



OPTIMIZE USING ROOFLINE AUTOMATION AND SIMD ANALYSIS TOOLS

Intel® Advisor GUI

Program metrics

Elapsed Time 51.60s
 Vector Instruction Set AVX512, AVX2, AVX
 Number of CPU Threads 48

GFLOPS 3.14
 GFLOP Count 161.877
 FP Arithmetic Intensity 0.12868
 GINTOPS 0.43

Performance characteristics



Vectorization Gain/Efficiency



OP/S and Bandwidth

Effective OP/S And Bandwidth	Utilization	Hardware Peak
> GFLOPS	3.137 0.083% 0.041%	out of 3789 (DP) FLOPS out of 7577 (SP) FLOPS
> GINTOPS	0.4308 0.021% 0.011%	out of 2046 (Int64) INTOPS out of 4094 (Int32) INTOPS
> CPU <-> Memory [L1+NTS GB/s]	24.38 0.14%	out of 17380 GB/s [bytes]
> L2 Bandwidth [GB/s]	30.7 0.51%	out of 6049 GB/s [cache line bytes]
> L3 Bandwidth [GB/s]	28.16 2.4%	out of 1171 GB/s [cache line bytes]
> DRAM Bandwidth [GB/s]	30.63 14%	out of 224.8 GB/s [cache line bytes]

Top time-consuming loops

Loop	Self Time	Total Time
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:95]	330.0675s	330.0675s
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:81]	140.8077s	140.8077s
[loop in lbm::stream\$omp\$parallel_for@22 at stream.cpp:26]	73.18691s	73.18691s
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:36]	63.32187s	63.32187s
[loop in lbm::init\$omp\$parallel@52 at init.cpp:88]	32.76916s	32.76916s

Loop in lbm::collision\$omp\$parallel@28 at collision.cpp:95

330.067s
 Total time

Vectorized (Body)

AVX512BW_128; 330.067s
 AVX512DQ_512; AVX512F_512
 Self time
 Instruction Set

Dynamic Instruction Mix Summary

- Memory 28% (7667136000)
- Compute 25% (6865344000)
- Vector 25% (6815232000)
- Scalar < 1% (50112000)
- Mixed 20% (5311872000)
- Other 27% (7316352000)

Elapsed time: 51.60s | Vectorized | Not Vectorized | FILTER: All Modules | collision.cpp | Loops And Functions | All Threads

Summary | Survey & Roofline | Refinement Reports

Function Call Sites and Loops	Performance Issues	CPU Time	Type	Why No Vectorization?	Vectorized Loops	
		Self Time	Total Time		Vector ... Efficiency	Gain Es...
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:95]	1 Possible ineffi...	330.067s	330.067s	Vectorized (Body)	AVX512 -30%	2.42x
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:81]	1 Possible ineffici...	140.808s	140.808s	Scalar		
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:36]	1 Possible ineffici...	63.322s	63.322s	Scalar		
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:47]		15.206s	15.206s	Vectorized (Body)	AVX512 -94%	7.55x
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:93]		3.560s	333.647s	Scalar		
[loop in lbm::collision\$omp\$parallel@28 at collision.cpp:79]	1 Assumed depen...	0.290s	141.098s	Scalar		

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

File: collision.cpp:95 lbm::collision\$omp\$parallel@28

Line	Source	Total Time	%	Loop/Function Time	%	Traits
94	#pragma ivdep					
95	for (k = 1; k <= N2; k++) { [loop in lbm::collision\$omp\$parallel@28 at collision.cpp:95] Vectorized AVX512BW_128; AVX512DQ_512; AVX512F_512 loop processes Float32; Float64; UInt64; UByte data type Peeled loop	4.730s		330.067s		
96						
97	//define some local values					
98	phin = phi[i][j][k];	0.120s				
99	phin2 = phin*phin;					
100	invRhon = 1.0/rho[i][j][k];	4.500s				Appr...
101						
102	//----- Hydrodynamics (velocity, pressure, interfacial force) -----					
Selected (Total Time):		4.730s				

Optimization Notice

Copyright © 2019, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



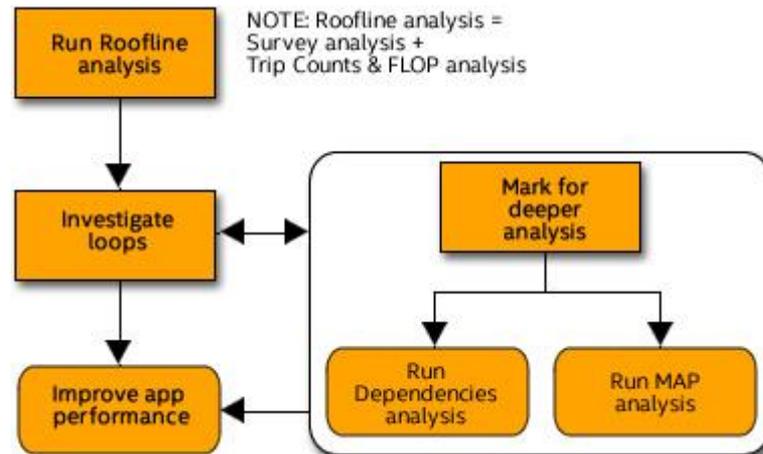
Typical Vectorization Optimization Workflow

There is no need to recompile or relink the application, but the use of `-g` is recommended.

