Fast insights to optimized vectorization and memory using cache-aware roofline analysis

Kevin O’Leary, Intel Technical Consulting Engineer

Fast insights to optimized vectorization and memory using cache-aware roofline analysis

- The Roofline model
- Intel® Advisor Roofline analysis
- Intel® Advisor Demo
- Intel® Advisor Case Study
- Intel® Advisor “What’s New” and 2018 beta
- Summary

The Roofline model

Acknowledgments/References

Roofline model proposed by Williams, Waterman, Patterson: http://www.eecs.berkeley.edu/~waterman/papers/roofline.pdf

"Cache-aware Roofline model: Upgrading the loft" (Ilic, Pratas, Sousa, INESC-ID/IST, Thec Uni of Lisbon) http://www.inesc-id.pt/ficheiros/publicacoes/9068.pdf

At Intel:

Roman Belenov, **Zakhar Matveev**, Julia Fedorova
SSG product teams, Hugh Caffey,
in collaboration with **Philippe Thierry**

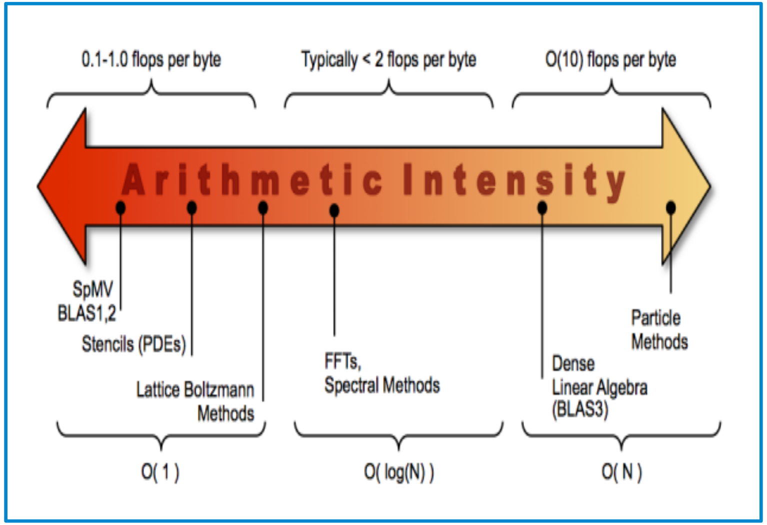
Roofline Model – A visually intuitive performance model

Combines

- memory utilization/demand
- CPU utilization

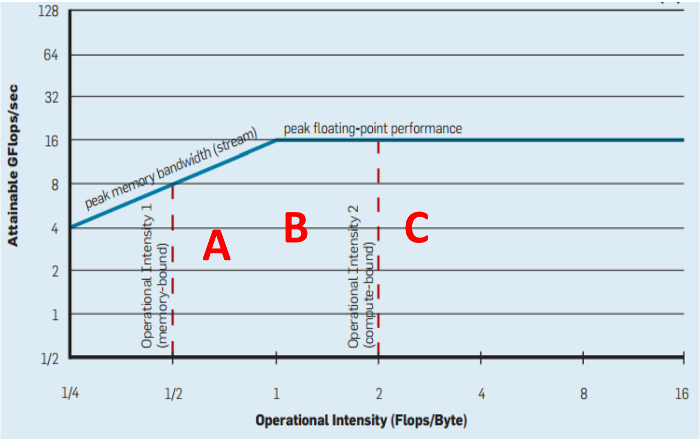
Into the same performance analysis

and modeling space

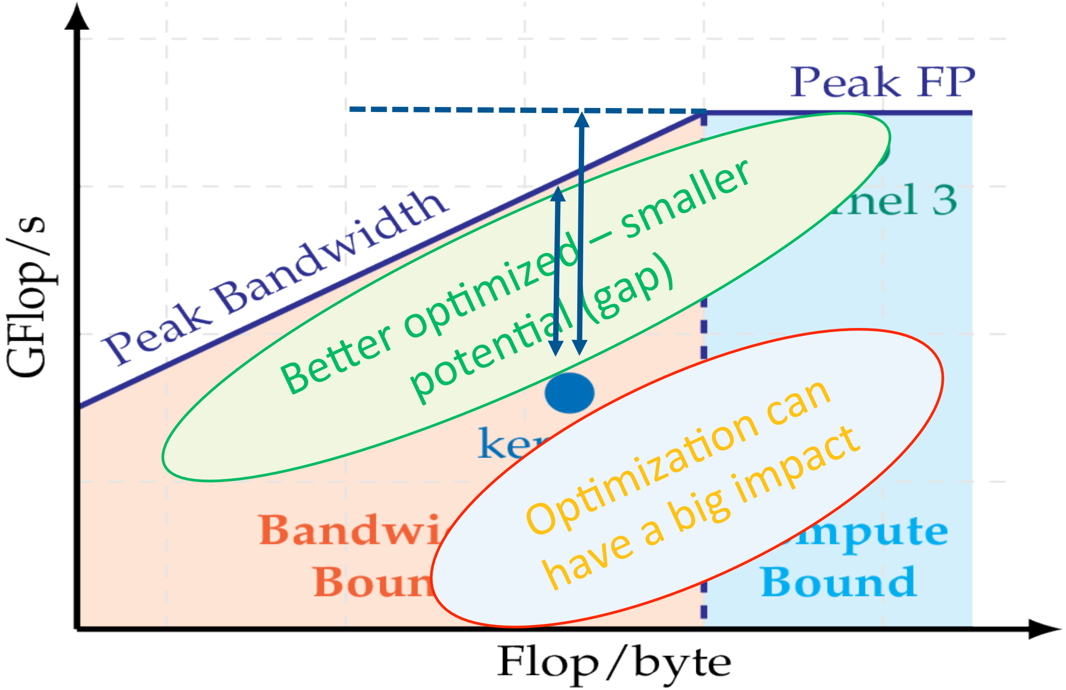


Arithmetic Intensity (AI) = # FLOPs / # BYTEs

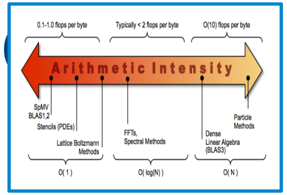
Roofline model: Am I bound by VPU/CPU or by Memory?



What makes loops
A, B, C different?



