

Software

# Roofline: should I optimize for compute, memory or both?

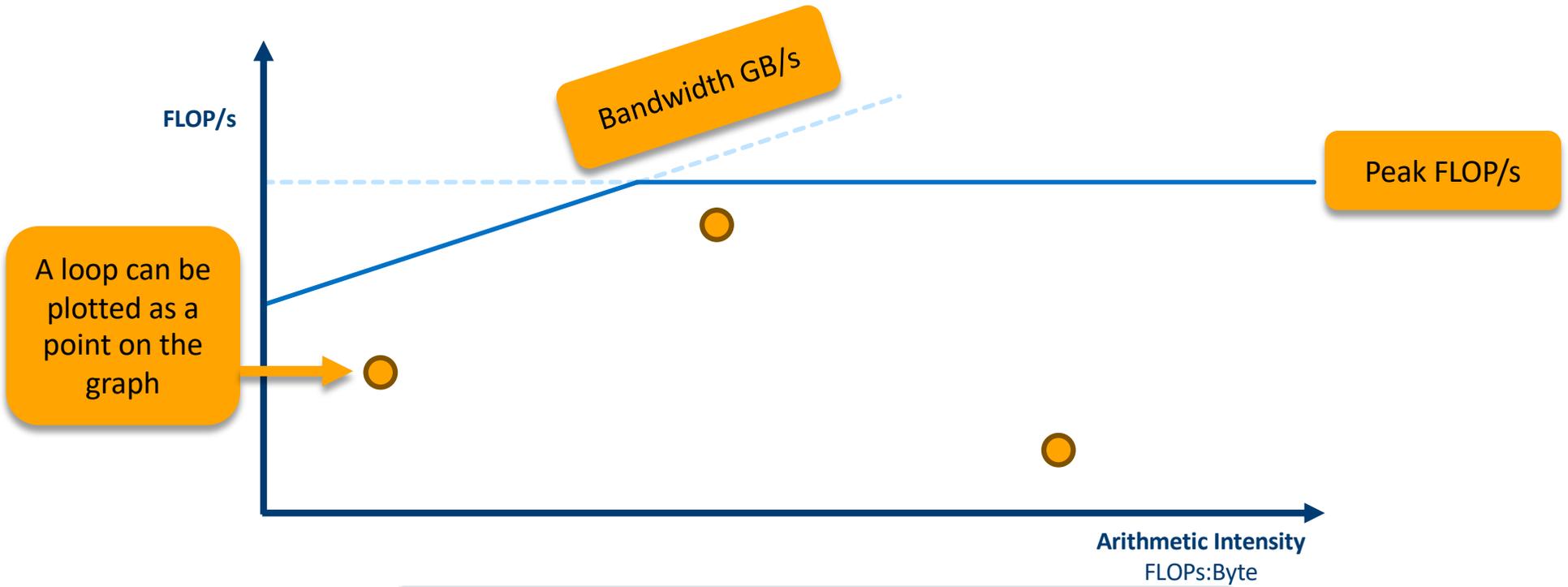
Dunni Aribuki, Technical Consulting Engineer  
Compute Performance & Developer Products Division, Intel