In a rush: Collect Survey data and analyze loops iteratively

Looking for detail:

1. Collect survey and tripcounts data **[Roofline]**
 - Investigate application place within roofline model
 - Determine vectorization efficiency and opportunities for improvement
2. Collect memory access pattern data
 - Determine data structure optimization needs
3. Collect dependencies
 - Differentiate between real and assumed issues blocking vectorization



What is the Roofline Model?

Characterization of your application performance in the context of the hardware

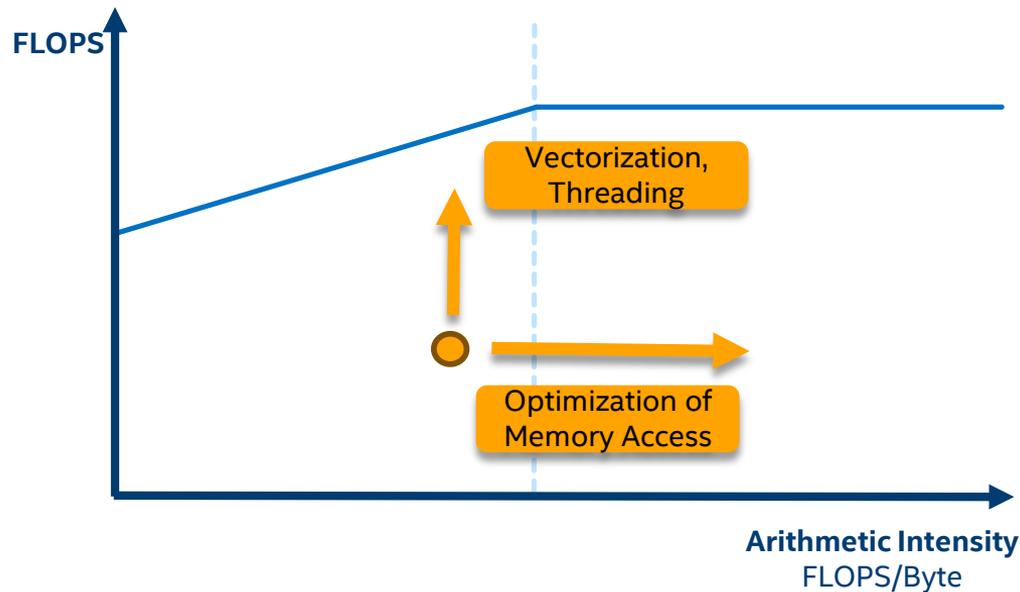
It uses two simple metrics

- Flop count
- Bytes transferred

2 Operations

$$a_i = b_i + c_i * d_i$$

$$1W+3R = 4*4\text{bytes} = 16 \text{ bytes}$$



Roofline first proposed by University of California at Berkeley:
[Roofline: An Insightful Visual Performance Model for Multicore Architectures](#), 2009
Cache-aware variant proposed by University of Lisbon:
[Cache-Aware Roofline Model: Upgrading the Loft](#), 2013

Roofline Model in Intel® Advisor

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
- Instrumentation - Bytes and Flops
- Sampling - Time

