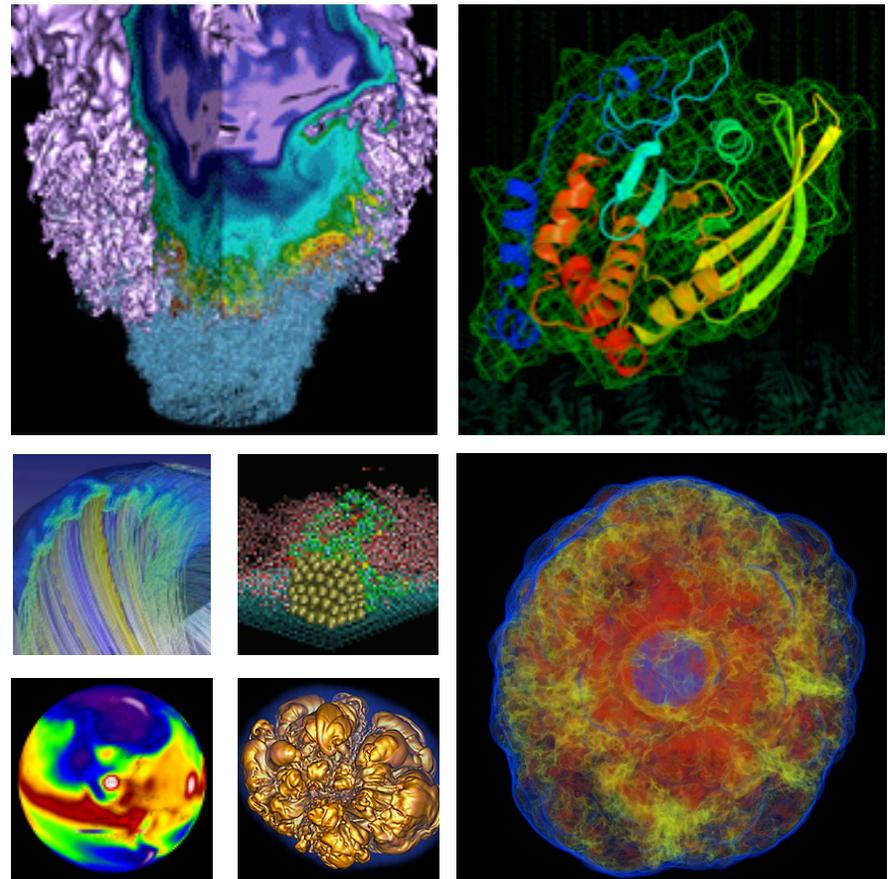


Intel Advisor and Roofline Automation



Tuomas Koskela (NERSC),
Zakhar Matveev (Intel)

June 14, 2017

5 Steps to Efficient Vectorization - Vector Advisor



1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization	
			Loop Type	Why No Vectorization?
[loop in runCforallLambdaLoops]	0.094s	0.094s	Scalar	vector dependence prevents vector...
[loop in runCforallLambdaLoops]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::Complex_base<double,struct _C_double_complex>:i...	0.031s	0.031s	Vectorized (Bovd)	

Vectorized SSE; SSE2 loop processing Float32; Float64 data type
Peeled loop; loop stmts were reordered

2. Guidance: detect problem and recommend how to fix it

Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

Recommendation: Align memory access
Projected maximum performance gain: High
Projection confidence: Medium

Use one of the memory accesses in the source loop does not align on a byte boundary and tell the compiler your memory access is aligned to a byte boundary:

```
SIZE*sizeof(float), 32);
```

3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

Total Time	Trip Counts			Iteration Duration	Call Count
	Median	Min	Max		
3.151s	1	1	1	3.1509s	1
0.440s	1	1	1	< 0.0001s	2408000
0.010s	1	1	2	< 0.0001s	207506
0.010s	2	1	9	< 0.0001s	1173619
0.010s	3	1	5	< 0.0001s	1312315