# Topics

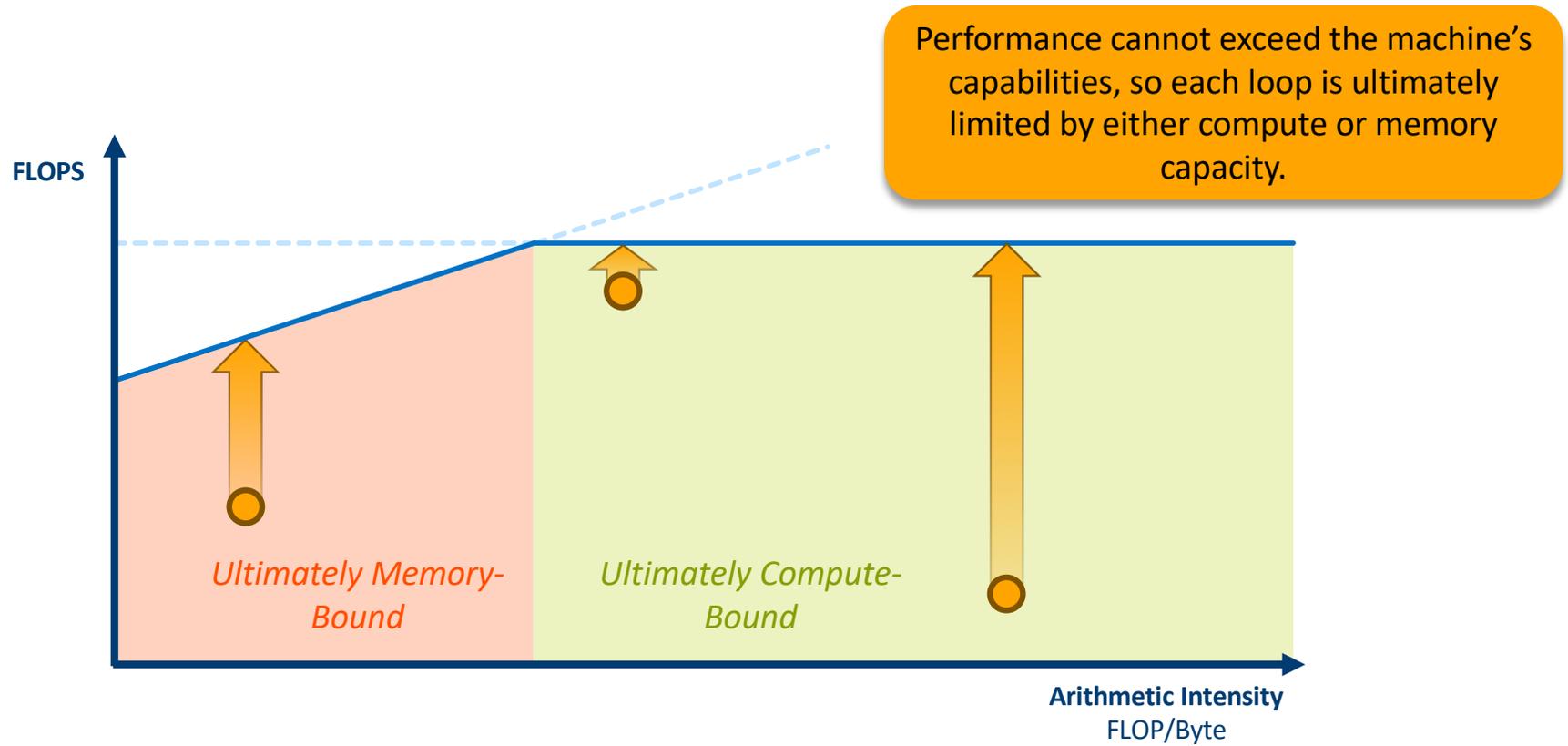
- Roofline Introduction
- Roofline in Intel® Advisor
  - Demo
- Optimizing Using Roofline

# Roofline Chart



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

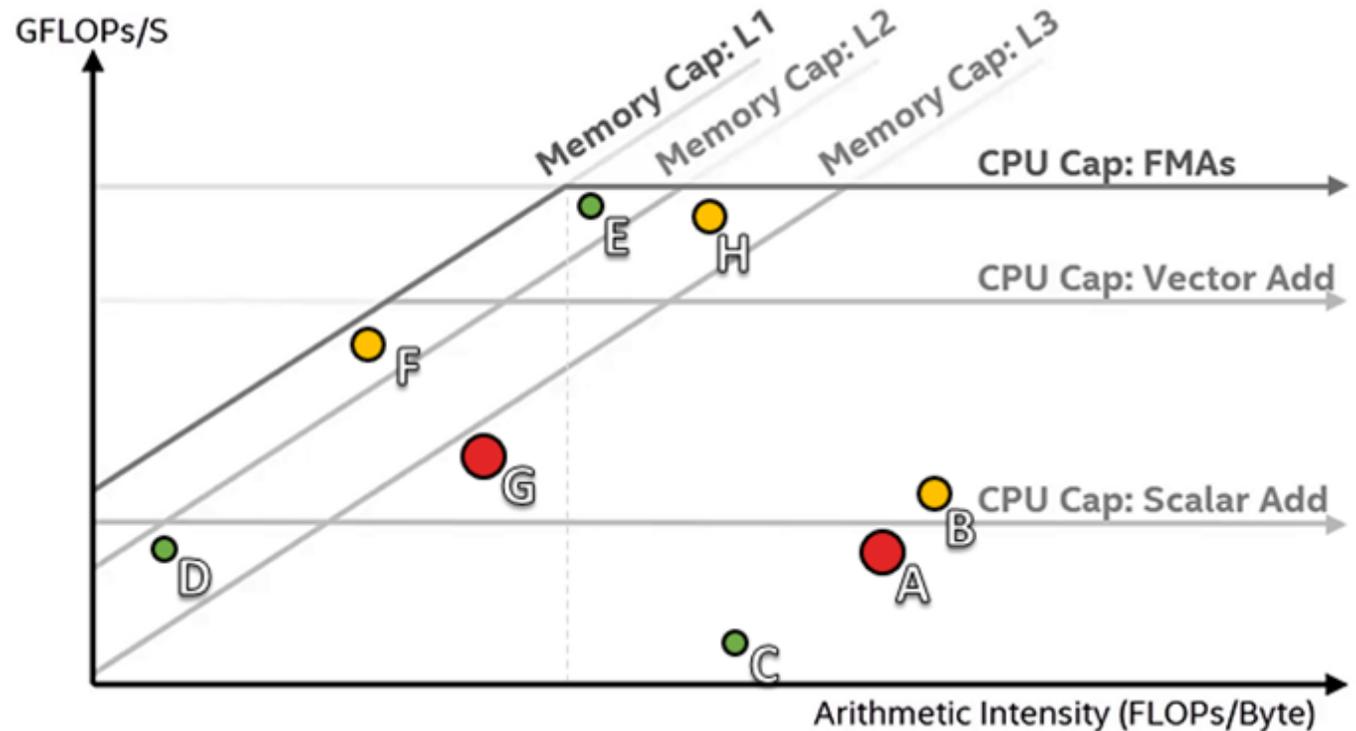
# Ultimate Performance Limits



# Identifying Good Optimization Candidates

Focus optimization effort where it makes the most difference.