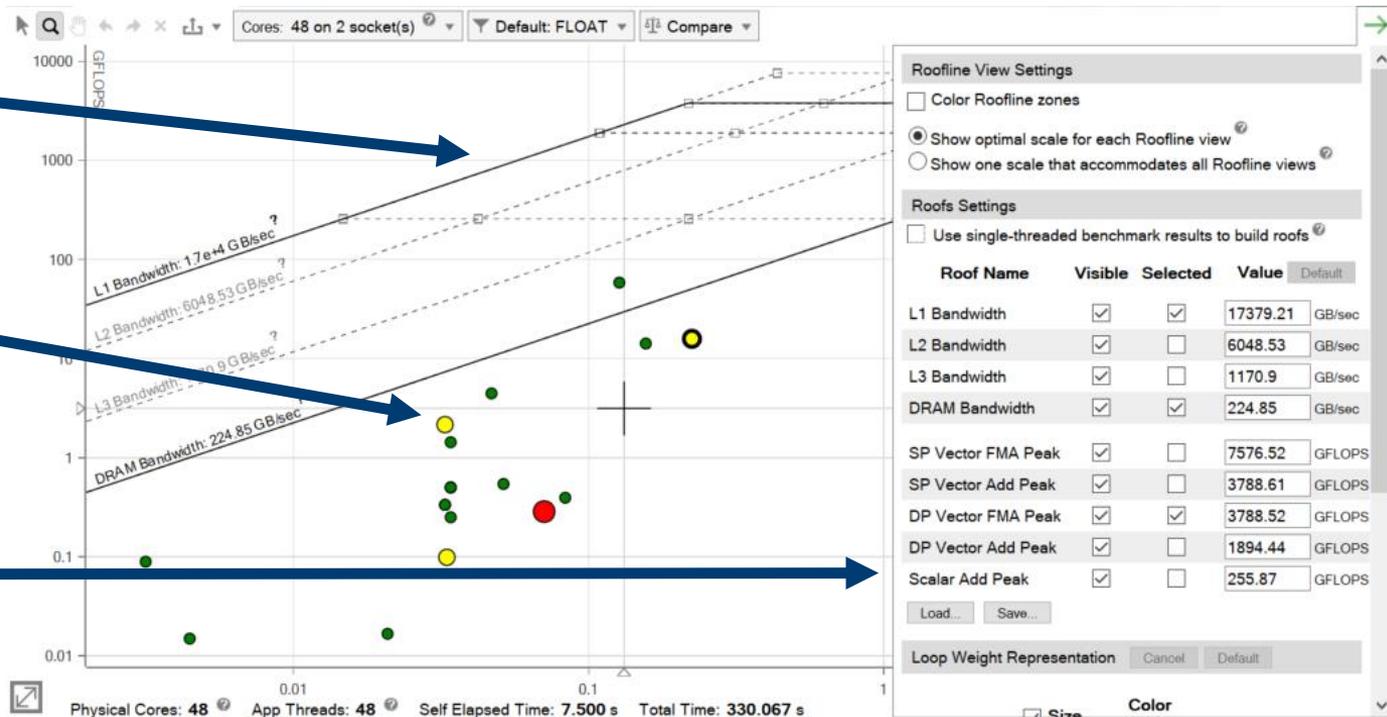
Advantage of CARM	Disadvantage of CARM
No Hardware counters	Only vertical movements !
Affordable overhead (at worst =~10x)	Difficult to interpret
Algorithmic (cumulative L1/L2/LLC)	How to improve performance ?

Roofline Chart in Intel® Advisor

Roof values are measured

Dots represent profiled loops and functions

High level of customization



TUNING A SMALL EXAMPLE WITH ROOFLINE

A Short Walk Through the Process

Example Code

A Short Walk Through the Process

The example loop runs through an array of structures and does some generic math on some of its elements, then stores the results into a vector. It repeats this several times to artificially pad the short run time of the simple example.

```
51  [-] for (int r = 0; r < REPEAT; r++)
52      {
53  [-]     for (int i = 0; i < SIZE; i++)
54          {
55              X[i] = ((7.4 * Y[i].a + 14.2) + Y[i].b * 3.1) * Y[i].a + 42.0;
56          }
57      }
```

```
26     vector<double> X(SIZE);
27  [-] typedef struct AoS
28      {
29         double a;
30         double b;
31         double pad1;
32         double pad2;
33     } AoS;
34     AoS Y[SIZE];
```


Overcoming the Initial Bottleneck

A Short Walk Through the Process

The recommendations tab elaborates: the dependency is only assumed.

Site Location	Loop-Carried Dependencies	Performance Issues
+ ↺ [loop in main at roofline.cpp:54]	✔ No dependencies found	💡 1 Assumed depe...

Memory Access Patterns Report	Dependencies Report	💡 Recommendations
-------------------------------	---------------------	---

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.

💡 **Recommendation: Confirm dependency is real**

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

Running a Dependencies analysis confirms that it's false, and recommends forcing vectorization with a pragma.

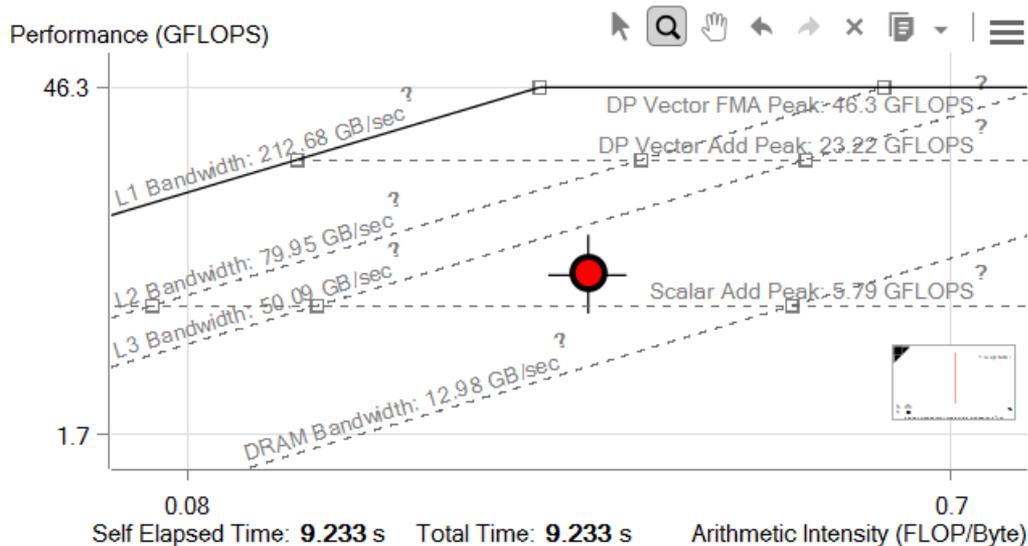
The Second Bottleneck

A Short Walk Through the Process

Adding a pragma to force the loop to vectorize successfully overcomes the Scalar Add Peak. It is now below L3 Bandwidth.