4. Loop-Carried Dependency Analysis

Problems and Messages

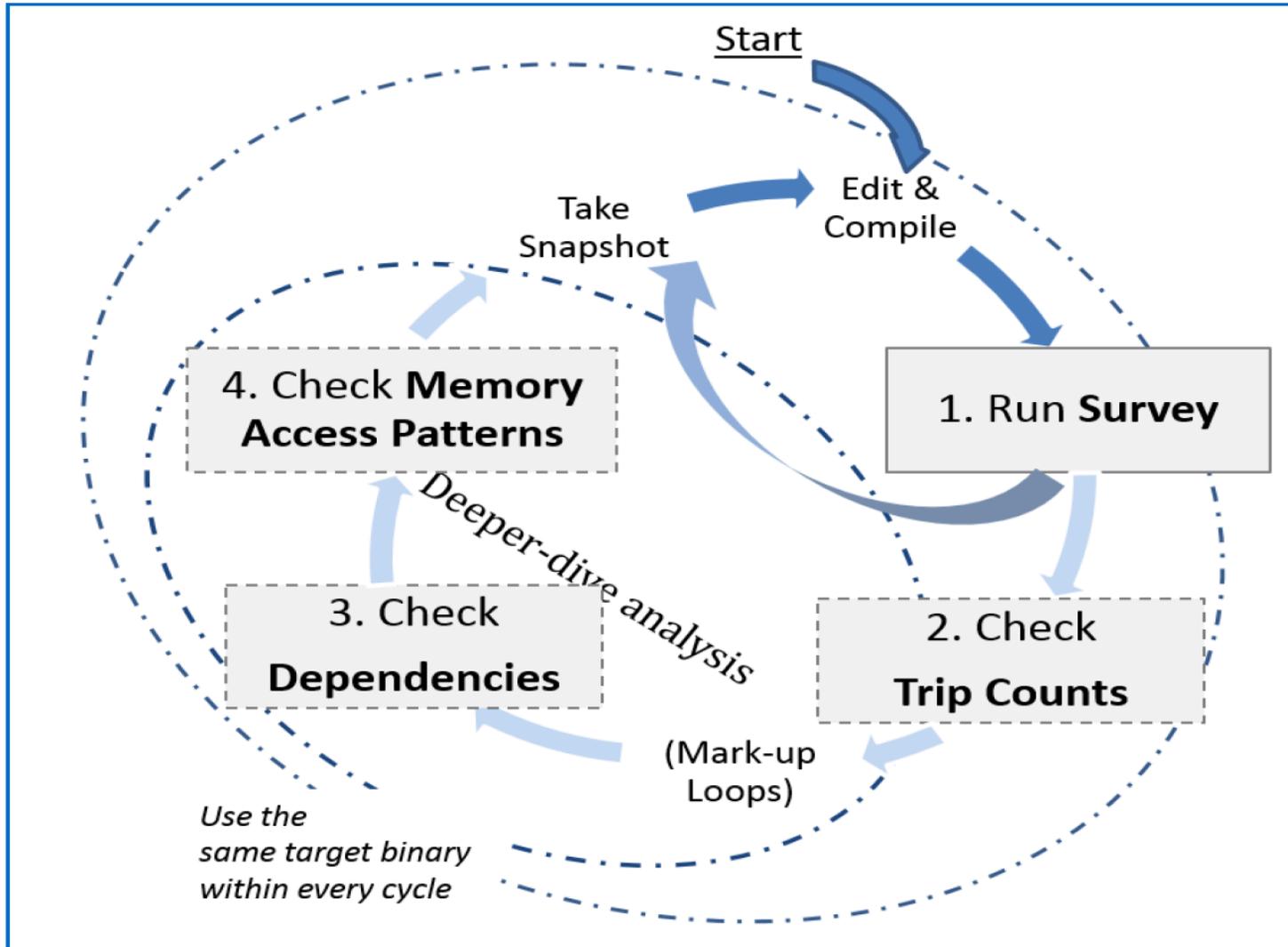
ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6	Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7	Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