Cache-Aware vs. Classic Rooflin

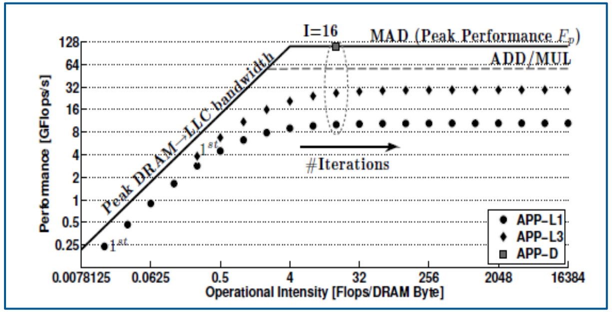


AI = # FLOP / # BYTE

AI_DRAM =

 # FLOP/ # BYTES (CPU & Cache ⇔ **DRAM)**

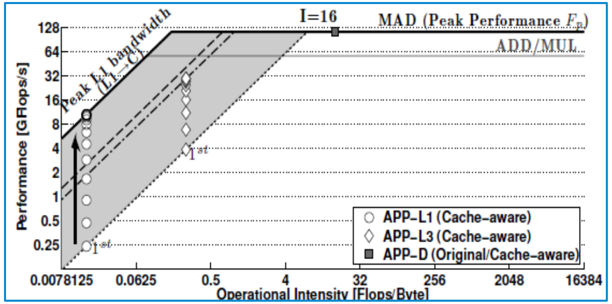
- “DRAM traffic” (or MCDRAM-traffic-based)
- Variable for the same code/platform (varies with dataset size/trip count)
- Can be measured relative to different memory hierarchy levels – cache level, HBM, DRAM



AI_CARM =

 # FLOP / # BYTES (CPU ⇔ Memory Sub-system**)**

- “Algorithmic”, “Cumulative (L1+L2+LLC+DRAM)” traffic-based
- Invariant for the given code / platform combination
- Typically AI_CARM < AI_DRAM

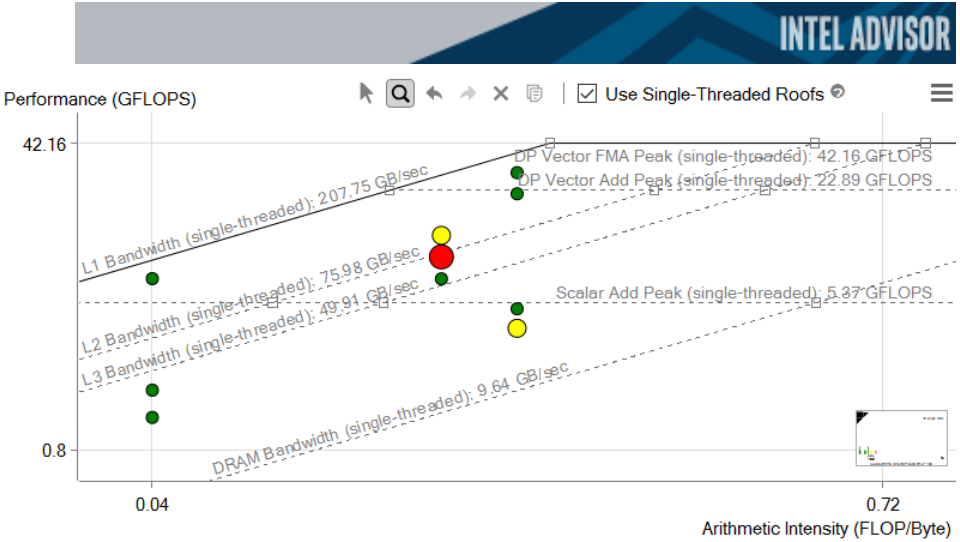


Intel® Advisor roofline analysis

Find Effective Optimization Strategies

Intel Advisor: Cache-aware roofline analysis

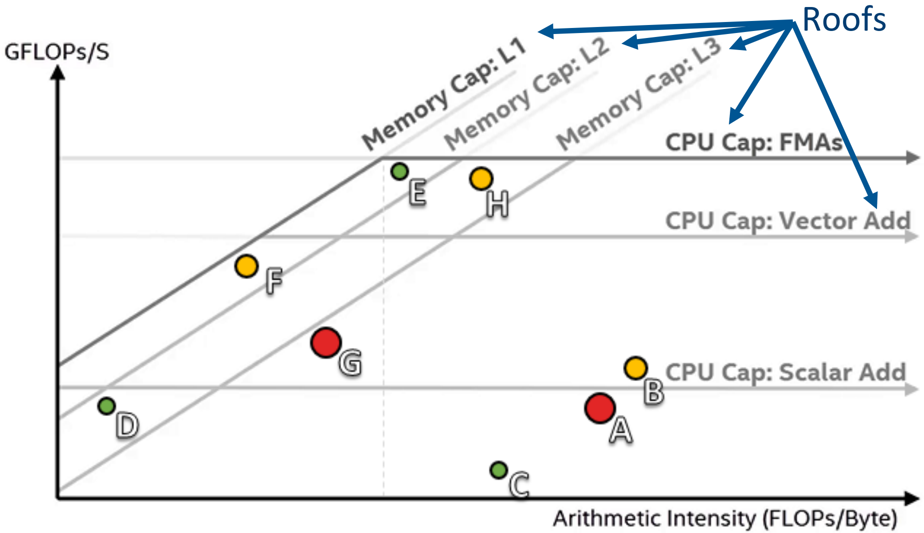
- Roofline Performance Insights
  - Highlights poor performing loops
  - Shows performance “headroom” for each loop
    - Which can be improved
    - Which are worth improving
  - Shows likely causes of bottlenecks
  - Suggests next optimization steps