The compiler is not making the same algorithmic optimizations, so the AI has also changed.

```
50 for (int r = 0; r < REPEAT; r++)
51 {
52     #pragma omp simd
53     for (int i = 0; i < SIZE; i++)
54     {
55         X[i] = ((7.4 * Y[i].a + 14.2) + Y[i].b * 3.1) * Y[i].a + 42.0;
56     }
57 }
```



Diagnosing Inefficiency

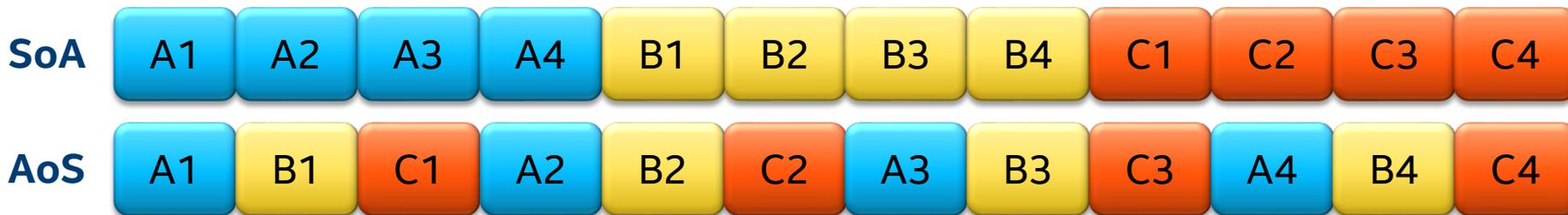
A Short Walk Through the Process

While the loop is now vectorized, it is inefficient. Inefficient vectorization and excessive cache traffic both often result from poor access patterns, which can be confirmed with a MAP analysis.

Function Call Sites and Loops		Vectorized Loops				
+	-	Vector ...	Efficiency	Gain E...	VL (Ve...	
		[loop in main at roofline.cpp:53]	AVX	43%	1.73x	4

Site Location	Strides Distribution	Recommendations
[loop in main at roofline.cpp:53]	50% / 50% / 0%	💡 1 Inefficient memory access patterns present

Array of Structures is an inefficient data layout, particularly for vectorization.