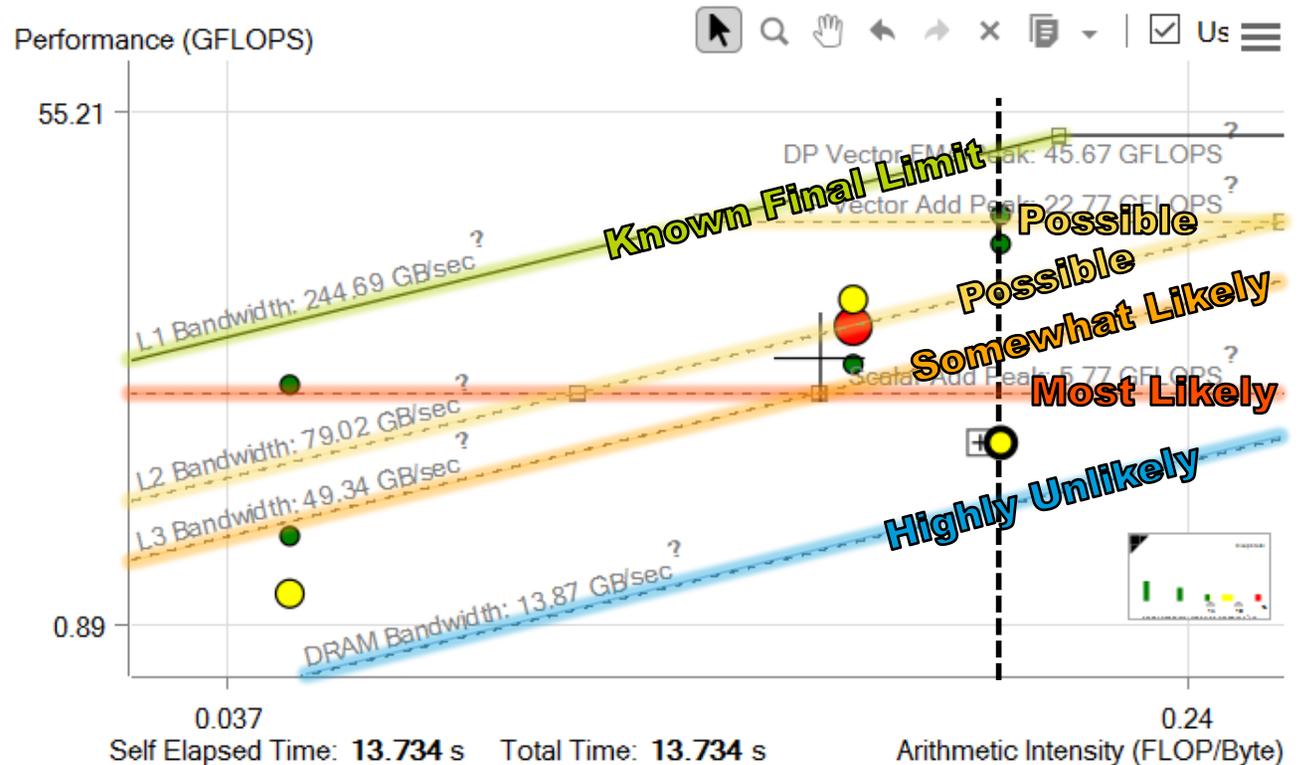
- Large, red loops have the most impact.
- Loops far from the upper roofs have more room to improve.



# Identifying Potential Bottlenecks

Final roofs *do* apply;  
sub-roofs *may* apply.

- Roofs above indicate *potential* bottlenecks
- Closer roofs are the most likely suspects
- Roofs below may contribute but are generally not primary bottlenecks