5. Memory Access Patterns Analysis

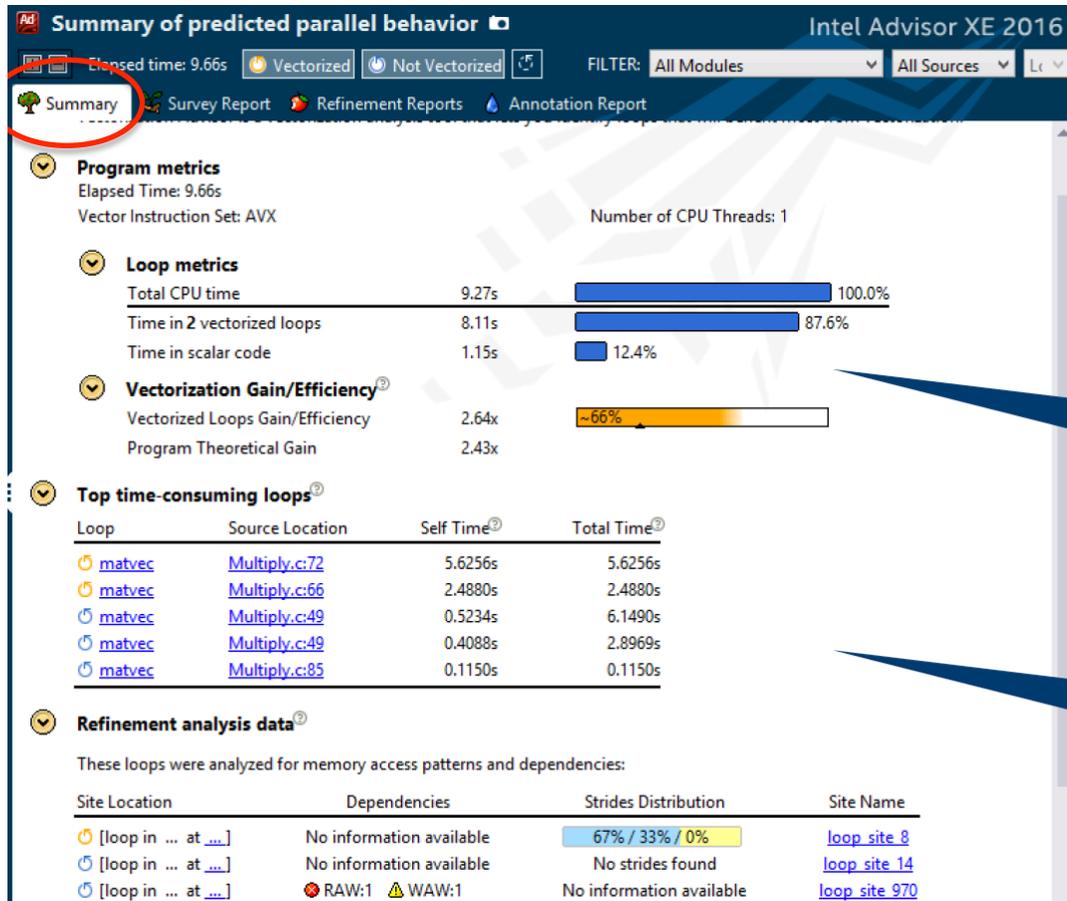
Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.coc1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.coc622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.coc925	No information available	100% / 0% / 0%	All unit strides

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.coc637	lcalcs.exe	
P23	0; 0	Unit stride	runCRawLoops.coc638	lcalcs.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.coc628	lcalcs.exe	

Vectorization Analysis Workflow



Quickly Characterize the Vectorization Efficiency of Your Code: Advisor Summary



Use the summary view to quickly characterize your program

Time in Scalar vs. Vector loops. SIMD Efficiency.

Focus on Hottest kernels

Advisor Survey: Focus and Characterize



Where should I add vectorization and/or threading parallelism? Intel Advisor XE 201

Elapsed time: 9.49s Vectorized Not Vectorized FILTER: All Modules All Sources Loops All Threads

Summary Survey Report Refinement Reports Annotation Report

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops			
						Vect...	Efficiency	Gain...	VL (...)
[loop in matvec at Multiply.c:72]	1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~55%	2.19x	4
[loop in matvec at Multiply.c:66]	1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4
[loop in matvec at Multiply.c:49]		0.531s	5.812s	Scalar					
[loop in matvec at Multiply.c:49]		0.516s	3.344s	Scalar					
[loop in matvec at Multiply.c:85]	1 Assumed d...	0.063s	0.063s	Scalar	vector dependence...				
[loop in main at driver.c:151]		0.016s	9.297s	Scalar	loop in function ...				
[loop in matvec at Multiply.c:49]			0.000s	Scalar					

hotspots

HOW to improve performance

ISA

Efficiency

What prevents vectorization?

All the data you need for effective vectorization

Code Analytics



Elapsed time: 2:20c Vectorized Not Vectorized FILTER: All Modules All Sources Loops All Threads OFF Smart Mode

Summary **Survey & Roofline** Refinement Reports INTEL ADVISOR 20

Function, Call Sites and Loops	Total Time	Type	Vectorized Loops				FLOPS And AVX-512 Mask Usage		
			Vector I...	Efficiency	Gain...	VL ..	Vector Issues	GFLOPS	AI
[loop in fCalcInteraction_Sh...	0,050s 20,8%	Vectorized (Rem...	AVX512	44%	3,53x	8	0,847	0,097	50,0%
[loop in fGetEquilibriumF at I...	0,050s 20,8%	Vectorized (Body; ...	AVX512	~36%	5,79x	16; 8	3,666	0,345	79,2%
[loop in fCalcInteraction_Sha...	0,030s	Vectorized (Remai...	AVX512	44%	3,53x	8	1,482	0,097	50,0%
[loop in fGetOneMassSite at I...	0,020s	Vectorized (Remai...	AVX512	23%	1,84x	8	0,768	0,125	79,2%
[loop in fSiteFluidCollisionBG...	0,010s	Vectorized (Remai...	AVX512	~38%	3,05x	8	0,724	0,113	37,5%
[loop in fGetOneMassSite at I...	0,010s	Vectorized (Remai...	AVX512	~24%	1,94x	8	1,529	0,125	79,2%