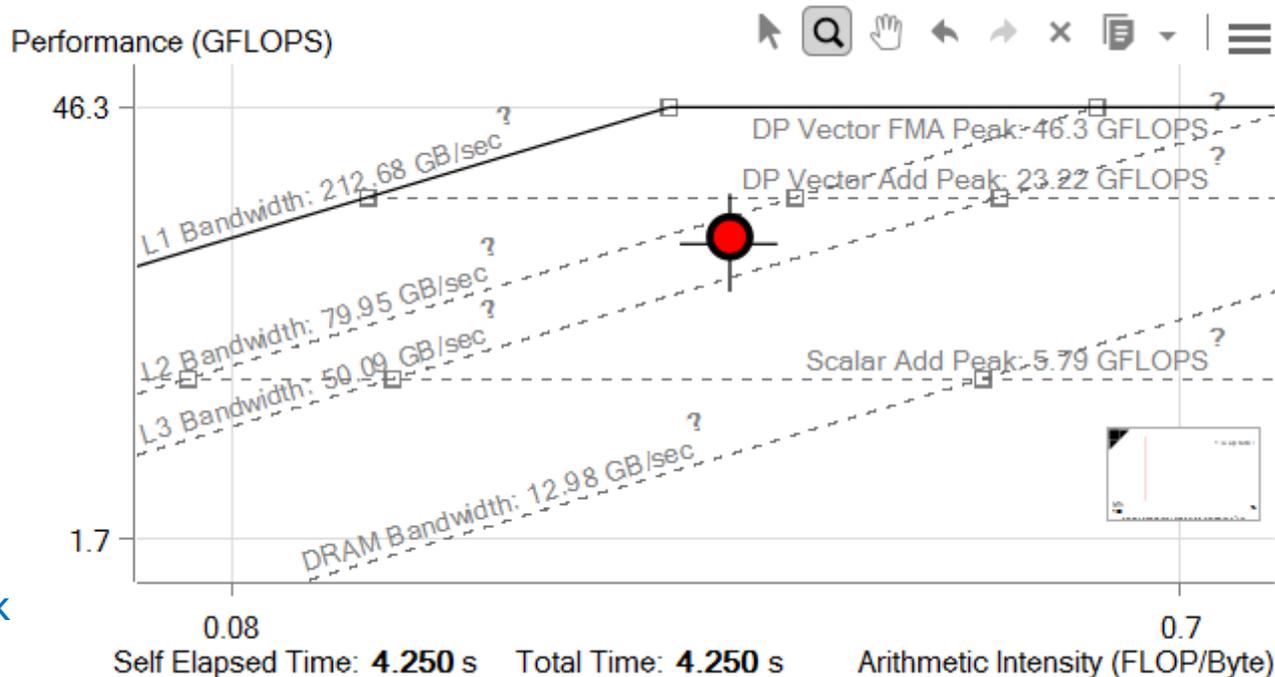
A New Data Layout

A Short Walk Through the Process

Changing Y to SoA layout moved performance up again.

```
26 vector<double> X(SIZE);
27 typedef struct SoA
28 {
29     double a[SIZE];
30     double b[SIZE];
31     double pad1[SIZE];
32     double pad2[SIZE];
33 } SoA;
34 SoA Y;
```

Either the Vector Add Peak or L2 Bandwidth could be the problem now.



Improving the Instruction Set

A Short Walk Through the Process

Because it's so close to an intersection, it's hard to tell whether the Bandwidth or Computation roof is the bottleneck. Checking the Recommendations tab guides us to recompile with a flag for AVX2 vector instructions.

Issue: Potential underutilization of FMA instructions

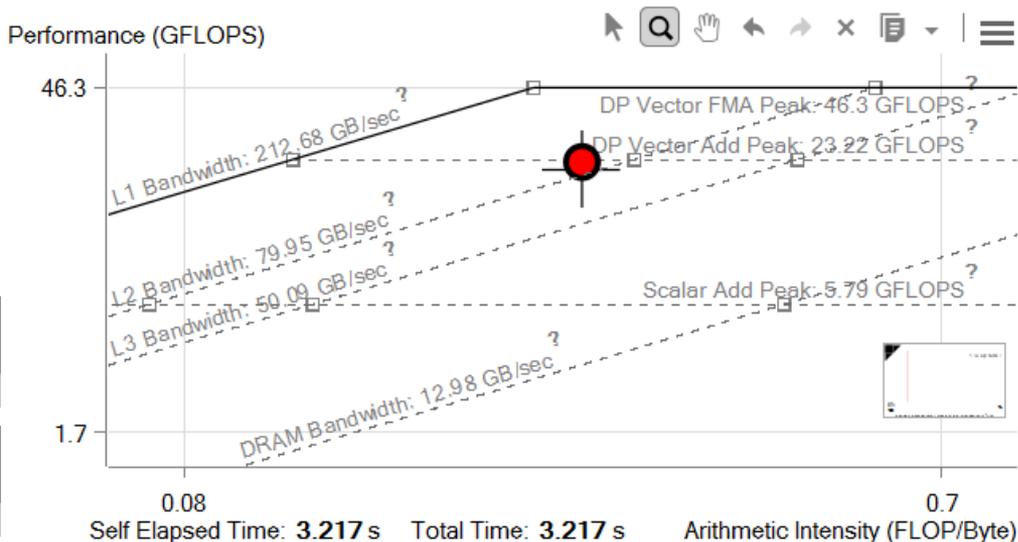
Your current hardware supports the AVX2 instruction set architecture (ISA), which enables the use of fused multiply-add (FMA) instructions. Improve performance by utilizing FMA instructions.

Recommendation: Target the higher ISA

Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Use the following compiler options:

Before		Loops	
Function Call Sites and Loops	Efficiency	Gain E...	VL (Ve...
[loop in main at roofline.cpp:53]	AVX 83%	3.30x	4

After		Loops	
Function Call Sites and Loops	Efficiency	Gain E...	VL (Ve...
[loop in main at roofline.cpp:53]	AVX2 100%	4.00x	4



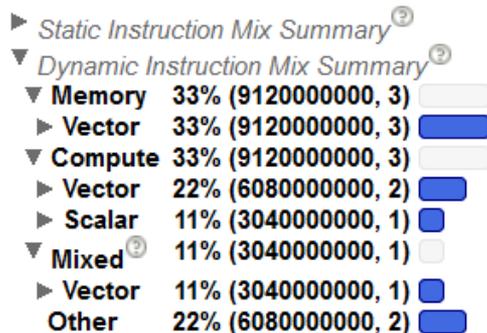
Assembly Detective Work

A Short Walk Through the Process

The dot is now sitting directly on the Vector Add Peak, so it is meeting but not exceeding the machine's vector capabilities. The next roof is the FMA peak. The Assembly tab shows that the loop is making good use of FMAs, too.

The Code Analytics tab reveals an unexpectedly high percentage of scalar compute instructions.

The only scalar math op present is in the loop control.



