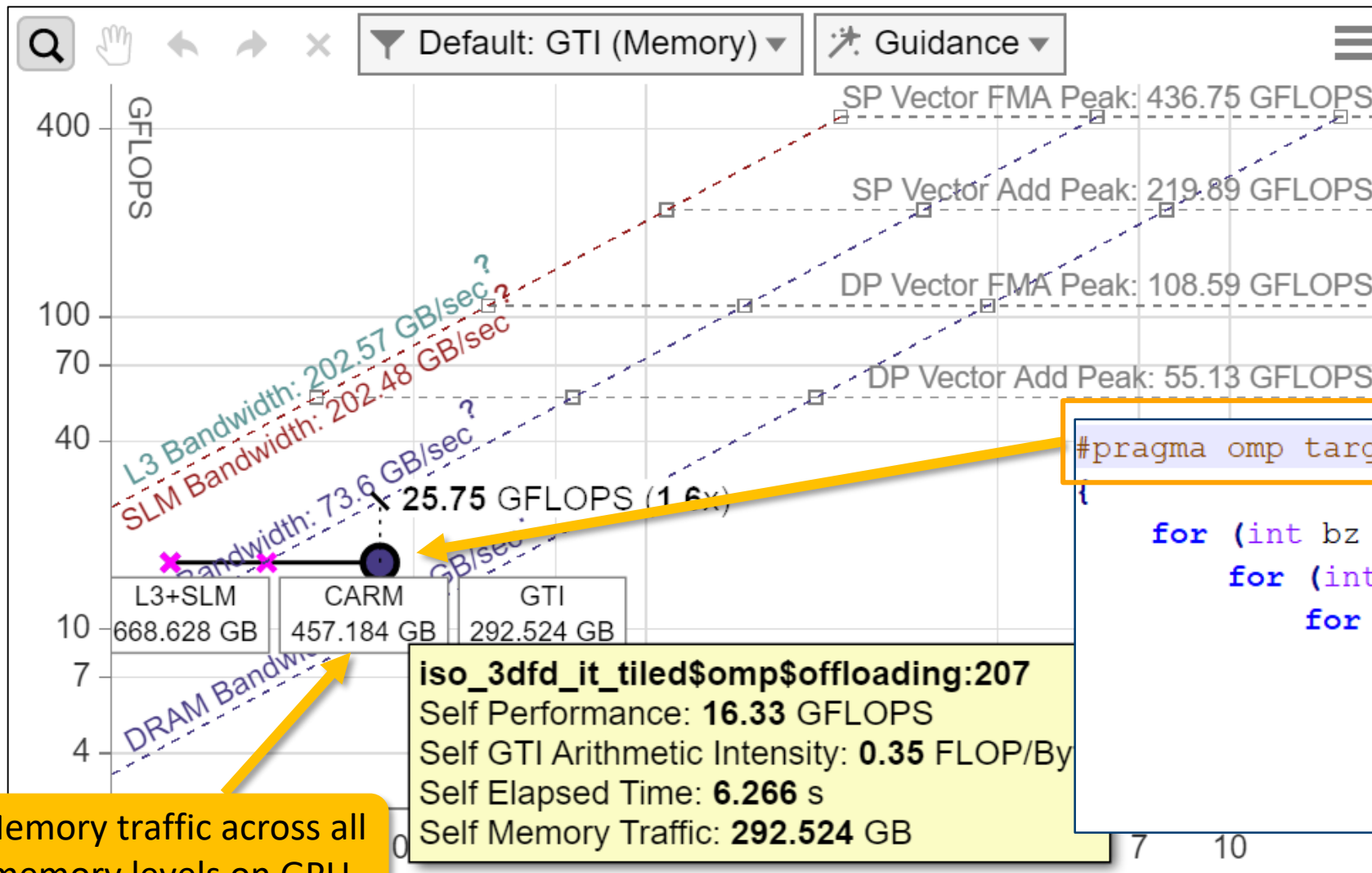
Data Transfer <			map (to : a)	map (from : a)	map (tofrom : a)
Total Data Transfer (MB)	Total Data Transferred from CPU to GPU (MB)	Total Data Transferred from GPU to CPU (MB)	Memory Mapped To Device (MB)	Memory Mapped From Device (MB)	Memory Mapped ToFrom Device (MB)
465.83MB	232.91	232.91	0	0	232.91494
357.97MB	116.47	241.50	0	125.02630	116.46976

Total estimated data traffic

# GPU Roofline

Characterize and optimize GPU code

# Roofline for GPU



- Preview feature, supports Gen9 GPU
- Command line data collection
- HTML export as UI