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

Loop in fCalcInteraction_ShanChen_Boundary at lbpFORCE.cpp:188

0,050s
Vectorized (Remainder) Total time

AVX2; AVX512F_512 0,050s
Instruction Set Self time

Memory 41% (7)
Compute 35% (6)
Other 24% (4)
Instruction Mix Summary

44% Vectorization Efficiency 3,53x Vectorization Gain

Average Trip Counts: 1

Traits: FMA, Mask Manipulations

Code Optimizations: Compiler: Intel(R) C++ Intel(R) 64 Compiler for applications running on Intel(R) 64, Version: 16.0.2.181 Build 20160204, Compiler estimated gain: 4,85x

Code Optimizations Applied By Compiler During Vectorization: Masked Loop Vectorization, Unaligned Access in Vector Loop

GFLOPS: 0.8474
AVX-512 Mask Usage: 50

Instruction Mix: Memory: 7 Compute: 6 Other: 4 Number of Vector Registers: 7

Advisor Memory Access Pattern



Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in fPropagationSwap at lbpSUB.cpp:1247]	No information available	33% / 5% / 62%	Mixed strides	loop_site_60

blue color: fraction of unit stride accesses
yellow: "fixed" stride accesses ratio
red color: fraction of irregular (variable stride) accesses

Memory Access Patterns Report | Dependencies Report

ID	Stride	Type	Source	Site Name	Variable
----	--------	------	--------	-----------	----------

P1	3	16% / 84% / 0%	Mixed strides		
----	---	----------------	---------------	--	--

```

1246 #endif
1247     for (int m=1; m<=half; m++) {
1248         nextx = fCpMod(i + lbv[3*m]
1249         nexty = fCpMod(j + lbv[3*m+
1250         nextz = fCpMod(k + lbv[3*m+
    
```

P11	0; 1				
-----	------	--	--	--	--

P12	-289559; -274359; -14477; -13717; -13679; 723; 302519;				
-----	--	--	--	--	--

```

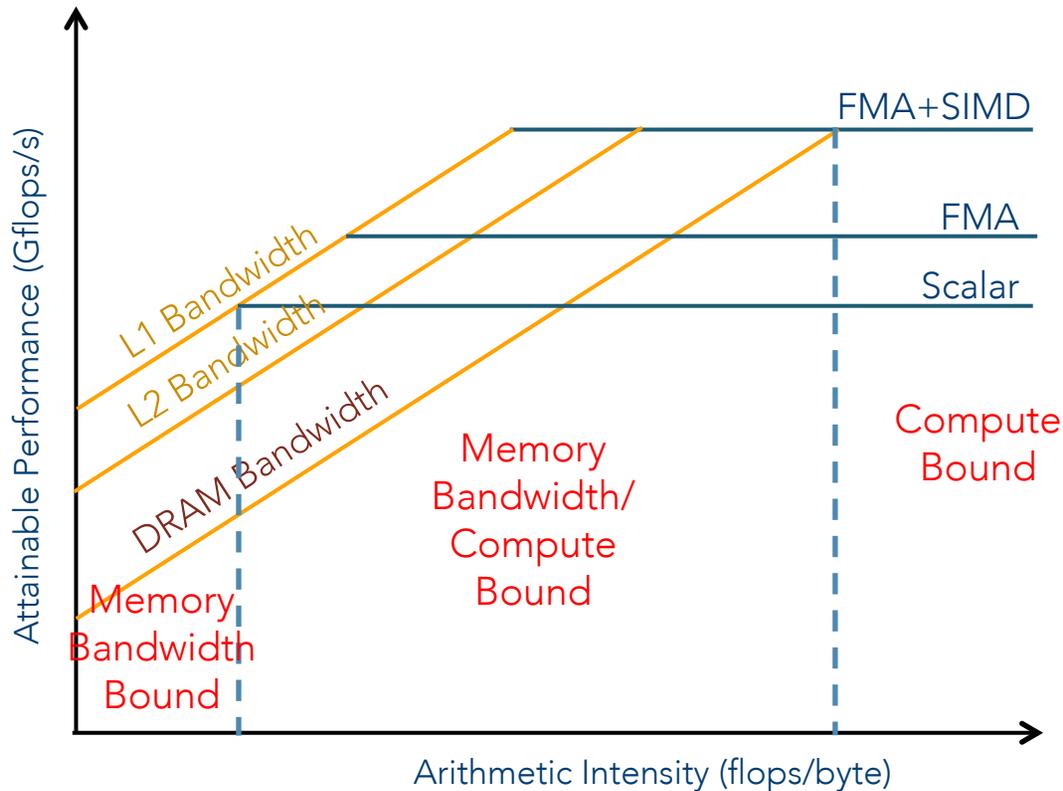
1251         ilnext = (nextx * Ymax + nex
1252 #ifndef SWAP_OVERLAP
1253     fSwapPair (lbf[il*lbsitlength + 1*lbsy.
    
```