Source	Top Down	Code Analytics	Assembly	Recommendations	Why Not
Module: roofline_demo_samples.exe!0x140001124					
Address	Line	Assembly			
0x140001124		Block 1: 3040000000 [Ⓢ]			
0x140001124	55	vmovupd ymm4, ymmword ptr [r8+rcx*8+0x151e0]			
0x14000112e	55	vmovdqa ymm5, ymm1			
0x140001132	55	vfmadd213pd ymm5, ymm4, ymm2			
0x140001137	55	vfmadd231pd ymm5, ymm0, ymmword ptr [r8+rcx*8+0x177e0]			
0x140001141	55	vfmadd213pd ymm5, ymm4, ymm3			
0x140001146	55	vmovupd ymmword ptr [rax+rcx*8], ymm5			
0x14000114b	53	add rcx, 0x4			
0x14000114f	53	cmp rcx, 0x4c0			
0x140001156	53	jb 0x140001124 <Block 1>			

Loop Body

Loop Control

One More Optimization

A Short Walk Through the Process

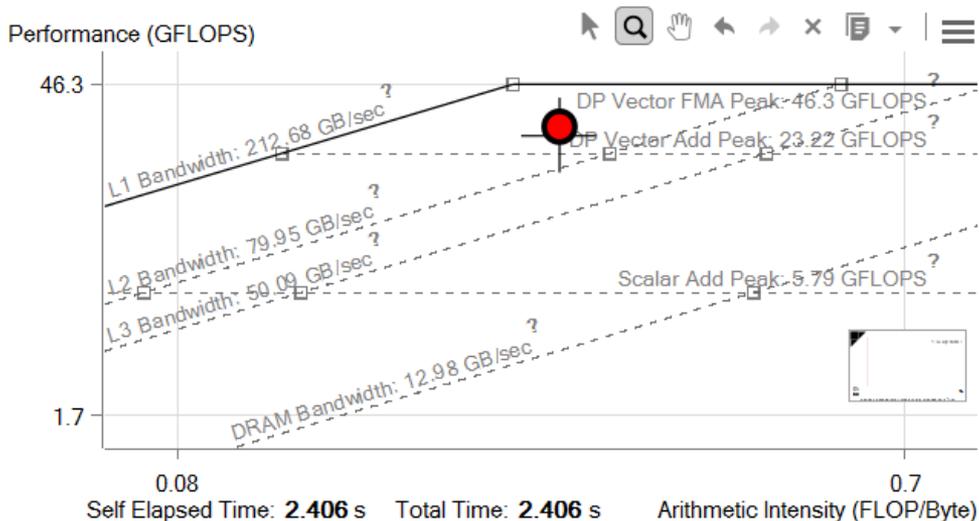
Scalar instructions in the loop control are slowing the loop down.

Unrolling a loop duplicates its body multiple times per iteration, so control makes up proportionately less of the loop.

Static Instruction Mix Summary

Dynamic Instruction Mix Summary

▼ Memory	47%	(9120000000, 24)	
▶ Vector	47%	(9120000000, 24)	52
▼ Compute	33%	(6460000000, 17)	
▶ Vector	31%	(6080000000, 16)	53
▶ Scalar	2%	(380000000, 1)	54
▼ Mixed	16%	(3040000000, 8)	
▶ Vector	16%	(3040000000, 8)	55
▶ Other	4%	(760000000, 2)	56
			57



```
#pragma unroll(8)
#pragma omp simd
for (int i = 0; i < SIZE; i++)
{
    X[i] = ((7.4 * Y.a[i] + 14.2) + Y.b[i] * 3.1) * Y.a[i] + 42.0;
}
```

Recap

A Short Walk Through the Process

17.156s

Original scalar loop.

9.233s

Vectorized with a pragma.

4.250s

Switched from AoS to SoA.