Find Effective Optimization Strategies

Intel Advisor: Cache-aware roofline analysis

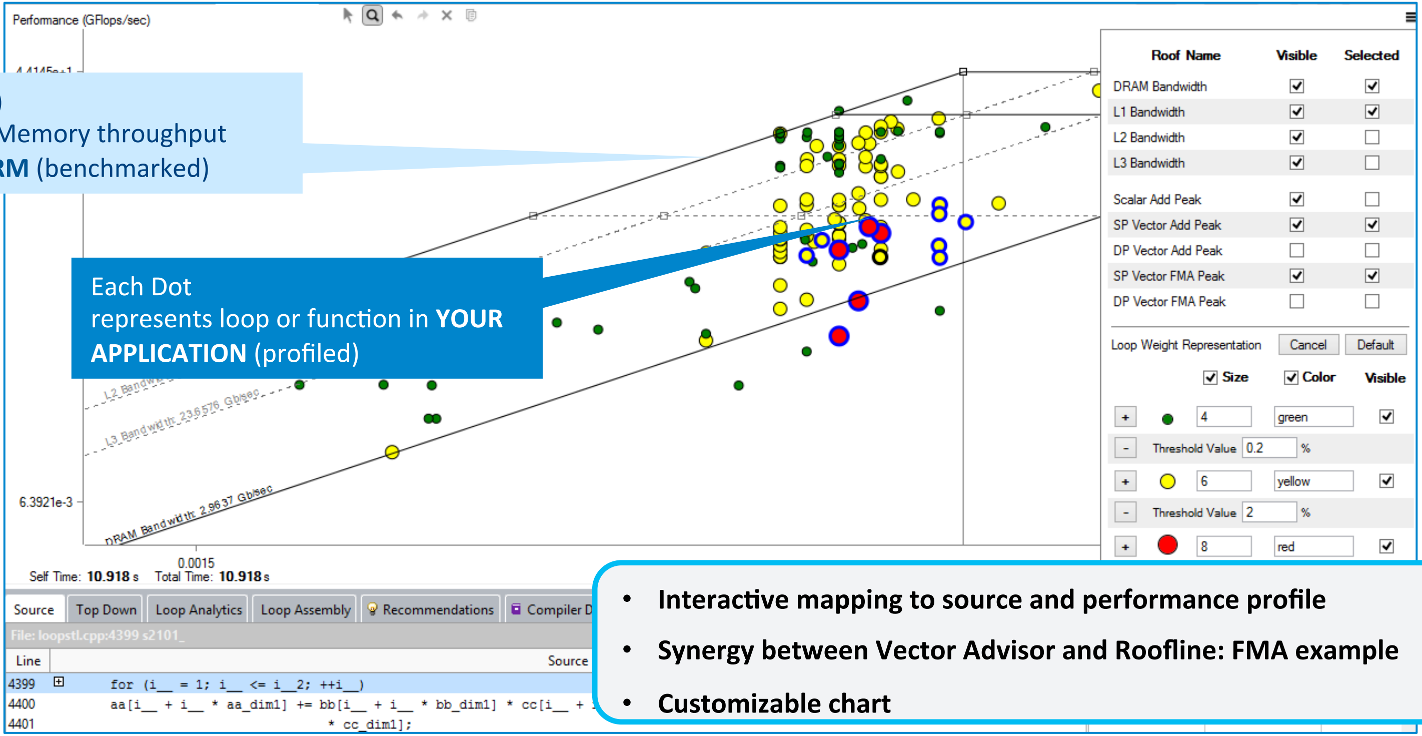
- Roofs Show Platform Limits
  - Memory, cache & compute limits
- Dots Are Loops
  - Bigger, red dots take more time so optimization has a bigger impact
  - Dots farther from a roof have more room for improvement
- Higher Dot = Higher GFLOPs/sec
  - Optimization moves dots up
  - Algorithmic changes move dots horizontally



Which loops should we optimize?

- A and G are the best candidates
- B has room to improve, but will have less impact
- E, C, D, and H are poor candidates

Roofline Automation in Intel Advisor 2017 update 2 and 2018 beta



Each Roof (slope)
Gives peak CPU/Memory throughput
of your **PLATFORM** (benchmarked)

Each Dot
represents loop or function in **YOUR APPLICATION** (profiled)

- **Interactive mapping to source and performance profile**
- **Synergy between Vector Advisor and Roofline: FMA example**
- **Customizable chart**

Intel® Advisor Roofline: under the hood

Roofline application profile:

Axis Y: **FLOP/S = #FLOP** (mask aware) / **#Seconds**

Axis X: **AI = #FLOP / #Bytes**

Seconds

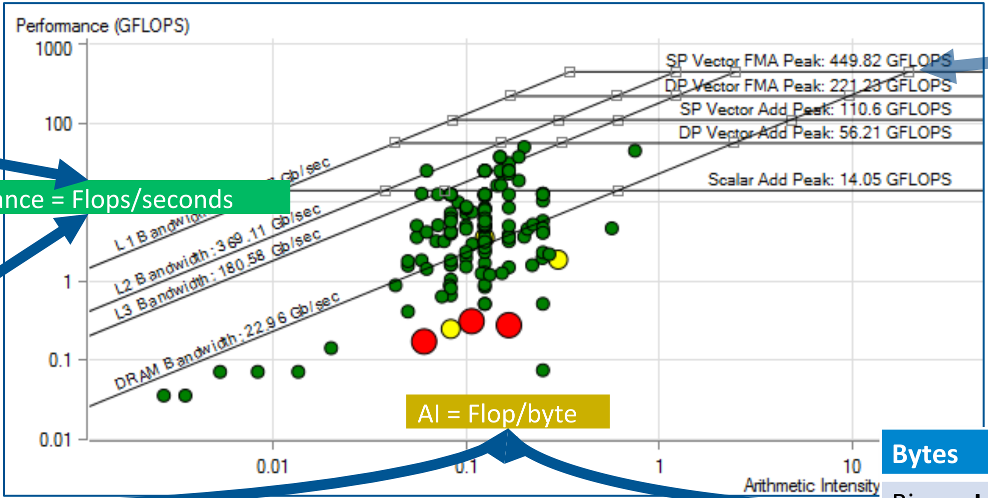
User-mode **sampling**

Root access not needed

Roofs

Microbenchmarks

Actual peak for the current configuration



#FLOP

Binary **Instrumentation**

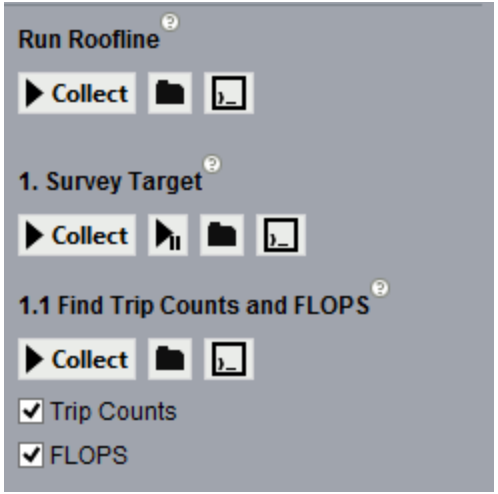
Does not rely on CPU counters

Bytes

Binary **Instrumentation**

Counts operands size (not cachelines)

Getting Roofline data in Intel®Advisor



FLOP/S **= #FLOP/Seconds**	**Seconds**	**#FLOP** **- Mask Utilization** **- #Bytes**
Step 1: Survey - Non intrusive. *Representative* - Output: Seconds (+much more)	✓	
Step 2: Trip counts+FLOPS - Precise, instrumentation based - Physically count Num-Instructions - Output: #FLOP, #Bytes		✓

Interpreting Roofline Data

Final Limits

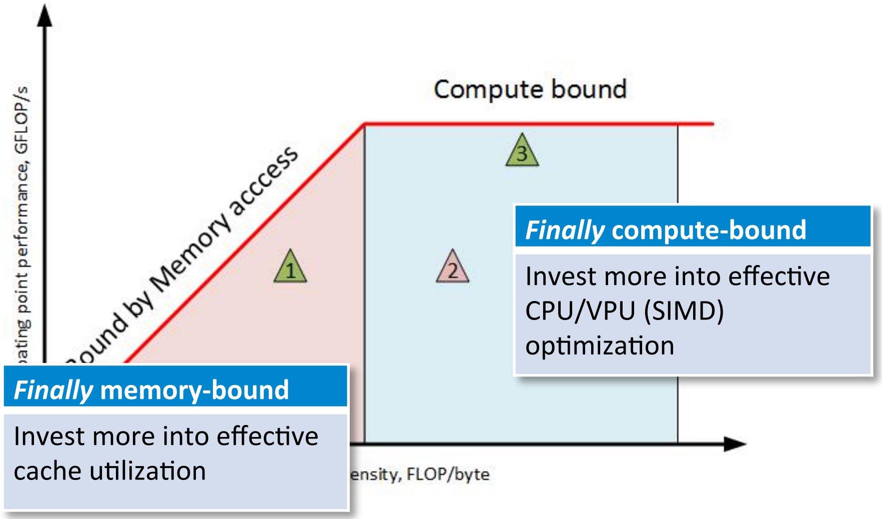
(assuming perfect optimization)