- 16%**: percentage of memory instructions with unit stride or stride 0 accesses
 - Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
 - Stride 0 = Instruction accesses the same memory from iteration to iteration
- 84%**: percentage of memory instructions with fixed or constant non-unit stride accesses
 - Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration
 - Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration
- 0%**: percentage of memory instructions with irregular (variable or random) stride accesses
 - Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration
 - Typically observed for indirect indexed array accesses, for example, a[index[i]]
 - gather (irregular) accesses**, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture

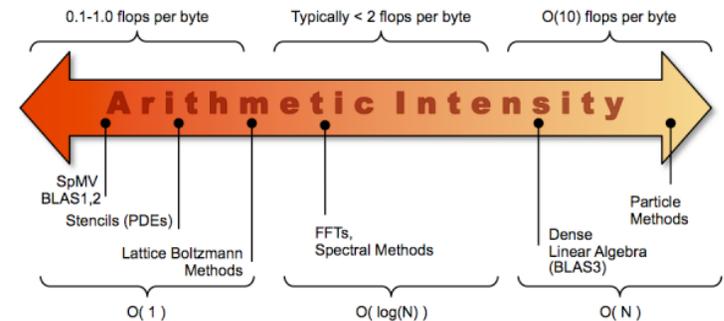
Roofline Performance Model: Am I bound by VPU/CPU or Memory



Roofline reflects an absolute performance bound (Gflops/s) of the system as a function of Arithmetic Intensity (flops/byte) of the application.



$$\text{Arithmetic Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred}}$$

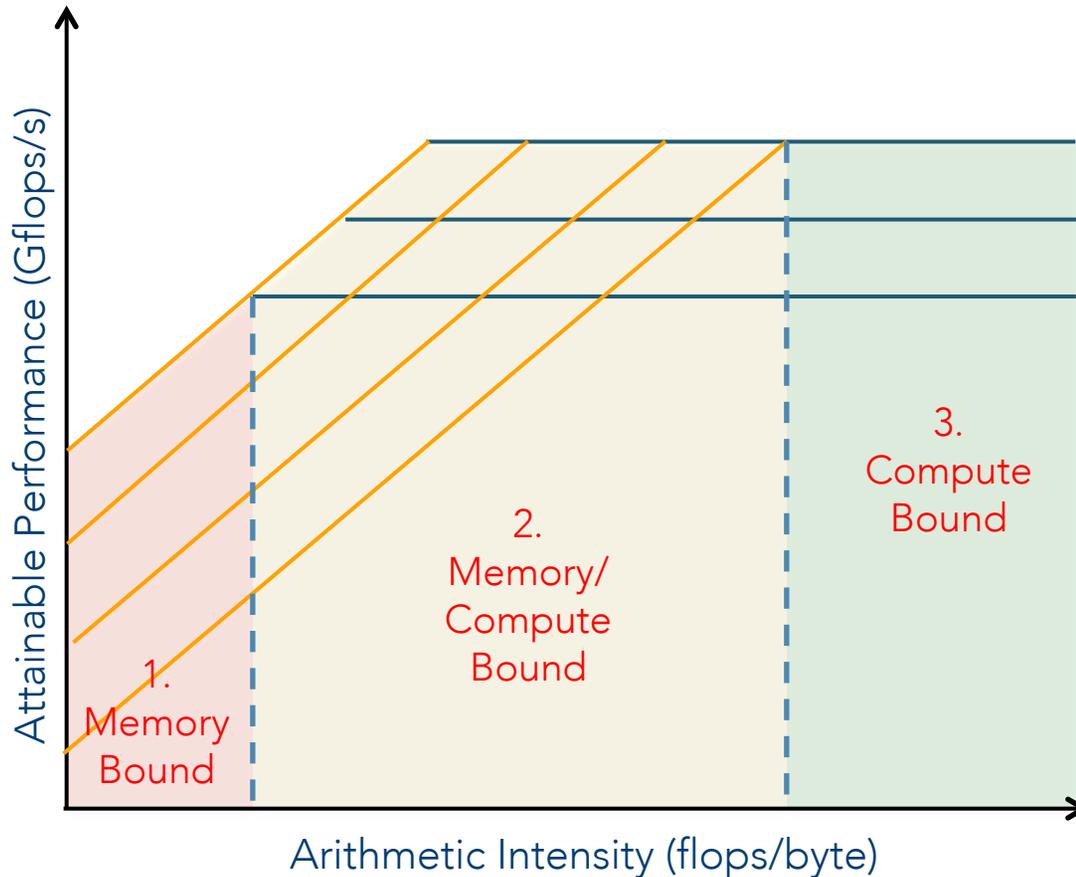


Why Do We Need the Roofline Model?