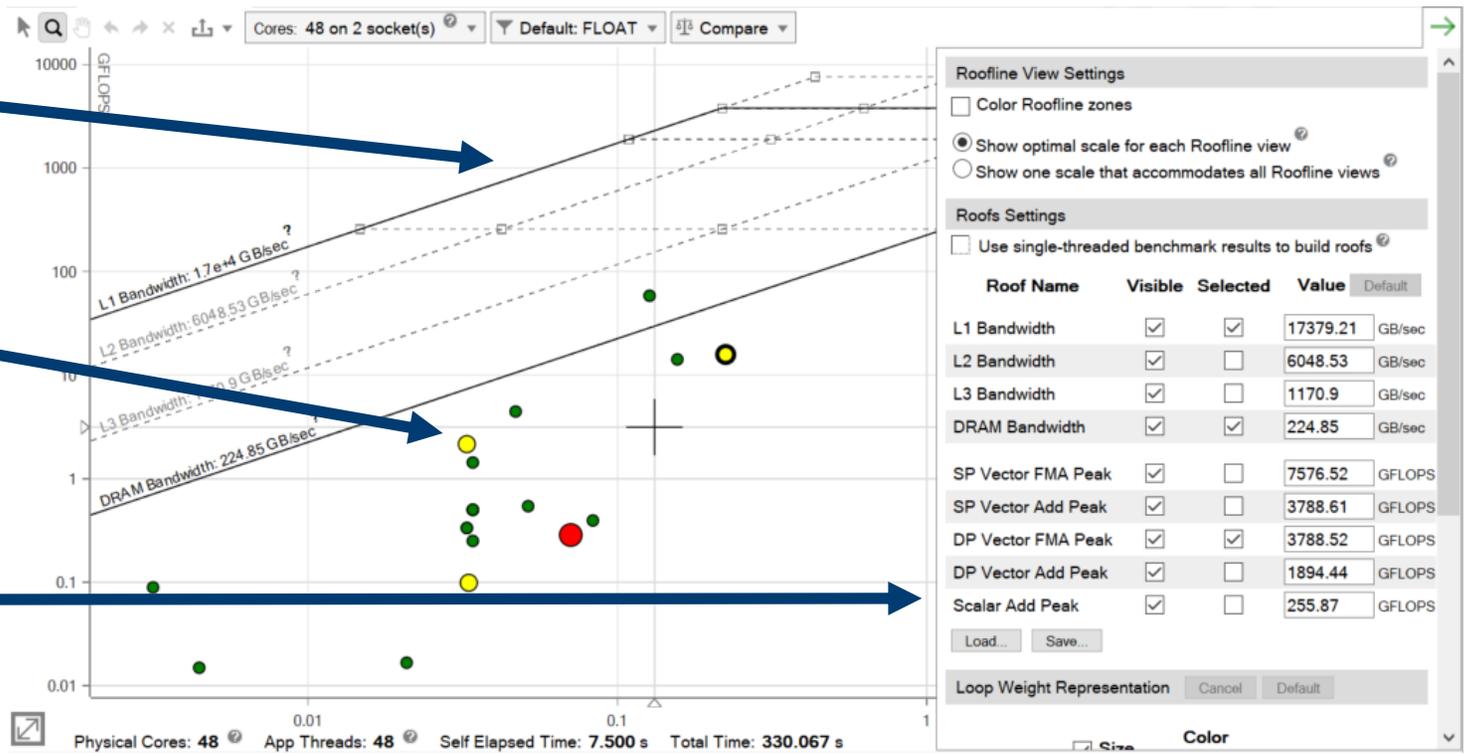


# Roofline Chart in Intel® Advisor

Roof values are measured

Dots represent profiled loops and functions

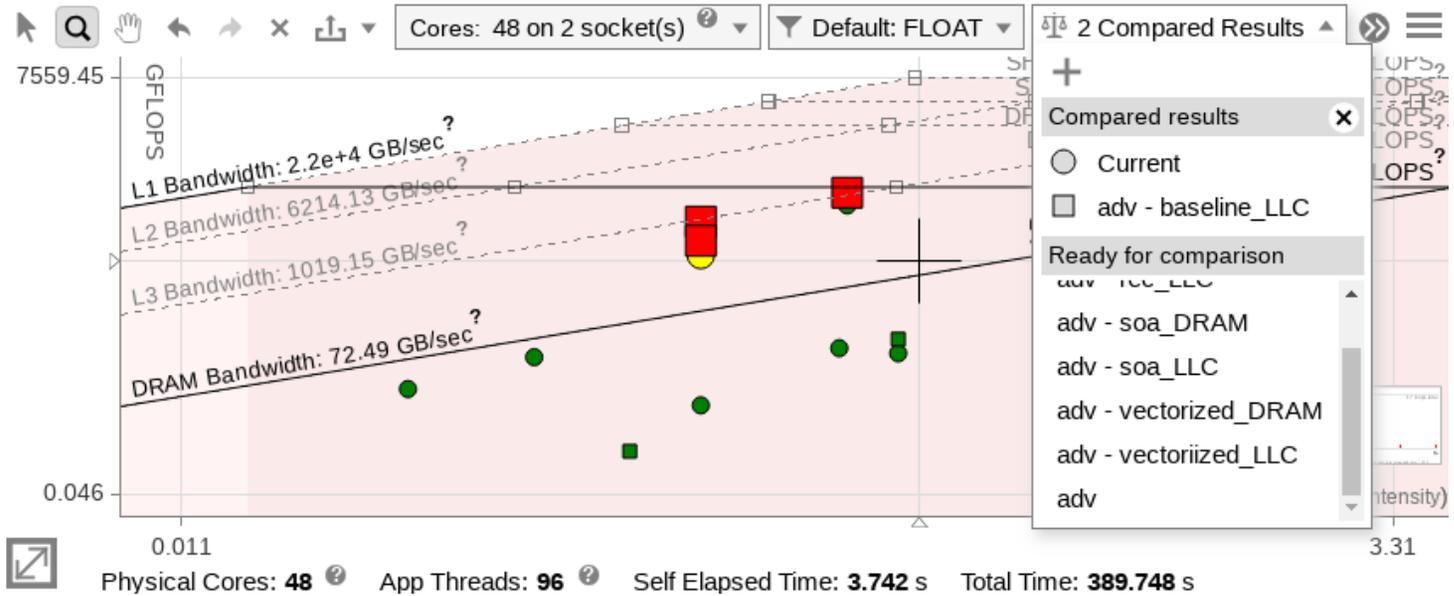
High level of customization



# Intel® Advisor New Feature: Roofline Compare

See multiple rooflines on one chart!

Track your optimization progress!



We would love to know what you think of this new feature!  
Email [vector\\_advisor@intel.com](mailto:vector_advisor@intel.com) and give us your feedback!

# Intel® Advisor roofline demo

# Tuning a real world Example with Roofline

A Walk-through of the Optimization Process

# Pica Library

*pica* is a C++ kernel library for particle-in-cell plasma simulation