Long-term ROI, optimization strategy

Current Limits

(what are my current bottlenecks)

Next step, optimization tactics

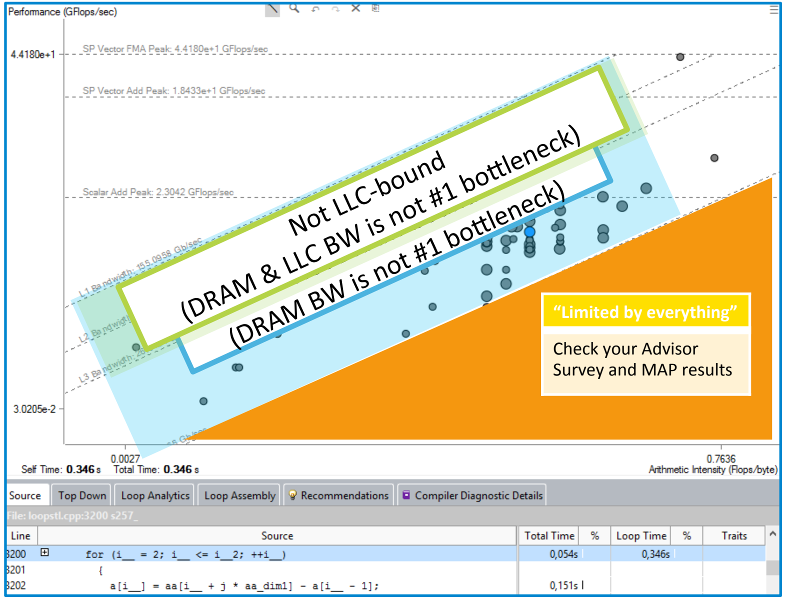


***Finally* compute-bound**

Invest more into effective CPU/VPU (SIMD) optimization

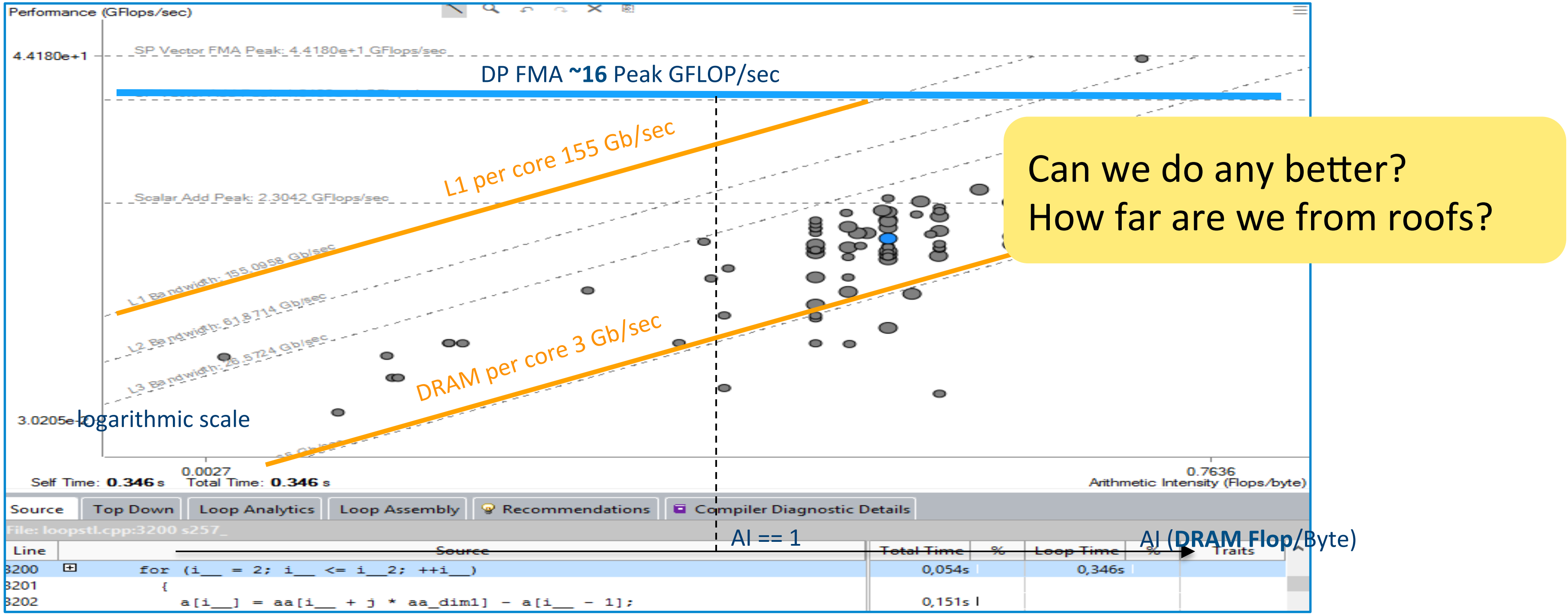
***Finally* memory-bound**

Invest more into effective cache utilization



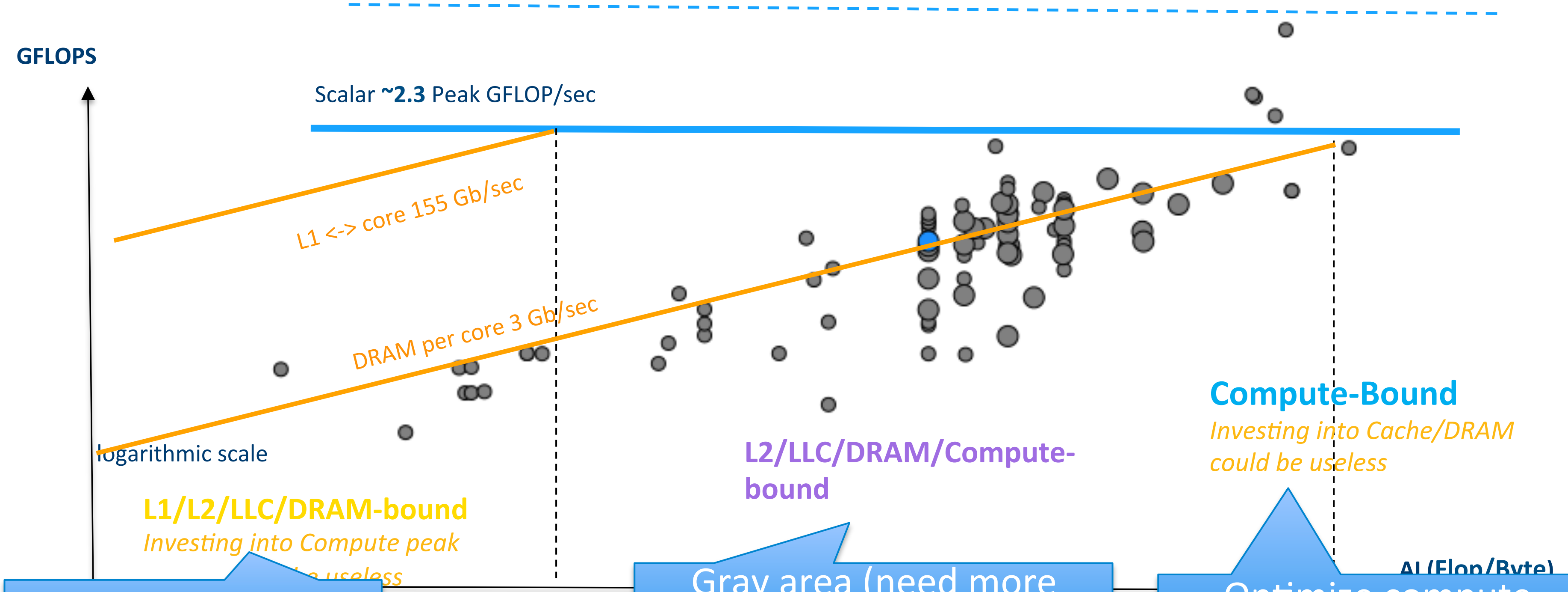
What are my memory and compute peaks?

How far away from peak system performance is my application?



Perform the right optimization for your region

Roofline: characterization regions



Optimize memory (cache blocking, etc)

Gray area (need more data to determine right strategy)

Optimize compute (threading, vectorization, etc)

Intel®Advisor demo

Intel®Advisor case studY

Roofline Analysis to Tune an MRI Image Reconstruction Benchmark

The 514.pomriq SPEC ACCEL Benchmark

- An MRI image reconstruction kernel described in Stone et al. (2008). MRI image reconstruction is a conversion from sampled radio responses to magnetic field gradients. The sample coordinates are in the space of magnetic field gradients, or K-space.
- The algorithm examines a large set of input, representing the intended MRI scanning trajectory and the points that will be sampled.
- The input to 514.pomriq consists of one file containing the number of K-space values, the number of X-space values, and then the list of K-space coordinates, X-space coordinates, and Phi-field complex values for the K-space samples.

Hot loop is vectorized

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.

Program metrics

Elapsed Time: 36.93s

Vector Instruction Set: AVX512

Number of CPU Threads: 136

Total GFLOP Count: 19293.90

Total GFLOPS: 522.51

Loop metrics

Total CPU time	4267.88s	100.0%
Time in **1** vectorized loop	4206.25s	98.6%
Time in scalar code	61.62s	

Vectorization Gain/Efficiency (Not available)

Top time-consuming loops

Loop	Self Time	Total Time	Trip Counts
[loop in ComputeQCPU at computeQ.c:65]	1957.548s	4206.254s	12500
[loop in ComputeQCPU at computeQ.c:58]	6.963s	4213.216s	15420