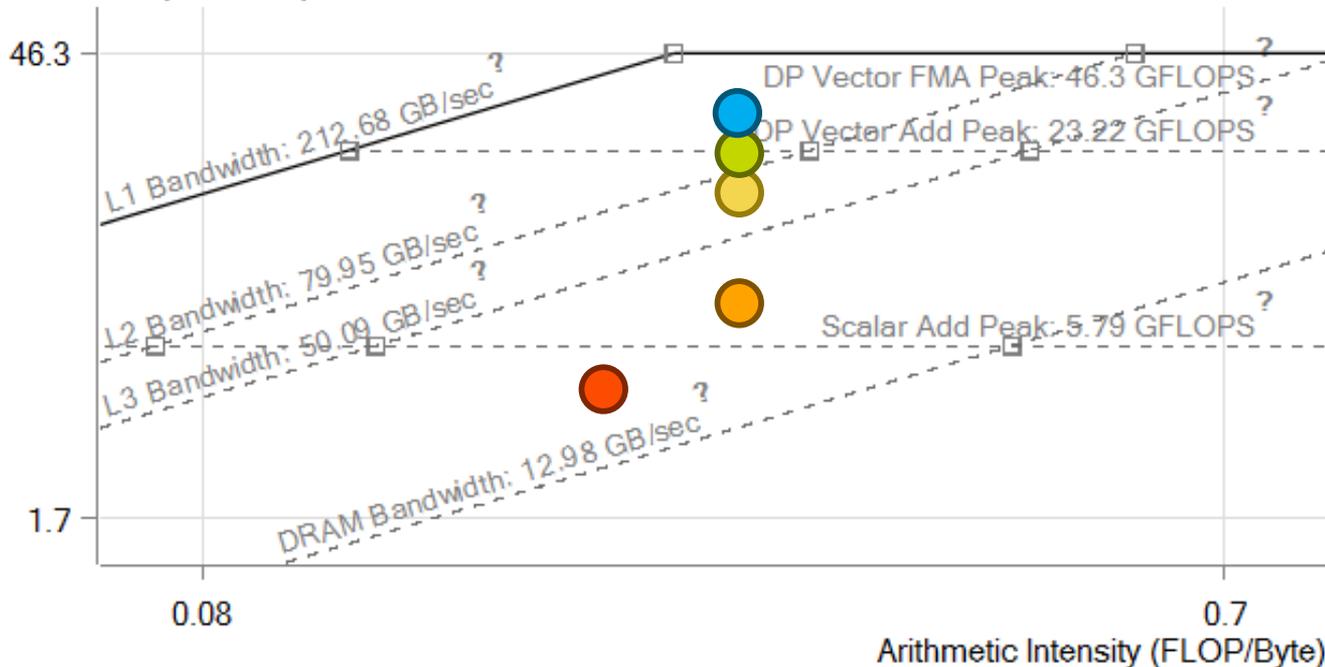
3.217s

Compiled for AVX2.

2.406s

Unrolled with a pragma.

Performance (GFLOPS)