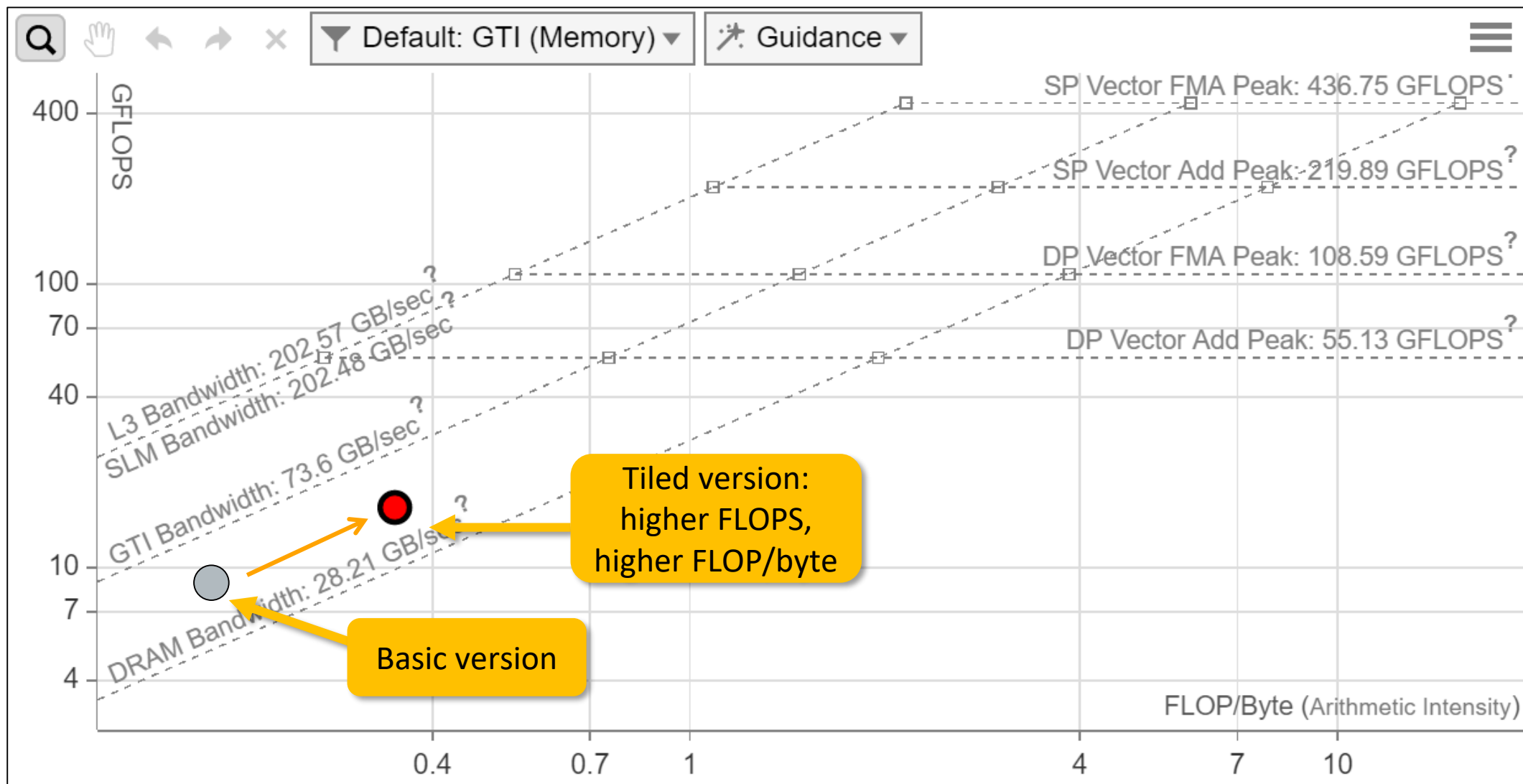
```
#pragma omp target teams distribute collapse(3)
```

```
{
    for (int bz = HALF_LENGTH; bz < n3End; bz += n3_Tblock)
        for (int by = HALF_LENGTH; by < n2End; by += n2_Tblock)
            for (int bx = HALF_LENGTH; bx < n1End; bx += n1_Tblock)
                // ...
            int izEnd = MIN(bz + n3_Tblock, n3End);
            int iyEnd = MIN(by + n2_Tblock, n2End);
            int ixEnd = MIN(bx + n1_Tblock, n1End);
        }
}
```

Memory traffic across all memory levels on GPU

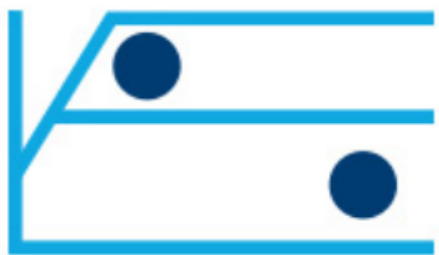


# GPU Roofline



# Intel Advisor

Roofline: Characterize and optimize on **CPU**



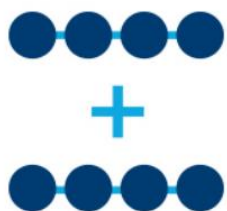
Offload Advisor: identify best offload candidates



Roofline: Characterize and optimize on **GPU**



Vectorization



Memory



Threading



```
#pragma omp simd  
#pragma omp declare simd  
...
```

```
#pragma omp simd  
#pragma omp declare simd  
...
```

```
#pragma omp target  
omp teams  
...
```