[loop in outputData at file.c:70]	0.040s	4.160s	2097152
[loop in start_thread at ?]	0s	49.660s	
[loop in [OpenMP worker at z_Linux_util.c:769]	0s	49.660s	

Refinement analysis data

These loops were analyzed for memory access patterns and dependencies:

Site Location	Dependencies	Strides Distribution
[loop in ComputeQCPU at computeQ.c:66]	No information available	96% / 0% / 4%

Collection details

Platform information

CPU Name:	Intel(R) Xeon Phi(TM) CPU 7250 @000000 1.40GHz
Frequency:	1.40 GHz
Logical CPU Count:	272
Operating System:	Linux

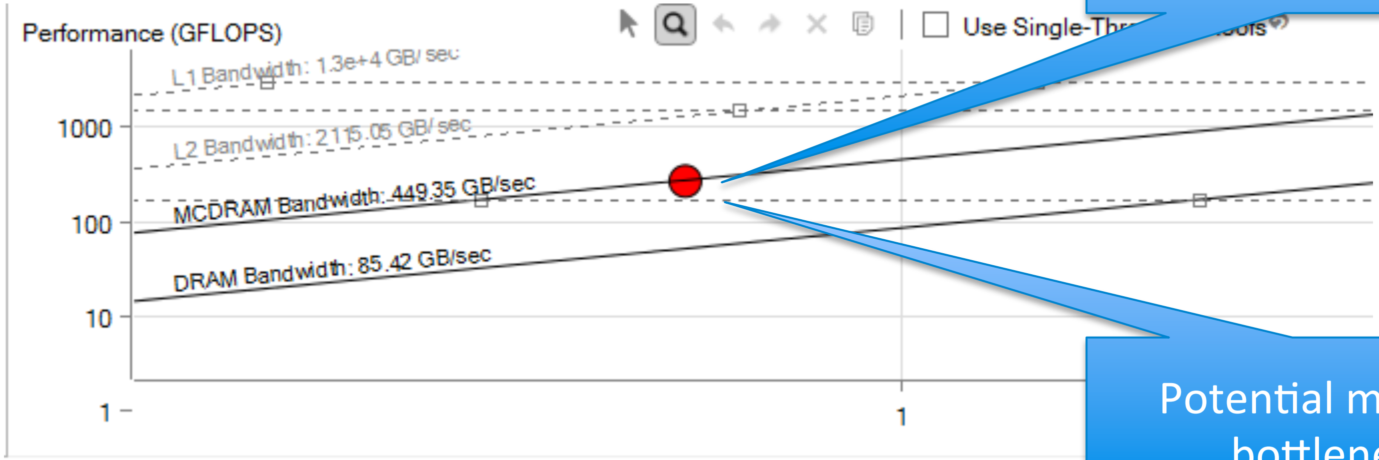
Intel Advisor summary view

1 vectorized loop that we spend 98.8% of our time in

Need more information to see if we can get more performance

What is our performance?
Relative to peak system performance

Our hot loop is below the MCDRAM roof



Potential memory bottleneck

Get detailed Advice from intel® Advisor

Intel® Advisor code analytics

Loop in ComputeQCPU at computeQ.c:65

4206.254s
Vectorized (Body) *Total time*

AVX512F_512 1957.548s
Instruction Set *Self time*

- Memory 16% (4)
- Compute 33% (8)
- Other 51% (12)

Instruction Mix Summary

Traits

Gathers, FMA, Mask Manipulations

Average Trip Counts: 12500

Instruction Mix

Memory: 4 *Compute:* 8 *Other:* 12 *Number of Vector Registers:* 13

Statistics for FLOPS And Data Transfers

GFLOPS	***266.242***	Giga Floating-point Operations Per Second Per-loop GFLOPS = Total FLOP / Elapsed Time. Elapsed time is the exclusive (self-time-based) wall time from the beginning to the end of loop/function execution. For single-threaded applications Elapsed time is equal to Self-Time.
AI	***0.606***	AI - Arithmetic Intesity - Ratio of Floating-point Operations to L1 Transferred Bytes
Mask Utilization	***100***	Ratio of Utilized Vector Elements to Total Vector Elements
GFLOP	***4194.304***	Giga Floating-point Operations
FLOP Per Iteration	***160***	Floating-point Operations Per Loop Iteration

Data transfers between CPU and memory sub-system (total traffic, including L1, L2, LLC and DRAM traffic)

Possible inefficient memory access

Gather stride access!