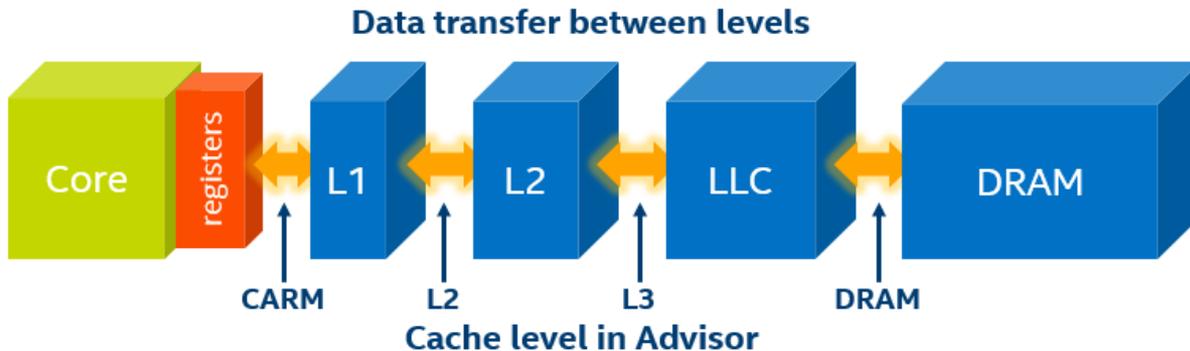


INTEGRATED ROOFLINE

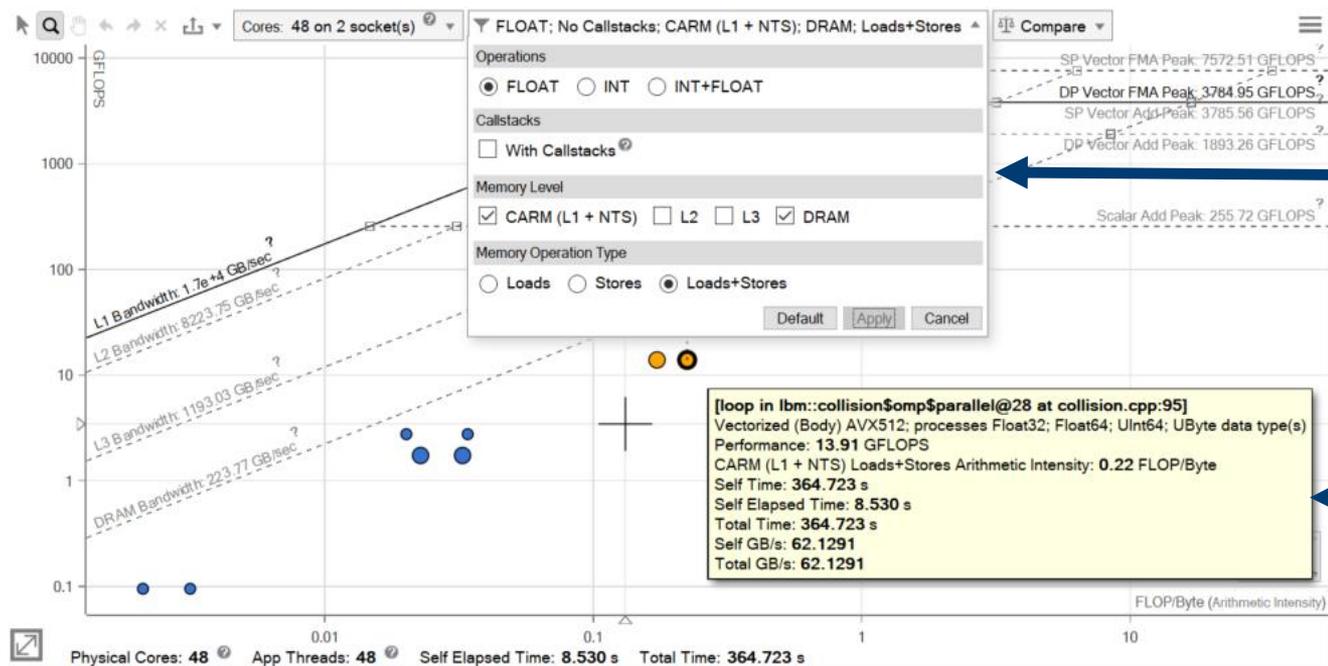
Beyond CARM: Integrated Roofline

New capability in Intel® Advisor: use simulation based method to estimate specific traffic across memory hierarchies.

- Record load/store instructions
- Use knowledge of processor cache structure and size
- Produce estimates of traffic generated at each level by individuals loops/functions



Integrated Roofline Representation



Choose memory level

Hover for details

New and improved summary

Program metrics

Elapsed Time	154.92s	▶ INT+FLOAT Giga OPS	11.89
Vector Instruction Set	AVX512, AVX2, AVX, SSE2, SSE	▶ GFLOPS	10.16
Number of CPU Threads	1	▶ GINTOPS	1.72

Effective Program Characteristics	Utilization	Hardware Peak
> GFLOPS	10.16 10%	out of 100.1 (DP) FLOPS 201.7 (SP) FLOPS
> GINTOPS	1.723 3.2%	out of 53.94 (Int64) INTOPS 106.2 (Int32) INTOPS
> CPU <-> Memory [L1+NTS GB/s]	34.71 1.2e+3%	out of 450.6 GB/s [bytes]

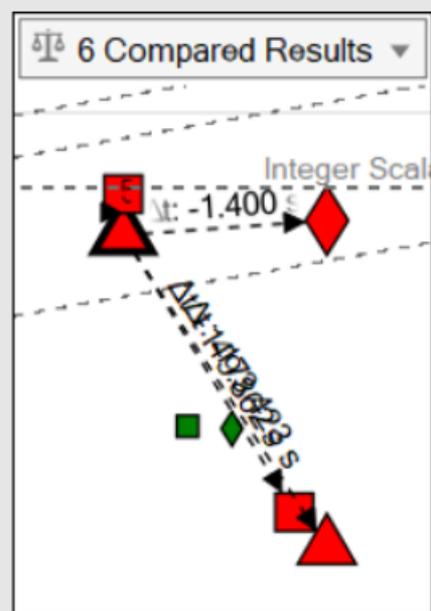
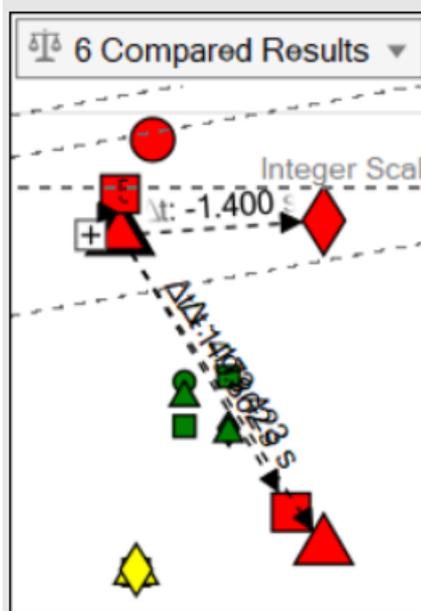
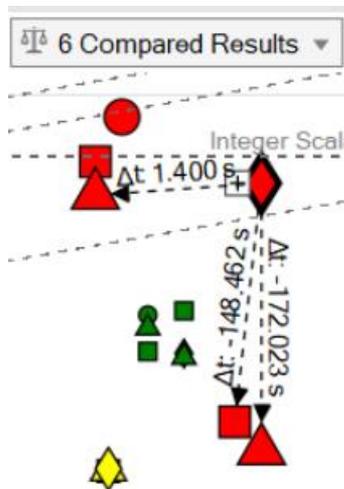
Performance characteristics

Metrics	Total	
Total CPU time	154.55s	100%
Time in 3 vectorized loops	142.89s	92.5%
Time in scalar code	11.66s	7.5%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency ^②	3.37x	42%
Program Approximate Gain ^②	3.19x	

Roofline compare



Legal Disclaimer & Optimization Notice

Performance results are based on testing as of 2/22/2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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

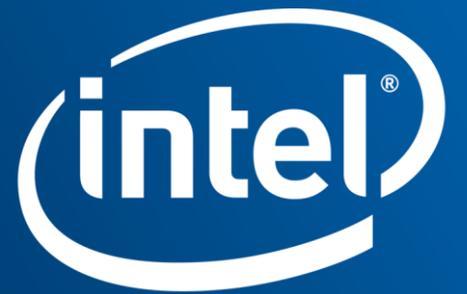
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright © 2019, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries 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