- Need a sense of absolute performance when optimizing applications
 - How do I know if my performance is good?
 - Why am I not getting peak performance of the platform?
- Many potential optimization directions
 - How do I know which one to apply?
 - What is the limiting factor in my app's performance?
 - How do I know when to stop?

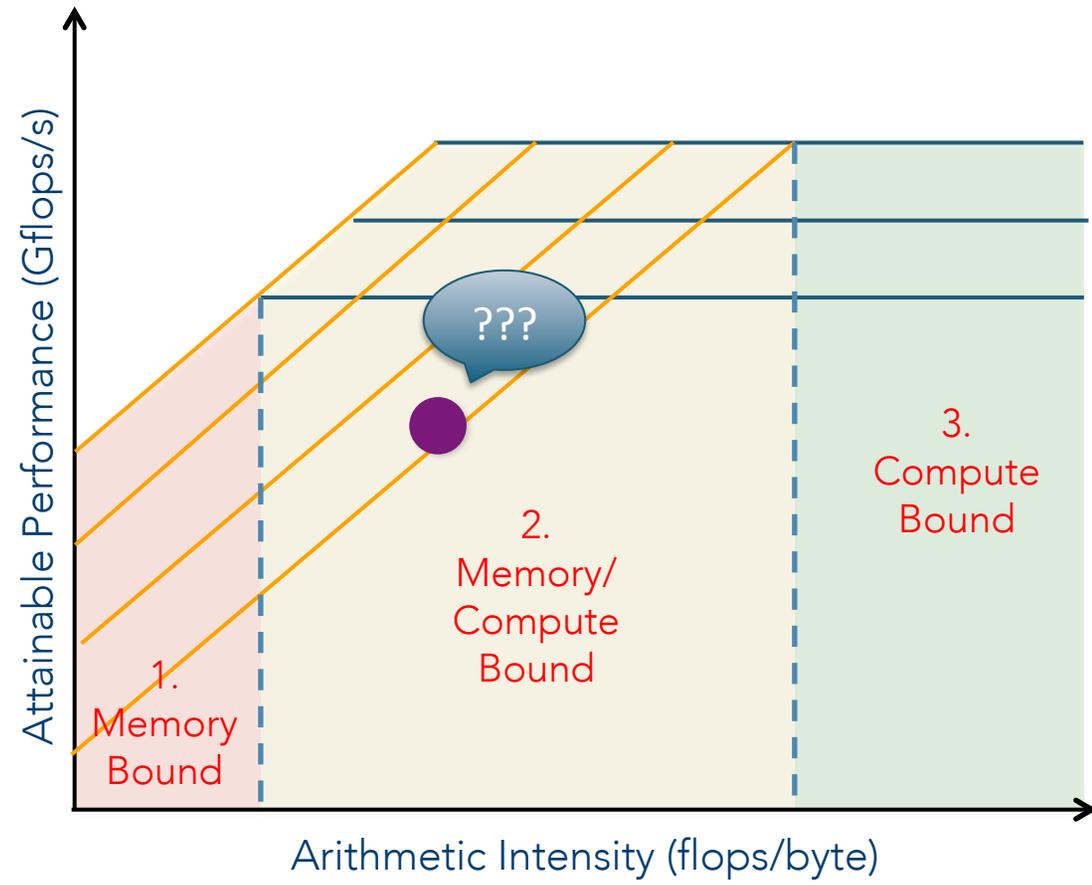
Is My Application Bound by a Memory Bandwidth Peak or a Compute Peak?