Issue: Possible inefficient memory access patterns present

Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

Recommendation: Confirm inefficient memory access patterns Confidence: Need More Data

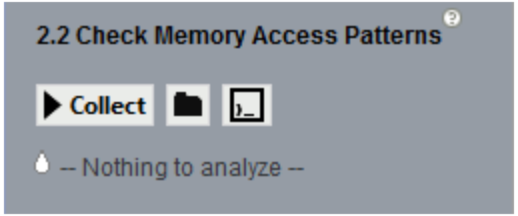
There is no confirmation inefficient memory access patterns are present. To confirm: Run a Memory Access Patterns analysis.

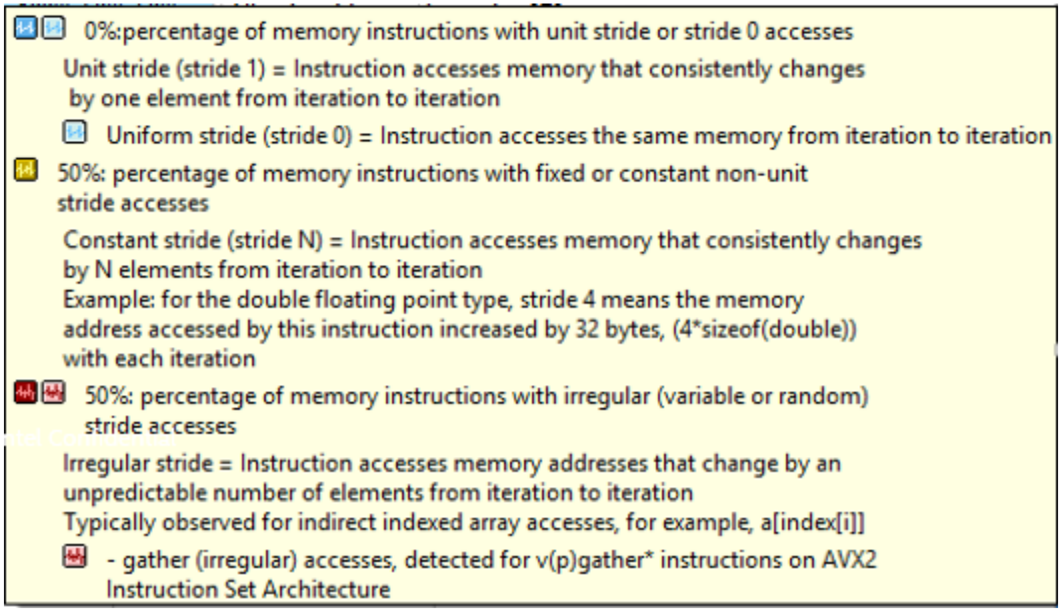
Recommendations – need more information, confirm inefficient memory access

Irregular access patterns decreases performance!

Gather profiling

- Run Memory Access Pattern Analysis (MAP)





Irregular access patterns

Bad for vectorization performance

Details View

Gather (irregular) access

Operand Size (bits): 32

Operand Type: bit*16;float32*16

Vector Length: 16

Memory access footprint: 3MB

Gather/scatter details

Pattern: "Constant (non-unit)"

Instruction accesses values with constant offset from the base:
- stride within instruction = X
- stride between iterations = X*vector length

Horizontal stride (bytes): 16

Vertical stride (bytes): 256

Mask is constant

Mask: [1111111111111111]

Active elements in the mask: 100.0%

Variable references

Names: block 0x7f0045867010 allocated at main.c:99

Hint: use the Intel Advisor details!

Specific recommendation for your application

Issue: Inefficient gather/scatter instructions present

The compiler assumes indirect or irregular stride access to data used for vector operations. Improve memory access by alerting the compiler to detected regular stride access patterns, such as:

Pattern	Description
Invariant	The instruction accesses values in the same memory throughout the loop.
Uniform (Horizontal Invariant)	The instruction accesses values in the same memory within the vector iteration.
Vertical Invariant	The instruction accesses the memory locations using the same offset across all vector iterations.
Unit	The instruction accesses values in contiguous memory throughout the loop, and the stride between vector iterations = vector length.

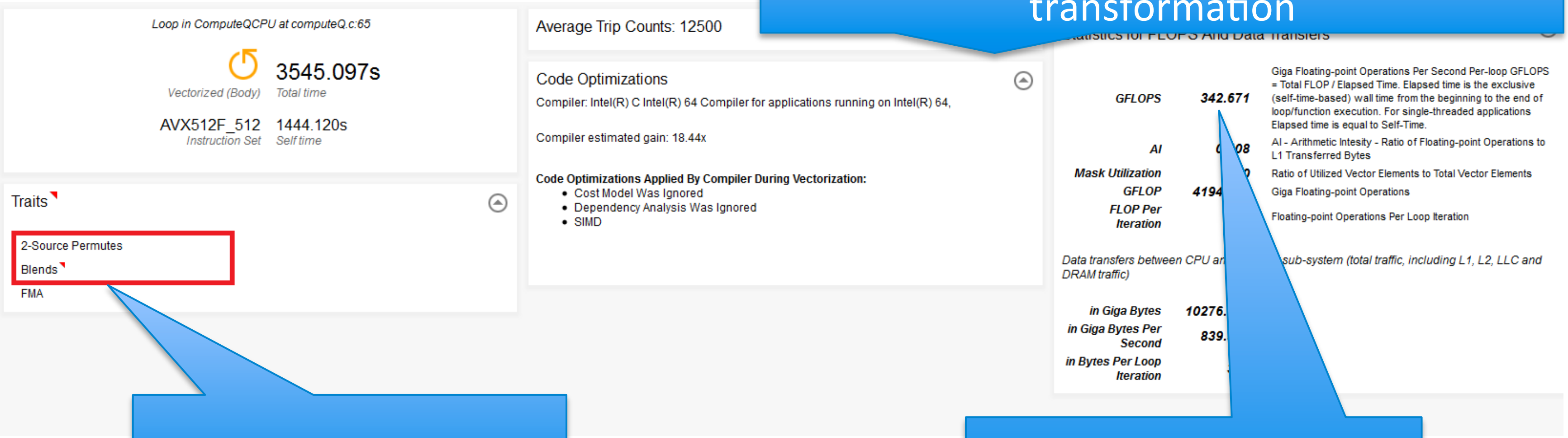
Recommendation: Refactor code with detected regular stride access patterns Confidence: Low