The main routines are implemented for 1D, 2D and 3D Cartesian coordinates, are optimized and OpenMP-parallelized for multicore CPUs

Get the code of benchmarks from

<https://github.com/pictools/pica-benchmark>

*Work based on V. Volokitin, I. Surmin, S. Bastrakov, I. Meyerov  
Lobachevsky State University of Nizhni Novgorod, Russia*

# Pica Application

For each level of our optimization, we will do 2 runs:

- **LLC:**

- 01-app --nparticles 320000 --niterations 10000*

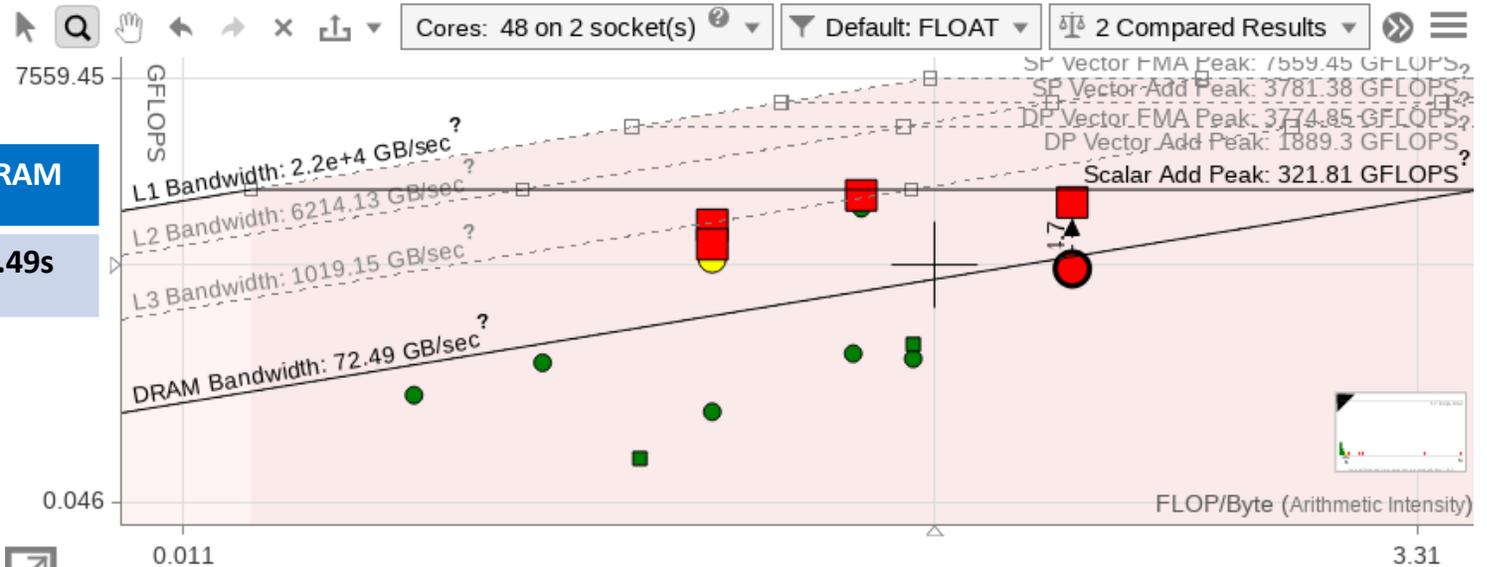
- **DRAM:**

- 01-app --nparticles 3200000 --niterations 1000*

# Step #1: Get baseline

Steps	LLC	DRAM
Baseline	0.48s	6.49s

- LLC run
- DRAM run



## 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

Run the [Dependencies analysis](#) to identify real data dependencies.