Often it's a combination of the two

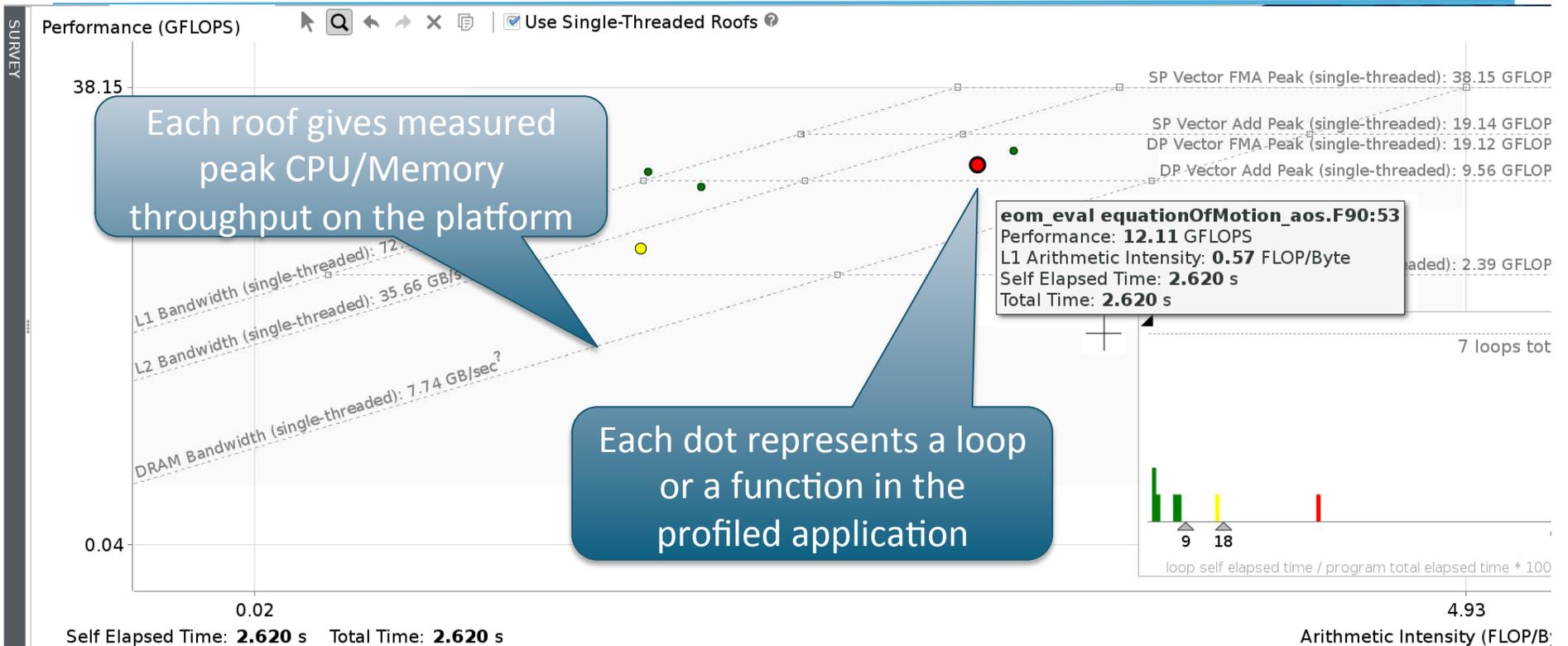
- Applications in **area 1** are purely memory bandwidth bound
- Applications in **area 3** are purely compute bound
- In **area 2** we need more information

Place your application on the roofline, then ask yourself “Why am I Here?”



Usually, the analysis is not straightforward... You won't be on any ceiling. Or if you are, it is a coincidence. BUT - asking the questions “why am I not on a higher ceiling?” and “what should I do to reach it?” is always productive.

Roofline Automation in Advisor 2017



Source	Top Down	Code Analytics	Assembly	Recommendations	Why No Vectorization?	
File: equationOfMotion_aos.F90:53 eom_eval						
Lin.	Source	Total Time	%	Loop/Function Time	%	Trail
46	double precision :: v					
47	double precision :: Fr					
48	double precision :: Fp					
49	double precision :: Fz					
50						
51	err = 0					
52						
53	do iv = 1,veclen	0.140s		2.620s		
	[loop in eom_eval at equationOfMotion_aos.F90:53]					
	Vectorized AVX512ER_512; AVX512F_512 loop processes Float32; Float64; Int64 data type(s) and includes Appr. Reciprocals()					
	No loop transformations applied					
54						
55	! Norms					

Advisor Roofline Under the Hood



Roofline application profile:

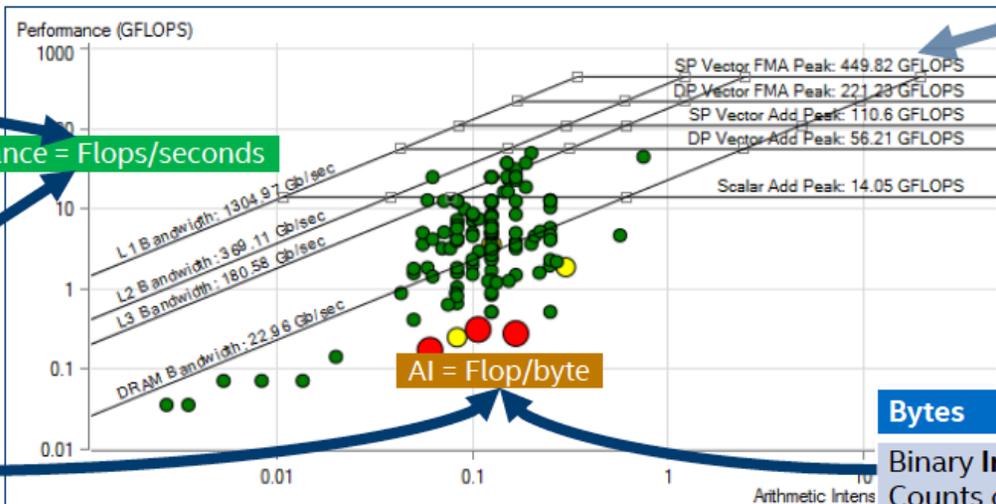
Axis Y: $\text{FLOP/S} = \# \text{FLOP (mask aware)} / \# \text{Seconds}$