The Memory Access Patterns Report shows the following regular stride access(es):

Remove gather instructions

step #1 – use newer version of the intel compiler can recognize the access pattern

Gathers replacement is performed by the “Gather to Shuffle/Permutes” compiler transformation

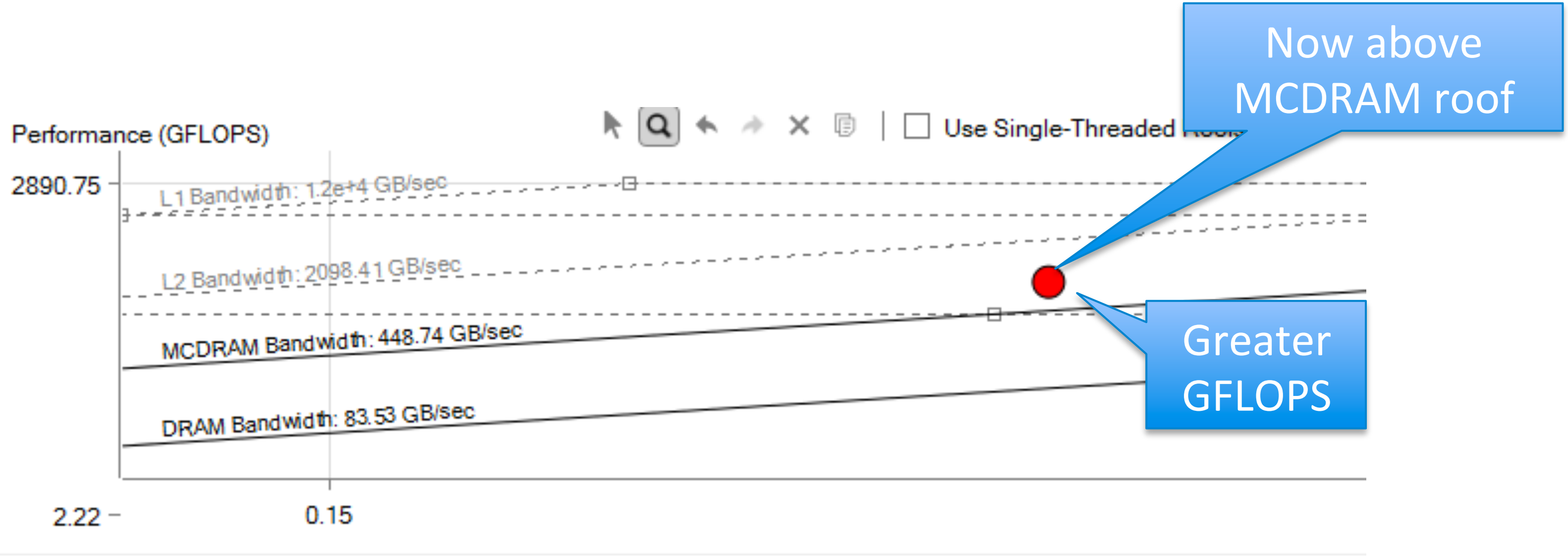


Removed gathers

Increased GFLOPS (from 266.42 to 342.67)

Remove gather instructions

step #1 – newer version of the intel compiler can recognize the access pattern



Remove gather instructions

step #2 - Use structure of arrays instead of array of structures

T

```
struct kValues {
  float Kx;
  float Ky;
  float Kz;
  float PhiMag;
};

SDLT_PRIMITIVE(kValues, Kx, Ky, Kz, PhiMag)


sdlt::soa1d_container<kValues> inputKValues(numK);
auto kValues = inputKValues.access();

  for (k = 0; k < numK; k++) {
    kValues [k].Kx() = kx[k];
    kValues [k].Ky() = ky[k];
    kValues [k].Kz() = kz[k];
    kValues [k].PhiMag() = phiMag[k];
  }


auto kVals = inputKValues.const_access();
#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiS
      for (indexK = 0; indexK < numK; indexK++) {
        expArg = PIx2 * (kVals[indexK].Kx() * x[indexX] +
        kVals[indexK].Ky() * y[indexX] +
        kVals[indexK].Kz() * z[indexX]);

        cosArg = cosf(expArg);
        sinArg = sinf(expArg);

        float phi = kVals[indexK].PhiMag();
        QrSum += phi * cosArg;
        QiSum += phi * sinArg;
      }
```

This is a classic vectorization efficiency strategy

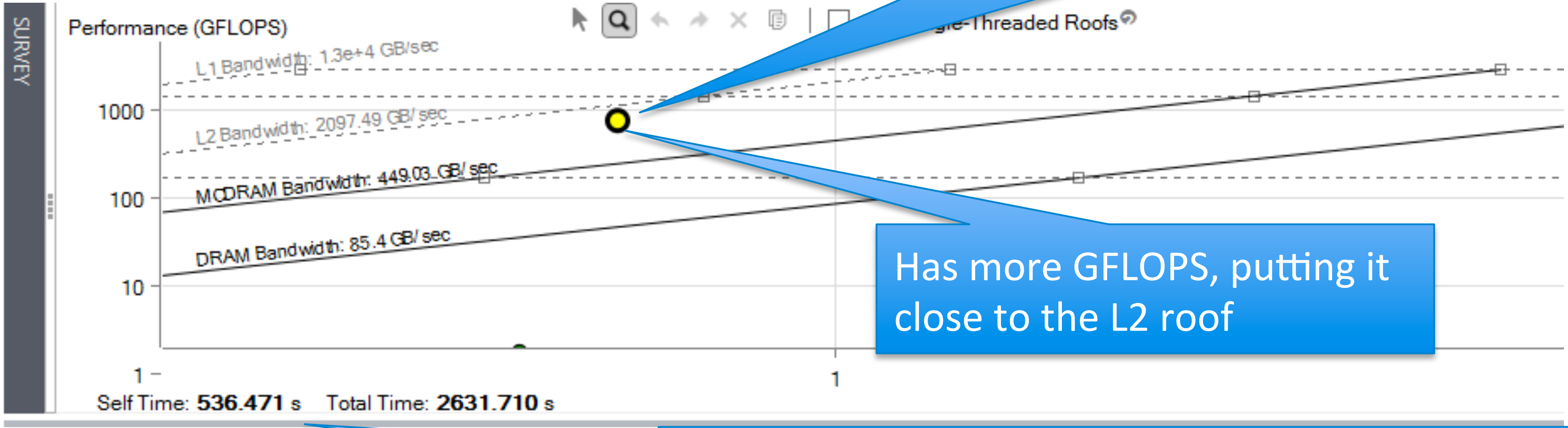
But it can yield poorly designed code

Intel® SIMD Data Layout Templates makes this transformation easy and painless!