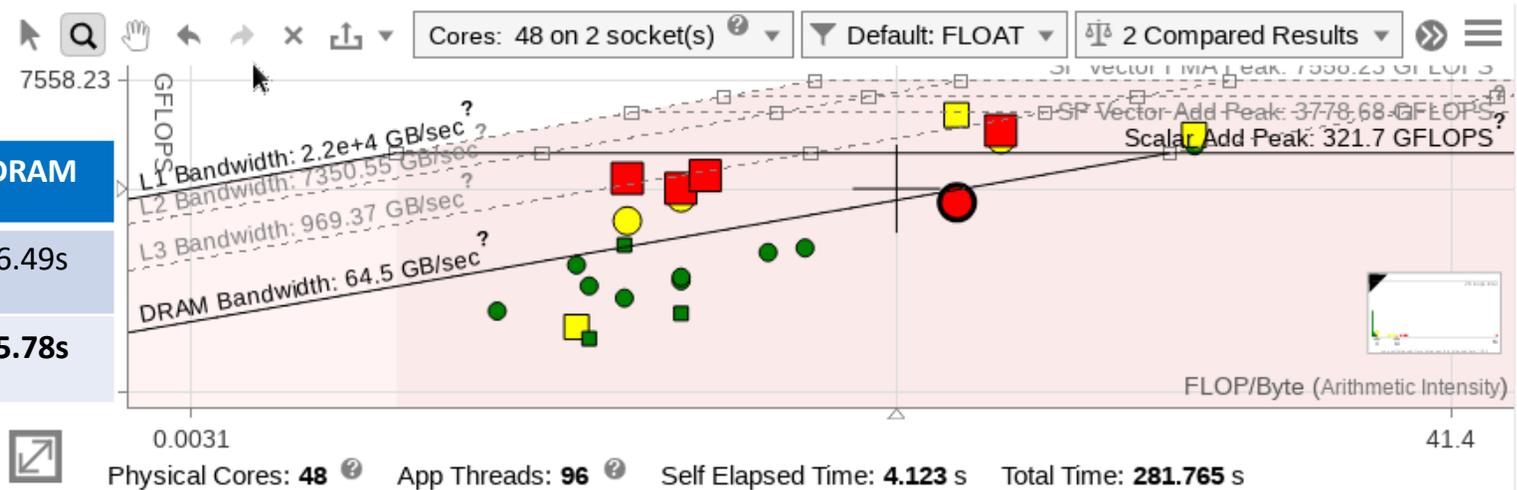
There is no confirmation that a real (proven) dependency is present in the loop.

# Step #2: Vectorize main bottleneck loop

## LLC

- Elapsed time 0.375s

Steps	LLC	DRAM
Baseline	0.48s	6.49s
<b>Vectorized</b>	<b>0.375s</b>	<b>5.78s</b>



□ LLC run

○ DRAM run

Vectorized Loops				
Vector ISA	Efficiency	Gain ...	VL (Vector Length)	Compiler Estimated Gain
AVX512	37%	2.93x	8	2.85x

Main loop is now vectorized but at only 37% efficiency. We need to investigate this!

# Step #3: Overcome memory bottlenecks

As previously mentioned, we are currently only vectorized at 37% efficiency. There are several features in the Intel® Advisor survey that can assist in our analysis.

You can also see a Performance Issue flagged.

Under the Code Analytics tab, you can see all of the instruction traits the compiler used in our loop.

The screenshot displays the Intel Advisor interface. At the top, a table titled "Vectorized Loops" shows the following data:

Vector ISA	Efficiency	Gain ...	VL (Vector Length)	Compiler Estimated Gain
AVX512	37%	2.93x	8	2.85x

Below the table, a "Performance Issues" section is highlighted with a blue bar, indicating "1 Possible inefficient memory access patterns present".

At the bottom, the "Code Analytics" tab is selected, showing a list of instruction traits:

- Divisions
- Square Roots
- Gathers
- Shuffles
- Blends
- FMA
- 2-Source Permutes
- Scatters

# Overcome memory bottlenecks

To overcome our vector inefficiency and make more effective use of our caches, we reorganized our data structure to use Structures of arrays instead of an array of structures.