Axis X: $\text{AI} = \# \text{FLOP} / \# \text{Bytes}$

Seconds
User-mode **sampling**
Root access not needed

Roofs
Microbenchmarks
Actual peak for the current configuration

Performance = Flops/seconds



#FLOP
Binary **Instrumentation**
Does not rely on CPU counters

Bytes
Binary **Instrumentation**
Counts operands size (not cachelines)

What to Do With This Information?



- **Compute bound applications**
 - Make sure you have good OpenMP scalability. Look to see thread activity for major OpenMP regions.
 - Make sure your code is vectorizing. Look at Cycles per Instruction (CPI) and VPU utilization.
- **Memory bandwidth bound applications**
 - Try to improve memory locality, cache reuse
 - Identify the key arrays leading to high memory bandwidth usage and make sure they are/will-be allocated in HBM on KNL
 - Profit by getting ~ 5x more bandwidth GB/s.

Advisor Roofline access and how-to command line example



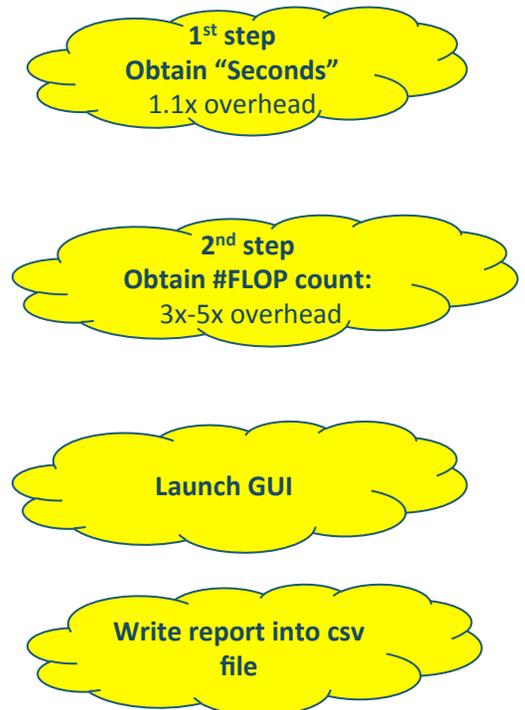
```
> module load advisor

> advixe-cl --collect survey
--project-dir ./your_project
-- <your-executable-with-parameters>

> advixe-cl --collect tripcounts
-flops-and-masks --project-dir
./your_project
-- <your-executable-with-parameters>

> advixe-gui ./your_project

> advixe-cl --report survey
--project-dir <your-project-dir>
--show-all-columns --format=csv
--report-output <output-file.csv>
```



Command line MPI example (slurm)



1st step:

```
> srun -n <num-of-ranks> -c <num_of_cores_per_rank>  
advixe-cl -v -collect survey  
-project-dir=<same_dir_name>  
-data-limit=0 <your_executable>
```

2nd step:

```
> srun -n <num-of-ranks> -c <num_of_cores_per_rank>  
advixe-cl -v -collect tripcounts  
-flops-and-masks -project-dir=<same_dir_name>  
-data-limit=0 <your_executable>
```

Further Reading and Viewing



- <http://www.nersc.gov/users/software/performance-and-debugging-tools/advisor/>
- S. Williams et al. *CACM* (2009)
<http://crd.lbl.gov/departments/computer-science/PAR/research/roofline>
- A. Ilic et al., *IEEE Computer Architecture Letters* (2014)
- <http://www.intel.com/content/www/us/en/events/hpcdevcon/parallel-programming-track.html#utilizing>
- <https://www.youtube.com/watch?v=h2QEM1HpFgg>
- <https://software.intel.com/en-us/articles/getting-started-with-intel-advisor-roofline-feature>
- <http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/>
- <https://www.codeproject.com/Articles/1169323/Intel-Advisor-Roofline-Analysis>
- <https://software.intel.com/sites/default/files/managed/1e/19/roofing-a-house.pdf>
- <https://github.com/tkoskela/pyAdvisor>



National Energy Research Scientific Computing Center