Remove gather instructions

step #2 - Transform code using the Intel® SIMD Data Layout Templates

The loop is no longer red. This means it takes less time now



Has more GFLOPS, putting it close to the L2 roof

The total performance improvement is almost 3x for the kernel and 50% for the entire application.

Transform code using the Intel® SIMD Data Layout Templates

Summary of improvements

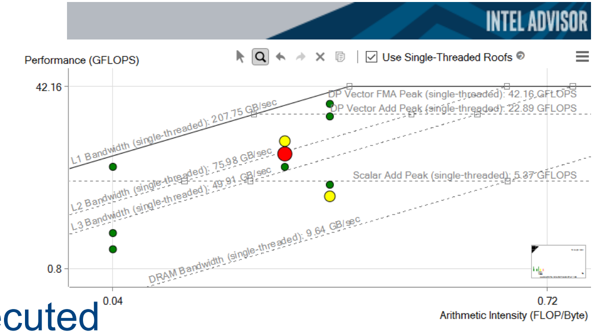
- After applying this optimization, the dot is no longer red. This means it takes less time now
- Has more GFLOPS, putting it close to the L2 roof
- The loop now has unit stride access and, as a result, no special memory manipulations
- The total performance improvement is almost 3x for the kernel and 50% for the entire application.

New!

Intel Advisor 2018 beta

Intel® Advisor – Vectorization Optimization

- Roofline analysis helps you optimize effectively
  - Find high impact, but under optimized loops
  - Does it need cache or vectorization optimization?
  - Is a more numerically intensive algorithm a better choice?
- Faster data collection
  - Filter by module - Calculate only what is needed.
  - Track refinement analysis – Stop when every site has executed



- Make better decisions with more data, more recommendations
  - Intel MKL friendly – Is the code optimized? Is the best variant used?
  - Function call counts in addition to trip counts
  - Top 5 recommendations added to summary
  - Dynamic instruction mix – Expert feature shows exact count of each instruction
- Easier MPI launching
  - MPI support in the command line dialog

summary

Call to Action

- Modernize your Code
  - To get the most out of your hardware, you need to modernize your code with vectorization and threading.
  - Taking a methodical approach such as the one outlined in this presentation, and taking advantage of the powerful tools in Intel® Parallel Studio XE, can make the modernization task dramatically easier.
    - Download the latest here: https://software.intel.com/en-us/intel-parallel-studio-xe
    - The Professional and Cluster Edition both include Advisor
  - Join the 2018 beta of Intel Parallel Studio XE to get the latest version
  - Send e-mail to vector_advisor@intel.com to get the latest information on some exciting new capabilities that are currently under development.

Q/A

Resources

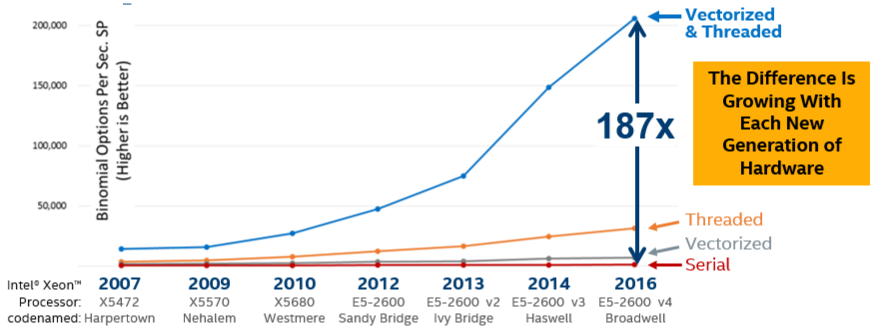
- Intel® Advisor Links
  - Vectorization Guide
    - http://bit.ly/autovectorize-guide
  - Explicit Vector Programming in Fortran
    -
  - Optimization Reports
    -
  - Beta Registration & Download
    -

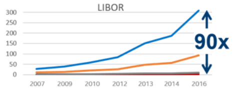
- Code Modernization Links
  - Modern Code Developer Community
    - software.intel.com/modern-code
  - Intel Code Modernization Enablement Program
    - software.intel.com/code-modernization-enablement
  - Intel Parallel Computing Centers
    - software.intel.com/ipcc
  - Technical Webinar Series Registration
    - http://bit.ly/spring16-tech-webinars
  - Intel Parallel Universe Magazine
    - software.intel.com/intel-parallel-universe-magazine

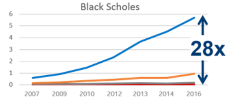
Additional Resources

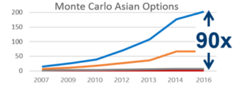
- For Intel® Xeon Phi™ coprocessors, but also applicable:
- Intel® Parallel Studio XE Composer Edition User and Reference Guides:
- Compiler User Forums

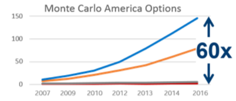
Configurations for 2007-2016 Benchn











Key:
Vectorized & Threaded
Threaded
Vectorized
Serial

Platform Hardware and Software Configuration

Platform	Unscaled Core Frequency	Cores/ Socket	Num Sockets	L1 Data Cache	L2 Cache	L3 Cache	Memory	Memory Frequency	Memory Access	H/W Prefetchers Enabled	HT Enabled	Turbo Enabled	C States	O/S Name	Operating System	Compiler Version
Intel® Xeon™ 5472 Processor	3.0 GHZ	4	2	32K	6 MB	None	32 GB	800 MHz	UMA	Y	N	N	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5570 Processor	2.9 GHZ	4	2	32K	256K	8 MB	48 GB	1333 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	256K	12 MB	48 MB	1333 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	256K	20 MB	64 GB	1600 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	256K	30 MB	64 GB	1867 MHz	NUMA	Y	Y	Y	Disabled	RHEL 7.1	3.10.0-229.el7.x86_64	icc version 14.0.1
Intel® Xeon™ E5 2600v3 Processor	2.2 GHz	18	2	32K	256K	46 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.13.5-202.fc20	icc version 14.0.1
Intel® Xeon™ E5 2600v4 Processor	2.3 GHz	18	2	32K	256K	46 MB	256 GB	2400 MHz	NUMA	Y	Y	Y	Disabled	RHEL 7.0	3.10.0-123. el7.x86_64	icc version 14.0.1
Intel® Xeon™ E5 2600v4 Processor	2.2 GHz	22	2	32K	256K	56 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	Disabled	CentOS 7.2	3.10.0-327. el7.x86_64	icc version 14.0.1

Performance measured in Intel Labs by Intel employees.

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

intel®