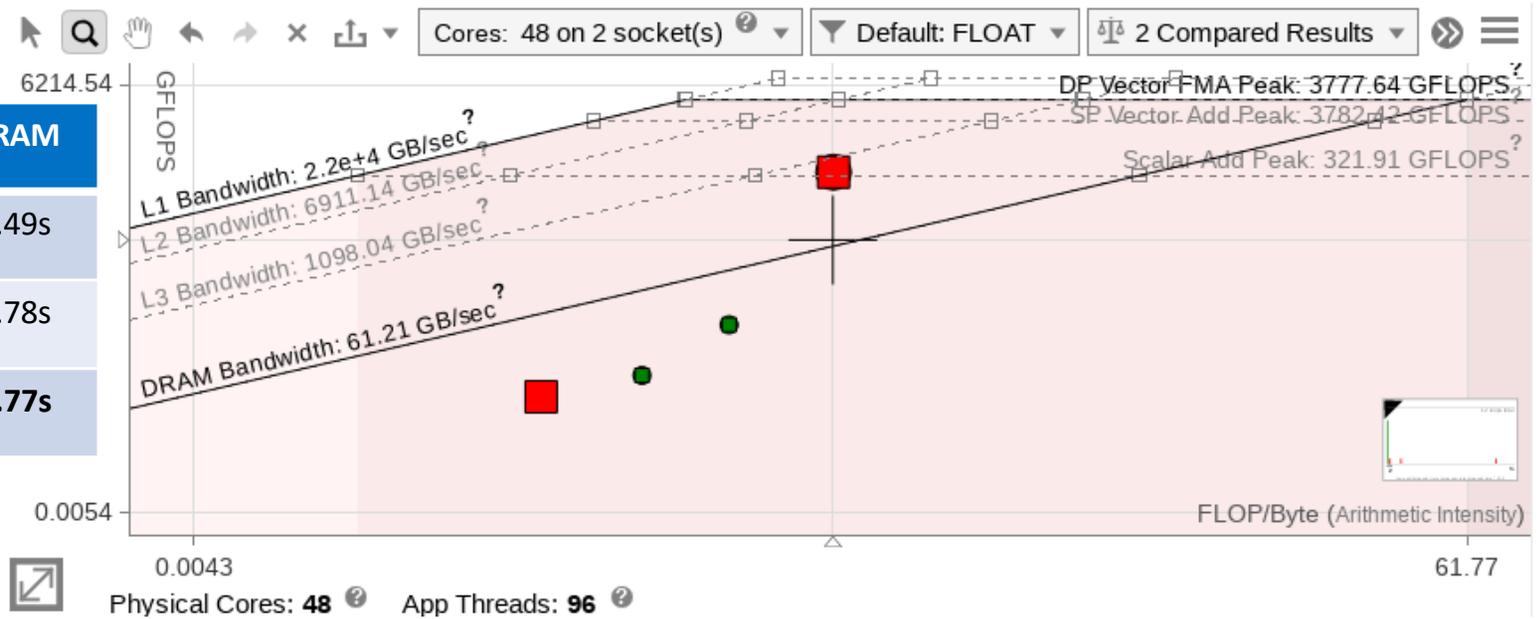
Doing this gives us a unit stride access.



# Overcome memory bottlenecks

Steps	LLC	DRAM
Baseline	0.48s	6.49s
Vectorized	0.375s	5.78s
Memory	<b>0.34s</b>	<b>5.77s</b>

- LLC run
- DRAM run



# Step #4: Optimize instructions

Use the **Code Analytics**, **Assembly** and **Recommendations** tabs to do a deep dive into the instructions we are using and to get advice on optimizing our code.

- Divisions and square roots take significant time
- Fast reciprocal instructions could improve performance

Address	Line	Assembly	Total Time	%	Self Time	%	
0x40906f	105	vfmadd213pd %zmm24, %zmm22, %k0, %zmm22	0,500s		0,500s		FMA
0x409075	105	vsqrtpd %zmm22, %k0, %zmm22	0,677s		0,677s		Square Roots
0x40907b	107	vdivpd %zmm22, %zmm18, %k0, %zmm23	5,779s		5,779s		Divisions
0x409081	108	vmovupsz (%r10,%r13,8), %k0, %zmm22	4,480s		4,480s		
0x409088	108	vfmadd231pd %zmm23, %zmm25, %k0, %zmm22	0,010s		0,010s		FMA

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

## ! Unoptimized floating point operation processing possible

Improve performance by enabling approximate operations instructions.

### Enable the use of approximate division instructions

Static analysis presumes the loop may benefit from using approximate calculations. Independent divisors will be pre-calculated and replaced with multipliers. To fix: Fine-tune your usage of the following compiler option:

### Enable the use of approximate sqrt instructions

Static analysis presumes the loop may benefit from using approximate sqrt instructions, but the precision and floating-point model settings may prevent the compiler from using these instructions. To fix: Fine-tune your usage of the following compiler option:

# Optimize instructions

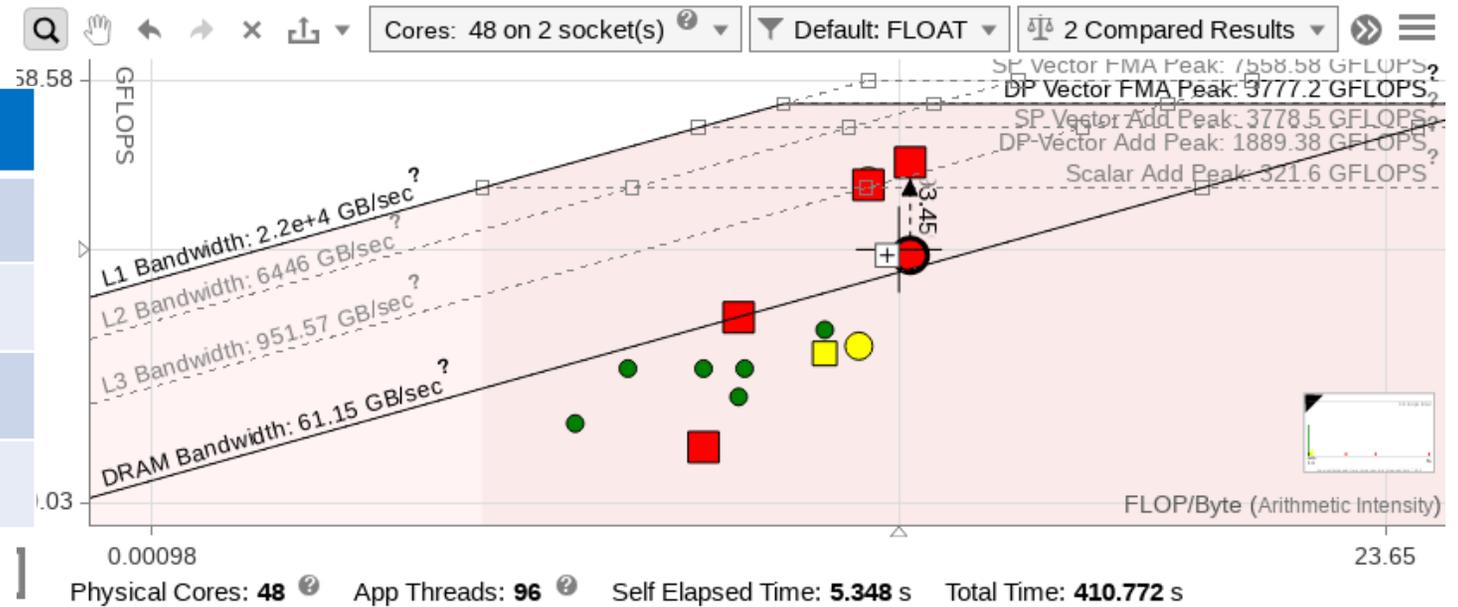
Divisions: recompile with `-no-prec-div`

File: Main.cpp:83 main\$omp\$parallel_for@75							
Line	Source	Total Time	%	Loop/Function Time	%	Traits	
97	<code>double uprimeY = umY + umY * tX - umX * tZ;</code>	1,160s				FMA	
98	<code>double uprimeZ = umZ + umX * tY - umY * tX;</code>	0,080s				FMA	
99	<code>cf = 2.0 / (1.0 + tX * tX + tY * tY + tZ * tZ);</code>	4,279s				Appr. Re...	
100	<code>double sX = tX * cf;</code>	0,650s					
101	<code>double sY = tY * cf;</code>	0,100s					
Selected (Total Time):		4,279s					
Module: pusher-vectorized-soa-rcp!0x408a3c							
Address	Line	Assembly	Total Time	%	Self Time	%	Traits
0x40925a	99	<code>vfmadd231pd %zmm3, %zmm3, %k0, %zmm20</code>	0,010s		0,010s		FMA
0x409260	99	<code>vfmadd231pd %zmm4, %zmm4, %k0, %zmm20</code>	0,080s		0,080s		FMA
0x409266	99	<code>vrcp14pd %zmm20, %k0, %zmm1</code>	0,649s		0,649s		Appr. Re...
0x40926c	99	<code>vfmadd213pdq 0x24ce2(%rip){1to8}, %zmm1, %k0, %zmm20</code>	1,320s		1,320s		FMA
0x409276	99	<code>vfpclasspd \$0x1e, %zmm1, %k0, %k0</code>	0,590s		0,590s		

Inverse square roots: use the `invsqrt` call from SVML

# Optimize instructions

Steps	LLC	DRAM
Baseline	0.48s	6.49s
Vectorized	0.375s	5.78s
Memory	0.34s	5.77s
<b>Optimized</b>	<b>0.30s</b>	<b>5.59s</b>



- LLC run
- DRAM run

# Results

Steps	LLC	DRAM
1-Baseline	0.48s	6.49s
2-Vectorized	0.375s	5.78s
3-Memory	0.343s	5.77s
4-Optimized	0.30s	5.59

# Summary

You need to optimize for both compute and memory

The Roofline chart is a powerful feature to visualize your compute and memory bottlenecks

Intel® Advisor plots Roofline charts automatically and provides various additional analysis and recommendations to guide optimizations

The Intel® Advisor's Roofline Compare feature lets you easily test different configurations

# Next Steps & Resources

[Download Intel® Advisor and start optimizing!](#)

[Intel® Advisor - Home Page](#)

[Intel® Advisor Roofline](#)

[Intel® Advisor Cookbook](#)

# Legal Disclaimer & Optimization Notice

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](http://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 © 2018, 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

# QUESTIONS AND ANSWERS